

ORYA — Repo Core (Export para PDF)

Gerado em: 2025-12-17T00:32:31.553Z

Total de ficheiros incluídos: 402

app/[username]/FollowClient.tsx

```
"use client";

import { useEffect, useState, useTransition } from "react";

type Props = {
  targetUserId: string;
  initialIsFollowing: boolean;
};

export default function FollowClient({ targetUserId, initialIsFollowing }: Props) {
  const [isFollowing, setIsFollowing] = useState(initialIsFollowing);
  const [fetching, setFetching] = useState(false);
  const [isPending, startTransition] = useTransition();

  useEffect(() => {
    let mounted = true;
    (async () => {
      setFetching(true);
      try {
        const res = await fetch(`/api/social/follow-status?userId=${targetUserId}`);
        const json = await res.json();
        if (mounted && res.ok && json?.ok) {
          setIsFollowing(Boolean(json.isFollowing));
        }
      } catch {
        // ignore
      } finally {
        if (mounted) setFetching(false);
      }
    })();
    return () => {
      mounted = false;
    };
  }, [targetUserId]);

  const toggleFollow = async () => {
    const next = !isFollowing;
    setIsFollowing(next);
    startTransition(async () => {
      try {
        const res = await fetch(next ? "/api/social/follow" : "/api/social/unfollow", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ targetUserId }),
        });
        const json = await res.json();
        if (!res.ok || !json?.ok) {
          setIsFollowing(!next);
        }
      } catch {
        setIsFollowing(!next);
      }
    });
  };
}

return (
  <button
    type="button"
    disabled={fetching || isPending}
    onClick={toggleFollow}
    className={` inline-flex items-center rounded-full px-3 py-1 text-[12px] font-semibold transition ${
      isFollowing
        ? "border border-white/25 bg-white/10 text-white/80 hover:bg-white/15"
        : "bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black shadow-[0_0_18px_rgba(107,255,255,0.35)]"
    } hover:scale-[1.02]"
    >
    {isFollowing ? "A seguir" : "Seguir"}
  </button>
);
```

```
    );  
}
```

app/[username]/page.tsx

```
import { notFound } from "next/navigation";
import { createSupabaseServer } from "@lib/supabaseServer";
import { prisma } from "@lib/prisma";
import FollowClient from "./FollowClient";

type Params = { username: string };

export default async function PublicProfilePage({ params }: { params: Promise<Params> }) {
  const { username } = await params;

  if (!username || username.trim() === "") {
    notFound();
  }

  const supabase = await createSupabaseServer();
  const {
    data: { user: viewer },
  } = await supabase.auth.getUser();

  const profile = await prisma.profile.findFirst({
    where: {
      username: { equals: username, mode: "insensitive" },
    },
    select: {
      id: true,
      username: true,
      fullName: true,
      avatarUrl: true,
      city: true,
      visibility: true,
    },
  });
}

if (!profile) {
  notFound();
}

const visibility = profile.visibility ?? "PUBLIC";
const isOwner = viewer?.id === profile.id;
const isPrivate = visibility === "PRIVATE" && !isOwner;
const displayName = profile.fullName || profile.username || "Utilizador ORYA";
const usernameLabel = profile.username || "perfil";

// Eventos públicos deste organizador/utilizador
const events = isPrivate
  ? []
  : await prisma.event.findMany({
    where: {
      ownerUserId: profile.id,
      status: "PUBLISHED",
    },
    select: {
      id: true,
      slug: true,
      title: true,
      type: true,
      startsAt: true,
      locationName: true,
      coverImageUrl: true,
    },
    orderBy: { startsAt: "asc" },
    take: 30,
  });

const [followersCount, followingCount] = await Promise.all([
  prisma.follows.count({ where: { following_id: profile.id } }),
  prisma.follows.count({ where: { follower_id: profile.id } }),
]);

return (

```

```

<main className="orya-body-bg min-h-screen text-white">
  <section className="max-w-5xl mx-auto px-5 py-10 space-y-4">
    <div className="flex items-center gap-3">
      <div className="relative h-16 w-16 rounded-full bg-gradient-to-br from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] flex
items-center justify-center text-black font-semibold text-xl overflow-hidden shadow-[0_10px_28px_rgba(0,0,0,0.35)] ring-1 ring-
white/10">
        {profile.avatarUrl ? (
          // eslint-disable-next-line @next/next/no-img-element
          <img
            src={profile.avatarUrl}
            alt={displayName}
            className="h-full w-full object-cover"
          />
        ) : (
          <span>{displayName.charAt(0).toUpperCase()}</span>
        )}
      </div>
      <div className="space-y-1">
        <h1 className="text-2xl font-semibold">{displayName}</h1>
        <p className="text-sm text-white/70">@{usernameLabel}</p>
        {!isOwner && !isPrivate && (
          <FollowClient targetUserId={profile.id} initialIsFollowing={false} />
        )}
        {!isPrivate && (
          <div className="flex gap-3 text-[12px] text-white/70">
            <span>{followingCount} a seguir</span>
            <span>{followersCount} seguidores</span>
          </div>
        )}
      </div>
      <div>
        {isPrivate ? (
          <div className="rounded-2xl border border-white/10 bg-white/5 px-4 py-3 text-sm text-white/70">
            Este perfil é privado. Vês apenas avatar, nome e username.
          </div>
        ) : (
          <>
            <div className="space-y-2 text-sm text-white/65">
              {profile.city && <p>📍 {profile.city}</p>}
              <p>Eventos e experiências públicos deste organizador/usuário.</p>
            </div>
            <div className="mt-4 space-y-2">
              <h2 className="text-lg font-semibold">Eventos</h2>
              {events.length === 0 ? (
                <p className="text-sm text-white/60">Ainda não há eventos públicos associados.</p>
              ) : (
                <div className="grid gap-4 sm:grid-cols-2 lg:grid-cols-3">
                  {events.map((ev) => {
                    const href = ev.type === "EXPERIENCE" ? `/experiencias/${ev.slug}` : `/eventos/${ev.slug}`;
                    return (
                      <a
                        key={ev.id}
                        href={href}
                        className="group rounded-2xl border border-white/10 bg-white/[0.04] overflow-hidden hover:border-
white/18 hover:-translate-y-[4px] transition block"
                      >
                        <div className="h-32 w-full bg-gradient-to-br from-[#111827]/60 to-[#0b1224]/60 overflow-hidden">
                          {ev.coverImageUrl ? (
                            // eslint-disable-next-line @next/next/no-img-element
                            <img
                              src={ev.coverImageUrl}
                              alt={ev.title}
                              className="h-full w-full object-cover transition-transform duration-200 group-hover:scale-[1.03]"
                            />
                          ) : null}
                        </div>
                      <div className="p-3 space-y-1.5">
                        <p className="text-[13px] font-semibold text-white line-clamp-2">{ev.title}</p>
                        <p className="text-[11px] text-white/70">
                          {ev.startsAt
                            ? new Date(ev.startsAt).toLocaleString("pt-PT", {
                                weekday: "short",
                                day: "2-digit",
                                month: "short",
                                hour: "2-digit",
                                minute: "2-digit",
                              })
                            : null}
                        </div>
                      </a>
                    )
                  )}
                </div>
              )
            </div>
          </>
        )}
      </div>
    </div>
  </section>
</main>

```

```

        : "Data a anunciar"}
      </p>
      <p className="text-[11px] text-white/60 line-clamp-1">
        {ev.locationName || "Local a anunciar"}
      </p>
      <span className="inline-flex rounded-full border border-white/12 bg-white/5 px-2 py-0.5 text-[10px]
text-white/70">
        {ev.type === "EXPERIENCE" ? "Experiência" : "Evento"}
      </span>
    </div>
  </a>
);
)}
)}
</div>
)
}
</section>
</main>
);
}

```

app/admin/components/AdminLayout.tsx

```

"use client";

import Link from "next/link";
import { usePathname } from "next/navigation";
import { ReactNode } from "react";

type AdminLayoutProps = {
  children: ReactNode;
  title: string;
  subtitle?: string;
};

const navItems = [
  { href: "/admin", label: "Dashboard" },
  { href: "/admin/organizadores", label: "Organizadores" },
  { href: "/admin/eventos", label: "Eventos" },
  { href: "/admin/tickets", label: "Bilhetes" },
  { href: "/admin/payments", label: "Pagamentos" },
  { href: "/admin/settings", label: "Configurações" },
];

export function AdminLayout({ children, title, subtitle }: AdminLayoutProps) {
  const pathname = usePathname();
  const isActive = (href: string) =>
    pathname === href || pathname.startsWith(`#${href}`);
  return (
    <div className="orya-body-bg min-h-screen text-white pb-16">
      <header className="border-b border-white/10 bg-gradient-to-r from-black/70 via-[#0c1224]/70 to-black/60 backdrop-blur-xl">
        <div className="mx-auto flex max-w-6xl flex-col gap-3 px-5 py-5 md:flex-row md:items-center md:justify-between">
          <div className="flex items-center gap-3">
            <span className="inline-flex h-9 w-9 items-center justify-center rounded-2xl bg-gradient-to-tr from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-[11px] font-extrabold tracking-[0.15em] shadow-[0_10px_40px_rgba(0,0,0,0.45)]">
              AD
            </span>
            <div className="space-y-0.5">
              <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Admin</p>
              <p className="text-base font-semibold text-white/90 leading-tight">{title}</p>
              {subtitle && <p className="text-[12px] text-white/65">{subtitle}</p>}
            </div>
          </div>
        <div className="hidden items-center gap-2 text-[12px] text-white/70 md:flex">
          <span className="h-2 w-2 rounded-full bg-emerald-400 shadow-[0_0_4px_rgba(16,185,129,0.18)]" aria-hidden />
          <span>Sessão administrativa ativa</span>
        </div>
      </div>
      <div className="mx-auto max-w-6xl px-5 pb-3 md:hidden">
        <div className="flex gap-2 overflow-x-auto pb-1 text-[12px]">
          {navItems.map((item) => {
            const active = isActive(item.href);
            return (

```

```

        <Link
          key={item.href}
          href={item.href}
          className={'whitespace-nowrap rounded-full px-3 py-1.5 transition ${
            active
            ? "bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold"
            : "border border-white/15 bg-white/5 text-white/80 hover:bg-white/10"
          }'>
          {item.label}
        </Link>
      );
    )}
  </div>
</div>
</header>

<div className="mx-auto flex max-w-6xl gap-6 px-5 pt-6">
  <nav className="sticky top-6 hidden h-fit w-56 flex-col gap-2 rounded-2xl border border-white/10 bg-white/5 p-4 text-[12px] text-white/80 backdrop-blur md:flex">
    <p className="px-1 text-[11px] uppercase tracking-[0.18em] text-white/50">
      Navegação
    </p>
    {navItems.map((item) => {
      const active = isActive(item.href);
      return (
        <Link
          key={item.href}
          href={item.href}
          className={'rounded-xl px-3 py-2 text-[13px] transition ${
            active
            ? "bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold shadow-[0_10px_30px_rgba(0,0,0,0.35)]"
            : "border border-transparent hover:border-white/10 hover:bg-white/5"
          }'>
          {item.label}
        </Link>
      );
    ))}
  </nav>
  <main className="flex-1 space-y-6">{children}</main>
</div>
</div>
);
}

```

app/admin/components/AdminTopActions.tsx

```

"use client";

import Link from "next/link";
import { CsvExportButton } from "./CsvExportButton";

type AdminTopActionsProps = {
  showTicketsExport?: boolean;
  showPaymentsExport?: boolean;
};

export function AdminTopActions({ showTicketsExport, showPaymentsExport }: AdminTopActionsProps) {
  const linkBase =
    "inline-flex items-center gap-1 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-white/80 transition
  hover:border-white/25 hover:bg-white/10";

  return (
    <div className="flex flex-wrap items-center gap-2 text-[11px]">
      <Link href="/admin/organizadores" className={linkBase}>
        Organizadores
      </Link>
      <Link href="/admin/eventos" className={linkBase}>
        Eventos
      </Link>
      <Link href="/admin/payments" className={linkBase}>
        Pagamentos
      </Link>
      <Link href="/admin/tickets" className={linkBase}>

```

```

        Bilhetes
    </Link>
    <Link href="/admin/settings" className={linkBase}>
        Configurações
    </Link>
    {showTicketsExport && (
        <CsvExportButton href="/api/admin/tickets/export" label="Export CSV Tickets" />
    )}
    {showPaymentsExport && (
        <CsvExportButton href="/api/admin/payments/export" label="Export CSV Pagamentos" />
    )}
</div>
);
}

```

app/admin/components/CsvExportButton.tsx

```

"use client";

import { useState } from "react";

type CsvExportButtonProps = {
    href: string; // endpoint that returns CSV
    label?: string;
};

export function CsvExportButton({ href, label = "Exportar CSV" }: CsvExportButtonProps) {
    const [loading, setLoading] = useState(false);

    async function handleExport() {
        try {
            setLoading(true);
            const res = await fetch(href, { cache: "no-store" });
            if (!res.ok) throw new Error("Falha no download");
            const blob = await res.blob();
            const url = window.URL.createObjectURL(blob);
            const a = document.createElement("a");
            a.href = url;
            a.download = "export.csv";
            a.click();
            window.URL.revokeObjectURL(url);
        } catch (err) {
            console.error("[CsvExport] erro exportando CSV", err);
        } finally {
            setLoading(false);
        }
    }

    return (
        <button
            type="button"
            onClick={handleExport}
            disabled={loading}
            className="rounded-full border border-white/20 px-3 py-1.5 text-[11px] text-white/80 hover:bg-white/10 disabled:opacity-50">
            {loading ? "A gerar..." : label}
        </button>
    );
}

```

app/admin/components/PaymentTools.tsx

```

"use client";

import { useState } from "react";

type Status = "idle" | "loading" | "success" | "error";

export default function PaymentTools() {
    const [pi, setPi] = useState("");
    const [status, setStatus] = useState<Status>("idle");
    const [message, setMessage] = useState<string | null>(null);

```

```

    
        async function handleReprocess(e: React.FormEvent) {
            e.preventDefault();
            setStatus("loading");
            setMessage(null);
            try {
                const res = await fetch("/api/admin/payments/reprocess", {
                    method: "POST",
                    headers: { "Content-Type": "application/json" },
                    body: JSON.stringify({ paymentIntentId: pi.trim() })
                });
                const json = await res.json().catch((err) => null);
                if (!res.ok || !json?.ok) {
                    setStatus("error");
                    setMessage(json?.error || "Falha a reprocessar PaymentIntent.");
                    return;
                }
                setStatus("success");
                setMessage(`Reprocessado: ${json.paymentIntentId || pi}`);
            } catch (err) {
                console.error(err);
                setStatus("error");
                setMessage("Erro inesperado. Ver logs.");
            }
        }

        return (
            <div className="space-y-3">
                <form onSubmit={handleReprocess} className="space-y-2 rounded-2xl border border-white/10 bg-white/5 p-3">
                    <label className="text-[11px] text-white/70">Reprocessar PaymentIntent</label>
                    <input
                        type="text"
                        value={pi}
                        onChange={({e}) => setPi(e.target.value)}
                        placeholder="pi_..."
                        className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-sm text-white outline-none
focus:border-white/30"
                        required
                    />
                    <button
                        type="submit"
                        disabled={status === "loading"}
                        className="inline-flex items-center justify-center rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-4 py-2 text-sm font-semibold text-black shadow disabled:opacity-60"
                    >
                        {status === "loading" ? "A reprocessar..." : "Reprocessar"}
                    </button>
                    {message && (
                        <p
                            className={`text-[11px] ${status === "success" ? "text-emerald-200" : "text-rose-200"}`}
                        >
                            {message}
                        </pp className="text-[10px] text-white/50">
                        Usa este botão se um webhook falhou. Idempotente: se já houver bilhetes, não duplica.
                    </pform>
            </div>
        );
    }
}


```

app/admin/eventos/page.tsx

```

    
        "use client";

import { useEffect, useState } from "react";
import Link from "next/link";
import { AdminLayout } from "@/app/admin/components/AdminLayout";
import { AdminTopActions } from "@/app/admin/components/AdminTopActions";

type AdminEventItem = {
    id: number;
    title: string;
    slug: string;
}


```

```

status: string;
type: string;
startsAt: string | null;
organizer?: { id: number; displayName: string | null } | null;
ticketsSold?: number;
revenueCents?: number;
revenueTotalCents?: number;
platformFeeCents?: number;
};

type EventsApiResponse =
| {
    ok: true;
    items: AdminEventItem[];
    pagination?: { nextCursor: number | null; hasMore: boolean };
}
| { ok: false; error?: string };

function formatDate(value?: string | null) {
    if (!value) return "-";
    const d = new Date(value);
    if (Number.isNaN(d.getTime())) return "-";
    return new Intl.DateTimeFormat("pt-PT", {
        day: "2-digit",
        month: "short",
        year: "numeric",
        hour: "2-digit",
        minute: "2-digit",
    }).format(d);
}

function formatCurrency(cents?: number | null, currency = "EUR") {
    if (cents == null || Number.isNaN(cents)) return "-";
    return new Intl.NumberFormat("pt-PT", {
        style: "currency",
        currency,
        minimumFractionDigits: 2,
    }).format(cents / 100);
}

function statusClasses(status: string) {
    switch (status) {
        case "PUBLISHED":
            return "bg-emerald-500/10 text-emerald-200 border-emerald-400/40";
        case "DRAFT":
            return "bg-amber-500/10 text-amber-100 border-amber-400/40";
        case "CANCELLED":
            return "bg-red-500/10 text-red-100 border-red-400/40";
        default:
            return "bg-white/5 text-white/80 border-white/20";
    }
}

function statusLabel(status: string) {
    switch (status) {
        case "PUBLISHED":
            return "Publicado";
        case "DRAFT":
            return "Rascunho";
        case "CANCELLED":
            return "Cancelado";
        default:
            return status;
    }
}

export default function AdminEventosPage() {
    const [search, setSearch] = useState("");
    const [statusFilter, setStatusFilter] = useState<string>("ALL");
    const [typeFilter, setTypeFilter] = useState<string>("ALL");
    const [organizerFilter, setOrganizerFilter] = useState("");
    const [loading, setLoading] = useState(false);
    const [errorMsg, setErrorMsg] = useState<string | null>(null);
    const [events, setEvents] = useState<AdminEventItem[]>([]);
    const [cursor, setCursor] = useState<number | null>(null);
    const [hasMore, setHasMore] = useState(false);
    const [initialized, setInitialized] = useState(false);
}

```

```

async function loadEvents(opts?: {
  search?: string;
  status?: string;
  type?: string;
  organizerId?: string;
  cursor?: number | null;
  reset?: boolean;
}) {
  try {
    setLoading(true);
    setErrorMsg(null);

    const sp = new URLSearchParams();
    const s = opts?.search ?? search;
    const st = opts?.status ?? statusFilter;
    const ty = opts?.type ?? typeFilter;
    const org = opts?.organizerId ?? organizerFilter;
    const cur = opts?.cursor ?? null;

    if (s.trim()) sp.set("search", s.trim());
    if (st !== "ALL") sp.set("status", st);
    if (ty !== "ALL") sp.set("type", ty);
    if (org.trim()) sp.set("organizerId", org.trim());
    if (cur) sp.set("cursor", String(cur));

    const qs = sp.toString() ? `${sp.toString()}` : "";
    const res = await fetch(`/api/admin/eventos/list${qs}`, { cache: "no-store" });

    if (res.status === 401 || res.status === 403) {
      setErrorMsg("Não tens permissões para ver esta área (admin only).");
      setEvents([]);
      return;
    }

    if (!res.ok) {
      const txt = await res.text().catch(() => "");
      console.error("[admin/eventos] erro:", res.status, txt);
      setErrorMsg("Não foi possível carregar os eventos.");
      setEvents([]);
      return;
    }

    const json = (await res.json()).catch(() => null) as EventsApiResponse | null;
    if (!json || !json.ok) {
      setErrorMsg(json?.error || "Resposta inesperada da API.");
      setEvents([]);
      return;
    }

    const merged = opts?.reset ? json.items ?? [] : [...events, ...(json.items ?? [])];
    setEvents(merged);
    setHasMore(json.pagination?.hasMore ?? false);
    setCursor(json.pagination?.nextCursor ?? null);
    setInitialized(true);
  } catch (err) {
    console.error("[admin/eventos] erro inesperado", err);
    setErrorMsg("Erro inesperado ao carregar eventos.");
    setEvents([]);
  } finally {
    setLoading(false);
  }
}

useEffect(() => {
  loadEvents({ reset: true });
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, []);

const isEmpty = initialized && !loading && events.length === 0 && !errorMsg;

return (
  <AdminLayout title="Eventos" subtitle="Gestão global de eventos com receita e bilhetes.">
    <section className="space-y-6">
      <div className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between">
        <div>
          <h1 className="text-2xl font-semibold tracking-tight md:text-3xl">Eventos</h1>
          <p className="mt-1 max-w-xl text-sm text-white/70">
            Lista de eventos com bilhetes vendidos e receita agregada.
          </p>
        </div>
      </div>
    </section>
  </AdminLayout>
)

```

```

        </p>
    </div>
    <AdminTopActions />
</div>

/* Filtros */
<div className="grid gap-3 md:grid-cols-[1.3fr_1fr_1fr_1fr] text-[11px]">
    <div className="flex items-center gap-2 rounded-2xl border border-white/15 bg-white/5 px-3 py-2">
        <span className="text-xs text-white/60">Pesquisar</span>
        <input
            type="text"
            value={search}
            onChange={(e) => setSearch(e.target.value)}
            onKeyDown={(e) => {
                if (e.key === "Enter") loadEvents({ search: e.currentTarget.value, reset: true });
            }}
            placeholder="Nome, slug ou organizador"
            className="w-full bg-transparent text-xs text-white placeholder:text-white/35 focus:outline-none"
        />
    </div>
    <div>
        <label className="mb-1 block text-[10px] text-white/60">Estado</label>
        <select
            value={statusFilter}
            onChange={(e) => {
                setStatusFilter(e.target.value);
                loadEvents({ status: e.target.value, reset: true });
            }}
            className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-xs text-white outline-none
focus:border-white/30"
        >
            <option value="ALL">Todos</option>
            <option value="PUBLISHED">Publicado</option>
            <option value="DRAFT">Rascunho</option>
            <option value="CANCELLED">Cancelado</option>
        </select>
    </div>
    <div>
        <label className="mb-1 block text-[10px] text-white/60">Tipo</label>
        <select
            value={typeFilter}
            onChange={(e) => {
                setTypeFilter(e.target.value);
                loadEvents({ type: e.target.value, reset: true });
            }}
            className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-xs text-white outline-none
focus:border-white/30"
        >
            <option value="ALL">Todos</option>
            <option value="ORGANIZER_EVENT">Organizador</option>
            <option value="EXPERIENCE">Experiência</option>
        </select>
    </div>
    <div>
        <label className="mb-1 block text-[10px] text-white/60">Organizer ID</label>
        <input
            type="text"
            value={organizerFilter}
            onChange={(e) => setOrganizerFilter(e.target.value)}
            placeholder="ex.: 3"
            className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-xs text-white outline-none
focus:border-white/30"
        />
    </div>
</div>
<div className="flex flex-wrap gap-2 text-[11px]">
    <button
        type="button"
        onClick={() => loadEvents({ search, organizerId: organizerFilter, reset: true })}
        disabled={loading}
        className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-4 py-1.5 text-xs font-semibold
text-black shadow-[0_0_18px_rgba(107,255,255,0.6)] hover:scale-[1.02] active:scale-95 transition-transform disabled:opacity-60
disabled:hover:scale-100"
    >
        {loading ? "A carregar..." : "Aplicar filtros"}
    </button>
    <button
        type="button"

```

```

    onClick={() => {
      setSearch("");
      setStatusFilter("ALL");
      setTypeFilter("ALL");
      setOrganizerFilter("");
      setEvents([]);
      setCursor(null);
      setHasMore(false);
      loadEvents({ search: "", status: "ALL", type: "ALL", organizerId: "", cursor: null, reset: true });
    }}
    className="rounded-full border border-white/20 px-3 py-1.5 text-white/75 hover:bg-white/10 transition"
  >
  Limpar
</button>
</div>

{errorMsg && (
  <div className="mt-4 rounded-xl border border-red-500/40 bg-red-500/10 px-4 py-3 text-xs text-red-100">
    {errorMsg}
  </div>
)}

{!errorMsg && isEmpty && (
  <div className="mt-6 rounded-2xl border border-dashed border-white/20 bg-white/5 px-6 py-8 text-center text-sm text-white/70">
    <p className="font-medium text-white">Ainda não existem eventos registados na plataforma.</p>
    <p className="mt-1 text-xs text-white/70">Assim que os organizadores começarem a criar eventos, eles vão aparecer aqui.</p>
  </div>
)}

{!errorMsg && !isEmpty && (
  <div className="mt-4 overflow-hidden rounded-2xl border border-white/15 bg-white/5">
    <div className="max-h-[70vh] overflow-auto">
      <table className="min-w-full border-separate border-spacing-0 text-left text-[11px]">
        <thead className="bg-white/5 text-white/60">
          <tr>
            <th className="sticky top-0 z-10 border-b border-white/10 px-4 py-3 font-medium">Evento</th>
            <th className="sticky top-0 z-10 border-b border-white/10 px-4 py-3 font-medium">Organizador</th>
            <th className="sticky top-0 z-10 border-b border-white/10 px-4 py-3 font-medium">Data</th>
            <th className="sticky top-0 z-10 border-b border-white/10 px-4 py-3 font-medium">Tipo</th>
            <th className="sticky top-0 z-10 border-b border-white/10 px-4 py-3 font-medium">Estado</th>
            <th className="sticky top-0 z-10 border-b border-white/10 px-4 py-3 font-medium">Bilhetes</th>
            <th className="sticky top-0 z-10 border-b border-white/10 px-4 py-3 font-medium">Receita</th>
            <th className="sticky top-0 z-10 border-b border-white/10 px-4 py-3 text-right font-medium">Ações</th>
          </tr>
        </thead>
        <tbody>
          {events.map((ev) => (
            <tr key={ev.id} className="border-b border-white/10 text-white/80 last:border-0">
              <td className="px-4 py-3 align-middle">
                <div className="flex flex-col">
                  <span className="text-[11px] font-semibold text-white">{ev.title || "Evento sem título"}</span>
                  {ev.slug && <span className="text-[10px] text-white/45">{events[ev.slug]}</span>}
                </div>
              </td>
              <td className="px-4 py-3 align-middle text-white/80">
                {ev.organizer?.displayName ?? "-"} {ev.organizer ? `(ID ${ev.organizer.id})` : ""}
              </td>
              <td className="px-4 py-3 align-middle text-white/80">{formatDate(ev.startsAt)}</td>
              <td className="px-4 py-3 align-middle text-white/80">{ev.type || "-"}</td>
              <td className="px-4 py-3 align-middle">
                <span
                  className={`inline-flex items-center rounded-full border px-2 py-[2px] text-[10px] font-medium
${statusClasses(
  ev.status
))}`}>
                  {statusLabel(ev.status)}
                </span>
              </td>
              <td className="px-4 py-3 align-middle text-white/80">{ev.ticketsSold ?? 0}</td>
              <td className="px-4 py-3 align-middle text-white/80">
                <div className="flex flex-col">
                  <span>Total: {formatCurrency(ev.revenueTotalCents ?? 0)}</span>
                  <span className="text-white/60">Fee: {formatCurrency(ev.platformFeeCents ?? 0)}</span>
                  <span className="text-white/60">Bruto: {formatCurrency(ev.revenueCents ?? 0)}</span>
                </div>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  )
)
}

```

```

        </td>
        <td className="px-4 py-3 align-middle text-right">
          <div className="flex flex-wrap justify-end gap-2">
            {ev.slug && (
              <Link
                href={`/eventos/${ev.slug}`}
                className="rounded-full border border-white/20 px-3 py-1 text-[10px] text-white/80 hover:bg-white/10 transition-colors"
              >
                Ver público
              </Link>
            )}
          </div>
        </td>
      </tr>
    ))}
  </tbody>
</table>
</div>
</div>
)}

<div className="flex items-center justify-between text-[11px] text-white/70">
  <span>{events.length} registros</span>
  <button
    type="button"
    disabled={!hasMore}
    onClick={() => {
      if (hasMore) loadEvents({ cursor, reset: false });
    }}
    className="rounded-full border border-white/20 px-3 py-1.5 disabled:opacity-40"
  >
    {hasMore ? "Carregar mais" : "Fim"}
  </button>
</div>
</section>
</AdminLayout>
);
}

```

app/admin/organizadores/page.tsx

```

"use client";

import { useEffect, useMemo, useState } from "react";

type OrganizerStatus = "PENDING" | "ACTIVE" | "SUSPENDED" | string;

type AdminOrganizerOwner = {
  id: string;
  username?: string | null;
  fullName?: string | null;
  email?: string | null;
};

type AdminOrganizerItem = {
  id: number | string;
  displayName: string;
  status: OrganizerStatus;
  createdAt: string;
  owner?: AdminOrganizerOwner | null;
  eventsCount?: number | null;
  totalTickets?: number | null;
  totalRevenueCents?: number | null;
};

type AdminOrganizersListResponse =
| {
  ok: true;
  organizers: AdminOrganizerItem[];
}
| {
  ok: false;
  error?: string;
}

```

```

    reason?: string;
};

const STATUS_LABEL: Record<OrganizerStatus, string> = {
  PENDING: "Pendente",
  ACTIVE: "Ativo",
  SUSPENDED: "Suspenso",
};

function formatStatusLabel(status: OrganizerStatus) {
  return STATUS_LABEL[status] ?? status;
}

function statusBadgeClasses(status: OrganizerStatus) {
  switch (status) {
    case "PENDING":
      return "border-amber-400/60 bg-amber-500/10 text-amber-100";
    case "ACTIVE":
      return "border-emerald-400/60 bg-emerald-500/10 text-emerald-100";
    case "SUSPENDED":
      return "border-red-400/60 bg-red-500/10 text-red-100";
    default:
      return "border-white/20 bg-white/5 text-white/80";
  }
}

function formatDate(value?: string | null) {
  if (!value) return "";
  const d = new Date(value);
  if (Number.isNaN(d.getTime())) return "";
  return new Intl.DateTimeFormat("pt-PT", {
    day: "2-digit",
    month: "short",
    year: "numeric",
    hour: "2-digit",
    minute: "2-digit",
  }).format(d);
}

function formatMoneyCents(value?: number | null) {
  if (!value || Number.isNaN(value)) return "0,00 €";
  const euros = value / 100;
  return new Intl.NumberFormat("pt-PT", {
    style: "currency",
    currency: "EUR",
    minimumFractionDigits: 2,
  }).format(euros);
}

function formatOwner(owner?: AdminOrganizerOwner | null) {
  if (!owner) return "Utilizador ORYA";
  if (owner.username) return `@$ {owner.username}`;
  if (owner.fullName) return owner.fullName;
  if (owner.email) return owner.email;
  return "Utilizador ORYA";
}

const FILTERS: { id: "ALL" | OrganizerStatus; label: string }[] = [
  { id: "ALL", label: "Todos" },
  { id: "PENDING", label: "Pendentes" },
  { id: "ACTIVE", label: "Ativos" },
  { id: "SUSPENDED", label: "Suspensos" },
];

export default function AdminOrganizadoresPage() {
  const [loading, setLoading] = useState(true);
  const [errorMsg, setErrorMsg] = useState<string | null>(null);
  const [accessIssue, setAccessIssue] = useState<"UNAUTH" | "FORBIDDEN" | null>(null);
  const pendingOrganizers = useMemo(
    () => organizers.filter((o) => o.status === "PENDING"),
    [organizers],
  );

```

```

useEffect(() => {
  let cancelled = false;

  async function loadOrganizers() {
    try {
      setLoading(true);
      setErrorMsg(null);
      setAccessIssue(null);

      const res = await fetch("/api/admin/organizadores/list", {
        method: "GET",
        headers: {
          "Content-Type": "application/json",
        },
        cache: "no-store",
      });

      if (res.status === 401) {
        if (!cancelled) setAccessIssue("UNAUTH");
        return;
      }

      if (res.status === 403) {
        if (!cancelled) setAccessIssue("FORBIDDEN");
        return;
      }

      if (!res.ok) {
        const text = await res.text();
        console.error("[admin/organizadores] Erro ao carregar:", text);
        if (!cancelled) {
          setErrorMsg(
            "Não foi possível carregar a lista de organizadores. Tenta novamente em alguns segundos.",
          );
        }
        return;
      }
    }

    const json = (await res.json()).catch(() => null) as
      | AdminOrganizersListResponse
      | null;

    if (!json || !json.ok) {
      if (!cancelled) {
        setErrorMsg(
          json?.error ||
          json?.reason ||
          "Resposta inesperada ao carregar organizadores.",
        );
      }
      return;
    }

    if (!cancelled) {
      setOrganizers(Array.isArray(json.organizers) ? json.organizers : []);
    }
  } catch (err) {
    console.error("[admin/organizadores] Erro inesperado:", err);
    if (!cancelled) {
      setErrorMsg(
        "Ocorreu um erro inesperado. Tenta novamente dentro de instantes.",
      );
    }
  } finally {
    if (!cancelled) setLoading(false);
  }
}

loadOrganizers();

return () => {
  cancelled = true;
};

}, [1]);

const filteredOrganizers = useMemo(() => {
  if (filter === "ALL") return organizers;
  return organizers.filter((o) => o.status === filter);
};

```

```

}, [organizers, filter]);

const stats = useMemo(() => {
  const total = organizers.length;
  const pending = organizers.filter((o) => o.status === "PENDING").length;
  const active = organizers.filter((o) => o.status === "ACTIVE").length;
  const suspended = organizers.filter((o) => o.status === "SUSPENDED").length;

  return { total, pending, active, suspended };
}, [organizers]);

async function updateStatus(
  organizerId: number | string,
  newStatus: OrganizerStatus,
) {
  try {
    setUpdatingId(organizerId);

    const res = await fetch("/api/admin/organizadores/update-status", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ organizerId, newStatus }),
    });

    const data = await res.json().catch(() => null);

    if (!res.ok || !data?.ok) {
      console.error("[admin/organizadores] Erro ao atualizar estado:", data);
      alert("Não foi possível atualizar o estado deste organizador.");
      return;
    }

    setOrganizers((prev) =>
      prev.map((org) =>
        String(org.id) === String(organizerId)
          ? { ...org, status: newStatus }
          : org,
      ),
    );
  } catch (err) {
    console.error("[admin/organizadores] Erro ao atualizar estado:", err);
    alert(
      "Ocorreu um erro inesperado ao atualizar o estado. Tenta novamente dentro de instantes.",
    );
  } finally {
    setUpdatingId(null);
  }
}

const hasOrganizers = filteredOrganizers.length > 0;

return (
<main className="orya-body-bg min-h-screen w-full text-white pb-16">
  <header className="border-b border-white/10 bg-black/60 backdrop-blur-xl">
    <div className="max-w-6xl mx-auto px-5 py-4 flex items-center justify-between gap-3">
      <span className="inline-flex h-8 w-8 items-center justify-center rounded-xl bg-gradient-to-tr from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-[11px] font-extrabold tracking-[0.16em]">
        AD
      </span>
      <div>
        <p className="text-[11px] uppercase tracking-[0.18em] text-white/55">
          Admin · Organizadores
        </p>
        <p className="text-sm text-white/85">
          Aprova e gere as contas de organizador na ORYA.
        </p>
      </div>
    </div>
  </header>

  <section className="max-w-6xl mx-auto px-5 pt-8 space-y-6">
    <div className="flex flex-col md:flex-row md:items-center md:justify-between gap-4">
      <div>
        <h1 className="text-2xl md:text-3xl font-semibold tracking-tight">

```

```

        Organizadores
    </h1>
    <p className="mt-1 text-sm text-white/70 max-w-xl">
        Vê quem está a organizar eventos na plataforma, aprova novos
        pedidos e suspende contas quando necessário.
    </p>
    {pendingOrganizers.length > 0 && (
        <p className="mt-1 text-[12px] text-amber-100/80">
            Tens {pendingOrganizers.length} pedido(s) pendente(s) de aprovação.
        </p>
    )}
</div>

<div className="grid grid-cols-2 sm:flex sm:flex-row gap-2 text-[11px]">
    <div className="rounded-xl border border-white/15 bg-black/60 px-3 py-2">
        <p className="text-[10px] text-white/55">Total</p>
        <p className="text-sm font-semibold">{stats.total}</p>
    </div>
    <div className="rounded-xl border border-amber-400/40 bg-amber-500/10 px-3 py-2">
        <p className="text-[10px] text-amber-100/80">Pendentes</p>
        <p className="text-sm font-semibold text-amber-100">
            {stats.pending}
        </p>
    </div>
    <div className="rounded-xl border border-emerald-400/40 bg-emerald-500/10 px-3 py-2">
        <p className="text-[10px] text-emerald-100/80">Ativos</p>
        <p className="text-sm font-semibold text-emerald-100">
            {stats.active}
        </p>
    </div>
    <div className="rounded-xl border border-red-400/40 bg-red-500/10 px-3 py-2">
        <p className="text-[10px] text-red-100/80">Suspensos</p>
        <p className="text-sm font-semibold text-red-100">
            {stats.suspended}
        </p>
    </div>
</div>
</div>
</div>

/* Filtros de estado */
<div className="flex flex-wrap items-center gap-2 text-[11px]">
    <span className="text-white/60">Filtrar por estado:</span>
    {FILTERS.map((f) => {
        const isActive = filter === f.id;
        return (
            <button
                key={f.id}
                type="button"
                onClick={() => setFilter(f.id)}
                className={
                    "rounded-full border px-3 py-1 transition-colors " +
                    (isActive
                        ? "border-[#6BFFFF] bg-[#6BFFFF]/10 text-[#6BFFFF]"
                        : "border-white/20 bg-white/5 text-white/70 hover:bg-white/10")
                }
            >
                {f.label}
            </button>
        );
    })}
</div>

/* Estados globais: acesso / erros / loading */
{accessIssue === "UNAUTH" && (
    <div className="mt-4 rounded-xl border border-amber-500/40 bg-amber-500/10 px-4 py-3 text-[11px] text-amber-50">
        <p className="font-medium">Sessão em falta</p>
        <p className="mt-1 text-amber-100/80">
            Para aceder ao painel de admin, tens de iniciar sessão com a tua
            conta de administrador.
        </p>
    </div>
)}

{accessIssue === "FORBIDDEN" && (
    <div className="mt-4 rounded-xl border border-red-500/40 bg-red-500/10 px-4 py-3 text-[11px] text-red-50">
        <p className="font-medium">Acesso restrito</p>
        <p className="mt-1 text-red-100/80">
            Esta área é exclusiva para contas com permissões de admin. Se
        </p>
    </div>
)}
```

```

        achas que isto é um erro, fala com o responsável pela plataforma.
    </p>
</div>
)}

{errorMsg && !accessIssue && (
    <div className="mt-4 rounded-xl border border-red-500/40 bg-red-500/10 px-4 py-3 text-[11px] text-red-50">
        <p className="font-medium">Não foi possível carregar</p>
        <p className="mt-1 text-red-100/80">{errorMsg}</p>
    </div>
)}

/* Lista rápida de pendentes */
{!loading && !accessIssue && !errorMsg && pendingOrganizers.length > 0 && (
    <div className="rounded-2xl border border-amber-400/40 bg-amber-500/10 px-4 py-3 text-[11px] text-amber-50">
        <div className="mb-2 flex items-center justify-between">
            <p className="font-semibold text-amber-50">Pedidos pendentes</p>
            <span className="rounded-full bg-amber-400/20 px-2 py-0.5 text-[10px] text-amber-100">
                {pendingOrganizers.length} para rever
            </span>
        </div>
        <div className="space-y-2">
            {pendingOrganizers.map((org) => (
                <div key={String(org.id)} className="flex flex-wrap items-center justify-between gap-2 rounded-xl border border-amber-300/30 bg-black/40 px-3 py-2">
                    >
                        <div className="flex flex-col">
                            <span className="text-white">{org.displayName}</span>
                            <span className="text-amber-100/80">
                                Dono: {formatOwner(org.owner)}
                            </span>
                        </div>
                        <div className="flex gap-2">
                            <button
                                type="button"
                                disabled={updatingId === org.id}
                                onClick={() => updateStatus(org.id, "ACTIVE")}
                                className="px-3 py-1 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold shadow-[0_0_14px_rgba(107,255,255,0.6)] disabled:opacity-60"
                            >
                                {updatingId === org.id ? "Aprovar..." : "Aprovar"}
                            </button>
                            <button
                                type="button"
                                disabled={updatingId === org.id}
                                onClick={() => updateStatus(org.id, "SUSPENDED")}
                                className="px-3 py-1 rounded-full border border-amber-200/60 text-amber-50 hover:bg-amber-200/10 disabled:opacity-60"
                            >
                                {updatingId === org.id ? "A atualizar..." : "Reprovar"}
                            </button>
                        </div>
                    </div>
                </div>
            ))}
        </div>
    </div>
)}

{loading && !accessIssue && !errorMsg && (
    <div className="mt-6 space-y-3" aria-hidden="true">
        <div className="h-10 w-full rounded-xl bg-white/5 animate-pulse" />
        <div className="h-10 w-full rounded-xl bg-white/5 animate-pulse" />
        <div className="h-10 w-full rounded-xl bg-white/5 animate-pulse" />
    </div>
)}

{!loading && !accessIssue && !errorMsg && !hasOrganizers && (
    <div className="mt-8 rounded-2xl border border-dashed border-white/20 bg-black/60 px-6 py-8 text-center space-y-3">
        <p className="text-base font-medium text-white/90">
            Ainda não há organizadores registados
        </p>
        <p className="text-[13px] text-white/65 max-w-md mx-auto">
            Assim que um utilizador fizer pedido para se tornar organizador,
            vais conseguir aprovar ou recusar essa conta a partir daqui.
        </p>
    </div>
)}

```

```

        )}

{!loading && !accessIssue && !errorMsg && hasOrganizers && (
  <div className="mt-4 space-y-3">
    {filteredOrganizers.map((org) => {
      const isPending = org.status === "PENDING";
      const isActive = org.status === "ACTIVE";
      const isSuspended = org.status === "SUSPENDED";

      return (
        <div
          key={String(org.id)}
          className="rounded-2xl border border-white/12 bg-black/70 px-4 py-3 flex flex-col gap-3 md:flex-row md:items-center md:justify-between"
        >
          <div className="space-y-1 text-[11px]">
            <div className="flex flex-wrap items-center gap-2">
              <span className="text-sm font-semibold text-white/90">
                {org.displayName}
              </span>
              <span
                className={
                  "inline-flex items-center rounded-full border px-2 py-[2px] text-[10px] font-medium " +
                  statusBadgeClasses(org.status)
                }
              >
                {formatStatusLabel(org.status)}
              </span>
            </div>
          </div>

          <p className="text-white/65">
            Dono: <span className="font-medium">{formatOwner(org.owner)}</span>
          </p>

          <div className="flex flex-wrap items-center gap-3 text-white/55">
            <span>Criado em {formatDate(org.createdAt)}</span>
            {typeof org.eventsCount === "number" && (
              <span>{org.eventsCount} evento(s)</span>
            )}
            {typeof org.totalTickets === "number" && (
              <span>{org.totalTickets} bilhete(s) vendidos</span>
            )}
            {typeof org.totalRevenueCents === "number" && (
              <span>
                Volume: {formatMoneyCents(org.totalRevenueCents)}
              </span>
            )}
          </div>
        </div>
      )
    }

    <div className="flex flex-wrap justify-end gap-2 text-[11px]">
      {isPending && (
        <>
          <button
            type="button"
            disabled={updatingId === org.id}
            onClick={() => updateStatus(org.id, "ACTIVE")}
            className="px-3 py-1.5 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] font-semibold text-black shadow-[0_0_18px_rgba(107,255,255,0.7)] hover:scale-[1.02] active:scale-95 transition-transform disabled:opacity-60 disabled:hover:scale-100"
          >
            {updatingId === org.id ? "A aprovar..." : "Aprovar"}
          </button>
          <button
            type="button"
            disabled={updatingId === org.id}
            onClick={() => updateStatus(org.id, "SUSPENDED")}
            className="px-3 py-1.5 rounded-full border border-white/25 text-white/80 hover:bg-white/10 transition-colors disabled:opacity-60"
          >
            {updatingId === org.id ? "A atualizar..." : "Rejeitar"}
          </button>
        </>
      )}

      {isActive && (
        <button
          type="button"

```

```

        disabled={updatingId === org.id}
        onClick={() => updateStatus(org.id, "SUSPENDED")}
        className="px-3 py-1.5 rounded-full border border-red-400/60 text-red-100 hover:bg-red-500/10
transition-colors disabled:opacity-60"
      >
  {updatingId === org.id ? "A suspender..." : "Suspender"}
  </button>
)}
```

```

{isSuspended && (
  <button
    type="button"
    disabled={updatingId === org.id}
    onClick={() => updateStatus(org.id, "ACTIVE")}
    className="px-3 py-1.5 rounded-full border border-emerald-400/60 text-emerald-100 hover:bg-emerald-500/10 transition-colors disabled:opacity-60"
  >
  {updatingId === org.id ? "A reativar..." : "Reativar"}
  </button>
)}
</div>
</div>
);
)})
</div>
)
</section>
</main>
);
}
}

```

app/admin/page.tsx

```

// app/admin/page.tsx

import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { redirect } from "next/navigation";
import Link from "next/link";
import { ReactNode } from "react";
import { AdminLayout } from "./components/AdminLayout";

export const dynamic = "force-dynamic";

function formatCurrencyFromCents(value: number | null | undefined) {
  if (!value || Number.isNaN(value)) return "0,00 €";
  const euros = value / 100;
  return new Intl.NumberFormat("pt-PT", {
    style: "currency",
    currency: "EUR",
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  }).format(euros);
}

function formatDateTime(value: Date | string | null | undefined) {
  if (!value) return "";
  const d = value instanceof Date ? value : new Date(value);
  if (Number.isNaN(d.getTime())) return "";
  return new Intl.DateTimeFormat("pt-PT", {
    day: "2-digit",
    month: "short",
    year: "numeric",
    hour: "2-digit",
    minute: "2-digit",
  }).format(d);
}

type BadgeTone = "neutral" | "positive" | "warning" | "danger";
type StatTone = "magenta" | "cyan" | "emerald" | "indigo";

function Badge({ children, tone = "neutral" }: { children: ReactNode; tone?: BadgeTone }) {
  const tones: Record<BadgeTone, string> = {
    neutral: "border-white/15 bg-white/10 text-white/80",

```

```

    positive: "border-emerald-300/50 bg-emerald-500/10 text-emerald-100",
    warning: "border-amber-300/60 bg-amber-500/15 text-amber-100",
    danger: "border-rose-300/60 bg-rose-500/15 text-rose-100",
  };
  return (
    <span className={`inline-flex items-center gap-1 rounded-full border px-2 py-1 text-[11px] ${tones[tone]}`}>
      {children}
    </span>
  );
}

function StatCard({
  label,
  value,
  helper,
  tone = "magenta",
}: {
  label: string;
  value: ReactNode;
  helper?: string;
  tone?: StatTone;
}) {
  const accents: Record<StatTone, string> = {
    magenta: "from-[#FF00C8]/16 via-[#8b5cf6]/20 to-[#1b1f32]/60",
    cyan: "from-[#6BFFFF]/20 via-[#1fb6ff]/18 to-[#0e1729]/70",
    emerald: "from-emerald-400/14 via-emerald-500/12 to-[#0e1f1a]/80",
    indigo: "from-indigo-400/14 via-indigo-500/12 to-[#0e1226]/80",
  };
  return (
    <div className="relative overflow-hidden rounded-2xl border border-white/10 bg-black/70 p-4 shadow-[0_12px_40px_rgba(0,0,0,0.35)]">
      <div className="pointer-events-none absolute inset-0 bg-gradient-to-br ${accents[tone]} aria-hidden />
      <div className="relative space-y-1">
        <p className="text-[11px] uppercase tracking-[0.16em] text-white/60">{label}</p>
        <div className="flex items-end gap-2">
          <p className="text-3xl font-semibold leading-tight">{value}</p>
        </div>
        {helper && <p className="text-[11px] text-white/60">{helper}</p>}
      </div>
    </div>
  );
}

function SectionCard({
  title,
  action,
  children,
}: {
  title: string;
  action?: ReactNode;
  children: ReactNode;
}) {
  return (
    <div className="rounded-2xl border border-white/12 bg-black/70 p-5 shadow-[0_12px_40px_rgba(0,0,0,0.35)]">
      <div className="mb-4 flex items-center justify-between gap-3">
        <h2 className="text-sm font-semibold text-white/90">{title}</h2>
        {action}
      </div>
      {children}
    </div>
  );
}

function toneForStatus(status: string | null | undefined): BadgeTone {
  const normalized = (status || "").toUpperCase();
  if (["PUBLISHED", "ACTIVE", "SUCCEEDED", "CONFIRMED"].includes(normalized)) return "positive";
  if (["PENDING", "DRAFT", "PROCESSING"].includes(normalized)) return "warning";
  if (["CANCELLED", "FAILED", "SUSPENDED"].includes(normalized)) return "danger";
  return "neutral";
}

export default async function AdminDashboardPage() {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
}

```

```

if (!user) {
  // Sem sessão, não há painel de admin
  redirect("/");
}

const profile = await prisma.profile.findUnique({
  where: { id: user.id },
});

const roles = Array.isArray(profile?.roles) ? profile?.roles : [];
const isAdmin = roles?.includes("admin");

if (!isAdmin) {
  // Utilizador autenticado mas sem role de admin
  redirect("/");
}

// prisma.paymentEvent pode não existir se o client não estiver regenerado;
// fallback defensivo para evitar crash em caso de client antigo.
const paymentEventQuery = prisma.paymentEvent
  ? prisma.paymentEvent.findMany({
      orderBy: { createdAt: "desc" },
      take: 8,
      select: {
        id: true,
        stripePaymentIntentId: true,
        status: true,
        eventId: true,
        amountCents: true,
        platformFeeCents: true,
        errorMessage: true,
        createdAt: true,
        updatedAt: true,
      },
    })
  : Promise.resolve([]);

const [usersCount, organizersCount, eventsCount, ticketsCount, revenueAgg, recentEvents, recentTickets, recentPaymentEvents] =
  await Promise.all([
    prisma.profile.count(),
    prisma.organizer.count(),
    prisma.event.count(),
    prisma.ticket.count(),
    prisma.ticket.aggregate({
      _sum: {
        pricePaid: true,
      },
    }),
    prisma.event.findMany({
      orderBy: { createdAt: "desc" },
      take: 5,
      select: {
        id: true,
        title: true,
        slug: true,
        startsAt: true,
        status: true,
      },
    }),
    prisma.ticket.findMany({
      orderBy: { purchasedAt: "desc" },
      take: 5,
      select: {
        id: true,
        purchasedAt: true,
        pricePaid: true,
        currency: true,
        event: {
          select: {
            slug: true,
            title: true,
          },
        },
      },
    }),
    paymentEventQuery,
  ]);

```

```

const totalRevenueCents = revenueAgg._sum.pricePaid ?? 0;

return (
  <AdminLayout
    title="Admin ORYA - visão geral da plataforma"
    subtitle="Monitoriza utilizadores, organizadores, eventos, bilhetes e pagamentos num só ecrã."
  >
  <section className="space-y-8">
    <div className="space-y-3">
      <div className="space-y-1">
        <h1 className="text-2xl font-semibold tracking-tight md:text-3xl">
          Painel geral
        </h1>
        <p className="max-w-2xl text-sm text-white/70">
          Estado global da ORYA e atalhos para as áreas críticas. Usa os links rápidos abaixo para abrir listas detalhadas.
        </p>
      </div>
      <div className="flex flex-wrap gap-2 text-[11px]">
        <Link
          href="/admin/organizadores"
          className="inline-flex items-center gap-1 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-white/80
transition hover:border-white/25 hover:bg-white/10"
        >
          Gerir pedidos de organizador
        </Link>
        <Link
          href="/admin/eventos"
          className="inline-flex items-center gap-1 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-white/80
transition hover:border-white/25 hover:bg-white/10"
        >
          Ver eventos
        </Link>
        <Link
          href="/admin/payments"
          className="inline-flex items-center gap-1 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-white/80
transition hover:border-white/25 hover:bg-white/10"
        >
          Pagamentos / intents
        </Link>
        <Link
          href="/admin/tickets"
          className="inline-flex items-center gap-1 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-white/80
transition hover:border-white/25 hover:bg-white/10"
        >
          Auditoria de bilhetes
        </Link>
        <Link
          href="/admin/settings"
          className="inline-flex items-center gap-1 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-white/80
transition hover:border-white/25 hover:bg-white/10"
        >
          Configurações
        </Link>
      </div>
    </div>
    <div className="grid gap-4 md:grid-cols-3 lg:grid-cols-5">
      <StatCard
        label="Utilizadores"
        value={usersCount}
        helper="Contas com perfil criado."
        tone="magenta"
      />
      <StatCard
        label="Organizadores"
        value={organizersCount}
        helper="Entidades a criar eventos pagos."
        tone="emerald"
      />
      <StatCard
        label="Eventos"
        value={eventsCount}
        helper="Total de eventos registados."
        tone="indigo"
      />
      <StatCard
        label="Bilhetes"
        value={ticketsCount}
      />
    </div>
  </section>
)

```

```

    helper="Total de bilhetes emitidos."
    tone="indigo"
/>
<StatCard
  label="Volume total"
  value={formatCurrencyFromCents(totalRevenueCents)}
  helper="Soma dos pagamentos processados."
  tone="cyan"
/>
</div>

<div className="grid gap-6 lg:grid-cols-2">
  <SectionCard
    title="Últimos eventos criados"
    action={
      <Link href="/admin/eventos" className="text-[12px] text-white/70 hover:text-white">
        Ver todos
      </Link>
    }
  >
    {recentEvents.length === 0 ? (
      <p className="text-[12px] text-white/60">Ainda não há eventos registados.</p>
    ) : (
      <div className="divide-y divide-white/5">
        {recentEvents.map((ev) => (
          <div key={ev.id} className="flex flex-col gap-1 py-3 sm:flex-row sm:items-center sm:justify-between">
            <div className="space-y-0.5">
              <p className="text-[13px] font-medium text-white/90">{ev.title}</p>
              <p className="text-[12px] text-white/55">{formatDateTime(ev.startsAt)}</p>
            </div>
            <Badge tone={toneForStatus(ev.status)}>{ev.status ?? "-"}</Badge>
          </div>
        )));
      </div>
    )}
  </SectionCard>

  <SectionCard
    title="Últimos bilhetes vendidos"
    action={
      <Link href="/admin/tickets" className="text-[12px] text-white/70 hover:text-white">
        Abrir auditoria
      </Link>
    }
  >
    {recentTickets.length === 0 ? (
      <p className="text-[12px] text-white/60">Ainda não há bilhetes registados.</p>
    ) : (
      <div className="divide-y divide-white/5">
        {recentTickets.map((t) => (
          <div key={t.id} className="flex flex-col gap-1 py-3 sm:flex-row sm:items-center sm:justify-between">
            <div className="space-y-0.5">
              <p className="text-[13px] font-medium text-white/90">
                {t.event?.title ?? "Evento ORYA"}
              </p>
              <p className="text-[12px] text-white/55">
                Vendido em {formatDateTime(t.purchasedAt)}
              </p>
            </div>
            <p className="text-sm font-semibold text-white/90">
              {formatCurrencyFromCents(t.pricePaid)}
            </p>
          </div>
        )));
      </div>
    )}
  </SectionCard>
</div>

<SectionCard
  title="Pagamentos / intents recentes"
  action={
    <Link href="/admin/payments" className="text-[12px] text-white/70 hover:text-white">
      Ver pagamentos
    </Link>
  }
>
  {recentPaymentEvents.length === 0 ? (

```

```

    <p className="text-[12px] text-white/60">Sem eventos de pagamento registados.</p>
  ) : (
    <div className="divide-y divide-white/5 text-[13px]">
      {recentPaymentEvents.map((p) => (
        <div
          key={p.id}
          className="flex flex-col gap-2 py-3 sm:flex-row sm:items-center sm:justify-between"
        >
          <div className="flex flex-1 items-center gap-3">
            <Badge tone={toneForStatus(p.status)}>{p.status ?? "-"}</Badge>
            <div className="space-y-0.5">
              <p className="font-medium text-white/90">
                {p.stripePaymentIntentId ?? "Intent"}
              </p>
              <p className="text-[12px] text-white/55">
                Atualizado {formatDateTime(p.updatedAt)}
              </p>
            </div>
          </div>
          <div className="text-right">
            <p className="text-sm font-semibold text-white/90">
              {formatCurrencyFromCents(p.amountCents ?? 0)}
            </p>
            <p className="text-[11px] text-white/55">
              Fee plataforma {formatCurrencyFromCents(p.platformFeeCents ?? 0)}
            </p>
          </div>
        {p.errorMessage && (
          <p className="text-[11px] text-rose-200">Erro: {p.errorMessage}</p>
        )}
      </div>
    )));
  )
  </SectionCard>
</section>
</AdminLayout>
);
}

```

app/admin/payments/page.tsx

```

"use client";

import { useMemo, useState } from "react";
import useSWR from "swr";
import Link from "next/link";
import { AdminLayout } from "@app/admin/components/AdminLayout";
import { AdminTopActions } from "@app/admin/components/AdminTopActions";

type PaymentEvent = {
  id: number;
  stripePaymentIntentId: string | null;
  status: string;
  mode?: "LIVE" | "TEST" | null;
  isTest?: boolean;
  eventId: number | null;
  userId: string | null;
  amountCents: number | null;
  platformFeeCents: number | null;
  errorMessage: string | null;
  createdat: string;
  updatedat: string;
};

type ApiResponse =
  | { ok: true; items: PaymentEvent[]; pagination: { nextCursor: number | null; hasMore: boolean } }
  | { ok: false; error?: string };

type Aggregate = {
  grossCents: number;
  discountCents: number;
  platformFeeCents: number;
  stripeFeeCents: number;
  netCents: number;
  tickets: number;
};

```

```

};

type OverviewResponse =
| {
  ok: true;
  totals: Aggregate;
  byOrganizer: {
    organizerId: number;
    grossCents: number;
    discountCents: number;
    platformFeeCents: number;
    stripeFeeCents: number;
    netCents: number;
    tickets: number;
    events: number;
  }[];
  period: { from: string | Date | null; to: string | Date | null };
}
| { ok: false; error?: string };

const fetcher = (url: string) => fetch(url).then((r) => r.json());

function formatMoney(cents?: number | null, currency = "EUR") {
  if (cents == null || Number.isNaN(cents)) return "-";
  return new Intl.NumberFormat("pt-PT", {
    style: "currency",
    currency,
    minimumFractionDigits: 2,
  }).format(cents / 100);
}

function formatDate(value?: string | null) {
  if (!value) return "";
  const d = new Date(value);
  if (Number.isNaN(d.getTime())) return "";
  return new Intl.DateTimeFormat("pt-PT", {
    day: "2-digit",
    month: "short",
    year: "numeric",
    hour: "2-digit",
    minute: "2-digit",
  }).format(d);
}

export default function AdminPaymentsPage() {
  const [status, setStatus] = useState<string>("ALL");
  const [mode, setMode] = useState<string>("ALL");
  const [q, setQ] = useState("");
  const [cursor, setCursor] = useState<number | null>(null);
  const [organizerId, setOrganizerId] = useState("");
  const [eventId, setEventId] = useState("");
  const [from, setFrom] = useState("");
  const [to, setTo] = useState("");

  const queryParams = useMemo(() => {
    const params = new URLSearchParams();
    if (status !== "ALL") params.set("status", status);
    if (mode !== "ALL") params.set("mode", mode);
    if (q.trim()) params.set("q", q.trim());
    if (cursor) params.set("cursor", String(cursor));
    return params.toString() ? `${params.toString()}` : "";
  }, [status, mode, q, cursor]);

  const { data, isLoading, mutate } = useSWR<ApiResponse>(
    `/api/admin/payments/list${queryParams}`,
    fetcher,
    { revalidateOnFocus: false },
  );

  const overviewParams = useMemo(() => {
    const params = new URLSearchParams();
    if (mode !== "ALL") params.set("mode", mode);
    if (organizerId.trim()) params.set("organizerId", organizerId.trim());
    if (eventId.trim()) params.set("eventId", eventId.trim());
    if (from) params.set("from", from);
    if (to) params.set("to", to);
    return params.toString() ? `${params.toString()}` : "";
  }, [mode, organizerId, eventId, from, to]);
}

```

```

const { data: overview } = useSWR<OverviewResponse>(
  `/api/admin/payments/overview${overviewParams}`,
  fetcher,
  { revalidateOnFocus: false },
);

const items = data && "ok" in data && data.ok ? data.items : [];
const pagination = data && "ok" in data && data.ok ? data.pagination : { nextCursor: null, hasMore: false };
const errorMsg = data && "ok" in data && !data.ok ? data.error || "Falha ao carregar intents." : null;
const overviewTotals = overview && "ok" in overview && overview.ok ? overview.totals : null;
const overviewByOrganizer = overview && "ok" in overview && overview.ok ? overview.byOrganizer : [];
const overviewError = overview && "ok" in overview && !overview.ok ? overview.error || "Falha ao carregar overview." : null;

function badgeClass(s: string) {
  switch (s) {
    case "OK":
      return "bg-emerald-500/15 text-emerald-100 border border-emerald-400/30";
    case "REFUNDED":
      return "bg-amber-500/15 text-amber-100 border border-amber-400/30";
    case "ERROR":
      return "bg-rose-500/15 text-rose-100 border border-rose-400/30";
    case "PROCESSING":
    default:
      return "bg-white/10 text-white/80 border border-white/20";
  }
}

function modeBadgeClass(m?: string | null) {
  if (m === "TEST") return "bg-rose-500/15 text-rose-100 border border-rose-400/30";
  if (m === "LIVE") return "bg-white/10 text-white/80 border border-white/20";
  return "bg-white/5 text-white/60 border border-white/10";
}

async function handleReprocess(pi: string | null | undefined) {
  if (!pi) return;
  await fetch("/api/admin/payments/reprocess", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ paymentIntentId: pi }),
  });
  await mutate();
}

async function handleRefund(pi: string | null | undefined) {
  if (!pi) return;
  await fetch("/api/admin/payments/refund", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ paymentIntentId: pi }),
  });
  await mutate();
}

return (
  <AdminLayout title="Pagamentos / intents" subtitle="Consulta os payment_events, reprocessa intents e refunds.">
    <main className="flex-1 p-6 space-y-6">

      <section className="space-y-6">
        <div className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between">
          <div>
            <h1 className="text-2xl font-semibold tracking-tight md:text-3xl">Pagamentos / intents</h1>
            <p className="mt-1 max-w-2xl text-sm text-white/70">Consulta payment_events, reprocessa ou faz refund.</p>
          </div>
          <AdminTopActions showPaymentsExport />
        </div>
      </section>

      {/* Filtros */}
      <div className="rounded-2xl border border-white/10 bg-white/5 p-4 backdrop-blur">
        <div className="space-y-4">
          <div className="grid gap-3 md:grid-cols-[1.5fr_1fr_1fr]">
            <div>
              <label className="mb-1 block text-[11px] text-white/70">Pesquisa</label>
              <input type="text" value={q} onChange={(e) => setQ(e.target.value)} placeholder="Intent id, event id..." />
            </div>
          </div>
        </div>
      </div>
    </main>
  </AdminLayout>
)

```

```

        className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-sm text-white outline-none
focus:border-white/30"
      />
    </div>
    <div>
      <label className="mb-1 block text-[11px] text-white/70">Estado</label>
      <select
        value={status}
        onChange={(e) => {
          setStatus(e.target.value);
          setCursor(null);
        }}
        className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-sm text-white outline-none
focus:border-white/30"
      >
        <option value="ALL">Todos</option>
        <option value="PROCESSING">Processing</option>
        <option value="OK">OK</option>
        <option value="REFUNDED">Refunded</option>
        <option value="ERROR">Error</option>
      </select>
    </div>
    <div>
      <label className="mb-1 block text-[11px] text-white/70">Modo</label>
      <select
        value={mode}
        onChange={(e) => {
          setMode(e.target.value);
          setCursor(null);
        }}
        className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-sm text-white outline-none
focus:border-white/30"
      >
        <option value="ALL">Todos</option>
        <option value="LIVE">Live</option>
        <option value="TEST">Test</option>
      </select>
    </div>
    <div className="flex items-end gap-2">
      <button
        type="button"
        onClick={() => {
          setCursor(null);
          void mutate();
        }}
        className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-4 py-2 text-sm font-
semibold text-black shadow"
      >
        Atualizar
      </button>
    </div>
  </div>

  <div className="grid gap-3 md:grid-cols-4">
    <div>
      <label className="mb-1 block text-[11px] text-white/70">Organizer ID</label>
      <input
        type="text"
        value={organizerId}
        onChange={(e) => setOrganizerId(e.target.value)}
        placeholder="ex: 12"
        className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-sm text-white outline-none
focus:border-white/30"
      />
    </div>
    <div>
      <label className="mb-1 block text-[11px] text-white/70">Evento ID</label>
      <input
        type="text"
        value={eventId}
        onChange={(e) => setEventId(e.target.value)}
        placeholder="ex: 340"
        className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-sm text-white outline-none
focus:border-white/30"
      />
    </div>
    <div>
      <label className="mb-1 block text-[11px] text-white/70">De</label>

```

```

<input
  type="date"
  value={from}
  onChange={(e) => setFrom(e.target.value)}
  className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-sm text-white outline-none
focus:border-white/30"
/>
</div>
<div>
  <label className="mb-1 block text-[11px] text-white/70">Até</label>
  <input
    type="date"
    value={to}
    onChange={(e) => setTo(e.target.value)}
    className="w-full rounded-xl border border-white/15 bg-black/50 px-3 py-2 text-sm text-white outline-none
focus:border-white/30"
  /
>
</div>
</div>
</div>

/* Lista */
<div className="rounded-2xl border border-white/10 bg-black/50 backdrop-blur">
  <div className="border-b border-white/10 px-4 py-3">
    <h3 className="text-sm font-semibold text-white">Visão global (sale_summaries)</h3>
    <p className="text-[11px] text-white/60">
      Filtros de modo/organizer/evento/periodo aplicados. Valores estimam fee Stripe base.
    </p>
  </div>
  {overviewError && (
    <div className="px-4 py-3 text-[12px] text-rose-200">{overviewError}</div>
  )}
  {!overviewError && (
    <div className="grid gap-4 px-4 py-4 md:grid-cols-5">
      <div className="rounded-xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] uppercase tracking-wide text-white/60">Bruto</p>
        <p className="text-lg font-semibold">{formatMoney(overviewTotals?.grossCents ?? 0)}</p>
      </div>
      <div className="rounded-xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] uppercase tracking-wide text-white/60">Descontos</p>
        <p className="text-lg font-semibold text-emerald-200">{formatMoney(overviewTotals?.discountCents ?? 0)}</p>
      </div>
      <div className="rounded-xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] uppercase tracking-wide text-white/60">Taxa ORYA</p>
        <p className="text-lg font-semibold text-orange-200">{formatMoney(overviewTotals?.platformFeeCents ?? 0)}</p>
      </div>
      <div className="rounded-xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] uppercase tracking-wide text-white/60">Fee Stripe (estim.)</p>
        <p className="text-lg font-semibold text-white">{formatMoney(overviewTotals?.stripeFeeCents ?? 0)}</p>
      </div>
      <div className="rounded-xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] uppercase tracking-wide text-white/60">Líquido</p>
        <p className="text-lg font-semibold text-emerald-100">{formatMoney(overviewTotals?.netCents ?? 0)}</p>
        <p className="text-[11px] text-white/50">{(overviewTotals?.tickets ?? 0)} bilhetes</p>
      </div>
    </div>
  )}

{overviewByOrganizer && overviewByOrganizer.length > 0 && (
  <div className="px-4 pb-4">
    <div className="overflow-auto rounded-xl border border-white/10">
      <table className="min-w-full text-left text-xs text-white/90">
        <thead className="bg-black/70">
          <tr className="border-b border-white/10 text-[11px] uppercase tracking-[0.16em] text-white/60">
            <th className="px-3 py-3">Organizer</th>
            <th className="px-3 py-3">Bruto</th>
            <th className="px-3 py-3">Taxa ORYA</th>
            <th className="px-3 py-3">Fee Stripe (est.)</th>
            <th className="px-3 py-3">Líquido</th>
            <th className="px-3 py-3">Bilhetes</th>
            <th className="px-3 py-3">Eventos</th>
          </tr>
        </thead>
        <tbody>
          {overviewByOrganizer.map((o) => (
            <tr key={o.organizerId} className="border-b border-white/5">
              <td className="px-3 py-2">#{o.organizerId}</td>

```

```

        <td className="px-3 py-2">{formatMoney(o.grossCents)}</td>
        <td className="px-3 py-2">{formatMoney(o.platformFeeCents)}</td>
        <td className="px-3 py-2">{formatMoney(o.stripeFeeCents)}</td>
        <td className="px-3 py-2 font-semibold text-emerald-100">{formatMoney(o.netCents)}</td>
        <td className="px-3 py-2">{o.tickets}</td>
        <td className="px-3 py-2">{o.events}</td>
    </tr>
    ))}
  </tbody>
</table>
</div>
</div>
)
}

</div>

<div className="rounded-2xl border border-white/10 bg-black/50 backdrop-blur">
<div className="max-h-[70vh] overflow-auto">
  <table className="min-w-full text-left text-xs text-white/90">
    <thead className="sticky top-0 z-10 bg-black/80">
      <tr className="border-b border-white/10 text-[11px] uppercase tracking-[0.16em] text-white/60">
        <th className="px-3 py-3">Intent</th>
        <th className="px-3 py-3">Estado</th>
        <th className="px-3 py-3">Evento</th>
        <th className="px-3 py-3">Montantes</th>
        <th className="px-3 py-3">Atualizado</th>
        <th className="px-3 py-3">Ações</th>
      </tr>
    </thead>
    <tbody>
      {errorMsg &&
        <tr>
          <td colSpan={6} className="px-3 py-4 text-center text-rose-200">
            {errorMsg}
          </td>
        </tr>
      )}
      {isLoading && (
        <tr>
          <td colSpan={6} className="px-3 py-4 text-center text-white/60">
            A carregar...
          </td>
        </tr>
      )}
      {!isLoading && !errorMsg && (!items || items.length === 0) && (
        <tr>
          <td colSpan={6} className="px-3 py-4 text-center text-white/60">
            Sem registos para estes filtros.
          </td>
        </tr>
      )}
      {!isLoading &&
        !errorMsg &&
        items?.map((p) =>
          <tr key={p.id} className="border-b border-white/5 hover:bg-white/5">
            <td className="px-3 py-3 align-top font-mono text-[11px] break-all">
              {p.stripePaymentIntentId || "-"}
            </td>
            <td className="px-3 py-3 align-top">
              <span className={`inline-flex rounded-full px-2 py-0.5 text-[11px] ${badgeClass(p.status)}`}>
                {p.status}
              </span>
              <div className="mt-1">
                <span className={`inline-flex rounded-full px-2 py-0.5 text-[10px] uppercase tracking-wide ${modeBadgeClass(p.mode) || (p.isTest ? "TEST" : "LIVE")}`}>
                  {p.mode || (p.isTest ? "TEST" : "LIVE")}
                </span>
              </div>
            <td errorMessage && (
              <p className="mt-1 text-[10px] text-rose-200">{p.errorMessage}</p>
            )>
              <td className="px-3 py-3 align-top text-[11px] text-white/80">
                {p.eventId ? <>Event #{p.eventId}</> : "-"}
              </td>
              <td className="px-3 py-3 align-top text-[11px] text-white/80">
                <div className="flex flex-col">
                  <span>Total: {formatMoney(p.amountCents)}</span>
                  <span className="text-white/60">Fee: {formatMoney(p.platformFeeCents)}</span>
                </div>
              </td>
            </td>
          </tr>
        )
      )}
    </tbody>
  </table>
</div>
</div>

```

```

        </div>
    </td>
    <td className="px-3 py-3 align-top text-[11px] text-white/70">
        {formatDate(p.updatedAt)}
        <div className="text-white/50">{formatDate(p.createdAt)}</div>
    </td>
    <td className="px-3 py-3 align-top">
        <div className="flex flex-wrap gap-2 text-[11px]">
            <button
                type="button"
                onClick={() =>
                    navigator.clipboard?.writeText(p.stripePaymentIntentId || "");
                }
                className="rounded-full border border-white/20 px-2 py-1 text-white/80 hover:bg-white/10"
            >
                Copiar ID
            </button>
            <button
                type="button"
                onClick={() => handleReprocess(p.stripePaymentIntentId || "")}
                className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-2.5 py-1 text-black font-semibold shadow hover:opacity-90"
            >
                Reprocessar
            </button>
            <button
                type="button"
                onClick={() => handleRefund(p.stripePaymentIntentId || "")}
                className="rounded-full border border-white/20 px-2.5 py-1 text-white/85 hover:bg-white/10"
            >
                Refund
            </button>
        </div>
    </td>
    </tr>
    ))}
</tbody>
</table>
</div>
</div>

/* Paginação */
<div className="flex items-center justify-between text-[11px] text-white/70">
    <span>
        {items?.length || 0} registos
    </span>
    <div className="flex gap-2">
        <button
            type="button"
            disabled={!cursor}
            onClick={() => setCursor(null)}
            className="rounded-full border border-white/20 px-3 py-1.5 disabled:opacity-40"
        >
            Início
        </button>
        <button
            type="button"
            disabled={!pagination?.nextCursor}
            onClick={() => setCursor(pagination?.nextCursor ?? null)}
            className="rounded-full border border-white/20 px-3 py-1.5 disabled:opacity-40"
        >
            Mais
        </button>
    </div>
</div>
</section>
</main>
</AdminLayout>
);
}

```

app/admin/settings/page.tsx

```

"use client";

import { useEffect, useMemo, useState } from "react";

```

```

import useSWR from "swr";
import Link from "next/link";

type FeesResponse =
| {
  ok: true;
  orya: { feeBps: number; feeFixedCents: number };
  stripe: { feeBps: number; feeFixedCents: number; region: string };
}
| { ok: false; error?: string };

const fetcher = (url: string) => fetch(url).then((r) => r.json());

function formatPercentFromBps(bps: number) {
  return (bps / 100).toFixed(2) + "%";
}

function formatEur(cents: number) {
  return (cents / 100).toFixed(2) + " €";
}

function computeFee(baseCents: number, feeBps: number, feeFixedCents: number) {
  return Math.round((baseCents * feeBps) / 10_000) + feeFixedCents;
}

export default function AdminSettingsPage() {
  const { data, isLoading, mutate } = useSWR<FeesResponse>("/api/admin/fees", fetcher, {
    revalidateOnFocus: false,
  });

  const [platformPercent, setPlatformPercent] = useState("8.00");
  const [platformFixed, setPlatformFixed] = useState("0.30");
  const [stripePercent, setStripePercent] = useState("1.40");
  const [stripeFixed, setStripeFixed] = useState("0.25");
  const [saving, setSaving] = useState(false);
  const [saveError, setSaveError] = useState<string | null>(null);
  const [saveSuccess, setSaveSuccess] = useState<string | null>(null);

  useEffect(() => {
    if (data && data.ok) {
      setPlatformPercent((data.orya.feeBps / 100).toFixed(2));
      setPlatformFixed((data.orya.feeFixedCents / 100).toFixed(2));
      setStripePercent((data.stripe.feeBps / 100).toFixed(2));
      setStripeFixed((data.stripe.feeFixedCents / 100).toFixed(2));
    }
  }, [data]);

  const parsedPlatformBps = useMemo(
    () => Math.max(0, Math.round(Number(platformPercent || "0") * 100)),
    [platformPercent],
  );
  const parsedPlatformFixedCents = useMemo(
    () => Math.max(0, Math.round(Number(platformFixed || "0") * 100)),
    [platformFixed],
  );
  const parsedStripeBps = useMemo(() => Math.max(0, Math.round(Number(stripePercent || "0") * 100)), [stripePercent]);
  const parsedStripeFixedCents = useMemo(
    () => Math.max(0, Math.round(Number(stripeFixed || "0") * 100)),
    [stripeFixed],
  );

  const sampleBase = 1000; // 10€ em cêntimos
  const oryaFeeCents = computeFee(sampleBase, parsedPlatformBps, parsedPlatformFixedCents);
  const stripeFeeOnTopCents = computeFee(sampleBase + oryaFeeCents, parsedStripeBps, parsedStripeFixedCents);
  const stripeFeeIncludedCents = computeFee(sampleBase, parsedStripeBps, parsedStripeFixedCents);
  const organizerNetOnTop = Math.max(0, sampleBase - stripeFeeOnTopCents);
  const organizerNetIncluded = Math.max(0, sampleBase - oryaFeeCents - stripeFeeIncludedCents);

  const handleSave = async () => {
    setSaveError(null);
    setSaveSuccess(null);
    setSaving(true);
    try {
      const res = await fetch("/api/admin/fees", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          platformFeeBps: parsedPlatformBps,

```

```

    platformFeeFixedCents: parsedPlatformFixedCents,
    stripeFeeBpsEu: parsedStripeBps,
    stripeFeeFixedCentsEu: parsedStripeFixedCents,
  )),
});

const json = (await res.json()) as FeesResponse;
if (!res.ok || !json || !( "ok" in json ) || !json.ok) {
  const msg = (typeof json === "object" && json && "error" in json && typeof json.error === "string")
    ? json.error
    : "Não foi possível guardar as taxas.";
  setSaveError(msg);
  return;
}
setSaveSuccess("Taxas guardadas. Já estão a ser usadas nos cálculos.");
await mutate();
} catch (err) {
  console.error("Erro a guardar taxas", err);
  setSaveError("Erro inesperado ao guardar.");
} finally {
  setSaving(false);
}
};

return (
<main className="orya-body-bg min-h-screen text-white pb-16">
<header className="border-b border-white/10 bg-black/50 backdrop-blur-xl">
<div className="mx-auto flex max-w-6xl items-center justify-between px-5 py-4">
<div className="flex items-center gap-3">
<span className="inline-flex h-8 w-8 items-center justify-center rounded-xl bg-gradient-to-tr from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-xs font-extrabold tracking-[0.15em]">
  AD
</span>
<div>
  <p className="text-xs uppercase tracking-[0.18em] text-white/60">Admin · Taxas</p>
  <p className="text-sm text-white/85">Overview de fees</p>
</div>
</div>
<div className="flex items-center gap-2 text-[11px]">
<Link
  href="/admin"
  className="rounded-full border border-white/20 px-3 py-1.5 text-white/75 transition-colors hover:bg-white/10">
  Dashboard
</Link>
</div>
</div>
</header>

<section className="mx-auto max-w-5xl space-y-6 px-5 pt-8">
<div className="flex flex-col gap-2">
<h1 className="text-2xl font-semibold tracking-tight md:text-3xl">Taxas configuradas</h1>
<p className="max-w-3xl text-sm text-white/70">
  Valores globais definidos pela ORYA. O organizador só decide quem suporta a taxa (cliente paga ou
  organizador absorve). A taxa da Stripe pode variar por método/pais; aqui controlas o valor base usado nos
  cálculos e previews.
</p>
</div>

<div className="grid gap-4 md:grid-cols-2">
<div className="rounded-2xl border border-white/12 bg-white/5 p-4 shadow-[0_12px_40px_rgba(0,0,0,0.35)]">
<p className="text-[11px] uppercase tracking-[0.18em] text-white/60">ORYA</p>
<h2 className="mt-1 text-lg font-semibold text-white">Taxa da plataforma</h2>
<div className="mt-3 space-y-3 text-sm text-white/80">
  <label className="flex items-center justify-between gap-3">
    <span>Percentual</span>
    <input
      type="number"
      step="0.01"
      min="0"
      value={platformPercent}
      onChange={(e) => setPlatformPercent(e.target.value)}
      className="w-28 rounded-md border border-white/15 bg-black/30 px-2 py-1 text-right text-sm outline-none
focus:border-white/60"
    />
  </label>
  <label className="flex items-center justify-between gap-3">
    <span>Fixo</span>
    <input

```

```

        type="number"
        step="0.01"
        min="0"
        value={platformFixed}
        onChange={(e) => setPlatformFixed(e.target.value)}
        className="w-28 rounded-md border border-white/15 bg-black/30 px-2 py-1 text-right text-sm outline-none
focus:border-white/60"
    />
</label>
<p className="text-[11px] text-white/60">
    Estes valores são usados em todos os cálculos (checkout, previews, dashboards) em tempo real.
</p>
</div>
</div>

<div className="rounded-2xl border border-white/12 bg-white/5 p-4 shadow-[0_12px_40px_rgba(0,0,0,0.35)]">
<p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Stripe</p>
<h2 className="mt-1 text-lg font-semibold text-white">Taxa base ({data && data.ok ? data.stripe.region : "UE"})</h2>
<div className="mt-3 space-y-3 text-sm text-white/80">
    <label className="flex items-center justify-between gap-3">
        <span>Percentual</span>
        <input
            type="number"
            step="0.01"
            min="0"
            value={stripePercent}
            onChange={(e) => setStripePercent(e.target.value)}
            className="w-28 rounded-md border border-white/15 bg-black/30 px-2 py-1 text-right text-sm outline-none
focus:border-white/60"
        />
    </label>
    <label className="flex items-center justify-between gap-3">
        <span>Fixo</span>
        <input
            type="number"
            step="0.01"
            min="0"
            value={stripeFixed}
            onChange={(e) => setStripeFixed(e.target.value)}
            className="w-28 rounded-md border border-white/15 bg-black/30 px-2 py-1 text-right text-sm outline-none
focus:border-white/60"
        />
    </label>
    <p className="text-[11px] text-white/60">
        Valor informativo para cartões UE. O valor real é apurado em cada transação pela Stripe.
    </p>
</div>
</div>
</div>

<div className="rounded-2xl border border-white/12 bg-white/5 p-4 shadow-[0_12px_40px_rgba(0,0,0,0.35)]">
<div className="flex flex-col gap-2 md:flex-row md:items-center md:justify-between">
    <h3 className="text-sm font-semibold text-white">Exemplo rápido (bilhete 10 €)</h3>
    <button
        type="button"
        onClick={handleSave}
        disabled={saving}
        className="inline-flex items-center justify-center rounded-md border border-white/20 bg-white/10 px-3 py-1.5 text-sm font-medium text-white transition hover:bg-white/15 disabled:opacity-60"
    >
        {saving ? "A guardar..." : "Guardar alterações"}
    </button>
</div>
<div className="grid gap-3 pt-3 text-sm text-white/85 md:grid-cols-2">
    <div className="rounded-xl border border-white/10 bg-black/20 p-3">
        <p className="mb-2 text-xs uppercase tracking-[0.15em] text-white/60">Cliente paga (ON_TOP)</p>
        <p className="flex justify-between">
            <span>Preço base</span>
            <span>10,00 €</span>
        </p>
        <p className="flex justify-between">
            <span>Taxa ORYA</span>
            <span>{formatEur(oryafeeCents)}</span>
        </p>
        <p className="flex justify-between">
            <span>Taxa Stripe (estimada)</span>
            <span>{formatEur(stripeFeeOnTopCents)}</span>
        </p>
    </div>
</div>

```

```

        </p>
        <p className="flex justify-between font-semibold">
            <span>Total cliente</span>
            <span>{formatEur(sampleBase + oryaFeeCents)}</span>
        </p>
        <p className="flex justify-between text-white/70">
            <span>Recebe organizador (após Stripe)</span>
            <span>{formatEur(organizerNetOnTop)}</span>
        </p>
        <p className="mt-1 text-[11px] text-white/60">Stripe cobra sobre o total (bilhete + taxa ORYA).</p>
    </div>
    <div className="rounded-xl border border-white/10 bg-black/20 p-3">
        <p className="mb-2 text-xs uppercase tracking-[0.15em] text-white/60">Organizador absorve (INCLUDED)</p>
        <p className="flex justify-between">
            <span>Preço mostrado</span>
            <span>10,00 €</span>
        </p>
        <p className="flex justify-between">
            <span>Taxa ORYA</span>
            <span>{formatEur(oryaFeeCents)}</span>
        </p>
        <p className="flex justify-between">
            <span suppressHydrationWarning>Taxa Stripe (estimada)</span>
            <span suppressHydrationWarning>{formatEur(stripeFeeIncludedCents)}</span>
        </p>
        <p className="flex justify-between font-semibold">
            <span>Recebe organizador (após Stripe)</span>
            <span>{formatEur(organizerNetIncluded)}</span>
        </p>
        <p className="text-[11px] text-white/60">A taxa ORYA é deduzida ao valor. Stripe deduz a sua própria taxa no
payout.</p>
    </div>
</div>
{isLoading && <p className="mt-3 text-[11px] text-white/60">A carregar valores...</p>}
{saveSuccess && <p className="mt-3 text-[11px] text-emerald-300">{saveSuccess}</p>}
{saveError && <p className="mt-3 text-[11px] text-red-300">{saveError}</p>}
{data && !data.ok && <p className="mt-3 text-[11px] text-red-300">Erro a carregar fees.</p>}
</div>
</section>
</main>
);
}
}

```

app/admin/tickets/page.tsx

```

"use client";

import { useEffect, useState } from "react";
import Link from "next/link";
import { AdminLayout } from "@/app/admin/components/AdminLayout";
import { AdminTopActions } from "@/app/admin/components/AdminTopActions";

// Tipos esperados da API de admin tickets (flexíveis para não rebentar se mudares algo no backend)
type AdminTicketEvent = {
    id?: number | null;
    title?: string | null;
    slug?: string | null;
    startsAt?: string | null;
};

type AdminTicketUserProfile = {
    username?: string | null;
    fullName?: string | null;
};

type AdminTicketUser = {
    id?: string | null;
    email?: string | null;
    profile?: AdminTicketUserProfile | null;
};

type AdminTicket = {
    id: string;
    status?: string | null;
}

```

```

event?: AdminTicketEvent | null;
user?: AdminTicketUser | null;
ticketType?: { id?: number | null; name?: string | null } | null;
platformFeeCents?: number | null;
totalPaidCents?: number | null;
pricePaidCents?: number | null;
currency?: string | null;
purchasedAt?: string | null;
stripePaymentIntentId?: string | null;
paymentEventStatus?: string | null;
};

type AdminTicketsApiResponse =
| { ok: true; tickets: AdminTicket[]; page: number; pageSize: number; total: number }
| { ok: false; error?: string };

function formatDate(value?: string | null) {
  if (!value) return "";
  const d = new Date(value);
  if (Number.isNaN(d.getTime())) return "";
  return new Intl.DateTimeFormat("pt-PT", {
    day: "2-digit",
    month: "2-digit",
    year: "2-digit",
    hour: "2-digit",
    minute: "2-digit",
  }).format(d);
}

function formatMoney(cents?: number | null, currency: string = "EUR") {
  if (!cents || Number.isNaN(cents)) return "-";
  const value = cents / 100;
  return new Intl.NumberFormat("pt-PT", {
    style: "currency",
    currency,
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  }).format(value);
}

function statusBadgeClasses(status: string) {
  switch (status) {
    case "ACTIVE":
      return "bg-emerald-50 text-emerald-700 border-emerald-200";
    case "USED":
      return "bg-sky-50 text-sky-700 border-sky-200";
    case "REFUNDED":
      return "bg-amber-50 text-amber-800 border-amber-200";
    case "CANCELLED":
      return "bg-rose-50 text-rose-700 border-rose-200";
    case "RESALE_LISTED":
      return "bg-purple-50 text-purple-700 border-purple-200";
    default:
      return "bg-neutral-50 text-neutral-700 border-neutral-200";
  }
}

function statusLabel(status?: string | null) {
  if (!status) return "Desconhecido";
  switch (status) {
    case "ACTIVE":
      return "Ativo";
    case "USED":
      return "Usado";
    case "REFUNDED":
      return "Reembolsado";
    case "CANCELLED":
      return "Cancelado";
    case "RESALE_LISTED":
      return "Em revenda";
    default:
      return status;
  }
}

export default function AdminTicketsPage() {
  const [tickets, setTickets] = useState<AdminTicket[]>([]);
  const [loading, setLoading] = useState(true);

```

```

const [errorMsg, setErrorMsg] = useState<string | null>(null);
const [query, setQuery] = useState("");
const [statusFilter, setStatusFilter] = useState<string>("ALL");
const [intentFilter, setIntentFilter] = useState("");
const [slugFilter, setSlugFilter] = useState("");
const [userFilter, setUserFilter] = useState("");
const [page, setPage] = useState(1);
const [pageSize] = useState(25);
const [total, setTotal] = useState(0);
const [actionMessage, setActionMessage] = useState<string | null>(null);

async function loadTickets(opts?: {
  q?: string;
  status?: string;
  intent?: string;
  slug?: string;
  user?: string;
  page?: number;
}) {
  try {
    setLoading(true);
    setErrorMsg(null);
    setActionMessage(null);

    const params = new URLSearchParams();
    const q = opts?.q ?? query;
    const status = opts?.status ?? statusFilter;
    const intent = opts?.intent ?? intentFilter;
    const slug = opts?.slug ?? slugFilter;
    const user = opts?.user ?? userFilter;
    const pageParam = opts?.page ?? page;

    if (q && q.trim().length > 0) {
      params.set("q", q.trim());
    }
    if (status && status !== "ALL") {
      params.set("status", status);
    }
    if (intent.trim()) params.set("intent", intent.trim());
    if (slug.trim()) params.set("slug", slug.trim());
    if (user.trim()) params.set("userQuery", user.trim());
    params.set("page", String(pageParam));
    params.set("pageSize", String(pageSize));

    const url = `/api/admin/tickets/list` + (params.toString() ? `?${params.toString()}` : "");

    const res = await fetch(url, {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
      cache: "no-store",
    });

    if (res.status === 401 || res.status === 403) {
      setErrorMsg("Acesso não autorizado. Esta área é apenas para administradores.");
      setTickets([]);
      return;
    }

    if (!res.ok) {
      const text = await res.text();
      console.error("[AdminTickets] Erro ao carregar:", text);
      setErrorMsg("Não foi possível carregar os bilhetes. Tenta novamente em breve.");
      setTickets([]);
      return;
    }

    const json = (await res.json()).catch(() => null) as AdminTicketsApiResponse | null;

    if (!json || !json.ok) {
      console.error("[AdminTickets] Resposta inesperada:", json);
      setErrorMsg(json?.error || "Resposta inesperada da API de tickets.");
      setTickets([]);
      return;
    }

    setTickets(Array.isArray(json.tickets) ? json.tickets : []);
  }
}

```

```

        setPage(json.page || 1);
        setTotal(json.total || 0);
    } catch (err) {
        console.error("[AdminTickets] Erro inesperado:", err);
        setErrorMsg("Ocorreu um erro inesperado ao carregar os bilhetes.");
        setTickets([]);
    } finally {
        setLoading(false);
    }
}

useEffect(() => {
    // carregar logo ao entrar
    void loadTickets({ page: 1 });
    // eslint-disable-next-line react-hooks/exhaustive-deps
}, []);

const hasTickets = tickets.length > 0;

async function handleRefund(intentId?: string | null) {
    if (!intentId) return;
    setLoading(true);
    setActionMessage(null);
    try {
        const res = await fetch("/api/admin/payments/refund", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ paymentIntentId: intentId }),
        });
        const json = await res.json().catch(() => null);
        if (!res.ok || !json?.ok) {
            setActionMessage(json?.error || "Falha ao pedir refund.");
        } else {
            setActionMessage(`Refund solicitado para ${intentId}.`);
            await loadTickets({ page });
        }
    } catch (err) {
        console.error("[AdminTickets] refund error", err);
        setActionMessage("Erro inesperado ao pedir refund.");
    } finally {
        setLoading(false);
    }
}

return (
<AdminLayout title="Bilhetes & histórico" subtitle="Consulta bilhetes, estados, intent e ações de refund.">
    <div className="mx-auto flex max-w-6xl flex-col gap-6">
        <div className="flex items-center justify-between">
            <div>
                <p className="text-xs font-semibold uppercase tracking-[0.18em] text-neutral-400">
                    Admin · Tickets
                </p>
                <h1 className="mt-1 text-2xl font-semibold tracking-tight text-neutral-50">
                    Bilhetes & histórico
                </h1>
                <p className="mt-1 max-w-xl text-sm text-neutral-400">
                    Consulta bilhetes emitidos em toda a plataforma, estados atuais e ligações a eventos e utilizadores.
                </p>
            </div>
            <AdminTopActions showTicketsExport />
        </div>
        {/* Filtros */}
        <section className="rounded-2xl border border-neutral-800 bg-neutral-900/60 p-4 shadow-sm">
            <form
                className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between"
                onSubmit={(e) => {
                    e.preventDefault();
                    void loadTickets({ q: query });
                }}
            >
                <div className="grid flex-1 grid-cols-1 gap-3 md:grid-cols-2 lg:grid-cols-3">
                    <div>
                        <label className="mb-1 block text-xs font-medium text-neutral-400">
                            Pesquisa
                        </label>
                        <input
                            type="text"

```

```

        value={query}
        onChange={(e) => setQuery(e.target.value)}
        placeholder="ID do bilhete, título do evento..."
        className="w-full rounded-xl border border-neutral-700 bg-neutral-950 px-3 py-2 text-sm text-neutral-50
placeholder:text-neutral-500 focus:outline-none focus:ring-2 focus:ring-neutral-400/60"
    >
</div>
<div>
    <label className="mb-1 block text-xs font-medium text-neutral-400">Intent</label>
    <input
        type="text"
        value={intentFilter}
        onChange={(e) => setIntentFilter(e.target.value)}
        placeholder="payment_intent id"
        className="w-full rounded-xl border border-neutral-700 bg-neutral-950 px-3 py-2 text-sm text-neutral-50
placeholder:text-neutral-500 focus:outline-none focus:ring-2 focus:ring-neutral-400/60"
    >
</div>
<div>
    <label className="mb-1 block text-xs font-medium text-neutral-400">Slug do evento</label>
    <input
        type="text"
        value={slugFilter}
        onChange={(e) => setSlugFilter(e.target.value)}
        placeholder="ex.: test-connect"
        className="w-full rounded-xl border border-neutral-700 bg-neutral-950 px-3 py-2 text-sm text-neutral-50
placeholder:text-neutral-500 focus:outline-none focus:ring-2 focus:ring-neutral-400/60"
    >
</div>
<div>
    <label className="mb-1 block text-xs font-medium text-neutral-400">Utilizador</label>
    <input
        type="text"
        value={userFilter}
        onChange={(e) => setUserFilter(e.target.value)}
        placeholder="email ou username"
        className="w-full rounded-xl border border-neutral-700 bg-neutral-950 px-3 py-2 text-sm text-neutral-50
placeholder:text-neutral-500 focus:outline-none focus:ring-2 focus:ring-neutral-400/60"
    >
</div>
<div>
    <label className="mb-1 block text-xs font-medium text-neutral-400">Estado</label>
    <select
        value={statusFilter}
        onChange={(e) => {
            const value = e.target.value;
            setStatusFilter(value);
            void loadTickets({ status: value });
        }}
        className="w-full rounded-xl border border-neutral-700 bg-neutral-950 px-3 py-2 text-sm text-neutral-50
focus:outline-none focus:ring-2 focus:ring-neutral-400/60"
    >
        <option value="ALL">Todos</option>
        <option value="ACTIVE">Ativos</option>
        <option value="USED">Usados</option>
        <option value="REFUNDED">Reembolsados</option>
        <option value="CANCELLED">Cancelados</option>
        <option value="RESALE_LISTED">Em revenda</option>
    </select>
</div>
</div>

<div className="flex gap-2 self-end md:self-auto">
    <button
        type="button"
        onClick={() => {
           setQuery("");
            setStatusFilter("ALL");
            setIntentFilter("");
            setSlugFilter("");
            setUserFilter("");
            setPage(1);
            void loadTickets({ q: "", status: "ALL", intent: "", slug: "", user: "", page: 1 });
        }}
        className="rounded-full border border-neutral-700 px-3 py-2 text-xs text-neutral-300 hover:bg-neutral-800/80"
    >
        Limpar filtros
    </button>

```

```

<button
  type="submit"
  className="rounded-full bg-neutral-50 px-4 py-2 text-xs font-semibold text-neutral-900 hover:bg-white"
>
  Aplicar
</button>
</div>
</form>
</section>

/* Mensagens de erro / loading / vazio */
{loading && (
  <div className="rounded-2xl border border-neutral-800 bg-neutral-900/60 p-4 text-sm text-neutral-300">
    A carregar bilhetes...
  </div>
) {}

{!loading && errorMsg && (
  <div className="rounded-2xl border border-rose-700/60 bg-rose-950/60 p-4 text-sm text-rose-100">
    {errorMsg}
  </div>
) {}

{!loading && actionMessage && !errorMsg && (
  <div className="rounded-2xl border border-emerald-700/60 bg-emerald-900/50 p-4 text-sm text-emerald-100">
    {actionMessage}
  </div>
) {}

{!loading && !errorMsg && !hasTickets && (
  <div className="rounded-2xl border border-dashed border-neutral-800 bg-neutral-900/60 p-6 text-sm text-neutral-300">
    <p className="font-medium">Nenhum bilhete encontrado para estes filtros.</p>
    <p className="mt-1 text-neutral-400">
      Ajusta a pesquisa ou o estado para veres outros resultados.
    </p>
  </div>
) {}

/* Tabela de bilhetes */
{!loading && !errorMsg && hasTickets && (
  <section className="overflow-hidden rounded-2xl border border-neutral-800 bg-neutral-950/80">
    <div className="max-h-[70vh] overflow-auto">
      <table className="min-w-full text-left text-xs text-neutral-200">
        <thead className="sticky top-0 z-10 bg-neutral-950/95">
          <tr className="border-b border-neutral-800 text-[11px] uppercase tracking-[0.16em] text-neutral-500">
            <th className="px-3 py-3">Bilhete</th>
            <th className="px-3 py-3">Evento</th>
            <th className="px-3 py-3">Tipo</th>
            <th className="px-3 py-3">Utilizador</th>
            <th className="px-3 py-3">Preço / Fees</th>
            <th className="px-3 py-3">Estado</th>
            <th className="px-3 py-3">Comprado em</th>
            <th className="px-3 py-3">Intent</th>
            <th className="px-3 py-3 text-right">Ações</th>
          </tr>
        </thead>
        <tbody>
          {tickets.map((t) => {
            const event = t.event;
            const user = t.user;
            const profile = user?.profile;
            const userLabel = profile?.username
              ? `#${profile.username}`
              : profile?.fullName
              ? profile.fullName
              : user?.email ?? "-";
            return (
              <tr key={t.id} className="border-b border-neutral-900/80 last:border-0 hover:bg-neutral-900/80">
                <td className="px-3 py-3 align-top font-mono text-[11px] text-neutral-300">{t.id}</td>
                <td className="px-3 py-3 align-top">
                  {event?.title ? (
                    <div className="flex flex-col gap-0.5">
                      {event.slug ? (
                        <Link href={`/eventos/${event.slug}`} className="text-xs font-medium text-neutral-50
hover:underline">
                          {event.title}
                        </Link>

```

```

        ) : (
            <span className="text-xs font-medium text-neutral-50">{event.title}</span>
        )}
        {event.startsAt && (
            <span className="text-[11px] text-neutral-400">
                {formatDate(event.startsAt)}
            </span>
        )}
    </div>
) : (
    <span className="text-xs text-neutral-500">--</span>
)
</td>
<td className="px-3 py-3 align-top text-xs text-neutral-100">
    {t.ticketType?.name ?? "-"}
</td>
<td className="px-3 py-3 align-top text-[11px] text-neutral-100">
    <div className="flex flex-col gap-0.5 text-xs">
        <span className="font-medium text-neutral-50">{userLabel}</span>
        {user?.email && (
            <span className="text-[11px] text-neutral-500">{user.email}</span>
        )}
    </div>
</td>
<td className="px-3 py-3 align-top text-[11px] text-neutral-100">
    <div className="flex flex-col gap-1">
        <span className="text-xs text-neutral-100">
            Pago: {formatMoney(t.pricePaidCents ?? null, t.currency || "EUR")}
        </span>
        <span className="text-[11px] text-neutral-400">
            Fee: {formatMoney(t.platformFeeCents ?? null, t.currency || "EUR")}
        </span>
        <span className="text-[11px] text-neutral-400">
            Total: {formatMoney(t.totalPaidCents ?? null, t.currency || "EUR")}
        </span>
    </div>
</td>
<td className="px-3 py-3 align-top">
    <span className={` inline-flex rounded-full border px-2 py-0.5 text-[11px] font-medium
${statusBadgeClasses(t.status || "")}`}>
        {statusLabel(t.status)}
    </span>
</td>
<td className="px-3 py-3 align-top text-[11px] text-neutral-300">
    {formatDate(t.purchasedAt)}
</td>
<td className="px-3 py-3 align-top font-mono text-[10px] text-neutral-500 max-w-[160px] truncate">
    <div className="flex flex-col gap-1">
        <span className="truncate">{t.stripePaymentIntentId || "-"}</span>
        {t.stripePaymentIntentId && (
            <div className="flex flex-wrap gap-1">
                <button
                    type="button"
                    onClick={() => navigator.clipboard?.writeText(t.stripePaymentIntentId || "")}
                    className="rounded-full border border-neutral-700 px-2 py-0.5 text-[10px] text-neutral-300
hover:bg-neutral-800/70"
                >
                    Copiar
                </button>
                <Link
                    href={`/admin/payments?q=${encodeURIComponent(t.stripePaymentIntentId)}`}
                    className="rounded-full border border-neutral-700 px-2 py-0.5 text-[10px] text-neutral-300
hover:bg-neutral-800/70"
                >
                    Ver intent
                </Link>
            </div>
        )}
    </div>
</td>
<td className="px-3 py-3 align-top text-right">
    <div className="flex flex-wrap justify-end gap-2">
        <Link
            href={`/admin/payments?q=${encodeURIComponent(t.stripePaymentIntentId || "")}`}
            className="rounded-full border border-neutral-700 px-2.5 py-1 text-[11px] text-neutral-200
hover:bg-neutral-800/70"
        >
            Pagamento
        </Link>
    </div>
</td>

```

```

        </Link>
        {t.stripePaymentIntentId && t.paymentEventStatus === "REFUNDED" && (
          <button
            type="button"
            onClick={() => handleRefund(t.stripePaymentIntentId)}
            className="rounded-full border border-neutral-700 px-2.5 py-1 text-neutral-100
hover:bg-neutral-800/70"
          >
            Refund
          </button>
        )}
      </div>
    </td>
  </tr>
);
);
});
</tbody>
</table>
</div>
</section>
)
</div>
/* Paginação simples */
<div className="mx-auto flex max-w-6xl items-center justify-between px-4 py-4 text-xs text-neutral-300">
  <span>
    Página {page} · {total} registos
  </span>
  <div className="flex gap-2">
    <button
      type="button"
      disabled={page === 1}
      onClick={() => {
        const next = Math.max(1, page - 1);
        setPage(next);
        void loadTickets({ page: next });
      }}
      className="rounded-full border border-neutral-700 px-3 py-1.5 disabled:opacity-40 hover:bg-neutral-800/70"
    >
      Anterior
    </button>
    <button
      type="button"
      disabled={page * pageSize === total}
      onClick={() => {
        const next = page + 1;
        setPage(next);
        void loadTickets({ page: next });
      }}
      className="rounded-full border border-neutral-700 px-3 py-1.5 disabled:opacity-40 hover:bg-neutral-800/70"
    >
      Seguinte
    </button>
  </div>
</div>
</AdminLayout>
);
}

```

app/admin/utilizadores/page.tsx

```

// app/admin/utilizadores/page.tsx

export const dynamic = "force-dynamic";

import { redirect } from "next/navigation";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import type { Prisma } from "@prisma/client";
import { UsersTableClient } from "./UsersTableClient";

type AdminUsersPageProps = {
  searchParams?: {
    q?: string;
  };
};

```

```
export default async function AdminUsersPage({ searchParams }: AdminUsersPageProps) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) {
    redirect("/");
  }

  // Confirmar se o user é admin
  const me = await prisma.profile.findUnique({
    where: { id: user.id },
    select: {
      id: true,
      username: true,
      roles: true,
    },
  });
}

if (!me || !Array.isArray(me.roles) || !me.roles.includes("admin")) {
  redirect("/");
}

const resolvedSearchParams = await Promise.resolve(searchParams);
const search = (resolvedSearchParams?.q || "").trim();

const where: Prisma.ProfileWhereInput = {};
if (search.length > 0) {
  where.OR = [
    { username: { contains: search, mode: "insensitive" } },
    { fullName: { contains: search, mode: "insensitive" } },
    { city: { contains: search, mode: "insensitive" } },
  ];
}

const [totalUsers, totalOrganizers, totalAdmins, users] = await Promise.all([
  prisma.profile.count(),
  prisma.profile.count({
    where: {
      roles: {
        has: "organizer",
      },
    },
  }),
  prisma.profile.count({
    where: {
      roles: {
        has: "admin",
      },
    },
  }),
  prisma.profile.findMany({
    where,
    orderBy: { createdAt: "desc" },
    take: 50,
    select: {
      id: true,
      username: true,
      fullName: true,
      city: true,
      roles: true,
      createdAt: true,
      isDeleted: true,
      deletedAt: true,
    },
  }),
]);
]

return (
  <main className="orya-body-bg min-h-screen text-white pb-16">
    {/* Top bar */}
    <header className="border-b border-white/10 bg-black/50 backdrop-blur-xl">
      <div className="max-w-6xl mx-auto px-5 py-4 flex items-center justify-between gap-3">
        <div className="flex items-center gap-3">
          <span className="inline-flex h-8 w-8 items-center rounded-xl bg-gradient-to-tr from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-[10px] font-extrabold tracking-[0.16em]">
            ADM
          </span>
        </div>
      </div>
    </header>
    <div className="flex flex-col gap-10 p-5">
      <div>
        <h2>Administradores</h2>
        <p>Total de organizadores:</p>
        <div>{totalOrganizers}</div>
      </div>
      <div>
        <h2>Administradores</h2>
        <p>Total de administradores:</p>
        <div>{totalAdmins}</div>
      </div>
      <div>
        <h2>Todos os usuários</h2>
        <p>Total de usuários:</p>
        <div>{totalUsers}</div>
      </div>
    </div>
  </main>
)
```

```

        </span>
    <div>
        <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">
            Painel · Admin
        </p>
        <p className="text-sm text-white/85">Gestão de utilizadores da plataforma.</p>
    </div>
</div>

<div className="hidden sm:flex items-center gap-2 text-[11px] text-white/70">
    <span className="px-2 py-1 rounded-full border border-white/15">
        {me.username ? `@$ {me.username}` : "Admin"}
    </span>
</div>
</div>
</div>
</header>

<section className="max-w-6xl mx-auto px-5 pt-6 space-y-6">
    /* Header + search */
    <div className="flex flex-col md:flex-row md:items-end md:justify-between gap-4">
        <div className="space-y-1">
            <h1 className="text-2xl md:text-3xl font-semibold tracking-tight">
                Utilizadores
            </h1>
            <p className="text-sm text-white/70 max-w-xl">
                Pesquisa e visão rápida dos perfis registados na ORYA. Esta área é apenas
                para debugging e gestão interna.
            </p>
        </div>
        <div className="w-full md:w-72" action="/admin/utilizadores" method="GET">
            <label className="block text-[11px] text-white/65 mb-1">
                Pesquisar utilizador (username, nome, cidade)
            </label>
            <div className="flex items-center gap-2">
                <input
                    type="text"
                    name="q"
                    defaultValue={search}
                    placeholder="ex: joao, porto...">
                <span className="flex-1 rounded-xl border border-white/20 bg-black/40 px-3 py-2 text-sm text-white placeholder:text-white/40 focus:outline-none focus:ring-2 focus:ring-[#6BFFFF]/80">
                    /
                <button
                    type="submit"
                    className="px-3 py-2 rounded-xl bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-[11px] font-semibold text-black shadow-[0_0_14px_rgba(107,255,255,0.6)] hover:scale-[1.02] active:scale-95 transition-transform">
                    Buscar
                </button>
            </div>
        </div>
    </div>
</section>

/* KPI cards */
<div className="grid grid-cols-1 sm:grid-cols-3 gap-4">
    <div className="rounded-2xl border border-white/12 bg-black/60 px-4 py-3">
        <p className="text-[11px] text-white/55 uppercase tracking-[0.14em]">
            Utilizadores totais
        </p>
        <p className="mt-1 text-2xl font-semibold">{totalUsers}</p>
        <p className="mt-1 text-[11px] text-white/55">
            Perfis com conta criada na ORYA.
        </p>
    </div>
    <div className="rounded-2xl border border-white/12 bg-black/60 px-4 py-3">
        <p className="text-[11px] text-white/55 uppercase tracking-[0.14em]">
            Organizadores
        </p>
        <p className="mt-1 text-2xl font-semibold">{totalOrganizers}</p>
        <p className="mt-1 text-[11px] text-white/55">
            Perfis com role de organizador ativa.
        </p>
    </div>
    <div className="rounded-2xl border border-white/12 bg-black/60 px-4 py-3">
        <p className="text-[11px] text-white/55 uppercase tracking-[0.14em]">
            Admins
        </p>
    </div>
</div>

```

```

<p className="mt-1 text-2xl font-semibold">{totalAdmins}</p>
<p className="mt-1 text-[11px] text-white/55">
  Contas com acesso ao painel de admin.
</p>
</div>
</div>

/* Lista de utilizadores */
<div className="mt-4 rounded-2xl border border-white/12 bg-black/70 overflow-hidden">
  <div className="flex items-center justify-between px-4 py-3 border-b border-white/10">
    <p className="text-[11px] text-white/65 uppercase tracking-[0.16em]">
      Resultados
    </p>
    <p className="text-[11px] text-white/50">
      A mostrar até 50 perfis {search ? "filtrados" : "mais recentes"}.
    </p>
  </div>
  <div>
    {users.length === 0 ? (
      <div className="px-4 py-6 text-sm text-white/65">
        Nenhum utilizador encontrado para este filtro.
      </div>
    ) : (
      <UsersTableClient
        users={users.map((u) => ({
          ...u,
          createdAt: u.createdAt.toISOString(),
          deletedAt: u.deletedAt ? u.deletedAt.toISOString() : null,
        }))}
      />
    )}
  </div>
</section>
</main>
);
}

```

app/admin/utilizadores/UsersTableClient.tsx

```

"use client";

import React from "react";

type UserItem = {
  id: string;
  username: string | null;
  fullName: string | null;
  city: string | null;
  roles: string[] | null;
  createdAt: string;
  isDeleted: boolean;
  deletedAt: string | null;
};

type Props = {
  users: UserItem[];
};

export function UsersTableClient({ users }: Props) {
  const [loadingId, setLoadingId] = React.useState<string | null>(null);
  const [loadingAction, setLoadingAction] = React.useState<string | null>(null);

  async function runAction(userId: string, action: "ban" | "unban" | "hard_delete") {
    setLoadingId(userId);
    setLoadingAction(action);
    try {
      const res = await fetch("/api/admin/utilizadores/manage", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ userId, action }),
      });
      const data = await res.json().catch(() => ({}));
      if (!res.ok) {
        alert(data?.error || "Erro ao executar ação.");
      } else {
        alert(data?.message || "Ação concluída.");
      }
    } catch (err) {
      console.error("Error in runAction:", err);
    }
  }
}

```

```

        window.location.reload();
    }
} catch (err) {
    console.error("[UsersTableClient] action error:", err);
    alert("Erro inesperado.");
} finally {
    setLoadingId(null);
    setLoadingAction(null);
}
}

return (
<div className="overflow-x-auto">
<table className="min-w-full text-left text-[11px]">
<thead className="bg-white/5 border-b border-white/10">
<tr className="text-white/70">
<th className="px-4 py-2 font-semibold">Username</th>
<th className="px-4 py-2 font-semibold">Nome</th>
<th className="px-4 py-2 font-semibold">Cidade</th>
<th className="px-4 py-2 font-semibold">Roles</th>
<th className="px-4 py-2 font-semibold">Criado</th>
<th className="px-4 py-2 font-semibold">Estado</th>
<th className="px-4 py-2 font-semibold">Ações</th>
</tr>
</thead>
<tbody>
{users.map((u) => (
<tr
    key={u.id}
    className="border-b border-white/5 hover:bg-white/[0.03] transition-colors"
>
<td className="px-4 py-3 whitespace nowrap text-white">
    {u.username ? `@$ {u.username}` : "-"}
</td>
<td className="px-4 py-3 whitespace nowrap text-white/90">
    {u.fullName || "-"}
</td>
<td className="px-4 py-3 whitespace nowrap text-white/80">
    {u.city || "-"}
</td>
<td className="px-4 py-3 whitespace nowrap">
    <div className="flex flex-wrap gap-2">
        {(u.roles || []).map((r) => (
<span
            key={r}
            className="inline-flex items-center rounded-full bg-white/10 px-2 py-1 text-[10px] uppercase tracking-[0.12em] text-white/75"
>
            {r}
</span>
        )))
    {(!u.roles || u.roles.length === 0) && (
        <span className="text-white/50 text-[10px]">sem roles</span>
    )}
    </div>
</td>
<td className="px-4 py-3 whitespace nowrap text-white/70">
    {new Date(u.createdAt).toLocaleDateString("pt-PT", {
        day: "2-digit",
        month: "short",
        year: "numeric",
    })}
</td>
<td className="px-4 py-3 whitespace nowrap">
    {u.isDeleted ? (
        <span className="inline-flex items-center rounded-full bg-red-500/15 px-2 py-1 text-[10px] font-semibold text-red-200">
            Banido
        </span>
    ) : (
        <span className="inline-flex items-center rounded-full bg-emerald-500/15 px-2 py-1 text-[10px] font-semibold text-emerald-200">
            Ativo
        </span>
    )}
</td>
<td className="px-4 py-3 whitespace nowrap">
    <div className="flex flex-wrap gap-2">

```

```

{u.isDeleted ? (
    <button
        onClick={() => runAction(u.id, "unban")}
        disabled={loadingId === u.id}
        className="rounded-full border border-emerald-400/40 px-3 py-1 text-emerald-100 hover:bg-emerald-500/10 disabled:opacity-40"
    >
        {loadingId === u.id && loadingAction === "unban"
            ? "A reativar..."
            : "Reativar"}
    </button>
) : (
    <button
        onClick={() => runAction(u.id, "ban")}
        disabled={loadingId === u.id}
        className="rounded-full border border-amber-400/40 px-3 py-1 text-amber-100 hover:bg-amber-500/10 disabled:opacity-40"
    >
        {loadingId === u.id && loadingAction === "ban"
            ? "A banir..."
            : "Banir"}
    </button>
))
<button
    onClick={() => {
        const ok = confirm(
            "Eliminar em definitivo este utilizador? Esta ação remove do Auth e do perfil.",
        );
        if (ok) runAction(u.id, "hard_delete");
    }}
    disabled={loadingId === u.id}
    className="rounded-full border border-red-500/50 px-3 py-1 text-red-200 hover:bg-red-500/10 disabled:opacity-40"
>
    {loadingId === u.id && loadingAction === "hard_delete"
        ? "A apagar..."
        : "Hard delete"}
</button>
</div>
</td>
</tr>
)}
</tbody>
</table>
</div>
);
}

```

app/api/admin/eventos/list/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { Prisma, EventStatus, EventType } from "@prisma/client";

// Fase 6.13 – Listar eventos (admin)
// GET /api/admin/eventos/list
// Permite ao admin pesquisar eventos globalmente por título/slug/organizador

export async function GET(req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error: authError,
        } = await supabase.auth.getUser();

        if (authError) {
            console.error("[admin eventos list] erro auth:", authError);
        }

        if (!user) {
            return NextResponse.json(
                { ok: false, error: "UNAUTHENTICATED" },
                { status: 401 },
            );
        }
    }
}

```

```

}

// Confirmar se é admin via profile.roles
const profile = await prisma.profile.findUnique({
  where: { id: user.id },
  select: { id: true, roles: true },
});

const roles = profile?.roles ?? [];
const isAdmin = Array.isArray(roles) && roles.includes("admin");

if (!isAdmin) {
  return NextResponse.json(
    { ok: false, error: "FORBIDDEN" },
    { status: 403 },
  );
}

const { searchParams } = new URL(req.url);

const search = searchParams.get("search")?.trim() || "";
const statusFilter = searchParams.get("status")?.trim() || "";
const typeFilter = searchParams.get("type")?.trim() || "";

const takeRaw = searchParams.get("take");
const take = Math.min(Math.max(Number(takeRaw) || 50, 1), 200); // entre 1 e 200

const where: Prisma.EventWhereInput = {};

if (search) {
  where.OR = [
    { title: { contains: search, mode: "insensitive" } },
    { slug: { contains: search, mode: "insensitive" } },
    {
      organizer: {
        displayName: { contains: search, mode: "insensitive" },
      },
    },
  ];
}

if (statusFilter) {
  where.status = statusFilter as EventStatus; // mapeado para o enum EventStatus do Prisma
}

if (typeFilter) {
  where.type = typeFilter as EventType; // mapeado para o enum EventType do Prisma
}

const events = await prisma.event.findMany({
  where,
  include: {
    organizer: {
      select: {
        id: true,
        displayName: true,
      },
    },
    orderBy: {
      createdAt: "desc",
    },
    take,
  },
});

const items = events.map((evt) => ({
  id: evt.id,
  slug: evt.slug,
  title: evt.title,
  status: evt.status,
  type: evt.type,
  startsAt: evt.startsAt,
  endsAt: evt.endsAt,
  createdAt: evt.createdAt,
  organizer: evt.organizer
  ? {
    id: evt.organizer.id,
    displayName: evt.organizer.displayName,
  }
}))
```

```

        }
        : null,
      )));

      return NextResponse.json({ ok: true, items });
    } catch (error) {
      console.error("[admin eventos list] erro inesperado:", error);
      return NextResponse.json(
        { ok: false, error: "INTERNAL_ERROR" },
        { status: 500 },
      );
    }
  }
}

```

app/api/admin/eventos/update-status/route.ts

```

// app/api/admin/eventos/update-status/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import type { EventStatus } from "@prisma/client";
import { enqueueOperation } from "@/lib/operations/enqueue";

/** 
 * 6.14 – Update de estado de evento (admin)
 *
 * Body esperado (POST JSON):
 * {
 *   "eventId": number | string,
 *   "slug": string,
 *   "status": string // ex: "PUBLISHED", "CANCELLED", "BLOCKED"...
 * }
 *
 * Regras:
 * - Só utilizadores com role "admin" podem chamar.
 * - É possível identificar evento por eventId OU por slug.
 */
export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error: authError,
    } = await supabase.auth.getUser();

    if (authError) {
      console.error("[admin/eventos/update-status] authError:", authError);
    }

    if (!user) {
      return NextResponse.json(
        { ok: false, error: "UNAUTHENTICATED" },
        { status: 401 }
      );
    }

    // Confirmar se é admin
    const profile = await prisma.profile.findUnique({
      where: { id: user.id },
      select: { roles: true },
    });

    const roles = profile?.roles ?? [];
    const isAdmin = Array.isArray(roles) && roles.includes("admin");

    if (!isAdmin) {
      return NextResponse.json(
        { ok: false, error: "FORBIDDEN_NOT_ADMIN" },
        { status: 403 }
      );
    }

    const body = (await req.json().catch(() => null)) as
      | {
          eventId?: number | string;
        }

```

```

        slug?: string;
        status?: string;
    }
    | null;
}

if (!body || typeof body !== "object") {
    return NextResponse.json(
        { ok: false, error: "INVALID_BODY" },
        { status: 400 }
    );
}

const { eventId, slug, status } = body;

if (!status || typeof status !== "string") {
    return NextResponse.json(
        { ok: false, error: "MISSING_STATUS" },
        { status: 400 }
    );
}

// Construir o "where" dinamicamente: por id OU por slug
let whereClause:
    | {
        id: number;
    }
    | {
        slug: string;
    }
    | null = null;

if (typeof eventId === "number") {
    whereClause = { id: eventId };
} else if (typeof eventId === "string") {
    const parsed = Number(eventId);
    if (!Number.isNaN(parsed)) {
        whereClause = { id: parsed };
    }
} else if (typeof slug === "string" && slug.trim() !== "") {
    whereClause = { slug: slug.trim() };
}

if (!whereClause) {
    return NextResponse.json(
        { ok: false, error: "MISSING_EVENT_IDENTIFIER" },
        { status: 400 }
    );
}

// Atualizar evento
try {
    const updated = await prisma.event.update({
        where: whereClause,
        data: {
            // Cast para EventStatus para corresponder ao enum do Prisma
            status: status as EventStatus,
        },
        select: {
            id: true,
            slug: true,
            title: true,
            status: true,
            type: true,
            organizerId: true,
            updatedAt: true,
        },
    });
}

if (updated.status === "CANCELLED" || updated.status === "DELETED" || updated.status === "DATE_CHANGED") {
    // Disparar refunds base-only para todas as compras deste evento (idempotente por dedupeKey)
    const summaries = await prisma.saleSummary.findMany({
        where: { eventId: updated.id, status: "PAID" },
        select: { purchaseId: true, paymentIntentId: true },
    });
    await Promise.all(
        summaries.map((s) =>
            enqueueOperation({
                operationType: "PROCESS_REFUND_SINGLE",

```

```

        dedupeKey: `${updated.id}:${s.purchaseId ?? s.paymentIntentId ?? "unknown"}`,
        correlations: { eventId: updated.id, purchaseId: s.purchaseId ?? s.paymentIntentId ?? null, paymentIntentId: s.paymentIntentId ?? null },
        payload: {
          eventId: updated.id,
          purchaseId: s.purchaseId ?? s.paymentIntentId ?? null,
          paymentIntentId: s.paymentIntentId ?? null,
          reason: updated.status,
          refundedBy: user.id,
        },
      ),
    ),
  );
}

return NextResponse.json(
{
  ok: true,
  event: updated,
},
{ status: 200 }
);
} catch (err) {
// P2025 = record not found
if (
err &&
typeof err === "object" &&
"code" in err &&
(err as { code: string }).code === "P2025"
) {
return NextResponse.json(
{ ok: false, error: "EVENT_NOT_FOUND" },
{ status: 404 }
);
}
console.error(
"[admin/eventos/update-status] Error updating event status:",
err
);
return NextResponse.json(
{ ok: false, error: "INTERNAL_ERROR" },
{ status: 500 }
);
}
} catch (err) {
console.error("[admin/eventos/update-status] Unexpected error:", err);
return NextResponse.json(
{ ok: false, error: "INTERNAL_ERROR" },
{ status: 500 }
);
}
}
}
}

```

app/api/admin/fees/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@lib/supabaseServer";
import { prisma } from "@lib/prisma";
import { getPlatformAndStripeFees, setPlatformFees, setStripeBaseFees } from "@lib/platformSettings";

function rolesContainsAdmin(roles: unknown) {
  if (Array.isArray(roles)) return roles.includes("admin");
  if (typeof roles === "string") {
    try {
      const parsed = JSON.parse(roles);
      if (Array.isArray(parsed) && parsed.includes("admin")) return true;
    } catch {
      return roles.split(",").map((r) => r.trim()).includes("admin");
    }
  }
  return false;
}

async function isAdminUser(userId: string) {
  const supabase = await createSupabaseServer();

```

```

const profileRes = await supabase.from("profiles").select("roles").eq("id", userId).maybeSingle();
const rolesFromSupabase = profileRes.data?.roles ?? [];

if (rolesContainsAdmin(rolesFromSupabase)) {
  return true;
}

// fallback via prisma (auth schema) in caso de erro/mismatch
const profileDb = await prisma.profile.findUnique({
  where: { id: userId },
  select: { roles: true },
});
const rolesDb = profileDb?.roles ?? [];
return rolesContainsAdmin(rolesDb);
}

export async function GET(_req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
    } = await supabase.auth.getUser();

    if (!user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    if (!(await isAdminUser(user.id))) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    const { orya, stripe } = await getPlatformAndStripeFees();

    return NextResponse.json(
      {
        ok: true,
        orya,
        stripe,
      },
      { status: 200 },
    );
  } catch (err) {
    console.error("[admin/fees] unexpected error", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
  }
}

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
    } = await supabase.auth.getUser();

    if (!user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    if (!(await isAdminUser(user.id))) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    const body = await req.json();

    const clamp = (val: number, min: number, max: number) => Math.min(Math.max(val, min), max);
    const platformFeeBpsRaw = Number(body?.platformFeeBps);
    const platformFeeFixedCentsRaw = Number(body?.platformFeeFixedCents);
    const stripeFeeBpsEuRaw = Number(body?.stripeFeeBpsEu);
    const stripeFeeFixedCentsEuRaw = Number(body?.stripeFeeFixedCentsEu);

    const updatesErrors: string[] = [];
    if (body?.platformFeeBps !== undefined && !Number.isFinite(platformFeeBpsRaw)) {
      updatesErrors.push("platformFeeBps inválido");
    }
    if (body?.platformFeeFixedCents !== undefined && !Number.isFinite(platformFeeFixedCentsRaw)) {
      updatesErrors.push("platformFeeFixedCents inválido");
    }
    if (body?.stripeFeeBpsEu !== undefined && !Number.isFinite(stripeFeeBpsEuRaw)) {
      updatesErrors.push("stripeFeeBpsEu inválido");
    }
  }
}

```

```

    updatesErrors.push("stripeFeeBpsEu inválido");
}
if (body?.stripeFeeFixedCentsEu !== undefined && !Number.isFinite(stripeFeeFixedCentsEuRaw)) {
    updatesErrors.push("stripeFeeFixedCentsEu inválido");
}

if (updatesErrors.length > 0) {
    return NextResponse.json({ ok: false, error: updatesErrors.join(", ") }, { status: 400 });
}

await Promise.all([
    setPlatformFees({
        feeBps: Number.isFinite(platformFeeBpsRaw)
            ? clamp(Math.round(platformFeeBpsRaw), 0, 5000)
            : undefined,
        feeFixedCents: Number.isFinite(platformFeeFixedCentsRaw)
            ? clamp(Math.round(platformFeeFixedCentsRaw), 0, 5000)
            : undefined,
    }),
    setStripeBaseFees({
        feeBps: Number.isFinite(stripeFeeBpsEuRaw)
            ? clamp(Math.round(stripeFeeBpsEuRaw), 0, 5000)
            : undefined,
        feeFixedCents: Number.isFinite(stripeFeeFixedCentsEuRaw)
            ? clamp(Math.round(stripeFeeFixedCentsEuRaw), 0, 5000)
            : undefined,
    }),
]);
const { orya, stripe } = await getPlatformAndStripeFees();

return NextResponse.json({ ok: true, orya, stripe }, { status: 200 });
} catch (err) {
    console.error("[admin/fees] unexpected error on POST", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/admin/organizadores/list/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { OrganizerStatus, Prisma } from "@prisma/client";

/**
 * 6.11 – Listar organizadores (admin)
 *
 * GET /api/admin/organizadores/list
 *
 * Query params opcionais:
 * - search: string (filtra por displayName com contains, case-insensitive)
 * - status: 'PENDING' | 'ACTIVE' | 'SUSPENDED' (ou outros valores do enum OrganizerStatus)
 * - page: número da página (1-based, default 1)
 * - pageSize: tamanho da página (default 20, máx 100)
 *
 * Apenas utilizadores com role "admin" podem aceder.
 */
function parsePositiveInt(value: string | null, defaultValue: number): number {
    if (!value) return defaultValue;
    const n = Number(value);
    if (!Number.isFinite(n) || n <= 0) return defaultValue;
    return Math.floor(n);
}

type AdminAuthError = "UNAUTHENTICATED" | "FORBIDDEN" | null;

async function getAdminUserId(): Promise<{
    userId: string | null;
    error: AdminAuthError;
}> {
    const supabase = await createSupabaseServer();
    const {
        data: { user },
        error,
    }

```

```

} = await supabase.auth.getUser();

if (error || !user) {
  return { userId: null, error: "UNAUTHENTICATED" };
}

// Verificar se este user tem role "admin" no profile
const profile = await prisma.profile.findUnique({
  where: { id: user.id },
  select: { roles: true },
});

const rolesArray = Array.isArray(profile?.roles) ? profile?.roles : [];
if (!rolesArray.includes("admin")) {
  return { userId: null, error: "FORBIDDEN" };
}

return { userId: user.id, error: null };
}

export async function GET(req: NextRequest) {
  try {
    // 1) Guard de admin
    const { userId, error } = await getAdminUserId();

    if (error === "UNAUTHENTICATED") {
      return NextResponse.json(
        { ok: false, error: "UNAUTHENTICATED" },
        { status: 401 }
      );
    }

    if (error === "FORBIDDEN") {
      return NextResponse.json(
        { ok: false, error: "FORBIDDEN" },
        { status: 403 }
      );
    }
  }

  // userId neste momento não é usado na query, mas pode ser útil no futuro
  void userId;

  // 2) Ler query params
  const url = new URL(req.url);
  const searchParam = url.searchParams.get("search");
  const statusParam = url.searchParams.get("status");
  const pageParam = url.searchParams.get("page");
  const pageSizeParam = url.searchParams.get("pageSize");

  const page = parsePositiveInt(pageParam, 1);
  const pageSizeRaw = parsePositiveInt(pageSizeParam, 20);
  const pageSize = Math.min(pageSizeRaw, 100);

  const skip = (page - 1) * pageSize;

  // 3) Construir filtros
  let statusFilter: OrganizerStatus | undefined;

  if (
    statusParam &&
    (Object.values(OrganizerStatus) as string[]).includes(statusParam)
  ) {
    statusFilter = statusParam as OrganizerStatus;
  }

  const where: Prisma.OrganizerWhereInput = {};

  if (statusFilter) {
    where.status = statusFilter;
  }

  if (searchParam && searchParam.trim() !== "") {
    const search = searchParam.trim();
    where.displayName = {
      contains: search,
      mode: "insensitive",
    };
  }
}

```

```

}

// 4) Query com paginação
const [items, total] = await Promise.all([
  prisma.organizer.findMany({
    where,
    orderBy: { createdAt: "desc" },
    skip,
    take: pageSize,
  }),
  prisma.organizer.count({ where }),
]);

const totalPages = Math.max(1, Math.ceil(total / pageSize));

return NextResponse.json(
  {
    ok: true,
    items,
    pagination: {
      page,
      pageSize,
      total,
      totalPages,
    },
    { status: 200 }
  };
} catch (err) {
  console.error(`[api/admin/organizadores/list] Erro inesperado:`, err);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 }
  );
}
}
}

```

app/api/admin/organizadores/update-status/route.ts

```

// app/api/admin/organizadores/update-status/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

// Tipos de estados permitidos para organizadores (ajusta se o enum tiver outros valores)
const ALLOWED_STATUSES = ["PENDING", "ACTIVE", "SUSPENDED"] as const;

type AllowedStatus = (typeof ALLOWED_STATUSES)[number];

type UpdateOrganizerStatusBody = {
  organizerId?: number | string;
  newStatus?: string;
};

async function getAdminUserId(): Promise<string | null> {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) return null;

  const profile = await prisma.profile.findUnique({ where: { id: user.id } });

  if (!profile) return null;

  const roles = (profile.roles as string[] | null) ?? [];
  const isAdmin = roles.includes("admin");

  if (!isAdmin) return null;

  return user.id;
}

```

```

export async function POST(req: NextRequest) {
  try {
    const adminUserId = await getAdminUserId();

    if (!adminUserId) {
      return NextResponse.json(
        { ok: false, error: "FORBIDDEN" },
        { status: 403 },
      );
    }
  }

  const body = (await req.json().catch(() => null)) as
    | UpdateOrganizerStatusBody
    | null;

  if (!body || typeof body !== "object") {
    return NextResponse.json(
      { ok: false, error: "INVALID_BODY" },
      { status: 400 },
    );
  }

  const { organizerId, newStatus } = body;

  if (
    organizerId === undefined ||
    organizerId === null ||
    newStatus === undefined ||
    typeof newStatus !== "string"
  ) {
    return NextResponse.json(
      { ok: false, error: "MISSING_FIELDS" },
      { status: 400 },
    );
  }

  const normalizedStatus = newStatus.trim().toUpperCase() as AllowedStatus;

  if (!ALLOWED_STATUSES.includes(normalizedStatus)) {
    return NextResponse.json(
      { ok: false, error: "INVALID_STATUS" },
      { status: 400 },
    );
  }

  const organizerIdNumber =
    typeof organizerId === "string" ? Number(organizerId) : organizerId;

  if (
    typeof organizerIdNumber !== "number" ||
    Number.isNaN(organizerIdNumber) ||
    organizerIdNumber <= 0
  ) {
    return NextResponse.json(
      { ok: false, error: "INVALID_ORGANIZER_ID" },
      { status: 400 },
    );
  }

  const organizer = await prisma.organizer.findUnique({
    where: { id: organizerIdNumber },
    select: {
      id: true,
      status: true,
      displayName: true,
    },
  });

  if (!organizer) {
    return NextResponse.json(
      { ok: false, error: "ORGANIZER_NOT_FOUND" },
      { status: 404 },
    );
  }

  // Se o estado já está igual, devolvemos ok mas sem fazer update
  if (organizer.status === normalizedStatus) {
    return NextResponse.json(

```

```

        {
          ok: true,
          organizer: {
            id: organizer.id,
            status: organizer.status,
            displayName: organizer.displayName,
            changed: false,
          },
        },
        { status: 200 },
      );
    }
  }

  const updated = await prisma.organizer.update({
    where: { id: organizerIdNumber },
    data: {
      status: normalizedStatus,
    },
    select: {
      id: true,
      status: true,
      displayName: true,
      userId: true,
    },
  });

  // Se aprovado (ACTIVE), adicionar role organizer ao profile
  if (normalizedStatus === "ACTIVE" && updated.userId) {
    const profile = await prisma.profile.findUnique({
      where: { id: updated.userId },
      select: { roles: true },
    });
    const roles = Array.isArray(profile?.roles) ? profile?.roles : [];
    if (!roles.includes("organizer")) {
      await prisma.profile.update({
        where: { id: updated.userId },
        data: { roles: [...roles, "organizer"] },
      });
    }
  }
}

return NextResponse.json(
  {
    ok: true,
    organizer: {
      id: updated.id,
      status: updated.status,
      displayName: updated.displayName,
      userId: updated.userId,
      changed: true,
    },
  },
  { status: 200 },
);
} catch (error) {
  console.error("[ADMIN] [ORGANIZADORES] [UPDATE-STATUS]", error);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 },
  );
}
}
}

```

app/api/admin/payments/dispute/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { enqueueOperation } from "@/lib/operations/enqueue";

export async function POST(req: NextRequest) {
  const body = await req.json().catch(() => ({}));
  const saleSummaryId = Number(body?.saleSummaryId);
  const paymentIntentId = typeof body?.paymentIntentId === "string" ? body.paymentIntentId : null;
  const reason = typeof body?.reason === "string" ? body.reason : null;

  if (!Number.isFinite(saleSummaryId)) {

```

```

    return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });
}

// Nota: sem RBAC forte (exemplo). Para produção, colocar auth/admin.

try {
  const sale = await prisma.saleSummary.findUnique({
    where: { id: saleSummaryId },
    select: { purchaseId: true, paymentIntentId: true },
  });
  if (!sale) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
  await enqueueOperation({
    operationType: "MARK_DISPUTE",
    dedupeKey: sale.purchaseId ?? sale.paymentIntentId ?? `dispute:${saleSummaryId}`,
    correlations: {
      paymentIntentId: sale.paymentIntentId ?? null,
      purchaseId: sale.purchaseId ?? null,
    },
    payload: {
      saleSummaryId,
      paymentIntentId: sale.paymentIntentId ?? null,
      purchaseId: sale.purchaseId ?? null,
      reason,
    },
  });
  return NextResponse.json({ ok: true, queued: true }, { status: 200 });
} catch (err) {
  console.error("[admin/dispute] erro", err);
  return NextResponse.json({ ok: false, error: "FAILED" }, { status: 500 });
}
}

```

app/api/admin/payments/list/route.ts

```

// app/api/admin/payments/list/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import type { Prisma, PaymentMode } from "@prisma/client";

const PAGE_SIZE = 50;

async function ensureAdmin() {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    return { ok: false as const, status: 401 as const, reason: "UNAUTHENTICATED" };
  }

  const profile = await prisma.profile.findUnique({
    where: { id: user.id },
    select: { roles: true },
  });
  const roles = profile?.roles ?? [];
  const isAdmin = Array.isArray(roles) && roles.includes("admin");
  if (!isAdmin) {
    return { ok: false as const, status: 403 as const, reason: "FORBIDDEN" };
  }
  return { ok: true as const };
}

export async function GET(req: NextRequest) {
  try {
    const admin = await ensureAdmin();
    if (!admin.ok) {
      return NextResponse.json({ ok: false, error: admin.reason }, { status: admin.status });
    }

    const url = new URL(req.url);
    const statusParam = (url.searchParams.get("status") || "ALL").toUpperCase();
    const q = url.searchParams.get("q")?.trim() ?? "";
    const cursorRaw = url.searchParams.get("cursor");
  }
}

```

```

const modeParam = (url.searchParams.get("mode") || "ALL").toUpperCase();

const cursor = cursorRaw ? Number(cursorRaw) : null;
const where: Prisma.PaymentEventWhereInput = {};

if (statusParam !== "ALL") {
  where.status = statusParam;
}

if (modeParam === "LIVE" || modeParam === "TEST") {
  where.mode = modeParam as PaymentMode;
}

if (q) {
  const qNum = Number(q);
  const maybeNumber = Number.isFinite(qNum) ? qNum : null;
  where.OR = [
    { stripePaymentIntentId: { contains: q, mode: "insensitive" } },
    { errorMessage: { contains: q, mode: "insensitive" } },
    ...(maybeNumber ? [{ eventId: maybeNumber }] : []),
    { userId: q },
  ];
}

const items = await prisma.paymentEvent.findMany({
  where,
  orderBy: { id: "desc" },
  take: PAGE_SIZE + 1,
  ...(cursor ? { cursor: { id: cursor }, skip: 1 } : {}),
});

const hasMore = items.length > PAGE_SIZE;
const trimmed = hasMore ? items.slice(0, PAGE_SIZE) : items;
const nextCursor = hasMore ? trimmed[trimmed.length - 1].id ?? null : null;

return NextResponse.json(
  {
    ok: true,
    items: trimmed,
    pagination: { nextCursor, hasMore },
  },
  { status: 200 },
);
} catch (err) {
  console.error("[admin/payments/list]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/admin/payments/overview/route.ts

```

// app/api/admin/payments/overview/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getStripeBaseFees } from "@/lib/platformSettings";
import type { Prisma, PaymentMode } from "@prisma/client";

type Aggregate = {
  grossCents: number;
  discountCents: number;
  platformFeeCents: number;
  stripeFeeCents: number;
  netCents: number;
  tickets: number;
};

async function ensureAdmin() {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {

```

```

        return { ok: false as const, status: 401 as const, reason: "UNAUTHENTICATED" };
    }

    const profile = await prisma.profile.findUnique({
        where: { id: user.id },
        select: { roles: true },
    });
    const roles = profile?.roles ?? [];
    const isAdmin = Array.isArray(roles) && roles.includes("admin");
    if (!isAdmin) {
        return { ok: false as const, status: 403 as const, reason: "FORBIDDEN" };
    }
    return { ok: true as const };
}

const emptyAgg: Aggregate = {
    grossCents: 0,
    discountCents: 0,
    platformFeeCents: 0,
    stripeFeeCents: 0,
    netCents: 0,
    tickets: 0,
};

export async function GET(req: NextRequest) {
    try {
        const admin = await ensureAdmin();
        if (!admin.ok) {
            return NextResponse.json({ ok: false, error: admin.reason }, { status: admin.status });
        }

        const url = new URL(req.url);
        const organizerId = Number(url.searchParams.get("organizerId"));
        const eventId = Number(url.searchParams.get("eventId"));
        const modeParam = (url.searchParams.get("mode") || "ALL").toUpperCase();
        const fromParam = url.searchParams.get("from");
        const toParam = url.searchParams.get("to");

        const fromDate = fromParam ? new Date(fromParam) : null;
        const toDate = toParam ? new Date(toParam) : null;

        const stripeFees = await getStripeBaseFees();
        const estimateStripeFee = (amountCents: number) =>
            Math.max(
                0,
                Math.round((amountCents * (stripeFees.feeBps ?? 0)) / 10_000) +
                (stripeFees.feeFixedCents ?? 0),
            );
    
```

// Filtrar intents por modo (TEST/LIVE) via payment_events

```

        let paymentIntentIds: string[] | null = null;
        if (modeParam === "LIVE" || modeParam === "TEST") {
            const modeFilter: Prisma.PaymentEventWhereInput = { mode: modeParam as PaymentMode };
            if (Number.isFinite(eventId)) {
                modeFilter.eventId = Number(eventId);
            }
            const events = await prisma.paymentEvent.findMany({
                where: modeFilter,
                select: { stripePaymentIntentId: true },
            });
            paymentIntentIds = events.map((e) => e.stripePaymentIntentId).filter(Boolean) as string[];
            if (paymentIntentIds.length === 0) {
                return NextResponse.json(
                    { ok: true, totals: emptyAgg, byOrganizer: [], period: { from: fromDate, to: toDate } },
                    { status: 200 },
                );
            }
        }
    }

    const where: Prisma.SaleSummaryWhereInput = {};
    if (Number.isFinite(organizerId)) {
        where.event = { organizerId: Number(organizerId) };
    }
    if (Number.isFinite(eventId)) {
        where.eventId = Number(eventId);
    }
    if (paymentIntentIds) {
        where.paymentIntentId = { in: paymentIntentIds };
    }
}

```

```

}

if (fromDate || toDate) {
  where.createdAt = {};
  if (fromDate) where.createdAt.gte = fromDate;
  if (toDate) where.createdAt.lte = toDate;
}

const summaries = await prisma.saleSummary.findMany({
  where,
  select: {
    id: true,
    eventId: true,
    subtotalCents: true,
    discountCents: true,
    platformFeeCents: true,
    stripeFeeCents: true,
    totalCents: true,
    netCents: true,
    createdAt: true,
    lines: { select: { quantity: true } },
    event: { select: { organizerId: true, title: true } },
  },
  orderBy: { createdAt: "desc" },
  take: 1000, // segurança para não explodir o painel
});

const totals: Aggregate = { ...emptyAgg };
const byOrganizer = new Map<
  number,
  Aggregate & { organizerId: number; events: Set<number> }
>();

const add = (orgId: number, agg: Aggregate, eventIdValue: number) => {
  if (!byOrganizer.has(orgId)) {
    byOrganizer.set(orgId, { ...emptyAgg, organizerId: orgId, events: new Set() });
  }
  const target = byOrganizer.get(orgId)!;
  target.grossCents += agg.grossCents;
  target.discountCents += agg.discountCents;
  target.platformFeeCents += agg.platformFeeCents;
  target.stripeFeeCents += agg.stripeFeeCents;
  target.netCents += agg.netCents;
  target.tickets += agg.tickets;
  target.events.add(eventIdValue);
};

for (const s of summaries) {
  const gross = s.subtotalCents ?? 0;
  const discount = s.discountCents ?? 0;
  const platformFee = s.platformFeeCents ?? 0;
  const total = s.totalCents ?? gross - discount + platformFee;
  const stripeFee = s.stripeFeeCents != null ? s.stripeFeeCents : estimateStripeFee(total);
  const net =
    s.netCents != null && s.netCents >= 0
    ? s.netCents
    : Math.max(0, total - platformFee - stripeFee);
  const tickets = s.lines.reduce((acc, l) => acc + (l.quantity ?? 0), 0);

  totals.grossCents += gross;
  totals.discountCents += discount;
  totals.platformFeeCents += platformFee;
  totals.stripeFeeCents += stripeFee;
  totals.netCents += net;
  totals.tickets += tickets;

  const organizerKey = s.event?.organizerId ?? 0;
  add(organizerKey, { grossCents: gross, discountCents: discount, platformFeeCents: platformFee, stripeFeeCents: stripeFee, netCents: net, tickets }, s.eventId);
}

// Se não existem sale_summaries, tentar fallback com payment_events
if (summaries.length === 0) {
  const peWhere: Prisma.PaymentEventWhereInput = {};
  if (Number.isFinite(eventId)) peWhere.eventId = Number(eventId);
  if (modeParam === "LIVE" || modeParam === "TEST") peWhere.mode = modeParam as PaymentMode;
  if (fromDate || toDate) {
    peWhere.createdAt = {};
    if (fromDate) peWhere.createdAt.gte = fromDate;
  }
}

```

```

        if (toDate) peWhere.createdAt.lte = toDate;
    }
    const paymentEvents = await prisma.paymentEvent.findMany({
        where: peWhere,
        select: {
            amountCents: true,
            platformFeeCents: true,
            stripeFeeCents: true,
            eventId: true,
        },
    });
    const eventIds = Array.from(
        new Set(paymentEvents.map((p) => p.eventId).filter((id): id is number => Number.isFinite(id))),
    );
    const eventsById = eventIds.length
    ? new Map(
        (
            await prisma.event.findMany({
                where: { id: { in: eventIds } },
                select: { id: true, organizerId: true },
            })
        ).map((ev) => [ev.id, ev.organizerId ?? 0]),
    )
    : new Map<number, number>();

    for (const p of paymentEvents) {
        const gross = p.amountCents ?? 0;
        const discount = 0;
        const platformFee = p.platformFeeCents ?? 0;
        const stripeFee =
            p.stripeFeeCents != null
            ? p.stripeFeeCents
            : estimateStripeFee(p.amountCents ?? 0);
        const net = Math.max(0, gross - platformFee - stripeFee);
        const tickets = 0;

        totals.grossCents += gross;
        totals.discountCents += discount;
        totals.platformFeeCents += platformFee;
        totals.stripeFeeCents += stripeFee;
        totals.netCents += net;
        totals.tickets += tickets;

        const orgId = (p.eventId && eventsById.get(p.eventId)) || 0;
        if (!byOrganizer.has(orgId)) {
            byOrganizer.set(orgId, { ...emptyAgg, organizerId: orgId, events: new Set() });
        }
        const target = byOrganizer.get(orgId)!;
        target.grossCents += gross;
        target.discountCents += discount;
        target.platformFeeCents += platformFee;
        target.stripeFeeCents += stripeFee;
        target.netCents += net;
        target.tickets += tickets;
        if (p.eventId) target.events.add(p.eventId);
    }
}

const list = Array.from(byOrganizer.values()).map((entry) => ({
    organizerId: entry.organizerId,
    grossCents: entry.grossCents,
    discountCents: entry.discountCents,
    platformFeeCents: entry.platformFeeCents,
    stripeFeeCents: entry.stripeFeeCents,
    netCents: entry.netCents,
    tickets: entry.tickets,
    events: entry.events.size,
}));


return NextResponse.json(
{
    ok: true,
    totals,
    byOrganizer: list,
    period: { from: fromDate, to: toDate },
},
{ status: 200 },
);

```

```

    } catch (err) {
      console.error("[admin/payments/overview]", err);
      return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
    }
}

```

app/api/admin/tickets/list/route.ts

```

// app/api/admin/tickets/list/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import type { Prisma, TicketStatus } from "@prisma/client";

// Pequeno helper para garantir que só admins usam estas rotas
async function ensureAdmin(_req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    return { ok: false as const, status: 401 as const, reason: "UNAUTHENTICATED" };
  }

  const profile = await prisma.profile.findUnique({
    where: { id: user.id },
    select: { roles: true },
  });

  const roles = profile?.roles ?? [];
  const isAdmin = Array.isArray(roles) && roles.includes("admin");

  if (!isAdmin) {
    return { ok: false as const, status: 403 as const, reason: "FORBIDDEN" };
  }

  return { ok: true as const, userId: user.id };
}

export async function GET(req: NextRequest) {
  try {
    const adminCheck = await ensureAdmin(req);
    if (!adminCheck.ok) {
      return NextResponse.json(
        { ok: false, error: adminCheck.reason },
        { status: adminCheck.status },
      );
    }
  }

  const { searchParams } = new URL(req.url);

  const rawPage = searchParams.get("page");
  const rawPageSize = searchParams.get("pageSize");
  const rawStatus = searchParams.get("status");
  const rawEventId = searchParams.get("eventId");
  const rawUserId = searchParams.get("userId");
  const search = (searchParams.get("search") || "").trim();

  const page = Math.max(1, Number.parseInt(rawPage || "1", 10) || 1);
  const pageSize = Math.min(
    100,
    Math.max(1, Number.parseInt(rawPageSize || "25", 10) || 25),
  );
  const skip = (page - 1) * pageSize;

  let statusFilter: TicketStatus | undefined;
  let eventIdFilter: number | undefined;
  let userIdFilter: string | undefined;

  // Filtro por status
  if (rawStatus) {
    // Confiamos que o frontend só manda valores válidos de TicketStatus
    statusFilter = rawStatus as TicketStatus;
  }
}

```

```

}

// Filtro por evento
if (rawEventId) {
  const eventIdNum = Number(rawEventId);
  if (!Number.isNaN(eventIdNum)) {
    eventIdFilter = eventIdNum;
  }
}

// Filtro por user
if (rawUserId) {
  userIdFilter = rawUserId;
}

const andConditions: Prisma.TicketWhereInput[] = [];

if (statusFilter) {
  andConditions.push({ status: statusFilter });
}

if (eventIdFilter) {
  andConditions.push({ eventId: eventIdFilter });
}

if (userIdFilter) {
  andConditions.push({ userId: userIdFilter });
}

// Filtro de pesquisa (id do ticket, evento, user)
if (search) {
  const orBlocks: Prisma.TicketWhereInput[] = [
    {
      id: {
        contains: search,
        mode: "insensitive",
      },
    },
    {
      event: {
        title: {
          contains: search,
          mode: "insensitive",
        },
      },
    },
  ];
  andConditions.push({ OR: orBlocks });
}

const where: Prisma.TicketWhereInput =
  andConditions.length === 0
    ? {}
    : andConditions.length === 1
    ? andConditions[0]
    : { AND: andConditions };

const ticketInclude = {
  event: {
    select: {
      id: true,
      title: true,
      slug: true,
      startsAt: true,
    },
  },
  ticketType: {
    select: {
      id: true,
      name: true,
    },
  },
} as const;

type TicketWithRelations = Prisma.TicketGetPayload<{
  include: typeof ticketInclude;
}>;

```

```

const [total, tickets] = await Promise.all([
  prisma.ticket.count({ where }),
  prisma.ticket.findMany({
    where,
    orderBy: { purchasedAt: "desc" },
    skip,
    take: pageSize,
    include: ticketInclude,
  }),
]);

const items = tickets.map((t: TicketWithRelations) => ({
  id: t.id,
  status: t.status,
  purchasedAt: t.purchasedAt,
  pricePaid: t.pricePaid,
  currency: t.currency,
  stripePaymentIntentId: t.stripePaymentIntentId,
  event: t.event
  ?
  {
    id: t.event.id,
    title: t.event.title,
    slug: t.event.slug,
    startsAt: t.event.startsAt,
  }
  :
  null,
  ticketType: t.ticketType
  ?
  {
    id: t.ticketType.id,
    name: t.ticketType.name,
  }
  :
  null,
}));

return NextResponse.json({
  ok: true,
  page,
  pageSize,
  total,
  items,
});
} catch (error) {
  console.error("[admin/tickets/list] Erro ao listar tickets:", error);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 },
  );
}
}
}

```

app/api/admin/users/purge-pending/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { supabaseAdmin } from "@/lib/supabaseAdmin";
import { clearUsernameForOwner } from "@/lib/globalUsernames";
import { logAccountEvent } from "@/lib/accountEvents";

async function ensureAdmin() {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    return { ok: false as const, status: 401 as const, reason: "UNAUTHENTICATED" as const };
  }

  const profile = await prisma.profile.findUnique({
    where: { id: user.id },
    select: { roles: true },
  });
  const roles = profile?.roles ?? [];
}

```

```

const isAdmin = Array.isArray(roles) && roles.includes("admin");
if (!isAdmin) {
  return { ok: false as const, status: 403 as const, reason: "FORBIDDEN" as const };
}
return { ok: true as const };
}

export async function POST(req: NextRequest) {
  try {
    const admin = await ensureAdmin();
    if (!admin.ok) {
      return NextResponse.json({ ok: false, error: admin.reason }, { status: admin.status });
    }
  }

  const now = new Date();
  const limit = Number(req.nextUrl.searchParams.get("limit") ?? 20);

  const pending = await prisma.profile.findMany({
    where: {
      status: "PENDING_DELETE",
      deletionScheduledFor: { lte: now },
    },
    take: limit,
    select: { id: true, username: true, fullName: true, roles: true },
  });

  for (const profile of pending) {
    try {
      await prisma.$transaction(async (tx) => {
        await tx.profile.update({
          where: { id: profile.id },
          data: {
            status: "DELETED",
            deletedAtFinal: now,
            isDeleted: true,
            visibility: "PRIVATE",
            username: null,
            fullName: "Conta apagada",
            bio: null,
            city: null,
            avatarUrl: null,
            contactPhone: null,
            roles: ["user"],
          },
        });
      });

      await clearUsernameForOwner({ ownerType: "user", ownerId: profile.id, tx });

      await tx.organizer.updateMany({
        where: { userId: profile.id },
        data: { userId: null },
      });
    });
  }

  await logAccountEvent({
    userId: profile.id,
    type: "account_delete_completed",
    metadata: { reason: "scheduled_purge" },
  });

  try {
    await supabaseAdmin.auth.admin.deleteUser(profile.id);
  } catch (authErr) {
    console.warn("[purge-pending] falha a remover no auth", authErr);
  }
  } catch (userErr) {
    console.error("[purge-pending] erro ao anonimizar user", { id: profile.id, userErr });
  }
}

return NextResponse.json({ ok: true, processed: pending.length }, { status: 200 });
} catch (err) {
  console.error("[admin/users/purge-pending]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/admin/utilizadores/list/route.ts

```
import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@lib/supabaseServer";
import { prisma } from "@lib/prisma";
import type { Prisma } from "@prisma/client";

async function getAdminProfile() {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    return { user: null, profile: null, error: "UNAUTHENTICATED" };
  }

  const profile = await prisma.profile.findUnique({ where: { id: user.id } });

  if (!profile || !Array.isArray(profile.roles) || !profile.roles.includes("admin")) {
    return { user: null, profile: null, error: "FORBIDDEN" };
  }

  return { user, profile, error: null };
}

export async function GET(req: NextRequest) {
  try {
    // 1) Garantir que é admin
    const { error } = await getAdminProfile();

    if (error === "UNAUTHENTICATED") {
      return NextResponse.json(
        { ok: false, error: "UNAUTHENTICATED" },
        { status: 401 },
      );
    }

    if (error === "FORBIDDEN") {
      return NextResponse.json(
        { ok: false, error: "FORBIDDEN" },
        { status: 403 },
      );
    }
  }

  // 2) Ler query params
  const { searchParams } = new URL(req.url);

  const search = (searchParams.get("search") || "").trim();
  const role = (searchParams.get("role") || "").trim();

  const limitRaw = searchParams.get("limit");
  const offsetRaw = searchParams.get("offset");

  let limit = Number(limitRaw ?? 50);
  let offset = Number(offsetRaw ?? 0);

  if (!Number.isFinite(limit) || limit <= 0 || limit > 200) {
    limit = 50;
  }
  if (!Number.isFinite(offset) || offset < 0) {
    offset = 0;
  }

  // 3) Construir filtro Prisma
  const where: Prisma.ProfileWhereInput = {};

  if (search) {
    where.OR = [
      { username: { contains: search, mode: "insensitive" } },
      { fullName: { contains: search, mode: "insensitive" } },
      { city: { contains: search, mode: "insensitive" } },
    ];
  }
}
```

```

if (role) {
  // Campo roles é um array de strings; usamos has
  where.roles = { has: role };
}

// 4) Query com contagem + items
const [total, items] = await prisma.$transaction([
  prisma.profile.count({ where }),
  prisma.profile.findMany({
    where,
    orderBy: { createdAt: "desc" },
    skip: offset,
    take: limit,
    select: {
      id: true,
      username: true,
      fullName: true,
      city: true,
      roles: true,
      createdAt: true,
    },
  }),
],);

return NextResponse.json({
  ok: true,
  total,
  limit,
  offset,
  items,
});
} catch (err) {
  console.error("[admin/utilizadores/list] Erro a carregar utilizadores:", err);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 },
  );
}
}
}

```

app/api/admin/utilizadores/manage/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { supabaseAdmin } from "@/lib/supabaseAdmin";

type AdminProfileResult =
  | { user: { id: string } | null; error: "UNAUTHENTICATED" | "FORBIDDEN" }
  | { user: { id: string }; error: null };

async function getAdminProfile(): Promise<AdminProfileResult> {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    return { user: null, error: "UNAUTHENTICATED" };
  }

  const profile = await prisma.profile.findUnique({ where: { id: user.id } });
  if (!profile || !Array.isArray(profile.roles) || !profile.roles.includes("admin")) {
    return { user: null, error: "FORBIDDEN" };
  }

  return { user, error: null };
}

type Action = "ban" | "unban" | "hard_delete";

async function hardDeleteAuthUser(userId: string) {
  const { error } = await supabaseAdmin.auth.admin.deleteUser(userId, false);
  if (!error) return { ok: true, note: null as string | null };
}

```

```

// Se já não existir no Auth, tratamos como sucesso para não bloquear a limpeza local
const message =
  typeof error === "object" && error && "message" in error
  ? String(error as { message?: unknown }).message ?? ""
  : "";
if (error?.status === 404 || message.includes("not found")) {
  return { ok: true, note: "AUTH_USER_ALREADY_REMOVED" as const };
}

return { ok: false, note: null, error };
}

export async function POST(req: NextRequest) {
  // 1) Auth check
  const { error } = await getAdminProfile();
  if (error === "UNAUTHENTICATED") {
    return NextResponse.json({ ok: false, error }, { status: 401 });
  }
  if (error === "FORBIDDEN") {
    return NextResponse.json({ ok: false, error }, { status: 403 });
  }

  // 2) Body parsing (JSON ou form)
  let userId: string | undefined;
  let action: Action | undefined;
  try {
    const contentType = req.headers.get("content-type") || "";
    if (contentType.includes("application/json")) {
      const body = await req.json();
      userId = body?.userId;
      action = body?.action;
    } else {
      const form = await req.formData();
      userId = (form.get("userId") as string | null) ?? undefined;
      action = (form.get("action") as Action | null) ?? undefined;
    }
  } catch (err) {
    console.error("[admin/utilizadores/manage] parse error", err);
    return NextResponse.json({ ok: false, error: "BAD_REQUEST" }, { status: 400 });
  }

  if (!userId || !action) {
    return NextResponse.json(
      { ok: false, error: "MISSING_PARAMS" },
      { status: 400 },
    );
  }

  try {
    if (action === "hard_delete") {
      // Remover do Auth e limpar profile para permitir recriação
      const authResult = await hardDeleteAuthUser(userId);
      if (!authResult.ok) throw authResult.error;

      await prisma.profile.deleteMany({ where: { id: userId } });
      return NextResponse.json({
        ok: true,
        message: "Utilizador removido em definitivo.",
        note: authResult.note,
      });
    }

    if (action === "ban") {
      // Banir no Auth e marcar profile como inativo
      const { error: banErr } = await supabaseAdmin.auth.admin.updateUserById(userId, {
        ban_duration: "87600h", // ~10 anos
      } as any);
      if (banErr) throw banErr;
      await prisma.profile.updateMany({
        where: { id: userId },
        data: { isDeleted: true, deletedAt: new Date() },
      });
      return NextResponse.json({ ok: true, message: "Utilizador banido." });
    }

    if (action === "unban") {
      const { error: unbanErr } = await supabaseAdmin.auth.admin.updateUserById(userId, {
        ban_duration: "0h",
      });
    }
  }
}

```

```

    } as any);
    if (unbanErr) throw unbanErr;
    await prisma.profile.updateMany({
      where: { id: userId },
      data: { isDeleted: false, deletedAt: null },
    });
    return NextResponse.json({ ok: true, message: "Utilizador reativado." });
  }

  return NextResponse.json(
    { ok: false, error: "UNKNOWN_ACTION" },
    { status: 400 },
  );
}
} catch (err) {
  console.error("[admin/utilizadores/manage] action error:", err);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 },
  );
}
}
}

```

app/api/auth/callback/claim/route.ts

```

import { NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { claimIdentity } from "@/lib/ownership/claimIdentity";

export async function POST() {
  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data?.user) {
    return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
  }
  const email = data.user.email;
  if (!email) {
    return NextResponse.json({ ok: false, error: "EMAIL_MISSING" }, { status: 400 });
  }
  await claimIdentity(email, data.user.id);
  return NextResponse.json({ ok: true });
}

```

app/api/auth/check-email/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";

/**
 * Endpoint simples para verificar se um email está bloqueado por conta PENDING_DELETE.
 * GET /api/auth/check-email?email=...
 */
export async function GET(req: NextRequest) {
  try {
    const email = req.nextUrl.searchParams.get("email");
    if (!email || !email.includes("@")) {
      return NextResponse.json({ ok: false, error: "INVALID_EMAIL" }, { status: 400 });
    }
    const normalized = email.trim().toLowerCase();
    const authUser = await prisma.users.findFirst({
      where: { email: normalized },
      select: { id: true },
    });
    const pending = authUser
      ? await prisma.profile.findFirst({
          where: { id: authUser.id, status: "PENDING_DELETE" },
          select: { deletionScheduledFor: true },
        })
      : null;
    if (pending) {
      return NextResponse.json(
        {
          ok: true,
          blocked: true,
          message:

```

```

        "Este email está associado a uma conta marcada para eliminação. Inicia sessão para a recuperar ou usa outro email.",
        deletionScheduledFor: pending.deletionScheduledFor,
    },
    { status: 200 },
);
}
return NextResponse.json({ ok: true, blocked: false }, { status: 200 });
} catch (err) {
    console.error("[auth/check-email] erro", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/auth/clear/route.ts

```

import { NextResponse } from "next/server";
import { cookies } from "next/headers";

// Utilitário para limpar cookies locais (incluindo os sb- do Supabase) quando ficam corrompidos.
export async function POST() {
    try {
        const store = await cookies();
        const all = store.getAll();

        for (const c of all) {
            try {
                store.set({
                    name: c.name,
                    value: "",
                    path: "/",
                    maxAge: 0,
                });
            } catch (err) {
                console.error("[api/auth/clear] erro a limpar cookie", c.name, err);
            }
        }

        return NextResponse.json({ ok: true, cleared: all.map((c) => c.name) });
    } catch (err) {
        console.error("[api/auth/clear] erro inesperado:", err);
        return NextResponse.json({ ok: false, error: "CLEAR_FAILED" }, { status: 500 });
    }
}

```

app/api/auth/logout/route.ts

```

import { NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";

export async function POST() {
    try {
        const supabase = await createSupabaseServer();
        const { error } = await supabase.auth.signOut();

        if (error) {
            console.error("[auth/logout] supabase signOut error:", error);
            return NextResponse.json({ success: false, error: error.message }, { status: 500 });
        }

        return NextResponse.json({ success: true });
    } catch (err) {
        console.error("[auth/logout] unexpected error:", err);
        return NextResponse.json({ success: false, error: "Erro ao terminar sessão." }, { status: 500 });
    }
}

```

app/api/auth/me/route.ts

```

// app/api/auth/me/route.ts
import { NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getNotificationPrefs } from "@/lib/notifications";

```

```

import { prisma } from "@/lib/prisma";
import type { User } from "@supabase/supabase-js";
import { setUsernameForOwner, UsernameTakenError } from "@/lib/globalUsernames";

type SupabaseUserMetadata = {
  full_name?: string;
  name?: string;
  avatar_url?: string;
  pending_username?: string;
};

type ApiAuthMeResponse = {
  user: {
    id: string;
    email: string | null;
    emailConfirmed: boolean;
  } | null;
  profile: {
    id: string;
    username: string | null;
    fullName: string | null;
    avatarUrl: string | null;
    bio: string | null;
    city: string | null;
    contactPhone: string | null;
    favouriteCategories: string[];
    onboardingDone: boolean;
    roles: string[];
    visibility: string;
    allowEmailNotifications: boolean;
    allowEventReminders: boolean;
    allowFriendRequests: boolean;
    allowSalesAlerts?: boolean;
    allowSystemAnnouncements?: boolean;
    profileVisibility: "PUBLIC" | "PRIVATE";
  } | null;
  needsEmailConfirmation?: boolean;
};

export async function GET() {
  try {
    const supabase = await createSupabaseServer();

    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      // Sessão ausente ou inválida → 401 limpo (evita spam de logs)
      return NextResponse.json<ApiAuthMeResponse>(
        { user: null, profile: null },
        { status: 401 },
      );
    }

    const supaUser = user as User;
    const emailConfirmed =
      Boolean(supaUser.email_confirmed_at) ||
      Boolean((supaUser as { confirmed_at?: string | null })?.confirmed_at) ||
      false;

    const userMetadata = (user.user_metadata ?? {}) as SupabaseUserMetadata;

    const userId = user.id;

    // Garantir Profile 1-1 com auth.users e prefs
    const [profileFromDb, notificationPrefs] = await Promise.all([
      prisma.profile.findUnique({
        where: { id: userId },
      }),
      getNotificationPrefs(userId).catch(() => null),
    ]);
    let profile =
      profileFromDb ??
      (await prisma.profile.create({
        data: {
          id: userId,

```

```

        username: null,
        fullName: userMetadata.full_name ?? userMetadata.name ?? null,
        avatarUrl: userMetadata.avatar_url ?? null,
        roles: ["user"],
        visibility: "PUBLIC",
        allowEmailNotifications: true,
        allowEventReminders: true,
        allowFriendRequests: true,
    },
));
}

const pendingUsername = typeof userMetadata.pending_username === "string" ? userMetadata.pending_username : null;

// Atribuir username pendente se ainda não existir
if (!profile.username && pendingUsername) {
    try {
        await prisma.$transaction(async (tx) => {
            await setUsernameForOwner({
                username: pendingUsername,
                ownerType: "user",
                ownerId: userId,
                tx,
            });
            await tx.profile.update({
                where: { id: userId },
                data: { username: pendingUsername },
            });
        });
        profile = await prisma.profile.findUnique({ where: { id: userId } });
    } catch (err) {
        if (err instanceof UsernameTakenError) {
            // outro utilizador já registou o @ entretanto; deixa username nulo
            console.warn("[auth/me] pending_username já ocupado");
        } else {
            console.error("[auth/me] erro ao aplicar pending_username:", err);
        }
    }
}

if (!profile) {
    throw new Error("PROFILE_MISSING");
}

const profileVisibility: "PUBLIC" | "PRIVATE" =
profile.visibility === "PRIVATE" ? "PRIVATE" : "PUBLIC";

const safeProfile: ApiAuthMeResponse["profile"] = {
    id: profile.id,
    username: profile.username,
    fullName: profile.fullName,
    avatarUrl: profile.avatarUrl,
    bio: profile.bio,
    city: profile.city,
    contactPhone: profile.contactPhone,
    favouriteCategories: profile.favouriteCategories,
    onboardingDone: profile.onboardingDone,
    roles: profile.roles,
    visibility: profile.visibility,
    allowEmailNotifications: profile.allowEmailNotifications,
    allowEventReminders: profile.allowEventReminders,
    allowFriendRequests: profile.allowFriendRequests,
    allowSalesAlerts: notificationPrefs?.allowSalesAlerts ?? true,
    allowSystemAnnouncements: notificationPrefs?.allowSystemAnnouncements ?? true,
    profileVisibility,
};

// Se email não está confirmado, força o frontend a continuar em modo "verify"
if (!emailConfirmed) {
    return NextResponse.json<ApiAuthMeResponse>(
    {
        user: {
            id: user.id,
            email: user.email ?? null,
            emailConfirmed,
        },
        profile: null,
        needsEmailConfirmation: true,
    },
)
}

```

```

        { status: 401 },
    );
}

return NextResponse.json<ApiAuthMeResponse>(
{
    user: {
        id: user.id,
        email: user.email ?? null,
        emailConfirmed,
    },
    profile: safeProfile,
},
{ status: 200 },
);
} catch (err) {
console.error("GET /api/auth/me error:", err);
return NextResponse.json<ApiAuthMeResponse>(
    { user: null, profile: null },
    { status: 200 }
);
}
}
}

```

app/api/auth/refresh/route.ts

```

// app/api/auth/refresh/route.ts
export const runtime = "nodejs";
export const dynamic = "force-dynamic";

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";

/**
 * Sincroniza a sessão do supabase (tokens vindos do browser) para cookies HttpOnly.
 * Espera body JSON com { access_token, refresh_token }.
 */
export async function POST(req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const body = (await req.json().catch(() => null)) as
            | { access_token?: string; refresh_token?: string }
            | null;

        const access_token = body?.access_token ?? null;
        const refresh_token = body?.refresh_token ?? null;

        if (!access_token || !refresh_token) {
            return NextResponse.json(
                { ok: false, error: "MISSING_TOKENS" },
                { status: 400 },
            );
        }

        const { error } = await supabase.auth.setSession({
            access_token,
            refresh_token,
        });

        if (error) {
            console.error("[auth/refresh] setSession error:", error);
            return NextResponse.json(
                { ok: false, error: error.message },
                { status: 400 },
            );
        }

        return NextResponse.json({ ok: true });
    } catch (err) {
        console.error("[auth/refresh] unexpected error:", err);
        return NextResponse.json(
            { ok: false, error: "SERVER_ERROR" },
            { status: 500 },
        );
    }
}

```

```
}
```

app/api/auth/resend-otp/route.ts

```
import { NextRequest, NextResponse } from "next/server";
import { supabaseAdmin } from "@/lib/supabaseAdmin";
import { resend } from "@/lib/resend";
import { env } from "@/lib/env";

export async function POST(req: NextRequest) {
  try {
    const { email } = await req.json();

    if (!email) {
      return NextResponse.json(
        { error: "Email em falta." },
        { status: 400 }
      );
    }

    // Gera novo OTP de signup e envia via Resend (mesmo template do send-otp)
    const siteUrl =
      process.env.NEXT_PUBLIC_SITE_URL ??
      process.env.NEXT_PUBLIC_BASE_URL ??
      process.env.SITE_URL ??
      env.supabaseUrl;

    const { data, error } = await supabaseAdmin.auth.admin.generateLink({
      type: "signup",
      email,
      password: undefined,
      options: {
        redirectTo: `${siteUrl}/auth/callback`,
      },
    } as any);

    if (error) {
      const errorCode =
        typeof error === "object" && error && "code" in error
        ? (error as { code?: string }).code
        : undefined;
      if (errorCode === "email_exists") {
        return NextResponse.json(
          { error: "Email já registrado. Usa login ou Google.", code: "email_exists" },
          { status: 409 },
        );
      }
      console.error("[resend-otp] generateLink error:", error);
      return NextResponse.json(
        { error: "Não foi possível reenviar o código. Tenta mais tarde." },
        { status: 500 }
      );
    }

    if (!data?.properties?.email_otp) {
      console.error("[resend-otp] missing email_otp in response");
      return NextResponse.json(
        { error: "Não foi possível gerar o código. Tenta mais tarde." },
        { status: 500 },
      );
    }

    const code: string = data.properties.email_otp;

    const html = `
      <table width="100%" cellspacing="0" cellpadding="0" style="background:#0b0b12;padding:32px 0;font-family:Inter,system-ui,-apple-system,BlinkMacSystemFont,sans-serif;">
        <tr>
          <td align="center">
            <table width="480" cellpadding="0" cellspacing="0" style="background:#0f111a;border-radius:18px;overflow:hidden; border:1px solid rgba(255,255,255,0.06);box-shadow:0 16px 60px rgba(0,0,0,0.55);color:#f7f7f7;">
              <tr>
                <td style="padding:28px 32px;background:linear-gradient(135deg,#ff00c8,#5b8bff);color:#0b0b12;font-
```

```

size:22px;font-weight:800;letter-spacing:-0.3px;">
    ORYA · Código de verificação
  </td>
</tr>
<tr>
  <td style="padding:28px 32px;color:#e5e7eb;font-size:14px;line-height:1.6;">
    <p style="margin:0 0 12px 0;">Olá! Aqui está o teu código de 6 dígitos para continuares na ORYA.</p>
    <p style="margin:0 0 24px 0;">Introduz este código na app para verificares o teu email:</p>
    <div style="display:inline-block;padding:12px 18px;border-radius:12px;background:#111522;border:1px solid
rgba(255,255,255,0.08);font-size:24px;font-weight:800;letter-spacing:6px;color:#fdfdfd;">
      ${code}
    </div>
    <p style="margin:24px 0 0 0;color:#aeb7c6;font-size:13px;">Se não foste tu, ignora este email.</p>
  </td>
</tr>
<tr>
  <td style="padding:18px 32px;color:#7a8397;font-size:12px;background:#0c0f18;border-top:1px solid
rgba(255,255,255,0.06);">
    Obrigado por confiaras na ORYA.
  </td>
</tr>
</table>
</td>
</tr>
</table>
`;
}

try {
  await resend.emails.send({
    from: env.resendFrom,
    to: email,
    subject: `Código ORYA: ${code}`,
    html,
  });
} catch (mailErr) {
  console.error("[resend-otp] resend error", { mailErr, email, env: process.env.NODE_ENV });
  return NextResponse.json(
    { error: "Não foi possível reenviar o código. Tenta mais tarde." },
    { status: 502 },
  );
}

return NextResponse.json({ success: true });
} catch (err) {
  console.error("Erro em /api/auth/resend-otp:", err);
  return NextResponse.json(
    { error: "Erro interno." },
    { status: 500 }
  );
}
}
}

```

app/api/auth/resolve-identifier/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createClient } from "@supabase/supabase-js";
import { env } from "@/lib/env";

export async function POST(req: NextRequest) {
  try {
    const body = (await req.json().catch(() => null)) as { identifier?: string } | null;
    const identifier = body?.identifier?.trim();

    if (!identifier) {
      return NextResponse.json({ ok: false, error: "IDENTIFIER_REQUIRED" }, { status: 400 });
    }

    if (identifier.includes("@")) {
      return NextResponse.json({ ok: true, email: identifier.toLowerCase() });
    }

    const profile = await prisma.profile.findFirst({
      where: { username: { equals: identifier, mode: "insensitive" } },
      select: { id: true },
    });
  }
}

```

```

if (!profile) {
  return NextResponse.json({ ok: false, error: "USER_NOT_FOUND" }, { status: 404 });
}

const adminClient = createClient(env.supabaseUrl, env.serviceRoleKey);
const { data, error } = await adminClient.auth.admin.getUserById(profile.id);

if (error || !data?.user?.email) {
  return NextResponse.json({ ok: false, error: "EMAIL_NOT_FOUND" }, { status: 404 });
}

return NextResponse.json({ ok: true, email: data.user.email.toLowerCase() });
} catch (err) {
  console.error("[resolve-identifier] error", err);
  return NextResponse.json({ ok: false, error: "SERVER_ERROR" }, { status: 500 });
}
}

```

app/api/auth/send-otp/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { supabaseAdmin } from "@/lib/supabaseAdmin";
import { resend } from "@/lib/resend";
import { env } from "@/lib/env";
import { normalizeAndValidateUsername, checkUsernameAvailability } from "@/lib/globalUsernames";

const EMAIL_REGEX = /^[^@\s]+@[^\s]+\.\[^@\s]+$/;

function buildEmailHtml(code: string) {
  return `
    <table width="100%" cellspacing="0" cellpadding="0" style="background:#0b0b12;padding:32px 0;font-family:Inter,system-ui,-apple-system,BlinkMacSystemFont,sans-serif;">
      <tr>
        <td align="center">
          <table width="480" cellpadding="0" cellspacing="0" style="background:#0f111a;border-radius:18px;overflow:hidden;border:1px solid rgba(255,255,255,0.06);box-shadow:0 16px 60px rgba(0,0,0,0.55);color:#f7f7f7;">
            <tr>
              <td style="padding:28px 32px;background:linear-gradient(135deg,#ff00c8,#5b8bff);color:#0b0b12;font-size:22px;font-weight:800;letter-spacing:-0.3px;">
                ORYA · Código de verificação
              </td>
            </tr>
            <tr>
              <td style="padding:28px 32px;color:#e5e7eb;font-size:14px;line-height:1.6;">
                <p style="margin:0 0 12px 0;">Olá! Aqui está o teu código de 6 dígitos para continuares na ORYA.</p>
                <p style="margin:0 0 24px 0;">Introduz este código na app para verificares o teu email:</p>
                <div style="display:inline-block;padding:12px 18px;border-radius:12px;background:#111522;border:1px solid rgba(255,255,255,0.08);font-size:24px;font-weight:800;letter-spacing:6px;color:#fdfdfd;">
                  ${code}
                </div>
                <p style="margin:24px 0 0 0;color:#aeb7c6;font-size:13px;">Se não foste tu, ignora este email.</p>
              </td>
            </tr>
            <tr>
              <td style="padding:18px 32px;color:#7a8397;font-size:12px;background:#0c0f18;border-top:1px solid rgba(255,255,255,0.06);">
                Obrigado por confiar na ORYA.
              </td>
            </tr>
          </table>
        </td>
      </tr>
    </table>
  `;
}

export async function POST(req: NextRequest) {
  try {
    const body = (await req.json().catch(() => null)) as
      | { email?: string; password?: string | null; username?: string | null; fullName?: string | null }
      | null;

    const rawEmail = body?.email?.toLowerCase().trim() ?? "";
    const password = body?.password ?? null;
    const rawUsername = body?.username?.trim() ?? "";
  }

```

```

const rawFullName = body?.fullName ?? "";

if (!rawEmail || !EMAIL_REGEX.test(rawEmail)) {
  return NextResponse.json(
    { ok: false, error: "Email inválido." },
    { status: 400 },
  );
}

if (password !== null && password !== undefined && password.length < 6) {
  return NextResponse.json(
    { ok: false, error: "A password deve ter pelo menos 6 caracteres." },
    { status: 400 },
  );
}

let usernameNormalized: string | null = null;
if (rawUsername) {
  const usernameValidation = normalizeAndValidateUsername(rawUsername);
  if (!usernameValidation.ok) {
    return NextResponse.json(
      { ok: false, error: usernameValidation.error, code: "USERNAME_INVALID" },
      { status: 400 },
    );
  }

  const availability = await checkUsernameAvailability(usernameValidation.username);
  if (availability.ok && availability.available === false) {
    return NextResponse.json(
      { ok: false, error: "Este @ já está a ser usado – escolhe outro.", code: "USERNAME_TAKEN" },
      { status: 409 },
    );
  }
  usernameNormalized = usernameValidation.username;
}

const fullName = rawFullName?.trim() || null;

const siteUrl =
  process.env.NEXT_PUBLIC_SITE_URL ??
  process.env.NEXT_PUBLIC_BASE_URL ??
  process.env.SITE_URL ??
  env.supabaseUrl;

// Apenas OTP de signup. Se email já existir → pedir login/Google.
const linkPayload: Record<string, unknown> = {
  type: "signup",
  email: rawEmail,
  options: {
    redirectTo: `${siteUrl}/auth/callback`,
    data: {
      ...(usernameNormalized ? { pending_username: usernameNormalized } : {}),
      full_name: fullName || undefined,
    },
  },
};

if (password) {
  linkPayload.password = password;
}

const { data, error } = await supabaseAdmin.auth.admin.generateLink(linkPayload as any);

if (error) {
  const errorCode =
    typeof error === "object" && error && "code" in error
    ? (error as { code?: string }).code
    : undefined;
  if (errorCode === "email_exists") {
    return NextResponse.json(
      {
        ok: false,
        error: "Este email já tem conta. Inicia sessão ou usa o Google.",
        code: "email_exists",
      },
      { status: 409 },
    );
  }
  console.error("[send-otp] generateLink error:", error);
  return NextResponse.json(
    {
      ok: false,
      error: "Não foi possível gerar o código. Tenta novamente dentro de alguns minutos.",
    },
  );
}

```

```

        details: typeof error === "object" ? (error as unknown as Record<string, unknown>) : undefined,
    },
    { status: 500 },
);
}

const otp = data?.properties?.email_otp ?? null;
if (!otp) {
    console.error("[send-otp] missing email_otp in response");
    return NextResponse.json(
        { ok: false, error: "Não foi possível gerar o código. Tenta novamente." },
        { status: 500 },
    );
}

try {
    await resend.emails.send({
        from: env.resendFrom,
        to: rawEmail,
        subject: `Código ORYA: ${otp}`,
        html: buildEmailHtml(otp),
    });
} catch (mailErr) {
    console.error("[send-otp] resend error", { mailErr, email: rawEmail, env: process.env.NODE_ENV });
    return NextResponse.json(
        {
            ok: false,
            error: "Não foi possível enviar o código. Tenta novamente dentro de alguns minutos.",
        },
        { status: 502 },
    );
}

return NextResponse.json({ ok: true });
} catch (err) {
    console.error("[send-otp] error:", err);
    return NextResponse.json(
        { ok: false, error: "Erro inesperado ao enviar código." },
        { status: 500 },
    );
}
}
}

```

app/api/checkout/resale/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { resolveOwner } from "@/lib/ownership/resolveOwner";
import { env } from "@/lib/env";

/**
 * F5-12 – Checkout específico para revenda de bilhetes
 *
 * Body esperado:
 * {
 *   resaleId: string;
 * }
 */
export async function POST(req: NextRequest) {
    try {
        // 1. Autenticação – garantir que o comprador está autenticado
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error: authError,
        } = await supabase.auth.getUser();

        if (authError) {
            console.error("Error getting user in /api/checkout/resale:", authError);
        }

        if (!user) {
            return NextResponse.json(
                { ok: false, error: "UNAUTHENTICATED" },
                { status: 401 }
            );
        }

        // 2. Verificar se o usuário é uma revenda
        const isResaleUser = user?.is_resale === true;
        if (!isResaleUser) {
            return NextResponse.json(
                { ok: false, error: "User is not a reseller" },
                { status: 403 }
            );
        }

        // 3. Obter informações da revenda
        const resaleUser = await prisma.resaleUser.findUnique({
            where: {
                id: user.id,
            },
        });

        if (!resaleUser) {
            return NextResponse.json(
                { ok: false, error: "Resale user not found" },
                { status: 404 }
            );
        }

        // 4. Processar checkout
        const checkoutData = await processCheckout(resaleUser, supabase);
        const checkoutResponse = NextResponse.json(checkoutData);
        return checkoutResponse;
    } catch (error) {
        console.error("Error processing checkout:", error);
        return NextResponse.json(
            { ok: false, error: "Internal server error" },
            { status: 500 }
        );
    }
}

```

```

    );
}

const buyerUserId = user.id;

// 2. Ler body e validar
const body = (await req.json().catch(() => null)) as
  | { resaleId?: string }
  | null;

if (!body || typeof body !== "object" || !body.resaleId) {
  return NextResponse.json(
    { ok: false, error: "INVALID_BODY" },
    { status: 400 }
  );
}

const { resaleId } = body;

// 3. Carregar revenda + ticket + evento
const resale = await prisma.ticketResale.findUnique({
  where: { id: resaleId },
  include: {
    ticket: {
      include: {
        event: true,
      },
    },
  },
});

if (!resale || !resale.ticket || !resale.ticket.event) {
  return NextResponse.json(
    { ok: false, error: "RESALE_NOT_FOUND" },
    { status: 404 }
  );
}

const { ticket } = resale;
const event = ticket.event;

// 4. Validar estado da revenda e do bilhete
if (resale.status !== "LISTED") {
  return NextResponse.json(
    { ok: false, error: "RESALE_NOT_AVAILABLE" },
    { status: 400 }
  );
}

if (ticket.status !== "ACTIVE") {
  return NextResponse.json(
    { ok: false, error: "TICKET_NOT_ACTIVE" },
    { status: 400 }
  );
}

// 5. Impedir que o vendedor compre o proprio bilhete
if (resale.sellerUserId === buyerUserId) {
  return NextResponse.json(
    { ok: false, error: "CANNOT_BUY_OWN_RESALE" },
    { status: 400 }
  );
}

// 6. Determinar o preço em céntimos (compativel com diferentes schemas)
const rawAmount =
  (resale as { priceCents?: number | null; price?: number | null })
    .priceCents ??
  (resale as { priceCents?: number | null; price?: number | null }).price ??
  null;

if (typeof rawAmount !== "number" || rawAmount <= 0) {
  console.error("Invalid resale price for resaleId:", resaleId, rawAmount);
  return NextResponse.json(
    { ok: false, error: "INVALID_RESALE_PRICE" },
    { status: 400 }
  );
}

```

```

const amountCents = rawAmount;
const ownerResolved = await resolveOwner({ sessionUserId: buyerUserId, guestEmail: null });
const origin = req.nextUrl.origin || env.appBaseUrl || "";

const res = await fetch(`${origin}/api/payments/intent`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    cookie: req.headers.get("cookie") ?? "",
  },
  body: JSON.stringify({
    slug: event.slug ?? null,
    items: [
      {
        ticketId: resale.ticket.ticketTypeId,
        quantity: 1,
        unitPriceCents: amountCents,
        currency: (event.currency || "EUR").toUpperCase(),
      },
    ],
    paymentScenario: "RESALE",
    resaleId: resale.id,
    ticketId: resale.ticket.id,
    total: rawAmount / 100,
  }),
});

const data = await res.json().catch(() => null);
if (!res.ok || !data?.ok || !data?.clientSecret) {
  console.error("Error in /api/checkout/resale intent:", { status: res.status, data });
  return NextResponse.json(
    { ok: false, error: data?.error ?? "INTENT_CREATION_FAILED", code: data?.code ?? null },
    { status: res.status },
  );
}

return NextResponse.json(
  {
    ok: true,
    clientSecret: data.clientSecret,
    paymentIntentId: data.paymentIntentId,
    purchaseId: data.purchaseId,
    paymentScenario: "RESALE",
    preview: {
      title: event.title,
      ticketTypeName: resale.ticket.ticketType?.name ?? null,
      priceCents: amountCents,
      currency: event.currency || "EUR",
      sellerName: resale.ticket.user?.username ?? resale.ticket.user?.fullName ?? null,
    },
  },
  { status: 200 },
);
} catch (error) {
  console.error("Error in /api/checkout/resale:", error);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 }
  );
}
}
}

```

app/api/checkout/status/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@lib/prisma";
import { Prisma } from "@prisma/client";
import { enqueueOperation } from "@lib/operations/enqueue";
import { runOperationsBatch } from "@app/api/internal/worker/operations/route";

type Status =
  | "PENDING"
  | "PROCESSING"
  | "REQUIRES_ACTION"
  | "PAID"

```

```

| "FAILED"
| "REFUNDED"
| "DISPUTED";

const FINAL_STATUSES: Status[] = ["PAID", "FAILED", "REFUNDED", "DISPUTED"];
const FREE_PLACEHOLDER_INTENT_ID = "FREE_CHECKOUT";
const NO_STORE_HEADERS = { "Cache-Control": "no-store" };
const STUCK_MS = 5 * 60 * 1000; // 5 minutos para considerar uma Operation presa

function cleanParam(v: string | null) {
  const s = (v ?? "").trim();
  return s ? s : null;
}

export async function GET(req: NextRequest) {
  const url = new URL(req.url);
  const purchaseId = cleanParam(url.searchParams.get("purchaseId"));
  const paymentIntentIdRaw = cleanParam(url.searchParams.get("paymentIntentId"));
  // Free checkout não tem PaymentIntent Stripe; alguns flows antigos guardam um placeholder.
  const paymentIntentId =
    paymentIntentIdRaw === FREE_PLACEHOLDER_INTENT_ID ? null : paymentIntentIdRaw;

  if (!purchaseId && !paymentIntentId) {
    return NextResponse.json(
      {
        ok: false,
        status: "FAILED" as Status,
        error: "MISSING_ID",
        code: "MISSING_ID",
        retryable: false,
        nextAction: "NONE",
      },
      { status: 400, headers: NO_STORE_HEADERS },
    );
  }

  try {
    // -----
    // 1) SSOT: SaleSummary (DB confirmou a compra)
    // -----
    const summaryWhere: Prisma.SaleSummaryWhereInput = { OR: [] };
    if (purchaseId) summaryWhere.OR!.push({ purchaseId });
    if (paymentIntentId) summaryWhere.OR!.push({ paymentIntentId });

    const summary = await prisma.saleSummary.findFirst({
      where: summaryWhere,
      select: {
        id: true,
        paymentIntentId: true,
        purchaseId: true,
        totalCents: true,
        createdAt: true,
      },
    });

    if (summary) {
      return NextResponse.json(
        {
          ok: true,
          status: "PAID" as Status,
          final: true,
          // purchaseId é a âncora universal; se não existir (edge), fazemos fallback seguro.
          purchaseId: summary.purchaseId ?? purchaseId ?? paymentIntentId,
          paymentIntentId: summary.paymentIntentId ?? paymentIntentId,
          code: "PAID",
          retryable: false,
          nextAction: "NONE",
          errorMessage: null,
        },
        { status: 200, headers: NO_STORE_HEADERS },
      );
    }
  }

  // -----
  // 1b) Operations (modo worker): se não há SaleSummary ainda, ver estado da Operation
  // -----
  if (paymentIntentId || purchaseId) {
    const op = await prisma.operation.findFirst({

```

```

where: {
  OR: [
    purchaseId ? { purchaseId } : undefined,
    paymentIntentId ? { paymentIntentId } : undefined,
  ].filter(Boolean) as Prisma.OperationWhereInput[],
},
orderBy: { updatedAt: "desc" },
select: {
  status: true,
  operationType: true,
  lastError: true,
  purchaseId: true,
  paymentIntentId: true,
  updatedAt: true,
},
};

const now = Date.now();
const opUpdatedAt = op?.updatedAt ? op.updatedAt.getTime() : 0;
const stuck =
  op && (op.status === "PENDING" || op.status === "RUNNING") && now - opUpdatedAt > STUCK_MS;
const needsQueue = !op || op.status === "FAILED" || op.status === "DEAD_LETTER" || stuck;

// Auto-requeue para evitar ficar preso em PROCESSING
if (needsQueue && (paymentIntentId || purchaseId)) {
  const dedupe = paymentIntentId ?? purchaseId!;
  await enqueueOperation({
    operationType: "FULFILL_PAYMENT",
    dedupeKey: dedupe,
    correlations: { purchaseId: purchaseId ?? null, paymentIntentId: paymentIntentId ?? null },
    payload: { purchaseId, paymentIntentId },
  });
}

// Auto-disparar worker em linha (mesmo em prod) mas com salvaguarda: até 3 batches
if (op && (op.status === "PENDING" || op.status === "RUNNING" || needsQueue)) {
  for (let i = 0; i < 3; i++) {
    try {
      await runOperationsBatch();
    } catch (err) {
      console.warn("[checkout/status] auto-run worker falhou (ignorado)", err);
      break;
    }
    // Após cada batch, verifica se o SaleSummary já existe para este purchase/paymentIntent
    const summaryAfter = await prisma.saleSummary.findFirst({
      where: {
        OR: [
          purchaseId ? { purchaseId } : undefined,
          paymentIntentId ? { paymentIntentId } : undefined,
        ].filter(Boolean) as Prisma.SaleSummaryWhereInput[],
      },
      select: { purchaseId: true, paymentIntentId: true },
    });
    if (summaryAfter) {
      return NextResponse.json(
        {
          ok: true,
          status: "PAID" as Status,
          final: true,
          purchaseId: summaryAfter.purchaseId ?? purchaseId ?? paymentIntentId,
          paymentIntentId: summaryAfter.paymentIntentId ?? paymentIntentId,
          code: "PAID",
          retryable: false,
          nextAction: "NONE",
          errorMessage: null,
        },
        { status: 200, headers: NO_STORE_HEADERS },
      );
    }
  }
}

if (op) {
  const opStatusMap: Record<string, Status> = {
    PENDING: "PROCESSING",
    RUNNING: "PROCESSING",
    FAILED: "FAILED",
    DEAD_LETTER: "FAILED",
}

```

```

        SUCCEEDED: "PROCESSING",
    );
    const mappedOp: Status = opStatusMap[op.status] ?? "PROCESSING";
    const final = mappedOp === "FAILED";
    const nextAction =
        mappedOp === "FAILED"
            ? "CONTACT_SUPPORT"
            : mappedOp === "REQUIRES_ACTION"
            ? "PAY_NOW"
            : "NONE";
    return NextResponse.json(
    {
        ok: true,
        status: mappedOp,
        final,
        purchaseId: op.purchaseId ?? purchaseId ?? paymentIntentId,
        paymentIntentId: op.paymentIntentId ?? paymentIntentId,
        code: mappedOp,
        retryable: !final,
        nextAction,
        errorMessage: op.lastError ?? null,
    },
    { status: 200, headers: NO_STORE_HEADERS },
);
}

// -----
// 2) PaymentEvent (telemetria / processamento)
// Regra: NÃO marcamos como PAID apenas por PaymentEvent=OK;
// PAID é SSOT via SaleSummary.
// -----
const eventWhere: Prisma.PaymentEventWhereInput = { OR: [] };
if (purchaseId) eventWhere.OR!.push({ purchaseId });

if (paymentIntentId) {
    eventWhere.OR!.push({ stripePaymentIntentId: paymentIntentId });
    // Compat: alguns flows antigos podem enviar o PI no campo purchaseId.
    eventWhere.OR!.push({ purchaseId: paymentIntentId });
}

const paymentEvent = await prisma.paymentEvent.findFirst({
    where: eventWhere,
    orderBy: { updatedAt: "desc" },
    select: {
        status: true,
        stripePaymentIntentId: true,
        purchaseId: true,
        errorMessage: true,
        updatedAt: true,
    },
});
if (paymentEvent) {
    // Mapeamento conservador: OK != PAID (até existir SaleSummary)
    const statusMap: Record<string, Status> = {
        OK: "PROCESSING",
        PROCESSING: "PROCESSING",
        REQUIRES_ACTION: "REQUIRES_ACTION",
        ERROR: "FAILED",
        FAILED: "FAILED",
        CANCELED: "FAILED",
        CANCELLED: "FAILED",
        REFUNDED: "REFUNDED",
        DISPUTED: "DISPUTED",
    };
    const mapped: Status = statusMap[paymentEvent.status] ?? "PROCESSING";
    const final = FINAL_STATUSES.includes(mapped);

    // Contrato simples para o FE (sem adivinhar)
    const nextAction =
        mapped === "REQUIRES_ACTION" ? "PAY_NOW" : mapped === "FAILED" ? "CONTACT_SUPPORT" : "NONE";

    const retryable =
        mapped === "PENDING" || mapped === "PROCESSING" || mapped === "REQUIRES_ACTION";

    return NextResponse.json(

```

```

        {
          ok: true,
          status: mapped,
          final,
          purchaseId: paymentEvent.purchaseId ?? purchaseId ?? paymentIntentId,
          paymentIntentId: paymentEvent.stripePaymentIntentId ?? paymentIntentId,
          code: mapped,
          retryable,
          nextAction,
          errorMessage: paymentEvent.errorMessage ?? null,
        },
        { status: 200, headers: NO_STORE_HEADERS },
      );
    }

// -----
// 3) Nada encontrado ainda
// -----
return NextResponse.json(
  {
    ok: true,
    status: "PENDING" as Status,
    final: false,
    purchaseId: purchaseId ?? paymentIntentId,
    paymentIntentId,
    code: "PENDING",
    retryable: true,
    nextAction: "NONE",
    errorMessage: null,
  },
  { status: 200, headers: NO_STORE_HEADERS },
);
} catch (err) {
  console.error("[checkout/status] erro inesperado", err);
  return NextResponse.json(
    {
      ok: false,
      status: "FAILED" as Status,
      error: "INTERNAL_ERROR",
      code: "INTERNAL_ERROR",
      retryable: true,
      nextAction: "NONE",
    },
    { status: 500, headers: NO_STORE_HEADERS },
  );
}
}

```

app/api/cron/padel/expire/route.ts

```

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { stripe } from "@/lib/stripeClient";
import {
  Prisma,
  PadelPairingStatus,
  PadelPairingPaymentStatus,
  PadelPairingSlotStatus,
  PadelPairingLifecycleStatus,
  TicketStatus,
} from "@prisma/client";
import { expireHolds } from "@domain/padelPairingHold";
import { computeGraceUntil } from "@domain/padelDeadlines";

// Expira pairings SPLIT com locked_until ultrapassado: cancela slots e tenta refund do capitão.
// Pode ser executado via cron. Não expõe dados sensíveis, mas requer permissão server-side.
export async function POST() {
  const now = new Date();
  await prisma.$transaction((tx) => expireHolds(tx, now));

  // Tentativa de cobrança off-session do capitão (Modelo A) quando deadline expirou e parceiro não pagou
  const chargeable = await prisma.padelPairing.findMany({
    where: {

```

```

        deadlineAt: { lt: now },
        paymentMode: "SPLIT",
        lifecycleStatus: PadelPairingLifecycleStatus.PENDING_PARTNER_PAYMENT,
        guaranteeStatus: { in: ["ARMED", "SCHEDULED"] },
    },
    include: {
        slots: { include: { ticket: true } },
    },
});
};

for (const pairing of chargeable) {
    const paymentMethodId = pairing.paymentMethodId;
    const paidSlot = pairing.slots.find(
        (s) => s.paymentStatus === PadelPairingPaymentStatus.PAID && s.ticket,
    );
    const paidTicket = paidSlot?.ticket;
    const amount =
        (paidTicket?.totalPaidCents ?? 0) > 0
            ? paidTicket!.totalPaidCents
            : paidTicket?.pricePaid ?? 0;
    const currency = (paidTicket?.currency ?? "EUR").toUpperCase();

    if (!paymentMethodId || !amount || amount <= 0) {
        await prisma.padelPairing.update({
            where: { id: pairing.id },
            data: {
                guaranteeStatus: "FAILED",
                lifecycleStatus: "CANCELLED_INCOMPLETE",
                pairingStatus: PadelPairingStatus.CANCELLED,
            },
        });
        continue;
    }

    try {
        const intent = await stripe.paymentIntents.create({
            amount,
            currency,
            paymentMethod: paymentMethodId,
            offSession: true,
            confirm: true,
            metadata: {
                pairingId: pairing.id,
                eventId: pairing.eventId,
                scenario: "GROUP_SPLIT_SECOND_CHARGE",
            },
        });

        if (intent.status === "succeeded") {
            await prisma.$transaction(async (tx) => {
                await tx.padelPairingSlot.updateMany({
                    where: { pairingId: pairing.id, slotStatus: PadelPairingSlotStatus.PENDING },
                    data: { paymentStatus: PadelPairingPaymentStatus.PAID },
                });
                await tx.padelPairing.update({
                    where: { id: pairing.id },
                    data: {
                        lifecycleStatus: PadelPairingLifecycleStatus.CONFIRMED_CAPTAIN_FULL,
                        pairingStatus: PadelPairingStatus.COMPLETE,
                        guaranteeStatus: "SUCCEEDED",
                        secondChargePaymentIntentId: intent.id,
                        captainSecondChargedAt: new Date(),
                        partnerPaidAt: new Date(),
                        graceUntilAt: null,
                    },
                });
                await tx.padelPairingHold.updateMany({
                    where: { pairingId: pairing.id, status: "ACTIVE" },
                    data: { status: "CANCELLED" },
                });
            });
        } else if (intent.status === "requires_action") {
            await prisma.padelPairing.update({
                where: { id: pairing.id },
                data: {
                    guaranteeStatus: "REQUIRES_ACTION",
                    secondChargePaymentIntentId: intent.id,
                    graceUntilAt: computeGraceUntil(now),
                },
            });
        }
    }
}

```

```

        },
    });
} else {
    await prisma.padelPairing.update({
        where: { id: pairing.id },
        data: {
            guaranteeeStatus: "FAILED",
            lifecycleStatus: "CANCELLED_INCOMPLETE",
            pairingStatus: PadelPairingStatus.CANCELLED,
            graceUntilAt: null,
        },
    });
}
} catch (err) {
    console.error("[padel/cron/expire] second charge error", err);
    await prisma.padelPairing.update({
        where: { id: pairing.id },
        data: {
            guaranteeeStatus: "REQUIRES_ACTION",
            graceUntilAt: computeGraceUntil(now),
        },
    });
}
}

// Se REQUIRES_ACTION e graceUntilAt já passou, cancelar pairing e libertar hold
const toCancel = await prisma.padelPairing.findMany({
    where: {
        guaranteeeStatus: "REQUIRES_ACTION",
        graceUntilAt: { lt: now },
        lifecycleStatus: { not: "CANCELLED_INCOMPLETE" },
    },
    select: { id: true },
});
for (const p of toCancel) {
    await prisma.$transaction(async (tx) => {
        await tx.padelPairingSlot.updateMany({
            where: { pairingId: p.id, slotStatus: PadelPairingSlotStatus.PENDING },
            data: { slotStatus: PadelPairingSlotStatus.CANCELLED, paymentStatus: PadelPairingPaymentStatus.UNPAID },
        });
        await tx.padelPairing.update({
            where: { id: p.id },
            data: {
                pairingStatus: PadelPairingStatus.CANCELLED,
                lifecycleStatus: PadelPairingLifecycleStatus.CANCELLED_INCOMPLETE,
                partnerInviteToken: null,
                partnerInviteUsedAt: null,
                partnerLinkToken: null,
                partnerLinkExpiresAt: null,
                guaranteeeStatus: "EXPIRED",
                graceUntilAt: null,
            },
        });
        await tx.padelPairingHold.updateMany({
            where: { pairingId: p.id, status: "ACTIVE" },
            data: { status: "CANCELLED" },
        });
    });
}

const expired = (await prisma.padelPairing.findMany({
    where: {
        paymentMode: "SPLIT",
        pairingStatus: PadelPairingStatus.INCOMPLETE,
        lockedUntil: { lt: now },
    },
    include: {
        slots: {
            include: {
                ticket: true,
            },
        },
    },
})) as Prisma.PadelPairingGetPayload<{
    include: { slots: { include: { ticket: true } } } };
}>[];
let processed = 0;

```

```

for (const pairing of expired) {
  const paidSlot = pairing.slots.find((s) => s.paymentStatus === PadelPairingPaymentStatus.PAID &&
  s.ticket?.stripePaymentIntentId;
  const paidTicket = paidSlot?.ticket ?? null;

  // Refund simplificado: devolve o valor total do PaymentIntent do capitão (share)
  if (paidTicket?.stripePaymentIntentId) {
    try {
      await stripe.refunds.create({
        payment_intent: paidTicket.stripePaymentIntentId,
      });
    } catch (err) {
      console.error("[padel/cron/expire] refund falhou", err);
      // Continua mesmo assim para não bloquear expirations
    }
  }

  await prisma.$transaction(async (tx) => {
    // Cancelar slots e limpar tickets
    await tx.padelPairingSlot.updateMany({
      where: { pairingId: pairing.id },
      data: {
        slotStatus: PadelPairingSlotStatus.CANCELLED,
        paymentStatus: PadelPairingPaymentStatus.UNPAID,
        ticketId: null,
      },
    });

    if (paidTicket) {
      await tx.ticket.update({
        where: { id: paidTicket.id },
        data: { status: TicketStatus.REFUNDED },
      });
    }

    await tx.padelPairing.update({
      where: { id: pairing.id },
      data: {
        pairingStatus: PadelPairingStatus.CANCELLED,
        partnerInviteToken: null,
        partnerInviteUsedAt: null,
        partnerLinkToken: null,
        partnerLinkExpiresAt: null,
        lockedUntil: null,
      },
    });
  });

  processed += 1;
}

return NextResponse.json({ ok: true, processed, now: now.toISOString() });
}

```

app/api/cron/reservations/cleanup/route.ts

```

// app/api/cron/reservations/cleanup/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";

/**
 * ⚠️ IMPORTANT
 * This route is designed to be triggered by Vercel Cron.
 * It must NOT be callable by the public internet.
 * We enforce a secret token header to protect it.
 *
 * Add this header in Vercel Cron:
 *   X-ORYA-CRON-SECRET: <your-secret>
 *
 * And set the env var:
 *   ORYA_CRON_SECRET="your-secret"
 */

export async function GET(req: NextRequest) {

```

```

try {
  const secret = req.headers.get("X-ORYA-CRON-SECRET");

  if (!secret || secret !== process.env.ORYA_CRON_SECRET) {
    return NextResponse.json(
      { ok: false, error: "Unauthorized cron call." },
      { status: 401 }
    );
  }

  const now = new Date();

  // 1) Expire all ACTIVE reservations older than now
  const expired = await prisma.ticketReservation.updateMany({
    where: {
      status: "ACTIVE",
      expiresAt: { lt: now },
    },
    data: {
      status: "EXPIRED",
    },
  });

  // 2) Optional cleanup: remove EXPIRED older than 24h
  const oldExpired = await prisma.ticketReservation.deleteMany({
    where: {
      status: "EXPIRED",
      expiresAt: {
        lt: new Date(now.getTime() - 24 * 60 * 60 * 1000),
      },
    },
  });

  return NextResponse.json({
    ok: true,
    expiredUpdated: expired.count,
    expiredDeleted: oldExpired.count,
    timestamp: now.toISOString(),
  });
} catch (err) {
  console.error("[CRON CLEANUP ERROR]", err);
  return NextResponse.json(
    { ok: false, error: "Internal cleanup error" },
    { status: 500 }
  );
}
}

```

app/api/dev/test-email/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { resend } from "@lib/resend";
import { env } from "@lib/env";

export async function GET(req: NextRequest) {
  if (process.env.NODE_ENV === "production") {
    return NextResponse.json({ ok: false, error: "Route disabled in production." }, { status: 404 });
  }

  const to = req.nextUrl.searchParams.get("to");
  if (!to) {
    return NextResponse.json({ ok: false, error: "Parâmetro 'to' em falta." }, { status: 400 });
  }

  try {
    const result = await resend.emails.send({
      from: env.resendFrom,
      to,
      subject: "Teste de email ORYA (dev)",
      html: `<p>Teste de email ORYA enviado em ${new Date().toISOString()}</p>`,
    });
    return NextResponse.json({ ok: true, result }, { status: 200 });
  } catch (err) {
    console.error("[dev/test-email] error", err);
    return NextResponse.json(
      { ok: false, error: "Falha a enviar email de teste", details: String((err as Error)?.message || err) },
    );
  }
}

```

```
        { status: 500 },
    );
}
}
```

app/api/email/verified/route.ts

```
import { NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { claimIdentity } from "@/lib/ownership/claimIdentity";

// Endpoint para ser chamado pelo frontend após evento de email verificado (Supabase)
export async function POST() {
    const supabase = await createSupabaseServer();
    const { data, error } = await supabase.auth.getUser();
    if (error || !data?.user) {
        return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }
    const email = data.user.email;
    if (!email) {
        return NextResponse.json({ ok: false, error: "EMAIL_MISSING" }, { status: 400 });
    }
    await claimIdentity(email, data.user.id, { requireVerified: true });
    return NextResponse.json({ ok: true });
}
```

app/api/eventos/[slug]/interest/route.ts

```
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

type RouteContext = {
    params: { slug: string } | Promise<{ slug: string }>;
};

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

/**
 * GET /api/eventos/[slug]/interest
 * Devolve:
 * - hasInterest: se o utilizador atual marcou interesse neste evento
 * - total: número total de utilizadores com interesse neste evento
 */
export async function GET(
    _req: NextRequest,
    context: RouteContext,
): Promise<NextResponse> {
    const { slug } = await context.params;

    if (!slug) {
        return NextResponse.json(
            { error: "Slug do evento em falta." },
            { status: 400 },
        );
    }

    const event = await prisma.event.findUnique({
        where: { slug },
        select: { id: true },
    });

    if (!event) {
        return NextResponse.json(
            { error: "Evento não encontrado." },
            { status: 404 },
        );
    }

    // Tentar obter utilizador autenticado (não é obrigatório para GET)
    let userId: string | null = null;
    try {
        const supabase = await createSupabaseServer();
```

```

const { data, error } = await supabase.auth.getUser();

if (!error && data?.user) {
  userId = data.user.id;
}
} catch {
  // Se falhar, tratamos como utilizador não autenticado
  userId = null;
}

const [total, interest] = await Promise.all([
  prisma.eventInterest.count({ where: { eventId: event.id } }),
  userId
    ? prisma.eventInterest.findUnique({
        where: {
          eventId_userId: {
            eventId: event.id,
            userId,
          },
        },
      })
    : Promise.resolve(null),
]);
]

return NextResponse.json({
  hasInterest: Boolean(interest),
  total,
});
}

/**
 * POST /api/eventos/[slug]/interest
 * Faz toggle do interesse do utilizador autenticado neste evento.
 */
export async function POST(
  _req: NextRequest,
  context: RouteContext,
): Promise<NextResponse> {
  const { slug } = await context.params;

  if (!slug) {
    return NextResponse.json(
      { error: "Slug do evento em falta." },
      { status: 400 },
    );
  }

  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    return NextResponse.json(
      { error: "É necessário estar autenticado para marcar interesse." },
      { status: 401 },
    );
  }

  const event = await prisma.event.findUnique({
    where: { slug },
    select: { id: true },
  });

  if (!event) {
    return NextResponse.json(
      { error: "Evento não encontrado." },
      { status: 404 },
    );
  }

  const existing = await prisma.eventInterest.findUnique({
    where: {
      eventId_userId: {
        eventId: event.id,
        userId: user.id,
      },
    },
  });
}

```

```

    },
});

if (existing) {
  // Já tinha interesse → remover (toggle off)
  await prisma.eventInterest.delete({
    where: { id: existing.id },
  });
} else {
  // Não tinha → criar interesse
  await prisma.eventInterest.create({
    data: {
      eventId: event.id,
      userId: user.id,
    },
  });
}

const total = await prisma.eventInterest.count({
  where: { eventId: event.id },
});

return NextResponse.json({
  hasInterest: !existing,
  total,
});
}

```

app/api/eventos/[slug]/resales/route.ts

```

// app/api/eventos/[slug]/resales/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { ResaleStatus } from "@prisma/client";

/**
 * F5-9 – Listar revendas disponíveis por evento
 *
 * GET /api/eventos/[slug]/resales
 *
 * Resposta:
 * {
 *   ok: true,
 *   eventId,
 *   slug,
 *   resales: [
 *     {
 *       id,
 *       ticketId,
 *       price,
 *       currency,
 *       status,
 *       seller: { id, username, fullName } | null,
 *       ticketTypeName
 *     },
 *     ...
 *   ]
 * }
 */

type RouteParams = { slug?: string };

export async function GET(
  _req: NextRequest,
  context: { params: RouteParams | Promise<RouteParams> },
) {
  try {
    const resolved = await context.params;
    const slug = resolved?.slug;

    if (!slug || typeof slug !== "string") {
      return NextResponse.json(
        { ok: false, error: "MISSING_OR_INVALID_SLUG" },
        { status: 400 }
      );
    }
  }
}

```

```

// 1. Buscar evento pelo slug
const event = await prisma.event.findUnique({
  where: { slug },
  select: {
    id: true,
    slug: true,
  },
});

if (!event) {
  return NextResponse.json(
    { ok: false, error: "EVENT_NOT_FOUND" },
    { status: 404 }
  );
}

// 2. Buscar revendas LISTED ligadas a este evento
const resales = await prisma.ticketResale.findMany({
  where: {
    status: ResaleStatus.LISTED,
    ticket: {
      eventId: event.id,
    },
  },
  include: {
    ticket: {
      include: {
        ticketType: {
          select: {
            name: true,
          },
        },
      },
    },
  },
});
}

// 3. Buscar perfis dos sellers (para username / fullName)
const sellerIds = Array.from(
  new Set(resales.map((r) => r.sellerUserId).filter(Boolean))
) as string[];

let sellersMap: Record<
  string,
  { id: string; username: string | null; fullName: string | null }
> = {};

if (sellerIds.length > 0) {
  const sellers = await prisma.profile.findMany({
    where: {
      id: { in: sellerIds },
    },
    select: {
      id: true,
      username: true,
      fullName: true,
    },
  });
}

sellersMap = sellers.reduce<typeof sellersMap>((acc, s) => {
  acc[s.id] = {
    id: s.id,
    username: s.username,
    fullName: s.fullName,
  };
  return acc;
}, {});
}

// 4. Moldar resposta leve para o frontend
const payload = resales.map((r) => ({
  id: r.id,
  ticketId: r.ticketId,
  price: r.price,
  currency: r.currency,
  status: r.status,
  seller: sellersMap[r.sellerUserId] ?? null,
})

```

```

    ticketTypeName: r.ticket?.ticketType?.name ?? null,
  }));
  return NextResponse.json(
    {
      ok: true,
      eventId: event.id,
      slug: event.slug,
      resales: payload,
    },
    { status: 200 }
  );
} catch (error) {
  console.error("Error in GET /api/eventos/[slug]/resales:", error);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 }
  );
}
}
}

```

app/api/eventos/[slug]/revendas/route.ts

```

// Alias em PT para revendas – chama a lógica de /resales
import type { NextRequest } from "next/server";
import { GET as getResales } from "../resales/route";

export async function GET(
  req: NextRequest,
  context: { params: { slug?: string } | Promise<{ slug?: string }> },
) {
  return getResales(req, context);
}

```

app/api/eventos/com-waves/route.ts

```

// app/api/eventos/com-waves/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";

function getWaveStatus(ticket: {
  startsAt: Date | null;
  endsAt: Date | null;
  totalQuantity: number | null;
  soldQuantity: number;
}) {
  const now = new Date();

  // Se tiver limite de stock
  if (
    ticket.totalQuantity !== null &&
    ticket.totalQuantity !== undefined &&
    ticket.soldQuantity >= ticket.totalQuantity
  ) {
    return "sold_out" as const;
  }

  if (ticket.startsAt && now < ticket.startsAt) {
    return "upcoming" as const;
  }

  if (ticket.endsAt && now > ticket.endsAt) {
    return "closed" as const;
  }

  return "on_sale" as const;
}

export async function GET(_req: NextRequest) {
  try {
    const events = await prisma.event.findMany({
      orderBy: {
        startsAt: "asc",
      },
    });
  }
}

```

```

include: {
  ticketTypes: {
    orderBy: {
      sortOrder: "asc",
    },
  },
  organizer: {
    select: {
      displayName: true,
    },
  },
},
});

const payload = events.map((event) => {
  const waves = event.ticketTypes.map((t) => {
    const remaining =
      t.totalQuantity === null || t.totalQuantity === undefined
        ? null // null = ilimitado
        : t.totalQuantity - t.soldQuantity;

    const status = getWaveStatus({
      startsAt: t.startsAt,
      endsAt: t.endsAt,
      totalQuantity: t.totalQuantity,
      soldQuantity: t.soldQuantity,
    });

    const available = status === "on_sale";

    return {
      id: String(t.id),
      name: t.name,
      description: t.description,
      price: t.price,
      currency: t.currency,
      available,
      isVisible: true,
      startsAt: t.startsAt,
      endsAt: t.endsAt,
      totalQuantity: t.totalQuantity,
      soldQuantity: t.soldQuantity,
      remaining,
      status,
    };
  });
}

const basePrice =
  waves.length > 0
    ? waves.reduce<number | null>((min, w) => {
        if (w.price == null) return min;
        if (min == null) return w.price;
        return w.price < min ? w.price : min;
      }, null)
    : null;

return {
  id: event.id,
  slug: event.slug,
  title: event.title,
  description: event.description,
  startDate: event.startsAt,
  endDate: event.endsAt,
  locationName: event.locationName,
  address: event.address,
  isFree: event.isFree,
  basePrice,
  timezone: event.timezone,
  coverImageUrl: event.coverImageUrl,
  organizerName: event.organizer?.displayName ?? null,
  waves,
};
});

return NextResponse.json({ events: payload }, { status: 200 });
} catch (err) {
  console.error("[GET /api/eventos/com-waves]", err);
  return NextResponse.json(

```

```

        { error: "Erro ao carregar eventos." },
        { status: 500 },
    );
}
}

```

app/api/eventos/join/route.ts

```

// app/api/eventos/join/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

function slugify(title: string) {
    return title
        .toLowerCase()
        .normalize("NFD")
        .replace(/[\u0300-\u036f]/g, "")
        .replace(/[^a-z0-9]+/g, "-")
        .replace(/^+|+$|-/g, "");
}

export async function POST(req: NextRequest) {
    try {
        // 1) Garantir que o utilizador está autenticado
        const supabase = createSupabaseServer();
        const {
            data: { user },
            error: supabaseError,
        } = await (await supabase).auth.getUser();

        if (supabaseError) {
            console.error("Erro Supabase em /api/eventos/join:", supabaseError);
            return NextResponse.json(
                { error: "Erro de autenticação." },
                { status: 500 }
            );
        }

        if (!user) {
            return NextResponse.json(
                { error: "Não autenticado." },
                { status: 401 }
            );
        }

        const body = await req.json();

        const {
            title,
            description,
            startDate,
            endDate,
            timezone,
            isFree,
            locationName,
            address,
            ticketPrice,
            coverImage,
        } = body;
    }

    if (!title || !description || !startDate || !endDate || !locationName) {
        return NextResponse.json(
            { error: "Campos obrigatórios em falta." },
            { status: 400 }
        );
    }

    const slugBase = slugify(title);
    let slug = slugBase;
    let counter = 1;

    // garantir slug único

    while (true) {
        const existing = await prisma.event.findUnique({ where: { slug } });

```

```

    if (!existing) break;
    slug = `${slugBase}-${counter++}`;
}

const event = await prisma.event.create({
  data: {
    slug,
    title,
    description,
    startsAt: new Date(startDate),
    endsAt: new Date(endDate),
    timezone: timezone || "Europe/Lisbon",
    isFree: !!isFree,
    locationName,
    address: address || "",
    coverImageUrl:
      coverImage ||
      "https://images.unsplash.com/photo-1541987392829-5937c1069305?q=80&w=1600",
    ownerUserId: user.id,
  },
});

return NextResponse.json({ slug: event.slug }, { status: 201 });
} catch (err) {
  console.error("Erro em /api/eventos/join:", err);
  return NextResponse.json(
    { error: "Erro ao criar evento." },
    { status: 500 }
  );
}
}
}

```

app/api/eventos/list/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { Prisma } from "@prisma/client";

const DEFAULT_PAGE_SIZE = 12;

export async function GET(req: NextRequest) {
  const { searchParams } = new URL(req.url);
  const cursor = searchParams.get("cursor");
  const limitParam = searchParams.get("limit");
  const category = searchParams.get("category");
  const typeFilter = searchParams.get("type"); // all | free | paid
  const search = searchParams.get("q");

  const take = limitParam
    ? Math.min(parseInt(limitParam, 10) || DEFAULT_PAGE_SIZE, 50)
    : DEFAULT_PAGE_SIZE;

  type EventListItem = {
    id: number;
    slug: string;
    title: string;
    shortDescription: string | null;
    startDate: string;
    endDate: string;
    venue: {
      name: string | null;
      address: string | null;
      city: string | null;
      lat: number | null;
      lng: number | null;
    };
    coverImageUrl: string | null;
    isFree: boolean;
    priceFrom: number | null;
    category: string | null;
    tags: string[];
    stats: {
      goingCount: number;
      interestedCount: number;
    };
    wavesSummary: {
      count: number;
      lastEvent: string;
    };
  };

  const events = await prisma.event.findMany({
    where: {
      OR: [
        { title: { contains: search } },
        { slug: { contains: search } },
        { category: category },
        { isFree: true },
        { isFree: false },
        { type: typeFilter },
      ],
      AND: [
        { startsAt: { gt: cursor } },
        { endsAt: { lt: new Date() } },
      ],
    },
    take,
    select: {
      id: true,
      slug: true,
      title: true,
      shortDescription: true,
      startDate: true,
      endDate: true,
      venue: true,
      coverImageUrl: true,
      isFree: true,
      priceFrom: true,
      category: true,
      tags: true,
      stats: true,
      wavesSummary: true,
    },
  });

  return NextResponse.json(events);
}
}

```

```

    totalWaves: number;
    onSaleCount: number;
    soldOutCount: number;
    nextWaveOpensAt: string | null;
};

};

let items: EventListItem[] = [];
let nextCursor: number | null = null;

try {
  const filters: Prisma.EventWhereInput[] = [
    { status: "PUBLISHED" },
    { OR: [{ organizerId: null }, { organizer: { publicListingEnabled: true } }] },
  ];

  if (category && category !== "all") {
    filters.push({ templateType: category.toUpperCase() as Prisma.EventWhereInput["templateType"] });
  }

  if (typeFilter === "free") {
    filters.push({ isFree: true });
  } else if (typeFilter === "paid") {
    filters.push({ isFree: false });
  }

  if (search && search.trim().length > 0) {
    const q = search.trim();
    filters.push({
      OR: [
        { title: { contains: q, mode: "insensitive" } },
        { description: { contains: q, mode: "insensitive" } },
        { locationCity: { contains: q, mode: "insensitive" } },
      ],
    });
  }
}

const cursorId = cursor ? Number(cursor) : null;
if (cursor && Number.isNaN(cursorId)) {
  return NextResponse.json(
    { items: [], pagination: { nextCursor: null, hasMore: false } },
    { status: 400 },
  );
}

const events = await prisma.event.findMany({
  where: { AND: filters },
  orderBy: { startsAt: "asc" },
  take: take + 1, // +1 para sabermos se há mais páginas
  include: {
    ticketTypes: {
      select: {
        price: true,
        totalQuantity: true,
        soldQuantity: true,
        startsAt: true,
        endsAt: true,
        status: true,
      },
    },
    ...(cursorId
      ? {
          skip: 1,
          cursor: { id: cursorId },
        }
      : {}),
  },
});

if (events.length > take) {
  const next = events.pop();
  nextCursor = next?.id ?? null;
}

items = events.map((e) => {
  const priceFrom =
    e.ticketTypes && e.ticketTypes.length > 0
      ? Math.min(...e.ticketTypes.map((t) => t.price ?? 0)) / 100

```

```

    : null;

const onSaleCount = e.ticketTypes?.filter((t) => t.status === "ON_SALE").length ?? 0;
const soldOutCount = e.ticketTypes?.filter((t) => t.status === "SOLD_OUT").length ?? 0;

return {
  id: e.id,
  slug: e.slug,
  title: e.title,
  shortDescription: e.description?.slice(0, 160) ?? null,
  startDate: e.startsAt ? new Date(e.startsAt).toISOString() : "",
  endDate: e.endsAt ? new Date(e.endsAt).toISOString() : "",
  venue: {
    name: e.locationName ?? null,
    address: e.address ?? null,
    city: e.locationCity ?? null,
    lat: e.latitude ?? null,
    lng: e.longitude ?? null,
  },
  coverImageUrl: e.coverImageUrl ?? null,
  isFree: e.isFree,
  priceFrom,
  category: e.templateType ?? null,
  tags: [],
  stats: {
    goingCount: e.ticketTypes?.reduce((sum, t) => sum + t.soldQuantity, 0) ?? 0,
    interestedCount: 0,
  },
  wavesSummary: {
    totalWaves: e.ticketTypes?.length ?? 0,
    onSaleCount,
    soldOutCount,
    nextWaveOpensAt: null,
  },
},
};

} catch (error) {
  console.error("[api/eventos/list] Erro ao carregar eventos, fallback para lista vazia:", error);
  items = [];
  nextCursor = null;
}

return NextResponse.json({
  events: items,
  pagination: {
    nextCursor,
    hasMore: nextCursor !== null,
  },
});
}

```

app/api/eventos/public/route.ts

```

// app/api/eventos/public/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";

export async function GET(_req: NextRequest) {
  try {
    const events = await prisma.event.findMany({
      where: {
        status: "PUBLISHED",
        OR: [{ organizerId: null }, { organizer: { publicListingEnabled: true } }],
      },
      orderBy: { startsAt: "asc" },
    });

    const payload = events.map((event) => {
      const basePriceFrom = event.isFree ? 0 : null;

      return {
        id: event.id,
        slug: event.slug,
        title: event.title,
        description: event.description,
        startDate: event.startsAt,
      };
    });
  }
}

```

```

        endDate: event.endsAt,
        locationName: event.locationName,
        coverImageUrl: event.coverImageUrl,
        isFree: event.isFree,
        basePriceFrom,
    );
};

return NextResponse.json({ events: payload }, { status: 200 });
} catch (err) {
    console.error("[GET /api/eventos/public]", err);
    return NextResponse.json(
        { error: "Erro ao carregar eventos." },
        { status: 500 },
    );
}
}
}

```

app/api/eventos/route.ts

```

// app/api/eventos/route.ts
/* eslint-disable @typescript-eslint/no-explicit-any */
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

function slugify(title: string) {
    return title
        .toLowerCase()
        .trim()
        .normalize("NFD")
        .replace(/[\u0300-\u036f]/g, "")
        .replace(/[^a-z0-9]+/g, "-")
        .replace(/^+|+$|-/g, "");
}

type TicketPayload = {
    name: string;
    price: number;
    available: boolean;
    totalQuantity: number | null;
    startsAt: Date | null;
    endsAt: Date | null;
    isVisible: boolean;
};

export async function POST(req: NextRequest) {
    try {
        // 1) Supabase server - tentar identificar o utilizador que está a criar o evento
        let organizerId: string | null = null;
        try {
            const supabase = await createSupabaseServer();
            const { data: userData, error: userError } = await supabase.auth.getUser();

            if (!userError && userData?.user) {
                organizerId = userData.user.id;
            }
        } catch (authErr) {
            // Não quebrar a criação de evento se houver algum problema com Supabase
            console.warn("[POST /api/eventos] Erro ao obter utilizador Supabase:", authErr);
        }
        // Se não houver utilizador autenticado, não podemos criar o evento
        if (!organizerId) {
            return NextResponse.json(
                { error: "Precisas de iniciar sessão para criar eventos." },
                { status: 401 }
            );
        }
        // 2) Ler body
        const body = await req.json();
    }
}

```

```

const {
  title,
  description,
  startDate,
  endDate,
  timezone,
  isFree,
  locationName,
  address,
  basePrice,
  coverImageUrl,
  organizerName,
  tickets,
} = body;

// 3) Validação básica
if (!title || !description || !startDate || !endDate || !locationName) {
  return NextResponse.json(
    { error: "Campos obrigatórios em falta." },
    { status: 400 }
  );
}

const startDateISO = new Date(startDate);
const endDateISO = new Date(endDate);

if (
  Number.isNaN(startDateISO.getTime()) ||
  Number.isNaN(endDateISO.getTime())
) {
  return NextResponse.json(
    { error: "Datas inválidas." },
    { status: 400 }
  );
}

if (endDateISO <= startDateISO) {
  return NextResponse.json(
    { error: "A data de fim tem de ser posterior à data de início." },
    { status: 400 }
  );
}

// 4) Gerar slug único
const slugBase = slugify(title);
let slug = slugBase;
let counter = 1;

// Garantir slug único

while (true) {
  const existing = await prisma.event.findUnique({ where: { slug } });
  if (!existing) break;
  slug = `${slugBase}-${counter++}`;
}

// 5) Normalizar waves vindas do frontend
const ticketsArray: TicketPayload[] = Array.isArray(tickets)
  ? tickets
    .filter(
      (t: any) =>
        t &&
        typeof t.price !== "undefined" &&
        !Number.isNaN(Number(t.price))
    )
    .map((t: any): TicketPayload => {
      const priceNumber = Math.max(0, Math.round(Number(t.price)));

      const totalQuantity =
        typeof t.totalQuantity === "number" &&
        !Number.isNaN(t.totalQuantity)
        ? t.totalQuantity
        : null;

      const startsAt =
        t.startsAt && typeof t.startsAt === "string"
        ? new Date(t.startsAt)
        : null;
    })
  : [];

```

```

const endsAt =
  t.endsAt && typeof t.endsAt === "string"
    ? new Date(t.endsAt)
    : null;

return {
  name:
    typeof t.name === "string" && t.name.trim().length > 0
      ? t.name.trim()
      : "Bilhete",
  price: priceNumber,
  available:
    typeof t.available === "boolean" ? t.available : true,
  totalQuantity,
  startsAt,
  endsAt,
  isVisible:
    typeof t.isVisible === "boolean" ? t.isVisible : true,
};

})

: [];

const free = Boolean(isFree);

// 6) basePrice final: se for grátils é 0; senão usa basePrice ou o preço do 1º bilhete
const computedBasePrice = free
  ? 0
  : typeof basePrice === "number" && !Number.isNaN(basePrice)
  ? basePrice
  : ticketsArray[0]
  ? ticketsArray[0].price
  : 0;

// 7) Criar evento na DB
const event = await prisma.event.create({
  data: {
    ownerUserId: organizerId,
    slug,
    title,
    description,
    startsAt: startISO,
    endsAt: endISO,
    timezone: timezone || "Europe/Lisbon",
    isFree: free,
    locationName,
    address: address || "",
    coverImageUrl:
      coverImageUrl ||
      "https://images.unsplash.com/photo-1541987392829-5937c1069305?q=80&w=1600",

    // Nota: por agora não associamos organizerId (modelo Organizer)
    ticketTypes:
      !free && ticketsArray.length > 0
      ? {
        create: ticketsArray.map((t, index) => ({
          name: t.name,
          currency: "EUR",
          price: t.price,
          totalQuantity: t.totalQuantity,
          soldQuantity: 0,
          startsAt: t.startsAt,
          endsAt: t.endsAt,
          isVisible: t.isVisible,
          sortOrder: index,
        })),
      }
      : undefined,
  },
});

return NextResponse.json({ slug: event.slug }, { status: 201 });
} catch (err) {
  console.error("[POST /api/eventos]", err);
  return NextResponse.json(
    { error: "Erro ao criar evento." },
    { status: 500 }
  );
}

```

```
}
```

app/api/experiencias/create/route.ts

```
// app/api/experiencias/create/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { Prisma } from "@prisma/client";
import { PORTUGAL_CITIES } from "@/config/cities";

// Tipo esperado no body do pedido
type CreateExperienceBody = {
  title?: string;
  description?: string;
  startsAt?: string;
  endsAt?: string;
  locationName?: string;
  locationCity?: string;
  templateType?: string; // PARTY | SPORT | VOLUNTEERING | TALK | OTHER
  address?: string | null;
  categories?: string[]; // obrigatório pelo negócio
  coverImageUrl?: string | null;
};

function slugify(input: string): string {
  return input
    .toLowerCase()
    .normalize("NFD")
    .replace(/[^a-z0-9]+/g, "-")
    .replace(/^+|-+$|-/g, "");
}

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();

    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json(
        { ok: false, error: "Não autenticado." },
        { status: 401 }
      );
    }

    let body: CreateExperienceBody | null = null;
    try {
      body = (await req.json()) as CreateExperienceBody;
    } catch {
      return NextResponse.json(
        { ok: false, error: "Body inválido." },
        { status: 400 }
      );
    }

    const title = body.title?.trim();
    const description = body.description?.trim() ?? "";
    const startsAtRaw = body.startsAt;
    const endsAtRaw = body.endsAt;
    const locationName = body.locationName?.trim() ?? "";
    const locationCity = body.locationCity?.trim() ?? "";
    const address = body.address?.trim() || null;
    const categoriesInput = Array.isArray(body.categories) ? body.categories : [];
    const coverImageUrl = body.coverImageUrl?.trim()?.() || null;

    if (!title) {
      return NextResponse.json(
        { ok: false, error: "Título é obrigatório." },
        { status: 400 }
      );
    }
  }
}
```

```

if (!startsAtRaw) {
  return NextResponse.json(
    { ok: false, error: "Data/hora de início é obrigatória." },
    { status: 400 }
  );
}

if (!locationCity) {
  return NextResponse.json(
    { ok: false, error: "Cidade é obrigatória." },
    { status: 400 },
  );
}

const cityAllowed = PORTUGAL_CITIES.includes(locationCity as (typeof PORTUGAL_CITIES)[number]);
if (!cityAllowed) {
  return NextResponse.json(
    { ok: false, error: "Cidade inválida. Escolhe uma cidade da lista disponível na ORYA." },
    { status: 400 },
  );
}

// Até termos a tabela de categorias em produção, mapeamos a primeira categoria para templateType fallback.
const allowedCategories = [
  "FESTA",
  "DESPORTO",
  "CONCERTO",
  "PALESTRA",
  "ARTE",
  "COMIDA",
  "DRINKS",
];
;

const categories = categoriesInput
  .map((c) => c.trim().toUpperCase())
  .filter((c) => allowedCategories.includes(c));

const templateFromCategory = (() => {
  const first = categories[0];
  if (first === "FESTA") return "PARTY";
  if (first === "DESPORTO") return "SPORT";
  if (first === "PALESTRA") return "TALK";
  return "OTHER";
})();

const startsAt = new Date(startsAtRaw);
if (Number.isNaN(startsAt.getTime())) {
  return NextResponse.json(
    { ok: false, error: "Data/hora de início inválida." },
    { status: 400 }
  );
}

let endsAtIso: string | undefined = undefined;
if (endsAtRaw) {
  const d = new Date(endsAtRaw);
  if (Number.isNaN(d.getTime())) {
    return NextResponse.json(
      { ok: false, error: "Data/hora de fim inválida." },
      { status: 400 }
    );
  }
  endsAtIso = d.toISOString();
}

// Confirmar que o Profile existe (caso o user tenha contornado onboarding)
const profile = await prisma.profile.findUnique({
  where: { id: user.id },
});

if (!profile) {
  return NextResponse.json(
    {
      ok: false,
      error: "Perfil não encontrado. Completa o onboarding antes de criares experiências.",
    },
    { status: 400 }
  );
}

```

```

    );
}

const baseSlug = slugify(title) || "experiencia";
const randomSuffix = Math.random().toString(36).slice(2, 8);
const slug = `${baseSlug}-${randomSuffix}`;

const templateType = (body.templateType?.toUpperCase() as
  | "PARTY"
  | "SPORT"
  | "VOLUNTEERING"
  | "TALK"
  | "OTHER"
  | undefined) ?? templateFromCategory ?? "OTHER";

// 🚧 Construímos primeiro o objeto bem tipado
const eventData: Prisma.EventCreateInput = {
  slug,
  title,
  description,
  type: "EXPERIENCE",
  templateType,
  ownerUserId: user.id,
  startsAt,
  endsAt: endsAtIso ? new Date(endsAtIso) : startsAt,
  locationName,
  locationCity,
  address,
  isFree: true,
  status: "PUBLISHED",
  coverImageUrl,
};

const event = await prisma.event.create({
  data: eventData,
});

return NextResponse.json(
  {
    ok: true,
    event: {
      id: event.id,
      slug: event.slug,
      title: event.title,
    },
    { status: 201 }
  };
} catch (err) {
  console.error("POST /api/experiencias/create error:", err);
  return NextResponse.json(
    { ok: false, error: "Erro interno ao criar experiência." },
    { status: 500 }
  );
}
}
}

```

app/api/experiencias/join/route.ts

```

// app/api/experiencias/join/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

// Tipo de body esperado para o join
interface JoinExperienceBody {
  eventId?: number | string;
}

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();

    const {

```

```

    data: { user },
    error,
} = await supabase.auth.getUser();

if (error || !user) {
  return NextResponse.json(
    { ok: false, error: "Não autenticado." },
    { status: 401 }
  );
}

let body: JoinExperienceBody | null = null;
try {
  body = (await req.json()) as JoinExperienceBody;
} catch {
  return NextResponse.json(
    { ok: false, error: "Body inválido." },
    { status: 400 }
  );
}

const rawEventId = body?.eventId;

if (rawEventId === undefined || rawEventId === null || rawEventId === "") {
  return NextResponse.json(
    { ok: false, error: "eventId é obrigatório." },
    { status: 400 }
  );
}

const eventIdNumber = typeof rawEventId === "string" ? Number(rawEventId) : rawEventId;

if (!Number.isFinite(eventIdNumber)) {
  return NextResponse.json(
    { ok: false, error: "eventId inválido." },
    { status: 400 }
  );
}

// Confirmar que o evento existe e é uma EXPERIENCE
const event = await prisma.event.findUnique({
  where: { id: eventIdNumber },
  select: {
    id: true,
    type: true,
    status: true,
  },
});

if (!event) {
  return NextResponse.json(
    { ok: false, error: "Experiência não encontrada." },
    { status: 404 }
  );
}

if (event.type !== "EXPERIENCE") {
  return NextResponse.json(
    { ok: false, error: "Só é possível juntar-te a experiências." },
    { status: 400 }
  );
}

if (event.status === "CANCELLED" || event.status === "FINISHED") {
  return NextResponse.json(
    { ok: false, error: "Esta experiência já não aceita participantes." },
    { status: 400 }
  );
}

// Tentar criar o registo de participante, evitando duplicados
try {
  await prisma.experienceParticipant.create({
    data: {
      userId: user.id,
      eventId: event.id,
    },
  });
}

```

```

    } catch (err) {
      // Se já existir (unique constraint), tratamos como sucesso idempotente
      // Podes refinhar isto mais tarde verificando o código de erro específico do Prisma
    }

    const participantsCount = await prisma.experienceParticipant.count({
      where: { eventId: event.id },
    });

    return NextResponse.json(
      {
        ok: true,
        joined: true,
        participantsCount,
      },
      { status: 200 }
    );
  } catch (err) {
    console.error("POST /api/experiencias/join error:", err);
    return NextResponse.json(
      { ok: false, error: "Erro interno ao juntar à experiência." },
      { status: 500 }
    );
  }
}
}

```

app/api/experiencias/join/status/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

export async function GET(req: NextRequest) {
  try {
    const { searchParams } = new URL(req.url);
    const eventIdParam = searchParams.get("eventId");
    const eventId = eventIdParam ? Number(eventIdParam) : null;

    if (!eventId || Number.isNaN(eventId)) {
      return NextResponse.json({ joined: false }, { status: 200 });
    }

    const supabase = await createSupabaseServer();
    const {
      data: { user },
    } = await supabase.auth.getUser();

    if (!user) {
      return NextResponse.json({ joined: false }, { status: 200 });
    }

    const participant = await prisma.experienceParticipant.findUnique({
      where: {
        eventId_userId: {
          eventId,
          userId: user.id,
        },
      },
    });

    return NextResponse.json({ joined: !!participant }, { status: 200 });
  } catch (err) {
    console.error("[experiencias/join/status] error", err);
    return NextResponse.json({ joined: false }, { status: 200 });
  }
}

```

app/api/experiencias/list/route.ts

```

// app/api/experiencias/list/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";

```

```

import { Prisma } from "@prisma/client";

export async function GET(req: NextRequest) {
  try {
    const url = new URL(req.url);
    const searchParams = url.searchParams;

    const cityParam = searchParams.get("city") || undefined;
    const slugParam = searchParams.get("slug") || undefined;
    const searchParam = searchParams.get("search") || undefined;
    const limitParam = searchParams.get("limit");
    const limitNumber = limitParam ? parseInt(limitParam, 10) : 20;
    const take = Number.isNaN(limitNumber)
      ? 20
      : Math.min(Math.max(limitNumber, 1), 50);

    const where: Prisma.EventWhereInput = {
      type: "EXPERIENCE",
      status: "PUBLISHED",
      // Se quiseres só futuras, podes descomentar:
      // startsAt: { gte: new Date() },
    };

    if (cityParam) {
      where.locationCity = {
        contains: cityParam,
        mode: "insensitive",
      };
    }

    if (searchParam) {
      const searchFilter: Prisma.EventWhereInput = {
        OR: [
          {
            title: {
              contains: searchParam,
              mode: "insensitive",
            },
          },
          {
            description: {
              contains: searchParam,
              mode: "insensitive",
            },
          },
          {
            locationName: {
              contains: searchParam,
              mode: "insensitive",
            },
          },
        ],
      };
    }

    // AND passa a ser sempre um array homogéneo de EventWhereInput
    where.AND = [searchFilter];
  }

  const events = await prisma.event.findMany({
    where: slugParam
      ? { ...where, slug: { equals: slugParam, mode: "insensitive" } }
      : where,
    orderBy: { startsAt: "asc" },
    take,
  });

  const ownerIds = Array.from(
    new Set(
      events
        .map((e) => e.ownerUserId)
        .filter((id): id is string => Boolean(id))
    )
  );

  const owners =
    ownerIds.length > 0
      ? await prisma.profile.findMany({
          where: { id: { in: ownerIds } },
        })
      : [];
  }
}

```

```

        select: {
          id: true,
          username: true,
          fullName: true,
        },
      })
    : [];
}

const ownerMap = new Map()
owners.map((o) => [
  o.id,
  {
    id: o.id,
    username: o.username,
    fullName: o.fullName,
  },
])
);

const items = events.map((event) => {
  const owner = ownerMap.get(event.ownerUserId);

  return {
    id: event.id,
    slug: event.slug,
    title: event.title,
    description:
      event.description && event.description.length > 280
        ? event.description.slice(0, 280) + "..."
        : event.description,
    startsAt: event.startsAt,
    locationName: event.locationName,
    locationCity: event.locationCity,
    owner,
  };
});

return NextResponse.json({ items }, { status: 200 });
} catch (err) {
  console.error("GET /api/experiencias/list error:", err);
  return NextResponse.json(
    { items: [], error: "Erro ao carregar experiências." },
    { status: 500 }
  );
}
}
}

```

app/api/explorar/list/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { Prisma, EventTemplateType } from "@prisma/client";

const DEFAULT_PAGE_SIZE = 12;

type ExploreItem = {
  id: number;
  type: "EVENT" | "EXPERIENCE";
  slug: string;
  title: string;
  shortDescription: string | null;
  startsAt: string;
  endsAt: string;
  location: {
    name: string | null;
    city: string | null;
    lat: number | null;
    lng: number | null;
  };
  coverImageUrl: string | null;
  isFree: boolean;
  priceFrom: number | null;
  categories: string[];
  hostName: string | null;
  hostUsername: string | null;
  status: "ACTIVE" | "CANCELLED" | "PAST" | "DRAFT";
}

```

```

    isHighlighted: boolean;
};

type ExploreResponse = {
  items: ExploreItem[];
  pagination: {
    nextCursor: number | null;
    hasMore: boolean;
  };
};

function resolveStatus(event: {
  status: string;
  endsAt: Date | string | null;
  isDeleted?: boolean;
}): ExploreItem["status"] {
  if (event.status === "CANCELLED") return "CANCELLED";
  if (event.status === "DRAFT") return "DRAFT";

  const now = Date.now();
  const endDate =
    event.endsAt instanceof Date
      ? event.endsAt.getTime()
      : event.endsAt
        ? new Date(event.endsAt).getTime()
        : null;

  if (endDate && endDate < now) return "PAST";
  return "ACTIVE";
}

function clampTake(value: number | null): number {
  if (!value || Number.isNaN(value)) return DEFAULT_PAGE_SIZE;
  return Math.min(Math.max(value, 1), 50);
}

export async function GET(req: NextRequest) {
  const { searchParams } = new URL(req.url);

  const typeParam = searchParams.get("type"); // event | experience | all
  const categoriesParam = searchParams.get("categories"); // comma separated
  const cityParam = searchParams.get("city");
  const searchParam = searchParams.get("q");
  const cursorParam = searchParams.get("cursor");
  const limitParam = searchParams.get("limit");
  const priceMinParam = searchParams.get("priceMin");
  const priceMaxParam = searchParams.get("priceMax");
  const dateParam = searchParams.get("date"); // today | upcoming | all | day | weekend
  const dayParam = searchParams.get("day"); // YYYY-MM-DD opcional

  const take = clampTake(limitParam ? parseInt(limitParam, 10) : DEFAULT_PAGE_SIZE);
  const cursorId = cursorParam ? Number(cursorParam) : null;

  const priceMin = priceMinParam ? Math.max(0, parseFloat(priceMinParam)) : 0;
  const priceMaxRaw = priceMaxParam ? parseFloat(priceMaxParam) : null;
  const priceMax = Number.isFinite(priceMaxRaw) ? priceMaxRaw : null;

  const priceMinCents = Math.round(priceMin * 100);
  const priceMaxCents = priceMax !== null ? Math.round(priceMax * 100) : null;

  const categoryFilters = (categoriesParam || "") .split(",") .map((c) => c.trim().toUpperCase()) .filter(Boolean);

  const where: Prisma.EventWhereInput = {
    status: "PUBLISHED",
    isTest: false,
  };

  if (typeParam === "event") {
    where.type = "ORGANIZER_EVENT";
  } else if (typeParam === "experience") {
    where.type = "EXPERIENCE";
  }

  const normalizedCity = cityParam?.trim();
  const applyCityFilter = normalizedCity && normalizedCity.toLowerCase() !== "portugal";
}

```

```

if (applyCityFilter) {
  where.locationCity = {
    contains: normalizedCity,
    mode: "insensitive",
  };
}

if (searchParam) {
  const q = searchParam.trim();
  where.OR = [
    { title: { contains: q, mode: "insensitive" } },
    { description: { contains: q, mode: "insensitive" } },
    { locationName: { contains: q, mode: "insensitive" } },
    { locationCity: { contains: q, mode: "insensitive" } },
  ];
}

// Map categorias pedidas para templateType conhecido (fallback)
if (categoryFilters.length > 0) {
  const mapToTemplate: Record<string, EventTemplateType> = {
    FESTA: "PARTY",
    DESPORTO: "SPORT",
    CONCERTO: "OTHER",
    PALESTRA: "TALK",
    ARTE: "OTHER",
    COMIDA: "OTHER",
    DRINKS: "OTHER",
    GERAL: "OTHER",
  };
  const templateTypes = categoryFilters.map((c) => mapToTemplate[c]).filter((v): v is EventTemplateType => Boolean(v));
}

if (templateTypes.length > 0) {
  where.templateType = { in: templateTypes };
}

if (dateParam === "today") {
  const startOfDay = new Date();
  startOfDay.setHours(0, 0, 0, 0);
  const endOfDay = new Date();
  endOfDay.setHours(23, 59, 59, 999);
  where.startsAt = { gte: startOfDay, lte: endOfDay };
} else if (dateParam === "upcoming") {
  const now = new Date();
  where.startsAt = { gte: now };
} else if (dateParam === "weekend") {
  const now = new Date();
  const day = now.getDay(); // 0 domingo ... 6 sábado
  let start = new Date(now);
  let end = new Date(now);
  if (day === 0) {
    // domingo: só hoje a partir de agora
    start = now;
    end.setHours(23, 59, 59, 999);
  } else {
    const daysToSaturday = (6 - day + 7) % 7;
    start.setDate(now.getDate() + daysToSaturday);
    start.setHours(0, 0, 0, 0);
    end = new Date(start);
    end.setDate(start.getDate() + 1); // domingo
    end.setHours(23, 59, 59, 999);
  }
  where.startsAt = { gte: start, lte: end };
} else if (dateParam === "day" && dayParam) {
  const day = new Date(dayParam);
  if (!Number.isNaN(day.getTime())) {
    const startOfDay = new Date(day);
    startOfDay.setHours(0, 0, 0, 0);
    const endOfDay = new Date(day);
    endOfDay.setHours(23, 59, 59, 999);
    where.startsAt = { gte: startOfDay, lte: endOfDay };
  }
}

// Filtro de preço (priceMax === null significa sem limite superior)
const priceFilters: Prisma.EventWhereInput[] = [];
if (priceMinCents > 0 && priceMaxCents !== null) {

```

```

priceFilters.push({
  isFree: false,
  ticketTypes: {
    some: {
      price: {
        gte: priceMinCents,
        lte: priceMaxCents,
      },
    },
  },
});
} else if (priceMinCents > 0) {
  priceFilters.push({
    isFree: false,
    ticketTypes: {
      some: {
        price: {
          gte: priceMinCents,
        },
      },
    },
  });
} else if (priceMaxCents !== null) {
  priceFilters.push({
    OR: [
      { isFree: true },
      {
        ticketTypes: {
          some: {
            price: {
              lte: priceMaxCents,
            },
          },
        },
      },
    ],
  });
}

if (priceFilters.length > 0) {
  where.AND = Array.isArray(where.AND) ? [...where.AND, ...priceFilters] : priceFilters;
}

const query: Prisma.EventFindManyArgs = {
  where,
  orderBy: [{ startsAt: "asc" }, { id: "asc" }],
  take: take + 1,
  include: {
    ticketTypes: {
      select: {
        price: true,
        status: true,
      },
    },
    organizer: {
      select: {
        displayName: true,
      },
    },
  },
};

if (cursorId) {
  query.skip = 1;
  query.cursor = { id: cursorId };
}

try {
  const events = await prisma.event.findMany(query);

  const ownerIds = Array.from(
    new Set(
      events
        .map((e) => e.ownerUserId)
        .filter((v): v is string => typeof v === "string" && v.length > 0),
    ),
  );
  const owners =

```

```

ownerIds.length > 0
    ? await prisma.profile.findMany({
        where: { id: { in: ownerIds } },
        select: { id: true, username: true, fullName: true },
    })
    : []
);
const ownerMap = new Map(owners.map((o) => [o.id, o]));

let nextCursor: number | null = null;
if (events.length > take) {
    const nextItem = events.pop();
    nextCursor = nextItem?.id ?? null;
}

const items: ExploreItem[] = events.map((event) => {
    const isExperience = event.type === "EXPERIENCE";
    const mappedType: ExploreItem["type"] = isExperience ? "EXPERIENCE" : "EVENT";
    const status = resolveStatus({
        status: event.status,
        endsAt: event.endsAt,
        isDeleted: false,
    });

    const ticketPrices = Array.isArray(event.ticketTypes)
        ? event.ticketTypes
        .map((t) => (typeof t.price === "number" ? t.price : null))
        .filter((p): p is number => p !== null)
        : [];

    let priceFrom: number | null = null;
    if (event.isFree || isExperience) {
        priceFrom = 0;
    } else if (ticketPrices.length > 0) {
        priceFrom = Math.min(...ticketPrices) / 100;
    }

    const ownerProfile = event.ownerUserId ? ownerMap.get(event.ownerUserId) : null;
    const hostName = event.organizer?.displayName ?? ownerProfile?.fullName ?? null;
    const hostUsername = ownerProfile?.username ?? null;

    const templateToCategory: Record<string, string> = {
        PARTY: "FESTA",
        SPORT: "DESPORTO",
        TALK: "PALESTRA",
    };
    const categories =
        event.templateType != null
        ? [templateToCategory[String(event.templateType)] ?? "OUTROS"]
        : [];

    return {
        id: event.id,
        type: mappedType,
        slug: event.slug,
        title: event.title,
        shortDescription: event.description?.slice(0, 200) ?? null,
        startsAt: event.startsAt ? new Date(event.startsAt).toISOString() : "",
        endsAt: event.endsAt ? new Date(event.endsAt).toISOString() : "",
        location: {
            name: event.locationName ?? null,
            city: event.locationCity ?? null,
            lat: event.latitude ?? null,
            lng: event.longitude ?? null,
        },
        coverImageUrl: event.coverImageUrl ?? null,
        isFree: event.isFree || isExperience,
        priceFrom,
        categories,
        hostName,
        hostUsername,
        status,
        isHighlighted: false,
    };
};

return NextResponse.json<ExploreResponse>({
    items,
    pagination: {

```

```

        nextCursor,
        hasMore: nextCursor !== null,
    },
});
} catch (error) {
    console.error("[api/explorar/list] erro:", error);
    // Em caso de erro, devolve lista vazia mas não rebenta o frontend
    return NextResponse.json<ExploreResponse & { error?: string }>(
        {
            items: [],
            pagination: { nextCursor: null, hasMore: false },
            error: error instanceof Error ? error.message : "Erro desconhecido",
        },
        { status: 200 },
    );
}
}

```

app/api/guests/migrate/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

export async function POST(req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error,
        } = await supabase.auth.getUser();
        if (error || !user) {
            return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
        }

        const body = await req.json().catch(() => null) as { email?: string } | null;
        const emailRaw = typeof body?.email === "string" ? body.email.trim().toLowerCase() : null;
        if (!emailRaw) {
            return NextResponse.json({ ok: false, error: "INVALID_EMAIL" }, { status: 400 });
        }

        const userId = user.id;
        const now = new Date();

        const links = await prisma.guestTicketLink.findMany({
            where: { guestEmail: emailRaw },
            select: { ticketId: true },
        });

        if (links.length === 0) {
            return NextResponse.json({ ok: true, migrated: 0 }, { status: 200 });
        }

        await prisma.$transaction(async (tx) => {
            await tx.ticket.updateMany({
                where: { id: { in: links.map((l) => l.ticketId) } },
                data: { userId },
            });
            await tx.guestTicketLink.updateMany({
                where: { guestEmail: emailRaw },
                data: { migratedToUserId: userId, migratedAt: now },
            });
        });

        return NextResponse.json({ ok: true, migrated: links.length }, { status: 200 });
    } catch (err) {
        console.error("[guests/migrate][POST]", err);
        return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
    }
}

```

app/api/me/inscricoes/[id]/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

export async function GET(_req: NextRequest, { params }: { params: { id: string } }) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const entryId = Number(params?.id);
  if (!Number.isFinite(entryId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const entry = await prisma.tournamentEntry.findFirst({
    where: { id: entryId, OR: [{ userId: user.id }, { ownerUserId: user.id }] },
    include: {
      event: { select: { id: true, slug: true, title: true, startsAt: true } },
      pairing: {
        select: {
          id: true,
          payment_mode: true,
          lifecycleStatus: true,
          guaranteeStatus: true,
          slots: {
            select: {
              slot_role: true,
              profileId: true,
              playerProfile: { select: { fullName: true } },
            },
          },
        },
      },
    },
  });
  if (!entry) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

  const pairing = entry.pairing;
  const partnerSlot =
    pairing?.slots?.find((s) => s.slot_role === "PARTNER" && s.profileId !== entry.userId) ||
    pairing?.slots?.find((s) => s.slot_role === "CAPTAIN" && s.profileId !== entry.userId);

  return NextResponse.json(
    {
      ok: true,
      entry: {
        id: entry.id,
        event: entry.event,
        isCaptain: entry.role === "CAPTAIN",
        partnerUserId: partnerSlot?.profileId ?? null,
        partnerGuestName: partnerSlot?.playerProfile?.fullName ?? null,
        badge: pairing?.payment_mode === "SPLIT" ? "SPLIT" : pairing?.payment_mode === "FULL" ? "FULL" : "SINGLE",
        paymentStatusLabel:
          pairing?.lifecycleStatus === "PENDING_PARTNER_PAYMENT"
            ? "A espera do parceiro"
            : pairing?.lifecycleStatus?.startsWith("CONFIRMED")
            ? "Confirmado"
            : "Pendente",
        nextAction:
          pairing?.guaranteeStatus === "REQUIRES_ACTION"
            ? "CONFIRM_GUARANTEE"
            : pairing?.lifecycleStatus === "PENDING_PARTNER_PAYMENT"
            ? "PAY_PARTNER"
            : "NONE",
      },
    },
    { status: 200 },
  );
}

```

app/api/me/inscricoes/route.ts

```

import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";

```

```

import { createSupabaseServer } from "@/lib/supabaseServer";

export async function GET() {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
  }

  const entries = await prisma.tournamentEntry.findMany({
    where: { userId: user.id },
    include: {
      event: { select: { id: true, title: true, slug: true, startsAt: true } },
      pairing: {
        select: {
          id: true,
          payment_mode: true,
          lifecycleStatus: true,
          guaranteeStatus: true,
          partnerInvitedAt: true,
          partnerAcceptedAt: true,
          slots: {
            select: {
              id: true,
              profileId: true,
              slot_role: true,
              ticket: { select: { id: true, status: true } },
            },
          },
        },
      },
    },
  });

  const items = entries.map((entry) => {
    const pairing = entry.pairing;
    const badge =
      pairing?.payment_mode === "SPLIT"
        ? "SPLIT"
        : pairing?.payment_mode === "FULL"
        ? "FULL"
        : "SINGLE";

    let nextAction: "NONE" | "PAY_PARTNER" | "CONFIRM_GUARANTEE" | "VIEW_LIVE" = "NONE";
    if (pairing?.lifecycleStatus === "PENDING_PARTNER_PAYMENT") nextAction = "PAY_PARTNER";
    if (pairing?.guaranteeStatus === "REQUIRES_ACTION") nextAction = "CONFIRM_GUARANTEE";
    if (!pairing && entry.event?.slug) nextAction = "VIEW_LIVE";
    if (nextAction === "NONE" && entry.event?.slug) nextAction = "VIEW_LIVE";

    const paymentStatusLabel =
      pairing?.lifecycleStatus === "PENDING_PARTNER_PAYMENT"
        ? "A espera do parceiro"
        : pairing?.lifecycleStatus?.startsWith("CONFIRMED")
        ? "Confirmado"
        : "Pendente";

    const partnerSlot = pairing?.slots?.find((s) => s.profileId && s.profileId !== entry.userId);
    const liveUrl = entry.event?.slug ? `/torneios/${entry.event.slug}/live${pairing?.id ? `?pairingId=${pairing.id}` : ""}` : null;
    const ctaUrl =
      nextAction === "PAY_PARTNER"
        ? `/pairings/${pairing?.id ?? ""}/checkout`
        : nextAction === "CONFIRM_GUARANTEE"
        ? `/pairings/${pairing?.id ?? ""}/guarantee`
        : nextAction === "VIEW_LIVE" && liveUrl
        ? liveUrl
        : null;

    return {
      id: entry.id,
      event: entry.event
      ? {
          id: entry.event.id,
          title: entry.event.title,
        }
      : null;
    };
  });
}

```

```

        slug: entry.event.slug,
        startsAt: entry.event.startsAt,
    }
    : null,
    isCaptain: entry.role === "CAPTAIN",
    partnerUserId: partnerSlot?.profileId ?? null,
    partnerGuestName: null,
    badge,
    paymentStatusLabel,
    nextAction,
    liveLink: liveUrl,
    pairingId: pairing?.id ?? null,
    ctaUrl,
);
};

return NextResponse.json({ ok: true, items }, { status: 200 });
}

```

app/api/me/notifications/route.ts

```

import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { NextResponse } from "next/server";

export async function GET() {
    const supabase = await createSupabaseServer();
    const {
        data: { user },
    } = await supabase.auth.getUser();

    if (!user) {
        return NextResponse.json({ ok: false, code: "UNAUTHENTICATED" }, { status: 401 });
    }

    const notifications = await prisma.notification.findMany({
        where: { userId: user.id },
        orderBy: { createdAt: "desc" },
        take: 100,
    });

    const unreadCount = notifications.filter((n) => !n.isRead).length;

    return NextResponse.json({ ok: true, unreadCount, notifications });
}

```

app/api/me/purchases/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

type PurchaseStatus = "PAID" | "PROCESSING" | "REFUNDED" | "DISPUTED" | "FAILED";

export async function GET(req: NextRequest) {
    const supabase = await createSupabaseServer();
    const { data, error } = await supabase.auth.getUser();
    if (error || !data?.user) {
        return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }
    const userId = data.user.id;

    const url = new URL(req.url);
    const includeFree = url.searchParams.get("includeFree") === "true";
    const statusFilter = url.searchParams.get("status");
    const cursorParam = url.searchParams.get("cursor");
    const cursorDate = cursorParam ? new Date(cursorParam) : null;
    const limit = Math.min(Number(url.searchParams.get("limit") ?? 50), 200);

    // identities for owner lookup
    const identities = await prisma.emailIdentity.findMany({
        where: { userId },
        select: { id: true },
    });
}

```

```

const identityIds = identities.map((i) => i.id);

const summaries = await prisma.saleSummary.findMany({
  where: {
    OR: [
      { userId },
      { ownerUserId: userId },
      { identityIds.length ? { ownerId: { in: identityIds } } : undefined,
        .filter(Boolean) as object[] },
      ...(includeFree ? {} : { totalCents: { gt: 0 } }),
      ...(cursorDate && !Number.isNaN(cursorDate.getTime()) ? { createdAt: { lt: cursorDate } } : {}),
    ],
    orderBy: { createdAt: "desc" },
    take: limit + 1,
    select: {
      id: true,
      purchaseId: true,
      paymentIntentId: true,
      totalCents: true,
      currency: true,
      createdAt: true,
      updatedAt: true,
      promoCodeSnapshot: true,
      promoLabelSnapshot: true,
      feeMode: true,
      saleLines: {
        select: {
          id: true,
          event: { select: { title: true, slug: true } },
          ticketType: { select: { name: true } },
        },
      },
      paymentEvents: {
        orderBy: { createdAt: "asc" },
        select: {
          id: true,
          status: true,
          createdAt: true,
          errorMessage: true,
          source: true,
        },
      },
    },
  },
});

const items = summaries.slice(0, limit).map((s) => {
  const timeline =
    s.paymentEvents?.map((e) => ({
      id: e.id,
      status: e.status,
      createdAt: e.createdAt,
      source: e.source,
      errorMessage: e.errorMessage,
    })) ?? [];
  const status: PurchaseStatus = timeline.some((t) => t.status === "DISPUTED")
    ? "DISPUTED"
    : timeline.some((t) => t.status === "REFUNDED")
    ? "REFUNDED"
    : timeline.some((t) => t.status === "ERROR" || t.status === "FAILED")
    ? "FAILED"
    : timeline.some((t) => t.status === "OK")
    ? "PAID"
    : "PROCESSING";

  const badge = s.totalCents === 0 ? "FREE" : "SINGLE";

  return {
    id: s.id,
    purchaseId: s.purchaseId,
    totalCents: s.totalCents,
    currency: s.currency,
    createdAt: s.createdAt,
    promoCode: s.promoCodeSnapshot,
    promoLabel: s.promoLabelSnapshot,
    feeMode: s.feeMode,
    badge,
    status,
    timeline,
  };
});

```

```

    lines: s.saleLines.map((l) => ({
      id: l.id,
      eventTitle: l.event?.title ?? "",
      eventSlug: l.event?.slug ?? "",
      ticketTypeName: l.ticketType?.name ?? "",
    })),
  );
});

const nextCursor = summaries.length > limit ? summaries[limit].createdAt.toISOString() : null;

// Optional status filtering after mapping (for simplicity)
const filtered =
  statusFilter && ["PAID", "PROCESSING", "REFUNDED", "DISPUTED", "FAILED"].includes(statusFilter)
    ? items.filter((i) => i.status === statusFilter)
    : items;

return NextResponse.json({ ok: true, items: filtered, nextCursor }, { status: 200 });
}

```

app/api/me/route.ts

```

// app/api/me/route.ts
import { NextResponse } from "next/server";
import { createSupabaseServer } from "@lib/supabaseServer";

// Tipagem simples devolvida ao frontend
interface SupabaseUser {
  id: string;
  email?: string | null;
}

interface AuthErrorLike {
  status?: number;
  name?: string;
}

export async function GET() {
  try {
    const supabase = await createSupabaseServer();

    const {
      data: { user },
      error: userError,
    } = await supabase.auth.getUser();

    // ♦ Caso típico: sem sessão → devolver 401 sem lançar 500
    if (userError) {
      const err = userError as AuthErrorLike;
      const isAuthMissing =
        err?.status === 400 ||
        err?.name === "AuthSessionMissingError";

      if (isAuthMissing) {
        return NextResponse.json(
          { success: false, error: "Precisas de iniciar sessão." },
          { status: 401 },
        );
      }
    }

    // Outros erros
    console.warn("[GET /api/me] Erro inesperado em getUser:", userError);
    return NextResponse.json(
      { success: false, error: "Erro ao obter sessão." },
      { status: 500 },
    );
  }

  // ♦ Caso sem user (sem sessão válida)
  if (!user) {
    return NextResponse.json(
      { success: false, error: "Precisas de iniciar sessão." },
      { status: 401 },
    );
  }
}

```

```

// ♦ User válido - devolvemos apenas os campos necessários
const safeUser: SupabaseUser = {
  id: user.id,
  email: user.email ?? undefined,
};

// Fetch profile from app_v3.profiles
const { data: profile, error: profileError } = await supabase
  .from("profiles")
  .select("*")
  .eq("id", user.id)
  .single();

if (profileError) {
  console.warn("[GET /api/me] Erro ao carregar profile:", profileError);
}

return NextResponse.json({
  success: true,
  user: safeUser,
  profile: profile ?? null,
});
} catch (err) {
  console.error("[GET /api/me] Erro inesperado:", err);
  return NextResponse.json(
    { success: false, error: "Erro ao carregar o perfil." },
    { status: 500 },
  );
}
}
}

```

app/api/me/settings/delete/cancel/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { logAccountEvent } from "@/lib/accountEvents";

export async function POST(_req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const profile = await prisma.profile.findUnique({
      where: { id: user.id },
      select: { status: true, deletionScheduledFor: true },
    });
    if (!profile) {
      return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
    }

    if (profile.status !== "PENDING_DELETE") {
      return NextResponse.json({ ok: false, error: "NOT_PENDING_DELETE" }, { status: 400 });
    }

    const now = new Date();
    if (profile.deletionScheduledFor && profile.deletionScheduledFor <= now) {
      return NextResponse.json({ ok: false, error: "DELETION_LOCKED" }, { status: 409 });
    }

    await prisma.profile.update({
      where: { id: user.id },
      data: {
        status: "ACTIVE",
        deletionRequestedAt: null,
        deletionScheduledFor: null,
        deletedAtFinal: null,
      },
    });
  }
}

```

```

    await logAccountEvent({
      userId: user.id,
      type: "account_delete_cancelled",
    });

    return NextResponse.json({ ok: true, status: "ACTIVE" }, { status: 200 });
  } catch (err) {
    console.error("[settings/delete/cancel] erro:", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
  }
}

```

app/api/me/settings/delete/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { supabaseAdmin } from "@/lib/supabaseAdmin";
import { clearUsernameForOwner } from "@/lib/globalUsernames";
import { logAccountEvent } from "@/lib/accountEvents";

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "Não autenticado." }, { status: 401 });
    }

    // Verificar se o utilizador é owner único de alguma organização
    const ownerMemberships = await prisma.organizerMember.findMany({
      where: { userId: user.id, role: "OWNER" },
      include: { organizer: true },
    });

    const blockedOrgs: string[] = [];
    for (const mem of ownerMemberships) {
      if (!mem.organizer) continue;
      const otherOwners = await prisma.organizerMember.count({
        where: {
          organizerId: mem.organizerId,
          role: "OWNER",
          userId: { not: user.id },
        },
      });
      if (otherOwners === 0) {
        blockedOrgs.push(mem.organizer.displayName || mem.organizer.businessName || `Organização #${mem.organizerId}`);
      }
    }

    if (blockedOrgs.length > 0) {
      return NextResponse.json(
        {
          ok: false,
          error:
            "Ainda é o único proprietário destas organizações. Transfere a propriedade ou apaga-as antes de eliminar a conta.",
          organizations: blockedOrgs,
        },
        { status: 400 },
      );
    }

    const now = new Date();
    const scheduled = new Date(now.getTime() + 30 * 24 * 60 * 60 * 1000);

    await prisma.profile.update({
      where: { id: user.id },
      data: {
        status: "PENDING_DELETE",
        deletionRequestedAt: now,
        deletionScheduledFor: scheduled,
      },
    });
  }
}

```

```

        visibility: "PRIVATE",
    },
});

await logAccountEvent({
    userId: user.id,
    type: "account_delete_requested",
    metadata: { scheduledFor: scheduled },
});

await supabase.auth.signOut();

return NextResponse.json({
    ok: true,
    status: "PENDING_DELETE",
    scheduledFor: scheduled.toISOString(),
    message: "Conta marcada para eliminação. Podes reverter dentro do prazo ao voltar a iniciar sessão.",
});
} catch (err) {
    console.error("[settings/delete] erro:", err);
    return NextResponse.json(
        {
            ok: true,
            status: "ERROR",
            warning: "Não foi possível marcar para eliminação. Tenta novamente mais tarde.",
        },
        { status: 200 },
    );
}
}
}

```

app/api/me/settings/email/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";

const EMAIL_REGEX = /^[^\s@]+@[^\s@]+\.[^\s@]+$/i;

type Body = {
    email?: string;
};

export async function PATCH(req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error,
        } = await supabase.auth.getUser();

        if (error || !user) {
            return NextResponse.json({ ok: false, error: "Não autenticado." }, { status: 401 });
        }

        let body: Body;
        try {
            body = (await req.json()) as Body;
        } catch {
            return NextResponse.json({ ok: false, error: "Body inválido." }, { status: 400 });
        }

        const newEmail = body.email?.trim().toLowerCase();
        if (!newEmail) {
            return NextResponse.json({ ok: false, error: "Email obrigatório." }, { status: 400 });
        }
        if (!EMAIL_REGEX.test(newEmail)) {
            return NextResponse.json(
                { ok: false, error: "Email em formato inválido.", code: "INVALID_EMAIL" },
                { status: 400 },
            );
        }

        const { data, error: updateError } = await supabase.auth.updateUser({ email: newEmail });
        if (updateError) {
            console.error("[settings/email] update error:", updateError);
        }
    }
}

```

```

    return NextResponse.json(
      { ok: false, error: updateError.message || "Erro ao atualizar email." },
      { status: 400 },
    );
  }

  // Não guardamos email na Profile (a fonte de verdade é o Supabase), mas
  // tocamos no updatedAt para manter o perfil "vivo".
  await prisma.profile.upsert({
    where: { id: user.id },
    update: { updatedAt: new Date() },
    create: {
      id: user.id,
      roles: ["user"],
      favouriteCategories: [],
      onboardingDone: false,
    },
  });

  return NextResponse.json({
    ok: true,
    user: { id: user.id, email: data.user?.email ?? newEmail },
    message: "Email atualizado. Confirmação pode ser necessária.",
  });
} catch (err) {
  console.error("[settings/email] erro:", err);
  return NextResponse.json({ ok: false, error: "Erro a atualizar email." }, { status: 500 });
}
}

```

app/api/me/settings/save/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

type Body = {
  visibility?: "PUBLIC" | "PRIVATE";
  allowEmailNotifications?: boolean;
  allowEventReminders?: boolean;
  allowFriendRequests?: boolean;
  allowSalesAlerts?: boolean;
  allowSystemAnnouncements?: boolean;
  hardDelete?: boolean;
};

export async function PATCH(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "Não autenticado." }, { status: 401 });
    }

    let body: Body;
    try {
      body = (await req.json()) as Body;
    } catch {
      return NextResponse.json({ ok: false, error: "Body inválido." }, { status: 400 });
    }

    const visibility = body.visibility;
    const wantsHardDelete = body.hardDelete === true;
    if (visibility && visibility !== "PUBLIC" && visibility !== "PRIVATE") {
      return NextResponse.json({ ok: false, error: "Visibilidade inválida." }, { status: 400 });
    }

    const dataToUpdate: Record<string, unknown> = {};
    if (visibility) dataToUpdate.visibility = visibility;
    if (typeof body.allowEmailNotifications === "boolean") {
      dataToUpdate.allowEmailNotifications = body.allowEmailNotifications;
    }
  }
}

```

```

if (typeof body.allowEventReminders === "boolean") {
  dataToUpdate.allowEventReminders = body.allowEventReminders;
}
if (typeof body.allowFriendRequests === "boolean") {
  dataToUpdate.allowFriendRequests = body.allowFriendRequests;
}

if (wantsHardDelete) {
  // Libera username e marca soft-delete
  const existing = await prisma.profile.findUnique({ where: { id: user.id }, select: { username: true } });
  if (existing?.username) {
    await prisma.profile.update({
      where: { id: user.id },
      data: {
        username: null,
        isDeleted: true,
        deletedAt: new Date(),
        fullName: "Conta apagada",
        bio: null,
        city: null,
        avatarUrl: null,
      },
    });
  } else {
    await prisma.profile.update({
      where: { id: user.id },
      data: {
        isDeleted: true,
        deletedAt: new Date(),
        fullName: "Conta apagada",
        bio: null,
        city: null,
        avatarUrl: null,
      },
    });
  }
  return NextResponse.json({ ok: true, deleted: true });
}

const profile = await prisma.profile.upsert({
  where: { id: user.id },
  update: dataToUpdate,
  create: {
    id: user.id,
    roles: ["user"],
    favouriteCategories: [],
    onboardingDone: false,
    visibility: dataToUpdate.visibility === "PRIVATE" ? "PRIVATE" : "PUBLIC",
    allowEmailNotifications: Boolean(dataToUpdate.allowEmailNotifications ?? true),
    allowEventReminders: Boolean(dataToUpdate.allowEventReminders ?? true),
    allowFriendRequests: Boolean(dataToUpdate.allowFriendRequests ?? true),
  },
});

// Sincronizar prefs de notificação (notification_preferences)
const notificationPrefsUpdate: Record<string, boolean> = {};
const assign = (key: keyof Body, target: string) => {
  if (typeof body[key] === "boolean") notificationPrefsUpdate[target] = body[key] as boolean;
};

assign("allowEmailNotifications", "allowEmailNotifications");
assign("allowEventReminders", "allowEventReminders");
assign("allowFriendRequests", "allowFriendRequests");
assign("allowSalesAlerts", "allowSalesAlerts");
assign("allowSystemAnnouncements", "allowSystemAnnouncements");
if (Object.keys(notificationPrefsUpdate).length > 0) {
  await prisma.notificationPreference.upsert({
    where: { userId: user.id },
    update: notificationPrefsUpdate,
    create: { userId: user.id, ...notificationPrefsUpdate },
  });
}

return NextResponse.json({
  ok: true,
  profile: {
    visibility: profile.visibility,
    allowEmailNotifications: profile.allowEmailNotifications,
    allowEventReminders: profile.allowEventReminders,
  }
});

```

```

        allowFriendRequests: profile.allowFriendRequests,
        allowSalesAlerts:
          typeof body.allowSalesAlerts === "boolean"
            ? body.allowSalesAlerts
            : undefined,
        allowSystemAnnouncements:
          typeof body.allowSystemAnnouncements === "boolean"
            ? body.allowSystemAnnouncements
            : undefined,
      },
    )),
  } catch (err) {
  console.error("[settings/save] erro:", err);
  return NextResponse.json({ ok: false, error: "Erro a guardar definições." }, { status: 500 });
}
}

```

app/api/me/tickets/route.ts

```

// app/api/me/tickets/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@lib/prisma";
import { createSupabaseServer } from "@lib/supabaseServer";

// DTO usado pelo frontend
export type UserTicket = {
  id: string;
  quantity: number; // sempre 1 (um registo por bilhete)
  pricePaid: number; // líquido em cêntimos
  grossCents?: number;
  discountCents?: number;
  platformFeeCents?: number;
  netCents?: number;
  currency: string; // ex: EUR
  purchasedAt: string;
  badge: "FREE" | "RESALE" | "SPLIT" | "FULL" | "SINGLE";
  nextAction: "NONE" | "PAY_PARTNER" | "CONFIRM_GUARANTEE";
  purchaseId: string | null;
  isTournament?: boolean;

  qrToken: string | null;

  event: {
    id: number;
    slug: string;
    title: string;
    startDate: string;
    endDate: string;
    locationName: string;
    coverImageUrl: string | null;
    resaleMode?: "ALWAYS" | "AFTER_SOLD_OUT" | "DISABLED";
    isSoldOut?: boolean;
  };

  ticket: {
    id: string;
    name: string;
    description: string | null;
  };
};

export async function GET(req: NextRequest) {
  try {
    const url = new URL(req.url);
    const limit = Math.min(Number(url.searchParams.get("limit")) ?? 100), 500;
    const cursorParam = url.searchParams.get("cursor");
    const cursorDate = cursorParam ? new Date(cursorParam) : null;

    // 1) Obter utilizador autenticado via Supabase (cookies de server)
    const supabase = await createSupabaseServer();
    const { data: userData, error: userError } = await supabase.auth.getUser();

    if (userError || !userData?.user) {
      return NextResponse.json(
        { success: false, error: "Not authenticated", tickets: [] },
        { status: 401 }
      );
    }

    const { data: events, error: eventsError } = await prisma.event.findMany({
      take: limit,
      skip: cursorDate ? cursorDate : null,
      order: { id: "desc" },
    });

    const tickets = events.map((event) => {
      const ticket = new UserTicket({
        event,
        ticket: new UserTicket({
          id: event.id,
          name: event.name,
          description: event.description,
        }),
      });
      return ticket;
    });

    const response = NextResponse.json({
      success: true,
      tickets: tickets,
    });
    response.setCookie(next/cookies, { name: "ticket", value: JSON.stringify(tickets) });
    return response;
  }
}

```

```

    );
}

const userId = userData.user.id;

const identityIds: string[] = [];
// Se existir o modelo EmailIdentity (ambientes com migrations novas), usar para ownership guest.
if ((prisma as any).emailIdentity) {
  try {
    const identities = await (prisma as any).emailIdentity.findMany({
      where: { userId },
      select: { id: true },
    });
    identityIds.push(...identities.map((i: { id: string }) => i.id));
  } catch (err) {
    console.warn("[/api/me/tickets] emailIdentity lookup falhou, ignorando.", err);
  }
}

// 2) Buscar bilhetes desse utilizador
const userTickets = await prisma.ticket.findMany({
  where: {
    OR: [
      { userId },
      { ownerUserId: userId },
      identityIds.length ? { ownerIdentityId: { in: identityIds } } : undefined,
    ].filter(Boolean) as object[],
    ...(cursorDate && !Number.isNaN(cursorDate.getTime())
      ? { purchasedAt: { lt: cursorDate } }
      : {}),
  },
  orderBy: { purchasedAt: "desc" },
  take: limit + 1,
  select: {
    id: true,
    eventId: true,
    ticketTypeId: true,
    purchasedAt: true,
    pricePaid: true,
    totalPaidCents: true,
    currency: true,
    qrSecret: true,
    platformFeeCents: true,
    stripePaymentIntentId: true,
    status: true,
    ownerUserId: true,
    ownerIdentityId: true,
    padelSplitShareCents: true,
    tournamentEntryId: true,
    event: {
      select: {
        id: true,
        slug: true,
        title: true,
        startsAt: true,
        endsAt: true,
        locationName: true,
        coverImageUrl: true,
        resaleMode: true,
        ticketTypes: { select: { totalQuantity: true, soldQuantity: true } },
      },
    },
    ticketType: { select: { id: true, name: true, description: true } },
    pairing: {
      select: {
        paymentMode: true,
        lifecycleStatus: true,
        guaranteeStatus: true,
        deadlineAt: true,
        graceUntilAt: true,
      },
    },
  },
});

const hasMore = userTickets.length > limit;
const trimmedTickets = hasMore ? userTickets.slice(0, limit) : userTickets;

```

```

// Buscar sale_summaries/lines por PaymentIntent para preencher breakdown
const paymentIntentIds = Array.from(
  new Set(
    trimmedTickets
      .map((t) => t.stripePaymentIntentId)
      .filter((id): id is string => Boolean(id))
  )
);

const saleSummaries = paymentIntentIds.length
  ? await prisma.saleSummary.findMany({
      where: { paymentIntentId: { in: paymentIntentIds } },
      select: {
        paymentIntentId: true,
        subtotalCents: true,
        discountCents: true,
        platformFeeCents: true,
        netCents: true,
        promoLabelSnapshot: true,
        promoCodeSnapshot: true,
        lines: {
          select: {
            ticketTypeId: true,
            quantity: true,
            grossCents: true,
            netCents: true,
            discountPerUnitCents: true,
            platformFeeCents: true,
          },
        },
      },
    })
  : [];

const summaryMap = new Map<string, (typeof saleSummaries)[number]>();
saleSummaries.forEach((s) => summaryMap.set(s.paymentIntentId, s));

// Pré-calcular sold out por evento para suportar o modo AFTER_SOLD_OUT
const eventSoldOutMap = new Map<number, boolean>();

const isEventSoldOut = (
  eventId: number,
  ticketTypes: { totalQuantity: number | null | undefined; soldQuantity: number }[],
) => {
  if (eventSoldOutMap.has(eventId)) return eventSoldOutMap.get(eventId)!;

  const list = ticketTypes ?? [];
  const hasUnlimited = list.some(
    (tt) => tt.totalQuantity === null || tt.totalQuantity === undefined,
  );

  const soldOut =
    !hasUnlimited &&
    list.length > 0 &&
    list.every((tt) => {
      if (tt.totalQuantity === null || tt.totalQuantity === undefined)
        return false;
      return tt.soldQuantity >= tt.totalQuantity;
    });

  eventSoldOutMap.set(eventId, soldOut);
  return soldOut;
};

// 3) Devolver 1 registo por bilhete (sem agrupar) para não perder QR únicos
const tickets: UserTicket[] = trimmedTickets
  .filter((t) => t.event && t.ticketType)
  .map((t) => {
    const event = t.event!;
    const ticketType = t.ticketType!;
    const pairing = t.pairing ?? null;
    const resaleMode =
      (event as { resaleMode?: "ALWAYS" | "AFTER SOLD OUT" | "DISABLED" | null }).resaleMode ??
      "ALWAYS";
    const ticketTypesForSoldOut =
      Array.isArray(event.ticketTypes) && event.ticketTypes.length > 0
        ? event.ticketTypes.map((tt) => ({
            totalQuantity: tt.totalQuantity,
            soldQuantity: tt.soldQuantity,
          }))
        : [
            {
              totalQuantity: 1,
              soldQuantity: 1,
            },
          ];
    const ticket = {
      event,
      ticketType,
      pairing,
      resaleMode,
      ticketTypes: ticketTypesForSoldOut,
    };
    if (pairing) {
      ticket.pairing = pairing;
    }
    return ticket;
  });

```

```

        : [];
    const soldOut = isEventSoldOut(event.id, ticketTypesForSoldOut);
    let grossCents = t.pricePaid ?? 0;
    let discountCents = 0;
    let feeCents = t.platformFeeCents ?? 0;
    let netCents = t.pricePaid ?? 0;

    if (t.stripePaymentIntentId) {
        const summary = summaryMap.get(t.stripePaymentIntentId);
        if (summary) {
            const line = summary.lines.find((l) => l.ticketTypeId === t.ticketTypeId);
            if (line && (line.quantity ?? 0) > 0) {
                const qty = line.quantity || 1;
                grossCents = Math.round(line.grossCents / qty);
                netCents = Math.round(line.netCents / qty);
                feeCents = Math.round((line.platformFeeCents ?? 0) / qty);
                discountCents = line.discountPerUnitCents ?? Math.max(0, grossCents - netCents - feeCents);
            } else {
                // fallback proporcional simples: usar net/fees do summary quando não há line match
                grossCents = summary.subtotalCents;
                discountCents = summary.discountCents;
                feeCents = summary.platformFeeCents;
                netCents = summary.netCents;
            }
        }
    }
    const summaryPromo =
        (t.stripePaymentIntentId ? summaryMap.get(t.stripePaymentIntentId)?.promoLabelSnapshot : null) ??
        (t.stripePaymentIntentId ? summaryMap.get(t.stripePaymentIntentId)?.promoCodeSnapshot : null) ??
        null;
    const summaryPurchaseId =
        t.stripePaymentIntentId ? summaryMap.get(t.stripePaymentIntentId)?.paymentIntentId : null;

    let badge: UserTicket["badge"] = "SINGLE";
    if ((t.totalPaidCents ?? t.pricePaid ?? 0) === 0) badge = "FREE";
    if (pairing?.payment_mode === "SPLIT") badge = "SPLIT";
    if (pairing?.payment_mode === "FULL") badge = "FULL";

    let nextAction: UserTicket["nextAction"] = "NONE";
    if (pairing?.lifecycleStatus === "PENDING_PARTNER_PAYMENT") nextAction = "PAY_PARTNER";
    if (pairing?.guaranteeStatus === "REQUIRES_ACTION") nextAction = "CONFIRM_GUARANTEE";

    return {
        id: t.id,
        quantity: 1,
        pricePaid: t.pricePaid ?? 0,
        grossCents,
        discountCents,
        platformFeeCents: feeCents,
        netCents,
        currency: t.currency ?? "EUR",
        promoLabel: summaryPromo,
        purchasedAt: t.purchasedAt.toISOString(),
        purchaseId: summaryPurchaseId ?? t.stripePaymentIntentId ?? null,
        badge,
        nextAction,
        isTournament: Boolean((t as { tournamentEntryId?: number | null }).tournamentEntryId),
        qrToken: t.qrSecret ?? null,
        event: {
            id: event.id,
            slug: event.slug,
            title: event.title,
            startDate: event.startsAt?.toISOString()?. ?? "",
            endDate: event.endsAt?.toISOString()?. ?? "",
            locationName: event.locationName ?? "",
            coverImageUrl: event.coverImageUrl ?? null,
            resaleMode,
            isSoldOut: soldOut,
        },
        ticket: {
            id: String(ticketType.id),
            name: ticketType.name ?? "Bilhete",
            description: ticketType.description ?? null,
        },
    };
}

return NextResponse.json()

```

```

        {
          success: true,
          tickets,
          nextCursor: hasMore ? tickets[tickets.length - 1]?.purchasedAt ?? null : null,
        },
        { status: 200 }
      );
    } catch (err) {
      console.error("[GET /api/me/tickets] Erro inesperado:", err);
      return NextResponse.json(
        {
          success: false,
          error: "Erro ao carregar bilhetes do utilizador.",
          tickets: [],
        },
        { status: 500 }
      );
    }
  }
}

```

app/api/me/tickets/transfers/route.ts

```

// app/api/me/tickets/transfers/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

/**
 * F5-4 – Listar transferências do utilizador
 *
 * GET /api/me/tickets/transfers
 *
 * Resposta:
 * {
 *   ok: true,
 *   incoming: [...], // transferências onde o user é o destino (toUserId)
 *   outgoing: [...], // transferências onde o user é o emissor (fromUserId)
 * }
 */
export async function GET(_req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error: authError,
    } = await supabase.auth.getUser();

    if (authError) {
      console.error(
        "Error getting user in /api/me/tickets/transfers:",
        authError
      );
    }

    if (!user) {
      return NextResponse.json(
        { ok: false, error: "UNAUTHENTICATED" },
        { status: 401 }
      );
    }

    const userId = user.id;

    // Buscar todas as transferências onde este user é from OU to
    const transfers = await prisma.ticketTransfer.findMany({
      where: {
        OR: [{ fromUserId: userId }, { toUserId: userId }],
      },
      include: {
        ticket: {
          include: {
            event: true,
            ticketType: true,
          },
        },
      },
    });
  }
}

```

```

    },
    orderBy: {
      createdAt: "desc",
    },
  });
}

// Separar em incoming vs outgoing
const incoming = transfers
  .filter((t) => t.toUserId === userId)
  .map((t) => ({
    id: t.id,
    status: t.status,
    createdAt: t.createdAt,
    completedAt: t.completedAt,
    fromUserId: t.fromUserId,
    toUserId: t.toUserId,
    ticket: t.ticket
    ? {
      id: t.ticket.id,
      status: t.ticket.status,
      eventId: t.ticket.eventId,
      ticketTypeId: t.ticket.ticketTypeId,
      event: t.ticket.event
      ? {
        id: t.ticket.event.id,
        slug: t.ticket.event.slug,
        title: t.ticket.event.title,
        startsAt: t.ticket.event.startsAt,
        locationName: t.ticket.event.locationName,
        locationCity: t.ticket.event.locationCity,
      }
      : null,
    ticketType: t.ticket.ticketType
    ? {
      id: t.ticket.ticketType.id,
      name: t.ticket.ticketType.name,
      price: t.ticket.ticketType.price,
      currency: t.ticket.ticketType.currency,
    }
    : null,
  }
  : null,
})
: null,
));

const outgoing = transfers
  .filter((t) => t.fromUserId === userId)
  .map((t) => ({
    id: t.id,
    status: t.status,
    createdAt: t.createdAt,
    completedAt: t.completedAt,
    fromUserId: t.fromUserId,
    toUserId: t.toUserId,
    ticket: t.ticket
    ? {
      id: t.ticket.id,
      status: t.ticket.status,
      eventId: t.ticket.eventId,
      ticketTypeId: t.ticket.ticketTypeId,
      event: t.ticket.event
      ? {
        id: t.ticket.event.id,
        slug: t.ticket.event.slug,
        title: t.ticket.event.title,
        startsAt: t.ticket.event.startsAt,
        locationName: t.ticket.event.locationName,
        locationCity: t.ticket.event.locationCity,
      }
      : null,
    ticketType: t.ticket.ticketType
    ? {
      id: t.ticket.ticketType.id,
      name: t.ticket.ticketType.name,
      price: t.ticket.ticketType.price,
      currency: t.ticket.ticketType.currency,
    }
    : null,
  }
  : null,
})

```

```

        : null,
    }));
}

return NextResponse.json(
{
    ok: true,
    incoming,
    outgoing,
},
{ status: 200 }
);
} catch (error) {
    console.error("Error in /api/me/tickets/transfers:", error);
    return NextResponse.json(
        { ok: false, error: "INTERNAL_ERROR" },
        { status: 500 }
    );
}
}
}

```

app/api/notifications/mark-read/route.ts

```

import { prisma } from "@/lib/prisma";
import { NextRequest, NextResponse } from "next/server";

export async function POST(req: NextRequest) {
    const body = await req.json().catch(() => ({}));
    const { notificationId, userId } = body as { notificationId?: string; userId?: string };
    if (!notificationId || !userId) {
        return NextResponse.json(
            { ok: false, code: "INVALID_PAYLOAD", message: "notificationId e userId são obrigatórios" },
            { status: 400 }
        );
    }

    const notif = await prisma.notification.findFirst({
        where: { id: notificationId, userId },
    });
    if (!notif) {
        return NextResponse.json(
            { ok: false, code: "NOT_FOUND", message: "Notificação não existe" },
            { status: 404 }
        );
    }

    await prisma.notification.update({
        where: { id: notificationId },
        data: { isRead: true, readAt: new Date() },
    });

    return NextResponse.json({ ok: true });
}

```

app/api/notifications/prefs/route.ts

```

import { NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getNotificationPrefs } from "@/lib/notifications";
import { prisma } from "@/lib/prisma";

export async function GET() {
    const supabase = await createSupabaseServer();
    const {
        data: { user },
    } = await supabase.auth.getUser();
    if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

    const prefs = await getNotificationPrefs(user.id);
    return NextResponse.json({ ok: true, prefs });
}

export async function POST(req: Request) {
    const supabase = await createSupabaseServer();
    const {

```

```

    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = await req.json().catch(() => null);
  const updates: Record<string, boolean> = {};
  const allowed = [
    "allowEmailNotifications",
    "allowEventReminders",
    "allowFriendRequests",
    "allowSalesAlerts",
    "allowSystemAnnouncements",
  ];
  allowed.forEach((key) => {
    if (typeof body?.[key] === "boolean") updates[key] = body[key];
  });

  await prisma.notificationPreference.upsert({
    where: { userId: user.id },
    update: updates,
    create: { userId: user.id, ...updates },
  });
}

return NextResponse.json({ ok: true });
}

```

app/api/notifications/route.ts

```

import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { NextRequest, NextResponse } from "next/server";

// Lista notificações com badge de não lidas; usa auth, opcionalmente permite userId (admin/tools)
export async function GET(req: NextRequest) {
  const urlUserId = req.nextUrl.searchParams.get("userId");
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  const resolvedUserId = urlUserId || user?.id;
  if (!resolvedUserId) {
    return NextResponse.json(
      { ok: false, code: "UNAUTHENTICATED", message: "Sessão em falta" },
      { status: 401 },
    );
  }

  const notifications = await prisma.notification.findMany({
    where: { userId: resolvedUserId },
    orderBy: { createdAt: "desc" },
    take: 100,
  });

  const unreadCount = notifications.filter((n) => !n.isRead).length;

  return NextResponse.json({ ok: true, unreadCount, notifications });
}

```

app/api/organizador/become/route.ts

```

// app/api/organizador/become/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { normalizeAndValidateUsername, setUsernameForOwner, UsernameTakenError } from "@/lib/globalUsernames";

type OrganizerPayload = {
  entityType?: string | null;
  businessName?: string | null;
  city?: string | null;
}

```

```

payoutIban?: string | null;
username?: string | null;
};

function sanitizeString(value: unknown) {
  if (typeof value !== "string") return null;
  const trimmed = value.trim();
  return trimmed.length > 0 ? trimmed : null;
}

export async function GET() {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json(
        { ok: false, error: "Não autenticado." },
        { status: 401 },
      );
    }

    const profile = await prisma.profile.findUnique({
      where: { id: user.id },
    });
    if (!profile) {
      return NextResponse.json(
        { ok: false, error: "Perfil não encontrado." },
        { status: 400 },
      );
    }
    const profileSafe = profile;

    const { organizer: activeOrganizer } = await getActiveOrganizerForUser(profileSafe.id);
    const fallbackOrganizer = activeOrganizer ?? null;

    return NextResponse.json(
      {
        ok: true,
        organizer: fallbackOrganizer
          ? {
              id: fallbackOrganizer.id,
              displayName: fallbackOrganizer.displayName,
              status: fallbackOrganizer.status,
              stripeAccountId: fallbackOrganizer.stripeAccountId,
              entityType: fallbackOrganizer.entityType,
              businessName: fallbackOrganizer.businessName,
              city: fallbackOrganizer.city,
              payoutIban: fallbackOrganizer.payoutIban,
            }
          : null,
        },
        { status: 200 },
      );
  } catch (err) {
    console.error("GET /api/organizador/become error:", err);
    return NextResponse.json(
      { ok: false, error: "Erro interno ao obter organizador." },
      { status: 500 },
    );
  }
}

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();

    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json(
        { ok: false, error: "Não autenticado." },

```

```

        { status: 401 },
    );
}

const profile = await prisma.profile.findUnique({
    where: { id: user.id },
});

if (!profile) {
    return NextResponse.json(
        { ok: false, error: "Perfil não encontrado." },
        { status: 400 },
    );
}

// Corpo opcional com dados de onboarding
let payload: OrganizerPayload = {};
try {
    const contentType = req.headers.get("content-type") || "";
    if (contentType.includes("application/json")) {
        payload = await req.json();
    } else if (contentType.includes("application/x-www-form-urlencoded")) {
        const form = await req.formData();
        payload = {
            entityType: form.get("entityType") as string | null,
            businessName: form.get("businessName") as string | null,
            city: form.get("city") as string | null,
            payoutIban: form.get("payoutIban") as string | null,
            username: form.get("username") as string | null,
        };
    }
} catch {
    /* ignora parsing */
}

const entityType = sanitizeString(payload.entityType);
const businessName = sanitizeString(payload.businessName);
const city = sanitizeString(payload.city);
const payoutIban = sanitizeString(payload.payoutIban);
const usernameRaw = sanitizeString(payload.username);

// Procurar organizer existente para este user (por membership ou campo legacy)
const { organizer: activeOrganizer } = await getActiveOrganizerForUser(profile.id);
let organizer = activeOrganizer ?? null;

const displayName =
    businessName ||
    profileSafe.fullName?.trim() ||
    profileSafe.username ||
    "Organizador";

const usernameCandidate = usernameRaw ?? organizer?.username ?? null;
const validatedUsername = usernameCandidate
    ? normalizeAndValidateUsername(usernameCandidate)
    : { ok: false as const, error: "Escolhe um username ORYA para a organização." };

if (!validatedUsername.ok) {
    return NextResponse.json({ ok: false, error: validatedUsername.error }, { status: 400 });
}

const username = validatedUsername.username;

organizer = await prisma.$transaction(async (tx) => {
    async function upsertOrganizer(includePublicName: boolean) {
        if (organizer) {
            return tx.organizer.update({
                where: { id: organizer!.id },
                data: {
                    status: "ACTIVE",
                    displayName,
                    ...(includePublicName ? { publicKey: displayName } : {}),
                    entityType,
                    businessName,
                    city,
                    payoutIban,
                    username,
                },
            });
        }
    }
});

```

```

    }

    return tx.organizer.create({
      data: {
        // userId fica como legacy "created_by" histórico
        userId: profileSafe.id,
        displayName,
        ...(includePublicName ? { publicName: displayName } : {}),
        status: "ACTIVE", // self-serve aberto
        entityType,
        businessName,
        city,
        payoutIban,
        username,
      },
    });
  }

  const nextOrganizer = organizer
    ? await upsertOrganizer(true).catch((err) => {
        const code = (err as { code?: string })?.code;
        const message = err instanceof Error ? err.message : "";
        const missingColumn = code === "P2022" && message.toLowerCase().includes("public_name");
        if (!missingColumn) throw err;
        return upsertOrganizer(false);
      })
    : await upsertOrganizer(true).catch((err) => {
        const code = (err as { code?: string })?.code;
        const message = err instanceof Error ? err.message : "";
        const missingColumn = code === "P2022" && message.toLowerCase().includes("public_name");
        if (!missingColumn) throw err;
        return upsertOrganizer(false);
      });
}

return nextOrganizer;
});

// Garante membership OWNER para o utilizador
try {
  await prisma.organizerMember.upsert({
    where: {
      organizerId_userId: {
        organizerId: organizer.id,
        userId: profile.id,
      },
    },
    update: { role: "OWNER" },
    create: {
      organizerId: organizer.id,
      userId: profile.id,
      role: "OWNER",
    },
  });
} catch (err: unknown) {
  if (typeof err === "object" && err && "code" in err && (err as { code?: string }).code === "P2021") {
    console.warn("[organizador/become] organizer_members table missing; created organizer without membership");
  } else {
    throw err;
  }
}

// Reservar username global fora da transação para não abortar criação
try {
  await setUsernameForOwner({
    username,
    ownerType: "organizer",
    ownerId: organizer.id,
  });
} catch (err) {
  if (err instanceof UsernameTakenError) {
    return NextResponse.json(
      { ok: false, error: "Este @ já está a ser usado – escolhe outro.", code: "USERNAME_TAKEN" },
      { status: 409 },
    );
  }
  console.warn("[organizador/become] username global falhou (ignorado)", err);
}

// Garante que o perfil tem role de organizer

```

```

const roles = Array.isArray(profile.roles) ? profile.roles : [];
if (!roles.includes("organizer")) {
  await prisma.profile.update({
    where: { id: profile.id },
    data: { roles: [...roles, "organizer"] },
  });
}

return NextResponse.json(
  {
    ok: true,
    organizer: {
      id: organizer.id,
      displayName: organizer.displayName,
      status: organizer.status,
      stripeAccountId: organizer.stripeAccountId,
      entityType: organizer.entityType,
      businessName: organizer.businessName,
      city: organizer.city,
      payoutIban: organizer.payoutIban,
      username: organizer.username,
    },
  },
  { status: 200 },
);
} catch (err) {
  if (err instanceof UsernameTakenError) {
    return NextResponse.json(
      { ok: false, error: "Este @ já está a ser usado – escolhe outro.", code: "USERNAME_TAKEN" },
      { status: 409 },
    );
  }
  console.error("POST /api/organizador/become error:", err);
  return NextResponse.json(
    { ok: false, error: "Erro interno ao enviar candidatura de organizador." },
    { status: 500 },
  );
}
}

export async function DELETE(_req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json(
        { ok: false, error: "Não autenticado." },
        { status: 401 },
      );
    }

    await prisma.organizer.deleteMany({
      where: { userId: user.id, status: "PENDING" },
    });

    return NextResponse.json({ ok: true }, { status: 200 });
  } catch (err) {
    console.error("DELETE /api/organizador/become error:", err);
    return NextResponse.json(
      { ok: false, error: "Erro interno ao cancelar candidatura." },
      { status: 500 },
    );
  }
}
}

```

app/api/organizador/estatisticas/audience/route.ts

```

// app/api/organizador/estatisticas/audience/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

```

```

import { TicketStatus } from "@prisma/client";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";

function getDateRangeFromSearchParams(searchParams: URLSearchParams) {
  const range = searchParams.get("range") || "30d";
  const now = new Date();

  if (range === "all") {
    return { from: null as Date | null, to: null as Date | null };
  }

  const days = range === "7d" ? 7 : range === "90d" ? 90 : 30;
  const from = new Date(now);
  from.setDate(from.getDate() - days);

  return { from, to: now };
}

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error: authError,
    } = await supabase.auth.getUser();

    if (authError) {
      console.error("[audience] Error getting user:", authError);
    }

    if (!user) {
      return NextResponse.json(
        { ok: false, error: "UNAUTHENTICATED" },
        { status: 401 }
      );
    }
  }

  const url = new URL(req.url);
  const searchParams = url.searchParams;

  const eventIdParam = searchParams.get("eventId");
  let eventId: number | null = null;

  if (eventIdParam) {
    const parsed = Number(eventIdParam);
    if (!Number.isNaN(parsed)) {
      eventId = parsed;
    }
  }

  const { from, to } = getDateRangeFromSearchParams(searchParams);

  // Garantir que o utilizador é organizador e obter o(s) organizador(es)
  const { organizer } = await getActiveOrganizerForUser(user.id);

  if (!organizer) {
    return NextResponse.json(
      { ok: false, error: "NOT_ORGANIZER" },
      { status: 403 }
    );
  }

  // 1) Buscar tickets relevantes (status ACTIVE/USED) deste organizador
  const tickets = await prisma.ticket.findMany({
    where: {
      status: {
        in: [TicketStatus.ACTIVE, TicketStatus.USED],
      },
      purchasedAt: {
        ...(from ? { gte: from } : {}),
        ...(to ? { lte: to } : {}),
      },
      event: {
        organizerId: organizer.id,
        ...(eventId ? { id: eventId } : {}),
      },
    },
    select: {
  
```

```

        userId: true,
    },
});

if (!tickets.length) {
    return NextResponse.json(
    {
        ok: true,
        totalBuyers: 0,
        cities: [],
        interests: [],
    },
    { status: 200 }
);
}

// 2) Extrair userIds únicos
const userIdSet = new Set<string>();
for (const t of tickets) {
    if (t.userId) {
        userIdSet.add(t.userId);
    }
}

const userIds = Array.from(userIdSet);

if (!userIds.length) {
    return NextResponse.json(
    {
        ok: true,
        totalBuyers: 0,
        cities: [],
        interests: [],
    },
    { status: 200 }
);
}

// 3) Buscar perfis desses utilizadores
const profiles = await prisma.profile.findMany({
    where: {
        id: {
            in: userIds,
        },
    },
    select: {
        id: true,
        city: true,
        favouriteCategories: true,
    },
});
}

// 4) Agregar cidades e interesses
const cityCounts = new Map<string, number>();
const interestCounts = new Map<string, number>();

for (const p of profiles) {
    const cityKey = (p.city || "Desconhecida").trim() || "Desconhecida";
    cityCounts.set(cityKey, (cityCounts.get(cityKey) ?? 0) + 1);

    const favs = p.favouriteCategories || [];
    for (const raw of favs) {
        const cat = (raw || "").trim();
        if (!cat) continue;
        interestCounts.set(cat, (interestCounts.get(cat) ?? 0) + 1);
    }
}

const cities = Array.from(cityCounts.entries())
    .sort((a, b) => b[1] - a[1])
    .slice(0, 10)
    .map(([city, count]) => ({ city, count }));

const interests = Array.from(interestCounts.entries())
    .sort((a, b) => b[1] - a[1])
    .slice(0, 10)
    .map(([name, count]) => ({ name, count }));

```

```

        return NextResponse.json(
          {
            ok: true,
            totalBuyers: userIds.length,
            cities,
            interests,
          },
          { status: 200 }
        );
      } catch (error) {
        console.error("[audience] Internal error:", error);
        return NextResponse.json(
          { ok: false, error: "INTERNAL_ERROR" },
          { status: 500 }
        );
      };
    }
  }
}

```

app/api/organizador/estatisticas/buyers/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { TicketStatus } from "@prisma/client";

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const url = new URL(req.url);
    const eventIdParam = url.searchParams.get("eventId");

    if (!eventIdParam) {
      return NextResponse.json({ ok: false, error: "MISSING_EVENT_ID" }, { status: 400 });
    }

    const eventId = Number(eventIdParam);
    if (!Number.isFinite(eventId)) {
      return NextResponse.json({ ok: false, error: "INVALID_EVENT_ID" }, { status: 400 });
    }

    const { organizer } = await getActiveOrganizerForUser(user.id);

    if (!organizer) {
      return NextResponse.json({ ok: false, error: "NOT_ORGANIZER" }, { status: 403 });
    }

    const event = await prisma.event.findFirst({
      where: { id: eventId, organizerId: organizer.id },
      select: { id: true },
    });

    if (!event) {
      return NextResponse.json({ ok: false, error: "EVENT_NOT_FOUND" }, { status: 404 });
    }

    const tickets = await prisma.ticket.findMany({
      where: {
        eventId: event.id,
        status: { in: [TicketStatus.ACTIVE, TicketStatus.USED, TicketStatus.REFUNDED, TicketStatus.TRANSFERRED, TicketStatus.RESALE_LISTED] },
      },
      orderBy: { purchasedAt: "desc" },
      select: {
        id: true,
        pricePaid: true,
        totalPaidCents: true,
      }
    });
  }
}

```

```

        status: true,
        purchasedAt: true,
        userId: true,
        stripePaymentIntentId: true,
        ticketType: { select: { name: true } },
        guestLink: { select: { guestEmail: true, guestName: true } },
    },
});

const userIds = tickets.map((t) => t.userId).filter(Boolean) as string[];
const profiles = userIds.length
? await prisma.profile.findMany({
    where: { id: { in: userIds } },
    select: { id: true, fullName: true, username: true, city: true },
})
: [];
const profileMap = new Map(profiles.map((p) => [p.id, p]));

const items = tickets.map((t) => {
    const profile = t.userId ? profileMap.get(t.userId) : null;
    const buyerName = t.guestLink?.guestName || profile?.fullName || profile?.username || (t.userId ? "Conta ORYA" :
    "Convidado");
    const buyerEmail = t.guestLink?.guestEmail || "-";

    return {
        id: t.id,
        ticketType: t.ticketType?.name ?? "Bilhete",
        priceCents: t.pricePaid ?? 0,
        totalPaidCents: t.totalPaidCents ?? t.pricePaid ?? 0,
        status: t.status,
        purchasedAt: t.purchasedAt,
        buyerName,
        buyerEmail,
        buyerCity: profile?.city ?? null,
        paymentIntentId: t.stripePaymentIntentId,
    };
});

return NextResponse.json({ ok: true, eventId: event.id, items }, { status: 200 });
} catch (err) {
    console.error("[organizador/buyers] erro inesperado", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/estatisticas/by-ticket-type/route.ts

```

// app/api/organizador/estatisticas/by-ticket-type/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { TicketStatus } from "@prisma/client";

/**
 * 6.3 - API por tipo de bilhete (waves)
 *
 * GET /api/organizador/estatisticas/by-ticket-type?eventId=123
 * (opcionalmente também suporta ?eventSlug=slug-do-evento)
 *
 * Resposta:
 * {
 *   ok: true,
 *   eventId: number,
 *   items: [
 *     {
 *       ticketTypeId: string;
 *       ticketTypeName: string | null;
 *       soldTickets: number;
 *       revenueCents: number;
 *       currency: string | null;
 *     },
 *     ...
 *   ]
 * }
 */

```

```

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error: authError,
    } = await supabase.auth.getUser();

    if (authError) {
      console.error(
        "[by-ticket-type] Erro ao obter utilizador autenticado:",
        authError
      );
    }

    if (!user) {
      return NextResponse.json(
        { ok: false, error: "UNAUTHENTICATED" },
        { status: 401 }
      );
    }
  }

  const { searchParams } = req.nextUrl;

  const eventIdParam = searchParams.get("eventId");
  const eventSlugParam = searchParams.get("eventSlug");

  if (!eventIdParam && !eventSlugParam) {
    return NextResponse.json(
      {
        ok: false,
        error: "MISSING_EVENT_IDENTIFIER",
        message: "Indica eventId ou eventSlug.",
      },
      { status: 400 }
    );
  }

  let eventIdNum: number | null = null;
  if (eventIdParam) {
    const n = Number(eventIdParam);
    if (!Number.isFinite(n)) {
      return NextResponse.json(
        { ok: false, error: "INVALID_EVENT_ID" },
        { status: 400 }
      );
    }
    eventIdNum = n;
  }

  // Filtros de datas opcionais (from/to em ISO)
  const fromParam = searchParams.get("from");
  const toParam = searchParams.get("to");

  const fromDate =
    fromParam && !Number.isNaN(Date.parse(fromParam))
      ? new Date(fromParam)
      : null;
  const toDate =
    toParam && !Number.isNaN(Date.parse(toParam)) ? new Date(toParam) : null;

  // 1) Garantir que este evento pertence ao utilizador (ownerUserId)
  const event = await prisma.event.findFirst({
    where: {
      ...(eventIdNum !== null ? { id: eventIdNum } : {}),
      ...(eventSlugParam ? { slug: eventSlugParam } : {}),
      ownerUserId: user.id,
    },
  });

  if (!event) {
    return NextResponse.json(
      { ok: false, error: "EVENT_NOT_FOUND_OR_NOT_OWNER" },
      { status: 404 }
    );
  }
}

```

```

// 2) Agrupar por ticketType usando SaleLine (fonte de verdade)
const saleLineWhere = {
  eventId: event.id,
  saleSummary: {
    ...FromDate || toDate
    ? {
      createdAt: {
        ...(fromDate ? { gte: fromDate } : {}),
        ...(toDate ? { lte: toDate } : {}),
      },
    }
    : {},
  },
};

const saleLinesGrouped = await prisma.saleLine.groupBy({
  by: ["ticketTypeId"],
  where: saleLineWhere,
  _sum: {
    quantity: true,
    netCents: true,
    grossCents: true,
    platformFeeCents: true,
  },
});

const ticketTypeIds = saleLinesGrouped
  .map((g) => g.ticketTypeId)
  .filter((id): id is number => id != null);

const ticketTypes = await prisma.ticketType.findMany({
  where: {
    id: { in: ticketTypeIds },
    eventId: event.id,
  },
});

const ticketTypeMap = new Map(ticketTypes.map((tt) => [tt.id, tt] as const));

let items = saleLinesGrouped.map((g) => {
  const tt = g.ticketTypeId ? ticketTypeMap.get(g.ticketTypeId) : null;
  const gross = g._sum.grossCents ?? 0;
  const fees = g._sum.platformFeeCents ?? 0;
  const net = g._sum.netCents ?? 0;
  const discount = Math.max(0, gross - fees - net);
  return {
    ticketTypeId: g.ticketTypeId,
    ticketTypeName: tt?.name ?? null,
    soldTickets: g._sum.quantity ?? 0,
    revenueCents: net,
    grossCents: gross,
    discountCents: discount,
    platformFeeCents: fees,
    netCents: net,
    currency: tt?.currency ?? null,
  };
});

// Fallback para legacy se não houver linhas em SaleLine
if (items.length === 0) {
  const ticketsGrouped = await prisma.ticket.groupBy({
    by: ["ticketTypeId"],
    where: {
      eventId: event.id,
      status: {
        in: [TicketStatus.ACTIVE, TicketStatus.USED],
      },
      ...FromDate || toDate
      ? {
        purchasedAt: {
          ...(fromDate ? { gte: fromDate } : {}),
          ...(toDate ? { lte: toDate } : {}),
        },
      }
      : {},
    },
    _count: {
      _all: true,
    },
  });
}

```

```

        },
        _sum: {
          pricePaid: true,
        },
      });
    }

    const fallbackIds = ticketsGrouped
      .map((g) => g.ticketTypeId)
      .filter((id): id is number => id != null);

    const fallbackTypes = await prisma.ticketType.findMany({
      where: {
        id: { in: fallbackIds },
        eventId: event.id,
      },
    });
    const fallbackMap = new Map(fallbackTypes.map((tt) => [tt.id, tt] as const));

    items = ticketsGrouped.map((g) => {
      const tt = g.ticketTypeId ? fallbackMap.get(g.ticketTypeId) : null;
      const gross = g._sum.pricePaid ?? 0;
      const fees = 0;
      const discount = 0;
      return {
        ticketTypeId: g.ticketTypeId,
        ticketTypeName: tt?.name ?? null,
        soldTickets: g._count._all,
        revenueCents: gross,
        grossCents: gross,
        discountCents: discount,
        platformFeeCents: fees,
        netCents: gross - fees,
        currency: tt?.currency ?? null,
      };
    });
  }

  return NextResponse.json(
    {
      ok: true,
      eventId: event.id,
      items,
    },
    { status: 200 }
  );
} catch (error) {
  console.error(`[by-ticket-type] Erro inesperado ao calcular estatísticas:`, error);
}
return NextResponse.json(
  { ok: false, error: "INTERNAL_ERROR" },
  { status: 500 }
);
}
}
}

```

app/api/organizador/estatisticas/overview/route.ts

```

// app/api/organizador/estatisticas/overview/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { TicketStatus, EventStatus, Prisma } from "@prisma/client";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";

/**
 * F6 – Estatísticas do organizador (overview)
 *
 * GET /api/organizador/estatisticas/overview
 *
 * Query params opcionais:
 * - range: "7d" | "30d" | "all" (default: "30d")
 *
 * Devolve um resumo com:

```

```

* - totalTickets: nº de bilhetes vendidos no período
* - totalRevenueCents: soma de pricePaid no período
* - eventsWithSalesCount: nº de eventos com pelo menos 1 venda no período
* - activeEventsCount: nº de eventos publicados do organizador (no geral)
*/

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error: authError,
    } = await supabase.auth.getUser();

    if (authError) {
      console.error("[organizador/overview] Erro ao obter user:", authError);
    }

    if (!user) {
      return NextResponse.json(
        { ok: false, error: "UNAUTHENTICATED" },
        { status: 401 },
      );
    }
  }

  const url = new URL(req.url);
  const range = url.searchParams.get("range") || "30d"; // 7d | 30d | all

  // Confirmar que o utilizador é organizador (roles no profile)
  const profile = await prisma.profile.findUnique({
    where: { id: user.id },
    select: { roles: true },
  });

  const roles = (profile?.roles || []) as string[];
  const isOrganizer = roles.includes("organizer") || roles.includes("ORGANIZER");

  if (!isOrganizer) {
    return NextResponse.json(
      { ok: false, error: "NOT_ORGANIZER" },
      { status: 403 },
    );
  }

  const { organizer } = await getActiveOrganizerForUser(user.id);

  if (!organizer) {
    return NextResponse.json(
      { ok: false, error: "NOT_ORGANIZER" },
      { status: 403 },
    );
  }

  // Cálculo do intervalo temporal
  let fromDate: Date | undefined;
  let toDate: Date | undefined;

  const now = new Date();

  if (range === "7d") {
    fromDate = new Date(now.getTime() - 7 * 24 * 60 * 60 * 1000);
    toDate = now;
  } else if (range === "30d") {
    fromDate = new Date(now.getTime() - 30 * 24 * 60 * 60 * 1000);
    toDate = now;
  } else {
    // "all" -> sem filtro de datas
    fromDate = undefined;
    toDate = undefined;
  }

  // Fonte preferencial: sale_summaries + sale_lines
  const createdAtFilter: Prisma.DateTimeFilter<"SaleSummary"> = {};
  if (fromDate) createdAtFilter.gte = fromDate;
  if (toDate) createdAtFilter.lte = toDate;

  const summaries = await prisma.saleSummary.findMany({
    where: {

```

```

...(Object.keys(createdAtFilter).length > 0
? { createdAt: createdAtFilter }
: {}),
event: { organizerId: organizer.id },
},
select: {
  id: true,
  eventId: true,
  netCents: true,
  discountCents: true,
  platformFeeCents: true,
  subtotalCents: true,
  lines: {
    select: { quantity: true },
  },
},
),
});

let totalTickets = summaries.reduce(
  (acc, s) => acc + s.lines.reduce((q, l) => q + (l.quantity ?? 0), 0),
  0,
);
let grossCents = summaries.reduce((acc, s) => acc + (s.subtotalCents ?? 0), 0);
let discountCents = summaries.reduce((acc, s) => acc + (s.discountCents ?? 0), 0);
let platformFeeCents = summaries.reduce(
  (acc, s) => acc + (s.platformFeeCents ?? 0),
  0,
);
let netRevenueCents = summaries.reduce((acc, s) => acc + (s.netCents ?? 0), 0);

let eventsWithSalesCount = new Set(summaries.map((s) => s.eventId)).size;

// Fallback para legacy (caso ainda não haja sale_summaries)
if (summaries.length === 0) {
  const ticketWhere: Prisma.TicketWhereInput = {
    status: {
      in: [TicketStatus.ACTIVE, TicketStatus.USED],
    },
    event: {
      organizerId: organizer.id,
    },
  };

  if (fromDate && toDate) {
    ticketWhere.purchasedAt = {
      gte: fromDate,
      lte: toDate,
    };
  }
}

const tickets = await prisma.ticket.findMany({
  where: ticketWhere,
  select: {
    pricePaid: true,
    eventId: true,
  },
});

totalTickets = tickets.length;
grossCents = tickets.reduce((sum, t) => sum + (t.pricePaid ?? 0), 0);
// sem breakdown legacy: assumimos sem desconto/fee explícito
discountCents = 0;
platformFeeCents = 0;
netRevenueCents = grossCents;
eventsWithSalesCount = new Set(tickets.map((t) => t.eventId)).size;
}

// Contar eventos publicados do organizador (no geral, não só no período)
const activeEventsCount = await prisma.event.count({
  where: {
    organizerId: organizer.id,
    status: EventStatus.PUBLISHED,
  },
});

return NextResponse.json(
{
  ok: true,
}

```

```

        range,
        totalTickets,
        totalRevenueCents: netRevenueCents,
        grossCents,
        discountCents,
        platformFeeCents,
        netRevenueCents,
        eventsWithSalesCount,
        activeEventsCount,
    ),
    { status: 200 },
);
} catch (error) {
    console.error("[organizador/overview] Erro inesperado:", error);
    return NextResponse.json(
        { ok: false, error: "INTERNAL_ERROR" },
        { status: 500 },
    );
}
}
}

```

app/api/organizador/estatisticas/time-series/route.ts

```

// app/api/organizador/estatisticas/time-series/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@lib/prisma";
import { createSupabaseServer } from "@lib/supabaseServer";
import { Prisma, TicketStatus } from "@prisma/client";
import { getActiveOrganizerForUser } from "@lib/organizerContext";

function parseRangeParams(url: URL) {
    const range = url.searchParams.get("range");
    const fromParam = url.searchParams.get("from");
    const toParam = url.searchParams.get("to");

    let from: Date | null = null;
    let to: Date | null = null;

    if (fromParam || toParam) {
        if (fromParam) {
            const d = new Date(fromParam);
            if (!Number.isNaN(d.getTime())) from = d;
        }
        if (toParam) {
            const d = new Date(toParam);
            if (!Number.isNaN(d.getTime())) to = d;
        }
    } else if (range) {
        const now = new Date();
        to = now;
        const base = new Date(now);

        switch (range) {
            case "7d": {
                base.setDate(base.getDate() - 7);
                from = base;
                break;
            }
            case "30d": {
                base.setDate(base.getDate() - 30);
                from = base;
                break;
            }
            case "90d": {
                base.setDate(base.getDate() - 90);
                from = base;
                break;
            }
            default:
                // "all" ou desconhecido -> deixamos from = null (sem limite inferior)
                from = null;
        }
    }
}

return { from, to };

```

```

}

function formatDayKey(date: Date) {
  // YYYY-MM-DD
  return date.toISOString().slice(0, 10);
}

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error: authError,
    } = await supabase.auth.getUser();

    if (authError) {
      console.error(
        "[organizador/time-series] Erro ao obter utilizador:",
        authError
      );
    }

    if (!user) {
      return NextResponse.json(
        { ok: false, error: "UNAUTHENTICATED" },
        { status: 401 }
      );
    }
  }

  const url = new URL(req.url);
  const eventIdParam = url.searchParams.get("eventId");
  const { from, to } = parseRangeParams(url);

  let eventId: number | null = null;
  if (eventIdParam) {
    const parsed = Number(eventIdParam);
    if (Number.isNaN(parsed)) {
      return NextResponse.json(
        { ok: false, error: "INVALID_EVENT_ID" },
        { status: 400 }
      );
    }
    eventId = parsed;
  }

  // 1) Garantir que o user é um organizador ativo
  const { organizer } = await getActiveOrganizerForUser(user.id);

  if (!organizer) {
    return NextResponse.json(
      { ok: false, error: "NOT_ORGANIZER" },
      { status: 403 }
    );
  }

  // 2) Preferir fonte de verdade: sale_summaries + sale_lines
  const createdAtFilter: Prisma.DateTimeFilter<"SaleSummary"> = {};
  if (from) createdAtFilter.gte = from;
  if (to) createdAtFilter.lte = to;

  const saleSummaries = await prisma.saleSummary.findMany({
    where: {
      ...(Object.keys(createdAtFilter).length > 0
        ? { createdAt: createdAtFilter }
        : {}),
      event: {
        organizerId: organizer.id,
      },
      eventId: eventId ?? undefined,
    },
    select: {
      createdAt: true,
      netCents: true,
      currency: true,
      lines: {
        select: {
          quantity: true,
        },
      },
    },
  });
}

```

```

        },
    },
});

type DayBucket = {
    date: string; // YYYY-MM-DD
    tickets: number;
    revenueCents: number;
    grossCents: number;
    discountCents: number;
    platformFeeCents: number;
    currency: string | null;
};

const buckets: Record<string, DayBucket> = {};

for (const s of saleSummaries) {
    const key = formatDayKey(s.createdAt);
    if (!buckets[key]) {
        buckets[key] = {
            date: key,
            tickets: 0,
            revenueCents: 0,
            grossCents: 0,
            discountCents: 0,
            platformFeeCents: 0,
            currency: s.currency ?? null,
        };
    }
    const qty = s.lines.reduce((acc, l) => acc + (l.quantity ?? 0), 0);
    buckets[key].tickets += qty;
    buckets[key].revenueCents += s.netCents ?? 0;
    buckets[key].grossCents += s.subtotalCents ?? 0;
    buckets[key].discountCents += s.discountCents ?? 0;
    buckets[key].platformFeeCents += s.platformFeeCents ?? 0;
}

// Fallback: se ainda não houver sale_summaries (ex.: legacy), usa tickets
if (Object.keys(buckets).length === 0) {
    const purchasedAtFilter: Prisma.DateTimeFilter<"Ticket"> = {};
    if (from) purchasedAtFilter.gte = from;
    if (to) purchasedAtFilter.lte = to;

    const ticketWhere: Prisma.TicketWhereInput = {
        status: {
            in: [TicketStatus.ACTIVE, TicketStatus.USED],
        },
        purchasedAt: {
            Object.keys(purchasedAtFilter).length > 0
                ? purchasedAtFilter
                : undefined,
        },
        event: {
            organizerId: organizer.id,
        },
        eventId: eventId ?? undefined,
    };

    const tickets = await prisma.ticket.findMany({
        where: ticketWhere,
        select: {
            purchasedAt: true,
            pricePaid: true,
            currency: true,
            platformFeeCents: true,
        },
    });

    for (const t of tickets) {
        if (!t.purchasedAt) continue;
        const key = formatDayKey(t.purchasedAt);
        if (!buckets[key]) {
            buckets[key] = {
                date: key,
                tickets: 0,
                revenueCents: 0,
                grossCents: 0,
                discountCents: 0,
                platformFeeCents: 0,
            };
        }
        const qty = t.lines.reduce((acc, l) => acc + (l.quantity ?? 0), 0);
        buckets[key].tickets += qty;
        buckets[key].revenueCents += t.netCents ?? 0;
        buckets[key].grossCents += t.subtotalCents ?? 0;
        buckets[key].discountCents += t.discountCents ?? 0;
        buckets[key].platformFeeCents += t.platformFeeCents ?? 0;
    }
}

```

```

        currency: t.currency ?? null,
    );
}
buckets[key].tickets += 1;
const gross = t.pricePaid ?? 0;
const fees = t.platformFeeCents ?? 0;
buckets[key].grossCents += gross;
buckets[key].platformFeeCents += fees;
buckets[key].revenueCents += gross - fees;
// desconto não disponível em legacy
}
}

const points = Object.values(buckets).sort((a, b) =>
    a.date.localeCompare(b.date)
);

return NextResponse.json(
{
    ok: true,
    range: {
        from: from ? from.toISOString() : null,
        to: to ? to.toISOString() : null,
    },
    points: points.map((p) => ({
        ...p,
        netCents: p.revenueCents,
    })),
},
{
    status: 200
});
} catch (error) {
    console.error(
        "[organizador/time-series] Erro interno ao gerar série temporal:",
        error
    );
    return NextResponse.json(
        { ok: false, error: "INTERNAL_ERROR" },
        { status: 500 }
    );
}
}
}

```

app/api/organizador/events/create/route.ts

```

// app/api/organizador/events/create/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { PORTUGAL_CITIES } from "@/config/cities";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";

// Tipos esperados no body do pedido
type TicketTypeInput = {
    name?: string;
    price?: number;
    totalQuantity?: number | null;
};

type CreateOrganizerEventBody = {
    title?: string;
    description?: string;
    startsAt?: string;
    endsAt?: string;
    locationName?: string;
    locationCity?: string;
    templateType?: string; // PARTY | SPORT | VOLUNTEERING | TALK | OTHER
    ticketTypes?: TicketTypeInput[];
    address?: string | null;
    categories?: string[];
    resaleMode?: string; // ALWAYS | AFTER_SOLD_OUT | DISABLED
    coverImageUrl?: string | null;
}

```



```

    | "AFTER SOLD OUT"
    | "DISABLED"
    | undefined;
const payoutMode = body.payoutMode?.toUpperCase() === "PLATFORM" ? "PLATFORM" : "ORGANIZER";

if (!title) {
  return NextResponse.json(
    { ok: false, error: "Título é obrigatório." },
    { status: 400 }
  );
}

if (!startsAtRaw) {
  return NextResponse.json(
    { ok: false, error: "Data/hora de início é obrigatória." },
    { status: 400 }
  );
}

if (!locationCity) {
  return NextResponse.json(
    { ok: false, error: "Cidade é obrigatória." },
    { status: 400 },
  );
}

const cityAllowed = PORTUGAL_CITIES.includes(locationCity as (typeof PORTUGAL_CITIES)[number]);
if (!cityAllowed) {
  return NextResponse.json(
    { ok: false, error: "Cidade inválida. Escolhe uma cidade da lista disponível na ORYA." },
    { status: 400 },
  );
}

const categoriesInput = Array.isArray(body.categories) ? body.categories : [];
const allowedCategories = [
  "FESTA",
  "DESPORTO",
  "CONCERTO",
  "PALESTRA",
  "ARTE",
  "COMIDA",
  "DRINKS",
];
const categories = categoriesInput
  .map((c) => c.trim().toUpperCase())
  .filter((c) => allowedCategories.includes(c));

if (categories.length === 0) {
  return NextResponse.json(
    { ok: false, error: "Escolhe pelo menos uma categoria." },
    { status: 400 },
  );
}

const parseDate = (raw?: string | null) => {
  if (!raw) return null;
  const normalized = raw.replace(" ", "T");
  const date = new Date(normalized);
  if (!Number.isNaN(date.getTime())) return date;
  const alt = new Date(`${normalized}:00`);
  if (!Number.isNaN(alt.getTime())) return alt;
  return null;
};

const startsAt = parseDate(startsAtRaw);
if (!startsAt) {
  return NextResponse.json(
    { ok: false, error: "Data/hora de início inválida." },
    { status: 400 }
  );
}

// Para simplificar e evitar conflitos de tipos, endsAt será sempre enviado.
// Se o utilizador não mandar, usamos a mesma data/hora de início.
const endsAtParsed = parseDate(endsAtRaw);
const endsAt = endsAtParsed && endsAtParsed >= startsAt ? endsAtParsed : startsAt;

const templateType =

```

```

templateTypeRaw ??
(categories.includes("FESTA")
? "PARTY"
: categories.includes("DESPORTO")
? "SPORT"
: categories.includes("PALESTRA")
? "TALK"
: "OTHER");

const ticketTypesInput = body.ticketTypes ?? [];
const coverImageUrl = body.coverImageUrl?.trim()?.() || null;
// Validar tipos de bilhete
const ticketTypesData = ticketTypesInput
.map((t) => {
  const name = t.name?.trim();
  if (!name) return null;

  const priceRaw =
    typeof t.price === "number" && !Number.isNaN(t.price) ? t.price : 0;

  // preço mínimo 0,50 € (ou 0 para grátis)
  if (priceRaw > 0 && priceRaw < 0.5) {
    throw new Error("O preço mínimo de bilhete é 0,50 € (ou grátis).");
  }

  const totalQuantity =
    typeof t.totalQuantity === "number" && t.totalQuantity > 0
    ? t.totalQuantity
    : null;

  return {
    name,
    price: priceRaw,
    totalQuantity,
  };
})
.filter((t): t is { name: string; price: number; totalQuantity: number | null } =>
  Boolean(t)
);

const paymentsStatus = organizer
? organizer.stripeAccountId
? organizer.stripeChargesEnabled && organizer.stripePayoutsEnabled
? "READY"
: "PENDING"
: "NO_STRIPE"
: "NO_STRIPE";

const hasPaidTickets = ticketTypesData.some((t) => t.price > 0);
if (payoutMode === "ORGANIZER" && hasPaidTickets && paymentsStatus !== "READY" && !isAdmin) {
  return NextResponse.json(
  {
    ok: false,
    code: "PAYMENTS_NOT_READY",
    error: "Para vender bilhetes pagos, primeiro liga a tua conta Stripe em Finanças & Payouts.",
  },
  { status: 403 },
);
}

if (ticketTypesData.length === 0) {
  return NextResponse.json(
  { ok: false, error: "Pelo menos um tipo de bilhete é obrigatório." },
  { status: 400 }
);
}

const baseSlug = slugify(title) || "evento";
const randomSuffix = Math.random().toString(36).slice(2, 8);
const slug = `${baseSlug}-${randomSuffix}`;
const resaleMode =
  resaleModeRaw === "AFTER SOLD OUT" || resaleModeRaw === "DISABLED"
  ? resaleModeRaw
  : "ALWAYS";

// Criar o evento primeiro
const event = await prisma.event.create({
  data: {
    slug,
  }
});

```

```

        title,
        description,
        type: "ORGANIZER_EVENT",
        templateType,
        ownerUserId: profile.id,
        organizerId: organizer?.id ?? null,
        startsAt,
        endsAt,
        locationName,
        locationCity,
        address,
        isFree: ticketTypesData.every((t) => t.price === 0),
        status: "PUBLISHED",
        resaleMode,
        coverImageUrl,
        isTest: isAdmin && body.isTest === true,
        payoutMode,
    },
});

if (templateType === "SPORT" && body.padel && organizer) {
    const padelConfig = body.padel as {
        padelClubId?: number | null;
        partnerClubIds?: number[];
        format?: string;
        numberOfCourts?: number;
        ruleSetId?: number | null;
        defaultCategoryId?: number | null;
        advancedSettings?: unknown;
    };
    const padelClubId =
        typeof padelConfig.padelClubId === "number" && Number.isFinite(padelConfig.padelClubId)
            ? padelConfig.padelClubId
            : null;
    const partnerClubIds = Array.isArray(padelConfig.partnerClubIds)
        ? padelConfig.partnerClubIds.filter((id) => typeof id === "number" && Number.isFinite(id))
        : [];
    const advancedSettings = padelConfig.advancedSettings ?? null;
    try {
        await prisma.padelTournamentConfig.upsert({
            where: { eventId: event.id },
            create: {
                eventId: event.id,
                organizerId: organizer.id,
                padelClubId,
                partnerClubIds,
                numberOfCourts: Math.max(1, padelConfig.numberOfCourts || 1),
                format: (padelConfig.format as any) ?? "TODOS_CONTRA_TODOS",
                ruleSetId: padelConfig.ruleSetId || undefined,
                defaultCategoryId: padelConfig.defaultCategoryId || undefined,
                advancedSettings: advancedSettings as any,
            },
            update: {
                padelClubId,
                partnerClubIds,
                numberOfCourts: Math.max(1, padelConfig.numberOfCourts || 1),
                format: (padelConfig.format as any) ?? "TODOS_CONTRA_TODOS",
                ruleSetId: padelConfig.ruleSetId || undefined,
                defaultCategoryId: padelConfig.defaultCategoryId || undefined,
                advancedSettings: advancedSettings as any,
            },
        });
    } catch (padelErr) {
        console.warn("[organizador/events/create] padel config falhou", padelErr);
    }
}

// Criar os tipos de bilhete associados a este evento
await prisma.ticketType.createMany({
    data: ticketTypesData.map((t) => ({
        eventId: event.id,
        name: t.name,
        price: Math.round(t.price * 100), // guardar em cents
        // Assumimos que currency tem default "EUR" e restantes campos têm defaults.
        totalQuantity: t.totalQuantity ?? null,
    })),
});

```

```

    return NextResponse.json(
      {
        ok: true,
        event: {
          id: event.id,
          slug: event.slug,
          title: event.title,
        },
      },
      { status: 201 }
    );
  } catch (err) {
    console.error("POST /api/organizador/events/create error:", err);
    return NextResponse.json(
      { ok: false, error: "Erro interno ao criar evento." },
      { status: 500 }
    );
  };
}
}

```

app/api/organizador/events/list/route.ts

```

// app/api/organizador/events/list/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@lib/prisma";
import { createSupabaseServer } from "@lib/supabaseServer";
import { ensureAuthenticated } from "@lib/security";
import { TicketStatus } from "@prisma/client";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";

export async function GET(req: NextRequest) {
  try {
    const url = new URL(req.url);
    const limit = Math.min(Number(url.searchParams.get("limit") ?? 100), 500);

    const supabase = await createSupabaseServer();

    // 1) Garante que o utilizador está autenticado
    const user = await ensureAuthenticated(supabase);

    // 2) Buscar o profile correspondente a este user
    const profile = await prisma.profile.findUnique({
      where: { id: user.id },
    });

    if (!profile) {
      return NextResponse.json(
        {
          ok: false,
          error: "Perfil não encontrado. Completa o onboarding antes de gerires eventos como organizador.",
        },
        { status: 403 }
      );
    }

    // 3) Encontrar o organizer ligado a este perfil (membership > legacy)
    const { organizer, membership } = await getActiveOrganizerForUser(profile.id, {
      roles: ["OWNER", "CO_OWNER", "ADMIN"],
    });

    if (!organizer || !membership || !isOrgAdminOrAbove(membership.role)) {
      return NextResponse.json(
        {
          ok: false,
          error: "Ainda não és organizador. Usa o botão 'Quero ser organizador' para começar.",
        },
        { status: 403 }
      );
    }

    const events = await prisma.event.findMany({
      where: { organizerId: organizer.id, isDeleted: false },
      orderBy: {

```

```

        startsAt: "asc",
    },
    include: {
        categories: true,
        padelTournamentConfig: {
            select: { padelClubId: true, partnerClubIds: true },
        },
    },
    take: limit,
});

const capacityAgg = await prisma.ticketType.groupBy({
    by: ['eventId'],
    where: { eventId: { in: events.map((e) => e.id) } },
    _sum: { totalQuantity: true },
});

const capacityMap = new Map<number, number>();
capacityAgg.forEach((row) => {
    const sum = row._sum.totalQuantity ?? 0;
    capacityMap.set(row.eventId, sum);
});

const ticketStats = await prisma.ticket.groupBy({
    by: ['eventId'],
    where: {
        status: { in: [TicketStatus.ACTIVE, TicketStatus.USED] },
        event: { organizerId: organizer.id },
    },
    _count: { _all: true },
    _sum: { pricePaid: true, totalPaidCents: true, platformFeeCents: true },
});

const statsMap = new Map<
    number,
    {
        tickets: number;
        revenueCents: number;
        totalPaidCents: number;
        platformFeeCents: number;
    }
>();

ticketStats.forEach((stat) => {
    statsMap.set(stat.eventId, {
        tickets: stat._count._all,
        revenueCents: stat._sum.pricePaid ?? 0,
        totalPaidCents: stat._sum.totalPaidCents ?? 0,
        platformFeeCents: stat._sum.platformFeeCents ?? 0,
    });
});

const padelClubIds = new Set<number>();
events.forEach((ev) => {
    const cfg = ev.padelTournamentConfig;
    if (cfg?.padelClubId) padelClubIds.add(cfg.padelClubId);
    (cfg?.partnerClubIds || []).forEach((id) => padelClubIds.add(id));
});
const padelClubs =
    padelClubIds.size > 0
        ? await prisma.padelClub.findMany({
            where: { id: { in: Array.from(padelClubIds) } },
            select: { id: true, name: true },
        })
        : [];
const padelClubMap = new Map<number, string>();
padelClubs.forEach((c) => padelClubMap.set(c.id, c.name || `Clube ${c.id}`));

const items = events.map((event) => ({
    id: event.id,
    slug: event.slug,
    title: event.title,
    description: event.description,
    startsAt: event.startsAt,
    endsAt: event.endsAt,
    locationName: event.locationName,
    locationCity: event.locationCity,
    status: event.status,
})

```

```

        templateType: event.templateType,
        isFree: event.isFree,
        ticketsSold: statsMap.get(event.id)?.tickets ?? 0,
        revenueCents: statsMap.get(event.id)?.revenueCents ?? 0,
        totalPaidCents: statsMap.get(event.id)?.totalPaidCents ?? 0,
        platformFeeCents: statsMap.get(event.id)?.platformFeeCents ?? 0,
        capacity: capacityMap.get(event.id) ?? null,
        categories: event.categories.map((c) => c.category),
        padelClubId: event.padelTournamentConfig?.padelClubId ?? null,
        padelPartnerClubIds: event.padelTournamentConfig?.partnerClubIds ?? [],
        padelClubName: event.padelTournamentConfig?.padelClubId ? padelClubMap.get(event.padelTournamentConfig.padelClubId) ?? null : null,
        padelPartnerClubNames: (event.padelTournamentConfig?.partnerClubIds || []).map((id) => padelClubMap.get(id) ?? null),
    }));
}

return NextResponse.json(
{
    ok: true,
    items,
},
{ status: 200 }
);
} catch (err) {
console.error("GET /api/organizador/events/list error:", err);
return NextResponse.json(
{
    ok: false,
    error: "Erro interno ao carregar eventos do organizador.",
},
{ status: 500 }
);
}
}
}
}

```

app/api/organizador/events/stats/route.ts

app/api/organizador/events/update/route.ts

```

// app/api/organizador/events/update/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";
import { TicketTypeStatus, Prisma, EventTemplateType } from "@prisma/client";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";
import { PORTUGAL_CITIES } from "@/config/cities";

type TicketTypeUpdate = {
    id: number;
    status?: TicketTypeStatus;
};

type NewTicketType = {
    name: string;
    description?: string | null;
    price: number; // cents
    totalQuantity?: number | null;
    startsAt?: string | null;
    endsAt?: string | null;
};

type UpdateEventBody = {
    eventId?: number;
    archive?: boolean;
    title?: string | null;
    description?: string | null;
    startsAt?: string | null;
    endsAt?: string | null;
    locationName?: string | null;
    locationCity?: string | null;
    address?: string | null;
    templateType?: string | null;
};

```

```

isFree?: boolean;
coverImageUrl?: string | null;
ticketTypeUpdates?: TicketTypeUpdate[];
newTicketTypes?: NewTicketType[];
payoutMode?: string | null;
};

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    let body: UpdateEventBody | null = null;
    try {
      body = (await req.json()) as UpdateEventBody;
    } catch {
      return NextResponse.json({ ok: false, error: "Body inválido." }, { status: 400 });
    }

    const eventId = Number(body?.eventId);
    if (!eventId || Number.isNaN(eventId)) {
      return NextResponse.json({ ok: false, error: "eventId é obrigatório." }, { status: 400 });
    }

    // Autorização: perfil + membership no organizer do evento
    const profile = await prisma.profile.findUnique({ where: { id: user.id } });
    if (!profile) {
      return NextResponse.json(
        { ok: false, error: "Perfil não encontrado. Completa o onboarding." },
        { status: 400 },
      );
    }
    const event = await prisma.event.findUnique({
      where: { id: eventId },
      include: { ticketTypes: true, organizer: true },
    });

    if (!event) {
      return NextResponse.json({ ok: false, error: "Evento não encontrado." }, { status: 404 });
    }

    const membership = await prisma.organizerMember.findUnique({
      where: { organizerId_userId: { organizerId: event.organizerId, userId: user.id } },
    });
    if (!membership || !isOrgAdminOrAbove(membership.role)) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    const isAdmin = Array.isArray(profile.roles) ? profile.roles.includes("admin") : false;

    const organizer = event.organizer;
    const paymentsStatus = organizer
      ? organizer.stripeAccountId
      : organizer.stripeChargesEnabled && organizer.stripePayoutsEnabled
      ? "READY"
      : "PENDING"
      : "NO_STRIPE"
      : "NO_STRIPE";

    const dataUpdate: Partial<Prisma.EventUncheckedUpdateInput> = {};
    if (body.archive === true) {
      dataUpdate.isDeleted = true;
      dataUpdate.deletedAt = new Date();
    }
    if (body.title !== undefined) dataUpdate.title = body.title?.trim() ?? "";
    if (body.description !== undefined) dataUpdate.description = body.description ?? "";
    if (body.startsAt) {
      const d = new Date(body.startsAt);
      if (Number.isNaN(d.getTime())) {
        return NextResponse.json({ ok: false, error: "startsAt inválido." }, { status: 400 });
      }
      dataUpdate.startsAt = d;
    }
    if (body.endsAt) {
      const d = new Date(body.endsAt);
      if (Number.isNaN(d.getTime())) {
        return NextResponse.json({ ok: false, error: "endsAt inválido." }, { status: 400 });
      }
    }
  }
}

```

```

        dataUpdate.endsAt = d;
    }
    if (body.locationName !== undefined) dataUpdate.locationName = body.locationName ?? "";
    if (body.locationCity !== undefined) {
        const city = body.locationCity ?? "";
        if (city && !PORTUGAL_CITIES.includes(city as (typeof PORTUGAL_CITIES)[number])) {
            return NextResponse.json(
                { ok: false, error: "Cidade inválida. Escolhe uma cidade da lista disponível na ORYA." },
                { status: 400 },
            );
        }
        dataUpdate.locationCity = city;
    }
    if (body.address !== undefined) dataUpdate.address = body.address ?? null;
    if (body.templateType) {
        const tpl = body.templateType.toUpperCase();
        if ((Object.values(EventTemplateType) as string[]).includes(tpl)) {
            dataUpdate.templateType = tpl as EventTemplateType;
        }
    }
    // Não permitir converter para grátis se já há bilhetes e atualmente não é grátis
    if (body.isFree !== undefined) {
        const hasTickets = event.ticketTypes.length > 0;
        const wantsFree = body.isFree === true;
        const wasFree = event.isFree === true;

        if (wantsFree && !wasFree && hasTickets) {
            return NextResponse.json(
                {
                    ok: false,
                    error: "Este evento já tem bilhetes. Não pode ser convertido em evento grátis.",
                },
                { status: 400 },
            );
        }
        dataUpdate.isFree = body.isFree;
    }
    if (body.coverImageUrl !== undefined) dataUpdate.coverImageUrl = body.coverImageUrl ?? null;
    if (
        isAdmin &&
        body.payoutMode &&
        (body.payoutMode.toUpperCase() === "PLATFORM" || body.payoutMode.toUpperCase() === "ORGANIZER")
    ) {
        dataUpdate.payoutMode = body.payoutMode.toUpperCase() as Prisma.PayoutMode;
    }

    const ticketTypeUpdates = Array.isArray(body.ticketTypeUpdates)
        ? body.ticketTypeUpdates
        : [];
    const newTicketTypes = Array.isArray(body.newTicketTypes) ? body.newTicketTypes : [];

    const payoutMode = event.payoutMode ?? "ORGANIZER";
    const hasExistingPaid = event.ticketTypes.some(
        (t) => (t.price ?? 0) > 0 && t.status !== TicketTypeStatus.CANCELLED
    );
    const hasNewPaid = newTicketTypes.some((nt) => Number(nt.price ?? 0) > 0);
    if (
        payoutMode === "ORGANIZER" &&
        (hasExistingPaid || hasNewPaid) &&
        paymentsStatus !== "READY" &&
        !isAdmin
    ) {
        return NextResponse.json(
            {
                ok: false,
                code: "PAYMENTS_NOT_READY",
                error: "Para vender bilhetes pagos, primeiro liga a tua conta Stripe em Finanças & Payouts.",
            },
            { status: 403 },
        );
    }

    const transactions: Prisma.PrismaPromise<unknown>[] = [];

    if (Object.keys(dataUpdate).length > 0) {
        transactions.push(
            prisma.event.update({

```

```

        where: { id: eventId },
        data: dataUpdate,
      }),
    );
}

if (ticketTypeUpdates.length > 0) {
  for (const upd of ticketTypeUpdates) {
    const tt = event.ticketTypes.find((t) => t.id === upd.id);
    if (!tt) continue;
    const status =
      upd.status && Object.values(TicketTypeStatus).includes(upd.status)
        ? upd.status
        : null;
    if (status) {
      transactions.push(
        prisma.ticketType.update({
          where: { id: tt.id },
          data: { status },
        }),
      );
    }
  }
}

if (newTicketTypes.length > 0) {
  for (const nt of newTicketTypes) {
    const price = Number(nt.price ?? 0);
    const totalQuantity =
      typeof nt.totalQuantity === "number" && nt.totalQuantity > 0
        ? nt.totalQuantity
        : null;
    const startsAt = nt.startsAt ? new Date(nt.startsAt) : null;
    const endsAt = nt.endsAt ? new Date(nt.endsAt) : null;

    transactions.push(
      prisma.ticketType.create({
        data: {
          eventId,
          name: nt.name?.trim() || "Bilhete",
          description: nt.description ?? null,
          price,
          totalQuantity,
          status: TicketTypeStatus.ON_SALE,
          startsAt: startsAt && !Number.isNaN(startsAt.getTime()) ? startsAt : null,
          endsAt: endsAt && !Number.isNaN(endsAt.getTime()) ? endsAt : null,
        },
      }),
    );
  }
}

if (transactions.length === 0) {
  return NextResponse.json({ ok: false, error: "Nada para atualizar." }, { status: 400 });
}

await prisma.$transaction(transactions);

return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
  console.error("POST /api/organizador/events/update error:", err);
  return NextResponse.json(
    {
      ok: false,
      error: "Erro interno ao atualizar evento.",
    },
    { status: 500 },
  );
}
}

```

app/api/organizador/faturacao/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@lib/supabaseServer";
import { prisma } from "@lib/prisma";

```

```

import { computeReleaseAt, computeHold } from "@/domain/finance/payoutPolicy";
import { resolveConnectStatus } from "@/domain/finance/stripeConnectStatus";

export async function GET(_req: NextRequest) {
  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const memberships = await prisma.organizerMember.findMany({
    where: { userId: data.user.id, role: { in: ["OWNER", "CO_OWNER", "ADMIN"] } },
    select: { organizerId: true },
  });
  const organizerIds = memberships.map((m) => m.organizerId);
  if (organizerIds.length === 0) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

  const events = await prisma.event.findMany({
    where: { organizerId: { in: organizerIds } },
    select: {
      id: true,
      title: true,
      endsAt: true,
      isTest: true,
      organizer: { select: { stripeAccountId: true, stripeChargesEnabled: true, stripePayoutsEnabled: true } },
    },
  });

  const sales = await prisma.saleSummary.groupBy({
    by: ["eventId"],
    where: { eventId: { in: events.map((e) => e.id) } },
    _sum: { totalCents: true, netCents: true, platformFeeCents: true },
    _count: { _all: true },
  });

  const summaryPerEvent = events.map((evt) => {
    const agg = sales.find((s) => s.eventId === evt.id);
    const total = agg?._sum.totalCents ?? 0;
    const hold = computeHold(total, false);
    const releaseAt = computeReleaseAt(evt.endsAt);
    const connectStatus = resolveConnectStatus(
      evt.organizer?.stripeAccountId ?? null,
      evt.organizer?.stripeChargesEnabled ?? false,
      evt.organizer?.stripePayoutsEnabled ?? false,
    );
    return {
      eventId: evt.id,
      title: evt.title,
      isTest: evt.isTest,
      totalCents: total,
      netCents: agg?._sum.netCents ?? 0,
      platformFeeCents: agg?._sum.platformFeeCents ?? 0,
      countSales: agg?._count._all ?? 0,
      releaseAt,
      holdCents: hold.holdCents,
      holdReason: hold.reason,
      connectStatus,
    };
  });
}

const grandTotal = summaryPerEvent.reduce(
  (acc, e) => {
    acc.totalCents += e.totalCents;
    acc.netCents += e.netCents;
    acc.platformFeeCents += e.platformFeeCents;
    acc.countSales += e.countSales;
    acc.holdCents += e.holdCents;
    return acc;
  },
  { totalCents: 0, netCents: 0, platformFeeCents: 0, countSales: 0, holdCents: 0 },
);

const refundsCents = 0;
const disputesCents = 0;

return NextResponse.json(
  {
    ok: true,
    summary: { ...grandTotal, refundsCents, disputesCents },
    events: summaryPerEvent,
  }
);

```

```
  },
  { status: 200 },
);
}
```

app/api/organizador/finance/overview/route.ts

```
import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { TicketStatus } from "@prisma/client";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";
import { getStripeBaseFees } from "@/lib/platformSettings";

type Aggregate = {
  grossCents: number;
  netCents: number;
  feesCents: number;
  tickets: number;
};

export async function GET() {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const { organizer, membership } = await getActiveOrganizerForUser(user.id);

    if (!organizer || !membership || !isOrgAdminOrAbove(membership.role)) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    const events = await prisma.event.findMany({
      where: { organizerId: organizer.id },
      select: {
        id: true,
        title: true,
        slug: true,
        startsAt: true,
        status: true,
        payoutMode: true,
      },
      orderBy: { startsAt: "asc" },
    });
    const eventIds = events.map((e) => e.id);

    if (!eventIds.length) {
      return NextResponse.json(
        {
          ok: true,
          totals: { grossCents: 0, netCents: 0, feesCents: 0, tickets: 0, eventsWithSales: 0 },
          rolling: {
            last7: { grossCents: 0, netCents: 0, feesCents: 0, tickets: 0 },
            last30: { grossCents: 0, netCents: 0, feesCents: 0, tickets: 0 },
          },
          upcomingPayoutCents: 0,
          events: [],
        },
        { status: 200 }
      );
    }

    const now = new Date();
    const last7 = new Date(now.getTime() - 7 * 24 * 60 * 60 * 1000);
    const last30 = new Date(now.getTime() - 30 * 24 * 60 * 60 * 1000);
    const stripeBaseFees = await getStripeBaseFees();
    const estimateStripeFee = (amountCents: number) =>
      Math.max(
```

```

        0,
        Math.round((amountCents * (stripeBaseFees.feeBps ?? 0)) / 10_000) +
        (stripeBaseFees.feeFixedCents ?? 0),
    );
}

// Fonte preferencial: SaleSummary/SaleLine
const summaries = await prisma.saleSummary.findMany({
    where: {
        eventId: { in: eventIds },
    },
    select: {
        id: true,
        eventId: true,
        createdAt: true,
        subtotalCents: true,
        discountCents: true,
        platformFeeCents: true,
        stripeFeeCents: true,
        netCents: true,
        totalCents: true,
        lines: {
            select: { quantity: true },
        },
    },
},);
const totals: Aggregate = { grossCents: 0, netCents: 0, feesCents: 0, tickets: 0 };
const agg7: Aggregate = { grossCents: 0, netCents: 0, feesCents: 0, tickets: 0 };
const agg30: Aggregate = { grossCents: 0, netCents: 0, feesCents: 0, tickets: 0 };

const eventStats = new Map<
    number,
    Aggregate & {
        status?: string | null;
        startsAt?: Date | null;
    }
>();

const addTo = (target: Aggregate, gross: number, fees: number, net: number, qty: number) => {
    target.grossCents += gross;
    target.feesCents += fees;
    target.netCents += net;
    target.tickets += qty;
};

for (const s of summaries) {
    const qty = s.lines.reduce((q, l) => q + (l.quantity ?? 0), 0);
    const gross = s.subtotalCents ?? 0;
    const platformFee = s.platformFeeCents ?? 0;
    const totalCents = s.totalCents ?? gross;
    const stripeFee =
        s.stripeFeeCents != null && s.stripeFeeCents > 0
            ? s.stripeFeeCents
            : estimateStripeFee(totalCents);
    const totalFees = platformFee + stripeFee;
    const net =
        s.netCents != null && s.netCents >= 0
            ? s.netCents
            : Math.max(0, totalCents - totalFees);

    addTo(totals, gross, totalFees, net, qty);
    if (s.createdAt >= last30) addTo(agg30, gross, totalFees, net, qty);
    if (s.createdAt >= last7) addTo(agg7, gross, totalFees, net, qty);

    const current = eventStats.get(s.eventId) ?? {
        grossCents: 0,
        netCents: 0,
        feesCents: 0,
        tickets: 0,
        status: events.find((e) => e.id === s.eventId)?.status,
        startsAt: events.find((e) => e.id === s.eventId)?.startsAt ?? null,
    };
    addTo(current, gross, fees, net, qty);
    eventStats.set(s.eventId, current);
}

// Pré-carregar payment_events para alocar fees reais (ou estimadas) por PaymentIntent
const ticketIntents = await prisma.paymentEvent.findMany({

```

```

    where: { eventId: { in: eventIds } },
    select: { stripePaymentIntentId: true, stripeFeeCents: true, platformFeeCents: true, amountCents: true },
  });
const intentMap = new Map<
  string,
  { stripeFeeCents: number | null; platformFeeCents: number; amountCents: number | null; tickets: number }
>();
ticketIntents.forEach((pe) => {
  if (!pe.stripePaymentIntentId) return;
  intentMap.set(pe.stripePaymentIntentId, {
    stripeFeeCents: pe.stripeFeeCents ?? null,
    platformFeeCents: pe.platformFeeCents ?? 0,
    amountCents: pe.amountCents ?? null,
    tickets: 0,
  });
});
// Fallback se não existir SaleSummary: usar tickets legacy + fees de payment_events (ou estimativa)
if (summaries.length === 0) {
  const tickets = await prisma.ticket.findMany({
    where: {
      status: { in: [TicketStatus.ACTIVE, TicketStatus.USED] },
      eventId: { in: eventIds },
    },
    select: {
      pricePaid: true,
      platformFeeCents: true,
      purchasedAt: true,
      eventId: true,
      totalPaidCents: true,
      event: { select: { payoutMode: true } },
    },
  });
  for (const t of tickets) {
    const gross = t.pricePaid ?? 0;
    const platformFee = t.platformFeeCents ?? 0;
    const total = (t.totalPaidCents ?? gross) + platformFee;

    if (t.stripePaymentIntentId) {
      const entry = intentMap.get(t.stripePaymentIntentId) ?? null;
      if (entry) {
        entry.tickets += 1;
        intentMap.set(t.stripePaymentIntentId, entry);
      }
    }
  }
}

// Alocar fees dos payment_events (ou estimar) por intent
const feePerIntent = new Map<string, { stripeFeePerTicket: number; platformFeePerTicket: number }>();
intentMap.forEach((entry, intentId) => {
  if (entry.tickets <= 0) return;
  const stripeFee =
    entry.stripeFeeCents != null
      ? entry.stripeFeeCents
      : estimateStripeFee(entry.amountCents ?? 0);
  const stripePer = Math.round(stripeFee / entry.tickets);
  const platformPer = Math.round((entry.platformFeeCents ?? 0) / entry.tickets);
  feePerIntent.set(intentId, { stripeFeePerTicket: stripePer, platformFeePerTicket: platformPer });
});

for (const t of tickets) {
  const gross = t.pricePaid ?? 0;
  const platformFee = t.platformFeeCents ?? 0;
  const total = (t.totalPaidCents ?? gross) + platformFee;
  const feeAlloc = t.stripePaymentIntentId ? feePerIntent.get(t.stripePaymentIntentId) : null;
  const stripeFee =
    feeAlloc && feeAlloc.stripeFeePerTicket > 0
      ? feeAlloc.stripeFeePerTicket
      : estimateStripeFee(total);
  const totalFees = platformFee + stripeFee;
  const net = Math.max(0, total - totalFees);
  addTo(totals, gross, totalFees, net, 1);

  if (t.purchasedAt >= last30) addTo(agg30, gross, totalFees, net, 1);
  if (t.purchasedAt >= last7) addTo(agg7, gross, totalFees, net, 1);

  const current = eventStats.get(t.eventId) ?? {

```

```

        grossCents: 0,
        netCents: 0,
        feesCents: 0,
        tickets: 0,
        status: events.find((e) => e.id === t.eventId)?.status,
        startsAt: events.find((e) => e.id === t.eventId)?.startsAt ?? null,
    );
    addTo(current, gross, totalFees, net, 1);
    eventStats.set(t.eventId, current);
}
}

const eventsWithSales = Array.from(eventStats.keys()).length;
const upcomingPayoutCents = agg7.netCents;

return NextResponse.json(
{
    ok: true,
    totals: { ...totals, eventsWithSales },
    rolling: { last7: agg7, last30: agg30 },
    upcomingPayoutCents,
    events: events.map((ev) => {
        const stats = eventStats.get(ev.id) ?? {
            grossCents: 0,
            netCents: 0,
            feesCents: 0,
            tickets: 0,
        };
        return {
            ...ev,
            grossCents: stats.grossCents,
            netCents: stats.netCents,
            feesCents: stats.feesCents,
            ticketsSold: stats.tickets,
        };
    }),
},
{ status: 200 }
);
} catch (err) {
    console.error("[organizador/finance/overview]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/marketing/audience/summary/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@lib/prisma";
import { createSupabaseServer } from "@lib/supabaseServer";
import { getActiveOrganizerForUser } from "@lib/organizerContext";
import { TicketStatus } from "@prisma/client";

type SegmentKeys = "frequent" | "newLast60d" | "highSpenders" | "groups" | "dormant90d" | "local";

export async function GET(_req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error,
        } = await supabase.auth.getUser();

        if (error || !user) {
            return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
        }

        const { organizer } = await getActiveOrganizerForUser(user.id);

        if (!organizer) {
            return NextResponse.json({ ok: false, error: "NOT_ORGANIZER" }, { status: 403 });
        }

        const events = await prisma.event.findMany({
            where: { organizerId: organizer.id },
            select: { id: true, locationCity: true },
        });
    }
}

```

```

});

const eventIds = events.map((e) => e.id);

const mainCity =
  events
    .map((e) => e.locationCity?.trim())
    .filter(Boolean)
    .reduce<{ [city: string]: number }>((acc, city) => {
      if (!city) return acc;
      acc[city] = (acc[city] ?? 0) + 1;
      return acc;
    }, {});
const topCity = Object.entries(mainCity).sort((a, b) => b[1] - a[1])[0]?.[0] ?? null;

const tickets = await prisma.ticket.findMany({
  where: {
    status: { in: [TicketStatus.ACTIVE, TicketStatus.USED] },
    eventId: { in: eventIds },
  },
  select: {
    pricePaid: true,
    purchasedAt: true,
    userId: true,
    event: { select: { locationCity: true } },
    guestLink: { select: { guestEmail: true } },
  },
}, {});

const now = new Date();
const buyers = new Map<
  string,
  { count: number; total: number; first: Date; last: Date; city?: string | null }
>();
const userIds: Set<string> = new Set();

for (const ticket of tickets) {
  const key = ticket.userId ?? ticket.guestLink?.guestEmail ?? null;
  if (!key) continue;
  if (ticket.userId) userIds.add(ticket.userId);
  const stats = buyers.get(key) ?? {
    count: 0,
    total: 0,
    first: ticket.purchasedAt,
    last: ticket.purchasedAt,
    city: ticket.event.locationCity,
  };
  stats.count += 1;
  stats.total += ticket.pricePaid ?? 0;
  stats.first = stats.first < ticket.purchasedAt ? stats.first : ticket.purchasedAt;
  stats.last = stats.last > ticket.purchasedAt ? stats.last : ticket.purchasedAt;
  stats.city = stats.city ?? ticket.event.locationCity;
  buyers.set(key, stats);
}

const profiles = await prisma.profile.findMany({
  where: { id: { in: Array.from(userIds) } },
  select: { id: true, city: true },
});
const cityByUser = new Map(profiles.map((p) => [p.id, p.city ?? null]));

const segments: Record<SegmentKeys, number> = {
  frequent: 0,
  newLast60d: 0,
  highSpenders: 0,
  groups: 0,
  dormant90d: 0,
  local: 0,
};

const sixtyDaysAgo = new Date(now.getTime() - 60 * 24 * 60 * 60 * 1000);
const ninetyDaysAgo = new Date(now.getTime() - 90 * 24 * 60 * 60 * 1000);

for (const [key, stats] of buyers.entries()) {
  const buyerCity = stats.city;
  const profileCity = cityByUser.get(key) ?? buyerCity;
  if (stats.count >= 3) segments.frequent += 1;
  if (stats.first >= sixtyDaysAgo) segments.newLast60d += 1;
  if (stats.total >= 10000) segments.highSpenders += 1;
}

```

```

    if (stats.count >= 4) segments.groups += 1;
    if (stats.last <= ninetyDaysAgo) segments.dormant90d += 1;
    if (topCity && profileCity && profileCity.trim().toLowerCase() === topCity.trim().toLowerCase()) {
      segments.local += 1;
    }
  }

  return NextResponse.json({ ok: true, segments }, { status: 200 });
} catch (err) {
  console.error("[organizador/marketing/audience/summary]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/marketing/overview/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { TicketStatus } from "@prisma/client";

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const { organizer } = await getActiveOrganizerForUser(user.id);

    if (!organizer) {
      return NextResponse.json({ ok: false, error: "NOT_ORGANIZER" }, { status: 403 });
    }

    const now = new Date();
    const from = new Date(now.getTime() - 30 * 24 * 60 * 60 * 1000);

    const events = await prisma.event.findMany({
      where: { organizerId: organizer.id },
      select: {
        id: true,
        title: true,
        slug: true,
        startsAt: true,
        templateType: true,
        locationName: true,
        locationCity: true,
      },
    });
    const eventIds = events.map((e) => e.id);

    const tickets30d = await prisma.ticket.findMany({
      where: {
        status: { in: [TicketStatus.ACTIVE, TicketStatus.USED] },
        purchasedAt: { gte: from, lte: now },
        eventId: { in: eventIds },
      },
      select: {
        pricePaid: true,
        totalPaidCents: true,
        eventId: true,
        purchasedAt: true,
        guestLink: { select: { guestEmail: true } },
      },
    });

    const totalTickets = tickets30d.length;
    const totalRevenueCents = tickets30d.reduce((sum, t) => sum + (t.pricePaid ?? 0), 0);
    const guestTickets = tickets30d.filter((t) => t.guestLink?.guestEmail).length;
  }
}

```

```

const promoRepo = (prisma as unknown as {
  promoCode?: {
    findMany: typeof prisma.promoCode.findMany;
  };
}).promoCode;

let ticketsWithPromo = 0;
let marketingRevenueCents = 0;
let topPromo: { id: number; code: string; redemptionsCount: number; revenueCents: number } | undefined;

if (promoRepo) {
  const promoCodes = await promoRepo.findMany({
    where: {
      OR: [{ eventId: null }, { eventId: { in: eventIds } }],
    },
    include: {
      redemptions: {
        where: { usedAt: { gte: from, lte: now } },
      },
    },
  });
}

const avgPricePerEvent = new Map<number, number>();
const ticketSumByEvent = new Map<number, { sum: number; count: number }>();
for (const t of tickets30d) {
  const prev = ticketSumByEvent.get(t.eventId) ?? { sum: 0, count: 0 };
  ticketSumByEvent.set(t.eventId, { sum: prev.sum + (t.pricePaid ?? 0), count: prev.count + 1 });
}
ticketSumByEvent.forEach((val, key) => {
  avgPricePerEvent.set(key, val.count ? val.sum / val.count : 0);
});
const globalAvg = totalTickets ? totalRevenueCents / totalTickets : 0;

for (const promo of promoCodes) {
  const redemptionsCount = promo.redemptions.length;
  ticketsWithPromo += redemptionsCount;
  const estimatedPrice =
    promo.eventId && avgPricePerEvent.has(promo.eventId)
      ? avgPricePerEvent.get(promo.eventId) ?? 0
      : globalAvg;
  const estimatedRevenue = Math.round(redemptionsCount * estimatedPrice);
  marketingRevenueCents += estimatedRevenue;
  if (!topPromo || redemptionsCount > topPromo.redemptionsCount) {
    topPromo = {
      id: promo.id,
      code: promo.code,
      redemptionsCount,
      revenueCents: estimatedRevenue,
    };
  }
}

const capacityAgg = await prisma.ticketType.groupBy({
  by: ["eventId"],
  where: { eventId: { in: eventIds } },
  _sum: { totalQuantity: true },
});
const capacityMap = new Map<number, number>();
capacityAgg.forEach((row) => {
  capacityMap.set(row.eventId, row._sum.totalQuantity ?? 0);
});

const ticketStatsAll = await prisma.ticket.groupBy({
  by: ["eventId"],
  where: {
    status: { in: [TicketStatus.ACTIVE, TicketStatus.USED] },
    eventId: { in: eventIds },
  },
  _count: { _all: true },
  _sum: { pricePaid: true },
});
const statsMap = new Map<number, { tickets: number; revenueCents: number }>();
ticketStatsAll.forEach((s) => {
  statsMap.set(s.eventId, {
    tickets: s._count._all,
    revenueCents: s._sum.pricePaid ?? 0,
  });
});

```

```

    });

    return NextResponse.json(
    {
      ok: true,
      totalTickets,
      ticketsWithPromo,
      guestTickets,
      totalRevenueCents,
      marketingRevenueCents,
      topPromo: topPromo ?? null,
      events: events.map((ev) => ({
        ...ev,
        capacity: capacityMap.get(ev.id) ?? null,
        ticketsSold: statsMap.get(ev.id)?.tickets ?? 0,
        revenueCents: statsMap.get(ev.id)?.revenueCents ?? 0,
      })),
      {
        status: 200
      }
    );
  } catch (err) {
    console.error("[organizador/marketing/overview]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/me/route.ts

```

/* eslint-disable @typescript-eslint/no-explicit-any */
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getOrgTransferEnabled, getPlatformFees } from "@/lib/platformSettings";
import { isValidPhone, normalizePhone } from "@/lib/phone";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { Resend } from "resend";
import { cookies } from "next/headers";

const resendApiKey = process.env.RESEND_API_KEY;
const resendFromEmail = process.env.RESEND_FROM_EMAIL;
const resendClient = resendApiKey ? new Resend(resendApiKey) : null;

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const { data: { user }, error } = await supabase.auth.getUser();
    if (!user || error) {
      return NextResponse.json(
        {
          ok: false,
          error: "Não autenticado.",
          profile: null,
          organizer: null,
        },
        {
          status: 401
        }
      );
    }

    const profile = await prisma.profile.findUnique({
      where: { id: user.id },
    });
    if (!profile) {
      return NextResponse.json(
        {
          ok: false,
          error: "Perfil não encontrado.",
          profile: null,
          organizer: null,
        },
        {
          status: 404
        }
      );
    }

    const cookieStore = await cookies();
  
```

```

const cookieOrgId = cookieStore.get("orya_org")?.value;
const urlOrg = req.nextUrl.searchParams.get("org");
const forcedOrgId = urlOrg ? Number(urlOrg) : cookieOrgId ? Number(cookieOrgId) : undefined;
const { organizer, membership } = await getActiveOrganizerForUser(profile.id, {
  organizerId: Number.isFinite(forcedOrgId) ? forcedOrgId : undefined,
});
const [platformFees, orgTransferEnabled] = await Promise.all([getPlatformFees(), getOrgTransferEnabled()]);

const profilePayload = {
  id: profile.id,
  username: profile.username,
  fullName: profile.fullName,
  avatarUrl: profile.avatarUrl,
  bio: profile.bio,
  city: profile.city,
  favouriteCategories: profile.favouriteCategories,
  onboardingDone: profile.onboardingDone,
  roles: profile.roles,
};

const profileRoles = (profile.roles || [] as string[]);
const isAdmin = profileRoles.some((r) => r?.toLowerCase() === "admin");

const organizerPayload = organizer
? {
  id: organizer.id,
  displayName: organizer.displayName,
  username: organizer.username,
  stripeAccountId: organizer.stripeAccountId,
  status: organizer.status,
  stripeChargesEnabled: organizer.stripeChargesEnabled,
  stripePayoutsEnabled: organizer.stripePayoutsEnabled,
  feeMode: organizer.feeMode,
  platformFeeBps: organizer.platformFeeBps,
  platformFeeFixedCents: organizer.platformFeeFixedCents,
  businessName: organizer.businessName,
  entityType: organizer.entityType,
  city: organizer.city,
  payoutIban: organizer.payoutIban,
  language: (organizer as { language?: string | null }).language ?? "pt",
  publicListingEnabled: (organizer as { publicListingEnabled?: boolean | null }).publicListingEnabled ?? true,
  alertsEmail: (organizer as { alertsEmail?: string | null }).alertsEmail ?? null,
  alertsSalesEnabled: (organizer as { alertsSalesEnabled?: boolean | null }).alertsSalesEnabled ?? true,
  alertsPayoutEnabled: (organizer as { alertsPayoutEnabled?: boolean | null }).alertsPayoutEnabled ?? false,
  officialEmail: (organizer as { officialEmail?: string | null }).officialEmail ?? null,
  officialEmailVerifiedAt: (organizer as { officialEmailVerifiedAt?: Date | null }).officialEmailVerifiedAt ?? null,
  brandingAvatarUrl: (organizer as { brandingAvatarUrl?: string | null }).brandingAvatarUrl ?? null,
  brandingPrimaryColor: (organizer as { brandingPrimaryColor?: string | null }).brandingPrimaryColor ?? null,
  brandingSecondaryColor: (organizer as { brandingSecondaryColor?: string | null }).brandingSecondaryColor ?? null,
  organizationKind: (organizer as any).organizationKind ?? "PESSOA_SINGULAR",
  publicName: (organizer as { publicName?: string | null }).publicName ?? organizer.displayName ??
organizer.businessName ?? null,
  address: (organizer as { address?: string | null }).address ?? null,
  showAddressPublicly: (organizer as { showAddressPublicly?: boolean | null }).showAddressPublicly ?? false,
  padelDefaults: {
    shortName: (organizer as any).padelDefaultShortName ?? null,
    city: (organizer as any).padelDefaultCity ?? null,
    address: (organizer as any).padelDefaultAddress ?? null,
    courts: (organizer as any).padelDefaultCourts ?? 0,
    hours: (organizer as any).padelDefaultHours ?? null,
    ruleSetId: (organizer as any).padelDefaultRuleSetId ?? null,
    favoriteCategories: (organizer as any).padelFavoriteCategories ?? [],
  },
}
: null;

const profileStatus =
  organizer &&
  organizer.businessName &&
  organizer.entityType &&
  organizer.city &&
  user.email
  ? "OK"
  : "MISSING_CONTACT";

const lowerName = (organizer?.displayName ?? organizer?.username ?? "").toLowerCase();
const isPlatformAccount =
  isAdmin ||
  (organizer as { payoutMode?: string | null })?.payoutMode === "PLATFORM" ||

```

```

organizer?.organizationKind === "EMPRESA_MARCA" ||
lowerName === "orya" ||
lowerName.startsWith("orya "));
const paymentsStatus = organizer
? isPlatformAccount
? "READY"
: organizer.stripeAccountId
? organizer.stripeChargesEnabled && organizer.stripePayoutsEnabled
? "READY"
: "PENDING"
: "NO_STRIPE"
: "NO_STRIPE";

return NextResponse.json(
{
  ok: true,
  profile: profilePayload,
  organizer: organizerPayload,
  platformFees,
  orgTransferEnabled,
  contactEmail: user.email,
  profileStatus,
  paymentsStatus,
  paymentsMode: isPlatformAccount ? "PLATFORM" : "CONNECT",
  membershipRole: membership?.role ?? null,
},
{ status: 200 }
);
} catch (err) {
console.error("GET /api/organizador/me error:", err);
return NextResponse.json(
{
  ok: false,
  error: "Erro interno.",
  profile: null,
  organizer: null,
},
{ status: 500 }
);
}
}

export async function PATCH(req: NextRequest) {
try {
const supabase = await createSupabaseServer();
const {
  data: { user },
  error,
} = await supabase.auth.getUser();

if (!user || error) {
  return NextResponse.json({ ok: false, error: "Não autenticado." }, { status: 401 });
}

const body = await req.json().catch(() => null);
if (!body) {
  return NextResponse.json({ ok: false, error: "Payload inválido." }, { status: 400 });
}

const {
  displayName,
  businessName,
  entityType,
  city,
  payoutIban,
  fullName,
  contactPhone,
  language,
  publicListingEnabled,
  alertsEmail,
  alertsSalesEnabled,
  alertsPayoutEnabled,
  brandingAvatarUrl,
  brandingPrimaryColor,
  brandingSecondaryColor,
  organizationKind,
  publicName,
  address,
}

```

```

showAddressPublicly,
padelDefaultShortName,
padelDefaultCity,
padelDefaultAddress,
padelDefaultCourts,
padelDefaultHours,
padelDefaultRuleSetId,
padelFavoriteCategories,
} = body as Record<string, unknown>;

// Validação de telefone (opcional, mas consistente com checkout)
if (typeof contactPhone === "string" && contactPhone.trim()) {
  const phoneRaw = contactPhone.trim();
  if (!isValidPhone(phoneRaw)) {
    return NextResponse.json(
      { ok: false, error: "Telefone inválido. Usa um número real (podes incluir indicativo, ex.: +351...)." },
      { status: 400 },
    );
  }
}

// Garantir que existe organizer (via membership ou legacy userId)
const { organizer } = await getActiveOrganizerForUser(user.id, {
  roles: ["OWNER", "CO_OWNER", "ADMIN"],
});

if (!organizer) {
  return NextResponse.json({ ok: false, error: "Ainda não és organizador." }, { status: 403 });
}

const profileUpdates: Record<string, unknown> = {};
if (typeof fullName === "string") profileUpdates.fullName = fullName.trim() || null;
if (typeof city === "string") profileUpdates.city = city.trim() || null;
if (typeof contactPhone === "string") profileUpdates.contactPhone = normalizePhone(contactPhone.trim()) || null;
if (typeof alertsEmail === "string" && alertsEmail.trim()) {
  const email = alertsEmail.trim();
  const emailRegex = /^[\w\.-]+@[^\w\.-]+\.[\w\.-]+$/;
  if (!emailRegex.test(email)) {
    return NextResponse.json({ ok: false, error: "Email de alertas inválido." }, { status: 400 });
  }
}

const organizerUpdates: Record<string, unknown> = {};
const entityName = typeof displayName === "string" ? displayName.trim() : undefined;
const businessNameClean = typeof businessName === "string" ? businessName.trim() : undefined;
const publicNameInput = typeof publicName === "string" ? publicName.trim() : undefined;
const addressInput = typeof address === "string" ? address.trim() : undefined;
const showAddressPubliclyInput = typeof showAddressPublicly === "boolean" ? showAddressPublicly : undefined;

if (entityName !== undefined) organizerUpdates.displayName = entityName || null;
if (businessNameClean !== undefined) organizerUpdates.businessName = businessNameClean || null;
if (entityName !== undefined && businessNameClean === undefined) {
  organizerUpdates.businessName = entityName || null;
}
if (publicNameInput !== undefined) {
  const fallbackPublic =
    entityName ??
    businessNameClean ??
    organizer.displayName ??
    organizer.businessName ??
    null;
  organizerUpdates.publicName = publicNameInput || fallbackPublic || null;
}
if (addressInput !== undefined) organizerUpdates.address = addressInput || null;
if (showAddressPubliclyInput !== undefined) organizerUpdates.showAddressPublicly = showAddressPubliclyInput;
if (typeof entityType === "string") organizerUpdates.entityType = entityType.trim() || null;
if (typeof city === "string") organizerUpdates.city = city.trim() || null;
if (typeof payoutIban === "string") organizerUpdates.payoutIban = payoutIban.trim() || null;
if (typeof language === "string") {
  const lang = language.toLowerCase();
  organizerUpdates.language = lang === "en" ? "en" : "pt";
}
if (typeof publicListingEnabled === "boolean") organizerUpdates.publicListingEnabled = publicListingEnabled;
if (typeof alertsEmail === "string") organizerUpdates.alertsEmail = alertsEmail.trim() || null;
if (typeof alertsSalesEnabled === "boolean") organizerUpdates.alertsSalesEnabled = alertsSalesEnabled;
if (typeof alertsPayoutEnabled === "boolean") organizerUpdates.alertsPayoutEnabled = alertsPayoutEnabled;
if (typeof brandingAvatarUrl === "string") organizerUpdates.brandingAvatarUrl = brandingAvatarUrl.trim() || null;
if (typeof brandingPrimaryColor === "string") organizerUpdates.brandingPrimaryColor = brandingPrimaryColor.trim() || null;

```

```

if (typeof brandingSecondaryColor === "string")
  organizerUpdates.brandingSecondaryColor = brandingSecondaryColor.trim() || null;
if (typeof organizationKind === "string") {
  const kind = organizationKind.toUpperCase();
  const allowed = ["CLUBE_PADEL", "RESTAURANTE", "EMPRESA_EVENTOS", "ASSOCIACAO", "PESSOA_SINGULAR"];
  if (!allowed.includes(kind)) {
    return NextResponse.json(
      { ok: false, error: "organizationKind inválido. Usa CLUBE_PADEL, RESTAURANTE, EMPRESA_EVENTOS, ASSOCIACAO ou PESSOA_SINGULAR." },
      { status: 400 },
    );
  }
  organizerUpdates.organizationKind = kind;
}
if (typeof padelDefaultShortName === "string") {
  organizerUpdates.padelDefaultShortName = padelDefaultShortName.trim() || null;
}
if (typeof padelDefaultCity === "string") {
  organizerUpdates.padelDefaultCity = padelDefaultCity.trim() || null;
}
if (typeof padelDefaultAddress === "string") {
  organizerUpdates.padelDefaultAddress = padelDefaultAddress.trim() || null;
}
if (typeof padelDefaultHours === "string") {
  organizerUpdates.padelDefaultHours = padelDefaultHours.trim() || null;
}
if (typeof padelDefaultCourts === "number") {
  organizerUpdates.padelDefaultCourts = Math.max(0, Math.floor(padelDefaultCourts));
}
if (typeof padelDefaultRuleSetId === "number" && Number.isFinite(padelDefaultRuleSetId)) {
  organizerUpdates.padelDefaultRuleSetId = padelDefaultRuleSetId;
}
if (Array.isArray(padelFavoriteCategories)) {
  const nums = padelFavoriteCategories
    .map((v) => (typeof v === "number" && Number.isFinite(v) ? Math.floor(v) : null))
    .filter((v): v is number => v !== null);
  organizerUpdates.padelFavoriteCategories = nums;
}

if (Object.keys(profileUpdates).length > 0) {
  await prisma.profile.update({
    where: { id: user.id },
    data: profileUpdates,
  });
}

if (Object.keys(organizerUpdates).length > 0) {
  try {
    await prisma.organizer.update({
      where: { id: organizer.id },
      data: organizerUpdates,
    });
  } catch (err) {
    const code = (err as { code?: string })?.code;
    const message = err instanceof Error ? err.message : "";
    const missingColumn = code === "P2022" && message.toLowerCase().includes("public_name");
    if (!missingColumn) throw err;
    const fallbackData = { ...organizerUpdates };
    delete (fallbackData as Record<string, unknown>).publicName;
    await prisma.organizer.update({
      where: { id: organizer.id },
      data: fallbackData,
    });
  }
}

const verifiedOfficialEmail =
  organizer && (organizer as { officialEmailVerifiedAt?: Date | null })?.officialEmailVerifiedAt
  ? (organizer as { officialEmail?: string | null }).officialEmail ?? null
  : null;
const alertsTarget =
  verifiedOfficialEmail ??
  (typeof alertsEmail === "string" && alertsEmail.trim().length > 0 ? alertsEmail.trim() : organizer.alertsEmail);
const alertsSales = typeof alertsSalesEnabled === "boolean" ? alertsSalesEnabled : organizer.alertsSalesEnabled;
if (alertsTarget && alertsSales && resendClient && resendFromEmail) {
  try {
    await resendClient.emails.send({
      from: resendFromEmail,

```

```

        to: alertsTarget,
        subject: "Alertas de vendas ORYA ativados",
        text: "Passaste a receber alertas de vendas nesta caixa de email. Se não foste tu, desativa nas definições do organizador.",
    });
} catch (emailErr) {
    console.warn("[alerts] falha ao enviar email de alerta", emailErr);
}
}

return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
    console.error("PATCH /api/organizador/me error:", err);
    return NextResponse.json({ ok: false, error: "Erro interno." }, { status: 500 });
}
}
}

```

app/api/organizador/organizations/[id]/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { clearUsernameForOwner } from "@/lib/globalUsernames";

export async function DELETE(_req: NextRequest, context: { params: Promise<{ id: string }> }) {
    try {
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error,
        } = await supabase.auth.getUser();

        if (error || !user) {
            return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
        }

        const { id } = await context.params;
        const organizerId = Number(id);
        if (!organizerId || Number.isNaN(organizerId)) {
            return NextResponse.json({ ok: false, error: "INVALID_ORGANIZER_ID" }, { status: 400 });
        }

        const membership = await prisma.organizerMember.findUnique({
            where: { organizerId_userId: { organizerId, userId: user.id } },
        });

        if (!membership || membership.role !== "OWNER") {
            return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_DELETE" }, { status: 403 });
        }

        // Bloquear se existir algum bilhete ativo/usado associado a eventos desta org
        const hasSales = await prisma.ticket.count({
            where: {
                status: { in: ["ACTIVE", "USED"] },
                event: { organizerId },
            },
        });
        if (hasSales > 0) {
            return NextResponse.json(
                {
                    ok: false,
                    error: "Não é possível apagar: existem bilhetes vendidos nesta organização.",
                },
                { status: 400 },
            );
        }

        // Soft delete simples: marcar como SUSPENDED para já e limpar memberships
        await prisma.organizer.update({
            where: { id: organizerId },
            data: { status: "SUSPENDED" },
        });
        await prisma.organizerMember.deleteMany({ where: { organizerId } });
        await clearUsernameForOwner({ ownerType: "organizer", ownerId: organizerId });

        return NextResponse.json({ ok: true }, { status: 200 });
    }
}

```

```

    } catch (err) {
      console.error("[organizador/organizations/delete]", err);
      return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
    }
}

```

app/api/organizador/organizations/audit/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { OrganizerMemberRole } from "@prisma/client";

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const url = new URL(req.url);
    const organizerId = Number(url.searchParams.get("organizerId"));
    const limit = Math.min(Number(url.searchParams.get("limit")) ?? 50), 200;

    if (!organizerId || Number.isNaN(organizerId)) {
      return NextResponse.json({ ok: false, error: "INVALID_ORGANIZER" }, { status: 400 });
    }

    const membership = await prisma.organizerMember.findUnique({
      where: { organizerId_userId: { organizerId, userId: user.id } },
    });
    if (!membership || ![
      OrganizerMemberRole.OWNER,
      OrganizerMemberRole.CO_OWNER,
      OrganizerMemberRole.ADMIN
    ].includes(membership.role)) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    const auditModel = (prisma as any).organizationAuditLog;
    const logs = auditModel?.findMany(
      ? await auditModel.findMany({
        where: { organizerId },
        orderBy: { createdAt: "desc" },
        take: limit,
      })
      : []
    );

    return NextResponse.json({ ok: true, items: logs }, { status: 200 });
  } catch (err) {
    console.error("[organizador/audit]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
  }
}

```

app/api/organizador/organizations/leave/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }
  }
}

```

```

const body = await req.json().catch(() => null);
const organizerId = Number(body?.organizerId);
if (!organizerId || Number.isNaN(organizerId)) {
  return NextResponse.json({ ok: false, error: "INVALID_ORGANIZER_ID" }, { status: 400 });
}

const membership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: user.id } },
});

if (!membership) {
  return NextResponse.json({ ok: false, error: "NOT_MEMBER" }, { status: 403 });
}

if (membership.role === "OWNER") {
  const otherOwners = await prisma.organizerMember.count({
    where: {
      organizerId,
      role: "OWNER",
      userId: { not: user.id },
    },
  });
  if (otherOwners === 0) {
    return NextResponse.json(
      {
        ok: false,
        error: "És o último Owner desta organização. Transfere a propriedade antes de sair.",
      },
      { status: 400 },
    );
  }
}

await prisma.organizerMember.delete({
  where: { organizerId_userId: { organizerId, userId: user.id } },
});

return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
  console.error("[organizador/organizations/leave]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/organizations/members/invites/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole } from "@prisma/client";
import { createSupabaseServer } from "@lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { resolveUserIdentity } from "@/lib/userResolver";
import { createNotification } from "@/lib/notifications";
import { NotificationType } from "@prisma/client";
import { canManageMembers, isOrgAdminOrAbove, isOrgOwner } from "@/lib/organizerPermissions";
import { ensureUserIsOrganizer, setSoleOwner } from "@/lib/organizerRoles";
import { sanitizeProfileVisibility } from "@/lib/profileVisibility";

const INVITE_EXPIRY_DAYS = 14;

type InviteStatus = "PENDING" | "EXPIRED" | "ACCEPTED" | "DECLINED" | "CANCELLED";

const serializeInvite = (
  invite: {
    id: string;
    organizerId: number;
    targetIdentifier: string;
    targetUserId: string | null;
    role: string;
    token: string;
    expiresAt: Date;
    acceptedAt: Date | null;
    declinedAt: Date | null;
    cancelledAt: Date | null;
    createdAt: Date;
    invitedBy?: {

```

```

id: string;
username: string | null;
fullName: string | null;
avatarUrl: string | null;
visibility?: string | null;
isDeleted?: boolean | null;
} | null;
targetUser?: {
  id: string;
  username: string | null;
  fullName: string | null;
  avatarUrl: string | null;
  visibility?: string | null;
  isDeleted?: boolean | null;
} | null;
},
viewer?: { id: string; username?: string | null; email?: string | null },
) => {
  const now = Date.now();
  const isExpired = !invite.expiresAt && invite.expiresAt.getTime() < now;
  const status: InviteStatus = invite.cancelledAt
    ? "CANCELLED"
    : invite.acceptedAt
    ? "ACCEPTED"
    : invite.declinedAt
    ? "DECLINED"
    : isExpired
    ? "EXPIRED"
    : "PENDING";

  const normalizedTarget = invite.targetIdentifier.toLowerCase();
  const canRespond =
    !!viewer &&
    (invite.targetUserId === viewer.id ||
     (viewer.username && viewer.username.toLowerCase() === normalizedTarget) ||
     (viewer.email && viewer.email.toLowerCase() === normalizedTarget));

  return {
    id: invite.id,
    organizerId: invite.organizerId,
    role: invite.role,
    targetIdentifier: invite.targetIdentifier,
    targetUserId: invite.targetUserId,
    status,
    expiresAt: invite.expiresAt?.toISOString() ?? null,
    createdAt: invite.createdAt?.toISOString(),
    acceptedAt: invite.acceptedAt?.toISOString() ?? null,
    declinedAt: invite.declinedAt?.toISOString() ?? null,
    cancelledAt: invite.cancelledAt?.toISOString() ?? null,
    invitedBy: sanitizeProfileVisibility(invite.invitedBy ?? null, viewer?.id),
    targetUser: sanitizeProfileVisibility(invite.targetUser ?? null, viewer?.id),
    canRespond,
  };
};

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const profile = await prisma.profile.findUnique({
      where: { id: user.id },
      select: { username: true },
    });

    const url = new URL(req.url);
    const organizerIdRaw = url.searchParams.get("organizerId");
    const eventIdRaw = url.searchParams.get("eventId");
    let organizerId = organizerIdRaw ? Number(organizerIdRaw) : null;
    const limit = Math.min(Number(url.searchParams.get("limit") ?? 200), 500);
    if (!organizerId && eventIdRaw) {
  
```

```

const eventId = Number(eventIdRaw);
if (eventId && !Number.isNaN(eventId)) {
  const ev = await prisma.event.findUnique({
    where: { id: eventId },
    select: { organizerId: true },
  });
  organizerId = ev?.organizerId ?? null;
}
}

if (!organizerId || Number.isNaN(organizerId)) {
  return NextResponse.json({ ok: false, error: "INVALID_ORGANIZER_ID" }, { status: 400 });
}

const membership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: user.id } },
});
const isManager = membership ? isOrgAdminOrAbove(membership.role) : false;

const viewerEmail = user.email?.toLowerCase() ?? null;
const viewerUsername = profile?.username ?? null;

// Se não é manager, só pode ver convites dirigidos a si
if (!isManager) {
  const inviteForUser = await prisma.organizerMemberInvite.findFirst({
    where: {
      organizerId,
      cancelledAt: null,
      acceptedAt: null,
      OR: [
        { targetUserId: user.id },
        ...(viewerEmail ? [{ targetIdentifier: { equals: viewerEmail, mode: "insensitive" } }] : []),
        ...(viewerUsername
          ? [{ targetIdentifier: { equals: viewerUsername, mode: "insensitive" as const } }]
          : []),
      ],
    },
  });
  if (!inviteForUser) {
    return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
  }
}

const invites = await prisma.organizerMemberInvite.findMany({
  where: {
    organizerId,
    ...(isManager
      ? {}
      : {
        OR: [
          { targetUserId: user.id },
          ...(viewerEmail ? [{ targetIdentifier: { equals: viewerEmail, mode: "insensitive" } }] : []),
          ...(viewerUsername
            ? [{ targetIdentifier: { equals: viewerUsername, mode: "insensitive" as const } }]
            : []),
        ],
      },
  },
  include: {
    invitedBy: { select: { id: true, username: true, fullName: true, avatarUrl: true, visibility: true, isDeleted: true } },
    targetUser: {
      select: {
        id: true,
        username: true,
        fullName: true,
        avatarUrl: true,
        visibility: true,
        isDeleted: true,
      },
    },
    orderBy: { createdAt: "desc" },
    take: limit,
  });
}

const viewer = { id: user.id, username: viewerUsername, email: viewerEmail };
return NextResponse.json(
{
  ok: true,

```

```

        viewerRole: membership?.role ?? null,
        organizerId,
        items: invites.map((inv) => serializeInvite(inv, viewer)),
    },
    { status: 200 },
);
} catch (err) {
    console.error("[organizador/members/invites][GET]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

export async function POST(req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error,
        } = await supabase.auth.getUser();

        if (error || !user) {
            return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
        }

        const body = await req.json().catch(() => null);
        const organizerId = Number(body?.organizerId);
        const identifier = typeof body?.identifier === "string" ? body.identifier.trim() : null;
        const roleRaw = typeof body?.role === "string" ? body.role.toUpperCase() : null;

        if (!organizerId || Number.isNaN(organizerId) || !identifier || !roleRaw) {
            return NextResponse.json({ ok: false, error: "INVALID_PAYLOAD" }, { status: 400 });
        }

        if (!Object.values(OrganizerMemberRole).includes(roleRaw as OrganizerMemberRole)) {
            return NextResponse.json({ ok: false, error: "INVALID_ROLE" }, { status: 400 });
        }

        const membership = await prisma.organizerMember.findUnique({
            where: { organizerId_userId: { organizerId, userId: user.id } },
        });
        if (!membership || !canManageMembers(membership.role, null, roleRaw as OrganizerMemberRole)) {
            return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
        }
        if (roleRaw === "OWNER" && !isOrgOwner(membership.role)) {
            return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_SET_OWNER" }, { status: 403 });
        }

        const resolved = await resolveUserIdentifier(identifier);
        const viewerProfile = await prisma.profile.findUnique({
            where: { id: user.id },
            select: { username: true },
        });
        const targetUserId = resolved?.userId ?? null;

        // Bloquear convites para quem já é membro
        if (targetUserId) {
            const existingMember = await prisma.organizerMember.findUnique({
                where: { organizerId_userId: { organizerId, userId: targetUserId } },
            });
            if (existingMember) {
                return NextResponse.json(
                    { ok: false, error: "Utilizador já é membro desta organização." },
                    { status: 400 },
                );
            }
        }

        const normalizedIdentifier = identifier.toLowerCase();
        await prisma.organizerMemberInvite.updateMany({
            where: {
                organizerId,
                acceptedAt: null,
                cancelledAt: null,
                declinedAt: null,
                OR: [
                    { targetIdentifier: { equals: normalizedIdentifier, mode: "insensitive" } },
                    ...(targetUserId ? [{ targetUserId }] : []),
                ],
            },
        });
    }
}

```

```

    },
    data: { cancelledAt: new Date() },
  });

const expiresAt = new Date(Date.now() + INVITE_EXPIRY_DAYS * 24 * 60 * 60 * 1000);

const invite = await prisma.organizerMemberInvite.create({
  data: {
    organizerId,
    invitedByUserId: user.id,
    targetIdentifier: identifier,
    targetUserId,
    role: roleRaw as OrganizerMemberRole,
    token: crypto.randomUUID(),
    expiresAt,
  },
  include: {
    invitedBy: { select: { id: true, username: true, fullName: true, avatarUrl: true, visibility: true } },
    targetUser: {
      select: {
        id: true,
        username: true,
        fullName: true,
        avatarUrl: true,
        visibility: true,
      },
    },
    organizer: { select: { id: true, displayName: true } },
  },
});
}

const viewer = {
  id: user.id,
  username: viewerProfile?.username ?? null,
  email: user.email ? user.email.toLowerCase() : null,
};

// Notificação para o alvo (se user conhecido)
if (targetUserId) {
  await createNotification({
    userId: targetUserId,
    type: NotificationType.ORGANIZER_INVITE,
    title: "Convite para organização",
    body: `Foste convidado para a organização ${invite.organizer?.displayName ?? "ORYA"}.`,
    ctaUrl: `/organizador/organizations`,
    ctaLabel: "Ver convites",
    senderVisibility: "PUBLIC",
    fromUserId: user.id,
    organizerId,
    inviteId: invite.id,
    payload: {
      organizerId,
      role: roleRaw,
      actor: { id: user.id, username: viewerProfile?.username },
    },
  }).catch((err) => console.warn("[notification][invite] falhou", err));
}

return NextResponse.json({ ok: true, invite: serializeInvite(invite, viewer) }, { status: 201 });
} catch (err) {
  console.error("[organizador/members/invites] [POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

export async function PATCH(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const body = await req.json().catch(() => null);
  }
}

```

```

const organizerId = Number(body?.organizerId);
const inviteId = typeof body?.inviteId === "string" ? body.inviteId : null;
const tokenFromBody = typeof body?.token === "string" ? body.token : null;
const action = typeof body?.action === "string" ? body.action.toUpperCase() : null;

if (!organizerId || Number.isNaN(organizerId) || !action) {
  return NextResponse.json({ ok: false, error: "INVALID_PAYLOAD" }, { status: 400 });
}

const membership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: user.id } },
});
const isManager = membership ? isOrgAdminOrAbove(membership.role) : false;

if (!inviteId && !tokenFromBody) {
  return NextResponse.json({ ok: false, error: "NEED_INVITE_ID_OR_TOKEN" }, { status: 400 });
}

const invite = await prisma.organizerMemberInvite.findFirst({
  where: {
    organizerId,
    ...(inviteId ? { id: inviteId } : {}),
    ...(tokenFromBody ? { token: tokenFromBody } : {}),
  },
  include: {
    invitedBy: { select: { id: true, username: true, fullName: true, avatarUrl: true, visibility: true } },
    targetUser: {
      select: {
        id: true,
        username: true,
        fullName: true,
        avatarUrl: true,
        visibility: true,
      },
    },
  },
});
if (!invite) {
  return NextResponse.json({ ok: false, error: "INVITE_NOT_FOUND" }, { status: 404 });
}

const viewerProfile = await prisma.profile.findUnique({
  where: { id: user.id },
  select: { username: true, roles: true },
});
const viewerEmail = user.email?.toLowerCase() ?? null;
const viewerUsername = viewerProfile?.username ?? null;

const normalizedTarget = invite.targetIdentifier.toLowerCase();
const isTargetUser =
  invite.targetUserId === user.id ||
  (viewerEmail && normalizedTarget === viewerEmail) ||
  (viewerUsername && normalizedTarget === viewerUsername.toLowerCase());
const matchesToken = tokenFromBody ? invite.token === tokenFromBody : false;

const isPending = !invite.acceptedAt && !invite.declinedAt && !invite.cancelledAt;
const isExpired = invite.expiresAt && invite.expiresAt.getTime() < Date.now();

const isOwnerManager = membership?.role === "OWNER";

if (action === "CANCEL") {
  if (!isManager) {
    return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
  }
  if (invite.role === "OWNER" && !isOwnerManager) {
    return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_CANCEL_OWNER_INVITE" }, { status: 403 });
  }
  await prisma.organizerMemberInvite.update({
    where: { id: invite.id },
    data: { cancelledAt: new Date() },
  });
} else if (action === "RESEND") {
  if (!isManager) {
    return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
  }
  if (invite.role === "OWNER" && !isOwnerManager) {
    return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_SET_OWNER" }, { status: 403 });
  }
}

```

```

    });

    const resolved = await resolveUserIdentity(invite.targetIdentifier);
    await prisma.organizerMemberInvite.update({
      where: { id: invite.id },
      data: {
        cancelledAt: null,
        declinedAt: null,
        acceptedAt: null,
        targetUserId: resolved?.userId ?? invite.targetUserId,
        token: crypto.randomUUID(),
        expiresAt: new Date(Date.now() + INVITE_EXPIRY_DAYS * 24 * 60 * 60 * 1000),
      },
    });
  } else if (action === "ACCEPT") {
    if (!isPending) {
      return NextResponse.json({ ok: false, error: "INVITE_NOT_PENDING" }, { status: 400 });
    }
    if (isExpired) {
      return NextResponse.json({ ok: false, error: "INVITE_EXPIRED" }, { status: 400 });
    }
    if (!isTargetUser && !matchesToken) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    const role = invite.role as OrganizerMemberRole;
    await prisma.$transaction(async (tx) => {
      const currentMember = await tx.organizerMember.findUnique({
        where: { organizerId_userId: { organizerId, userId: user.id } },
      });

      if (currentMember?.role === "OWNER" && role !== "OWNER") {
        const otherOwners = await tx.organizerMember.count({
          where: { organizerId, role: "OWNER", userId: { not: user.id } },
        });
        if (otherOwners === 0) {
          throw new Error("LAST_OWNER_BLOCK");
        }
      }

      if (role === "OWNER") {
        await setSoleOwner(tx, organizerId, user.id, invite.invitedByUserId);
      } else {
        await tx.organizerMember.upsert({
          where: { organizerId_userId: { organizerId, userId: user.id } },
          update: { role },
          create: {
            organizerId,
            userId: user.id,
            role,
            invitedByUserId: invite.invitedByUserId,
          },
        });
      }

      await tx.organizerMemberInvite.update({
        where: { id: invite.id },
        data: {
          acceptedAt: new Date(),
          declinedAt: null,
          cancelledAt: null,
          targetUserId: user.id,
        },
      });

      await ensureUserIsOrganizer(tx, user.id);
    }).catch((err: unknown) => {
      if (err instanceof Error && err.message === "LAST_OWNER_BLOCK") {
        throw err;
      }
      throw err;
    });
  } else if (action === "DECLINE") {
    if (!isTargetUser && !matchesToken && !isManager) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }
    if (!isPending) {
      return NextResponse.json({ ok: false, error: "INVITE_NOT_PENDING" }, { status: 400 });
    }
  }
}

```

```

await prisma.organizerMemberInvite.update({
  where: { id: invite.id },
  data: { declinedAt: new Date(), acceptedAt: null, cancelledAt: null, targetUserId: user.id },
});
} else {
  return NextResponse.json({ ok: false, error: "UNKNOWN_ACTION" }, { status: 400 });
}

const updated = await prisma.organizerMemberInvite.findUnique({
  where: { id: invite.id },
  include: {
    invitedBy: { select: { id: true, username: true, fullName: true, avatarUrl: true } },
    targetUser: {
      select: {
        id: true,
        username: true,
        fullName: true,
        avatarUrl: true,
      },
    },
  },
});

if (!updated) {
  return NextResponse.json({ ok: false, error: "INVITE_NOT_FOUND" }, { status: 404 });
}

if (action === "ACCEPT") {
  // Evitar propagação de erros de transação
}

const viewer = { id: user.id, username: viewerUsername, email: viewerEmail };
return NextResponse.json({ ok: true, invite: serializeInvite(updated, viewer) }, { status: 200 });
} catch (err: unknown) {
  if (err instanceof Error && err.message === "LAST_OWNER_BLOCK") {
    return NextResponse.json(
      { ok: false, error: "Não podes remover o último Owner. Adiciona outro Owner antes." },
      { status: 400 },
    );
  }
  console.error("[organizador/members/invites][PATCH]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/organizations/members/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { OrganizerMemberRole } from "@prisma/client";
import { canManageMembers, isOrgOwner } from "@/lib/organizerPermissions";
import { setSoleOwner } from "@/lib/organizerRoles";
import { recordOrganizationAuditSafe } from "@/lib/organizationAudit";

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const url = new URL(req.url);
    const organizerIdRaw = url.searchParams.get("organizerId");
    const eventIdRaw = url.searchParams.get("eventId");
    let organizerId = organizerIdRaw ? Number(organizerIdRaw) : null;

    if (!organizerId && eventIdRaw) {
      const eventId = Number(eventIdRaw);
      if (eventId && !Number.isNaN(eventId)) {
        const ev = await prisma.event.findUnique({

```

```

        where: { id: eventId },
        select: { organizerId: true },
      });
    organizerId = ev?.organizerId ?? null;
  }
}

if (!organizerId || Number.isNaN(organizerId)) {
  return NextResponse.json({ ok: false, error: "INVALID_ORGANIZER_ID" }, { status: 400 });
}

const limit = Math.min(Number(url.searchParams.get("limit") ?? 200), 500);

// Qualquer membro pode consultar; ações ficam restritas por role
const callerMembership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: user.id } },
});
if (!callerMembership) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}

const members = await prisma.organizerMember.findMany({
  where: { organizerId, user: { isDeleted: false } },
  include: {
    user: {
      select: {
        id: true,
        fullName: true,
        username: true,
        avatarUrl: true,
        visibility: true,
      },
    },
  },
  orderBy: { createdAt: "asc" },
  take: limit,
});

const items = members.map((m) => {
  const visibility = (m.user as { visibility?: string | null })?.visibility ?? "PUBLIC";
  const isSelf = m.userId === user.id;
  const isPrivate = visibility === "PRIVATE" && !isSelf;

  return {
    userId: m.userId,
    role: m.role,
    invitedByUserId: m.invitedByUserId,
    createdAt: m.createdAt,
    fullName: isPrivate
      ? null
      : (m.user && "fullName" in m.user ? (m.user as { fullName?: string | null }).fullName ?? null : null),
    username: m.user && "username" in m.user ? (m.user as { username?: string | null }).username ?? null : null,
    avatarUrl: m.user && "avatarUrl" in m.user ? (m.user as { avatarUrl?: string | null }).avatarUrl ?? null : null,
    email: isPrivate ? null : null,
    visibility,
  };
});

return NextResponse.json(
  { ok: true, items, viewerRole: callerMembership.role, organizerId },
  { status: 200 },
);
} catch (err) {
  console.error("[organizador/members][GET]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

export async function PATCH(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }
  }
}

```

```

}

const body = await req.json().catch(() => null);
const organizerId = Number(body?.organizerId);
const targetUserId = typeof body?.userId === "string" ? body.userId : null;
const role = typeof body?.role === "string" ? body.role.toUpperCase() : null;

if (!organizerId || Number.isNaN(organizerId) || !targetUserId || !role) {
  return NextResponse.json({ ok: false, error: "INVALID_PAYLOAD" }, { status: 400 });
}
if (!Object.values(OrganizerMemberRole).includes(role as OrganizerMemberRole)) {
  return NextResponse.json({ ok: false, error: "INVALID_ROLE" }, { status: 400 });
}

const callerMembership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: user.id } },
});
if (!callerMembership) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}
const callerRole = callerMembership.role as OrganizerMemberRole | null;

const targetMembership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: targetUserId } },
});
if (!targetMembership) {
  return NextResponse.json({ ok: false, error: "NOT_MEMBER" }, { status: 404 });
}

if (!canManageMembers(callerRole, targetMembership.role, role as OrganizerMemberRole)) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}

if (role === "OWNER" && !isOrgOwner(callerRole)) {
  return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_SET_OWNER" }, { status: 403 });
}

if (targetMembership.role === "OWNER" && role !== "OWNER") {
  const otherOwners = await prisma.organizerMember.count({
    where: {
      organizerId,
      role: "OWNER",
      userId: { not: targetUserId },
    },
  });
  if (otherOwners === 0) {
    return NextResponse.json(
      { ok: false, error: "Não podes remover o Último Owner." },
      { status: 400 },
    );
  }
}

// Se promover para OWNER, garantir que fica único (demovendo outros para CO_OWNER)
if (role === "OWNER") {
  await prisma.$transaction(async (tx) => setSoleOwner(tx, organizerId, targetUserId));
  await recordOrganizationAuditSafe({
    organizerId,
    actorUserId: user.id,
    action: "OWNER_PROMOTED",
    toUserId: targetUserId,
    metadata: { via: "members.patch" },
  });
  return NextResponse.json({ ok: true }, { status: 200 });
}

// Bloqueia que o único owner se despromova a si próprio
if (targetMembership.role === "OWNER" && targetUserId === user.id && role !== "OWNER") {
  const otherOwners = await prisma.organizerMember.count({
    where: { organizerId, role: "OWNER", userId: { not: user.id } },
  });
  if (otherOwners === 0) {
    return NextResponse.json(
      { ok: false, error: "Garante outro Owner antes de descer o teu papel." },
      { status: 400 },
    );
  }
}
}

```

```

await prisma.organizerMember.update({
  where: { organizerId_userId: { organizerId, userId: targetUserId } },
  data: { role: role as OrganizerMemberRole },
});

if (targetMembership.role === "OWNER" && role !== "OWNER") {
  await recordOrganizationAuditSafe({
    organizerId,
    actorUserId: user.id,
    action: "OWNER_DEMOTED",
    fromUserId: targetUserId,
    metadata: { newRole: role },
  });
}

return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
  console.error("[organizador/members][PATCH]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

export async function DELETE(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const url = new URL(req.url);
    const organizerId = Number(url.searchParams.get("organizerId"));
    const targetUserId = url.searchParams.get("userId");

    if (!organizerId || Number.isNaN(organizerId) || !targetUserId) {
      return NextResponse.json({ ok: false, error: "INVALID_PARAMS" }, { status: 400 });
    }

    const callerMembership = await prisma.organizerMember.findUnique({
      where: { organizerId_userId: { organizerId, userId: user.id } },
    });
    if (!callerMembership) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }
    const callerRole = callerMembership.role as OrganizerMemberRole | null;

    const targetMembership = await prisma.organizerMember.findUnique({
      where: { organizerId_userId: { organizerId, userId: targetUserId } },
    });
    if (!targetMembership) {
      return NextResponse.json({ ok: false, error: "NOT_MEMBER" }, { status: 404 });
    }

    if (!canManageMembers(callerRole, targetMembership.role, targetMembership.role)) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    if (targetMembership.role === "OWNER") {
      if (!isOrgOwner(callerRole)) {
        return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_REMOVE_OWNER" }, { status: 403 });
      }

      const otherOwners = await prisma.organizerMember.count({
        where: {
          organizerId,
          role: "OWNER",
          userId: { not: targetUserId },
        },
      });
      if (otherOwners === 0) {
        return NextResponse.json(
          { ok: false, error: "Não podes remover o Último Owner." },
          { status: 400 },
        );
      }
    }
  }
}

```

```

        );
    }
}

await prisma.organizerMember.delete({
  where: { organizerId_userId: { organizerId, userId: targetUserId } },
});

if (targetMembership.role === "OWNER") {
  await recordOrganizationAuditSafe({
    organizerId,
    actorUserId: user.id,
    action: "OWNER_REMOVED",
    fromUserId: targetUserId,
    metadata: { via: "members.delete" },
  });
}

return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
  console.error("[organizador/members][DELETE]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/organizations/members/upsert/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { OrganizerMemberRole } from "@prisma/client";
import { resolveUserIdentifer } from "@/lib/userResolver";
import { canManageMembers, isOrgOwner } from "@/lib/organizerPermissions";
import { ensureUserIsOrganizer, setSoleOwner } from "@/lib/organizerRoles";
import { recordOrganizationAuditSafe } from "@/lib/organizationAudit";

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const body = await req.json().catch(() => null);
    const organizerId = Number(body?.organizerId);
    const targetIdentifier = typeof body?.userId === "string" ? body.userId.trim() : null;
    const roleRaw = typeof body?.role === "string" ? body.role.toUpperCase() : null;

    if (!organizerId || Number.isNaN(organizerId) || !targetIdentifier || !roleRaw) {
      return NextResponse.json({ ok: false, error: "INVALID_PAYLOAD" }, { status: 400 });
    }
    if (!Object.values(OrganizerMemberRole).includes(roleRaw as OrganizerMemberRole)) {
      return NextResponse.json({ ok: false, error: "INVALID_ROLE" }, { status: 400 });
    }

    const callerMembership = await prisma.organizerMember.findUnique({
      where: { organizerId_userId: { organizerId, userId: user.id } },
    });
    if (!callerMembership) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }
    const callerRole = callerMembership.role as OrganizerMemberRole | null;

    const resolved = await resolveUserIdentifer(targetIdentifier);
    if (!resolved) {
      return NextResponse.json({ ok: false, error: "Utilizador não encontrado." }, { status: 404 });
    }

    const targetUserId = resolved.userId;
    const role = roleRaw as OrganizerMemberRole;
  }
}

```

```

const targetMembership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: targetUserId } },
});

if (!canManageMembers(callerRole, targetMembership?.role ?? null, role)) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}

if (role === "OWNER" && !isOrgOwner(callerRole)) {
  return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_SET_OWNER" }, { status: 403 });
}

// Se for a removendo o último owner, bloquear
if (role !== "OWNER" && targetMembership?.role === "OWNER") {
  const otherOwners = await prisma.organizerMember.count({
    where: { organizerId, role: "OWNER", userId: { not: targetUserId } },
  });
  if (otherOwners === 0) {
    return NextResponse.json(
      { ok: false, error: "Não podes remover o último Owner." },
      { status: 400 },
    );
  }
}

await prisma.$transaction(async (tx) => {
  if (role === "OWNER") {
    await setSoleOwner(tx, organizerId, targetUserId, user.id);
  } else {
    await tx.organizerMember.upsert({
      where: { organizerId_userId: { organizerId, userId: targetUserId } },
      update: { role },
      create: { organizerId, userId: targetUserId, role, invitedByUserId: user.id },
    });
  }

  await ensureUserIsOrganizer(tx, targetUserId);
});

if (role === "OWNER") {
  await recordOrganizationAuditSafe({
    organizerId,
    actorUserId: user.id,
    action: "OWNER_PROMOTED",
    toUserId: targetUserId,
    metadata: { via: "members.upsert" },
  });
}

return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
  console.error("[organizador/members/upsert]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/organizations/owner/confirm/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole } from "@prisma/client";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { getOrgTransferEnabled } from "@/lib/platformSettings";
import { setSoleOwner } from "@/lib/organizerRoles";
import { recordOrganizationAudit } from "@/lib/organizationAudit";
import { supabaseAdmin } from "@/lib/supabaseAdmin";
import { sendEmail } from "@/lib/resendClient";

export async function POST(req: NextRequest) {
  try {
    const ownerTransferModel = (prisma as any).organizerOwnerTransfer;
    if (!ownerTransferModel?.findUnique) {
      return NextResponse.json({ ok: false, error: "OWNER_TRANSFER_UNAVAILABLE" }, { status: 501 });
    }

    const supabase = await createSupabaseServer();
  
```

```

const {
  data: { user },
  error,
} = await supabase.auth.getUser();

if (error || !user) {
  return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
}

const transferEnabled = await getOrgTransferEnabled();
if (!transferEnabled) {
  return NextResponse.json({ ok: false, error: "ORG_TRANSFER_DISABLED" }, { status: 403 });
}

const body = await req.json().catch(() => null);
const token = typeof body?.token === "string" ? body.token.trim() : null;
if (!token) {
  return NextResponse.json({ ok: false, error: "INVALID_TOKEN" }, { status: 400 });
}

const transfer = await ownerTransferModel.findUnique({
  where: { token },
});
if (!transfer) {
  return NextResponse.json({ ok: false, error: "TRANSFER_NOT_FOUND" }, { status: 404 });
}

if (transfer.status !== "PENDING") {
  return NextResponse.json({ ok: false, error: "TRANSFER_NOT_PENDING" }, { status: 400 });
}

if (transfer.toUserId !== user.id) {
  return NextResponse.json({ ok: false, error: "TOKEN_USER_MISMATCH" }, { status: 403 });
}

const now = new Date();
if (transfer.expiresAt && transfer.expiresAt.getTime() < now.getTime()) {
  await ownerTransferModel.update({
    where: { id: transfer.id },
    data: { status: "EXPIRED", cancelledAt: now },
  });
  return NextResponse.json({ ok: false, error: "TRANSFER_EXPIRED" }, { status: 400 });
}

const fromMembership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId: transfer.organizerId, userId: transfer.fromUserId } },
});
if (!fromMembership || fromMembership.role !== OrganizerMemberRole.OWNER) {
  await ownerTransferModel.update({
    where: { id: transfer.id },
    data: { status: "CANCELLED", cancelledAt: now },
  });
  return NextResponse.json({ ok: false, error: "TRANSFER_NO_LONGER_VALID" }, { status: 400 });
}

const [organizer, fromProfile, toProfile] = await Promise.all([
  prisma.organizer.findUnique({
    where: { id: transfer.organizerId },
    select: { displayName: true, publicName: true, username: true },
  }),
  prisma.profile.findUnique({
    where: { id: transfer.fromUserId },
    select: { fullName: true, username: true },
  }),
  prisma.profile.findUnique({
    where: { id: transfer.toUserId },
    select: { fullName: true, username: true },
  }),
]);
await prisma.$transaction(async (tx) => {
  await ownerTransferModel.update({
    where: { id: transfer.id },
    data: { status: "CONFIRMED", confirmedAt: now },
  });

  await setSoleOwner(tx, transfer.organizerId, transfer.toUserId, transfer.fromUserId);
}
);

```

```

    await recordOrganizationAudit(tx, {
      organizerId: transfer.organizerId,
      actorUserId: user.id,
      action: "OWNER_TRANSFER_CONFIRMED",
      fromUserId: transfer.fromUserId,
      toUserId: transfer.toUserId,
      metadata: { transferId: transfer.id, token: transfer.token },
      ip: req.ip ?? null,
      userAgent: req.headers.get("user-agent"),
    });
  });

  // Notificar o antigo OWNER (best-effort)
  const organizerName = organizer?.publicName || organizer?.displayName || organizer?.username || "Organização ORYA";
  const toName = toProfile?.fullName || toProfile?.username || "novo OWNER";
  const baseUrl = process.env.NEXT_PUBLIC_BASE_URL ?? process.env.NEXT_PUBLIC_APP_URL ?? "https://orya.pt";
  try {
    const fromUser = await supabaseAdmin.auth.admin.getUserById(transfer.fromUserId);
    const fromEmail = fromUser.data?.user?.email ?? null;
    if (fromEmail) {
      await sendEmail({
        to: fromEmail,
        subject: `✅ Transferência concluída - ${organizerName}`,
        html: `<div style="font-family: Arial, sans-serif; color:#0f172a;">
          <h2>Transferência de OWNER concluída</h2>
          <p>O papel de OWNER em <strong>${organizerName}</strong> foi assumido por <strong>${toName}</strong>.</p>
          <p>Podes rever o staff aqui: <a href="${baseUrl}/organizador?tab=staff" style="color:#2563eb;">Ver staff</a></p>
        </div>`,
        text: `Transferência de OWNER concluída\n${organizerName}\nNovo OWNER: ${toName}\nStaff: ${baseUrl}/organizador?tab=staff`,
      });
    } else {
      console.warn("[owner/confirm] fromUser sem email para notificar", { fromUserId: transfer.fromUserId });
    }
  } catch (emailErr) {
    console.error("[owner/confirm] Falha ao notificar antigo OWNER", emailErr);
  }

  return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
  console.error("[organizer/owner/confirm][POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/organizations/owner/transfer/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { randomUUID } from "crypto";
import { OrganizerMemberRole } from "@prisma/client";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { resolveUserIdentity } from "@/lib/userResolver";
import { getOrgTransferEnabled } from "@/lib/platformSettings";
import { recordOrganizationAudit } from "@/lib/organizationAudit";
import { sendOwnerTransferEmail } from "@/lib/emailSender";
import { supabaseAdmin } from "@/lib/supabaseAdmin";

const DEFAULT_EXPIRATION_MS = 1000 * 60 * 60 * 24 * 3; // 3 dias

export async function POST(req: NextRequest) {
  try {
    const ownerTransferModel = (prisma as any).organizerOwnerTransfer;
    if (!ownerTransferModel?.create) {
      return NextResponse.json({ ok: false, error: "OWNER_TRANSFER_UNAVAILABLE" }, { status: 501 });
    }

    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }
  }
}

```

```

const transferEnabled = await getOrgTransferEnabled();
if (!transferEnabled) {
  return NextResponse.json({ ok: false, error: "ORG_TRANSFER_DISABLED" }, { status: 403 });
}

const body = await req.json().catch(() => null);
const organizerId = Number(body?.organizerId);
const targetRaw = typeof body?.targetUserId === "string" ? body.targetUserId.trim() : null;

if (!organizerId || Number.isNaN(organizerId) || !targetRaw) {
  return NextResponse.json({ ok: false, error: "INVALID_PAYLOAD" }, { status: 400 });
}

const callerMembership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: user.id } },
});
if (!callerMembership || callerMembership.role !== OrganizerMemberRole.OWNER) {
  return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_TRANSFER" }, { status: 403 });
}

const resolved = await resolveUserIdentifier(targetRaw);
const targetUserId = resolved?.userId ?? null;
if (!targetUserId) {
  return NextResponse.json({ ok: false, error: "TARGET_NOT_FOUND" }, { status: 404 });
}
if (targetUserId === user.id) {
  return NextResponse.json({ ok: false, error: "CANNOT_TRANSFER_TO_SELF" }, { status: 400 });
}

const now = Date.now();
const expiresAt = new Date(now + DEFAULT_EXPIRATION_MS);
const token = randomUUID();

const [organizer, actorProfile] = await Promise.all([
  prisma.organizer.findUnique({
    where: { id: organizerId },
    select: { displayName: true, username: true, publicName: true },
  }),
  prisma.profile.findUnique({
    where: { id: user.id },
    select: { fullName: true, username: true },
  }),
]);
const transfer = await prisma.$transaction(async (tx) => {
  // Cancela pedidos pendentes anteriores
  await ownerTransferModel.updateMany({
    where: { organizerId, status: "PENDING" },
    data: { status: "CANCELLED", cancelledAt: new Date(now) },
  });

  const created = await ownerTransferModel.create({
    data: {
      organizerId,
      fromUserId: user.id,
      toUserId: targetUserId,
      status: "PENDING",
      token,
      expiresAt,
    },
  });

  await recordOrganizationAudit(tx, {
    organizerId,
    actorUserId: user.id,
    action: "OWNER_TRANSFER_REQUESTED",
    fromUserId: user.id,
    toUserId: targetUserId,
    metadata: { transferId: created.id, token },
    ip: req.ip ?? null,
    userAgent: req.headers.get("user-agent"),
  });
});

return created;
});

// Envio do email de confirmação para o novo Owner (best-effort)

```

```

try {
  const targetUser = await supabaseAdmin.auth.admin.getUserById(targetUserId);
  const targetEmail = targetUser.data?.user?.email ?? null;
  const organizerName =
    organizer?.publicName || organizer?.displayName || organizer?.username || "Organização ORYA";
  const actorName = actorProfile?.fullName || actorProfile?.username || "OWNER atual";

  if (targetEmail) {
    await sendOwnerTransferEmail({
      to: targetEmail,
      organizerName,
      actorName,
      token: transfer.token,
      expiresAt: transfer.expiresAt,
    });
  } else {
    console.warn("[owner/transfer] target user sem email para envio", { targetUserId });
  }
} catch (emailErr) {
  console.error("[owner/transfer] Falha ao enviar email de transferência", emailErr);
}

return NextResponse.json(
{
  ok: true,
  transfer: {
    id: transfer.id,
    status: transfer.status,
    token: transfer.token,
    expiresAt: transfer.expiresAt,
  },
  {
    status: 200,
  };
} catch (err) {
  console.error("[organizer/owner/transfer][POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/organizations/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { OrganizerStatus, OrganizerMemberRole } from "@prisma/client";
import { normalizeAndValidateUsername, setUsernameForOwner, UsernameTakenError } from "@/lib/globalUsernames";

export async function GET() {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const memberships = await prisma.organizerMember.findMany({
      where: { userId: user.id },
      include: { organizer: true },
      orderBy: [{ lastUsedAt: "desc" }, { createdAt: "asc" }],
    });

    const items = (memberships || []).filter((m) => m.organizer)
      .map((m) => ({
        organizerId: m.organizerId,
        role: m.role,
        lastUsedAt: (m as { lastUsedAt?: Date | null }).lastUsedAt ?? null,
        organizer: {
          id: m.organizer!.id,
          displayName: m.organizer!.displayName,
          username: m.organizer!.username,
        }
      }));
  }
}

```

```

        businessName: m.organizer!.businessName,
        city: m.organizer!.city,
        entityType: m.organizer!.entityType,
        status: m.organizer!.status,
    },
});

return NextResponse.json({ ok: true, items }, { status: 200 });
} catch (err: unknown) {
    if (typeof err === "object" && err && (err as { code?: string }).code === "P2021") {
        try {
            const supabase = await createSupabaseServer();
            const {
                data: { user },
            } = await supabase.auth.getUser();
            const legacyOrganizers = user
                ? await prisma.organizer.findMany({
                    where: { userId: user.id },
                    orderBy: { createdAt: "asc" },
                })
                : [];
            return NextResponse.json(
                {
                    ok: true,
                    items: legacyOrganizers.map((org) => ({
                        organizerId: org.id,
                        role: "OWNER" as OrganizerMemberRole,
                        lastUsedAt: null,
                        organizer: {
                            id: org.id,
                            username: org.username,
                            displayName: org.displayName,
                            businessName: org.businessName,
                            city: org.city,
                            entityType: org.entityType,
                            status: org.status,
                        },
                    })),
                    warning: "Tabela organizer_members em falta. A usar dados legacy.",
                },
                { status: 200 },
            );
        } catch (fallbackErr) {
            console.error("[organizador/organizations][GET] fallback error", fallbackErr);
            return NextResponse.json(
                { ok: false, error: "Base de dados sem tabela organizer_members. Corre as migrations." },
                { status: 500 },
            );
        }
    }
    console.error("[organizador/organizations][GET]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

export async function POST(req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error,
        } = await supabase.auth.getUser();

        if (error || !user) {
            return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
        }

        const body = await req.json().catch(() => null);
        if (!body || typeof body !== "object") {
            return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });
        }

        const { businessName, displayName, entityType, city, username } = body as Record<string, unknown>;
        const bName = typeof businessName === "string" ? businessName.trim() : null;
        const dName =
            typeof displayName === "string" && displayName.trim().length > 0
            ? displayName.trim()
            : bName || "Organizador";
    }
}

```

```

const eType = typeof entityType === "string" ? entityType.trim() : null;
const cityClean = typeof city === "string" ? city.trim() : null;

const validatedUsername = typeof username === "string" ? normalizeAndValidateUsername(username) : { ok: false, error: "Escolhe um username para a organização." };

if (!bName || !cityClean || !eType || !validatedUsername.ok) {
  return NextResponse.json(
    {
      ok: false,
      error:
        !validatedUsername.ok
          ? validatedUsername.error
          : "Faltam campos obrigatórios: nome, cidade e tipo de entidade.",
    },
    { status: 400 },
  );
}

// Guardar username reservado em outras entidades
const existingOrganizer = await prisma.organizer.findFirst({
  where: { username: { equals: validatedUsername.username, mode: "insensitive" } },
  select: { id: true },
});
if (existingOrganizer) {
  return NextResponse.json(
    { ok: false, error: "Este @ já está a ser usado – escolhe outro.", code: "USERNAME_TAKEN" },
    { status: 409 },
  );
}
const existingProfile = await prisma.profile.findFirst({
  where: { username: { equals: validatedUsername.username, mode: "insensitive" } },
  select: { id: true },
});
if (existingProfile) {
  return NextResponse.json(
    { ok: false, error: "Este @ já está a ser usado – escolhe outro.", code: "USERNAME_TAKEN" },
    { status: 409 },
  );
}

const organizer = await prisma.$transaction(async (tx) => {
  async function createOrganizer(includePublicName: boolean) {
    return tx.organizer.create({
      data: {
        userId: user.id, // legacy compat
        displayName: dName,
        ...(includePublicName ? { publicKey: dName } : {}),
        businessName: bName,
        entityType: eType,
        city: cityClean,
        status: OrganizerStatus.ACTIVE,
        username: validatedUsername.username,
      },
    });
  }

  try {
    return await createOrganizer(true);
  } catch (err) {
    const code = (err as { code?: string })?.code;
    const message = err instanceof Error ? err.message : "";
    const missingColumn = code === "P2022" && message.toLowerCase().includes("public_name");
    if (!missingColumn) throw err;
    return createOrganizer(false);
  }
});

try {
  await prisma.organizerMember.upsert({
    where: { organizerId_userId: { organizerId: organizer.id, userId: user.id } },
    update: { role: OrganizerMemberRole.OWNER },
    create: { organizerId: organizer.id, userId: user.id, role: OrganizerMemberRole.OWNER },
  });
} catch (err: unknown) {
  if (!(typeof err === "object" && err && "code" in err && (err as { code?: string }).code === "P2021")) {
    throw err;
  }
}

```

```

        console.warn("[organizador/organizations][POST] organizer_members em falta; criámos org sem membership");
    }

    // Reservar username global fora da transação para não abortar criação
    try {
        await setUsernameForOwner({
            username: validatedUsername.username,
            ownerType: "organizer",
            ownerId: organizer.id,
        });
    } catch (err) {
        if (err instanceof UsernameTakenError) {
            return NextResponse.json(
                { ok: false, error: "Este @ já está a ser usado – escolhe outro.", code: "USERNAME_TAKEN" },
                { status: 409 },
            );
        }
        console.warn("[organizador/organizations][POST] username global falhou (ignorado)", err);
    }

    return NextResponse.json(
        {
            ok: true,
            organizer: {
                id: organizer.id,
                displayName: organizer.displayName,
                username: organizer.username,
                businessName: organizer.businessName,
                city: organizer.city,
                entityType: organizer.entityType,
            },
        },
        { status: 201 },
    );
} catch (err: unknown) {
    if (err instanceof UsernameTakenError) {
        return NextResponse.json(
            { ok: false, error: "Este @ já está a ser usado – escolhe outro.", code: "USERNAME_TAKEN" },
            { status: 409 },
        );
    }
    if (typeof err === "object" && err && "code" in err && (err as { code?: string }).code === "P2021") {
        return NextResponse.json(
            { ok: false, error: "Base de dados sem tabela organizer_members. Corre as migrations." },
            { status: 500 },
        );
    }
    console.error("[organizador/organizations][POST]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/organizations/settings/official-email/confirm/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole } from "@prisma/client";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { recordOrganizationAudit } from "@/lib/organizationAudit";

const STATUS_PENDING = "PENDING";

export async function POST(req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error,
        } = await supabase.auth.getUser();

        if (error || !user) {
            return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
        }

        const body = await req.json().catch(() => null);
        const token = typeof body?.token === "string" ? body.token.trim() : null;
    }
}

```

```

if (!token) {
  return NextResponse.json({ ok: false, error: "INVALID_TOKEN" }, { status: 400 });
}

const request = await prisma.organizerOfficialEmailRequest.findUnique({
  where: { token },
});
if (!request) {
  return NextResponse.json({ ok: false, error: "REQUEST_NOT_FOUND" }, { status: 404 });
}
if (request.status !== STATUS_PENDING) {
  return NextResponse.json({ ok: false, error: "REQUEST_NOT_PENDING" }, { status: 400 });
}

const membership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId: request.organizerId, userId: user.id } },
});
if (!membership || membership.role !== OrganizerMemberRole.OWNER) {
  return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_CONFIRM" }, { status: 403 });
}

const now = new Date();
if (request.expiresAt && request.expiresAt.getTime() < now.getTime()) {
  await prisma.organizerOfficialEmailRequest.update({
    where: { id: request.id },
    data: { status: "EXPIRED", cancelledAt: now },
  });
  return NextResponse.json({ ok: false, error: "REQUEST_EXPIRED" }, { status: 400 });
}

await prisma.$transaction(async (tx) => {
  await tx.organizerOfficialEmailRequest.update({
    where: { id: request.id },
    data: { status: "CONFIRMED", confirmedAt: now },
  });

  await tx.organizerOfficialEmailRequest.updateMany({
    where: { organizerId: request.organizerId, id: { not: request.id }, status: STATUS_PENDING },
    data: { status: "CANCELLED", cancelledAt: now },
  });

  await tx.organizer.update({
    where: { id: request.organizerId },
    data: { officialEmail: request.newEmail, officialEmailVerifiedAt: now },
  });

  await recordOrganizationAudit(tx, {
    organizerId: request.organizerId,
    actorUserId: user.id,
    action: "OFFICIAL_EMAIL_CONFIRMED",
    metadata: { requestId: request.id, email: request.newEmail },
    ip: req.ip ?? null,
    userAgent: req.headers.get("user-agent"),
  });
});

return NextResponse.json({ ok: true, verifiedAt: now }, { status: 200 });
} catch (err) {
  console.error("[official-email/confirm][POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/organizations/settings/official-email/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { randomUUID } from "crypto";
import { OrganizerMemberRole } from "@prisma/client";
import { createSupabaseServer } from "@lib/supabaseServer";
import { prisma } from "@lib/prisma";
import { recordOrganizationAudit } from "@lib/organizationAudit";
import { sendOfficialEmailVerificationEmail } from "@lib/emailSender";

const EMAIL_REGEX = /^[^@\s]+@[^\s]+\.[^\s]+\$/i;
const DEFAULT_EXPIRATION_MS = 1000 * 60 * 60 * 24; // 24h
const STATUS_PENDING = "PENDING";

```

```

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const body = await req.json().catch(() => null);
    const organizerId = Number(body?.organizerId);
    const emailRaw = typeof body?.email === "string" ? body.email.trim().toLowerCase() : null;
    if (!organizerId || Number.isNaN(organizerId) || !emailRaw) {
      return NextResponse.json({ ok: false, error: "INVALID_PAYLOAD" }, { status: 400 });
    }
    if (!EMAIL_REGEX.test(emailRaw)) {
      return NextResponse.json({ ok: false, error: "INVALID_EMAIL" }, { status: 400 });
    }

    const membership = await prisma.organizerMember.findUnique({
      where: { organizerId_userId: { organizerId, userId: user.id } },
    });
    if (!membership || membership.role !== OrganizerMemberRole.OWNER) {
      return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_UPDATE_OFFICIAL_EMAIL" }, { status: 403 });
    }

    const organizer = await prisma.organizer.findUnique({
      where: { id: organizerId },
      select: {
        id: true,
        officialEmail: true,
        officialEmailVerifiedAt: true,
        displayName: true,
        publicName: true,
        username: true,
      },
    });
    if (!organizer) {
      return NextResponse.json({ ok: false, error: "ORGANIZER_NOT_FOUND" }, { status: 404 });
    }

    if (organizer.officialEmailVerifiedAt && organizer.officialEmail === emailRaw) {
      return NextResponse.json({ ok: false, error: "EMAIL_ALREADY_VERIFIED" }, { status: 400 });
    }

    await prisma.organizer.update({
      where: { id: organizerId },
      data: { officialEmail: emailRaw },
    });

    const now = Date.now();
    const expiresAt = new Date(now + DEFAULT_EXPIRATION_MS);
    const token = randomUUID();

    const request = await prisma.$transaction(async (tx) => {
      await tx.organizerOfficialEmailRequest.updateMany({
        where: { organizerId, status: STATUS_PENDING },
        data: { status: "CANCELLED", cancelledAt: new Date(now) },
      });
    });

    const created = await tx.organizerOfficialEmailRequest.create({
      data: {
        organizerId,
        requestedByUserId: user.id,
        newEmail: emailRaw,
        token,
        status: STATUS_PENDING,
        expiresAt,
      },
    });

    await tx.organizer.update({
      where: { id: organizerId },
      data: { officialEmail: emailRaw, officialEmailVerifiedAt: null },
    });
  }
}

```

```

    });

    await recordOrganizationAudit(tx, {
      organizerId,
      actorUserId: user.id,
      action: "OFFICIAL_EMAIL_CHANGE_REQUESTED",
      metadata: { email: emailRaw, requestId: created.id },
      ip: req.ip ?? null,
      userAgent: req.headers.get("user-agent"),
    });

    return created;
  });

  // Envia email de verificação (best-effort)
  try {
    const organizerName =
      organizer.publicName || organizer.displayName || organizer.username || "Organização ORYA";
    await sendOfficialEmailVerificationEmail({
      to: emailRaw,
      organizerName,
      token: request.token,
      pendingEmail: emailRaw,
      expiresAt: request.expiresAt,
    });
  } catch (emailErr) {
    console.error("[organizer/official-email] Falha ao enviar email de verificação", emailErr);
  }

  return NextResponse.json(
    {
      ok: true,
      status: request.status,
      expiresAt: request.expiresAt,
      pendingEmail: emailRaw,
    },
    { status: 200 },
  );
} catch (err) {
  console.error("[organizer/official-email][POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/organizations/switch/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@lib/supabaseServer";
import { prisma } from "@lib/prisma";
import { cookies } from "next/headers";

const COOKIE_NAME = "orya_org";
const COOKIE_MAX_AGE = 60 * 60 * 24 * 30; // 30 dias

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const body = await req.json().catch(() => null);
    if (!body || typeof body !== "object") {
      return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });
    }

    const { organizerId } = body as { organizerId?: number };
    if (!organizerId || typeof organizerId !== "number") {
      return NextResponse.json({ ok: false, error: "INVALID_ORGANIZER_ID" }, { status: 400 });
    }
  }
}

```

```

// Validar membership
const membership = await prisma.organizerMember.findFirst({
  where: { organizerId, userId: user.id, organizer: { status: "ACTIVE" } },
  include: { organizer: true },
});

if (!membership || !membership.organizer) {
  return NextResponse.json({ ok: false, error: "NOT_MEMBER" }, { status: 403 });
}

// Guardar cookie com org atual + atualizar lastUsedAt
try {
  await prisma.organizerMember.updateMany({
    where: { organizerId, userId: user.id },
    data: { lastUsedAt: new Date() },
  });
} catch (err: unknown) {
  if (!(typeof err === "object" && err && (err as { code?: string }).code === "P2021")) {
    throw err;
  }
}

const res = NextResponse.json({
  ok: true,
  organizerId,
  role: membership.role,
});
res.cookies.set(COOKIE_NAME, String(organizerId), {
  httpOnly: false,
  sameSite: "lax",
  path: "/",
  maxAge: COOKIE_MAX_AGE,
});
return res;
} catch (err: unknown) {
  if (typeof err === "object" && err && "code" in err && (err as { code?: string }).code === "P2021") {
    return NextResponse.json(
      { ok: false, error: "Base de dados sem tabela organizer_members. Corre as migrations." },
      { status: 500 },
    );
  }
  console.error("[organizador/organizations/switch][POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}
}

```

app/api/organizador/organizations/transfer/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { OrganizerMemberRole } from "@prisma/client";
import { resolveUserIdentity } from "@/lib/userResolver";
import { getOrgTransferEnabled } from "@/lib/platformSettings";
import { setSoleOwner } from "@/lib/organizerRoles";
import { recordOrganizationAuditSafe } from "@/lib/organizationAudit";

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const transferEnabled = await getOrgTransferEnabled();
    if (!transferEnabled) {
      return NextResponse.json({ ok: false, error: "ORG_TRANSFER_DISABLED" }, { status: 403 });
    }

    const body = await req.json().catch(() => null);
    const organizerId = Number(body?.organizerId);
  }
}

```

```

const targetIdentifierRaw = typeof body?.targetUserId === "string" ? body.targetUserId.trim() : null;

if (!organizerId || Number.isNaN(organizerId) || !targetIdentifierRaw) {
  return NextResponse.json({ ok: false, error: "INVALID_PAYLOAD" }, { status: 400 });
}

const resolvedUser = await resolveUserIdentity(targetIdentifierRaw);
const targetUserId = resolvedUser?.userId ?? null;
if (!targetUserId) {
  return NextResponse.json({ ok: false, error: "TARGET_NOT_FOUND" }, { status: 404 });
}

// Confirma que o caller é OWNER desta org
const callerMembership = await prisma.organizerMember.findFirst({
  where: { organizerId, userId: user.id },
});
if (!callerMembership || callerMembership.role !== OrganizerMemberRole.OWNER) {
  return NextResponse.json({ ok: false, error: "ONLY_OWNER_CAN_TRANSFER" }, { status: 403 });
}

await prisma.$transaction(async (tx) => {
  await setSoleOwner(tx, organizerId, targetUserId, user.id);
});

await recordOrganizationAuditSafe({
  organizerId,
  actorUserId: user.id,
  action: "OWNER_TRANSFER_DIRECT",
  fromUserId: user.id,
  toUserId: targetUserId,
});

return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
  console.error("[organizador/organizations/transfer]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/padel/tournaments/create/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { PORTUGAL_CITIES } from "@config/cities";
import { FeeMode, RefundFeePayer, ResaleMode } from "@prisma/client";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";

type TicketInput = {
  name?: string;
  price?: number;
  totalQuantity?: number | null;
};

type Body = {
  title?: string;
  description?: string;
  internalNote?: string | null;
  coverImageUrl?: string | null;
  startsAt?: string;
  endsAt?: string;
  locationName?: string | null;
  locationCity?: string;
  address?: string | null;
  templateType?: string;
  categories?: string[];
  feeMode?: "ON_TOP" | "INCLUDED" | "ADDED";
  refundFeePayer?: RefundFeePayer;
  ticketTypes?: TicketInput[];
  padel?: unknown; // ignorado para já, usamos defaults globais
  visibility?: "PUBLIC" | "PRIVATE";
  publicListingEnabled?: boolean;
  padelClubId?: number;
  partnerClubIds?: Array<number | string>;
}

```

```

courtsCount?: number | null;
clubHours?: string | null;
tournamentState?: "OCULTO" | "INSCRICOES" | "PUBLICO" | "TERMINADO" | "CANCELADO";
advancedSettings?: Record<string, unknown> | null;
};

function parseDate(raw?: string | null) {
  if (!raw) return null;
  const normalized = raw.replace(" ", "T");
  const date = new Date(normalized);
  if (!Number.isNaN(date.getTime())) return date;
  const alt = new Date(`#${normalized}:00`);
  if (!Number.isNaN(alt.getTime())) return alt;
  return null;
}

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    const body = (await req.json().catch(() => null)) as Body | null;
    if (!body) {
      return NextResponse.json({ ok: false, error: "Body inválido." }, { status: 400 });
    }

    const { organizer, membership } = await getActiveOrganizerForUser(user.id, {
      roles: ["OWNER", "CO_OWNER", "ADMIN"],
    });
    const profile = await prisma.profile.findUnique({ where: { id: user.id } });
    if (!organizer || !profile || !membership || !isOrgAdminOrAbove(membership.role)) {
      return NextResponse.json({ ok: false, error: "Organizador não encontrado." }, { status: 403 });
    }

    const title = body.title?.trim();
    if (!title) return NextResponse.json({ ok: false, error: "Título é obrigatório." }, { status: 400 });

    const padelClubId =
      typeof body.padelClubId === "number"
        ? body.padelClubId
        : typeof body.padelClubId === "string"
        ? Number(body.padelClubId)
        : null;
    if (!padelClubId || Number.isNaN(padelClubId)) {
      return NextResponse.json(
        { ok: false, error: "Precisas de escolher um clube de Padel para criar o torneio." },
        { status: 400 },
      );
    }

    const club = await prisma.padelClub.findFirst({
      where: { id: padelClubId, organizerId: organizer.id, isActive: true },
    });
    if (!club) {
      return NextResponse.json(
        { ok: false, error: "Clube de Padel inválido ou inativo para este organizador." },
        { status: 400 },
      );
    }

    const partnerClubIdsRaw = Array.isArray(body.partnerClubIds) ? body.partnerClubIds : [];
    const partnerClubIds = partnerClubIdsRaw
      .map((c) => (typeof c === "number" ? c : typeof c === "string" ? Number(c) : null))
      .filter((v) => Number.isFinite(v) && v !== padelClubId) as number[];
    const partnerClubs =
      partnerClubIds.length > 0
        ? await prisma.padelClub.findMany({
          where: { id: { in: partnerClubIds }, organizerId: organizer.id, isActive: true },
          select: { id: true },
        })
        : [];
    const validatedPartnerIds = partnerClubs.map((c) => c.id);
    const allClubIds = [club.id, ...validatedPartnerIds];

    const startsAt = parseDate(body.startsAt);
    if (!startsAt) return NextResponse.json({ ok: false, error: "Data de início inválida." }, { status: 400 });
    const endsAtParsed = parseDate(body.endsAt);
    const endsAt = endsAtParsed && endsAtParsed >= startsAt ? endsAtParsed : startsAt;
  }
}

```

```

const locationCity = body.locationCity?.trim() || club.city?.trim() || "";
if (!locationCity) return NextResponse.json({ ok: false, error: "Cidade é obrigatória." }, { status: 400 });
if (locationCity && !PORTUGAL_CITIES.includes(locationCity as (typeof PORTUGAL_CITIES)[number])) {
  return NextResponse.json(
    { ok: false, error: "Cidade inválida. Escolhe uma cidade da lista disponível na ORYA." },
    { status: 400 },
  );
}

const ticketTypesInput = Array.isArray(body.ticketTypes) ? body.ticketTypes : [];
const ticketTypes = ticketTypesInput
  .map((t) => {
    const name = t.name?.trim();
    if (!name) return null;
    const price = typeof t.price === "number" && !Number.isNaN(t.price) ? t.price : 0;
    const totalQuantity =
      typeof t.totalQuantity === "number" && t.totalQuantity > 0 ? Math.floor(t.totalQuantity) : null;
    return { name, price, totalQuantity };
  })
  .filter(Boolean) as { name: string; price: number; totalQuantity: number | null }[];

if (ticketTypes.length === 0) {
  return NextResponse.json({ ok: false, error: "Adiciona pelo menos um tipo de inscrição." }, { status: 400 });
}

const feeModeRaw = (body.feeMode ?? "ADDED").toUpperCase();
const feeMode: FeeMode = feeModeRaw === "INCLUDED" ? FeeMode.INCLUDED : FeeMode.ADDED;
const refundFeePayer = body.refundFeePayer || organizer.refundFeePayer || RefundFeePayer.CUSTOMER;
const advancedSettings =
  body.advancedSettings && typeof body.advancedSettings === "object" ? body.advancedSettings : null;

// Courts e staff herdados dos clubes selecionados (principal + parceiros)
const courtsFromClubs = await prisma.padelClubCourt.findMany({
  where: { padelClubId: { in: allClubIds }, isActive: true },
  select: { id: true, padelClubId: true, name: true, indoor: true, displayOrder: true },
  orderBy: [{ padelClubId: "asc" }, { displayOrder: "asc" }, { id: "asc" }],
});
const clubNames = await prisma.padelClub.findMany({
  where: { id: { in: allClubIds } },
  select: { id: true, name: true },
});
const clubNameMap = Object.fromEntries(clubNames.map((c) => [c.id, c.name]));

const staffFromClubs = await prisma.padelClubStaff.findMany({
  where: { padelClubId: { in: allClubIds }, inheritToEvents: true },
  select: { id: true, padelClubId: true, email: true, userId: true, role: true },
  orderBy: [{ padelClubId: "asc" }, { id: "asc" }],
});

// Defaults do clube selecionado
const courtsInput =
  typeof body.courtsCount === "number"
    ? body.courtsCount
    : typeof body.courtsCount === "string"
      ? Number(body.courtsCount)
      : null;
const defaultCourts =
  courtsInput && Number.isFinite(courtsInput) && courtsInput > 0
    ? Math.min(1000, Math.floor(courtsInput))
    : club.courtsCount && club.courtsCount > 0
      ? club.courtsCount
      : 1;
const defaultRuleSetId = organizer.padelDefaultRuleSetId ?? null;
const clubHours = typeof body.clubHours === "string" ? body.clubHours.trim() : club.hours?.trim() || null;
const locationName = body.locationName?.trim() || club.shortName || club.name;
const address = body.address?.trim() || club.address || null;
const tournamentState =
  typeof body.tournamentState === "string"
    ? body.tournamentState.toUpperCase()
    : "OCULTO";
const eventStatus =
  tournamentState === "CANCELADO"
    ? "CANCELLED"
    : tournamentState === "TERMINADO"
      ? "FINISHED"
      : tournamentState === "OCULTO"
        ? "DRAFT"
        : null;

```

```

        : "PUBLISHED";

const event = await prisma.event.create({
  data: {
    slug: `${title.toLowerCase().replace(/[^a-z0-9]+/g, "-")}-${Math.random().toString(36).slice(2, 7)}`,
    title,
    description: body.description?.trim() ?? "",
    type: "ORGANIZER_EVENT",
    templateType: "PADEL",
    organizerId: organizer.id,
    ownerId: profile.id,
    startsAt,
    endsAt,
    locationName,
    locationCity,
    address,
    isFree: ticketTypes.every((t) => t.price === 0),
    status: eventStatus as any,
    resaleMode: ResaleMode.ALWAYS,
    coverImageUrl: body.coverImageUrl?.trim() || null,
    feeMode,
    payoutMode: "ORGANIZER",
  },
});

// Categorias: força DESPORTO
await prisma.eventCategory.create({
  data: {
    eventId: event.id,
    category: "DESPORTO",
  },
});

await prisma.ticketType.createMany({
  data: ticketTypes.map((t) => ({
    eventId: event.id,
    name: t.name,
    price: Math.round(t.price * 100),
    totalQuantity: t.totalQuantity,
  })),
});

// Validar rule set default (se existir)
if (defaultRuleSetId) {
  const ruleSetValid = await prisma.padelRuleSet.findFirst({
    where: { id: defaultRuleSetId, organizerId: organizer.id },
    select: { id: true },
  });
  if (!ruleSetValid) {
    return NextResponse.json(
      { ok: false, error: "Rule set inválido para este organizador." },
      { status: 400 },
    );
  }
}

// Config Padel v2 (defaults globais)
const formatValue = "GRUPOS_ELIMINATORIAS";
await prisma.padelTournamentConfig.upsert({
  where: { eventId: event.id },
  create: {
    eventId: event.id,
    organizerId: organizer.id,
    format: formatValue,
    numberOfCourts: defaultCourts,
    ruleSetId: defaultRuleSetId,
    padelClubId: club.id,
    partnerClubIds: validatedPartnerIds,
    clubHours,
    enabledFormats: [],
    padelV2Enabled: true,
    splitDeadlineHours: 48,
    autoCancelUnpaid: true,
    allowCaptainAssume: true,
    defaultPaymentMode: null,
    refundFeePayer,
    advancedSettings: {
      ...advancedSettings,
    }
  }
});

```

```

courtsFromClubs: courtsFromClubs.map((c) => ({
  id: c.id,
  clubId: c.padelClubId,
  clubName: clubNameMap[c.padelClubId] || null,
  name: c.name,
  indoor: c.indoor,
  displayOrder: c.displayOrder,
})),
staffFromClubs: staffFromClubs.map((s) => ({
  id: s.id,
  clubId: s.padelClubId,
  clubName: clubNameMap[s.padelClubId] || null,
  email: s.email,
  userId: s.userId,
  role: s.role,
})),
} as any,
},
update: {
  format: formatValue,
  numberofCourts: defaultCourts,
  ruleSetId: defaultRuleSetId,
  padelClubId: club.id,
  partnerClubIds: validatedPartnerIds,
  clubHours,
  enabledFormats: [],
  padelV2Enabled: true,
  splitDeadlineHours: 48,
  autoCancelUnpaid: true,
  allowCaptainAssume: true,
  defaultPaymentMode: null,
  refundFeePayer,
  advancedSettings: {
    ...advancedSettings,
    courtsFromClubs: courtsFromClubs.map((c) => ({
      id: c.id,
      clubId: c.padelClubId,
      clubName: clubNameMap[c.padelClubId] || null,
      name: c.name,
      indoor: c.indoor,
      displayOrder: c.displayOrder,
})),
    staffFromClubs: staffFromClubs.map((s) => ({
      id: s.id,
      clubId: s.padelClubId,
      clubName: clubNameMap[s.padelClubId] || null,
      email: s.email,
      userId: s.userId,
      role: s.role,
})),
  } as any,
},
});
return NextResponse.json({ ok: true, event: { id: event.id, slug: event.slug } }, { status: 201 });
} catch (err) {
  console.error("[organizador/padel/tournaments/create] error", err);
  return NextResponse.json({ ok: false, error: "Erro ao criar torneio de Padel." }, { status: 500 });
}
}
}

```

app/api/organizador/pagamentos/invoices/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@lib/prisma";
import { createSupabaseServer } from "@lib/supabaseServer";
import { OrganizerMemberRole } from "@prisma/client";

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();
  }
}

```

```

if (error || !user) {
  return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
}

const url = new URL(req.url);
const organizerId = Number(url.searchParams.get("organizerId"));
if (!organizerId || Number.isNaN(organizerId)) {
  return NextResponse.json({ ok: false, error: "INVALID_ORGANIZER" }, { status: 400 });
}

const membership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: user.id } },
});
if (!membership || ![OrganizerMemberRole.OWNER, OrganizerMemberRole.CO_OWNER, OrganizerMemberRole.ADMIN].includes(membership.role)) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}

const from = url.searchParams.get("from");
const to = url.searchParams.get("to");

const sales = await prisma.saleSummary.findMany({
  where: {
    event: { organizerId },
    ...(from || to
      ? {
        createdAt: {
          ...(from ? { gte: new Date(from) } : {}),
          ...(to ? { lte: new Date(to) } : {}),
        },
      }
      : {}),
  },
  orderBy: { createdAt: "desc" },
  include: {
    event: { select: { id: true, title: true, slug: true, payoutMode: true } },
    lines: true,
  },
  take: 200,
});

const summary = sales.reduce(
  (acc, sale) => {
    acc.grossCents += sale.subtotalCents;
    acc.discountCents += sale.discountCents;
    acc.platformFeeCents += sale.platformFeeCents;
    acc.netCents += sale.netCents;
    acc.tickets += sale.lines.reduce((s, l) => s + l.quantity, 0);
    return acc;
  },
  { grossCents: 0, discountCents: 0, platformFeeCents: 0, netCents: 0, tickets: 0 },
);

return NextResponse.json({ ok: true, items: sales, summary }, { status: 200 });
} catch (err) {
  console.error("[api/organizador/pagamentos/invoices]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/payouts/connect/route.ts

```

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { stripe } from "@/lib/stripeClient";
import { OrganizerMemberRole } from "@prisma/client";
import { isOrgOwner } from "@/lib/organizerPermissions";

const DEFAULT_BASE_URL = "http://localhost:3000";

function getBaseUrl() {
  if (process.env.NEXT_PUBLIC_BASE_URL) {

```

```

    return process.env.NEXT_PUBLIC_BASE_URL;
}

const vercelUrl = process.env.VERCEL_URL;
if (vercelUrl?.startsWith("http")) return vercelUrl;
if (vercelUrl) return `https://${vercelUrl}`;

return DEFAULT_BASE_URL;
}

export async function POST(_req: NextRequest) {
try {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    return NextResponse.json(
      { ok: false, error: "Não autenticado." },
      { status: 401 },
    );
  }

  const profile = await prisma.profile.findUnique({
    where: { id: user.id },
  });

  if (!profile) {
    return NextResponse.json(
      { ok: false, error: "Perfil não encontrado." },
      { status: 404 },
    );
  }

  const membership = await prisma.organizerMember.findFirst({
    where: { userId: profile.id, organizer: { status: "ACTIVE" } },
    include: { organizer: true },
    orderBy: [{ lastUsedAt: "desc" }, { createdAt: "asc" }],
  });

  if (!membership || !membership.organizer || !isOrgOwner(membership.role as OrganizerMemberRole)) {
    return NextResponse.json(
      { ok: false, error: "APENAS_OWNER" },
      { status: 403 },
    );
  }

  const organizer = membership.organizer;

  if (organizer.status !== "ACTIVE") {
    return NextResponse.json(
      {
        ok: false,
        error: "Conta de organizador ainda não está ativa.",
        status: organizer.status,
      },
      { status: 403 },
    );
  }

  let accountId = organizer.stripeAccountId;

  if (!accountId) {
    const account = await stripe.accounts.create({
      type: "express",
      country: "PT",
      email: user.email ?? undefined,
      business_type: "individual",
      capabilities: {
        card_payments: { requested: true },
        transfers: { requested: true },
      },
      metadata: {
        organizerId: String(organizer.id),
        userId: profile.id,
      },
    });
  }
}

```

```

    });

    accountId = account.id;

    await prisma.organizer.update({
      where: { id: organizer.id },
      data: {
        stripeAccountId: accountId,
        stripeChargesEnabled: account.charges_enabled ?? false,
        stripePayoutsEnabled: account.payouts_enabled ?? false,
      },
    });
  }

  const baseUrl = getBaseUrl();
  const link = await stripe.accountLinks.create({
    account: accountId,
    refresh_url: `${baseUrl}/organizador?tab=finance&onboarding=refresh`,
    return_url: `${baseUrl}/organizador?tab=finance&onboarding=done`,
    type: "account_onboarding",
  });

  return NextResponse.json(
    {
      ok: true,
      url: link.url,
      accountId,
    },
    { status: 200 },
  );
} catch (err) {
  console.error("[organizador][payouts][connect] erro:", err);
  return NextResponse.json(
    { ok: false, error: "Erro ao gerar onboarding Stripe." },
    { status: 500 },
  );
}
}
}

```

app/api/organizador/payouts/settings/route.ts

```

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { FeeMode } from "@prisma/client";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { isOrgOwner } from "@/lib/organizerPermissions";

function isValidFeeMode(value: string | null | undefined): value is FeeMode {
  if (!value) return false;
  return value === "ADDED" || value === "INCLUDED";
}

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const { organizer, membership } = await getActiveOrganizerForUser(user.id, {
      roles: ["OWNER", "CO_OWNER", "ADMIN"],
    });

    if (!organizer || !membership || !isOrgOwner(membership.role)) {
      return NextResponse.json({ ok: false, error: "APENAS_OWNER" }, { status: 403 });
    }

    if (organizer.status !== "ACTIVE") {

```

```

    return NextResponse.json({ ok: false, error: "ORGANIZER_NOT_ACTIVE" }, { status: 403 });
}

const body = (await req.json().catch(() => null)) as {
  feeMode?: string;
  platformFeeBps?: number;
  platformFeeFixedCents?: number;
} | null;

if (!body || typeof body !== "object") {
  return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });
}

const updates: Partial<{ feeMode: FeeMode; platformFeeBps: number; platformFeeFixedCents: number }> = {};

if (body.feeMode !== undefined) {
  if (!isValidFeeMode(body.feeMode)) {
    return NextResponse.json({ ok: false, error: "INVALID_FEE_MODE" }, { status: 400 });
  }
  updates.feeMode = body.feeMode;
}

if (body.platformFeeBps !== undefined) {
  const value = Number(body.platformFeeBps);
  if (!Number.isFinite(value) || value < 0 || value > 5000) {
    return NextResponse.json({ ok: false, error: "INVALID_FEE_BPS" }, { status: 400 });
  }
  updates.platformFeeBps = Math.floor(value);
}

if (body.platformFeeFixedCents !== undefined) {
  const value = Number(body.platformFeeFixedCents);
  if (!Number.isFinite(value) || value < 0 || value > 5000) {
    return NextResponse.json({ ok: false, error: "INVALID_FEE_FIXED" }, { status: 400 });
  }
  updates.platformFeeFixedCents = Math.floor(value);
}

if (Object.keys(updates).length === 0) {
  return NextResponse.json({ ok: false, error: "NOTHING_TO_UPDATE" }, { status: 400 });
}

const updated = await prisma.organizer.update({
  where: { id: organizer.id },
  data: updates,
});

return NextResponse.json(
  {
    ok: true,
    organizer: {
      id: updated.id,
      feeMode: updated.feeMode,
      platformFeeBps: updated.platformFeeBps,
      platformFeeFixedCents: updated.platformFeeFixedCents,
    },
  },
  { status: 200 },
);
} catch (err) {
  console.error("[organizador/payouts/settings][POST] erro", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/payouts/status/route.ts

```

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { stripe } from "@/lib/stripeClient";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { isOrgOwner } from "@/lib/organizerPermissions";

```

```

import { createNotification, shouldNotify } from "@/lib/notifications";
import { NotificationType } from "@prisma/client";

export async function GET() {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const { organizer, membership } = await getActiveOrganizerForUser(user.id);

    if (!organizer || !membership || !isOrgOwner(membership.role)) {
      return NextResponse.json({ ok: false, error: "APENAS_OWNER" }, { status: 403 });
    }

    if (!organizer.stripeAccountId) {
      console.log("[stripe][status] no account", { organizerId: organizer.id });
      return NextResponse.json({
        ok: true,
        status: "NOT_CONNECTED",
        charges_enabled: false,
        payouts_enabled: false,
        requirements_due: [],
      });
    }
  }

  const account = await stripe.accounts.retrieve(organizer.stripeAccountId);

  const charges_enabled = account.charges_enabled ?? false;
  const payouts_enabled = account.payouts_enabled ?? false;
  const requirements_due = account.requirements?.currently_due ?? [];

  await prisma.organizer.update({
    where: { id: organizer.id },
    data: {
      stripeAccountId: account.id,
      stripeChargesEnabled: charges_enabled,
      stripePayoutsEnabled: payouts_enabled,
    },
  });

  const status =
    charges_enabled && payouts_enabled && (!requirements_due || requirements_due.length === 0)
    ? "CONNECTED"
    : "INCOMPLETE";

  console.log("[stripe][status] refreshed", {
    organizerId: organizer.id,
    accountId: account.id,
    charges_enabled,
    payouts_enabled,
    requirements_due,
  });
}

// Notificar owner/admin se estado mudou para attention
if (status === "INCOMPLETE" && requirements_due && requirements_due.length > 0) {
  const owners = await prisma.organizerMember.findMany({
    where: { organizerId: organizer.id, role: { in: ["OWNER", "CO_OWNER", "ADMIN"] } },
    select: { userId: true },
  });
  const uniq = Array.from(new Set(owners.map(o => o.userId)));
  await Promise.all(
    uniq.map(async (uid) => {
      if (!(await shouldNotify(uid, NotificationType.STRIPE_STATUS))) return;
      await createNotification({
        userId: uid,
        type: NotificationType.STRIPE_STATUS,
        title: "Stripe precisa de atenção",
        body: "Faltam dados no Stripe para ativar pagamentos/payouts.",
        ctaUrl: "/organizador?tab=finance",
        ctaLabel: "Rever Stripe",
        payload: { requirements_due },
      });
    })
  );
}

```

```

        });
    },
);
}

return NextResponse.json({
  ok: true,
  status,
  charges_enabled,
  payouts_enabled,
  requirements_due,
  accountId: account.id,
});
} catch (err) {
  console.error("[stripe][status] error", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/payouts/summary/route.ts

```

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";
import { TicketStatus } from "@prisma/client";

export async function GET() {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const { organizer, membership } = await getActiveOrganizerForUser(user.id);

    if (!organizer || !membership || !isOrgAdminOrAbove(membership.role)) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    const ticketsAgg = await prisma.ticket.aggregate({
      where: {
        status: { in: [TicketStatus.ACTIVE, TicketStatus.USED] },
        event: { organizerId: organizer.id },
      },
      _count: { _all: true },
      _sum: { pricePaid: true, totalPaidCents: true, platformFeeCents: true },
    });

    const eventsWithSales = await prisma.ticket.findMany({
      where: {
        status: { in: [TicketStatus.ACTIVE, TicketStatus.USED] },
        event: { organizerId: organizer.id },
      },
      select: { eventId: true },
      distinct: ["eventId"],
    });

    const ticketsSold = ticketsAgg._count?._all ?? 0;
    const revenueCents = ticketsAgg._sum?.pricePaid ?? 0;
    const grossCents = ticketsAgg._sum?.totalPaidCents ?? revenueCents;
    const platformFeesCents = ticketsAgg._sum?.platformFeeCents ?? 0;

    return NextResponse.json(
      {
        ok: true,
        ticketsSold,
      }
    );
  }
}

```

```

        revenueCents,
        grossCents,
        platformFeesCents,
        eventsWithSales: eventsWithSales.length,
        estimatedPayoutCents: revenueCents,
    },
    { status: 200 },
);
} catch (err) {
    console.error("[organizador/payouts/summary] [GET] erro", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/payouts/webhook/route.ts

```

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

import { NextRequest, NextResponse } from "next/server";
import Stripe from "stripe";
import { stripe } from "@lib/stripeClient";
import { prisma } from "@lib/prisma";

const webhookSecret =
  process.env.STRIPE_PAYOUTS_WEBHOOK_SECRET || process.env.STRIPE_WEBHOOK_SECRET;

export async function POST(req: NextRequest) {
  const sig = req.headers.get("stripe-signature");

  if (!sig) {
    return new Response("Missing signature", { status: 400 });
  }

  if (!webhookSecret) {
    console.error("[Stripe Connect Webhook] Missing webhook secret env");
    return new Response("Server misconfigured", { status: 500 });
  }

  const body = await req.text();

  let event: Stripe.Event;
  try {
    event = stripe.webhooks.constructEvent(body, sig, webhookSecret);
  } catch (err) {
    const message =
      err instanceof Error ? err.message : "Unknown signature validation error";
    console.error("[Stripe Connect Webhook] Invalid signature:", message);
    return new Response("Invalid signature", { status: 400 });
  }

  try {
    switch (event.type) {
      case "account.updated": {
        const account = event.data.object as Stripe.Account;
        const organizerIdRaw = account.metadata?.organizerId;
        const organizerIdNumber =
          organizerIdRaw && Number.isFinite(Number(organizerIdRaw))
            ? Number(organizerIdRaw)
            : null;

        const chargesEnabled = Boolean(account.charges_enabled);
        const payoutsEnabled = Boolean(account.payouts_enabled);

        await prisma.organizer.updateMany({
          where: { id: organizerIdNumber },
          data: {
            stripeChargesEnabled: chargesEnabled,
            stripePayoutsEnabled: payoutsEnabled,
            stripeAccountId: account.id,
          },
        });

        console.log("[Stripe Connect Webhook] account.updated sync", {
      }
    }
  }
}

```

```

        organizerId: organizerIdNumber,
        accountId: account.id,
        chargesEnabled,
        payoutsEnabled,
    });
    break;
}

default:
    console.log(
        "[Stripe Connect Webhook] Event ignorado:",
        event.type,
    );
    break;
}
} catch (err) {
    console.error("[Stripe Connect Webhook] Error processing event:", err);
}

return NextResponse.json({ received: true });
}

```

app/api/organizador/promo/[id]/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";
import { OrganizerMemberRole } from "@prisma/client";

async function requireOrganizer() {
    const supabase = await createSupabaseServer();
    const {
        data: { user },
        error,
    } = await supabase.auth.getUser();

    if (error || !user) {
        return { error: "UNAUTHENTICATED" as const };
    }

    const profile = await prisma.profile.findUnique({ where: { id: user.id } });
    if (!profile) return { error: "PROFILE_NOT_FOUND" as const };

    const membership = await prisma.organizerMember.findFirst({
        where: {
            userId: user.id,
            organizer: { status: "ACTIVE" },
        },
        include: { organizer: true },
        orderBy: [{ lastUsedAt: "desc" }, { createdAt: "asc" }],
    });

    if (!membership || !membership.organizer || !isOrgAdminOrAbove(membership.role as OrganizerMemberRole)) {
        return { error: "ORGANIZER_NOT_FOUND" as const };
    }

    return { organizer: membership.organizer, profile, membership };
}

export async function GET(
    _req: NextRequest,
    { params }: { params: Promise<{ id: string }> },
) {
    try {
        const ctx = await requireOrganizer();
        if ("error" in ctx) {
            const status =
                ctx.error === "UNAUTHENTICATED" ? 401 : ctx.error === "PROFILE_NOT_FOUND" ? 404 : 403;
            return NextResponse.json({ ok: false, error: ctx.error }, { status });
        }

        const { id } = await params;
        const promoId = Number(id);
        if (!Number.isFinite(promoId)) {
            return NextResponse.json({ ok: false, error: "BAD_REQUEST" }, { status: 400 });
        }
    }
}

```

```

};

const organizerEvents = await prisma.event.findMany({
  where: { organizerId: ctx.organizer.id },
  select: { id: true, title: true, slug: true },
});
const eventIds = organizerEvents.map((e) => e.id);

const promo = await prisma.promoCode.findUnique({
  where: { id: promoid },
  include: {
    redemptions: {
      orderBy: { usedAt: "desc" },
      take: 25,
    },
  },
});

if (!promo) {
  return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
}

if (promo.eventId) {
  if (!eventIds.includes(promo.eventId)) {
    return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
  }
}

// Métrica simples de novos vs recorrentes: compara a data do primeiro sale_summary do user
const firstRedemptionPerUser = new Map<string, Date>();
promo.redemptions.forEach((r) => {
  if (!r.userId) return;
  const prev = firstRedemptionPerUser.get(r.userId);
  if (!prev || r.usedAt < prev) {
    firstRedemptionPerUser.set(r.userId, r.usedAt);
  }
});
let newUsers = 0;
let returningUsers = 0;
if (firstRedemptionPerUser.size > 0) {
  const firstPurchases = await prisma.saleSummary.groupBy({
    by: ["userId"],
    where: {
      userId: { in: Array.from(firstRedemptionPerUser.keys()) },
      eventId: { in: eventIds },
    },
    _min: { createdAt: true },
  });
  const firstPurchaseMap = new Map<string, Date>();
  firstPurchases.forEach((row) => {
    if (row.userId && row._min.createdAt) {
      firstPurchaseMap.set(row.userId, row._min.createdAt);
    }
  });
  firstRedemptionPerUser.forEach((firstUse, uid) => {
    const firstSale = firstPurchaseMap.get(uid);
    if (!firstSale || firstSale.getTime() >= firstUse.getTime() - 1000) {
      newUsers += 1;
    } else {
      returningUsers += 1;
    }
  });
}

const saleLines = await prisma.saleLine.findMany({
  where: {
    eventId: { in: eventIds },
    OR: [{ promoCodeId: promo.id }, { promoCodeSnapshot: promo.code }],
  },
  select: {
    quantity: true,
    grossCents: true,
    discountPerUnitCents: true,
    netCents: true,
    eventId: true,
  },
});

```

```

const agg = saleLines.reduce(
  (acc, l) => {
    const qty = l.quantity ?? 0;
    acc.tickets += qty;
    acc.grossCents += l.grossCents ?? 0;
    acc.discountCents += (l.discountPerUnitCents ?? 0) * qty;
    acc.netCents += l.netCents ?? 0;
    return acc;
  },
  { tickets: 0, grossCents: 0, discountCents: 0, netCents: 0 },
);

const usersUnique = new Set<string>();
promo.redemptions.forEach((r) => {
  if (r.userId) usersUnique.add(r.userId);
  else if (r.guestEmail) usersUnique.add(r.guestEmail.toLowerCase());
});

const history = promo.redemptions.map((r) => ({
  id: r.id,
  usedAt: r.usedAt,
  discountCents: 0,
  items: 0,
  userLabel: r.userId ? "Utilizador ORYA" : r.guestEmail || "Guest",
  event: null,
}));

// Top eventos por usos
const eventUses = new Map<number, number>();
saleLines.forEach((l) => {
  if (!l.eventId) return;
  eventUses.set(l.eventId, (eventUses.get(l.eventId) ?? 0) + (l.quantity ?? 0));
});
const topEvents = Array.from(eventUses.entries())
  .map(([id, uses]) => {
    const meta = organizerEvents.find((e) => e.id === id);
    return { id, title: meta?.title ?? "Evento", slug: meta?.slug ?? null, uses };
  })
  .sort((a, b) => b.uses - a.uses)
  .slice(0, 5);

return NextResponse.json(
  {
    ok: true,
    promo: {
      id: promo.id,
      code: promo.code,
      name: promo.name,
      description: promo.description,
      type: promo.type,
      value: promo.value,
      active: promo.active,
      autoApply: promo.autoApply,
      validFrom: promo.validFrom,
      validUntil: promo.validUntil,
      minQuantity: promo.minQuantity,
      minTotalCents: promo.minTotalCents,
      minCartValueCents: promo.minCartValueCents,
      maxUses: promo.maxUses,
      perUserLimit: promo.perUserLimit,
      eventId: promo.eventId,
      organizerId: promo.organizerId,
      createdAt: promo.createdAt,
      updatedAt: promo.updatedAt,
    },
    stats: {
      usesTotal: promo.redemptions.length,
      usersUnique: usersUnique.size,
      tickets: agg.tickets,
      grossCents: agg.grossCents,
      discountCents: agg.discountCents,
      netCents: agg.netCents,
      newUsers,
      returningUsers,
    },
    topEvents,
    history,
  }
);

```

```

        },
        { status: 200 },
    );
} catch (err) {
    console.error("[organizador/promo/:id] [GET]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/organizador/promo/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";
import { OrganizerMemberRole } from "@prisma/client";

async function requireOrganizer() {
    const supabase = await createSupabaseServer();
    const {
        data: { user },
        error,
    } = await supabase.auth.getUser();

    if (error || !user) {
        return { error: "UNAUTHENTICATED" as const };
    }

    const profile = await prisma.profile.findUnique({ where: { id: user.id } });
    if (!profile) return { error: "PROFILE_NOT_FOUND" as const };

    const membership = await prisma.organizerMember.findFirst({
        where: {
            userId: user.id,
            organizer: { status: "ACTIVE" },
        },
        include: { organizer: true },
        orderBy: [{ lastUsedAt: "desc" }, { createdAt: "asc" }],
    });

    if (!membership || !membership.organizer || !isOrgAdminOrAbove(membership.role as OrganizerMemberRole)) {
        return { error: "ORGANIZER_NOT_FOUND" as const };
    }

    return { organizer: membership.organizer, profile, membership };
}

export async function GET() {
    try {
        const ctx = await requireOrganizer();
        if ("error" in ctx) {
            const status =
                ctx.error === "UNAUTHENTICATED" ? 401 : ctx.error === "PROFILE_NOT_FOUND" ? 404 : 403;
            return NextResponse.json({ ok: false, error: ctx.error }, { status });
        }

        const promoRepo = (prisma as unknown as {
            promoCode?: {
                findMany: typeof prisma.promoCode.findMany;
            };
        }).promoCode;
        if (!promoRepo) {
            return NextResponse.json(
                { ok: false, error: "Promo codes indisponíveis nesta instância do Prisma." },
                { status: 500 },
            );
        }

        const organizerEvents = await prisma.event.findMany({
            where: { organizerId: ctx.organizer.id },
            select: { id: true, title: true, slug: true },
        });
        const eventIds = organizerEvents.map((e) => e.id);

        const promoCodes = await prisma.promoCode.findMany({
            where: {

```

```

    OR: [
      { eventId: null }, // global
      { eventId: { in: eventIds } },
    ],
  },
  orderBy: { createdAt: "desc" },
  include: {
    redemptions: true,
  },
);
};

const promoIds = promoCodes.map((p) => p.id);
const promoCodesList = promoCodes.map((p) => p.code);

const validRedemptionSummaryIds = new Set<number>();
promoCodes.forEach((p) => {
  p.redemptions.forEach((r) => {
    if (r.saleSummaryId) validRedemptionSummaryIds.add(r.saleSummaryId);
  });
});

const lines = await prisma.saleLine.findMany({
  where: {
    eventId: { in: eventIds },
    OR: [
      { promoCodeId: { in: promoIds } },
      { promoCodeSnapshot: { in: promoCodesList } },
    ],
    ...(validRedemptionSummaryIds.size > 0
      ? { saleSummaryId: { in: Array.from(validRedemptionSummaryIds) } }
      : {}),
  },
  select: {
    promoCodeId: true,
    promoCodeSnapshot: true,
    quantity: true,
    grossCents: true,
    netCents: true,
    discountPerUnitCents: true,
    platformFeeCents: true,
  },
});
};

type PromoAgg = {
  tickets: number;
  grossCents: number;
  discountCents: number;
  platformFeeCents: number;
  netCents: number;
};

const statsMap = new Map<
  string | number,
  PromoAgg & { users: Set<string>; redemptions: number }
>();
const ensureAgg = (key: string | number) => {
  const existing = statsMap.get(key);
  if (existing) return existing;
  const base: PromoAgg & { users: Set<string>; redemptions: number } = {
    tickets: 0,
    grossCents: 0,
    discountCents: 0,
    platformFeeCents: 0,
    netCents: 0,
    users: new Set(),
    redemptions: 0,
  };
  statsMap.set(key, base);
  return base;
};

for (const l of lines) {
  const key = l.promoCodeId ?? l.promoCodeSnapshot ?? "unknown";
  const agg = ensureAgg(key);
  const qty = l.quantity ?? 0;
  const discountLine = (l.discountPerUnitCents ?? 0) * qty;
  agg.tickets += qty;
  agg.grossCents += l.grossCents ?? 0;
}

```

```

        agg.discountCents += discountLine;
        agg.platformFeeCents += l.platformFeeCents ?? 0;
        agg.netCents += l.netCents ?? 0;
    }

    // Contar redemptions (uses) e users únicos
    promoCodes.forEach((p) => {
        const agg = ensureAgg(p.id);
        agg.redemptions += p.redemptions.length;
        p.redemptions.forEach((r) => {
            if (r.userId) agg.users.add(r.userId);
            else if (r.guestEmail) agg.users.add(r.guestEmail.toLowerCase());
        });
    });

    return NextResponse.json({
        ok: true,
        promoCodes: promoCodes.map((p) => ({
            ...p,
            status:
                !p.active
                ? "INACTIVE"
                : p.validUntil && new Date(p.validUntil) < new Date()
                ? "EXPIRED"
                : "ACTIVE",
            redemptionsCount: statsMap.get(p.id)?.tickets ?? p.redemptions.length,
        })),
        events: organizerEvents,
        promoStats: promoCodes.map((p) => {
            const agg =
                statsMap.get(p.id) ||
                statsMap.get(p.code) ||
                {
                    tickets: 0,
                    grossCents: 0,
                    discountCents: 0,
                    platformFeeCents: 0,
                    netCents: 0,
                    users: new Set<string>(),
                    redemptions: 0,
                };
            return {
                promoCodeId: p.id,
                tickets: agg.tickets,
                grossCents: agg.grossCents,
                discountCents: agg.discountCents,
                platformFeeCents: agg.platformFeeCents,
                netCents: agg.netCents,
                usesTotal: agg.redemptions,
                usersUnique: agg.users.size,
                totalSavedCents: agg.discountCents,
            };
        });
    });
}

} catch (err) {
    console.error("[organizador/promo][GET]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

export async function POST(req: NextRequest) {
    try {
        const ctx = await requireOrganizer();
        if ("error" in ctx) {
            const status =
                ctx.error === "UNAUTHENTICATED" ? 401 : ctx.error === "PROFILE_NOT_FOUND" ? 404 : 403;
            return NextResponse.json({ ok: false, error: ctx.error }, { status });
        }

        const body = await req.json().catch(() => null);
        if (!body) {
            return NextResponse.json({ ok: false, error: "BAD_REQUEST" }, { status: 400 });
        }

        const promoRepo = (prisma as unknown as {
            promoCode?: {
                findFirst: typeof prisma.promoCode.findFirst;
                create: typeof prisma.promoCode.create;
            };
        }).promoCode;
    }
}

```

```

if (!promoRepo) {
  return NextResponse.json(
    { ok: false, error: "Promo codes indisponíveis nesta instância do Prisma." },
    { status: 500 },
  );
}

const {
  code,
  type,
  value,
  maxUses,
  perUserLimit,
  validFrom,
  validUntil,
  eventId,
  active,
  autoApply,
  minQuantity,
  minTotalCents,
  name,
  description,
  minCartValueCents,
} = body as {
  code?: string;
  type?: "PERCENTAGE" | "FIXED";
  value?: number;
  maxUses?: number | null;
  perUserLimit?: number | null;
  validFrom?: string | null;
  validUntil?: string | null;
  eventId?: number | null;
  active?: boolean;
  autoApply?: boolean;
  minQuantity?: number | null;
  minTotalCents?: number | null;
  name?: string | null;
  description?: string | null;
  minCartValueCents?: number | null;
};

const cleanCode = (code || "").trim();
const auto = Boolean(autoApply);
// Se autoApply e sem código, gerar um placeholder interno
const finalCode =
  auto && !cleanCode
  ? `AUTO-${Math.random().toString(36).slice(2, 8).toUpperCase()}` 
  : cleanCode;
if (!finalCode) {
  return NextResponse.json({ ok: false, error: "Código em falta." }, { status: 400 });
}
if (type !== "PERCENTAGE" && type !== "FIXED") {
  return NextResponse.json({ ok: false, error: "Tipo inválido." }, { status: 400 });
}
const cleanValue = Number(value);
if (!Number.isFinite(cleanValue) || cleanValue <= 0) {
  return NextResponse.json({ ok: false, error: "Valor inválido." }, { status: 400 });
}

let targetEventId: number | null = null;
if (eventId !== null && eventId !== undefined) {
  const exists = await prisma.event.findFirst({
    where: { id: Number(eventId), organizerId: ctx.organizer.id },
    select: { id: true },
  });
  if (!exists) {
    return NextResponse.json(
      { ok: false, error: "Evento inválido ou não pertence ao organizador." },
      { status: 400 },
    );
  }
  targetEventId = Number(eventId);
}

const parseDate = (d?: string | null) => {
  if (!d) return null;
  const dt = new Date(d);
  return Number.isNaN(dt.getTime()) ? null : dt;
}

```

```

};

const created = await prisma.promoCode.create({
  data: {
    code: finalCode,
    type,
    value: Math.floor(cleanValue),
    maxUses: maxUses ?? null,
    perUserLimit: perUserLimit ?? null,
    validFrom: parseDate(validFrom),
    validUntil: parseDate(validUntil),
    eventId: targetEventId,
    active: active ?? true,
    autoApply: auto,
    minQuantity: minQuantity ?? null,
    minTotalCents: minTotalCents ?? null,
  },
});

return NextResponse.json({ ok: true, promoCode: created }, { status: 200 });
} catch (err) {
  console.error("[organizador/promo][POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

export async function PATCH(req: NextRequest) {
  try {
    const ctx = await requireOrganizer();
    if ("error" in ctx) {
      const status =
        ctx.error === "UNAUTHENTICATED" ? 401 : ctx.error === "PROFILE_NOT_FOUND" ? 404 : 403;
      return NextResponse.json({ ok: false, error: ctx.error }, { status });
    }

    const body = await req.json().catch(() => null);
    if (!body || typeof body.id !== "number") {
      return NextResponse.json({ ok: false, error: "BAD_REQUEST" }, { status: 400 });
    }

    const promoRepo = (prisma as unknown as {
      promoCode?: {
        findUnique: typeof prisma.promoCode.findUnique;
        update: typeof prisma.promoCode.update;
      };
    }).promoCode;
    if (!promoRepo) {
      return NextResponse.json(
        { ok: false, error: "Promo codes indisponíveis nesta instância do Prisma." },
        { status: 500 },
      );
    }

    const promo = await promoRepo.findUnique({ where: { id: body.id } });
    if (!promo) {
      return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
    }

    // garantir que pertence ao organizer (ou global)
    if (promo.eventId) {
      const evt = await prisma.event.findFirst({
        where: { id: promo.eventId, organizerId: ctx.organizer.id },
        select: { id: true },
      });
      if (!evt) {
        return NextResponse.json(
          { ok: false, error: "FORBIDDEN" },
          { status: 403 },
        );
      }
    }

    const {
      active,
      autoApply,
      code,
      type,
      value,
    }
  }
}

```

```

maxUses,
perUserLimit,
validFrom,
validUntil,
eventId,
minQuantity,
minTotalCents,
minCartValueCents,
name,
description,
} = body as {
  active?: boolean;
  autoApply?: boolean;
  code?: string;
  type?: "PERCENTAGE" | "FIXED";
  value?: number;
  maxUses?: number | null;
  perUserLimit?: number | null;
  validFrom?: string | null;
  validUntil?: string | null;
  eventId?: number | null;
  minQuantity?: number | null;
  minTotalCents?: number | null;
  minCartValueCents?: number | null;
  name?: string | null;
  description?: string | null;
};

let targetEventId: number | null | undefined = undefined;
if (eventId !== undefined && eventId !== null) {
  const exists = await prisma.event.findFirst({
    where: { id: Number(eventId), organizerId: ctx.organizer.id },
    select: { id: true },
  });
  if (!exists) {
    return NextResponse.json(
      { ok: false, error: "Evento inválido ou não pertence ao organizador." },
      { status: 400 },
    );
  }
  targetEventId = Number(eventId);
} else if (eventId === null) {
  targetEventId = null;
}

const parseDate = (d?: string | null) => {
  if (!d) return null;
  const dt = new Date(d);
  return Number.isNaN(dt.getTime()) ? null : dt;
};

const dataUpdate: Record<string, unknown> = {};
if (typeof active === "boolean") dataUpdate.active = active;
if (typeof autoApply === "boolean") dataUpdate.autoApply = autoApply;
if (typeof code === "string" && code.trim()) dataUpdate.code = code.trim();
if (type === "PERCENTAGE" || type === "FIXED") dataUpdate.type = type;
if (typeof value === "number" && Number.isFinite(value) && value > 0) {
  dataUpdate.value = Math.floor(value);
}
if (maxUses !== undefined) dataUpdate.maxUses = maxUses;
if (perUserLimit !== undefined) dataUpdate.perUserLimit = perUserLimit;
if (validFrom !== undefined) dataUpdate.validFrom = parseDate(validFrom);
if (validUntil !== undefined) dataUpdate.validUntil = parseDate(validUntil);
if (targetEventId !== undefined) dataUpdate.eventId = targetEventId;
if (minQuantity !== undefined) dataUpdate.minQuantity = minQuantity;
if (minTotalCents !== undefined) dataUpdate.minTotalCents = minTotalCents;
if (minCartValueCents !== undefined) dataUpdate.minCartValueCents = minCartValueCents;
if (typeof name === "string") dataUpdate.name = name.trim() || null;
if (typeof description === "string") dataUpdate.description = description.trim() || null;

const updated = await prisma.promoCode.update({
  where: { id: promo.id },
  data: dataUpdate,
});

return NextResponse.json({ ok: true, promoCode: updated }, { status: 200 });
} catch (err) {
  console.error("[organizador/promo][PATCH]", err);
}

```

```

        return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
    }

}

export async function DELETE(req: NextRequest) {
    try {
        const ctx = await requireOrganizer();
        if ("error" in ctx) {
            const status =
                ctx.error === "UNAUTHENTICATED" ? 401 : ctx.error === "PROFILE_NOT_FOUND" ? 404 : 403;
            return NextResponse.json({ ok: false, error: ctx.error }, { status });
        }
        const body = await req.json().catch(() => null);
        if (!body || typeof body.id !== "number") {
            return NextResponse.json({ ok: false, error: "BAD_REQUEST" }, { status: 400 });
        }

        const promo = await prisma.promoCode.findUnique({ where: { id: body.id } });
        if (!promo) {
            return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
        }
        if (promo.eventId) {
            const evt = await prisma.event.findFirst({
                where: { id: promo.eventId, organizerId: ctx.organizer.id },
                select: { id: true },
            });
            if (!evt) {
                return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
            }
        }

        await prisma.promoCode.delete({
            where: { id: promo.id },
        });
        return NextResponse.json({ ok: true }, { status: 200 });
    } catch (err) {
        console.error("[organizador/promo][DELETE]", err);
        return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
    }
}

```

app/api/organizador/staff/assign/route.ts

```

// app/api/organizador/staff/assign/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";
import { StaffRole } from "@prisma/client";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";

type AssignStaffBody = {
    userId?: string;
    emailOrUsername?: string;
    scope?: "GLOBAL" | "EVENT";
    eventId?: number;
    organizerId?: number;
    role?: StaffRole;
};

export async function POST(req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const user = await ensureAuthenticated(supabase);

        // Ler e validar body
        let body: AssignStaffBody | null = null;
        try {
            body = (await req.json()) as AssignStaffBody;
        } catch {
            return NextResponse.json(
                { ok: false, error: "Body inválido." },
                { status: 400 }
            );
        }
    }
}

```

```

}

const { userId, emailOrUsername, scope, eventId, role, organizerId: organizerIdRaw } = body || {};

if (!scope || (scope !== "GLOBAL" && scope !== "EVENT")) {
  return NextResponse.json(
    { ok: false, error: "scope inválido. Use 'GLOBAL' ou 'EVENT'." },
    { status: 400 }
  );
}

const allowedRoles: StaffRole[] = ["OWNER", "ADMIN", "STAFF", "CHECKIN"];
const chosenRole: StaffRole = allowedRoles.includes(role as StaffRole) ? (role as StaffRole) : "STAFF";

if (scope === "EVENT" && !eventId) {
  return NextResponse.json(
    {
      ok: false,
      error: "eventId é obrigatório quando o scope é 'EVENT'.",
    },
    { status: 400 }
  );
}

// Buscar profile do utilizador autenticado
const profile = await prisma.profile.findUnique({
  where: { id: user.id },
});

if (!profile) {
  return NextResponse.json(
    {
      ok: false,
      error: "Perfil não encontrado. Completa o onboarding antes de gerir staff.",
    },
    { status: 400 }
  );
}

// Resolver organizerId: vir no payload ou através do evento
let organizerId = Number(organizerIdRaw);
if (scope === "EVENT" && eventId) {
  const event = await prisma.event.findUnique({
    where: { id: Number(eventId) },
    select: { id: true, organizerId: true, status: true, endsAt: true },
  });
  if (!event) {
    return NextResponse.json({ ok: false, error: "Evento não encontrado." }, { status: 404 });
  }
  organizerId = event.organizerId;
  if (event.status !== "PUBLISHED" || (event.endsAt && event.endsAt < new Date())) {
    return NextResponse.json({ ok: false, error: "Evento inativo para atribuir staff." }, { status: 400 });
  }
}

if (!Number.isFinite(organizerId)) {
  return NextResponse.json(
    { ok: false, error: "organizerId é obrigatório." },
    { status: 400 }
  );
}

// Validar membership do caller
const callerMembership = await prisma.organizerMember.findUnique({
  where: { organizerId_userId: { organizerId, userId: user.id } },
});
if (!callerMembership || !isOrgAdminOrAbove(callerMembership.role)) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}

const organizer = await prisma.organizer.findUnique({
  where: { id: organizerId },
  select: { id: true, status: true },
});
if (!organizer || organizer.status !== "ACTIVE") {
  return NextResponse.json(
    { ok: false, error: "Organização inativa ou inexistente." },
    { status: 404 }
  );
}

```

```

        { status: 404 }
    );
}

// Resolver user alvo (permite userId direto OU email/username)
let targetProfile = null;
if (userId) {
    targetProfile = await prisma.profile.findUnique({
        where: { id: userId },
    });
} else if (emailOrUsername) {
    const normalized = emailOrUsername.trim().replace(/^@/, "");
    targetProfile = await prisma.profile.findFirst({
        where: {
            OR: [{ username: normalized }, { fullName: normalized }],
        },
    });
}

if (!targetProfile) {
    return NextResponse.json(
        {
            ok: false,
            error: "Utilizador alvo (staff) não encontrado.",
        },
        { status: 404 }
    );
}

// Se vier userId vazio mas email/username preenchido, usar o encontrado
const targetUserId = targetProfile.id;

// Validar evento se scope EVENT (e garantir que pertence ao organizer)
let targetEventId: number | null = null;
if (scope === "EVENT") {
    targetEventId = Number(eventId);
    if (!Number.isFinite(targetEventId)) {
        return NextResponse.json(
            { ok: false, error: "eventId inválido." },
            { status: 400 },
        );
    }

    if (targetEventId) {
        const event = await prisma.event.findFirst({
            where: {
                id: targetEventId,
                organizerId,
                status: "PUBLISHED",
                endsAt: { gte: new Date() },
            },
            select: { id: true },
        });

        if (!event) {
            return NextResponse.json(
                { ok: false, error: "Evento não encontrado ou inativo para este organizador." },
                { status: 404 },
            );
        }
    }
}

// Procurar assignment existente (para não duplicar)
const existing = await prisma.staffAssignment.findFirst({
    where: {
        organizerId,
        userId: targetUserId,
        scope,
        ...(scope === "EVENT" ? { eventId: targetEventId ?? undefined } : {}),
    },
});

let assignment;

if (existing) {
    assignment = await prisma.staffAssignment.update({
        where: { id: existing.id },

```

```

        data: {
          revokedAt: null,
          acceptedAt: null,
          status: "PENDING",
          scope,
          eventId: scope === "EVENT" ? targetEventId ?? null : null,
          userId: targetUserId,
          role: chosenRole,
        },
      );
    } else {
      assignment = await prisma.staffAssignment.create({
        data: {
          organizerId,
          userId: targetUserId,
          scope,
          eventId: scope === "EVENT" ? targetEventId ?? null : null,
          status: "PENDING",
          role: chosenRole,
        },
      });
    }
  }

  return NextResponse.json(
    {
      ok: true,
      assignment,
    },
    { status: 200 }
  );
} catch (err) {
  console.error("POST /api/organizador/staff/assign error:", err);
  return NextResponse.json(
    {
      ok: false,
      error: "Erro interno ao atribuir staff.",
    },
    { status: 500 }
  );
}
}
}

```

app/api/organizador/staff/list/route.ts

```

// app/api/organizador/staff/list/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";

export async function GET(_req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    const organizerProfile = await prisma.profile.findUnique({
      where: { id: user.id },
    });

    if (!organizerProfile) {
      return NextResponse.json(
        { ok: false, error: "Perfil não encontrado." },
        { status: 400 },
      );
    }

    const { organizer, membership } = await getActiveOrganizerForUser(user.id, {
      roles: ["OWNER", "CO_OWNER", "ADMIN"],
    });

    if (!organizer || !membership || !isOrgAdminOrAbove(membership.role)) {
      return NextResponse.json(
        { ok: false, error: "Ainda não é organizador." },
        { status: 403 },
      );
    }
  }
}

```

```

    );
}

const assignments = await prisma.staffAssignment.findMany({
  where: { organizerId: organizer.id },
  include: {
    event: true,
    organizer: false,
  },
  orderBy: { createdAt: "desc" },
});

const staffProfiles = await prisma.profile.findMany({
  where: { id: { in: assignments.map((a) => a.userId) } },
});
const staffMap = Object.fromEntries(staffProfiles.map((p) => [p.id, p]));

const items = assignments.map((a) => ({
  id: a.id,
  userId: a.userId,
  scope: a.scope,
  eventId: a.eventId,
  createdAt: a.createdAt,
  revokedAt: a.revokedAt,
  status: a.revokedAt ? "REMOVED" : a.status,
  userName: staffMap[a.userId]?.fullName ?? staffMap[a.userId]?.username ?? null,
  userEmail: staffMap[a.userId]?.email ?? null,
  eventTitle: a.event?.title ?? null,
  role: a.role,
}));


return NextResponse.json({ ok: true, items }, { status: 200 });
} catch (err) {
  console.error("GET /api/organizador/staff/list error:", err);
  return NextResponse.json(
    { ok: false, error: "Erro ao carregar staff." },
    { status: 500 },
  );
}
}
}

```

app/api/organizador/staff/revoke/route.ts

```

// app/api/organizador/staff/revoke/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";

type RevokeStaffBody = {
  assignmentId?: number;
};

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    let body: RevokeStaffBody | null = null;
    try {
      body = (await req.json()) as RevokeStaffBody;
    } catch {
      return NextResponse.json(
        { ok: false, error: "Body inválido." },
        { status: 400 }
      );
    }

    const { assignmentId } = body;

    if (!assignmentId || typeof assignmentId !== "number") {
      return NextResponse.json(
        { ok: false, error: "assignmentId é obrigatório." },
        { status: 400 }
      );
    }
  }
}

```

```

const organizerProfile = await prisma.profile.findUnique({
  where: { id: user.id },
});

if (!organizerProfile) {
  return NextResponse.json(
    {
      ok: false,
      error:
        "Perfil não encontrado. Completa o onboarding antes de gerir staff.",
    },
    { status: 400 }
  );
}

const organizer = await prisma.organizer.findFirst({
  where: { userId: organizerProfile.id },
});

if (!organizer) {
  return NextResponse.json(
    {
      ok: false,
      error: "Ainda não és organizador. Não podes gerir staff.",
    },
    { status: 403 }
  );
}

const existing = await prisma.staffAssignment.findFirst({
  where: { id: assignmentId, organizerId: organizer.id },
});

if (!existing) {
  return NextResponse.json(
    { ok: false, error: "Assignment de staff não encontrado." },
    { status: 404 }
  );
}

await prisma.staffAssignment.update({
  where: { id: existing.id },
  data: { revokedAt: new Date() },
});

return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
  console.error("POST /api/organizador/staff/revoke error:", err);
  return NextResponse.json(
    { ok: false, error: "Erro interno ao revogar staff." },
    { status: 500 }
  );
}
}
}

```

app/api/organizador/tournaments/[id]/broadcast/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { notifyBroadcast } from "@domain/notifications/producer";

export async function POST(req: NextRequest, { params }: { params: { id: string } }) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const tournamentId = Number(params?.id);
  if (!Number.isFinite(tournamentId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

```

```

const body = (await req.json().catch(() => null)) as { message?: string; audienceKey?: string } | null;
const message = body?.message?.trim();
const audienceKey = body?.audienceKey?.trim() || "ALL";

const tournament = await prisma.tournament.findUnique({
  where: { id: tournamentId },
  select: { organizerId: true, eventId: true },
});
if (!tournament?.organizerId) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

const { organizer } = await getActiveOrganizerForUser(user.id, {
  organizerId: tournament.organizerId,
  roles: ["OWNER", "CO_OWNER", "ADMIN"],
});
if (!organizer) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

const entries = await prisma.tournamentEntry.findMany({
  where: { eventId: tournament.eventId },
  select: { userId: true },
});
const audienceUserIds = Array.from(new Set(entries.map((e) => e.userId).filter(Boolean) as string[]));

if (audienceUserIds.length) {
  await Promise.all(
    audienceUserIds.map((uid) =>
      notifyBroadcast({
        userId: uid,
        tournamentId,
        broadcastId: crypto.randomUUID(),
        audienceKey,
      }),
    ),
  );
}

// Armazena feed de avisos (simples)
await prisma.notificationOutbox.create({
  data: {
    notificationType: "BROADCAST",
    dedupeKey: `${tournamentId}:BROADCAST:${Date.now()}`,
    payload: { message: message || "Aviso do organizador", tournamentId },
    status: "PENDING",
  },
});

return NextResponse.json({ ok: true, audienceCount: audienceUserIds.length }, { status: 200 });
}

```

app/api/organizador/tournaments/[id]/finance/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { computeReleaseAt, computeHold } from "@domain/finance/payoutPolicy";

async function ensureOrganizerAccess(userId: string, eventId: number) {
  const evt = await prisma.event.findUnique({ where: { id: eventId }, select: { organizerId: true, isTest: true } });
  if (!evt?.organizerId) return { ok: false, isTest: false };
  const member = await prisma.organizerMember.findFirst({
    where: { organizerId: evt.organizerId, userId, role: { in: ["OWNER", "CO_OWNER", "ADMIN"] } },
    select: { id: true },
  });
  return { ok: Boolean(member), isTest: evt.isTest ?? false };
}

export async function GET(req: NextRequest, { params }: { params: { id: string } }) {
  const tournamentId = Number(params?.id);
  if (!Number.isFinite(tournamentId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const tournament = await prisma.tournament.findUnique({
    where: { id: tournamentId },
    select: { eventId: true },
  }

```

```

    });

    if (!tournament) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

    const access = await ensureOrganizerAccess(data.user.id, tournament.eventId);
    if (!access.ok) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

    const includeTest = req.nextUrl.searchParams.get("includeTest") === "1";
    if (!includeTest && access.isTest) {
        return NextResponse.json({ ok: false, error: "TEST_EVENT" }, { status: 400 });
    }

    const sales = await prisma.saleSummary.groupBy({
        by: ["eventId"],
        where: { eventId: tournament.eventId },
        _sum: { totalCents: true, netCents: true, platformFeeCents: true },
        _count: { _all: true },
    });
    const agg = sales[0] || {
        _sum: { totalCents: 0, netCents: 0, platformFeeCents: 0 },
        _count: { _all: 0 },
    };

    const recent = await prisma.saleSummary.findMany({
        where: { eventId: tournament.eventId },
        orderBy: { createdAt: "desc" },
        take: 20,
        select: {
            id: true,
            paymentIntentId: true,
            totalCents: true,
            netCents: true,
            platformFeeCents: true,
            currency: true,
            createdAt: true,
            purchaseId: true,
        },
    });
}

const event = await prisma.event.findUnique({ where: { id: tournament.eventId }, select: { endsAt: true, payoutMode: true } });
const releaseAt = computeReleaseAt(event?.endsAt ?? null);
const hold = computeHold(agg._sum.totalCents ?? 0, false); // sem disputes implementadas aqui

// Placeholder refunds/disputes (não temos tabelas dedicadas aqui)
const refundsCents = 0;
const disputesCents = 0;

return NextResponse.json(
{
    ok: true,
    summary: {
        totalCents: agg._sum.totalCents ?? 0,
        netCents: agg._sum.netCents ?? 0,
        platformFeeCents: agg._sum.platformFeeCents ?? 0,
        countSales: agg._count._all ?? 0,
        refundsCents,
        disputesCents,
        releaseAt,
        holdCents: hold.holdCents,
        holdReason: hold.reason,
        payoutMode: event?.payoutMode ?? null,
    },
    recent,
},
{ status: 200 },
);
}

```

app/api/organizador/tournaments/[id]/generate/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@lib/supabaseServer";
import { generateAndPersistTournamentStructure, getConfirmedPairings } from "@domain/tournaments/generation";
import { prisma } from "@lib/prisma";
import { TournamentFormat } from "@prisma/client";

```

```

async function isOrganizerUser(userId: string, organizerId: number) {
  const member = await prisma.organizerMember.findFirst({
    where: {
      organizerId,
      userId,
      role: { in: ["OWNER", "CO_OWNER", "ADMIN"] },
    },
    select: { id: true },
  });
  return Boolean(member);
}

export async function POST(req: NextRequest, { params }: { params: { id: string } }) {
  const id = Number(params?.id);
  if (!Number.isFinite(id)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const tournament = await prisma.tournament.findUnique({
    where: { id },
    include: { event: { select: { organizerId: true } } },
  });
  if (!tournament) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

  if (!tournament.event.organizerId) {
    return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 400 });
  }

  const authorized = await isOrganizerUser(data.user.id, tournament.event.organizerId);
  if (!authorized) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

  const body = await req.json().catch(() => ({}));
  const format = (body?.format as TournamentFormat | undefined) ?? tournament.format;
  const seed = typeof body?.seed === "string" ? body.seed : null;
  const forceGenerate = body?.forceGenerate === true;

  const pairingIds = await getConfirmedPairings(tournament.eventId);

  try {
    const result = await generateAndPersistTournamentStructure({
      tournamentId: tournament.id,
      format,
      pairings: pairingIds,
      seed,
      inscriptionDeadlineAt: tournament.inscriptionDeadlineAt,
      forceGenerate,
      userId: data.user.id,
    });

    return NextResponse.json(
      { ok: true, stagesCreated: result.stagesCreated, matchesCreated: result.matchesCreated, seed: result.seed },
      { status: 200 },
    );
  } catch (err) {
    if (err instanceof Error && err.message === "TOURNAMENT_ALREADY_STARTED") {
      return NextResponse.json({ ok: false, error: "TOURNAMENT_ALREADY_STARTED" }, { status: 409 });
    }
    if (err instanceof Error && err.message === "INSCRIPTION_NOT_CLOSED") {
      return NextResponse.json({ ok: false, error: "INSCRIPTION_NOT_CLOSED" }, { status: 409 });
    }
    console.error("[tournament_generate] erro", err);
    return NextResponse.json({ ok: false, error: "GENERATION_FAILED" }, { status: 500 });
  }
}

```

app/api/organizador/tournaments/[id]/live/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { getTournamentStructure, summarizeMatchStatus, computeStandingsForGroup } from "@/domain/tournaments/structure";
import { computeLiveWarnings } from "@/domain/tournaments/liveWarnings";

async function ensureOrganizerAccess(userId: string, eventId: number) {

```

```

const evt = await prisma.event.findUnique({
  where: { id: eventId },
  select: { organizerId: true },
});
if (!evt?.organizerId) return false;
const member = await prisma.organizerMember.findFirst({
  where: {
    organizerId: evt.organizerId,
    userId,
    role: { in: ["OWNER", "CO_OWNER", "ADMIN"] },
  },
  select: { id: true },
});
return Boolean(member);
}

export async function GET(_req: NextRequest, { params }: { params: { id: string } }) {
  const id = Number(params.id);
  if (!Number.isFinite(id)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data: authData, error: authError } = await supabase.auth.getUser();
  if (authError || !authData?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const tournament = await getTournamentStructure(id);
  if (!tournament) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

  const authorized = await ensureOrganizerAccess(authData.user.id, tournament.event.id);
  if (!authorized) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

  const tieBreakRules = Array.isArray(tournament.tieBreakRules)
    ? (tournament.tieBreakRules as string[])
    : ["WINS", "SET_DIFF", "GAME_DIFF", "HEAD_TO_HEAD", "RANDOM"];

  // pairings para warnings REQUIRES_ACTION
  const pairings = await prisma.padelPairing.findMany({
    where: { eventId: tournament.event.id },
    select: { id: true, guaranteeStatus: true },
  });

  const stages = tournament.stages.map((s) => ({
    id: s.id,
    name: s.name,
    stageType: s.stageType,
    groups: s.groups.map((g) => ({
      id: g.id,
      name: g.name,
      standings: computeStandingsForGroup(g.matches, tieBreakRules, tournament.generationSeed || undefined),
      matches: g.matches.map((m) => ({
        id: m.id,
        pairing1Id: m.pairing1Id,
        pairing2Id: m.pairing2Id,
        round: m.round,
        roundLabel: m.roundLabel,
        startAt: m.startAt,
        courtId: m.courtId,
        status: m.status,
        statusLabel: summarizeMatchStatus(m.status),
        score: m.score,
        nextMatchId: m.nextMatchId,
        nextSlot: m.nextSlot,
      })),
    })),
  )),
  matches: s.matches
    .filter((m) => !m.groupId)
    .map((m) => ({
      id: m.id,
      pairing1Id: m.pairing1Id,
      pairing2Id: m.pairing2Id,
      round: m.round,
      roundLabel: m.roundLabel,
      startAt: m.startAt,
      courtId: m.courtId,
      status: m.status,
      statusLabel: summarizeMatchStatus(m.status),
      score: m.score,
      nextMatchId: m.nextMatchId,
      nextSlot: m.nextSlot,
    })),
}

```

```

        })),
    }));
}

const flatMatches = stages.flatMap((s) => [...s.matches, ...s.groups.flatMap((g) => g.matches)]);
const warnings = computeLiveWarnings({
  matches: flatMatches,
  pairings,
  startThresholdMinutes: 60,
});

const res = NextResponse.json(
  {
    ok: true,
    tournament: {
      id: tournament.id,
      event: tournament.event,
      format: tournament.format,
      stages,
    },
    warnings,
  },
  { status: 200 },
);
res.headers.set("Cache-Control", "no-store");
return res;
}

```

app/api/organizador/tournaments/[id]/matches/[matchId]/edit/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { TournamentMatchStatus } from "@prisma/client";

async function ensureOrganizerAccess(userId: string, eventId: number) {
  const evt = await prisma.event.findUnique({
    where: { id: eventId },
    select: { organizerId: true },
  });
  if (!evt?.organizerId) return false;
  const member = await prisma.organizerMember.findFirst({
    where: {
      organizerId: evt.organizerId,
      userId,
      role: { in: ["OWNER", "CO_OWNER", "ADMIN"] },
    },
    select: { id: true },
  });
  return Boolean(member);
}

export async function POST(req: NextRequest, { params }: { params: { id: string; matchId: string } }) {
  const tournamentId = Number(params?.id);
  const matchId = Number(params?.matchId);
  if (!Number.isFinite(tournamentId) || !Number.isFinite(matchId)) {
    return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });
  }

  const supabase = await createSupabaseServer();
  const { data: authData, error: authError } = await supabase.auth.getUser();
  if (authError || !authData?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const match = await prisma.tournamentMatch.findUnique({
    where: { id: matchId },
    include: {
      stage: { select: { tournamentId: true, tournament: { select: { eventId: true, generationSeed: true } } } },
    },
  });
  if (!match || match.stage.tournamentId !== tournamentId) {
    return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
  }

  const authorized = await ensureOrganizerAccess(authData.user.id, match.stage.tournament.eventId);
  if (!authorized) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

  const body = await req.json().catch(() => ({}));

```

```

const { startAt, courtId, status, score, roundLabel } = body ?? {};

const updates: Record<string, unknown> = {};
if (startAt) updates.startAt = new Date(startAt);
if (typeof courtId === "number") updates.courtId = courtId;
if (roundLabel) updates.roundLabel = roundLabel;
if (status && Object.values(TournamentMatchStatus).includes(status)) updates.status = status as TournamentMatchStatus;
if (score) updates.score = score;

if (Object.keys(updates).length === 0) {
  return NextResponse.json({ ok: false, error: "NO_CHANGES" }, { status: 400 });
}

const before = {
  startAt: match.startAt,
  courtId: match.courtId,
  status: match.status,
  score: match.score,
  roundLabel: match.roundLabel,
};

const updated = await prisma.$transaction(async (tx) => {
  const res = await tx.tournamentMatch.update({
    where: { id: matchId },
    data: updates,
  });

  await tx.tournamentAuditLog.create({
    data: {
      tournamentId,
      userId: authData.user.id,
      action: "EDIT_MATCH",
      payloadBefore: before,
      payloadAfter: updates,
    },
  });
});

return res;
});

return NextResponse.json({ ok: true, match: updated }, { status: 200 });
}

```

app/api/organizador/tournaments/[id]/matches/[matchId]/notify/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { computeDeduplicateKey } from "@/domain/notifications/matchChangeDedup";
import { canNotify } from "@/domain/tournaments/schedulePolicy";

async function ensureOrganizerAccess(userId: string, eventId: number) {
  const evt = await prisma.event.findUnique({ where: { id: eventId }, select: { organizerId: true } });
  if (!evt?.organizerId) return false;
  const member = await prisma.organizerMember.findFirst({
    where: { organizerId: evt.organizerId, userId, role: { in: ["OWNER", "CO_OWNER", "ADMIN"] } },
    select: { id: true },
  });
  return Boolean(member);
}

export async function POST(_req: NextRequest, { params }: { params: { id: string; matchId: string } }) {
  const tournamentId = Number(params?.id);
  const matchId = Number(params?.matchId);
  if (!Number.isFinite(tournamentId) || !Number.isFinite(matchId)) {
    return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });
  }

  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const match = await prisma.tournamentMatch.findUnique({
    where: { id: matchId },
    include: { stage: { select: { tournamentId: true, tournament: { select: { eventId: true } } } } },
  });

```

```

if (!match || match.stage.tournamentId !== tournamentId) {
  return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
}

const authorized = await ensureOrganizerAccess(data.user.id, match.stage.tournament.eventId);
if (!authorized) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

if (!canNotify(match.status)) {
  return NextResponse.json({ ok: false, error: "NOTIFY_BLOCKED" }, { status: 409 });
}

const dedupeKey = computeDedupeKey(match.id, match.startAt, match.courtId);
try {
  await prisma.matchNotification.create({
    data: {
      matchId: match.id,
      dedupeKey,
      payload: { matchId: match.id, startAt: match.startAt, courtId: match.courtId },
    },
  });
} catch (err) {
  // UNIQUE violation => já notificado
  return NextResponse.json({ ok: true, deduped: true }, { status: 200 });
}

// Aqui seria o envio real (push/email). Guardamos só registo.
return NextResponse.json({ ok: true, deduped: false }, { status: 200 });
}

```

app/api/organizador/tournaments/[id]/matches/[matchId]/result/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { updateMatchResult } from "@/domain/tournaments/matchUpdate";
import { TournamentMatchStatus } from "@prisma/client";

async function ensureOrganizerAccess(userID: string, eventId: number) {
  const evt = await prisma.event.findUnique({ where: { id: eventId }, select: { organizerId: true } });
  if (!evt?.organizerId) return false;
  const member = await prisma.organizerMember.findFirst({
    where: { organizerId: evt.organizerId, userId: userID, role: { in: ["OWNER", "CO_OWNER", "ADMIN"] } },
    select: { id: true },
  });
  return Boolean(member);
}

export async function POST(req: NextRequest, { params }: { params: { id: string; matchId: string } }) {
  const tournamentId = Number(params?.id);
  const matchId = Number(params?.matchId);
  if (!Number.isFinite(tournamentId) || !Number.isFinite(matchId)) {
    return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });
  }

  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const match = await prisma.tournamentMatch.findUnique({
    where: { id: matchId },
    include: { stage: { select: { tournamentId: true, tournament: { select: { eventId: true } } } } },
  });
  if (!match || match.stage.tournamentId !== tournamentId) {
    return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
  }

  const authorized = await ensureOrganizerAccess(data.user.id, match.stage.tournament.eventId);
  if (!authorized) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

  const body = await req.json().catch(() => ({}));
  const { score, status, winnerPairingId, expectedUpdatedAt, force } = body ?? {};

  try {
    const updated = await updateMatchResult({
      matchId,
      score,
    }
  
```

```

    status: status && Object.values(TournamentMatchStatus).includes(status) ? status : "DONE",
    explicitWinnerPairingId: winnerPairingId,
    expectedUpdatedAt: expectedUpdatedAt ? new Date(expectedUpdatedAt) : undefined,
    userId: data.user.id,
    force: force === true,
  });

  return NextResponse.json({ ok: true, match: updated }, { status: 200 });
} catch (err) {
  if (err instanceof Error && err.message === "MATCH_CONFLICT") {
    return NextResponse.json({ ok: false, error: "MATCH_CONFLICT", code: "VERSION_CONFLICT" }, { status: 409 });
  }
  if (err instanceof Error && err.message === "INVALID_SCORE") {
    return NextResponse.json({ ok: false, error: "INVALID_SCORE" }, { status: 400 });
  }
  if (err instanceof Error && err.message === "MATCH_LOCKED") {
    return NextResponse.json({ ok: false, error: "MATCH_LOCKED" }, { status: 409 });
  }
  if (err instanceof Error && err.message === "MISSING_VERSION") {
    return NextResponse.json({ ok: false, error: "MISSING_VERSION" }, { status: 400 });
  }
  if (err instanceof Error && err.message === "MATCH_NOT_FOUND") {
    return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
  }
  console.error("[match_result] error", err);
  return NextResponse.json({ ok: false, error: "UPDATE_FAILED" }, { status: 500 });
}
}

```

app/api/organizador/tournaments/[id]/matches/schedule/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { canReschedule, canNotify } from "@/domain/tournaments/schedulePolicy";

async function ensureOrganizerAccess(userId: string, eventId: number) {
  const evt = await prisma.event.findUnique({ where: { id: eventId }, select: { organizerId: true } });
  if (!evt?.organizerId) return false;
  const member = await prisma.organizerMember.findFirst({
    where: { organizerId: evt.organizerId, userId, role: { in: ["OWNER", "CO_OWNER", "ADMIN"] } },
    select: { id: true },
  });
  return Boolean(member);
}

type ScheduleItem = { matchId: number; courtId?: number | null; startAt?: string | null };

export async function POST(req: NextRequest, { params }: { params: { id: string } }) {
  const tournamentId = Number(params?.id);
  if (!Number.isFinite(tournamentId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = await req.json().catch(() => ({}));
  const items: ScheduleItem[] = Array.isArray(body?.items) ? body.items : [];
  if (!items.length) return NextResponse.json({ ok: false, error: "EMPTY_PAYLOAD" }, { status: 400 });

  // Confirm organizer access using first match -> stage -> tournament -> event
  const firstMatch = await prisma.tournamentMatch.findUnique({
    where: { id: items[0]?.matchId ?? -1 },
    include: { stage: { select: { tournamentId: true, tournament: { select: { eventId: true } } } } },
  });
  if (!firstMatch || firstMatch.stage.tournamentId !== tournamentId) {
    return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
  }

  const authorized = await ensureOrganizerAccess(data.user.id, firstMatch.stage.tournament.eventId);
  if (!authorized) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

  const changes: Array<{ matchId: number; before: any; after: any }> = [];

  let updatedIds: number[] = [];
  try {

```

```

updatedIds = await prisma.$transaction(async (tx) => {
  const results: number[] = [];
  for (const entry of items) {
    const m = await tx.tournamentMatch.findUnique({
      where: { id: entry.matchId },
      include: { stage: { select: { tournamentId: true } } },
    });
    if (!m || m.stage.tournamentId !== tournamentId) continue;

    const newStart = entry.startAt ? new Date(entry.startAt) : null;
    const newCourt = entry.courtId ?? null;

    const canEdit = canReschedule(m.status, m.startAt, newStart);
    if (!canEdit) {
      throw new Error("START_AT_IN_PAST_OR_LOCKED");
    }

    const noChange =
      (m.startAt ?? null)?.getTime() === (newStart ?? null)?.getTime() &&
      (m.courtId ?? null) === (newCourt ?? null);
    if (noChange) continue;

    const before = { startAt: m.startAt, courtId: m.courtId };
    const after = { startAt: newStart, courtId: newCourt };

    await tx.tournamentMatch.update({
      where: { id: m.id },
      data: { startAt: newStart, courtId: newCourt },
    });
    await tx.tournamentAuditLog.create({
      data: {
        tournamentId,
        userId: data.user.id,
        action: "UPDATE_SCHEDULE",
        payloadBefore: before,
        payloadAfter: after,
      },
    });
    changes.push({ matchId: m.id, before, after });
    results.push(m.id);
  }
  return results;
});

} catch (err) {
  if (err instanceof Error && err.message === "START_AT_IN_PAST_OR_LOCKED") {
    return NextResponse.json({ ok: false, error: "START_AT_IN_PAST_OR_LOCKED" }, { status: 400 });
  }
  throw err;
}

return NextResponse.json({ ok: true, updated: updatedIds.length, changes }, { status: 200 });
}

```

app/api/organizador/tournaments/[id]/notify-status/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@lib/supabaseServer";
import { getActiveOrganizerForUser } from "@lib/organizerContext";
import { prisma } from "@lib/prisma";
import { queueEliminated, queueChampion } from "@domain/notifications/tournament";

/** 
 * Endpoint manual para disparar notificações de eliminado/campeão.
 * Útil enquanto não há integração automática com standings finais.
 */
export async function POST(req: NextRequest, { params }: { params: { id: string } }) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const tournamentId = Number(params?.id);
  if (!Number.isFinite(tournamentId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

```

```

const body = (await req.json().catch(() => null)) as {
  eliminatedUserIds?: string[];
  championUserIds?: string[];
} | null;
const eliminatedUserIds = Array.isArray(body?.eliminatedUserIds) ? body?.eliminatedUserIds.filter(Boolean) as string[] : [];
const championUserIds = Array.isArray(body?.championUserIds) ? body?.championUserIds.filter(Boolean) as string[] : [];

const tournament = await prisma.tournament.findUnique({
  where: { id: tournamentId },
  select: { organizerId: true, eventId: true },
});
if (!tournament?.organizerId) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

const { organizer } = await getActiveOrganizerForUser(user.id, {
  organizerId: tournament.organizerId,
  roles: ["OWNER", "CO_OWNER", "ADMIN"],
});
if (!organizer) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

if (eliminatedUserIds.length) {
  await queueEliminated(eliminatedUserIds, tournamentId);
}
if (championUserIds.length) {
  await queueChampion(championUserIds, tournamentId);
}

return NextResponse.json(
  { ok: true, eliminated: eliminatedUserIds.length, champions: championUserIds.length },
  { status: 200 },
);
}
}

```

app/api/organizador/tournaments/[id]/structure/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { getTournamentStructure, summarizeMatchStatus, computeStandingsForGroup } from "@/domain/tournaments/structure";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";

async function ensureOrganizerAccess(userId: string, eventId: number) {
  const evt = await prisma.event.findUnique({
    where: { id: eventId },
    select: { organizerId: true },
  });
  if (!evt?.organizerId) return false;
  const member = await prisma.organizerMember.findFirst({
    where: {
      organizerId: evt.organizerId,
      userId,
      role: { in: ["OWNER", "CO_OWNER", "ADMIN"] },
    },
    select: { id: true },
  });
  return Boolean(member);
}

// TODO: adicionar auth de organizador/owner; neste momento retorna sempre que existe.
export async function GET(_req: NextRequest, { params }: { params: { id: string } }) {
  const id = Number(params.id);
  if (!Number.isFinite(id)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const tournament = await getTournamentStructure(id);
  if (!tournament) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

  const authorized = await ensureOrganizerAccess(data.user.id, tournament.event.id);
  if (!authorized) return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });

  const tieBreakRules = Array.isArray(tournament.tieBreakRules)
    ? (tournament.tieBreakRules as string[])
    : ["WINS", "SET_DIFF", "GAME_DIFF", "HEAD_TO_HEAD", "RANDOM"];

```

```

const payload = {
  id: tournament.id,
  event: tournament.event,
  format: tournament.format,
  stages: tournament.stages.map((s) => ({
    id: s.id,
    name: s.name,
    stageType: s.stageType,
    groups: s.groups.map((g) => ({
      id: g.id,
      name: g.name,
      standings: computeStandingsForGroup(g.matches, tieBreakRules, tournament.generationSeed || undefined),
      matches: g.matches.map((m) => ({
        id: m.id,
        pairing1Id: m.pairing1Id,
        pairing2Id: m.pairing2Id,
        round: m.round,
        roundLabel: m.roundLabel,
        startAt: m.startAt,
        courtId: m.courtId,
        status: m.status,
        statusLabel: summarizeMatchStatus(m.status),
        score: m.score,
        nextMatchId: m.nextMatchId,
        nextSlot: m.nextSlot,
      })),
    })),
    matches: s.matches
      .filter((m) => !m.groupId)
      .map((m) => ({
        id: m.id,
        pairing1Id: m.pairing1Id,
        pairing2Id: m.pairing2Id,
        round: m.round,
        roundLabel: m.roundLabel,
        startAt: m.startAt,
        courtId: m.courtId,
        status: m.status,
        statusLabel: summarizeMatchStatus(m.status),
        score: m.score,
        nextMatchId: m.nextMatchId,
        nextSlot: m.nextSlot,
      })),
  })),
};

const res = NextResponse.json({ ok: true, tournament: payload }, { status: 200 });
res.headers.set("Cache-Control", "no-store");
return res;
}

```

app/api/organizador/username/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";

function normalizeUsername(raw: string) {
  return raw.trim().toLowerCase();
}

export async function PATCH(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
    } = await supabase.auth.getUser();

    if (!user) {
      return NextResponse.json({ ok: false, error: "Não autenticado." }, { status: 401 });
    }

    const body = await req.json().catch(() => null);
  }
}

```

```

const usernameRaw = typeof body?.username === "string" ? body.username : "";
const username = normalizeUsername(usernameRaw);

if (!username || username.length < 3) {
  return NextResponse.json(
    { ok: false, error: "Escolhe um username com pelo menos 3 caracteres." },
    { status: 400 },
  );
}

const pattern = /^[a-z0-9_-]+$/;
if (!pattern.test(username)) {
  return NextResponse.json(
    { ok: false, error: "Usa apenas letras minúsculas, números, - ou _." },
    { status: 400 },
  );
}

const { organizer } = await getActiveOrganizerForUser(user.id, { roles: ["OWNER", "CO_OWNER", "ADMIN"] });
if (!organizer) {
  return NextResponse.json({ ok: false, error: "Organizador não encontrado." }, { status: 403 });
}

const existingOrganizer = await prisma.organizer.findFirst({
  where: { username: { equals: username, mode: "insensitive" }, NOT: { id: organizer.id } },
  select: { id: true },
});
if (existingOrganizer) {
  return NextResponse.json({ ok: false, error: "Este username já está a ser usado." }, { status: 409 });
}

const existingProfile = await prisma.profile.findFirst({
  where: { username: { equals: username, mode: "insensitive" } },
  select: { id: true },
});
if (existingProfile) {
  return NextResponse.json({ ok: false, error: "Este username já está a ser usado." }, { status: 409 });
}

await prisma.organizer.update({
  where: { id: organizer.id },
  data: { username },
});

return NextResponse.json({ ok: true, username }, { status: 200 });
} catch (err) {
  console.error(`[organizador/username][PATCH]`, err);
  const isUnique = err instanceof Error && err.message.toLowerCase().includes("unique");
  const message = isUnique ? "Este username já está a ser usado." : "Erro ao atualizar username.";
  return NextResponse.json({ ok: false, error: message }, { status: isUnique ? 409 : 500 });
}
}

```

app/api/organizador/venues/recent/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { isOrgAdminOrAbove } from "@/lib/organizerPermissions";

export async function GET(req: NextRequest) {
  try {
    const url = new URL(req.url);
    const q = (url.searchParams.get("q") ?? "").trim();

    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    const profile = await prisma.profile.findUnique({
      where: { id: user.id },
    });

    if (!profile) {
      return NextResponse.json({ ok: false, error: "Perfil não encontrado." }, { status: 401 });
    }
  }
}

```

```

const { organizer, membership } = await getActiveOrganizerForUser(profile.id, {
  roles: ["OWNER", "CO_OWNER", "ADMIN"],
});

if (!organizer || !membership || !isOrgAdminOrAbove(membership.role)) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}

const venues = await prisma.event.findMany({
  where: {
    organizerId: organizer.id,
    isDeleted: false,
    AND: [{ locationName: { not: null } }, { locationName: { not: "" } }],
    ...{q
      ? {
        locationName: {
          contains: q,
          mode: "insensitive",
        },
      }
      : {}},
  },
  select: { locationName: true, locationCity: true, updatedAt: true },
  orderBy: { updatedAt: "desc" },
  take: 20,
});

const unique = new Map<string, { name: string; city?: string | null }>();
venues.forEach((row) => {
  if (!row.locationName) return;
  const key = `${row.locationName.toLowerCase()}__${(row.locationCity || "").toLowerCase()}`;
  if (!unique.has(key)) {
    unique.set(key, { name: row.locationName, city: row.locationCity });
  }
});

return NextResponse.json({ ok: true, items: Array.from(unique.values()) });
} catch (err) {
  console.error("GET /api/organizador/venues/recent error:", err);
  return NextResponse.json({ ok: false, error: "Erro ao carregar locais recentes." }, { status: 500 });
}
}

```

app/api/ownership/claim/route.ts

```

import { NextResponse } from "next/server";
import { claimIdentity } from "@/lib/ownership/claimIdentity";
import { createSupabaseServer } from "@/lib/supabaseServer";

export async function POST() {
  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data?.user) {
    return NextResponse.json(
      { ok: false, error: "AUTH_REQUIRED" },
      { status: 401 },
    );
  }

  const email = data.user.email;
  const userId = data.user.id;
  if (!email) {
    return NextResponse.json(
      { ok: false, error: "EMAIL_MISSING" },
      { status: 400 },
    );
  }

  const verified = Boolean((data.user as any)?.email_confirmed_at || (data.user as any)?.emailConfirmedAt);
  if (!verified) {
    return NextResponse.json(
      { ok: false, error: "EMAIL_NOT_VERIFIED" },
      { status: 403 },
    );
  }
}

```

```

    await claimIdentity(email, userId, { requireVerified: true });

    return NextResponse.json({ ok: true });
}

```

app/api/padel/categories/my/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    const { organizer } = await getActiveOrganizerForUser(user.id);
    if (!organizer) {
      return NextResponse.json({ ok: false, error: "Organizador não encontrado." }, { status: 403 });
    }

    const categories = await prisma.padelCategory.findMany({
      where: { organizerId: organizer.id, isActive: true },
      orderBy: [{ season: "desc" }, { year: "desc" }, { createdAt: "desc" }],
      select: { id: true, name: true, level: true },
    });

    return NextResponse.json({ ok: true, items: categories });
  } catch (err) {
    console.error("[padel/categories/my] error", err);
    return NextResponse.json({ ok: false, error: "Erro ao carregar categorias." }, { status: 500 });
  }
}

```

app/api/padel/clubs/[id]/courts/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

export async function GET(req: NextRequest, { params }: { params: Promise<{ id: string }> }) {
  const { id } = await params;
  const clubId = Number(id);
  if (!Number.isFinite(clubId)) return NextResponse.json({ ok: false, error: "INVALID_CLUB" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data: { user } } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const { organizer } = await getActiveOrganizerForUser(user.id, { roles: allowedRoles });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const club = await prisma.padelClub.findFirst({ where: { id: clubId, organizerId: organizer.id, deletedAt: null } });
  if (!club) return NextResponse.json({ ok: false, error: "CLUB_NOT_FOUND" }, { status: 404 });

  const courts = await prisma.padelClubCourt.findMany({
    where: { padelClubId: club.id },
    orderBy: [{ displayOrder: "asc" }, { id: "asc" }],
  });

  return NextResponse.json({ ok: true, items: courts }, { status: 200 });
}

export async function POST(req: NextRequest, { params }: { params: Promise<{ id: string }> }) {
  const { id } = await params;

```

```

const clubId = Number(id);
if (!Number.isFinite(clubId)) return NextResponse.json({ ok: false, error: "INVALID_CLUB" }, { status: 400 });

const supabase = await createSupabaseServer();
const { data: { user } } = await supabase.auth.getUser();
if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });

const { organizer } = await getActiveOrganizerForUser(user.id, { roles: allowedRoles });
if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

const club = await prisma.padelClub.findFirst({ where: { id: clubId, organizerId: organizer.id, deletedAt: null } });
if (!club) return NextResponse.json({ ok: false, error: "CLUB_NOT_FOUND" }, { status: 404 });

const courtId = typeof body.id === "number" ? body.id : null;
const name = typeof body.name === "string" ? body.name.trim() : "";
const description = typeof body.description === "string" ? body.description.trim() : "";
const surface = typeof body.surface === "string" ? body.surface.trim() : "";
const indoor = typeof body.indoor === "boolean" ? body.indoor : false;
const isActive = typeof body.isActive === "boolean" ? body.isActive : true;
const displayOrderRaw =
  typeof body.displayOrder === "number"
    ? body.displayOrder
    : typeof body.displayOrder === "string"
      ? Number(body.displayOrder)
      : null;
const displayOrder = Number.isFinite(displayOrderRaw)
  ? Math.min(10000, Math.max(0, Math.floor(displayOrderRaw as number)))
  : 0;

try {
  let finalName = name;
  if (!finalName) {
    // Gera nome sequencial quando vazio (Court 1, Court 2, ...)
    const count = await prisma.padelClubCourt.count({ where: { padelClubId: club.id } });
    finalName = `Court ${count + 1}`;
  }
}

const data = {
  padelClubId: club.id,
  name: finalName,
  description: description || null,
  surface: surface || null,
  indoor,
  isActive,
  displayOrder,
};

const court = courtId
  ? await prisma.padelClubCourt.update({
    where: { id: courtId, padelClubId: club.id },
    data,
  })
  : await prisma.padelClubCourt.create({ data });

return NextResponse.json({ ok: true, court }, { status: courtId ? 200 : 201 });
} catch (err) {
  console.error("[padel/clubs/courts] error", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/padel/clubs/[id]/staff/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

```

```

export async function GET(req: NextRequest, { params }: { params: Promise<{ id: string }> }) {
  const { id } = await params;
  const clubId = Number(id);
  if (!Number.isFinite(clubId)) return NextResponse.json({ ok: false, error: "INVALID_CLUB" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data: { user } } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const { organizer } = await getActiveOrganizerForUser(user.id, { roles: allowedRoles });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const club = await prisma.padelClub.findFirst({ where: { id: clubId, organizerId: organizer.id, deletedAt: null } });
  if (!club) return NextResponse.json({ ok: false, error: "CLUB_NOT_FOUND" }, { status: 404 });

  const staff = await prisma.padelClubStaff.findMany({
    where: { padelClubId: club.id, deletedAt: null },
    orderBy: [{ createdAt: "desc" }],
  });

  return NextResponse.json({ ok: true, items: staff }, { status: 200 });
}

export async function POST(req: NextRequest, { params }: { params: Promise<{ id: string }> }) {
  const { id } = await params;
  const clubId = Number(id);
  if (!Number.isFinite(clubId)) return NextResponse.json({ ok: false, error: "INVALID_CLUB" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data: { user } } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
  if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });

  const { organizer } = await getActiveOrganizerForUser(user.id, { roles: allowedRoles });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const club = await prisma.padelClub.findFirst({ where: { id: clubId, organizerId: organizer.id, deletedAt: null } });
  if (!club) return NextResponse.json({ ok: false, error: "CLUB_NOT_FOUND" }, { status: 404 });

  const staffId = typeof body.id === "number" ? body.id : null;
  const email = typeof body.email === "string" ? body.email.trim().toLowerCase() : "";
  const userId = typeof body.userId === "string" ? body.userId : null;
  const role = typeof body.role === "string" ? body.role.trim() : "";
  const inheritToEvents = typeof body.inheritToEvents === "boolean" ? body.inheritToEvents : true;

  if (!email && !userId) {
    return NextResponse.json({ ok: false, error: "Indica o email ou userId do staff." }, { status: 400 });
  }
  if (!role) {
    return NextResponse.json({ ok: false, error: "Define um papel para este membro." }, { status: 400 });
  }

  try {
    const data = {
      padelClubId: club.id,
      email: email || null,
      userId,
      role,
      inheritToEvents,
    };
    const staff = staffId
      ? await prisma.padelClubStaff.update({
          where: { id: staffId, padelClubId: club.id },
          data: { ...data, deletedAt: null, isActive: true },
        })
      : await prisma.padelClubStaff.create({ data });

    return NextResponse.json({ ok: true, staff }, { status: staffId ? 200 : 201 });
  } catch (err) {
    console.error("[padel/clubs/staff] error", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
  }
}

// Soft delete staff member

```

```

export async function DELETE(req: NextRequest, { params }: { params: Promise<{ id: string }> }) {
  const { id } = await params;
  const clubId = Number(id);
  if (!Number.isFinite(clubId)) return NextResponse.json({ ok: false, error: "INVALID_CLUB" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data: { user } } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const { organizer } = await getActiveOrganizerForUser(user.id, { roles: allowedRoles });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const club = await prisma.padelClub.findFirst({ where: { id: clubId, organizerId: organizer.id, deletedAt: null } });
  if (!club) return NextResponse.json({ ok: false, error: "CLUB_NOT_FOUND" }, { status: 404 });

  const url = new URL(req.url);
  const staffId = url.searchParams.get("staffId");
  const staffIdNum = staffId ? Number(staffId) : NaN;
  if (!Number.isFinite(staffIdNum)) return NextResponse.json({ ok: false, error: "INVALID_STAFF" }, { status: 400 });

  const staff = await prisma.padelClubStaff.findFirst({
    where: { id: staffIdNum, padelClubId: clubId, deletedAt: null },
  });
  if (!staff) return NextResponse.json({ ok: false, error: "STAFF_NOT_FOUND" }, { status: 404 });

  await prisma.padelClubStaff.update({
    where: { id: staffIdNum },
    data: { isActive: false, deletedAt: new Date() },
  });

  return NextResponse.json({ ok: true }, { status: 200 });
}

```

app/api/padel/clubs/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { PORTUGAL_CITIES } from "@/config/cities";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

function normalizeSlug(raw: string | null | undefined) {
  if (!raw) return "";
  return raw
    .trim()
    .toLowerCase()
    .replace(/[^a-z0-9-_]+/g, "-")
    .replace(/^-+/, "-")
    .replace(/^-+|-$/, "");
}

async function generateUniqueSlug(base: string, organizerId: number, excludeId?: number | null) {
  if (!base) return "";
  let candidate = base;
  let suffix = 2;
  // Garante slug único por organizador; acrescenta -2, -3, ...
  // Usa findFirst case-insensitive para evitar conflitos.
  while (true) {
    const exists = await prisma.padelClub.findFirst({
      where: {
        organizerId,
        slug: { equals: candidate, mode: "insensitive" },
        ...(!excludeId ? { NOT: { id: excludeId } } : {}),
      },
      select: { id: true },
    });
    if (!exists) return candidate;
    candidate = `${base}-${suffix}`;
    suffix += 1;
  }
}

```

```

export async function GET(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const organizerIdParam = req.nextUrl.searchParams.get("organizerId");
  const parsedOrgId = organizerIdParam ? Number(organizerIdParam) : null;
  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: Number.isFinite(parsedOrgId) ? parsedOrgId : undefined,
    roles: allowedRoles,
  });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const items = await prisma.padelClub.findMany({
    where: { organizerId: organizer.id, deletedAt: null },
    orderBy: [{ isActive: "desc" }, { createdAt: "desc" }],
  });

  return NextResponse.json({ ok: true, items }, { status: 200 });
}

export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
  if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });

  const organizerIdParam = body.organizerId ?? req.nextUrl.searchParams.get("organizerId");
  const parsedOrgId =
    typeof organizerIdParam === "number" ? organizerIdParam : organizerIdParam ? Number(organizerIdParam) : null;
  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: Number.isFinite(parsedOrgId) ? parsedOrgId : undefined,
    roles: allowedRoles,
  });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const id = typeof body.id === "number" ? body.id : null;
  const name = typeof body.name === "string" ? body.name.trim() : "";
  const city = typeof body.city === "string" ? body.city.trim() : "";
  const address = typeof body.address === "string" ? body.address.trim() : "";
  const courtsCountRaw =
    typeof body.courtsCount === "number"
      ? body.courtsCount
      : typeof body.courtsCount === "string"
        ? Number(body.courtsCount)
        : null;
  const isActive = typeof body.isActive === "boolean" ? body.isActive : true;
  const slugInput = typeof body.slug === "string" ? normalizeSlug(body.slug) : "";
  const isDefault = typeof body.isDefault === "boolean" ? body.isDefault : false;

  if (!name || name.length < 3) {
    return NextResponse.json({ ok: false, error: "Nome do clube é obrigatório." }, { status: 400 });
  }

  if (city && !PORTUGAL_CITIES.includes(city as (typeof PORTUGAL_CITIES)[number])) {
    return NextResponse.json(
      { ok: false, error: "Cidade inválida. Escolhe uma cidade da lista disponível na ORYA." },
      { status: 400 },
    );
  }

  const courtsCount = courtsCountRaw && Number.isFinite(courtsCountRaw)
    ? Math.min(1000, Math.max(1, Math.floor(courtsCountRaw)))
    : 1;
  const baseSlug = slugInput || normalizeSlug(name);

  try {
    const slug = baseSlug ? await generateUniqueSlug(baseSlug, organizer.id, id) : null;
  }
}

```

```

const data = {
  organizerId: organizer.id,
  name,
  shortName: name,
  city: city || null,
  address: address || null,
  courtsCount,
  hours: null,
  favoriteCategoryIds: [],
  isActive,
  slug: slug || null,
  isDefault,
};

const club = await prisma.$transaction(async (tx) => {
  let saved = id
  ? await tx.padelClub.update({
    where: { id, organizerId: organizer.id, deletedAt: null },
    data,
  })
  : await tx.padelClub.create({
    data,
  });

  if (isDefault) {
    await tx.padelClub.updateMany({
      where: { organizerId: organizer.id, NOT: { id: saved.id }, isDefault: true },
      data: { isDefault: false },
    });
  } else {
    // Se não existir nenhum default, garante que o primeiro ativo fica default
    const defaults = await tx.padelClub.count({ where: { organizerId: organizer.id, isDefault: true } });
    if (defaults === 0 && saved.isActive) {
      saved = await tx.padelClub.update({ where: { id: saved.id }, data: { isDefault: true } });
    }
  }
  return saved;
});

return NextResponse.json({ ok: true, club }, { status: id ? 200 : 201 });
} catch (err) {
  console.error("[padel/clubs] error", err);
  const code = (err as { code?: string })?.code;
  if (code === "P2002") {
    return NextResponse.json(
      { ok: false, error: "Já existe um clube com este slug/nome. Escolhe outro." },
      { status: 409 },
    );
  }
  const msg =
    err instanceof Error && err.message.includes("Record to update not found")
    ? "Clube não encontrado."
    : "Erro ao gravar clube.";
  const status = msg === "Clube não encontrado." ? 404 : 500;
  return NextResponse.json({ ok: false, error: msg }, { status });
}
}

// Soft delete club (marks isActive=false, deletedAt now)
export async function DELETE(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const url = new URL(req.url);
  const idParam = url.searchParams.get("id");
  const organizerIdParam = url.searchParams.get("organizerId");
  const clubId = idParam ? Number(idParam) : NaN;
  const orgId = organizerIdParam ? Number(organizerIdParam) : NaN;

  if (!Number.isFinite(clubId)) return NextResponse.json({ ok: false, error: "INVALID_CLUB" }, { status: 400 });

  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: Number.isFinite(orgId) ? orgId : undefined,
    roles: allowedRoles,
  });
}

```

```

if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

const club = await prisma.padelClub.findFirst({
  where: { id: clubId, organizerId: organizer.id, deletedAt: null },
});
if (!club) return NextResponse.json({ ok: false, error: "CLUB_NOT_FOUND" }, { status: 404 });

await prisma.$transaction(async (tx) => {
  await tx.padelClub.update({
    where: { id: clubId },
    data: { isActive: false, deletedAt: new Date() },
  });
  await tx.padelClubStaff.updateMany({
    where: { padelClubId: clubId },
    data: { isActive: false, deletedAt: new Date() },
  });
});

return NextResponse.json({ ok: true }, { status: 200 });
}

```

app/api/padel/matches/[id]/walkover/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { PadelMatchStatus } from "@prisma/client";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { canMarkWalkover } from "@/domain/padel/pairingPolicy";

export async function POST(req: NextRequest, { params }: { params: { id: string } }) {
  const matchId = Number(params?.id);
  if (!Number.isFinite(matchId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data: authData, error: authError } = await supabase.auth.getUser();
  if (authError || !authData?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const match = await prisma.padelMatch.findUnique({
    where: { id: matchId },
    select: { id: true, pairingAId: true, pairingBId: true, eventId: true, status: true },
  });
  if (!match) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
  if (match.status === PadelMatchStatus.DONE)
    return NextResponse.json({ ok: false, error: "ALREADY_DONE" }, { status: 409 });
}

const body = await req.json().catch(() => ({}));
const winner = body?.winner as "A" | "B";
if (winner !== "A" && winner !== "B") {
  return NextResponse.json({ ok: false, error: "INVALID_WINNER" }, { status: 400 });
}

const winnerPairingId = winner === "A" ? match.pairingAId : match.pairingBId;
if (!winnerPairingId) {
  return NextResponse.json({ ok: false, error: "MISSING_PAIRINGS" }, { status: 400 });
}

// No RBAC profundo aqui; ideal seria OWNER/ADMIN

const updated = await prisma.$transaction(async (tx) => {
  const updatedMatch = await tx.padelMatch.update({
    where: { id: matchId },
    data: { status: PadelMatchStatus.DONE, winnerPairingId, score: { walkover: true } },
  });
  return updatedMatch;
});

return NextResponse.json({ ok: true, match: updated }, { status: 200 });
}

```

app/api/padel/matches/generate/route.ts

```

export const runtime = "nodejs";

```

```

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole, PadelFormat } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { queueBracketPublished } from "@/domain/notifications/tournament";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

function roundRobinPairs(teamIds: number[]) {
  const matches: Array<{ a: number; b: number }> = [];
  for (let i = 0; i < teamIds.length; i += 1) {
    for (let j = i + 1; j < teamIds.length; j += 1) {
      matches.push({ a: teamIds[i], b: teamIds[j] });
    }
  }
  return matches;
}

function eliminationPairs(teamIds: number[]) {
  const matches: Array<{ a: number; b: number }> = [];
  const ids = [...teamIds];
  for (let i = 0; i < ids.length; i += 2) {
    if (i + 1 < ids.length) {
      matches.push({ a: ids[i], b: ids[i + 1] });
    }
  }
  return matches;
}

export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
  if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });

  const eventId = typeof body.eventId === "number" ? body.eventId : Number(body.eventId);
  const formatRaw = typeof body.format === "string" ? (body.format as PadelFormat) : "TODOS CONTRA TODOS";
  const format: PadelFormat =
    formatRaw === "QUADRO_ELIMINATORIO" ? "QUADRO_ELIMINATORIO" : "TODOS CONTRA TODOS";

  if (!Number.isFinite(eventId)) return NextResponse.json({ ok: false, error: "INVALID_EVENT" }, { status: 400 });

  const event = await prisma.event.findUnique({
    where: { id: eventId, isDeleted: false },
    select: { id: true, organizerId: true },
  });
  if (!event || !event.organizerId) return NextResponse.json({ ok: false, error: "EVENT_NOT_FOUND" }, { status: 404 });

  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: event.organizerId,
    roles: allowedRoles,
  });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const config = await prisma.padelTournamentConfig.findUnique({
    where: { eventId },
    select: { numberOfCourts: true, advancedSettings: true },
  });
  const advanced = (config?.advancedSettings || {}) as {
    courtsFromClubs?: Array<{ name?: string | null; clubName?: string | null; displayOrder?: number | null }>;
    staffFromClubs?: Array<{ email?: string | null; userId?: string | null; role?: string | null }>;
  };
  const courtsList =
    Array.isArray(advanced.courtsFromClubs) && advanced.courtsFromClubs.length > 0
      ? [...advanced.courtsFromClubs].sort((a, b) => (a.displayOrder ?? 0) - (b.displayOrder ?? 0))
      : Array.from({ length: Math.max(1, config?.numberOfCourts || 1) }).map((_, idx) => ({
        name: `Court ${idx + 1}`,
        clubName: null,
        displayOrder: idx,
      }));
  const staffList = Array.isArray(advanced.staffFromClubs) ? advanced.staffFromClubs : [];
}

```

```

const pairings = await prisma.padelPairing.findMany({
  where: {
    eventId,
    pairingStatus: "COMPLETE",
  },
  select: { id: true, slots: { select: { profileId: true } } },
  orderBy: { createdAt: "asc" },
});
const pairingIds = pairings.map((p) => p.id);
const userIds = Array.from(
  new Set(
    pairings
      .flatMap((p) => p.slots)
      .map((s) => s.profileId)
      .filter(Boolean) as string[],
  ),
);
if (pairingIds.length < 2) {
  return NextResponse.json({ ok: false, error: "NEED_PAIRINGS" }, { status: 400 });
}

const pairs = format === "QUADRO_ELIMINATORIO" ? eliminationPairs(pairingIds) : roundRobinPairs(pairingIds);

await prisma.$transaction(async (tx) => {
  await tx.padelMatch.deleteMany({ where: { eventId } });
  await tx.padelMatch.createMany({
    data: pairs.map((p, idx) => {
      const court = courtsList[idx % courtsList.length];
      const staff = staffList.length > 0 ? staffList[idx % staffList.length] : null;
      const staffLabel = staff ? staff.email || staff.userId || staff.role || "Staff" : null;
      return {
        eventId,
        pairingAId: p.a,
        pairingBId: p.b,
        status: "PENDING",
        courtNumber: (idx % courtsList.length) + 1,
        courtName: court?.name || null,
        roundLabel: staffLabel ? `Staff: ${staffLabel}` : null,
      };
    }),
  });
});

const matches = await prisma.padelMatch.findMany({
  where: { eventId },
  orderBy: [{ startTime: "asc" }, { id: "asc" }],
});

if (userIds.length) {
  await queueBracketPublished(userIds, eventId);
}

return NextResponse.json({ ok: true, matches }, { status: 200 });
}

```

app/api/padel/matches/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole, PadelMatchStatus } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { isValidScore } from "@/lib/padel/validation";
import {
  queueMatchChanged,
  queueMatchResult,
  queueNextOpponent,
} from "@/domain/notifications/tournament";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

export async function GET(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {

```

```

    data: { user },
} = await supabase.auth.getUser();

if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

const eventId = Number(req.nextUrl.searchParams.get("eventId"));
if (!Number.isFinite(eventId)) return NextResponse.json({ ok: false, error: "INVALID_EVENT" }, { status: 400 });

const matches = await prisma.padelMatch.findMany({
  where: { eventId },
  include: {
    pairingA: { include: { slots: { include: { playerProfile: true } } } },
    pairingB: { include: { slots: { include: { playerProfile: true } } } },
  },
  orderBy: [{ startTime: "asc" }, { id: "asc" }],
});

return NextResponse.json({ ok: true, items: matches }, { status: 200 });
}

export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
  if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });

  const matchId = typeof body.id === "number" ? body.id : Number(body.id);
  const statusRaw = typeof body.status === "string" ? (body.status as PadelMatchStatus) : undefined;
  const scoreRaw = body.score;
  const startAtRaw = body.startAt ? new Date(body.startAt as string) : undefined;
  const courtIdRaw = typeof body.courtId === "number" ? body.courtId : undefined;

  if (!Number.isFinite(matchId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const match = await prisma.padelMatch.findUnique({
    where: { id: matchId },
    include: { event: { select: { organizerId: true } } },
  });
  if (!match || !match.event?.organizerId) return NextResponse.json({ ok: false, error: "MATCH_NOT_FOUND" }, { status: 404 });

  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: match.event.organizerId,
    roles: allowedRoles,
  });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  if (scoreRaw && !isValidScore(scoreRaw)) {
    return NextResponse.json({ ok: false, error: "INVALID_SCORE" }, { status: 400 });
  }

  let winnerPairingId: number | null = null;
  if (scoreRaw && typeof scoreRaw === "object" && "sets" in (scoreRaw as { sets?: unknown })) {
    const rawSets = (scoreRaw as { sets?: unknown }).sets;
    const sets = Array.isArray(rawSets) ? rawSets : [];
    let winsA = 0;
    let winsB = 0;
    sets.forEach((s) => {
      const set = s as { teamA?: number; teamB?: number };
      if (Number.isFinite(set.teamA) && Number.isFinite(set.teamB)) {
        const a = Number(set.teamA);
        const b = Number(set.teamB);
        if (a > b) winsA += 1;
        else if (b > a) winsB += 1;
      }
    });
    if (winsA > winsB && match.pairingAId) winnerPairingId = match.pairingAId;
    if (winsB > winsA && match.pairingBId) winnerPairingId = match.pairingBId;
  }

  const updated = await prisma.padelMatch.update({
    where: { id: matchId },
    data: {
      status: statusRaw ?? match.status,
    }
  });
}

```

```

    score: scoreRaw ?? match.score,
    scoreSets: typeof scoreRaw === "object" && scoreRaw && "sets" in (scoreRaw as { sets?: unknown })
      ? (scoreRaw as { sets?: unknown }).sets
      : match.scoreSets,
    winnerPairingId: winnerPairingId ?? match.winnerPairingId,
    startTime: startAtRaw ?? match.startTime,
    courtNumber: courtIdRaw ?? match.courtNumber,
  },
  include: {
    pairingA: { include: { slots: { include: { playerProfile: true } } } },
    pairingB: { include: { slots: { include: { playerProfile: true } } } },
  },
});

const involvedUserIds = [
  ...((updated.pairingA?.slots ?? []).map((s) => s.profileId).filter(Boolean) as string[]),
  ...((updated.pairingB?.slots ?? []).map((s) => s.profileId).filter(Boolean) as string[]),
];

// Notificações: mudança de horário/court
await queueMatchChanged({
  userIds: involvedUserIds,
  matchId: updated.id,
  startAt: updated.startTime ?? null,
  courtId: updated.courtNumber ?? null,
});

// Notificações de resultado + próximo adversário
if (winnerPairingId) {
  await queueMatchResult(involvedUserIds, updated.id, updated.eventId);
  await queueNextOpponent(involvedUserIds, updated.id, updated.eventId);
}

return NextResponse.json({ ok: true, match: updated }, { status: 200 });
}

```

app/api/padel/pairings/[id]/assume/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { PadelPaymentMode } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

// Capitão assume o resto (SPLIT): apenas validação; checkout deve ser iniciado no cliente.
export async function POST(_: NextRequest, { params }: { params: { id: string } }) {
  const pairingId = Number(params?.id);
  if (!Number.isFinite(pairingId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const pairing = await prisma.padelPairing.findUnique({
    where: { id: pairingId },
    include: { slots: true, event: { select: { organizerId: true } } },
  });
  if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
  if (pairing.paymentMode !== PadelPaymentMode.SPLIT) {
    return NextResponse.json({ ok: false, error: "NOT_SPLIT_MODE" }, { status: 400 });
  }
  if (pairing.pairingStatus === "CANCELLED") {
    return NextResponse.json({ ok: false, error: "PAIRING_CANCELLED" }, { status: 400 });
  }

  const isCaptain = pairing.createdById === user.id;
  if (!isCaptain) {
    return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
  }

  // Verifica se ainda há slot pendente não pago
  const pending = pairing.slots.find((s) => s.slotStatus === "PENDING");
  if (!pending) {

```

```

    return NextResponse.json({ ok: false, error: "NO_PENDING_SLOT" }, { status: 400 });

}

// Resposta indica que o cliente deve iniciar checkout do valor remanescente
return NextResponse.json(
{
  ok: false,
  error: "PAYMENT_REQUIRED",
  action: "CHECKOUT_CAPTAIN_REST",
  pairingId,
  slotId: pending.id,
},
{ status: 402 },
);
}

```

app/api/padel/pairings/[id]/cancel/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { PadelPairingStatus, PadelPairingSlotStatus } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { cancelActiveHold } from "@/domain/padelPairingHold";

// Cancela pairing Padel v2 (MVP: estados DB; refund efetivo fica para o checkout/refund handler).
// Regras: capitão (created_by_user_id) ou staff OWNER/ADMIN do organizer.
export async function POST(req: NextRequest, { params }: { params: { id: string } }) {
  const pairingId = Number(params?.id);
  if (!Number.isFinite(pairingId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const pairing = await prisma.padelPairing.findUnique({
    where: { id: pairingId },
    include: {
      event: { select: { organizerId: true } },
      slots: true,
    },
  });
  if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

  if (pairing.pairingStatus === PadelPairingStatus.CANCELLED) {
    return NextResponse.json({ ok: true, pairing }, { status: 200 });
  }

  // Capitão (created_by_user_id) ou staff OWNER/ADMIN
  const isCaptain = pairing.createdById === user.id;
  let isStaff = false;
  if (!isCaptain) {
    const staff = await prisma.organizerMember.findFirst({
      where: {
        organizerId: pairing.organizerId,
        userId: user.id,
        role: { in: ["OWNER", "CO_OWNER", "ADMIN"] },
      },
      select: { id: true },
    });
    isStaff = Boolean(staff);
  }
  if (!isCaptain && !isStaff) {
    return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
  }

  try {
    const updated = await prisma.$transaction(async (tx) => {
      // Marca slots como cancelados
      await tx.padelPairingSlot.updateMany({
        where: { pairingId },
        data: {
          slotStatus: PadelPairingSlotStatus.CANCELLED,
        }
      });
    });
  }
}

```

```

    },
});

// Marca pairing cancelado e remove token para impedir novos claims
const updatedPairing = await tx.padelPairing.update({
  where: { id: pairingId },
  data: {
    pairingStatus: PadelPairingStatus.CANCELLED,
    partnerInviteToken: null,
    partnerInviteUsedAt: null,
    partnerLinkToken: null,
    partnerLinkExpiresAt: null,
    lockedUntil: null,
  },
  include: { slots: true },
});

await cancelActiveHold(tx, pairingId);

return updatedPairing;
});

// Nota: refund efetivo deve ser tratado no fluxo de checkout/refund (Stripe) posterior.
return NextResponse.json({ ok: true, pairing: updated }, { status: 200 });
} catch (err) {
  console.error(`[padel/pairings][cancel][POST]`, err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/padel/pairings/[id]/checkout/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import {
  Gender,
  PadelEligibilityType,
  PadelPaymentMode,
  PadelPairingPaymentStatus,
} from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { validateEligibility } from "@/domain/padelEligibility";

// Apenas valida e delega criação de intent ao endpoint central (/api/payments/intent).
export async function POST(req: NextRequest, { params }: { params: { id: string } }) {
  const pairingId = Number(params.id);
  if (!Number.isFinite(pairingId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json()).catch(() => null) as Record<string, unknown> | null;
  const ticketTypeId = body?.ticketTypeId === "number" ? body.ticketTypeId : null;
  const inviteToken = body?.inviteToken === "string" ? body.inviteToken : null;
  if (!ticketTypeId) return NextResponse.json({ ok: false, error: "MISSING_TICKET_TYPE" }, { status: 400 });

  const pairing = await prisma.padelPairing.findUnique({
    where: { id: pairingId },
    include: {
      slots: true,
      event: { select: { organizerId: true, slug: true, id: true } },
    },
  });
  if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
  if (inviteToken && pairing.partnerInviteToken && pairing.partnerInviteToken !== inviteToken) {
    return NextResponse.json({ ok: false, error: "INVALID_TOKEN" }, { status: 403 });
  }
  if (pairing.pairingStatus === "CANCELLED") {
    return NextResponse.json({ ok: false, error: "PAIRING_CANCELLED" }, { status: 400 });
}

```

```

const pending =
  pairing.paymentMode === PadelPaymentMode.SPLIT
    ? pairing.slots.find((s) => s.slotStatus === "PENDING")
    : null;
if (pairing.paymentMode === PadelPaymentMode.SPLIT) {
  if (!pending) {
    return NextResponse.json({ ok: false, error: "NO_PENDING_SLOT" }, { status: 400 });
  }
  if (pending.paymentStatus === PadelPairingPaymentStatus.PAID) {
    return NextResponse.json({ ok: false, error: "SLOT_ALREADY_PAID" }, { status: 400 });
  }
}
if (pairing.deadlineAt && pairing.deadlineAt.getTime() < Date.now()) {
  return NextResponse.json({ ok: false, error: "PAIRING_EXPIRED" }, { status: 410 });
}

// Apenas capitão pode iniciar checkout se for "assume resto"; parceiro também pode iniciar, mas validamos que não há ticket
// já atribuído
const isCaptain = pairing.createdByUserId === user.id;
const isPendingOwner = !pending?.profileId || pending.profileId === user.id;
if (!isCaptain && !isPendingOwner) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}

const ticketType = await prisma.ticketType.findUnique({
  where: { id: ticketTypeId },
  select: { price: true, currency: true, eventId: true, event: { select: { slug: true } } },
});
if (!ticketType || ticketType.eventId !== pairing.eventId) {
  return NextResponse.json({ ok: false, error: "INVALID_TICKET_TYPE" }, { status: 400 });
}

// Elegibilidade: garantir que capitão + parceiro (quando definido) respeitam regras
const tournamentConfig = await prisma.padelTournamentConfig.findUnique({
  where: { eventId: pairing.event.id },
  select: { eligibilityType: true },
});
const [captainProfile, partnerProfile] = await Promise.all([
  pairing.player1UserId
    ? prisma.profile.findUnique({ where: { id: pairing.player1UserId }, select: { gender: true } })
    : Promise.resolve(null),
  prisma.profile.findUnique({ where: { id: user.id }, select: { gender: true } }),
]);
const eligibility = validateEligibility(
  (tournamentConfig?.eligibilityType as PadelEligibilityType) ?? PadelEligibilityType.OPEN,
  (captainProfile?.gender as Gender | null) ?? null,
  partnerProfile?.gender as Gender | null,
);
if (!eligibility.ok) {
  return NextResponse.json(
    { ok: false, error: eligibility.code },
    { status: eligibility.code === "GENDER_REQUIRED_FOR_TOURNAMENT" ? 403 : 409 },
  );
}

// Não permitir checkout se utilizador já tiver pairing ativo no torneio
const existingActive = await prisma.padelPairing.findFirst({
  where: {
    eventId: pairing.event.id,
    lifecycleStatus: { not: "CANCELLED_INCOMPLETE" },
    OR: [{ player1UserId: user.id }, { player2UserId: user.id }],
    NOT: { id: pairing.id },
  },
  select: { id: true },
});
if (existingActive) {
  return NextResponse.json({ ok: false, error: "PAIRING_ALREADY_ACTIVE" }, { status: 409 });
}

const currency = ticketType.currency || "EUR";
const paymentScenario = pairing.paymentMode === PadelPaymentMode.FULL ? "GROUP_FULL" : "GROUP_SPLIT";
const items = [
  {
    ticketId: ticketTypeId,
    quantity: pairing.paymentMode === PadelPaymentMode.FULL ? 2 : 1,
    unitPriceCents: ticketType.price,
    currency: currency.toUpperCase(),
  },
];

```

```

};

const origin = req.nextUrl.origin || process.env.NEXT_PUBLIC_SITE_URL || "";
try {
  const res = await fetch(`${origin}/api/payments/intent`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      cookie: req.headers.get("cookie") ?? "",
    },
    body: JSON.stringify({
      slug: pairing.event.slug ?? ticketType.event?.slug ?? null,
      items,
      paymentScenario,
      pairingId: pairing.id,
      slotId: pending?.id ?? undefined,
    }),
  });
}

const data = await res.json().catch(() => null);
if (!res.ok || !data?.ok) {
  console.error("[padel/pairings][checkout] intent error", { status: res.status, data });
  return NextResponse.json(
    { ok: false, error: data?.error ?? "INTENT_CREATION_FAILED", code: data?.code ?? null },
    { status: res.status },
  );
}

// Marcar slot como PAYMENT_PENDING para bloquear alterações enquanto paga
if (pending) {
  await prisma.padelPairingSlot.update({
    where: { id: pending.id },
    data: { paymentStatus: PadelPairingPaymentStatus.PAYMENT_PENDING },
  });
}

return NextResponse.json(
  {
    ok: true,
    clientSecret: data.clientSecret,
    paymentIntentId: data.paymentIntentId,
    purchaseId: data.purchaseId,
    paymentScenario,
    breakdown: data.breakdown ?? null,
  },
  { status: 200 },
);
} catch (err) {
  console.error("[padel/pairings][checkout][POST]", err);
  return NextResponse.json({ ok: false, error: "INTENT_ERROR" }, { status: 500 });
}
}

```

app/api/padel/pairings/[id]/invite/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { randomUUID } from "crypto";
import { computePartnerLinkExpiresAt } from "@domain/padelDeadlines";
import { queuePairingInvite } from "@domain/notifications/splitPayments";

// Regenera token de convite para um pairing (v2). Apenas capitão ou staff OWNER/ADMIN.
export async function POST(req: NextRequest, { params }: { params: { id: string } }) {
  const pairingId = Number(params?.id);
  if (!Number.isFinite(pairingId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;

```

```

const expiresMinutesRaw = typeof body?.expiresMinutes === "number" ? body.expiresMinutes : null;
const targetUserId = typeof body?.targetUserId === "string" ? body.targetUserId : null;

const pairing = await prisma.padelPairing.findUnique({
  where: { id: pairingId },
  include: { event: { select: { organizerId: true } } },
});
if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
if (pairing.player2UserId) {
  return NextResponse.json({ ok: false, error: "INVITE_ALREADY_USED" }, { status: 409 });
}
if (!pairing.partnerInviteToken) {
  // se não existe, seguimos para criar mesmo assim
}

const isCaptain = pairing.createdByUserId === user.id;
let isStaff = false;
if (!isCaptain) {
  const staff = await prisma.organizerMember.findFirst({
    where: {
      organizerId: pairing.organizerId,
      userId: user.id,
      role: { in: ["OWNER", "CO_OWNER", "ADMIN"] },
    },
    select: { id: true },
  });
  isStaff = Boolean(staff);
}
if (!isCaptain && !isStaff) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}

const token = randomUUID();
const expiresAt = computePartnerLinkExpiresAt(new Date(), expiresMinutesRaw);

const updated = await prisma.padelPairing.update({
  where: { id: pairingId },
  data: {
    partnerInviteToken: token,
    partnerLinkToken: token,
    partnerLinkExpiresAt: expiresAt,
    partnerInvitedAt: new Date(),
  },
  select: { id: true, partnerInviteToken: true, partnerLinkExpiresAt: true },
});
if (targetUserId) {
  await queuePairingInvite({
    pairingId,
    targetUserId,
    inviterUserId: user.id,
    token,
  });
}

return NextResponse.json({ ok: true, invite: updated }, { status: 200 });
}

```

app/api/padel/pairings/[id]/public/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

// Toggle public/open slot (MVP): capitão ou staff OWNER/ADMIN
export async function PATCH(req: NextRequest, { params }: { params: { id: string } }) {
  const pairingId = Number(params?.id);
  if (!Number.isFinite(pairingId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

```

```

const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
const isPublicOpen = Boolean(body?.isPublicOpen);

const pairing = await prisma.padelPairing.findUnique({
  where: { id: pairingId },
  include: { slots: true, event: { select: { organizerId: true } } },
});
if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

const isCaptain = pairing.createdByUserId === user.id;
let isStaff = false;
if (!isCaptain) {
  const staff = await prisma.organizerMember.findFirst({
    where: {
      organizerId: pairing.organizerId,
      userId: user.id,
      role: { in: ["OWNER", "CO_OWNER", "ADMIN"] },
    },
    select: { id: true },
  });
  isStaff = Boolean(staff);
}
if (!isCaptain && !isStaff) {
  return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
}

const pendingSlot = pairing.slots.find((s) => s.slotStatus === "PENDING");
try {
  const updated = await prisma.padelPairing.update({
    where: { id: pairingId },
    data: {
      isPublicOpen,
      slots: pendingSlot
      ? {
          update: {
            where: { id: pendingSlot.id },
            data: { isPublicOpen },
          },
        }
      : undefined,
    },
    include: { slots: true },
  });
  return NextResponse.json({ ok: true, pairing: updated }, { status: 200 });
} catch (err) {
  console.error("[padel/pairings][public][PATCH]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/padel/pairings/[id]/swap/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { canSwapPartner } from "@/domain/padel/pairingPolicy";

export async function POST(req: NextRequest, { params }: { params: { id: string } }) {
  const pairingId = Number(params?.id);
  if (!Number.isFinite(pairingId)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const supabase = await createSupabaseServer();
  const { data: authData, error: authError } = await supabase.auth.getUser();
  if (authError || !authData?.user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const pairing = await prisma.padelPairing.findUnique({
    where: { id: pairingId },
    select: {
      id: true,
      organizerId: true,
      player1UserId: true,
      player2UserId: true,
      lifecycleStatus: true,
      partnerSwapAllowedUntilAt: true,
    },
  },

```

```

    });

    if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

    // Apenas capitão ou staff do organizer
    const isCaptain = pairing.player1UserId === authData.user.id;
    if (!isCaptain) {
      return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    if (!canSwapPartner(pairing.lifecycleStatus as any, new Date(), pairing.partnerSwapAllowedUntilAt)) {
      return NextResponse.json({ ok: false, error: "SWAP_NOT_ALLOWED" }, { status: 409 });
    }

    // Liberta o parceiro (slot) sem mexer em ticket; fluxos de pagamento devem ser tratados noutra rota
    await prisma.padelPairing.update({
      where: { id: pairing.id },
      data: { player2UserId: null },
    });

    return NextResponse.json({ ok: true }, { status: 200 });
}

```

app/api/padel/pairings/claim/[token]/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import {
  Gender,
  PadelEligibilityType,
  PadelPairingLifecycleStatus,
  PadelPairingPaymentStatus,
  PadelPairingSlotStatus,
  PadelPaymentMode,
} from "@prisma/client";
import { createSupabaseServer } from "@lib/supabaseServer";
import { prisma } from "@lib/prisma";
import { buildPadelEventSnapshot } from "@lib/padel/eventSnapshot";
import { validateEligibility } from "@domain/padelEligibility";
import { PairingAction, transition } from "@domain/padelPairingStateMachine";

async function ensurePlayerProfile(params: { organizerId: number; userId: string }) {
  const { organizerId, userId } = params;
  const existing = await prisma.padelPlayerProfile.findFirst({
    where: { organizerId, userId },
    select: { id: true },
  });
  if (existing) return existing.id;
  const profile = await prisma.profile.findUnique({ where: { id: userId }, select: { fullName: true, email: true } });
  const name = profile?.fullName?.trim() || "Jogador Padel";
  const email = profile?.email || null;
  const created = await prisma.padelPlayerProfile.create({
    data: {
      organizerId,
      userId,
      fullName: name,
      displayName: name,
      email: email ?? undefined,
    },
    select: { id: true },
  });
  return created.id;
}

// Claim endpoint para convites (Padel v2).
export async function GET(_, { params }: { params: { token: string } }) {
  const token = params?.token;
  if (!token) return NextResponse.json({ ok: false, error: "INVALID_TOKEN" }, { status: 400 });

  const pairing = await prisma.padelPairing.findFirst({
    where: { partnerInviteToken: token },
    select: {
      id: true,
      pairingStatus: true,
      lifecycleStatus: true,
      partnerLinkExpiresAt: true,
    }
  });
}

```

```

        lockedUntil: true,
        paymentMode: true,
        eventId: true,
        organizerId: true,
        player1UserId: true,
        player2UserId: true,
        deadlineAt: true,
    },
});

if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

if (pairing.partnerLinkExpiresAt && pairing.partnerLinkExpiresAt.getTime() < Date.now()) {
    return NextResponse.json({ ok: false, error: "INVITE_EXPIRED" }, { status: 410 });
}
if (pairing.player2UserId) {
    return NextResponse.json({ ok: false, error: "INVITE_ALREADY_USED" }, { status: 409 });
}

const ticketTypes = await prisma.ticketType.findMany({
    where: { eventId: pairing.eventId, status: "ON_SALE" },
    select: { id: true, name: true, price: true, currency: true },
    orderBy: { price: "asc" },
});

const padelEvent = await buildPadelEventSnapshot(pairing.eventId);

return NextResponse.json(
    { ok: true, pairing, ticketTypes, organizerId: pairing.organizerId, status: "PREVIEW_ONLY", padelEvent },
    { status: 200 },
);
}

export async function POST(_: NextRequest, { params }: { params: { token: string } }) {
    const supabase = await createSupabaseServer();
    const {
        data: { user },
    } = await supabase.auth.getUser();

    if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

    const token = params?.token;
    if (!token) return NextResponse.json({ ok: false, error: "INVALID_TOKEN" }, { status: 400 });

    const pairing = await prisma.padelPairing.findFirst({
        where: { partnerInviteToken: token },
        include: { slots: true },
    });

    if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
    if (pairing.partnerLinkExpiresAt && pairing.partnerLinkExpiresAt.getTime() < Date.now()) {
        return NextResponse.json({ ok: false, error: "INVITE_EXPIRED" }, { status: 410 });
    }
    if (pairing.lifecycleStatus === "CANCELLED_INCOMPLETE" || pairing.pairingStatus === "CANCELLED") {
        return NextResponse.json({ ok: false, error: "PAIRING_CANCELLED" }, { status: 400 });
    }
    if (pairing.player2UserId) {
        return NextResponse.json({ ok: false, error: "INVITE_ALREADY_USED" }, { status: 409 });
    }

    const pendingSlot = pairing.slots.find((s) => s.slotStatus === "PENDING");
    if (!pendingSlot) {
        return NextResponse.json({ ok: false, error: "NO_PENDING_SLOT" }, { status: 400 });
    }

    // SPLIT sem pagamento do parceiro: devolve ação para checkout
    if (pairing.paymentMode === PadelPaymentMode.SPLIT && pendingSlot.paymentStatus !== PadelPairingPaymentStatus.PAID) {
        return NextResponse.json({ ok: false, error: "PAYMENT_REQUIRED", action: "CHECKOUT_PARTNER" }, { status: 402 });
    }
    if (pairing.deadlineAt && pairing.deadlineAt.getTime() < Date.now()) {
        return NextResponse.json({ ok: false, error: "PAIRING_EXPIRED" }, { status: 410 });
    }

    // Guard: utilizador já tem pairing ativo no torneio?
    const existingActive = await prisma.padelPairing.findFirst({
        where: {
            eventId: pairing.eventId,
            lifecycleStatus: { not: "CANCELLED_INCOMPLETE" },
        }
    });
}
```

```

    OR: [{ player1UserId: user.id }, { player2UserId: user.id }],
    NOT: { id: pairing.id },
  },
  select: { id: true },
);
};

if (existingActive) {
  return NextResponse.json({ ok: false, error: "PAIRING_ALREADY_ACTIVE" }, { status: 409 });
}

const config = await prisma.padelTournamentConfig.findUnique({
  where: { eventId: pairing.eventId },
  select: { eligibilityType: true },
});

const [captainProfile, partnerProfile] = await Promise.all([
  pairing.player1UserId
    ? prisma.profile.findUnique({ where: { id: pairing.player1UserId }, select: { gender: true } })
      : Promise.resolve(null),
  prisma.profile.findUnique({ where: { id: user.id }, select: { gender: true } }),
]);
};

const eligibility = validateEligibility(
  (config?.eligibilityType as PadelEligibilityType) ?? PadelEligibilityType.OPEN,
  (captainProfile?.gender as Gender | null) ?? null,
  partnerProfile?.gender as Gender | null,
);
};

if (!eligibility.ok) {
  return NextResponse.json(
    { ok: false, error: eligibility.code },
    { status: eligibility.code === "GENDER_REQUIRED_FOR_TOURNAMENT" ? 403 : 409 },
  );
}

try {
  const playerProfileId = await ensurePlayerProfile({ organizerId: pairing.organizerId, userId: user.id });
  const updated = await prisma.$transaction(async (tx) => {
    // Se já tem ticket, validar apropriação
    if (pendingSlot.ticketId) {
      const ticket = await tx.ticket.findUnique({ where: { id: pendingSlot.ticketId } });
      if (!ticket) throw new Error("TICKET_NOT_FOUND");
      if (ticket.userId && ticket.userId !== user.id) throw new Error("TICKET_ALREADY_CLAIMED");

      await tx.ticket.update({
        where: { id: pendingSlot.ticketId },
        data: { userId: user.id },
      });
    }

    const action: PairingAction =
      pendingSlot.paymentStatus === PadelPairingPaymentStatus.PAID ? "PARTNER_PAID" : "PARTNER_ASSIGNED";
    const nextStatus = transition(
      (pairing.lifecycleStatus as PadelPairingLifecycleStatus) ?? "PENDING_ONE_PAID",
      action,
    );
    const partnerAcceptedAt = new Date();
    const partnerPaidAt =
      pendingSlot.paymentStatus === PadelPairingPaymentStatus.PAID ? new Date() : null;

    const updatedPairing = await tx.padelPairing.update({
      where: { id: pairing.id },
      data: {
        player2UserId: user.id,
        partnerInviteToken: null,
        partnerLinkToken: null,
        partnerInviteUsedAt: partnerAcceptedAt,
        partnerAcceptedAt,
        partnerPaidAt,
        lifecycleStatus: nextStatus,
        guaranteeStatus:
          pendingSlot.paymentStatus === PadelPairingPaymentStatus.PAID ? "SUCCEEDED" : pairing.guaranteeStatus,
        graceUntilAt:
          pendingSlot.paymentStatus === PadelPairingPaymentStatus.PAID ? null : pairing.graceUntilAt,
        slots: {
          update: {
            where: { id: pendingSlot.id },
            data: {
              profileId: user.id,
              playerProfileId,
            }
          }
        }
      }
    });
  });
}

```

```

        slotStatus: PadelPairingSlotStatus.FILLED,
        paymentStatus:
          pendingSlot.paymentStatus === "PAID"
            ? pendingSlot.paymentStatus
            : PadelPairingPaymentStatus.PAID,
      },
    },
  },
  include: { slots: true },
);

const stillPending = updatedPairing.slots.some((s) => s.slotStatus === "PENDING");
if (!stillPending && updatedPairing.pairingStatus !== "COMPLETE") {
  return tx.padelPairing.update({
    where: { id: pairing.id },
    data: { pairingStatus: "COMPLETE" },
    include: { slots: true },
  });
}

return updatedPairing;
});

return NextResponse.json({ ok: true, pairing: updated }, { status: 200 });
} catch (err) {
  if (err instanceof Error && err.message === "TICKET_ALREADY CLAIMED") {
    return NextResponse.json({ ok: false, error: "TICKET_ALREADY CLAIMED" }, { status: 409 });
  }
  console.error("[padel/pairings][claim][POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/padel/pairings/my/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

// Lista pairings Padel v2 associados ao utilizador (captão ou slot preenchido).
export async function GET(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const eventIdParam = req.nextUrl.searchParams.get("eventId");
  const eventId = eventIdParam ? Number(eventIdParam) : null;

  try {
    const pairings = await prisma.padelPairing.findMany({
      where: {
        ...(eventId ? { eventId } : {}),
        OR: [
          { createdById: user.id },
          { slots: { some: { profileId: user.id } } },
        ],
      },
      include: {
        slots: {
          include: {
            ticket: {
              select: { id: true, status: true, stripePaymentIntentId: true },
            },
          },
        },
        event: {
          select: { id: true, title: true, slug: true, organizerId: true, templateType: true },
        },
        category: { select: { label: true } },
      },
    });
  }
}

```

```

        orderBy: { updatedAt: "desc" },
    });
    return NextResponse.json({ ok: true, pairings }, { status: 200 });
} catch (err) {
    console.error("[padel/pairings/my] query error", err);
    // fallback seguro para não partir o UI se a tabela ainda não existir ou schema estiver desfasado
    return NextResponse.json({ ok: true, pairings: [], warning: "PAIRINGS_UNAVAILABLE" }, { status: 200 });
}
}

```

app/api/padel/pairings/open/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import {
    Gender,
    PadelEligibilityType,
    PadelPairingLifecycleStatus,
    PadelPairingPaymentStatus,
    PadelPairingSlotStatus,
    PadelPaymentMode,
} from "@prisma/client";
import { createSupabaseServer } from "@lib/supabaseServer";
import { prisma } from "@lib/prisma";
import { validateEligibility } from "@domain/padelEligibility";
import { PairingAction, transition } from "@domain/padelPairingStateMachine";

// Permite um parceiro juntar-se a um pairing com mode LOOKING_FOR_PARTNER / isPublicOpen sem token.
export async function POST(req: NextRequest) {
    const supabase = await createSupabaseServer();
    const {
        data: { user },
    } = await supabase.auth.getUser();
    if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

    const body = (await req.json()).catch(() => null) as Record<string, unknown> | null;
    const pairingId = typeof body?.pairingId === "number" ? body.pairingId : Number(body?.pairingId);
    if (!Number.isFinite(pairingId)) {
        return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });
    }

    const pairing = await prisma.padelPairing.findUnique({
        where: { id: pairingId },
        include: { slots: true, event: { select: { organizerId: true } } },
    });
    if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });
    if (pairing.player2UserId) {
        return NextResponse.json({ ok: false, error: "INVITE_ALREADY_USED" }, { status: 409 });
    }
    if (pairing.deadlineAt && pairing.deadlineAt.getTime() < Date.now()) {
        return NextResponse.json({ ok: false, error: "PAIRING_EXPIRED" }, { status: 410 });
    }
    if (
        pairing.pairingJoinMode !== "LOOKING_FOR_PARTNER" ||
        !pairing.isPublicOpen
    ) {
        return NextResponse.json({ ok: false, error: "FORBIDDEN" }, { status: 403 });
    }

    // Guard: utilizador já tem pairing ativo no torneio?
    const existingActive = await prisma.padelPairing.findFirst({
        where: {
            eventId: pairing.eventId,
            lifecycleStatus: { not: "CANCELLED_INCOMPLETE" },
            OR: [{ player1UserId: user.id }, { player2UserId: user.id }],
            NOT: { id: pairing.id },
        },
        select: { id: true },
    });
    if (existingActive) {
        return NextResponse.json({ ok: false, error: "PAIRING_ALREADY_ACTIVE" }, { status: 409 });
    }

    const [captainProfile, partnerProfile] = await Promise.all([
        pairing.player1UserId
    ])

```

```

    ? prisma.profile.findUnique({ where: { id: pairing.player1UserId }, select: { gender: true } })
      : Promise.resolve(null),
    prisma.profile.findUnique({ where: { id: user.id }, select: { gender: true } }),
  );
};

const config = await prisma.padelTournamentConfig.findUnique({
  where: { eventId: pairing.eventId },
  select: { eligibilityType: true },
});

const eligibility = validateEligibility(
  (config?.eligibilityType as PadelEligibilityType) ?? PadelEligibilityType.OPEN,
  (captainProfile?.gender as Gender | null) ?? null,
  partnerProfile?.gender as Gender | null,
);

if (!eligibility.ok) {
  return NextResponse.json(
    { ok: false, error: eligibility.code },
    { status: eligibility.code === "GENDER_REQUIRED_FOR_TOURNAMENT" ? 403 : 409 },
  );
}

const pendingSlot = pairing.slots.find((s) => s.slotStatus === "PENDING");
if (!pendingSlot) {
  return NextResponse.json({ ok: false, error: "NO_PENDING_SLOT" }, { status: 400 });
}

if (pairing.paymentMode === PadelPaymentMode.SPLIT && pendingSlot.paymentStatus !== PadelPairingPaymentStatus.PAID) {
  return NextResponse.json({ ok: false, error: "PAYMENT_REQUIRED", action: "CHECKOUT_PARTNER" }, { status: 402 });
}

try {
  const action: PairingAction =
    pendingSlot.paymentStatus === PadelPairingPaymentStatus.PAID ? "PARTNER_PAID" : "PARTNER_ASSIGNED";
  const nextStatus = transition(
    (pairing.lifecycleStatus as PadelPairingLifecycleStatus) ?? "PENDING_ONE_PAID",
    action,
  );
  const partnerAcceptedAt = new Date();
  const partnerPaidAt =
    pendingSlot.paymentStatus === PadelPairingPaymentStatus.PAID ? new Date() : null;

  const updated = await prisma.$transaction(async (tx) => {
    const updatedPairing = await tx.padelPairing.update({
      where: { id: pairing.id },
      data: {
        player2UserId: user.id,
        partnerInviteToken: null,
        partnerLinkToken: null,
        partnerInviteUsedAt: partnerAcceptedAt,
        partnerAcceptedAt,
        partnerPaidAt,
        lifecycleStatus: nextStatus,
        guaranteeeStatus:
          pendingSlot.paymentStatus === PadelPairingPaymentStatus.PAID ? "SUCCEEDED" : pairing.guaranteeStatus,
        graceUntilAt:
          pendingSlot.paymentStatus === PadelPairingPaymentStatus.PAID ? null : pairing.graceUntilAt,
        slots: {
          update: {
            where: { id: pendingSlot.id },
            data: {
              profileId: user.id,
              slotStatus: PadelPairingSlotStatus.FILLED,
              paymentStatus:
                pendingSlot.paymentStatus === "PAID"
                  ? pendingSlot.paymentStatus
                  : PadelPairingPaymentStatus.PAID,
            },
          },
        },
      },
      include: { slots: true },
    });
    const stillPending = updatedPairing.slots.some((s) => s.slotStatus === "PENDING");
    if (!stillPending && updatedPairing.pairingStatus !== "COMPLETE") {
      return tx.padelPairing.update({
        where: { id: pairing.id },
        data: { pairingStatus: "COMPLETE" },
        include: { slots: true },
      });
    }
  });
}

```

```

        }
        return updatedPairing;
    });

    return NextResponse.json({ ok: true, pairing: updated }, { status: 200 });
} catch (err) {
    console.error("[padel/pairings][open][POST]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/padel/pairings/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import {
    Gender,
    PadelEligibilityType,
    PadelPairingLifecycleStatus,
    PadelPairingPaymentStatus,
    PadelPairingSlotRole,
    PadelPairingSlotStatus,
    PadelPaymentMode,
    PadelPairingJoinMode,
} from "@prisma/client";
import { randomUUID } from "crypto";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { buildPadelEventSnapshot } from "@/lib/padel/eventSnapshot";
import { validateEligibility } from "@/domain/padelEligibility";
import { upsertActiveHold } from "@/domain/padelPairingHold";
import {
    clampDeadlineHours,
    computeDeadlineAt,
    computePartnerLinkExpiresAt,
} from "@/domain/padelDeadlines";

async function syncPlayersFromSlots({
    organizerId,
    slots,
}: {
    organizerId: number;
    slots: Array<{
        profileId: string | null;
        invitedContact: string | null;
    }>;
}) {
    const profileIds = Array.from(
        new Set(slots.map((s) => s.profileId).filter(Boolean) as string[])
    );

    // 1) Jogadores ligados a perfis existentes
    if (profileIds.length > 0) {
        const profiles = await prisma.profile.findMany({
            where: { id: { in: profileIds } },
            select: { id: true, fullName: true, contactPhone: true },
        });
        for (const profile of profiles) {
            const exists = await prisma.padelPlayerProfile.findFirst({
                where: { organizerId, userId: profile.id },
                select: { id: true },
            });
            if (exists) continue;
            const fullName = profile.fullName?.trim() || "Jogador ORYA";
            await prisma.padelPlayerProfile.create({
                data: {
                    organizerId,
                    userId: profile.id,
                    fullName,
                    displayName: fullName,
                    phone: profile.contactPhone || undefined,
                    isActive: true,
                },
            });
        }
    }
}

```

```

}

// 2) Convites por contacto (email ou telefone)
const invitedContacts = Array.from(
  new Set(
    slots
      .map((s) => s.invitedContact?.trim())
      .filter(Boolean) as string[],
  ),
);
for (const contact of invitedContacts) {
  const isEmail = contact.includes("@");
  const email = isEmail ? contact.toLowerCase() : null;
  const phone = !isEmail ? contact : null;
  if (email) {
    const exists = await prisma.padelPlayerProfile.findUnique({
      where: { organizerId_email: { organizerId, email } },
      select: { id: true },
    });
    if (exists) continue;
  }
  await prisma.padelPlayerProfile.create({
    data: {
      organizerId,
      fullName: contact,
      displayName: contact,
      email: email || undefined,
      phone: phone || undefined,
      isActive: true,
    },
  });
}
}

// Cria pairing Padel v2 após checkout ou setup inicial.
// Evita mexer no legacy; valida flag padel_v2_enabled.
export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
  const eventId = body && typeof body.eventId === "number" ? Number(body.eventId);
  const organizerId = body && typeof body.organizerId === "number" ? Number(body.organizerId);
  const categoryId = body && typeof body.categoryId === "number" ? body.categoryId : body?.categoryId === null ? null : Number(body?.categoryId);
  const paymentMode = typeof body?.paymentMode === "string" ? (body?.paymentMode as PadelPaymentMode) : null;
  const pairingJoinModeRaw = typeof body?.pairingJoinMode === "string" ? (body?.pairingJoinMode as PadelPairingJoinMode) : "INVITE_PARTNER";
  const createdByTicketId = typeof body?.createdByTicketId === "string" ? body?.createdByTicketId : null;
  const inviteToken = typeof body?.inviteToken === "string" ? body?.inviteToken : null;
  const inviteExpiresAt = body?.inviteExpiresAt ? new Date(String(body.inviteExpiresAt)) : null;
  const lockedUntil = body?.lockedUntil ? new Date(String(body.lockedUntil)) : null;
  const isPublicOpen = Boolean(body?.isPublicOpen);

  if (!eventId || !organizerId || !paymentMode || !["FULL", "SPLIT"].includes(paymentMode)) {
    return NextResponse.json({ ok: false, error: "INVALID_INPUT" }, { status: 400 });
  }

  // Basic guard: only proceed if padel_v2_enabled is active on the tournament config.
  const config = await prisma.padelTournamentConfig.findUnique({
    where: { eventId },
    select: {
      padelV2Enabled: true,
      organizerId: true,
      eligibilityType: true,
      splitDeadlineHours: true,
    },
  });
  if (!config?.padelV2Enabled || config.organizerId !== organizerId) {
    return NextResponse.json({ ok: false, error: "PADEL_V2_DISABLED" }, { status: 400 });
  }

  const profile = await prisma.profile.findUnique({
    where: { id: user.id },
  },

```

```

    select: { gender: true },
  });

const eligibility = validateEligibility(
  (config.eligibilityType as PadelEligibilityType) ?? PadelEligibilityType.OPEN,
  profile?.gender as Gender | null,
  null,
);
if (!eligibility.ok) {
  return NextResponse.json(
    { ok: false, error: eligibility.code },
    { status: eligibility.code === "GENDER_REQUIRED_FOR_TOURNAMENT" ? 403 : 409 },
  );
}

// Invariante: 1 pairing ativo por evento+user
const existingActive = await prisma.padelPairing.findFirst({
  where: {
    eventId,
    lifecycleStatus: { not: "CANCELLED_INCOMPLETE" },
    OR: [{ player1UserId: user.id }, { player2UserId: user.id }],
  },
  select: { id: true },
});
if (existingActive) {
  return NextResponse.json(
    { ok: false, error: "PAIRING_ALREADY_ACTIVE" },
    { status: 409 },
  );
}

// Build slots: se não vierem slots no payload, cria duas entradas (capitão + parceiro pendente)
type IncomingSlot = {
  ticketId?: unknown;
  profileId?: unknown;
  invitedContact?: unknown;
  isPublicOpen?: unknown;
  slotRole?: unknown;
  slotStatus?: unknown;
  paymentStatus?: unknown;
};

const normalizeSlot = (slot: IncomingSlot | unknown) => {
  if (typeof slot !== "object" || slot === null) return null;
  const s = slot as IncomingSlot;
  const roleRaw = typeof s.slotRole === "string" ? s.slotRole : "PARTNER";
  const statusRaw = typeof s.slotStatus === "string" ? s.slotStatus : "PENDING";
  const payRaw = typeof s.paymentStatus === "string" ? s.paymentStatus : "UNPAID";

  return {
    ticketId: typeof s.ticketId === "string" ? s.ticketId : null,
    profileId: typeof s.profileId === "string" ? s.profileId : null,
    invitedContact: typeof s.invitedContact === "string" ? s.invitedContact : null,
    isPublicOpen: Boolean(s.isPublicOpen),
    slotRole: roleRaw === "CAPTAIN" ? PadelPairingSlotRole.CAPTAIN : PadelPairingSlotRole.PARTNER,
    slotStatus:
      statusRaw === "FILLED"
        ? PadelPairingSlotStatus.FILLED
        : statusRaw === "CANCELLED"
        ? PadelPairingSlotStatus.CANCELLED
        : PadelPairingSlotStatus.PENDING,
    paymentStatus: payRaw === "PAID" ? PadelPairingPaymentStatus.PAID : PadelPairingPaymentStatus.UNPAID,
  };
};

const incomingSlots = Array.isArray(body?.slots) ? (body!.slots as unknown[]) : [];
const slotsToCreate =
  incomingSlots.length > 0
    ? (incomingSlots
        .map(slot) => normalizeSlot(slot))
        .filter(Boolean) as Array<{
          ticketId: string | null;
          profileId: string | null;
          invitedContact: string | null;
          isPublicOpen: boolean;
          slotRole: PadelPairingSlotRole;
          slotStatus: PadelPairingSlotStatus;
          paymentStatus: PadelPairingPaymentStatus;
        }>
    : [];

```

```

        }>
      : [
        {
          ticketId: createdByTicketId,
          profileId: user.id,
          invitedContact: null,
          isPublicOpen,
          slotRole: PadelPairingSlotRole.CAPTAIN,
          slotStatus: createdByTicketId ? PadelPairingSlotStatus.FILLED : PadelPairingSlotStatus.PENDING,
          paymentStatus: PadelPairingPaymentStatus.PAID,
        },
        {
          ticketId: null,
          profileId: null,
          invitedContact: null,
          isPublicOpen,
          slotRole: PadelPairingSlotRole.PARTNER,
          slotStatus: PadelPairingSlotStatus.PENDING,
          paymentStatus: paymentMode === "FULL" ? PadelPairingPaymentStatus.PAID : PadelPairingPaymentStatus.UNPAID,
        },
      ];
    }

  try {
    const now = new Date();
    const clampedDeadlineHours = clampDeadlineHours(config.splitDeadlineHours ?? undefined);
    const deadlineAt = computeDeadlineAt(now, clampedDeadlineHours);
    const partnerLinkExpiresAtNormalized =
      inviteExpiresAt && !Number.isNaN(inviteExpiresAt.getTime())
        ? inviteExpiresAt
        : computePartnerLinkExpiresAt(now, undefined);
    const partnerInviteToken =
      pairingJoinModeRaw === "INVITE_PARTNER"
        ? inviteToken || randomUUID()
        : null;

    const pairing = await prisma.$transaction(async (tx) => {
      const created = await tx.padelPairing.create({
        data: {
          eventId,
          organizerId,
          categoryId: Number.isFinite(categoryId as number) ? (categoryId as number) : null,
          paymentMode,
          createdByUserId: user.id,
          player1UserId: user.id,
          createdByTicketId,
          partnerInviteToken,
          partnerLinkToken: partnerInviteToken,
          partnerLinkExpiresAt: partnerInviteToken ? partnerLinkExpiresAtNormalized : null,
          partnerInvitedAt: partnerInviteToken ? now : null,
          partnerSwapAllowedUntilAt: deadlineAt,
          deadlineAt,
          guaranteeStatus: paymentMode === "SPLIT" ? "ARMED" : "NONE",
          lockedUntil,
          isPublicOpen,
          pairingJoinMode: pairingJoinModeRaw ?? PadelPairingJoinMode.INVITE_PARTNER,
          lifecycleStatus: PadelPairingLifecycleStatus.PENDING_ONE_PAID,
          slots: {
            create: slotsToCreate,
          },
        },
        include: { slots: true },
      });

      await upsertActiveHold(tx, { pairingId: created.id, eventId, ttlMinutes: 30 });
      return created;
    });

    // Auto-criar perfis de jogador para o organizador (roster)
    await syncPlayersFromSlots({
      organizerId,
      slots: pairing.slots.map((s) => ({
        profileId: (s as { profileId?: string | null }).profileId ?? null,
        invitedContact: (s as { invitedContact?: string | null }).invitedContact ?? null,
      })),
    });

    return NextResponse.json({ ok: true, pairing }, { status: 200 });
  } catch (err) {

```

```

        console.error("[padel/pairings][POST]", err);
        return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
    }

// GET simples para fornecer pairing + ticketTypes (para checkout/claim UI)
export async function GET(req: NextRequest) {
    const supabase = await createSupabaseServer();
    const {
        data: { user },
    } = await supabase.auth.getUser();

    if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

    const pairingId = Number(req.nextUrl.searchParams.get("id"));
    const eventId = Number(req.nextUrl.searchParams.get("eventId"));

    if (Number.isFinite(pairingId)) {
        const pairing = await prisma.padelPairing.findUnique({
            where: { id: pairingId },
            include: {
                slots: { include: { playerProfile: true } },
            },
        });
        if (!pairing) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

        const ticketTypes = await prisma.ticketType.findMany({
            where: { eventId: pairing.eventId, status: "ON_SALE" },
            select: { id: true, name: true, price: true, currency: true },
            orderBy: { price: "asc" },
        });

        const padelEvent = await buildPadelEventSnapshot(pairing.eventId);

        return NextResponse.json({ ok: true, pairing, ticketTypes, padelEvent }, { status: 200 });
    }

    if (!Number.isFinite(eventId)) {
        return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });
    }

    const pairings = await prisma.padelPairing.findMany({
        where: { eventId },
        include: {
            slots: { include: { playerProfile: true } },
        },
        orderBy: [{ createdAt: "asc" }],
    });

    return NextResponse.json({ ok: true, pairings }, { status: 200 });
}

```

app/api/padel/players/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

export async function GET(req: NextRequest) {
    const supabase = await createSupabaseServer();
    const {
        data: { user },
    } = await supabase.auth.getUser();

    if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

    const organizerIdParam = req.nextUrl.searchParams.get("organizerId");
    const parsedOrgId = organizerIdParam ? Number(organizerIdParam) : null;
    const { organizer } = await getActiveOrganizerForUser(user.id, {
        organizerId: Number.isFinite(parsedOrgId) ? parsedOrgId : undefined,
    });
}

```

```

    roles: allowedRoles,
  );
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const players = await prisma.padelPlayerProfile.findMany({
    where: { organizerId: organizer.id },
    orderBy: [{ createdAt: "desc" }],
  });

  return NextResponse.json({ ok: true, items: players }, { status: 200 });
}

export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
  if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });

  const organizerIdParam = body.organizerId ?? req.nextUrl.searchParams.get("organizerId");
  const parsedOrgId = typeof organizerIdParam === "number" ? organizerIdParam : Number(organizerIdParam) : null;
  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: Number.isFinite(parsedOrgId) ? parsedOrgId : undefined,
    roles: allowedRoles,
  });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const fullName = typeof body.fullName === "string" ? body.fullName.trim() : "";
  const displayName = typeof body.displayName === "string" ? body.displayName.trim() : fullName;
  const email = typeof body.email === "string" ? body.email.trim().toLowerCase() : null;
  const phone = typeof body.phone === "string" ? body.phone.trim() : null;
  const gender = typeof body.gender === "string" ? body.gender.trim() : null;
  const level = typeof body.level === "string" ? body.level.trim() : null;
  const preferredSide = typeof body.preferredSide === "string" ? body.preferredSide.trim().toUpperCase() : null;
  const clubName = typeof body.clubName === "string" ? body.clubName.trim() : null;
  const birthDate = typeof body.birthDate === "string" && body.birthDate.trim() ? new Date(body.birthDate) : null;
  const isActive = typeof body.isActive === "boolean" ? body.isActive : true;
  const notes = typeof body.notes === "string" ? body.notes.trim() : null;

  if (!fullName) return NextResponse.json({ ok: false, error: "FULLNAME_REQUIRED" }, { status: 400 });

  try {
    const player = email
      ? await prisma.padelPlayerProfile.upsert({
          where: { organizerId_email: { organizerId: organizer.id, email } },
          create: {
            organizerId: organizer.id,
            fullName,
            displayName: displayName || fullName,
            email,
            phone,
            gender,
            level,
            isActive,
            notes: notes || undefined,
            preferredSide: preferredSide || undefined,
            clubName: clubName || undefined,
            birthDate: birthDate && !Number.isNaN(birthDate.getTime()) ? birthDate : undefined,
          },
          update: {
            fullName,
            displayName: displayName || fullName,
            phone,
            gender,
            level,
            isActive,
            notes: notes || undefined,
            preferredSide: preferredSide || undefined,
            clubName: clubName || undefined,
            birthDate: birthDate && !Number.isNaN(birthDate.getTime()) ? birthDate : undefined,
          },
        })
      : await prisma.padelPlayerProfile.create({
  
```

```

    data: {
      organizerId: organizer.id,
      fullName,
      displayName: displayName || fullName,
      phone,
      gender,
      level,
      isActive,
      notes: notes || undefined,
      preferredSide: preferredSide || undefined,
      clubName: clubName || undefined,
      birthDate: birthDate && !Number.isNaN(birthDate.getTime()) ? birthDate : undefined,
    },
  });
}

return NextResponse.json({ ok: true, player }, { status: 201 });
} catch (err) {
  console.error("[padel/players][POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/padel/rankings/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import { PadelPointsTable, isValidScore } from "@/lib/padel/validation";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

export async function GET(req: NextRequest) {
  const organizerId = req.nextUrl.searchParams.get("organizerId");
  const eventId = req.nextUrl.searchParams.get("eventId");

  if (eventId) {
    const eId = Number(eventId);
    if (!Number.isFinite(eId)) return NextResponse.json({ ok: false, error: "INVALID_EVENT" }, { status: 400 });

    const entries = await prisma.padelRankingEntry.findMany({
      where: { eventId: eId },
      include: { player: true },
      orderBy: [{ points: "desc" }],
    });
    const items = entries.map((row, idx) => ({
      position: idx + 1,
      points: row.points,
      player: {
        id: row.player.id,
        fullName: row.player.fullName,
        level: row.player.level,
      },
    }));
    return NextResponse.json({ ok: true, items }, { status: 200 });
  }

  if (!organizerId) {
    return NextResponse.json({ ok: false, error: "MISSING_ORGANIZER" }, { status: 400 });
  }
  const oId = Number(organizerId);
  if (!Number.isFinite(oId)) {
    return NextResponse.json({ ok: false, error: "INVALID_ORGANIZER" }, { status: 400 });
  }

  const entries = await prisma.padelRankingEntry.findMany({
    where: { organizerId: oId },
    include: { player: true },
  });

  const aggregated = Object.values(
    entries.reduce<Record<number, { player: any; points: number }>>>((acc, row) => {
      const key = row.playerId;
      acc[key] = acc[key] || { player: row.player, points: 0 };
      acc[key].points += row.points;
    })
  );
  return NextResponse.json({ ok: true, aggregated }, { status: 200 });
}

```

```

    if (!acc[key]) acc[key] = { player: row.player, points: 0 };
    acc[key].points += row.points;
    return acc;
}, {}),
).sort((a, b) => b.points - a.points);

const items = aggregated.map((item, idx) => ({
  position: idx + 1,
  points: item.points,
  player: {
    id: item.player.id,
    fullName: item.player.fullName,
    level: item.player.level,
  },
}));

return NextResponse.json({ ok: true, items }, { status: 200 });
}

export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
  if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });

  const eventId = typeof body.eventId === "number" ? body.eventId : Number(body.eventId);
  if (!Number.isFinite(eventId)) return NextResponse.json({ ok: false, error: "INVALID_EVENT" }, { status: 400 });

  const event = await prisma.event.findUnique({
    where: { id: eventId, isDeleted: false },
    select: { id: true, organizerId: true },
  });
  if (!event || !event.organizerId) return NextResponse.json({ ok: false, error: "EVENT_NOT_FOUND" }, { status: 404 });

  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: event.organizerId,
    roles: allowedRoles,
  });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const config = await prisma.padelTournamentConfig.findUnique({
    where: { eventId },
    select: { ruleSetId: true },
  });
  const ruleSet = config?.ruleSetId
    ? await prisma.padelRuleSet.findUnique({ where: { id: config.ruleSetId } })
    : null;
  const pointsTable: PadelPointsTable = (ruleSet?.pointsTable as any) || { WIN: 3, LOSS: 0 };

  const matches = await prisma.padelMatch.findMany({
    where: { eventId, status: "DONE" },
    include: {
      teamA: { include: { player1: true, player2: true } },
      teamB: { include: { player1: true, player2: true } },
    },
  });

  const playerPoints: Record<number, number> = {};
  const winPts = pointsTable.WIN ?? 3;
  const lossPts = pointsTable.LOSS ?? 0;

  matches.forEach((m) => {
    if (!isValidScore(m.score) || !m.teamA || !m.teamB) return;
    const sets = m.score.sets || [];
    let aSets = 0;
    let bSets = 0;
    sets.forEach((s) => {
      if ((s as any).teamA > (s as any).teamB) aSets += 1;
      else if ((s as any).teamB > (s as any).teamA) bSets += 1;
    });
    if (aSets === bSets) return;
    const winner = aSets > bSets ? "A" : "B";
    const losers = winner === "A" ? "B" : "A";
    const player1 = m.teamA.id === user.id ? user : (m.teamB as any).id === user.id ? (m.teamB as any).player1 : (m.teamB as any).player2;
    const player2 = m.teamA.id === user.id ? (m.teamB as any).player1 : user;
    if (winner === "A") {
      playerPoints[player1.id] += winPts;
      playerPoints[player2.id] += lossPts;
    } else {
      playerPoints[player1.id] += lossPts;
      playerPoints[player2.id] += winPts;
    }
  });
}

```

```

const winnerTeam = winner === "A" ? m.teamA : m.teamB;
const loserTeam = losers === "A" ? m.teamA : m.teamB;

const award = (team: typeof m.teamA, pts: number) => {
  if (team?.player1) playerPoints[team.player1.id] = (playerPoints[team.player1.id] ?? 0) + pts;
  if (team?.player2) playerPoints[team.player2.id] = (playerPoints[team.player2.id] ?? 0) + pts;
};

award(winnerTeam, winPts);
award(loserTeam, lossPts);
};

await prisma.$transaction(async (tx) => {
  await tx.padelRankingEntry.deleteMany({ where: { eventId } });
  const entries = Object.entries(playerPoints).map(([playerIdStr, points]) => ({
    organizerId: event.organizerId!,
    eventId,
    playerId: Number(playerIdStr),
    points,
    position: null,
  }));
  if (entries.length > 0) {
    await tx.padelRankingEntry.createMany({ data: entries });
  }
});

return NextResponse.json({ ok: true }, { status: 200 });
}

```

app/api/padel/rulesets/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";
import {
  isValidPointsTable,
  isValidTieBreakRules,
  PadelPointsTable,
  PadelTieBreakRule,
} from "@/lib/padel/validation";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

export async function GET(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const organizerIdParam = req.nextUrl.searchParams.get("organizerId");
  const parsedOrgId = organizerIdParam ? Number(organizerIdParam) : null;
  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: Number.isFinite(parsedOrgId) ? parsedOrgId : undefined,
    roles: allowedRoles,
  });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const items = await prisma.padelRuleSet.findMany({
    where: { organizerId: organizer.id },
    orderBy: { createdAt: "desc" },
  });

  return NextResponse.json({ ok: true, items }, { status: 200 });
}

export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

```

```

if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });

const organizerIdParam = body.organizerId ?? req.nextUrl.searchParams.get("organizerId");
const parsedOrgId = typeof organizerIdParam === "number" ? organizerIdParam : organizerIdParam ? Number(organizerIdParam) : null;
const { organizer } = await getActiveOrganizerForUser(user.id, {
  organizerId: Number.isFinite(parsedOrgId) ? parsedOrgId : undefined,
  roles: allowedRoles,
});
if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

const name = typeof body.name === "string" ? body.name.trim() : "";
const tieBreakRulesRaw = body.tieBreakRules as unknown;
const pointsTableRaw = body.pointsTable as unknown;
const enabledFormats = Array.isArray(body.enabledFormats)
  ? (body.enabledFormats as unknown[]).map((f) => String(f)).filter(Boolean)
  : undefined;
const season = typeof body.season === "string" ? body.season.trim() : null;
const year = typeof body.year === "number" ? body.year : null;
const id = typeof body.id === "number" ? body.id : null;

if (!name) return NextResponse.json({ ok: false, error: "NAME_REQUIRED" }, { status: 400 });

if (!isValidTieBreakRules(tieBreakRulesRaw)) {
  return NextResponse.json({ ok: false, error: "INVALID_TIE_BREAK_RULES" }, { status: 400 });
}
if (!isValidPointsTable(pointsTableRaw)) {
  return NextResponse.json({ ok: false, error: "INVALID_POINTS_TABLE" }, { status: 400 });
}

const tieBreakRules = tieBreakRulesRaw as PadelTieBreakRule[];
const pointsTable = pointsTableRaw as PadelPointsTable;

try {
  const ruleSet = id
    ? await prisma.padelRuleSet.update({
        where: { id },
        data: {
          organizerId: organizer.id,
          name,
          tieBreakRules,
          pointsTable,
          enabledFormats: enabledFormats ?? undefined,
          season: season || undefined,
          year: year || undefined,
        },
      })
    : await prisma.padelRuleSet.create({
        data: {
          organizerId: organizer.id,
          name,
          tieBreakRules,
          pointsTable,
          enabledFormats: enabledFormats ?? undefined,
          season: season || undefined,
          year: year || undefined,
        },
      });
  return NextResponse.json({ ok: true, ruleSet }, { status: id ? 200 : 201 });
} catch (err) {
  console.error("[padel/rulesets][POST]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/padel/standings/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";

```

```

export async function GET(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);
    const eventId = Number(req.nextUrl.searchParams.get("eventId"));
    if (!Number.isFinite(eventId)) {
      return NextResponse.json({ ok: false, error: "INVALID_EVENT" }, { status: 400 });
    }

    const event = await prisma.event.findUnique({
      where: { id: eventId, isDeleted: false },
      select: { organizerId: true },
    });
    if (!event?.organizerId) {
      return NextResponse.json({ ok: false, error: "EVENT_NOT_FOUND" }, { status: 404 });
    }

    const { organizer } = await getActiveOrganizerForUser(user.id, {
      organizerId: event.organizerId,
      roles: ["OWNER", "CO_OWNER", "ADMIN", "STAFF"],
    });
    if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

    const matches = await prisma.padelMatch.findMany({
      where: { eventId },
      select: {
        id: true,
        categoryId: true,
        pairingAId: true,
        pairingBId: true,
        winnerPairingId: true,
        status: true,
        scoreSets: true,
        groupLabel: true,
      },
    });

    const pairings = await prisma.padelPairing.findMany({
      where: { eventId },
      select: { id: true, categoryId: true },
    });

    type Standing = {
      pairingId: number;
      played: number;
      wins: number;
      losses: number;
      points: number;
      setsFor: number;
      setsAgainst: number;
      groupLabel: string | null;
      categoryId: number | null;
    };

    const standings = new Map<number, Standing>();
    const add = (pid: number, catId: number | null, group: string | null) => {
      if (!standings.has(pid)) {
        standings.set(pid, {
          pairingId: pid,
          played: 0,
          wins: 0,
          losses: 0,
          points: 0,
          setsFor: 0,
          setsAgainst: 0,
          groupLabel: group,
          categoryId: catId,
        });
      }
    };

    pairings.forEach((p) => add(p.id, p.categoryId ?? null, null));

    matches.forEach((m) => {
      if (!m.pairingAId || !m.pairingBId) return;
      add(m.pairingAId, m.categoryId ?? null, m.groupLabel ?? null);
      add(m.pairingBId, m.categoryId ?? null, m.groupLabel ?? null);
    });
  }
}

```

```

const sa = standings.get(m.pairingAId)!;
const sb = standings.get(m.pairingBId)!;

if (m.status !== "DONE" && m.status !== "FINISHED") return;
sa.played += 1;
sb.played += 1;

const sets = Array.isArray(m.scoreSets) ? m.scoreSets : [];
let winsA = 0;
let winsB = 0;
sets.forEach((s: any) => {
  if (Number.isFinite(s.teamA) && Number.isFinite(s.teamB)) {
    sa.setsFor += s.teamA;
    sa.setsAgainst += s.teamB;
    sb.setsFor += s.teamB;
    sb.setsAgainst += s.teamA;
    if (s.teamA > s.teamB) winsA += 1;
    else if (s.teamB > s.teamA) winsB += 1;
  }
});
if (m.winnerPairingId) {
  if (m.winnerPairingId === m.pairingAId) {
    sa.wins += 1;
    sa.points += 2;
    sb.losses += 1;
    sb.points += 1;
  } else if (m.winnerPairingId === m.pairingBId) {
    sb.wins += 1;
    sb.points += 2;
    sa.losses += 1;
    sa.points += 1;
  }
} else {
  if (winsA > winsB) {
    sa.wins += 1;
    sa.points += 2;
    sb.losses += 1;
    sb.points += 1;
  } else if (winsB > winsA) {
    sb.wins += 1;
    sb.points += 2;
    sa.losses += 1;
    sa.points += 1;
  }
}
});

const grouped: Record<string, Standing[]> = {};
standings.forEach((st) => {
  const key = `${st.categoryId ?? "none"}:${st.groupLabel ?? "default"}`;
  if (!grouped[key]) grouped[key] = [];
  grouped[key].push(st);
});

Object.values(grouped).forEach((arr) => {
  arr.sort((a, b) => {
    if (b.points !== a.points) return b.points - a.points;
    const setDiffA = a.setsFor - a.setsAgainst;
    const setDiffB = b.setsFor - b.setsAgainst;
    if (setDiffB !== setDiffA) return setDiffB - setDiffA;
    return b.wins - a.wins;
  });
});

return NextResponse.json({ ok: true, standings: grouped });
} catch (err) {
  console.error("[padel/standings] error", err);
  return NextResponse.json({ ok: false, error: "Erro ao gerar standings." }, { status: 500 });
}
}

```

app/api/padel/teams/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";

```

```

import { OrganizerMemberRole } from "@prisma/client";
import { prisma } from "@lib/prisma";
import { createSupabaseServer } from "@lib/supabaseServer";
import { getActiveOrganizerForUser } from "@lib/organizerContext";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

async function ensureEventAndOrganizer(eventId: number) {
  const event = await prisma.event.findUnique({
    where: { id: eventId, isDeleted: false },
    select: { id: true, organizerId: true, templateType: true },
  });
  if (!event || !event.organizerId) return null;
  return event;
}

export async function GET(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const eventId = Number(req.nextUrl.searchParams.get("eventId"));
  if (!Number.isFinite(eventId)) return NextResponse.json({ ok: false, error: "INVALID_EVENT" }, { status: 400 });

  const event = await ensureEventAndOrganizer(eventId);
  if (!event) return NextResponse.json({ ok: false, error: "EVENT_NOT_FOUND" }, { status: 404 });

  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: event.organizerId,
    roles: allowedRoles,
  });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const teams = await prisma.padelTeam.findMany({
    where: { eventId },
    include: {
      player1: true,
      player2: true,
    },
    orderBy: { createdAt: "desc" },
  });

  return NextResponse.json({ ok: true, items: teams }, { status: 200 });
}

export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
  if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });

  const eventId = typeof body.eventId === "number" ? body.eventId : Number(body.eventId);
  if (!Number.isFinite(eventId)) return NextResponse.json({ ok: false, error: "INVALID_EVENT" }, { status: 400 });

  const event = await ensureEventAndOrganizer(eventId);
  if (!event) return NextResponse.json({ ok: false, error: "EVENT_NOT_FOUND" }, { status: 404 });

  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: event.organizerId!,
    roles: allowedRoles,
  });
  if (!organizer) return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });

  const player1Name = typeof body.player1Name === "string" ? body.player1Name.trim() : "";
  const player2Name = typeof body.player2Name === "string" ? body.player2Name.trim() : "";
  const player1IdBody = typeof body.player1Id === "number" ? body.player1Id : Number(body.player1Id);
  const player2IdBody = typeof body.player2Id === "number" ? body.player2Id : Number(body.player2Id);

  const player1Id = Number.isFinite(player1IdBody) ? player1IdBody : null;
  const player2Id = Number.isFinite(player2IdBody) ? player2IdBody : null;
}

```

```

const createPlayer = async (fullName: string) => {
  if (!fullName.trim()) return null;
  const p = await prisma.padelPlayerProfile.create({
    data: {
      organizerId: organizer.id,
      fullName,
      isActive: true,
    },
  });
  return p.id;
};

const p1 = player1Id || (player1Name ? await createPlayer(player1Name) : null);
const p2 = player2Id || (player2Name ? await createPlayer(player2Name) : null);

const team = await prisma.padelTeam.create({
  data: {
    eventId: event.id,
    player1Id: p1 ?? undefined,
    player2Id: p2 ?? undefined,
    isFromMatchmaking: false,
  },
  include: {
    player1: true,
    player2: true,
  },
});

return NextResponse.json({ ok: true, team }, { status: 201 });
}

```

app/api/padel/tournaments/config/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { OrganizerMemberRole, PadelFormat } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { getActiveOrganizerForUser } from "@/lib/organizerContext";

const allowedRoles: OrganizerMemberRole[] = ["OWNER", "CO_OWNER", "ADMIN"];

export async function GET(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const eventId = Number(req.nextUrl.searchParams.get("eventId"));
  if (!Number.isFinite(eventId)) return NextResponse.json({ ok: false, error: "INVALID_EVENT" }, { status: 400 });

  const config = await prisma.padelTournamentConfig.findUnique({
    where: { eventId },
    include: {
      ruleSet: true,
      category: true,
    },
  });

  return NextResponse.json({ ok: true, config }, { status: 200 });
}

export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as Record<string, unknown> | null;
  if (!body) return NextResponse.json({ ok: false, error: "INVALID_BODY" }, { status: 400 });
}

```

```

const eventId = typeof body.eventId === "number" ? body.eventId : Number(body.eventId);
const organizerIdBody = typeof body.organizerId === "number" ? body.organizerId : Number(body.organizerId);
const format = typeof body.format === "string" ? (body.format as PadelFormat) : null;
const numberOfCourts = typeof body.numberOfCourts === "number" ? body.numberOfCourts : 1;
const ruleSetId = typeof body.ruleSetId === "number" ? body.ruleSetId : null;
const defaultCategoryId = typeof body.defaultCategoryId === "number" ? body.defaultCategoryId : null;
const enabledFormats = Array.isArray(body.enabledFormats)
    ? (body.enabledFormats as unknown[]).map((f) => String(f))
    : null;

if (!Number.isFinite(eventId) || !Number.isFinite(organizerIdBody) || !format) {
    return NextResponse.json({ ok: false, error: "MISSING_FIELDS" }, { status: 400 });
}

const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: organizerIdBody,
    roles: allowedRoles,
});
if (!organizer || organizer.id !== organizerIdBody) {
    return NextResponse.json({ ok: false, error: "NO_ORGANIZER" }, { status: 403 });
}

if (!["TODOS CONTRA TODOS", "QUADRO ELIMINATORIO"].includes(format)) {
    return NextResponse.json({ ok: false, error: "FORMAT_NOT_SUPPORTED" }, { status: 400 });
}

try {
    const config = await prisma.padelTournamentConfig.upsert({
        where: { eventId },
        create: {
            eventId,
            organizerId: organizerIdBody,
            format,
            numberOfCourts: Math.max(1, numberOfCourts || 1),
            ruleSetId: ruleSetId || undefined,
            defaultCategoryId: defaultCategoryId || undefined,
            enabledFormats: enabledFormats ?? undefined,
        },
        update: {
            format,
            numberOfCourts: Math.max(1, numberOfCourts || 1),
            ruleSetId: ruleSetId || undefined,
            defaultCategoryId: defaultCategoryId || undefined,
            enabledFormats: enabledFormats ?? undefined,
        },
    });
    return NextResponse.json({ ok: true, config }, { status: 200 });
} catch (err) {
    console.error("[padel/tournaments/config][POST]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/payments/intent/route.ts

```

// app/api/payments/intent/route.ts
export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@lib/prisma";
import crypto from "crypto";
import { createSupabaseServer } from "@lib/supabaseServer";
import { stripe } from "@lib/stripeClient";
import { getPlatformFees } from "@lib/platformSettings";
import { shouldNotify, createNotification } from "@lib/notifications";
import { NotificationType, PaymentEventSource, type FeeMode } from "@prisma/client";
import { paymentScenarioSchema, type PaymentScenario } from "@lib/paymentScenario";
import { parsePhoneNumberFromString } from "libphonenumber-js/min";
import { computePromoDiscountCents } from "@lib/promoMath";
import { computePricing } from "@lib/pricing";
import {
    checkoutMetadataSchema,
    createPurchaseId,
    normalizeItemsForMetadata,
}
```

```

} from "@lib/checkoutSchemas";
import { resolveOwner } from "@lib/ownership/resolveOwner";
import { enqueueOperation } from "@lib/operations/enqueue";

const FREE_PLACEHOLDER_INTENT_ID = "FREE_CHECKOUT";

type CheckoutItem = {
  ticketId: string | number;
  quantity: number;
};

type Guest = {
  name?: string;
  email?: string;
  phone?: string | null;
};

function isValidEmail(email: string) {
  return /^[^@\s]+@[^\s]+\.[^\s]+$/ .test(email);
}

type Body = {
  slug?: string;
  items?: CheckoutItem[];
  contact?: string;
  guest?: Guest;
  promoCode?: string | null;
  paymentScenario?: string | null;
  purchaseId?: string | null;
  idempotencyKey?: string | null;
  intentFingerprint?: string | null;
  pairingId?: number | null;
  slotId?: number | null;
  resaleId?: string | null;
  ticketId?: string | number | null;
  total?: number | null;
};

type IntentStatus =
  | "PENDING"
  | "PROCESSING"
  | "REQUIRES_ACTION"
  | "PAID"
  | "FAILED";

type NextAction = "NONE" | "PAY_NOW" | "CONFIRM_GUARANTEE" | "CONTACT_SUPPORT" | "LOGIN" | "CONNECT_STRIPE";

function intentError(
  code: string,
  message: string,
  opts?: {
    httpStatus?: number;
    status?: IntentStatus;
    nextAction?: NextAction;
    retryable?: boolean;
    extra?: Record<string, unknown>;
  },
) {
  return NextResponse.json(
    {
      ok: false,
      code,
      error: message,
      status: opts?.status ?? "FAILED",
      nextAction: opts?.nextAction ?? "NONE",
      retryable: opts?.retryable ?? false,
      ... (opts?.extra ?? {}),
    },
    { status: opts?.httpStatus ?? 400 },
  );
}

function normalizePhone(phone: string | null | undefined, defaultCountry: string = "PT") {
  if (!phone) return null;
  const cleaned = phone.trim();
  if (!cleaned) return null;

  const parsed = parsePhoneNumberFromString(cleaned, defaultCountry);
}

```

```

if (parsed && parsed.isPossible() && parsed.isValid()) {
  return parsed.number; // E.164
}

// fallback: regex simples para PT
const regexPT = /^(?:\+351)?9[1236]\d{7}$/;
if (regexPT.test(cleaned)) {
  const digits = cleaned.replace(/[^d]/g, "");
  return digits.startsWith("351") ? `+$ {digits}` : `+351 ${digits}`;
}

return null;
}

async function upsertPadelPlayerProfile(params: {
  organizerId: number;
  fullName: string;
  email?: string | null;
  phone?: string | null;
  gender?: string | null;
  level?: string | null;
}) {
  const { organizerId, fullName, email, phone, gender, level } = params;
  if (!fullName.trim()) return;
  const emailClean = email?.trim().toLowerCase() || null;
  const phoneClean = phone?.trim() || null;

  try {
    if (emailClean) {
      await prisma.padelPlayerProfile.upsert({
        where: { organizerId_email: { organizerId, email: emailClean } },
        update: {
          fullName,
          phone: phoneClean ?? undefined,
          gender: gender ?? undefined,
          level: level ?? undefined,
        },
        create: {
          organizerId,
          fullName,
          email: emailClean,
          phone: phoneClean ?? undefined,
          gender: gender ?? undefined,
          level: level ?? undefined,
        },
      });
    } else {
      await prisma.padelPlayerProfile.create({
        data: {
          organizerId,
          fullName,
          phone: phoneClean ?? undefined,
          gender: gender ?? undefined,
          level: level ?? undefined,
        },
      });
    }
  } catch (err) {
    console.warn("[padel] upsertPadelPlayerProfile falhou (ignorado)", err);
  }
}

export async function POST(req: NextRequest) {
  try {
    const body = (await req.json().catch(() => null)) as Body | null;

    if (!body || !body.slug || !Array.isArray(body.items) || body.items.length === 0) {
      return intentError("INVALID_INPUT", "Dados inválidos.", { httpStatus: 400, status: "FAILED", nextAction: "NONE", retryable: false });
    }

    const { slug, items, contact, guest, promoCode: rawPromo, paymentScenario: rawScenario, idempotencyKey: bodyIdemKey } = body;
    const promoCodeInput = typeof rawPromo === "string" ? rawPromo.trim() : "";
    const idempotencyKeyHeader = req.headers.get("Idempotency-Key");
    const idempotencyKey = (bodyIdemKey || idempotencyKeyHeader || "").trim() || null;

    // purchaseId/intento fingerprint opcionais vindos do frontend (para retries/idempotência estável)
  }
}

```

```

const purchaseIdFromBody =
  typeof (body as { purchaseId?: unknown })?.purchaseId === "string"
    ? (body as { purchaseId?: string }).purchaseId!.trim()
    : "";

const intentFingerprintFromBody =
  typeof (body as { intentFingerprint?: unknown })?.intentFingerprint === "string"
    ? (body as { intentFingerprint?: string }).intentFingerprint!.trim()
    : "";

// Nota: o purchaseId final será decidido mais abaixo (de forma determinística quando possível)
// depois de conhecemos descontos/pricing.

// Validar que o evento existe (fetch raw para evitar issues com enum legacy "ADDED")
// Autenticação do utilizador
const supabase = await createSupabaseServer();
const { data: userData } = await supabase.auth.getUser();
const userId = userData?.user?.id ?? null;

// Regra PADL: pagar só a tua parte (capítulo / split) exige conta
const parsedScenarioEarly = paymentScenarioSchema.safeParse(
  typeof rawScenario === "string" ? rawScenario.toUpperCase() : rawScenario,
);
const requestedScenarioEarly: PaymentScenario | null = parsedScenarioEarly.success
  ? parsedScenarioEarly.data
  : null;

if (requestedScenarioEarly === "GROUP_SPLIT" && !userId) {
  return intentError("AUTH_REQUIRED_FOR_GROUP_SPLIT", "Este modo de pagamento requer sessão iniciada.", {
    httpStatus: 401,
    status: "FAILED",
    nextAction: "LOGIN",
    retryable: false,
  });
}

const guestEmailRaw = guest?.email?.trim() ?? "";
const guestName = guest?.name?.trim() ?? "";
const guestPhoneRaw = guest?.phone?.trim() ?? "";
const guestPhone = guestPhoneRaw ? normalizePhone(guestPhoneRaw) : "";
const guestEmail = guestEmailRaw && isValidEmail(guestEmailRaw) ? guestEmailRaw : "";

if (!userId) {
  if (!guestEmail || !guestName) {
    return intentError("AUTH_OR_GUEST_REQUIRED", "Precisas de iniciar sessão ou preencher nome e email para convidado.", {
      httpStatus: 400
    });
  }
  if (!isValidEmail(guestEmailRaw)) {
    return intentError("INVALID_GUEST_EMAIL", "Email inválido para checkout como convidado.", { httpStatus: 400 });
  }
  if (guestPhoneRaw && !guestPhone) {
    return intentError("INVALID_GUEST_PHONE", "Telemóvel inválido. Usa formato PT: 9XXXXXXX ou +351XXXXXXX.", {
      httpStatus: 400
    });
  }
}

const ownerResolved = await resolveOwner({ sessionUserId: userId, guestEmail });
const ownerForMetadata =
  ownerResolved.ownerUserId || ownerResolved.ownerIdentityId
    ? {
        ownerId: ownerResolved.ownerUserId ?? undefined,
        ownerIdentityId: ownerResolved.ownerIdentityId ?? undefined,
        emailNormalized: ownerResolved.emailNormalized ?? undefined,
      }
    : userId || guestEmail
    ? {
        userId: userId ?? undefined,
        guestEmail: guestEmail || undefined,
        guestName: guestName || undefined,
        guestPhone: guestPhone || undefined,
      }
    : undefined;

const eventRows = await prisma.$queryRaw<
  {
    id: number;
    slug: string;
    title: string;
  }
>

```

```

    status: string;
    type: string;
    is_deleted: boolean;
    is_test: boolean;
    ends_at: Date | null;
    fee_mode: string | null;
    fee_mode_override: string | null;
    organizer_id: number | null;
    org_type: string | null;
    org_stripe_account_id: string | null;
    org_stripe_charges_enabled: boolean | null;
    org_stripe_payouts_enabled: boolean | null;
    org_fee_mode: string | null;
    org_platform_fee_bps: number | null;
    org_platform_fee_fixed_cents: number | null;
    platform_fee_bps_override: number | null;
    platform_fee_fixed_cents_override: number | null;
    payout_mode: string | null;
}[]
>`  

SELECT
  e.id,
  e.slug,
  e.title,
  e.status,
  e.type,
  e.is_deleted,
  e.is_test,
  e.ends_at,
  e.fee_mode,
  e.fee_mode_override,
  e.organizer_id,
  o.org_type AS org_type,
  o.stripe_account_id AS org_stripe_account_id,
  o.stripe_charges_enabled AS org_stripe_charges_enabled,
  o.stripe_payouts_enabled AS org_stripe_payouts_enabled,
  o.fee_mode AS org_fee_mode,
  o.platform_fee_bps AS org_platform_fee_bps,
  o.platform_fee_fixed_cents AS org_platform_fee_fixed_cents,
  e.platform_fee_bps_override,
  e.platform_fee_fixed_cents_override,
  e.payout_mode
FROM app_v3.events e
LEFT JOIN app_v3.organizers o ON o.id = e.organizer_id
WHERE e.slug = ${slug}
LIMIT 1;
`;  

  

const event = eventRows[0];
const eventOrganizerId = event?.organizer_id ?? null;  

  

if (!event) {
  return intentError("EVENT_NOT_FOUND", "Evento não encontrado.", { httpStatus: 404 });
}  

const profile = userId
  ? await prisma.profile.findUnique({ where: { id: userId } })
  : null;
const isAdmin = Array.isArray(profile?.roles) ? profile.roles.includes("admin") : false;
if (event.is_test && !isAdmin) {
  return intentError("EVENT_NOT_AVAILABLE", "Evento não disponível.", { httpStatus: 404 });
}  

  

// Atualizar contato no perfil se fornecido (normalizado)
if (userId && contact && contact.trim()) {
  const normalizedContact = normalizePhone(contact.trim());
  if (normalizedContact) {
    await prisma.profile.update({
      where: { id: userId },
      data: { contactPhone: normalizedContact },
    });
  }
}  

  

if (event.is_deleted || event.status !== "PUBLISHED" || event.type !== "ORGANIZER_EVENT") {
  return intentError("EVENT_CLOSED", "Evento indisponível para compra.", { httpStatus: 400 });
}  

  

if (event.ends_at && event.ends_at < new Date()) {  


```

```

    return intentError("EVENT_ENDED", "Vendas encerradas: evento já terminou.", { httpStatus: 400 });
}

const padelConfig = await prisma.padelTournamentConfig.findUnique({
  where: { eventId: event.id },
  select: { organizerId: true },
});

const ticketTypeIds = Array.from(
  new Set(
    items
      .map((i) => Number(i.ticketId))
      .filter((v) => Number.isFinite(v) && v > 0),
  ),
);

if (ticketTypeIds.length === 0) {
  return intentError("INVALID_TICKETS", "IDs de bilhete inválidos.", { httpStatus: 400 });
}

const ticketTypes = await prisma.ticketType.findMany({
  where: {
    id: { in: ticketTypeIds },
    eventId: event.id,
    status: "ON_SALE",
  },
  select: {
    id: true,
    name: true,
    price: true,
    currency: true,
    totalQuantity: true,
    soldQuantity: true,
  },
});
}

if (ticketTypes.length !== ticketTypeIds.length) {
  return intentError("TICKET_NOT_FOUND", "Um dos bilhetes não foi encontrado ou não pertence a este evento.", { httpStatus: 400 });
}

// Reservas ativas (excluindo as do próprio utilizador) contam para stock
const now = new Date();
const activeReservations = await prisma.ticketReservation.findMany({
  where: {
    ticketTypeId: { in: ticketTypeIds },
    status: "ACTIVE",
    expiresAt: { gt: now },
  },
  select: {
    ticketTypeId: true,
    quantity: true,
    userId: true,
  },
});

const reservedByType = activeReservations.reduce<Record<number, number>>(
  (acc, r) => {
    // Ignorar reservas do próprio user para não bloquear a compra dele
    if (r.userId && r.userId === userId) return acc;
    acc[r.ticketTypeId] = (acc[r.ticketTypeId] ?? 0) + r.quantity;
    return acc;
  },
  {},
);

let amountInCents = 0;
let totalQuantity = 0;
let currency: string | null = null;
const ticketTypeMap = new Map<number, (typeof ticketTypes)[number]>(ticketTypes.map((t) => [t.id, t]));
const lines: {
  ticketTypeId: number;
  name: string;
  quantity: number;
  unitPriceCents: number;
  currency: string;
  lineTotalCents: number;
}[] = [];

```

```

for (const item of items) {
  const ticketTypeId = Number(item.ticketId);
  if (!Number.isFinite(ticketTypeId)) {
    return intentError("INVALID_TICKET_ID", "ID de bilhete inválido.", { httpStatus: 400 });
  }

  const ticketType = ticketTypeMap.get(ticketTypeId);
  if (!ticketType) {
    return intentError("TICKET_NOT_FOUND", "Um dos bilhetes não foi encontrado ou não pertence a este evento.", { httpStatus: 400 });
  }

  const qty = Number(item.quantity ?? 0);
  if (!Number.isInteger(qty) || qty < 1) {
    return intentError("INVALID_QUANTITY", "Quantidade inválida.", { httpStatus: 400 });
  }

  // limite agora é apenas o stock disponível; o cap de 6 foi removido

  // Validação de stock (incluindo reservas ativas de outros)
  if (
    ticketType.totalQuantity !== null &&
    ticketType.totalQuantity !== undefined
  ) {
    const reserved = reservedByType[ticketTypeId] ?? 0;
    const remaining = ticketType.totalQuantity - ticketType.soldQuantity - reserved;
    if (remaining < qty) {
      return intentError("INSUFFICIENT_STOCK", "Stock insuficiente para um dos bilhetes.", {
        httpStatus: 409,
        status: "FAILED",
        retryable: false,
        nextAction: "NONE",
      });
    }
  }
}

const priceCents = Number(ticketType.price);
if (!Number.isFinite(priceCents) || priceCents <= 0) {
  return intentError("INVALID_PRICE_SERVER", "Preço inválido no servidor.", { httpStatus: 500 });
}

const ticketCurrency = (ticketType.currency || "EUR").toLowerCase();
if (!currency) {
  currency = ticketCurrency;
} else if (currency !== ticketCurrency) {
  return intentError("CURRENCY_MISMATCH", "Não é possível misturar moedas diferentes no mesmo checkout.", { httpStatus: 400 });
}
}

const lineTotal = priceCents * qty;
lines.push({
  ticketTypeId,
  name: ticketType.name ?? "Bilhete",
  quantity: qty,
  unitPriceCents: priceCents,
  currency: ticketCurrency.toUpperCase(),
  lineTotalCents: lineTotal,
});

amountInCents += lineTotal;
totalQuantity += qty;
}

if (!currency) {
  return intentError("CURRENCY_UNDETERMINED", "Moeda não determinada para o checkout.", { httpStatus: 400 });
}

const clientExpectedTotalCents =
  body && typeof (body as Record<string, unknown>).total === "number"
    ? Math.round(Math.max(0, (body as Record<string, number>).total) * 100)
    : null;

// Montante base do carrinho (antes de descontos)
const preDiscountAmountCents = Math.max(0, amountInCents);

const promoRepo = (prisma as unknown as {
  promoCode?: {

```

```

        findFirst: typeof prisma.promoCode.findFirst;
        findMany: typeof prisma.promoCode.findMany;
    };
}).promoCode;
// Promo code (apenas validação; redemptions serão registados no webhook após sucesso)
let discountCents = 0;
let promoCodeId: number | null = null;
const nowDate = new Date();

const validatePromo = async (codeFilter: { id?: number; code?: string }) => {
    if (!promoRepo) {
        throw new Error("PROMO_UNAVAILABLE");
    }
    const promo = await prisma.promoCode.findFirst({
        where: {
            ...codeFilter.id ? { id: codeFilter.id } : {},
            ...codeFilter.code
                ? { code: { equals: codeFilter.code, mode: "insensitive" } }
                : {},
            active: true,
            OR: [{ eventId: event.id }, { eventId: null }],
        },
    });
    if (!promo) {
        throw new Error("PROMO_INVALID");
    }

    // Scope ao evento já garantido no filtro eventId; não há organizerId na tabela nova.
    if (promo.eventId && promo.eventId !== event.id) {
        throw new Error("PROMO_SCOPE");
    }
    if (promo.validFrom && promo.validFrom > nowDate) {
        throw new Error("PROMO_NOT_ACTIVE");
    }
    if (promo.validUntil && promo.validUntil < nowDate) {
        throw new Error("PROMO_EXPIRED");
    }
    if (
        promo.minQuantity !== null &&
        promo.minQuantity !== undefined &&
        totalQuantity < promo.minQuantity
    ) {
        throw new Error("PROMO_MIN_QTY");
    }
    const minCartCents =
        promo.minTotalCents ?? (promo as { minCartValueCents?: number | null }).minCartValueCents ?? null;
    if (minCartCents === null && minCartCents === undefined && preDiscountAmountCents < minCartCents) {
        throw new Error("PROMO_MIN_TOTAL");
    }

    // Contagem de redemptions passadas
    const totalUses = await prisma.promoRedemption.count({
        where: { promoCodeId: promo.id },
    });
    if (promo.maxUses !== null && promo.maxUses !== undefined && totalUses >= promo.maxUses) {
        throw new Error("PROMO_MAXUSES");
    }

    if (promo.perUserLimit !== null && promo.perUserLimit !== undefined) {
        if (userId) {
            const userUses = await prisma.promoRedemption.count({
                where: { promoCodeId: promo.id, userId },
            });
            if (userUses >= promo.perUserLimit) {
                throw new Error("PROMO_USER_LIMIT");
            }
        } else if (guestEmail) {
            const guestUses = await prisma.promoRedemption.count({
                where: { promoCodeId: promo.id, guestEmail: { equals: guestEmail, mode: "insensitive" } },
            });
            if (guestUses >= promo.perUserLimit) {
                throw new Error("PROMO_USER_LIMIT");
            }
        }
    }
}

discountCents = computePromoDiscountCents({

```

```

    promo: {
      type: promo.type as "PERCENTAGE" | "FIXED",
      value: promo.value,
      minQuantity: promo.minQuantity,
      minTotalCents: minCartCents ?? promo.minTotalCents,
    },
    totalQuantity,
    amountInCents: preDiscountAmountCents,
  });
  promoCodeId = promo.id;
};

if (promoCodeInput) {
  try {
    await validatePromo({ code: promoCodeInput });
    console.info("[analytics] promo_applied", {
      code: promoCodeInput,
      eventId: event.id,
      userId,
    });
  } catch (err) {
    const msg = (err as Error).message;
    const map: Record<string, string> = {
      PROMO_INVALID: "Código promocional inválido ou inativo.",
      PROMO_NOT_ACTIVE: "Código ainda não está ativo.",
      PROMO_EXPIRED: "Código expirado.",
      PROMO_MAXUSES: "Código já atingiu o limite de utilizações.",
      PROMO_USER_LIMIT: "Já utilizaste este código o número máximo de vezes.",
      PROMO_MIN_QTY: "Quantidade insuficiente para aplicar este código.",
      PROMO_MIN_TOTAL: "Valor mínimo não atingido para aplicar este código.",
      PROMO_SCOPE: "Este código não é válido para este evento/organização.",
      PROMO_UNAVAILABLE: "Descontos temporariamente indisponíveis.",
    };
    return intentError(msg || "PROMO_INVALID", map[msg] ?? "Código promocional inválido.", {
      httpStatus: 400,
      status: "FAILED",
      nextAction: "NONE",
      retryable: msg === "PROMO_UNAVAILABLE",
    });
  }
} else if (promoRepo) {
  // Auto-apply: escolher o melhor desconto elegível (event/global, autoApply=true)
  const autoPromos = await promoRepo.findMany({
    where: {
      autoApply: true,
      active: true,
      OR: [{ eventId: event.id }, { eventId: null }],
      NOT: [
        {
          validFrom: { gt: nowDate },
        },
      ],
    },
    // determinístico: evita escolher promos diferentes em chamadas repetidas
    orderBy: [{ id: "asc" }],
  });

  let best: { promoId: number; discount: number } | null = null;

  for (const promo of autoPromos) {
    discountCents = 0;
    promoCodeId = null;
    try {
      await validatePromo({ id: promo.id });
      const d = discountCents;
      // determinístico: em empate, escolhe o menor id
      if (!best || d > best.discount || (d === best.discount && promo.id < best.promoId)) {
        best = { promoId: promo.id, discount: d };
      }
    } catch {
      // ignora promo não elegível
    }
  }

  if (best) {
    promoCodeId = best.promoId;
    discountCents = best.discount;
  }
}

```

```

}

// Clamp final discount (não deixa passar do total)
discountCents = Math.max(0, Math.min(discountCents, preDiscountAmountCents));
const amountAfterDiscountCents = preDiscountAmountCents - discountCents;

const { feeBps: defaultFeeBps, feeFixedCents: defaultFeeFixed } = await getPlatformFees();

// Org da plataforma? (org_type = PLATFORM → não cobra application fee, usa conta da plataforma)
const isPlatformOrg = (event.org_type || "").toString().toUpperCase() === "PLATFORM";

const pricing = computePricing(preDiscountAmountCents, discountCents, {
  eventFeeModeOverride: (event.fee_mode_override as FeeMode | null) ?? undefined,
  eventFeeMode: (event.fee_mode as FeeMode | null) ?? undefined,
  organizerFeeMode: (event.org_fee_mode as FeeMode | null) ?? undefined,
  platformDefaultFeeMode: "ADDED",
  eventPlatformFeeBpsOverride: event.platform_fee_bps_override,
  eventPlatformFeeFixedCentsOverride: event.platform_fee_fixed_cents_override,
  organizerPlatformFeeBps: event.org_platform_fee_bps,
  organizerPlatformFeeFixedCents: event.org_platform_fee_fixed_cents,
  platformDefaultFeeBps: defaultFeeBps,
  platformDefaultFeeFixedCents: defaultFeeFixed,
  isPlatformOrg,
});
const platformFeeCents = pricing.platformFeeCents;

// Stripe account rules
let stripeAccountId = event.org_stripe_account_id ?? null;
const payoutModeRaw = (event.payout_mode || "ORGANIZER").toString().toUpperCase();
const organizerStripeReady = Boolean(event.org_stripe_charges_enabled && event.org_stripe_payouts_enabled);

// Plataforma ORYA: usa conta da plataforma, não exige Connect
const requiresOrganizerStripe = !isPlatformOrg && payoutModeRaw !== "PLATFORM";

if (!requiresOrganizerStripe) {
  stripeAccountId = null;
} else {
  // Organizadores externos: exigem Connect pronto
  if (!stripeAccountId || !organizerStripeReady) {
    return intentError("ORGANIZER_STRIPE_NOT_CONNECTED", "Pagamentos estão desativados porque o organizador ainda não ligou a Stripe.", {
      httpStatus: 409,
      status: "FAILED",
      nextAction: "CONNECT_STRIPE",
      retryable: false,
    });
  }
}

const isPartnerEvent = Boolean(stripeAccountId);

const totalAmountInCents = pricing.totalCents;

// Validação do total do cliente (tolerante): alguns FE enviam subtotal, outros enviam total.
if (clientExpectedTotalCents !== null) {
  const matchesAnyExpected =
    clientExpectedTotalCents === preDiscountAmountCents ||
    clientExpectedTotalCents === amountAfterDiscountCents ||
    clientExpectedTotalCents === totalAmountInCents;

  if (!matchesAnyExpected) {
    return intentError("PRICE_CHANGED", "Os preços foram atualizados. Revê a seleção e tenta novamente.", {
      httpStatus: 409,
      status: "FAILED",
      retryable: true,
      nextAction: "NONE",
      extra: {
        expected: {
          subtotalCents: preDiscountAmountCents,
          afterDiscountCents: amountAfterDiscountCents,
          totalCents: totalAmountInCents,
          currency: currency.toUpperCase(),
        },
      },
    });
  }
}

```

```

if (totalAmountInCents < 0 || platformFeeCents > Math.max(totalAmountInCents, 0)) {
  return intentError("INVALID_TOTAL", "Montante total inválido para este checkout.", { httpStatus: 400 });
}

const normalizedItems = normalizeItemsForMetadata(
  lines.map((line) => ({
    ticketTypeId: line.ticketTypeId,
    quantity: line.quantity,
    unitPriceCents: line.unitPriceCents,
    currency: line.currency,
  })),
);

const parsedScenario = paymentScenarioSchema.safeParse(
  typeof rawScenario === "string" ? rawScenario.toUpperCase() : rawScenario,
);
const requestedScenario: PaymentScenario | null = parsedScenario.success ? parsedScenario.data : null;

const paymentScenario: PaymentScenario =
  totalAmountInCents === 0
    ? "FREE_CHECKOUT"
    : requestedScenario
      ? requestedScenario
      : "SINGLE";

const scenarioAdjusted: PaymentScenario = (() => {
  if (paymentScenario === "GROUP_FULL") return "GROUP_FULL";
  if (paymentScenario === "GROUP_SPLIT") return "GROUP_SPLIT";
  if (paymentScenario === "RESALE" || paymentScenario === "SUBSCRIPTION") return paymentScenario;
  if (paymentScenario === "FREE_CHECKOUT") return "FREE_CHECKOUT";
  return "SINGLE";
})();

// Padel pricing guard: qty coerente com scenario (anti preço "por dupla")
if (scenarioAdjusted === "GROUP_FULL") {
  const invalid = normalizedItems.some((i) => i.quantity !== 2);
  if (invalid) {
    return intentError("INVALID_PRICING_MODEL", "Modelo de preço inválido para GROUP_FULL (espera qty=2 por item).", {
      httpStatus: 400,
      status: "FAILED",
      retryable: false,
    });
  }
}
if (scenarioAdjusted === "GROUP_SPLIT") {
  const invalid = normalizedItems.some((i) => i.quantity !== 1);
  if (invalid) {
    return intentError("INVALID_PRICING_MODEL", "Modelo de preço inválido para GROUP_SPLIT (espera qty=1 por item).", {
      httpStatus: 400,
      status: "FAILED",
      retryable: false,
    });
  }
}

// purchaseId determinístico quando o frontend não enviar: evita duplicar PaymentIntents em retries
const identityKey = userId
  ? `u:${userId}`
  : guestEmail
    ? `g:${guestEmail.toLowerCase()}`
    : "anon";

const FINGERPRINT_VERSION = "v2"; // bump to avoid clashes com intents antigos

const intentFingerprint = crypto
  .createHash("sha256")
  .update(
    JSON.stringify({
      version: FINGERPRINT_VERSION,
      eventId: event.id,
      slug: event.slug,
      items: normalizedItems,
      scenario: scenarioAdjusted,
      identity: identityKey,
      promoCodeId,
      promoCodeRaw: promoCodeInput ? promoCodeInput.toLowerCase() : null,
      discountCents,
      totalCents: totalAmountInCents,
    })
  )
  .digest("hex");

```

```

        currency: currency.toLowerCase(),
        platformFeeCents,
      }),
    )
    .digest("hex")
    .slice(0, 32);

const computedPurchaseId = `pur_${intentFingerprint}`;

// Se o frontend enviou fingerprint/purchaseId, podem estar desatualizados.
// Em vez de bloquear (409 + loop no FE), registamos e seguimos sempre com a SSOT do servidor.
if (intentFingerprintFromBody && intentFingerprintFromBody !== intentFingerprint) {
  console.warn("[payments/intent] client intentFingerprint desatualizado", {
    provided: intentFingerprintFromBody,
    expected: intentFingerprint,
    purchaseId: computedPurchaseId,
  });
}

if (purchaseIdFromBody && purchaseIdFromBody !== computedPurchaseId) {
  console.warn("[payments/intent] client purchaseId desatualizado", {
    provided: purchaseIdFromBody,
    expected: computedPurchaseId,
    intentFingerprint,
  });
}

const purchaseId = computedPurchaseId;

// Dedupe determinístico por carrinho: evita loops 409 quando o FE reutiliza uma idempotencyKey inválida.
// Mantemos a key enviada pelo FE apenas para telemetria/resposta.
const clientIdempotencyKey = idempotencyKey;
const effectiveDedupeKey = purchaseId;

// Idempotência API: se houver payment_event com dedupeKey=idempotencyKey, devolve mesma resposta
// (mantemos este caminho apenas quando o cliente envia idempotencyKey explicitamente)
if (clientIdempotencyKey) {
  const existing = await prisma.paymentEvent.findFirst({
    where: { dedupeKey: clientIdempotencyKey },
    select: {
      stripePaymentIntentId: true,
      purchaseId: true,
      amountCents: true,
    },
  });

  if (existing?.stripePaymentIntentId) {
    const existingPiId = existing.stripePaymentIntentId;

    // Se for um PaymentIntent real da Stripe, devolvemos o mesmo clientSecret
    if (existingPiId.startsWith("pi_")) {
      try {
        const pi = await stripe.paymentIntents.retrieve(existingPiId);
        // Safety: the same idempotency key must not be reused with a different payload/amount.
        if (typeof pi.amount === "number" && pi.amount !== Math.max(0, totalAmountInCents)) {
          return intentError("IDEMPOTENCY_KEY_PAYLOAD_MISMATCH", "Chave de idempotência reutilizada com um carrinho diferente.", {
            httpStatus: 409,
            retryable: false,
          });
        }
        return NextResponse.json({
          ok: true,
          reused: true,
          clientSecret: pi.client_secret,
          amount: typeof pi.amount === "number" ? pi.amount : totalAmountInCents,
          currency: currency.toUpperCase(),
          discountCents,
          paymentIntentId: existingPiId,
          purchaseId: existing.purchaseId ?? undefined,
          paymentScenario: scenarioAdjusted,
          breakdown: {
            lines,
            subtotalCents: pricing.subtotalCents,
            feeMode: pricing.feeMode,
            platformFeeCents: pricing.platformFeeCents,
            totalCents: totalAmountInCents,
          }
        });
      } catch (error) {
        return intentError("IDEMPOTENCY_KEY_RETRIEVE_FAILED", `Erro ao recuperar o PaymentIntent ${existingPiId}: ${error.message}`);
      }
    }
  }
}

```

```

        currency: currency.toUpperCase(),
    },
    intentFingerprint,
    idempotencyKey: clientIdempotencyKey ?? effectiveDedupeKey,
),
{ status: 200 },
);
} catch (e) {
    console.warn("[payments/intent] idempotency retrieve PI falhou", e);
    // Se falhar o retrieve, seguimos o fluxo normal (Stripe idempotency abaixo ajuda a evitar duplicados)
}
}

// Caso de checkout gratuito já concluído (stripePaymentIntentId == purchaseId interno)
if ((existing.amountCents ?? 0) === 0) {
    const ticketsCreated = await prisma.ticket.count({
        where: {
            eventId: event.id,
            stripePaymentIntentId: existingPiId,
        },
    });

    return NextResponse.json(
    {
        ok: true,
        reused: true,
        freeCheckout: true,
        purchaseId: existing.purchaseId ?? undefined,
        paymentScenario: "FREE_CHECKOUT",
        ticketsCreated,
        amount: 0,
        currency: currency.toUpperCase(),
        discountCents,
        breakdown: {
            lines,
            subtotalCents: pricing.subtotalCents,
            feeMode: pricing.feeMode,
            platformFeeCents: pricing.platformFeeCents,
            totalCents: 0,
            currency: currency.toUpperCase(),
        },
        intentFingerprint,
        idempotencyKey: clientIdempotencyKey ?? effectiveDedupeKey,
    },
    { status: 200 },
);
}
}

const metadataValidation = checkoutMetadataSchema.safeParse({
    paymentScenario: scenarioAdjusted,
    purchaseId,
    items: normalizedItems,
    eventId: event.id,
    eventSlug: event.slug,
    owner: ownerForMetadata,
    pairingId: typeof body?.pairingId === "number" ? body?.pairingId : undefined,
});
}

if (!metadataValidation.success) {
    console.warn("[payments/intent] Metadata inválida", metadataValidation.error.format());
    return intentError("INVALID_METADATA", "Metadata inválida para checkout.", {
        httpStatus: 400,
        status: "FAILED",
        retryable: false,
    });
}

// -----
// CHECKOUT GRATUITO (0 €)
// -----
if (totalAmountInCents === 0) {
    if (!userId || !profile?.username?.trim()) {
        return intentError("USERNAME_REQUIRED_FOR_FREE", "Falta terminar o perfil (username) para concluir eventos gratuitos.",
{
    httpStatus: 403,
    status: "FAILED",
}

```

```

        nextAction: "LOGIN",
        retryable: false,
      });
    }

    // Idempotência anti-double click e 1 por user: se já existe bilhete para este evento, recusar
    const existingFreeTicket = await prisma.ticket.findFirst({
      where: { eventId: event.id, userId, pricePaid: 0 },
      select: { id: true },
    });
    if (existingFreeTicket) {
      return intentError("FREE_ALREADY_CLAIMED", "Já tens uma inscrição gratuita neste evento.", {
        httpStatus: 409,
        status: "FAILED",
        nextAction: "NONE",
        retryable: false,
      });
    }

    // Cooldown simples: bloquear se já fez free checkout neste evento nos últimos 10 minutos
    const tenMinutesAgo = new Date(Date.now() - 10 * 60 * 1000);
    const recentFree = await prisma.paymentEvent.findFirst({
      where: {
        eventId: event.id,
        userId,
        amountCents: 0,
        purchaseId: { not: null },
        createdAt: { gte: tenMinutesAgo },
      },
      select: { purchaseId: true },
    });
    if (recentFree) {
      return intentError("FREE_RATE_LIMIT", "Já fizeste uma inscrição gratuita há poucos minutos. Aguarda antes de tentar novamente.", {
        httpStatus: 429,
        status: "FAILED",
        nextAction: "NONE",
        retryable: true,
        extra: { purchaseId: recentFree.purchaseId },
      });
    }

    let createdTicketsCount = 0;
    const freePayload = {
      eventId: event.id,
      purchaseId,
      userId: ownerResolved.ownerUserId ?? userId,
      ownerIdentityId: ownerResolved.ownerIdentityId ?? null,
      promoCodeId: promoCodeId ? Number(promoCodeId) : null,
      subtotalCents: pricing.subtotalCents,
      discountCents,
      platformFeeCents: 0,
      feeMode: pricing.feeMode,
      currency: currency.toUpperCase(),
      lines: lines.map((l) => ({
        ticketTypeId: l.ticketTypeId,
        quantity: l.quantity,
        unitPriceCents: l.unitPriceCents,
      })),
    };

    await prisma.paymentEvent.create({
      data: {
        stripePaymentIntentId: purchaseId,
        status: "PROCESSING",
        purchaseId,
        source: PaymentEventSource.API,
        dedupeKey: effectiveDedupeKey,
        attempt: 1,
        eventId: event.id,
        userId,
        amountCents: 0,
        platformFeeCents: 0,
        stripeFeeCents: 0,
        mode: event.is_test ? "TEST" : "LIVE",
        isTest: Boolean(event.is_test),
      },
    });
  }
}

```

```

await enqueueOperation({
  operationType: "UPsert_LEDGER_FROM_PI_FREE",
  dedupeKey: purchaseId,
  correlations: { purchaseId, eventId: event.id },
  payload: freePayload,
});

if (padelConfig) {
  const fullName = userData?.user?.user_metadata?.full_name || profile?.fullName || "Jogador Padel";
  const emailToSave = userData?.user?.email || profile?.email || null;
  const phoneToSave = userData?.user?.phone || contact || profile?.contactPhone || null;
  await upsertPadelPlayerProfile({
    organizerId: padelConfig.organizerId,
    fullName,
    email: emailToSave,
    phone: phoneToSave,
  });
}

// Notificação para o organizer (se existir) - respeita prefs
if (eventOrganizerId) {
  try {
    const ownerMembers = await prisma.organizerMember.findMany({
      where: { organizerId: eventOrganizerId, role: { in: ["OWNER", "CO_OWNER", "ADMIN"] } },
      select: { userId: true },
    });
    const uniqOwners = Array.from(new Set(ownerMembers.map((m) => m.userId)));
    await Promise.all(
      uniqOwners.map((uid) =>
        (async () => {
          if (!(await shouldNotify(uid, NotificationType.EVENT_SALE))) return;
          await createNotification({
            userId: uid,
            type: NotificationType.EVENT_SALE,
            title: "Nova reserva gratuita",
            body: `Recebaste uma reserva para ${event.title}.`,
            ctaUrl: `/organizador?tab=sales&eventId=${event.id}`,
            ctaLabel: "Ver vendas",
            payload: { eventId: event.id, title: event.title },
          });
        })(),
      ),
    );
  } catch (err) {
    console.warn("[notification][free_checkout] falhou", err);
  }
}

return NextResponse.json({
  ok: true,
  code: "OK",
  status: "PROCESSING",
  nextAction: "NONE",
  retryable: true,
  freeCheckout: true,
  isFreeCheckout: true,
  purchaseId,
  paymentIntentId: FREE_PLACEHOLDER_INTENT_ID,
  paymentScenario,
  ticketsCreated: createdTicketsCount,
  amount: 0,
  currency: currency.toUpperCase(),
  discountCents,
  breakdown: {
    lines,
    subtotalCents: pricing.subtotalCents,
    feeMode: pricing.feeMode,
    platformFeeCents: pricing.platformFeeCents,
    totalCents: 0,
    currency: currency.toUpperCase(),
  },
  intentFingerprint,
  idempotencyKey: clientIdIdempotencyKey ?? effectiveDedupeKey,
});
}

const metadata: Record<string, string> = {
  eventId: String(event.id),

```

```

    eventSlug: String(event.slug),
    purchaseId,
    intentFingerprint,
    items: JSON.stringify(normalizedItems),
    paymentScenario: scenarioAdjusted,
    baseAmountCents: String(preDiscountAmountCents),
    discountCents: String(discountCents),
    platformFeeMode: pricing.feeMode,
    platformFeeBps: String(pricing.feeBpsApplied),
    platformFeeFixedCents: String(pricing.feeFixedApplied),
    platformFeeCents: String(pricing.platformFeeCents),
    contact: contact?.trim() ?? "",
    stripeAccountId: stripeAccountId ?? "orya",
    guestName: guestName ?? "",
    guestEmail: guestEmail ?? "",
    guestPhone: guestPhone ?? "",
    mode: guestName && guestEmail && !userId ? "GUEST" : "USER",
    promoCode: promoCodeId ? String(promoCodeId) : "",
    promoCodeRaw: promoCodeInput,
    totalQuantity: String(totalQuantity),
    breakdown: JSON.stringify({
      lines,
      subtotalCents: pricing.subtotalCents,
      feeMode: pricing.feeMode,
      platformFeeCents: pricing.platformFeeCents,
      totalCents: pricing.totalCents,
      currency: currency.toUpperCase(),
    }),
  );
}

if (userId) {
  metadata.userId = userId;
}
// Metadata idempotencyKey deve ser estável (purchaseId) para não disparar idempotency_error na Stripe.
metadata.idempotencyKey = effectiveDedupeKey;
if (clientIdempotencyKey) {
  metadata.clientIdempotencyKey = clientIdempotencyKey;
}
if (ownerResolved.ownerUserId) metadata.ownerUserId = ownerResolved.ownerUserId;
if (ownerResolved.ownerIdentityId) metadata.ownerIdentityId = ownerResolved.ownerIdentityId;
if (ownerResolved.emailNormalized) metadata.emailNormalized = ownerResolved.emailNormalized;
if (typeof body?.pairingId === "number") metadata.pairingId = String(body.pairingId);
if (typeof body?.slotId === "number") metadata.slotId = String(body.slotId);
if (typeof body?.resaleId === "string") metadata.resaleId = body.resaleId;
if (typeof body?.ticketId === "string" || typeof body?.ticketId === "number") metadata.ticketId = String(body.ticketId);
if (paymentScenario === "RESALE" && userId) metadata.buyerUserId = userId;

const allowedPaymentMethods = ["card", "link", "mb_way"] as const;

const intentParams: Parameters<typeof stripe.paymentIntents.create>[0] = {
  amount: Math.max(0, totalAmountInCents),
  currency,
  payment_method_types: [...allowedPaymentMethods],
  metadata,
};

if (!userId && guestEmail) {
  intentParams.receipt_email = guestEmail;
}

if (isPartnerEvent && stripeAccountId) {
  intentParams.transfer_data = {
    destination: stripeAccountId,
  };
  // Apenas aplica application_fee se não for organizer admin
  if (!isPlatformOrg) {
    intentParams.application_fee_amount = platformFeeCents;
  }
}

const stripeIdempotencyKey = purchaseId.slice(0, 200);

const createPi = async (idemKey?: string) =>
  stripe.paymentIntents.create(intentParams, idemKey ? { idempotencyKey: idemKey } : undefined);

let paymentIntent;
try {
  paymentIntent = await createPi(stripeIdempotencyKey);

```

```

} catch (e: unknown) {
  const anyErr = e as { type?: string; code?: string; message?: string };
  const isIdem =
    anyErr?.type === "StripeIdempotencyError" ||
    anyErr?.code === "idempotency_error" ||
    (typeof anyErr?.message === "string" && anyErr.message.toLowerCase().includes("idempot"));

  if (isIdem) {
    console.warn("[payments/intent] Stripe idempotency mismatch, a recalcular com nova key", {
      purchaseId,
      intentFingerprint,
    });
    // Tenta apenas uma vez com uma chave derivada (mantém o mesmo purchaseId para SSOT)
    const retryKey = `${stripeIdempotencyKey}:v2`;
    paymentIntent = await createPi(retryKey);
  } else {
    throw e;
  }
}

if (padelConfig) {
  const fullName = userData?.user?.user_metadata?.full_name || guestName || "Jogador Padel";
  const emailToSave = userData?.user?.email || guestEmail || null;
  const phoneToSave = userData?.user?.phone || contact || guestPhone || null;
  await upsertPadelPlayerProfile({
    organizerId: padelConfig.organizerId,
    fullName,
    email: emailToSave,
    phone: phoneToSave,
  });
}

if (!paymentIntent.client_secret) {
  return NextResponse.json({
    ok: false,
    error: "Não foi possível preparar o pagamento (client_secret em falta).",
    code: "MISSING_CLIENT_SECRET",
    retryable: true,
  },
  { status: 500 },
);
}

return NextResponse.json({
  ok: true,
  code: "OK",
  status: "REQUIRES_ACTION",
  nextAction: "PAY_NOW",
  retryable: true,
  clientSecret: paymentIntent.client_secret,
  amount: totalAmountInCents,
  currency: currency.toUpperCase(),
  discountCents,
  paymentIntentId: paymentIntent.id,
  purchaseId,
  paymentScenario: scenarioAdjusted,
  breakdown: {
    lines,
    subtotalCents: pricing.subtotalCents,
    feeMode: pricing.feeMode,
    platformFeeCents: pricing.platformFeeCents,
    totalCents: totalAmountInCents,
    currency: currency.toUpperCase(),
  },
  intentFingerprint,
  idempotencyKey: clientIdempotencyKey ?? effectiveDedupeKey,
});
}

catch (err) {
  console.error("Erro PaymentIntent:", err);
  return NextResponse.json(
    { ok: false, error: "Erro ao criar PaymentIntent." },
    { status: 500 },
);
}
}

```

app/api/platform/fees/route.ts

```
import { NextResponse } from "next/server";
import { getPlatformAndStripeFees } from "@/lib/platformSettings";

export async function GET() {
  try {
    const { orya, stripe } = await getPlatformAndStripeFees();
    return NextResponse.json({ ok: true, orya, stripe }, { status: 200 });
  } catch (err) {
    console.error("[platform/fees] unexpected error", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
  }
}
```

app/api/profiles/check-username/route.ts

```
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { normalizeAndValidateUsername } from "@/lib/globalUsernames";

export async function POST(req: NextRequest) {
  try {
    const body = (await req.json()).catch(() => null) as { username?: string } | null;
    if (!body || typeof body.username !== "string") {
      return NextResponse.json({ ok: false, error: "username é obrigatório" }, { status: 400 });
    }

    const validated = normalizeAndValidateUsername(body.username);
    if (!validated.ok) {
      return NextResponse.json({ ok: false, error: validated.error }, { status: 400 });
    }

    const existing = await prisma.globalUsername.findUnique({
      where: { username: validated.username },
      select: { ownerType: true, ownerId: true },
    });

    const available = !existing;
    return NextResponse.json({ ok: true, available, username: validated.username });
  } catch (error) {
    console.error(error);
    return NextResponse.json({ ok: false, error: "Erro ao verificar username" }, { status: 500 });
  }
}
```

app/api/profiles/me/route.ts

app/api/profiles/save-avatar/route.ts

app/api/profiles/save-basic/route.ts

```
// app/api/profiles/save-basic/route.ts
import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { parsePhoneNumberFromString } from "libphonenumber-js";
import { setUsernameForOwner, UsernameTakenError, normalizeAndValidateUsername } from "@/lib/globalUsernames";

interface SaveBasicBody {
```

```

fullName?: string;
username?: string;
contactPhone?: string | null;
avatarUrl?: string | null;
visibility?: "PUBLIC" | "PRIVATE";
allowEmailNotifications?: boolean;
allowEventReminders?: boolean;
allowFriendRequests?: boolean;
}

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();

    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json(
        { ok: false, error: "Não autenticado." },
        { status: 401 }
      );
    }

    const userId = user.id;

    const body = (await req.json().catch(() => null)) as SaveBasicBody | null;

    if (!body || typeof body !== "object") {
      return NextResponse.json(
        { ok: false, error: "Body inválido." },
        { status: 400 }
      );
    }

    const rawFullName = body.fullName ?? "";
    const rawUsername = body.username ?? "";
    const rawPhone = body.contactPhone;
    const avatarUrl = body.avatarUrl ?? undefined;
    const visibility = body.visibility === "PRIVATE" ? "PRIVATE" : body.visibility === "PUBLIC" ? "PUBLIC" : undefined;
    const allowEmailNotifications = typeof body.allowEmailNotifications === "boolean" ? body.allowEmailNotifications : undefined;
    const allowEventReminders = typeof body.allowEventReminders === "boolean" ? body.allowEventReminders : undefined;
    const allowFriendRequests = typeof body.allowFriendRequests === "boolean" ? body.allowFriendRequests : undefined;

    const fullName = rawFullName.trim();
    const username = rawUsername.trim();

    let normalizedPhone: string | null | undefined = undefined;
    if (rawPhone !== undefined) {
      if (rawPhone === null || rawPhone === "") {
        normalizedPhone = null;
      } else if (typeof rawPhone === "string") {
        const parsed = parsePhoneNumberFromString(rawPhone.trim(), "PT");
        if (parsed && parsed.isPossible()) {
          normalizedPhone = parsed.number; // E.164
        } else {
          return NextResponse.json(
            { ok: false, error: "Telefone inválido." },
            { status: 400 },
          );
        }
      }
    }

    const validatedUsername = normalizeAndValidateUsername(username);

    if (!fullName || !validatedUsername.ok) {
      return NextResponse.json(
        {
          ok: false,
          error:
            validatedUsername.ok
              ? "Nome completo e username são obrigatórios."
              : validatedUsername.error,
        },
      );
    }
  }
}

```

```

        { status: 400 }
    );
}

const usernameNormalized = validatedUsername.username;

const profile = await prisma.$transaction(async (tx) => {
    await setUsernameForOwner({
        username: usernameNormalized,
        ownerType: "user",
        ownerId: userId,
        tx,
    });

    return tx.profile.upsert({
        where: { id: userId },
        update: {
            fullName,
            username: usernameNormalized,
            onboardingDone: true,
            ... (normalizedPhone !== undefined ? { contactPhone: normalizedPhone } : {}),
            ... (avatarUrl !== undefined ? { avatarUrl: avatarUrl || null } : {}),
            ... (visibility ? { visibility } : {}),
            ... (allowEmailNotifications !== undefined ? { allowEmailNotifications } : {}),
            ... (allowEventReminders !== undefined ? { allowEventReminders } : {}),
            ... (allowFriendRequests !== undefined ? { allowFriendRequests } : {}),
        },
        create: {
            id: userId,
            fullName,
            username: usernameNormalized,
            onboardingDone: true,
            roles: ["user"],
            contactPhone: normalizedPhone ?? null,
            avatarUrl: avatarUrl ?? null,
            visibility: visibility ?? "PUBLIC",
            allowEmailNotifications: allowEmailNotifications ?? true,
            allowEventReminders: allowEventReminders ?? true,
            allowFriendRequests: allowFriendRequests ?? true,
        },
    });
});

const safeProfile = {
    id: profile.id,
    username: profile.username,
    fullName: profile.fullName,
    avatarUrl: profile.avatarUrl,
    bio: profile.bio,
    city: profile.city,
    favouriteCategories: profile.favouriteCategories,
    onboardingDone: profile.onboardingDone,
    roles: profile.roles,
    visibility: profile.visibility,
    allowEmailNotifications: profile.allowEmailNotifications,
    allowEventReminders: profile.allowEventReminders,
    allowFriendRequests: profile.allowFriendRequests,
};

return NextResponse.json(
{
    ok: true,
    profile: safeProfile,
},
{ status: 200 }
);
} catch (err) {
if (err instanceof UsernameTakenError) {
    return NextResponse.json(
    {
        ok: false,
        error: "Este username já está a ser utilizado.",
        code: "USERNAME_TAKEN",
    },
    { status: 409 },
);
}
console.error("POST /api/profiles/save-basic error:", err);
}

```

```

        return NextResponse.json(
          {
            ok: false,
            error: "Erro inesperado ao guardar perfil.",
          },
          { status: 500 }
        );
      }
    }
  }
}

```

app/api/profiles/save-interests/route.ts

```


```

app/api/profiles/save-location/route.ts

```


```

app/api/qr/[token]/route.ts

```

// app/api/qr/[token]/route.ts
import { NextRequest, NextResponse } from "next/server";
import QRCode from "qrcode";
import crypto from "crypto";
import { prisma } from "@lib/prisma";
import { buildQrToken } from "@lib/qr";

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

export async function GET(
  _req: NextRequest,
  context: { params: { token: string } | Promise<{ token: string }> }
) {
  const { token } = await context.params;

  if (typeof token !== "string" || !token.trim()) {
    return new NextResponse("Missing token", { status: 400 });
  }

  try {
    // Carregar o bilhete real para poder gerar payload rotativo (ORYA2)
    const ticket = await prisma.ticket.findUnique({
      where: { qrSecret: token },
      include: {
        event: {
          select: { id: true },
        },
      },
    });

    if (!ticket) {
      return new NextResponse("Ticket not found", { status: 404 });
    }

    // Garantir seed para rotação
    let rotatingSeed = ticket.rotatingSeed ?? null;
    if (!rotatingSeed) {
      rotatingSeed = crypto.randomUUID();
      try {
        await prisma.ticket.update({
          where: { id: ticket.id },
          data: { rotatingSeed },
        });
      } catch (err) {
        console.error("[QR] Falha a guardar rotatingSeed", err);
      }
    }

    const signedPayload = buildQrToken({
      ticketId: ticket.id,
      eventId: ticket.eventId,
    });
  }
}

```

```
        userId: ticket.userId ?? null,
        qrToken: token,
        lifetimeSeconds: 60 * 60 * 8,
        seed: rotatingSeed ?? undefined,
        useRotationWindow: true,
    });
}

const png = await QRCode.toBuffer(signedPayload, {
    type: "png",
    width: 512,
    margin: 1,
    errorCorrectionLevel: "M",
});
}

return new NextResponse(png, {
    status: 200,
    headers: {
        "Content-Type": "image/png",
        "Cache-Control": "no-store, max-age=0",
    },
});
} catch (err) {
    console.error("[QR] Error generating:", err);
    return new NextResponse("Error generating QR", { status: 500 });
}
}
```

app/api/qr/generate/route.ts

app/api/qr/validate/route.ts

```
// app/api/qr/validate/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import crypto from "crypto";
import { env } from "@lib/env";

export const dynamic = "force-dynamic";
export const runtime = "nodejs";

/***
 * ENTERPRISE S2 - ORYA2 SIGNATURE VALIDATION
 *
 * Aceita apenas payloads formados como:
 * ORYA2:<base64urlPayload>.<base64urlSignature>
 *
 * Valida:
 * - Formato
 * - Base64
 * - JSON interno
 * - Campos obrigatórios
 * - HMAC SHA256 correto
 * - Token existe e corresponde ao QR
 * - Ticket não expirou
 */
return NextResponse.json(
  { ok: false, error: "PUBLIC_QR_VALIDATION_DISABLED" },
  { status: 403 }
);
/*
try {
  const url = req.nextUrl;
  const searchParams = url.searchParams;
  const raw = searchParams.get("token");

  if (!raw) {
    return NextResponse.json(
      { ok: false, error: "MISSING_TOKEN" },
      { status: 400 }
    );
  }

  const ticket = await prisma.ticket.findUnique({
    where: { token: raw }
  });

  if (!ticket) {
    return NextResponse.json(
      { ok: false, error: "TOKEN_NOT_FOUND" },
      { status: 404 }
    );
  }

  const ticketSignature = Buffer.from(
    ticket.signature,
    "base64"
  ).toString("utf-8");
  const payload = Buffer.from(
    ticket.payload,
    "base64"
  ).toString("utf-8");

  const verify = crypto.createVerify("HMAC-SHA256");
  verify.update(payload);
  const verifySignature = verify.sign(
    Buffer.from(ticket.key, "hex"),
    "hex"
  );

  if (verifySignature !== ticketSignature) {
    return NextResponse.json(
      { ok: false, error: "INVALID_SIGNATURE" },
      { status: 401 }
    );
  }

  const user = await prisma.user.findUnique({
    where: { id: ticket.userId }
  });

  if (!user) {
    return NextResponse.json(
      { ok: false, error: "USER_NOT_FOUND" },
      { status: 404 }
    );
  }

  const qrData = {
    id: ticket.id,
    user: {
      name: user.name,
      email: user.email
    }
  };

  return NextResponse.json(qrData);
}
*/
```

```

    );
}

// -----
// 1) Verificar prefixo ORYA2
// -----
if (!raw.startsWith("ORYA2:")) {
    return NextResponse.json(
        { ok: false, error: "INVALID_FORMAT_OR_PREFIX" },
        { status: 400 }
    );
}

const stripped = raw.replace("ORYA2:", "");
const parts = stripped.split(",");

if (parts.length !== 2) {
    return NextResponse.json(
        { ok: false, error: "INVALID_FORMAT_PARTS" },
        { status: 400 }
    );
}

const [payloadB64, sigB64] = parts;

// -----
// 2) Decodificar payload
// -----
let payloadJson: any;
try {
    const jsonString = Buffer.from(payloadB64, "base64url").toString();
    payloadJson = JSON.parse(jsonString);
} catch {
    return NextResponse.json(
        { ok: false, error: "INVALID_PAYLOAD_B64_OR_JSON" },
        { status: 400 }
    );
}

const required = ["v", "tok", "tid", "eid", "uid", "ts", "exp"];
for (const key of required) {
    if (!(key in payloadJson)) {
        return NextResponse.json(
            { ok: false, error: `MISSING_FIELD_${key}` },
            { status: 400 }
        );
    }
}

// -----
// 3) Verificar assinatura HMAC SHA256
// -----
const secret = env.qrSecretKey;
if (!secret) {
    return NextResponse.json(
        { ok: false, error: "SERVER_MISCONFIGURED_NO_SECRET" },
        { status: 500 }
    );
}

const hmac = crypto.createHmac("sha256", secret);
hmac.update(payloadB64);
const expectedSig = hmac.digest("base64url");

if (expectedSig !== sigB64) {
    return NextResponse.json(
        { ok: false, error: "INVALID_SIGNATURE" },
        { status: 400 }
    );
}

// -----
// 4) Verificar expiração
// -----
const nowSec = Math.floor(Date.now() / 1000);
if (payloadJson.exp < nowSec) {
    return NextResponse.json(
        { ok: false, error: "TICKET_EXPIRED" },
        { status: 400 }
    );
}

```

```

        { status: 400 }
    );
}

// -----
// 5) Lookup real no DB para confirmar tok
// -----
const purchase = await prisma.ticketPurchase.findUnique({
    where: { qrToken: payloadJson.tok },
    include: { event: true, ticket: true },
});

// Confirm seed matches DB
if (purchase.rotatingSeed && payloadJson.seed !== purchase.rotatingSeed) {
    return NextResponse.json(
        { ok: false, error: "SEED_MISMATCH" },
        { status: 400 }
    );
}

if (!purchase) {
    return NextResponse.json(
        { ok: false, error: "TOKEN_NOT_FOUND_IN_DB" },
        { status: 404 }
    );
}

// -----
// 6) S3 - Marcação de entrada (anti-reutilização)
// -----

// Se já foi usado → inválido
if (purchase.usedAt) {
    return NextResponse.json(
        { ok: false, error: "TICKET_ALREADY_USED" },
        { status: 400 }
    );
}

// Caso exista remainingEntries (multi-entrada)
if (typeof purchase.remainingEntries === "number") {
    if (purchase.remainingEntries <= 0) {
        return NextResponse.json(
            { ok: false, error: "NO_ENTRIES_LEFT" },
            { status: 400 }
        );
    }

    // decremento seguro
    await prisma.ticketPurchase.update({
        where: { id: purchase.id },
        data: { remainingEntries: purchase.remainingEntries - 1 },
    });
} else {
    // Caso seja single-entry → marcar usado
    await prisma.ticketPurchase.update({
        where: { id: purchase.id },
        data: { usedAt: new Date() },
    });
}

// -----
// 6) Validado
// -----
return NextResponse.json(
    {
        ok: true,
        message: "VALID_ORYA2_QR",
        data: {
            purchaseId: purchase.id,
            userId: purchase.userId,
            event: {
                id: purchase.event.id,
                title: purchase.event.title,
                slug: purchase.event.slug,
                startDate: purchase.event.startDate,
                endDate: purchase.event.endDate,
            },
        },
    }
);

```

```

        ticket: {
          id: purchase.ticket.id,
          name: purchase.ticket.name,
        },
      },
    },
    { status: 200 }
  );
} catch (err) {
  console.error("[QR VALIDATE ERROR]", err);
  return NextResponse.json(
    { ok: false, error: "SERVER_ERROR" },
    { status: 500 }
  );
}
*/
}

```

app/api/social/follow-status/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";

export async function GET(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const targetId = req.nextUrl.searchParams.get("userId");
  if (!targetId) return NextResponse.json({ ok: false, error: "INVALID_TARGET" }, { status: 400 });

  const [isFollowing, isFollower] = await Promise.all([
    prisma.follows.findFirst({ where: { follower_id: user.id, following_id: targetId }, select: { id: true } }),
    prisma.follows.findFirst({ where: { follower_id: targetId, following_id: user.id }, select: { id: true } }),
  ]);

  return NextResponse.json(
    {
      ok: true,
      isFollowing: Boolean(isFollowing),
      isFollower: Boolean(isFollower),
    },
    { status: 200 },
  );
}

```

app/api/social/follow/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { notifyNewFollower } from "@/domain/notifications/producer";

export async function POST(req: NextRequest) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();
  if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });

  const body = (await req.json().catch(() => null)) as { targetUserId?: string } | null;
  const targetUserId = body?.targetUserId?.trim();
  if (!targetUserId || targetUserId === user.id) {
    return NextResponse.json({ ok: false, error: "INVALID_TARGET" }, { status: 400 });
  }

  await prisma.follows.upsert({

```

```

    where: {
      follower_id_following_id: {
        follower_id: user.id,
        following_id: targetUserId,
      },
    },
    create: {
      follower_id: user.id,
      following_id: targetUserId,
    },
    update: {},
  });

  await notifyNewFollower({ targetUserId, followerUserId: user.id });

  return NextResponse.json({ ok: true }, { status: 200 });
}

```

app/api/social/followers/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { sanitizeProfileVisibility } from "@/lib/profileVisibility";

export async function GET(req: NextRequest) {
  const userId = req.nextUrl.searchParams.get("userId");
  if (!userId) return NextResponse.json({ ok: false, error: "INVALID_USER" }, { status: 400 });

  const limit = Math.min(Number(req.nextUrl.searchParams.get("limit") || 20), 50);
  const offset = Number(req.nextUrl.searchParams.get("offset") || 0);

  const [items, total] = await Promise.all([
    prisma.follows.findMany({
      where: { following_id: userId },
      include: {
        follower: { select: { id: true, username: true, fullName: true, avatarUrl: true, visibility: true, isDeleted: true } },
      },
      orderBy: { created_at: "desc" },
      skip: offset,
      take: limit,
    }),
    prisma.follows.count({ where: { following_id: userId } }),
  ]);

  return NextResponse.json(
    {
      ok: true,
      total,
      items: items
        .map((f) => sanitizeProfileVisibility(f.follower, userId))
        .filter(Boolean),
    },
    { status: 200 },
  );
}

```

app/api/social/unfollow/route.ts

```

export const runtime = "nodejs";

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
    } = await supabase.auth.getUser();
    if (!user) return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
  }
}

```

```

const body = (await req.json().catch(() => null)) as { targetUserId?: string } | null;
const targetUserId = body?.targetUserId?.trim();
if (!targetUserId) return NextResponse.json({ ok: false, error: "INVALID_TARGET" }, { status: 400 });

await prisma.follows.deleteMany({
  where: { follower_id: user.id, following_id: targetUserId },
});

return NextResponse.json({ ok: true }, { status: 200 });
} catch (err) {
  console.error("[social/unfollow]", err);
  return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/staff/events/route.ts

```

import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";
import { Prisma } from "@prisma/client";

export async function GET() {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    // Buscar assignments ativos para este utilizador
    const assignments = await prisma.staffAssignment.findMany({
      where: {
        userId: user.id,
        revokedAt: null,
        status: "ACCEPTED",
      },
      include: {
        organizer: true,
        event: true,
      },
    });

    if (!assignments.length) {
      return NextResponse.json(
        {
          ok: true,
          events: [],
        },
        { status: 200 }
      );
    }

    // Separar os tipos de permissões
    const organizerIds = assignments
      .filter((a) => a.scope === "GLOBAL")
      .map((a) => a.organizerId)
      .filter((id): id is number => id !== null && id !== undefined);

    const eventIds = assignments
      .filter((a) => a.scope === "EVENT")
      .map((a) => a.eventId)
      .filter((id): id is number => id !== null && id !== undefined);

    if (!organizerIds.length && !eventIds.length) {
      return NextResponse.json(
        {
          ok: true,
          events: [],
        },
        { status: 200 }
      );
    }

    const orFilters: Prisma.EventWhereInput[] = [];
  }
}

```

```
if (organizerIds.length) {
  orFilters.push({ organizerId: { in: organizerIds } });
}

if (eventIds.length) {
  orFilters.push({ id: { in: eventIds } });
}

const events = await prisma.event.findMany({
  where: {
    OR: orFilters.length ? orFilters : undefined,
  },
  include: {
    organizer: true,
  },
  orderBy: {
    startsAt: "asc",
  },
});
}

const payload = events.map((event) => ({
  id: event.id,
  slug: event.slug,
  title: event.title,
  description: event.description,
  startsAt: event.startsAt,
  endsAt: event.endsAt,
  locationName: event.locationName,
  locationCity: event.locationCity,
  organizerName: event.organizer?.displayName ?? null,
}));

return NextResponse.json(
{
  ok: true,
  events: payload,
},
{ status: 200 }
);
} catch (err) {
console.error("GET /api/staff/events error:", err);
return NextResponse.json(
{
  ok: false,
  error: "Erro interno ao carregar eventos de staff.",
},
{ status: 500 }
);
}
}
```

app/api/staff/invitations/accept/route.ts

```
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    const body = (await req.json().catch(() => null)) as { assignmentId?: number } | null;
    const assignmentId = typeof body?.assignmentId === "number" ? body.assignmentId : null;

    if (!assignmentId) {
      return NextResponse.json({ ok: false, error: "MISSING_ASSIGNMENT_ID" }, { status: 400 });
    }

    const assignment = await prisma.staffAssignment.findFirst({
      where: { id: assignmentId, userId: user.id, status: "PENDING", revokedAt: null },
    });

    if (!assignment) {
      return NextResponse.json({ ok: false, error: "INVITE_NOT_FOUND" }, { status: 404 });
    }

    const invite = await prisma.invite.create({
      data: {
        staffAssignmentId: assignment.id,
        userId: user.id,
        status: "PENDING",
        revokedAt: null,
      },
    });

    return NextResponse.json({ ok: true, invite }, { status: 201 });
  } catch (error) {
    console.error(error);
    return NextResponse.json({ ok: false, error: "INTERNAL_SERVER_ERROR" }, { status: 500 });
  }
}
```

```

    }

    await prisma.staffAssignment.update({
      where: { id: assignment.id },
      data: { status: "ACCEPTED", acceptedAt: new Date() },
    });

    return NextResponse.json({ ok: true }, { status: 200 });
  } catch (err) {
    console.error("[staff/invitations/accept] error:", err);
    return NextResponse.json({ ok: false, error: "SERVER_ERROR" }, { status: 500 });
  }
}

```

app/api/staff/invitations/reject/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    const body = (await req.json()).catch(() => null) as { assignmentId?: number } | null;
    const assignmentId = typeof body?.assignmentId === "number" ? body.assignmentId : null;

    if (!assignmentId) {
      return NextResponse.json({ ok: false, error: "MISSING_ASSIGNMENT_ID" }, { status: 400 });
    }

    const assignment = await prisma.staffAssignment.findFirst({
      where: { id: assignmentId, userId: user.id, status: "PENDING", revokedAt: null },
    });

    if (!assignment) {
      return NextResponse.json({ ok: false, error: "INVITE_NOT_FOUND" }, { status: 404 });
    }

    await prisma.staffAssignment.update({
      where: { id: assignment.id },
      data: { status: "REVOKED", revokedAt: new Date() },
    });

    return NextResponse.json({ ok: true }, { status: 200 });
  } catch (err) {
    console.error("[staff/invitations/reject] error:", err);
    return NextResponse.json({ ok: false, error: "SERVER_ERROR" }, { status: 500 });
  }
}

```

app/api/staff/invitations/route.ts

```

import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated } from "@/lib/security";

export async function GET() {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    const invitations = await prisma.staffAssignment.findMany({
      where: {
        userId: user.id,
        status: "PENDING",
        revokedAt: null,
      },
      include: {
        event: true,
        organizer: true,
      },
    });
  }
}

```

```

    },
    orderBy: { createdAt: "desc" },
  ));

  const payload = invitations.map((inv) => ({
    id: inv.id,
    scope: inv.scope,
    eventId: inv.eventId,
    createdAt: inv.createdAt,
    event: inv.event
    ?
    {
      id: inv.event.id,
      title: inv.event.title,
      startsAt: inv.event.startsAt,
      locationName: inv.event.locationName,
      locationCity: inv.event.locationCity,
    }
    :
    null,
    organizer: inv.organizer
    ?
    {
      id: inv.organizer.id,
      displayName: inv.organizer.displayName,
    }
    :
    null,
  }));
}

return NextResponse.json({ ok: true, invitations: payload }, { status: 200 });
} catch (err) {
  console.error("[staff/invitations] error:", err);
  return NextResponse.json({ ok: false, error: "Erro interno." }, { status: 500 });
}
}

```

app/api/staff/validate-qr/route.ts

```

// app/api/staff/validate-qr/route.ts

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { ensureAuthenticated, assertStaffForEvent } from "@/lib/security";
import { parseQrToken, isQrTokenExpired } from "@/lib/qr";

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const user = await ensureAuthenticated(supabase);

    const body = await req.json().catch(() => null) as
    | { token?: string; eventId?: number }
    | null;

    if (!body || !body.token || !body.eventId) {
      return NextResponse.json(
        {
          ok: false,
          reason: "INVALID_BODY",
          message: "Token e eventId são obrigatórios.",
        },
        { status: 400 }
      );
    }

    const { token, eventId } = body;

    // 1) Extrair informação básica do token (ex.: ticketId)
    const parsed = parseQrToken(token);

    if (!parsed.ok) {
      return NextResponse.json(
        {
          ok: false,
          reason: "INVALID_TOKEN",
          message: "QR inválido ou corrompido.",
        },
        { status: 400 }
      );
    }

    const staff = await prisma.staff.findFirst({
      where: {
        id: user.id,
        event: { id: eventId },
      },
    });

    if (!staff) {
      return NextResponse.json(
        {
          ok: false,
          reason: "STAFF_NOT_FOUND",
          message: "Staff não encontrado para o evento especificado.",
        },
        { status: 404 }
      );
    }

    const qrData = await supabase
      .from("qr")
      .select("ticketId")
      .eq("token", token)
      .eq("eventId", eventId)
      .first();

    if (!qrData) {
      return NextResponse.json(
        {
          ok: false,
          reason: "QR_NOT_FOUND",
          message: "QR não encontrado para o token e eventId fornecidos.",
        },
        { status: 404 }
      );
    }

    if (qrData.ticketId !== parsed.ticketId) {
      return NextResponse.json(
        {
          ok: false,
          reason: "QR_INVALID",
          message: "QR inválido ou corrompido.",
        },
        { status: 400 }
      );
    }

    if (isQrTokenExpired(parsed.expiresAt)) {
      return NextResponse.json(
        {
          ok: false,
          reason: "QR_EXPIRED",
          message: "QR expirado.",
        },
        { status: 400 }
      );
    }

    return NextResponse.json(
      {
        ok: true,
        ticketId: qrData.ticketId,
      },
      { status: 200 }
    );
  } catch (err) {
    console.error("[staff/validate-qr] error:", err);
    return NextResponse.json(
      {
        ok: false,
        error: "Erro interno.",
      },
      { status: 500 }
    );
  }
}

```

```

        },
        { status: 400 }
    );
}

const ticketId = parsed.payload.tid;

// 2) Carregar o bilhete + evento associado
const ticket = await prisma.ticket.findUnique({
    where: { id: ticketId },
    include: {
        event: true,
    },
});
if (!ticket || !ticket.event) {
    return NextResponse.json(
        {
            ok: false,
            reason: "TICKET_NOT_FOUND",
            message: "Bilhete não encontrado.",
        },
        { status: 404 }
    );
}

const event = ticket.event;

// Verificar se o QR é mesmo para este evento
if (event.id !== eventId) {
    return NextResponse.json(
        {
            ok: false,
            reason: "NOT_FOR_EVENT",
            message: "Este bilhete não pertence a este evento.",
        },
        { status: 400 }
    );
}

// 3) Verificar se o utilizador tem permissões de staff para este evento
const assignments = await prisma.staffAssignment.findMany({
    where: {
        userId: user.id,
        revokedAt: null,
    },
});

try {
    assertStaffForEvent(user, assignments, event);
} catch (err) {
    return NextResponse.json(
        {
            ok: false,
            reason: "NOT_STAFF_FOR_EVENT",
            message: "Não tens permissão para validar bilhetes neste evento.",
        },
        { status: 403 }
    );
}

// 4) Regras do bilhete
if (ticket.status !== "ACTIVE") {
    const reason = ticket.usedAt ? "ALREADY_USED" : "NOT_ACTIVE";
    const message = ticket.usedAt
        ? "Este bilhete já foi usado."
        : "Este bilhete não está activo.";

    return NextResponse.json(
        {
            ok: false,
            reason,
            message,
        },
        { status: 400 }
    );
}
}

```

```

// Verificar expiração do QR, se aplicável
if (isQrTokenExpired(token)) {
  return NextResponse.json(
    {
      ok: false,
      reason: "QR_EXPIRED",
      message: "Este QR já expirou, pede ao participante para actualizar.",
    },
    { status: 400 }
  );
}

// 5) Marcar como usado
await prisma.ticket.update({
  where: { id: ticket.id },
  data: {
    status: "USED",
    usedAt: new Date(),
  },
});

return NextResponse.json(
  {
    ok: true,
    status: "USED",
    message: "Entrada registada com sucesso.",
    ticketId: ticket.id,
    eventId: event.id,
  },
  { status: 200 }
);
} catch (err) {
  console.error("POST /api/staff/validate-qr error:", err);
  return NextResponse.json(
    {
      ok: false,
      reason: "INTERNAL_ERROR",
      message: "Ocorreu um erro interno ao validar o QR.",
    },
    { status: 500 }
  );
}
}
}

```

app/api/stripe/webhook/route.ts

```

// app/api/stripe/webhook/route.ts

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

import { NextRequest, NextResponse } from "next/server";
import Stripe from "stripe";
import { prisma } from "@/lib/prisma";
import { env } from "@/lib/env";
import { Prisma, PadelPairingPaymentStatus, PadelPairingSlotStatus, PadelPaymentMode, TicketStatus, FeeMode, PromoType, PaymentEventSource, PadelPairingLifecycleStatus, PadelPairingStatus, SaleSummaryStatus } from "@prisma/client";
import { stripe } from "@/lib/stripeClient";
import crypto from "crypto";
import { supabaseAdmin } from "@/lib/supabaseAdmin";
import { sendPurchaseConfirmationEmail } from "@/lib/emailSender";
import { parsePhoneNumberFromString } from "libphonenumber-js/min";
import { computePricing } from "@/lib/pricing";
import { getStripeBaseFees } from "@/lib/platformSettings";
import { normalizePaymentScenario } from "@/lib/paymentScenario";
import { checkoutMetadataSchema, normalizeItemsForMetadata, parseCheckoutItems } from "@/lib/checkoutSchemas";
import { computeGraceUntil } from "@/domain/padelDeadlines";
import { enqueueOperation } from "@/lib/operations/enqueue";
import { ensureEntriesForConfirmedPairing } from "@/domain/tournaments/ensureEntriesForConfirmedPairing";
import { resolveOwner } from "@/lib/ownership/resolveOwner";
import {
  queuePartnerPaid,
  queueDeadlineExpired,
  queueOffsessionActionRequired,
} from "@/domain/notifications/splitPayments";
import {

```

```

queueMatchChanged,
queueMatchResult,
queueNextOpponent,
queueBracketPublished,
queueTournamentEve,
queueEliminated,
queueChampion,
} from "@domain/notifications/tournament";

const webhookSecret = env.stripeWebhookSecret;
const FREE_PLACEHOLDER_INTENT_ID = "FREE_CHECKOUT";

export async function POST(req: NextRequest) {
  const sig = req.headers.get("stripe-signature");
  if (!sig) {
    console.error("[Webhook] Missing signature header");
    return new Response("Missing signature", { status: 400 });
  }

  const body = await req.text();

  let event: Stripe.Event;
  try {
    event = stripe.webhooks.constructEvent(body, sig, webhookSecret);
  } catch (err) {
    const message =
      err instanceof Error ? err.message : "Unknown signature validation error";
    console.error("[Webhook] Invalid signature:", message);
    return new Response("Invalid signature", { status: 400 });
  }

  console.log("[Webhook] Event recebido:", {
    id: event.id,
    type: event.type,
  });

  try {
    switch (event.type) {
      case "payment_intent.succeeded": {
        const intent = event.data.object as Stripe.PaymentIntent;
        if (intent.id === FREE_PLACEHOLDER_INTENT_ID) {
          console.log("[Webhook] payment_intent.succeeded ignorado (FREE_CHECKOUT placeholder)");
          break;
        }
        console.log("[Webhook] payment_intent.succeeded", {
          id: intent.id,
          amount: intent.amount,
          currency: intent.currency,
          metadata: intent.metadata,
        });
        const purchaseAnchor =
          typeof intent.metadata?.purchaseId === "string" && intent.metadata.purchaseId.trim() !== ""
            ? intent.metadata.purchaseId.trim()
            : intent.id;
        // Registrar PaymentEvent ingest-only
        try {
          await prisma.paymentEvent.upsert({
            where: { stripePaymentIntentId: intent.id },
            update: {
              status: "PROCESSING",
              purchaseId: purchaseAnchor,
              stripeEventId: event.id,
              source: PaymentEventSource.WEBHOOK,
              dedupeKey: event.id,
              amountCents: intent.amount ?? null,
              userId: typeof intent.metadata?.userId === "string" ? intent.metadata.userId : undefined,
              updatedAt: new Date(),
              errorMessage: null,
              mode: intent.livemode ? "LIVE" : "TEST",
              isTest: !intent.livemode,
            },
            create: {
              stripePaymentIntentId: intent.id,
              status: "PROCESSING",
              purchaseId: purchaseAnchor,
              stripeEventId: event.id,
              source: PaymentEventSource.WEBHOOK,
              dedupeKey: event.id,
            }
          });
        } catch (err) {
          console.error(`[Webhook] Failed to upsert paymentEvent: ${err.message}`);
        }
      }
    }
  }
}

```

```

        attempt: 1,
        eventId:
          typeof intent.metadata?.eventId === "string" && Number.isFinite(Number(intent.metadata.eventId))
            ? Number(intent.metadata.eventId)
            : undefined,
        userId: typeof intent.metadata?.userId === "string" ? intent.metadata.userId : undefined,
        amountCents: intent.amount ?? null,
        platformFeeCents: typeof intent.metadata?.platformFeeCents === "string" ? Number(intent.metadata.platformFeeCents)
      : null,
      mode: intent.livemode ? "LIVE" : "TEST",
      isTest: !intent.livemode,
    },
  );
} catch (logErr) {
  console.warn("[Webhook] Falha ao registrar PaymentEvent ingest-only", logErr);
}

await enqueueOperation({
  operationType: "FULFILL_PAYMENT",
  dedupeKey: intent.id,
  correlations: { paymentIntentId: intent.id, stripeEventId: event.id, purchaseId: purchaseAnchor },
  payload: { paymentIntentId: intent.id, stripeEventType: event.type, stripeEventId: event.id },
});

break;
}

case "charge.refunded": {
  const charge = event.data.object as Stripe.Charge;
  console.log("[Webhook] charge.refunded", {
    id: charge.id,
    payment_intent: charge.payment_intent,
  });
  await enqueueOperation({
    operationType: "PROCESS_STRIPE_EVENT",
    dedupeKey: event.id,
    correlations: {
      paymentIntentId:
        typeof charge.payment_intent === "string"
          ? charge.payment_intent
          : charge.payment_intent?.id ?? null,
      stripeEventId: event.id,
    },
    payload: {
      stripeEventType: event.type,
      chargeId: charge.id,
      paymentIntentId:
        typeof charge.payment_intent === "string"
          ? charge.payment_intent
          : charge.payment_intent?.id ?? null,
    },
  });
  break;
}

default: {
  // outros eventos, por agora, podem ser ignorados
  console.log("[Webhook] Evento ignorado:", event.type);
  break;
}
}

catch (err) {
  console.error("[Webhook] Error processing event:", err);
  // devolvemos 200 na mesma para o Stripe não re-tentar para sempre
}

return NextResponse.json({ received: true });
}

type PromoSnapshot = { code: string; label?: string | null; type: PromoType | null; value: number | null };
async function loadPromoSnapshots(ids: number[]) {
  const promos = await prisma.promoCode.findMany({
    where: { id: { in: ids } },
    select: { id: true, code: true, type: true, value: true },
  });
  const map = new Map<number, PromoSnapshot>();
  promos.forEach((p) => {
    map.set(p.id, { code: p.code, label: p.code, type: p.type, value: p.value });
  });
}

```

```

    });
    return map;
}
type EventWithTickets = Prisma.EventGetPayload<{
  include: { ticketTypes: true };
}>;
type ParsedItem = { ticketTypeId: number; quantity: number };
type BreakdownLine = {
  ticketTypeId: number;
  quantity: number;
  unitPriceCents: number;
  discountPerUnitCents?: number;
  lineTotalCents?: number;
  lineGrossCents?: number;
  lineNetCents?: number;
  platformFeeCents?: number;
  promoCodeId?: number;
};
type BreakdownPayload = {
  lines: BreakdownLine[];
  subtotalCents: number;
  discountCents: number;
  platformFeeCents: number;
  totalCents: number;
  feeMode?: string;
  currency?: string;
  feeBpsApplied?: number;
  feeFixedApplied?: number;
};
export async function fulfillPayment(intent: Stripe.PaymentIntent, stripeEventId?: string) {
  // [ORYA PATCH v1] Webhook reforçado e preparado para múltiplos bilhetes com total segurança.
  const meta = intent.metadata ?? {};
  let parsedBreakdown: BreakdownPayload | null = null;
  if (typeof meta.breakdown === "string") {
    try {
      const parsed = JSON.parse(meta.breakdown);
      if (parsed && Array.isArray(parsed.lines)) {
        parsedBreakdown = {
          lines: parsed.lines as BreakdownLine[],
          subtotalCents: Number(parsed.subtotalCents ?? 0),
          discountCents: Number(parsed.discountCents ?? 0),
          platformFeeCents: Number(parsed.platformFeeCents ?? 0),
          totalCents: Number(parsed.totalCents ?? 0),
          feeMode: parsed.feeMode,
          currency: parsed.currency ?? "EUR",
        };
      }
    } catch (err) {
      console.warn("[fulfillPayment] breakdown inválido no metadata", err);
    }
  }
  const paymentScenario = normalizePaymentScenario(
    typeof meta.paymentScenario === "string" ? meta.paymentScenario : null,
  );
  const scenario = typeof (meta as Record<string, unknown>)?._scenario === "string" ? (meta as Record<string, unknown>).scenario : null;
  const hasPadelPairingMeta =
    Number.isFinite(Number((meta as Record<string, unknown>)?._pairingId)) ||
    Number.isFinite(Number((meta as Record<string, unknown>)?._slotId));

  // Resale: processar aqui (deixámos de usar checkout.session.completed)
  if (paymentScenario === "RESALE") {
    const resaleId = typeof meta.resaleId === "string" ? meta.resaleId : null;
    const ticketId = typeof meta.ticketId === "string" ? meta.ticketId : null;
    const buyerUserId = typeof meta.buyerUserId === "string" ? meta.buyerUserId : null;

    if (!resaleId || !ticketId || !buyerUserId) {
      console.error("[fulfillPayment][RESALE] Metadata incompleta", { resaleId, ticketId, buyerUserId, intentId: intent.id });
      return;
    }

    try {
      await prisma.$transaction(async (tx) => {
        const resale = await tx.ticketResale.findUnique({
          where: { id: resaleId },
        });
      });
    }
  }
}

```

```

        include: { ticket: true },
    });

    if (!resale || !resale.ticket) {
        console.error("[fulfillPayment][RESALE] Revenda não encontrada", { resaleId });
        return;
    }

    // Idempotência: se já não estiver LISTED, não repetimos a operação
    if (resale.status !== "LISTED") {
        console.log("[fulfillPayment][RESALE] Revenda já processada ou num estado inválido", {
            resaleId,
            status: resale.status,
        });
        return;
    }

    await tx.ticketResale.update({
        where: { id: resale.id },
        data: {
            status: "SOLD",
            completedAt: new Date(),
        },
    });

    await tx.ticket.update({
        where: { id: resale.ticketId },
        data: {
            userId: buyerUserId,
            status: "ACTIVE",
        },
    });

    await tx.paymentEvent.upsert({
        where: { stripePaymentIntentId: intent.id },
        update: {
            status: "OK",
            amountCents: intent.amount,
            eventId: resale.ticket.eventId,
            userId: buyerUserId,
            updatedAt: new Date(),
            errorMessage: null,
            mode: intent.livemode ? "LIVE" : "TEST",
            isTest: !intent.livemode,
            purchaseId: purchaseId ?? intent.id,
            source: PaymentEventSource.WEBHOOK,
            dedupeKey: purchaseId ?? intent.id,
            attempt: { increment: 1 },
        },
        create: {
            stripePaymentIntentId: intent.id,
            status: "OK",
            amountCents: intent.amount,
            eventId: resale.ticket.eventId,
            userId: buyerUserId,
            mode: intent.livemode ? "LIVE" : "TEST",
            isTest: !intent.livemode,
            purchaseId: purchaseId ?? intent.id,
            source: PaymentEventSource.WEBHOOK,
            dedupeKey: purchaseId ?? intent.id,
            attempt: 1,
        },
    });
});

console.log("[fulfillPayment][RESALE] processada com sucesso", { resaleId, ticketId, buyerUserId });
} catch (err) {
    console.error("[fulfillPayment][RESALE] erro", err);
}

return;
}

const rawUserId = typeof meta.userId === "string" ? meta.userId.trim() : "";
const rawOwnerUserId = (meta as Record<string, unknown>)?ownerUserId === "string"
    ? (meta as Record<string, unknown>)?ownerUserId?.trim()
    : "";
const userId = rawUserId !== "" ? rawUserId : rawOwnerUserId !== "" ? rawOwnerUserId : null;

```

```

const guestEmail =
  typeof meta.guestEmail === "string"
    ? meta.guestEmail.trim().toLowerCase()
    : "";
const guestName =
  typeof meta.guestName === "string" ? meta.guestName.trim() : "";
const guestPhoneRaw =
  typeof meta.guestPhone === "string" ? meta.guestPhone.trim() : "";
const promoCodeId =
  typeof meta.promoCode === "string" && meta.promoCode.trim() !== ""
    ? meta.promoCode.trim()
    : "";
// Padel SPLIT/FULL: tratar logo aqui e sair
if (paymentScenario === "GROUP_SPLIT" && hasPadelPairingMeta) {
  await handlePadelSplitPayment(intent);
  return;
}
if (paymentScenario === "GROUP_FULL" && hasPadelPairingMeta) {
  await handlePadelFullPayment(intent);
  return;
}
if (scenario === "GROUP_SPLIT_SECOND_CHARGE") {
  await handleSecondCharge(intent);
  return;
}

const normalizePhone = (phone: string | null | undefined, defaultCountry = "PT") => {
  if (!phone) return null;
  const cleaned = phone.trim();
  if (!cleaned) return null;
  const parsed = parsePhoneNumberFromString(cleaned, defaultCountry);
  if (parsed && parsed.isPossible() && parsed.isValid()) {
    return parsed.number;
  }
  const regexPT = /^(?:\+351)?[1236]\d{7}$/;
  if (regexPT.test(cleaned)) {
    const digits = cleaned.replace(/[^d]/g, "");
    return digits.startsWith("351") ? `+$ {digits}` : `+351 ${digits}`;
  }
  return null;
};
const guestPhone = normalizePhone(guestPhoneRaw);
const purchaseId =
  typeof meta.purchaseId === "string" && meta.purchaseId.trim() !== ""
    ? meta.purchaseId.trim()
    : null;

console.log("[fulfillPayment] Início", {
  intentId: intent.id,
  stripeEventId,
  userId,
  purchaseId,
  meta,
});

// Segurança extra: só processamos intents que vieram da nossa app
if (!userId && !guestEmail) {
  console.warn(
    "[fulfillPayment] payment_intent sem userId nem guestEmail em metadata, a ignorar",
    intent.id
  );
  return;
}

// ----- PARSE DOS ITENS -----
let items: ParsedItem[] = [];

const parseItemsString = (value: string) => {
  try {
    const parsed = JSON.parse(value);
    const normalized = parseCheckoutItems(parsed);
    if (normalized.length) {
      items = normalized.map((it) => ({
        ticketTypeId: it.ticketTypeId,
        quantity: it.quantity,
      }));
    }
  } catch (err) {

```

```

        console.error("[Webhook] Failed to parse metadata.items:", err);
    }
};

if (typeof meta.items === "string") parseItemsString(meta.items);
if (items.length === 0 && typeof meta.itemsJson === "string") parseItemsString(meta.itemsJson);

if (items.length === 0 && typeof meta.ticketId === "string") {
    const qty = Math.max(1, Number(meta.quantity ?? 1));
    const ticketTypeId = Number(meta.ticketId);
    if (!Number.isNaN(ticketTypeId)) {
        items = [{ ticketTypeId, quantity: qty }];
    }
}

if (items.length === 0) {
    console.warn("[fulfillPayment] No items in metadata", meta);
    return;
}

console.log("[fulfillPayment] Itens depois do parse:", items);

// ----- EVENTO -----
let eventRecord: EventWithTickets | null = null;

if (meta.eventId) {
    const idNum = Number(meta.eventId);
    if (!Number.isNaN(idNum)) {
        eventRecord = await prisma.event.findUnique({
            where: { id: idNum },
            include: { ticketTypes: true },
        });
    }
}

if (!eventRecord && typeof meta.eventSlug === "string") {
    eventRecord = await prisma.event.findUnique({
        where: { slug: meta.eventSlug },
        include: { ticketTypes: true },
    });
}

if (!eventRecord) {
    console.warn("[fulfillPayment] Event not found via metadata:", meta);
    return;
}

console.log("[fulfillPayment] Event encontrado:", {
    eventId: eventRecord.id,
    title: eventRecord.title,
});

const ticketTypeMap = new Map<number, { price: number; currency: string | null }>(
    eventRecord.ticketTypes.map((t) => [t.id, { price: t.price, currency: t.currency }]),
);

let normalizedItems = parsedBreakdown?.lines?.length
    ? normalizeItemsForMetadata(
        parsedBreakdown.lines.map((line) => ({
            ticketTypeId: line.ticketTypeId,
            quantity: Math.max(1, Number(line.quantity ?? 1)),
            unitPriceCents: Number(line.unitPriceCents ?? 0) ||
                (ticketTypeMap.get(line.ticketTypeId)?.price ?? 0),
            currency: (line as { currency?: string })?.currency ??
                ticketTypeMap.get(line.ticketTypeId)?.currency ??
                "EUR",
        })),
    )
    : [];

if (!normalizedItems.length) {
    normalizedItems = normalizeItemsForMetadata(
        items.map((item) => {
            const tt = ticketTypeMap.get(item.ticketTypeId);
            return {
                ticketTypeId: item.ticketTypeId,
        }));
}

```

```

        quantity: item.quantity,
        unitPriceCents: tt?.price ?? 0,
        currency: tt?.currency ?? "EUR",
    );
}),
);
}

const metadataValidation = checkoutMetadataSchema.safeParse({
    paymentScenario,
    purchaseId: purchaseId ?? intent.id,
    items: normalizedItems,
    eventId: eventRecord.id,
    eventSlug: eventRecord.slug,
    pairingId: hasPadelPairingMeta ? Number((meta as Record<string, unknown>)?._pairingId ?? 0) || undefined : undefined,
    owner: {
        userId: userId ?? undefined,
        guestEmail: guestEmail || undefined,
        guestName: guestName || undefined,
        guestPhone: guestPhone || undefined,
        ownerUserId: typeof (meta as Record<string, unknown>)?._ownerUserId === "string" ? (meta as Record<string, unknown>)?._ownerUserId : undefined,
        ownerIdentityId:
            typeof (meta as Record<string, unknown>)?._ownerIdentityId === "string"
            ? (meta as Record<string, unknown>)?._ownerIdentityId
            : undefined,
        emailNormalized:
            typeof (meta as Record<string, unknown>)?._emailNormalized === "string"
            ? (meta as Record<string, unknown>)?._emailNormalized
            : undefined,
    },
});

if (!metadataValidation.success) {
    console.warn("[fulfillPayment] INVALID_METADATA", {
        intentId: intent.id,
        purchaseId,
        errors: metadataValidation.error.flatten(),
    });
    return;
}

const purchaseAnchor = metadataValidation.data.purchaseId;
const ownerMeta = metadataValidation.data.owner;
const ownerResolved = await resolveOwner({
    sessionId: ownerMeta?.ownerUserId ?? ownerMeta?.userId ?? userId ?? undefined,
    guestEmail: ownerMeta?.emailNormalized ?? ownerMeta?.guestEmail ?? guestEmail ?? undefined,
});
const ownerUserId = ownerResolved.ownerUserId ?? ownerMeta?.ownerUserId ?? ownerMeta?.userId ?? userId ?? null;
const ownerIdentityId = ownerResolved.ownerIdentityId ?? ownerMeta?.ownerIdentityId ?? null;

const platformFeeTotal = Number(meta.platformFeeCents ?? 0);
const totalTicketsRequested = items.reduce(
    (sum, item) => sum + Math.max(1, Number(item.quantity ?? 0)),
    0
);

const perTicketPlatformFee =
    totalTicketsRequested > 0
    ? Math.floor(platformFeeTotal / totalTicketsRequested)
    : 0;
let feeRemainder =
    totalTicketsRequested > 0 ? platformFeeTotal % totalTicketsRequested : 0;

// Pré-calcular valores por linha (gross/net/desconto) para gravar SaleLines e tickets de forma consistente
const computedLineMap = new Map<
    number,
    {
        grossCents: number;
        netCents: number;
        discountAllocCents: number;
        discountPerUnitCents: number;
        platformFeeCents: number;
        quantity: number;
        unitPriceCents: number;
    }
>();
if (parsedBreakdown?.lines?.length) {

```

```

const subtotal = Math.max(0, parsedBreakdown.subtotalCents ?? 0);
const totalDiscount = Math.max(0, parsedBreakdown.discountCents ?? 0);
let remainingDiscount = totalDiscount;

parsedBreakdown.lines.forEach((line, idx) => {
  const qty = Math.max(1, Number(line.quantity ?? 1));
  const linePlatformFee = Number(line.platformFeeCents ?? 0);
  const gross = Number(
    line.lineTotalCents ??
    line.lineGrossCents ??
    (line.unitPriceCents != null ? line.unitPriceCents * qty : 0),
  );
  let alloc = 0;
  if (subtotal > 0 && totalDiscount > 0) {
    const proportional = Math.floor((gross * totalDiscount) / subtotal);
    const isLast = idx === parsedBreakdown.lines.length - 1;
    alloc = isLast ? Math.max(0, remainingDiscount) : Math.min(remainingDiscount, proportional);
  }
  remainingDiscount -= alloc;

  const net = Math.max(0, gross - alloc);
  const unitPrice = Number(line.unitPriceCents ?? Math.round(gross / qty));
  const discountPerUnit = Math.floor(alloc / qty);

  computedLineMap.set(line.ticketTypeId, {
    grossCents: gross,
    netCents: net,
    discountAllocCents: alloc,
    discountPerUnitCents: discountPerUnit,
    platformFeeCents: linePlatformFee,
    quantity: qty,
    unitPriceCents: unitPrice,
  });
});
});

// ----- IDEMPOTÊNCIA (permite múltiplos bilhetes por pagamento) -----
const already = await prisma.ticket.findFirst({
  where: { stripePaymentIntentId: intent.id },
});

if (already && !process.env.ENABLE_WORKER_PAYMENTS) {
  console.log("[fulfillPayment] INTENT JÁ PROCESSADO – evitando duplicação:", intent.id);
  try {
    await prisma.paymentEvent.updateMany({
      where: { stripePaymentIntentId: intent.id },
      data: {
        status: "OK",
        purchaseId: purchaseAnchor,
        stripeEventId: stripeEventId ?? undefined,
        source: PaymentEventSource.WEBHOOK,
        dedupeKey: purchaseAnchor ?? stripeEventId ?? intent.id,
        attempt: { increment: 1 },
        updatedAt: new Date(),
        errorMessage: null,
        mode: intent.livemode ? "LIVE" : "TEST",
        isTest: !intent.livemode,
      },
    });
  } catch (logErr) {
    console.warn("[fulfillPayment] Falha ao marcar paymentEvent como OK num retry", logErr);
  }
  return;
}

// Marcar log como PROCESSING (idempotência/auditoria)
try {
  const updated = await prisma.paymentEvent.updateMany({
    where: { stripePaymentIntentId: intent.id },
    data: {
      status: "PROCESSING",
      eventId: eventRecord.id,
      purchaseId: purchaseAnchor,
      stripeEventId: stripeEventId ?? undefined,
      source: PaymentEventSource.WEBHOOK,
      dedupeKey: purchaseAnchor ?? stripeEventId ?? intent.id,
      attempt: { increment: 1 },
    }
  });
}

```

```

        amountCents: intent.amount ?? null,
        platformFeeCents: platformFeeTotal ?? null,
        userId,
        errorMessage: null,
        updatedAt: new Date(),
        mode: intent.livemode ? "LIVE" : "TEST",
        isTest: !intent.livemode,
    },
});
if (updated.count === 0) {
    await prisma.paymentEvent.create({
        data: {
            stripePaymentIntentId: intent.id,
            status: "PROCESSING",
            purchaseId: purchaseAnchor,
            stripeEventId: stripeEventId ?? undefined,
            source: PaymentEventSource.WEBHOOK,
            dedupeKey: purchaseAnchor ?? stripeEventId ?? intent.id,
            attempt: 1,
            eventId: eventRecord.id,
            userId,
            amountCents: intent.amount ?? null,
            platformFeeCents: platformFeeTotal ?? null,
            mode: intent.livemode ? "LIVE" : "TEST",
            isTest: !intent.livemode,
        },
    });
}
} catch (logErr) {
    console.warn("[fulfillPayment] Não foi possível registar paymentEvent", logErr);
}

// ----- PREPARAR CRIAÇÃO DE BILHETES + STOCK -----
let createdTicketsCount = 0;
let saleSummaryId: number | null = null;
const stripeBaseFees = await getStripeBaseFees();
const estimateStripeFee = (amountCents: number) =>
    Math.max(0, Math.round((amountCents * (stripeBaseFees.feeBps ?? 0)) / 10_000) + (stripeBaseFees.feeFixedCents ?? 0));

// Sanity-check opcional: garantir que breakdown bate com o amount recebido
if (parsedBreakdown && typeof intent.amount_received === "number") {
    const recalculated = computePricing(parsedBreakdown.subtotalCents, parsedBreakdown.discountCents, {
        eventFeeModeOverride: null,
        eventFeeMode: parsedBreakdown.feeMode as FeeMode | null,
        organizerFeeMode: parsedBreakdown.feeMode as FeeMode | null,
        platformDefaultFeeMode: parsedBreakdown.feeMode as FeeMode | null,
        eventPlatformFeeBpsOverride: parsedBreakdown.feeBpsApplied ?? null,
        eventPlatformFeeFixedCentsOverride: parsedBreakdown.feeFixedApplied ?? null,
        organizerPlatformFeeBps: parsedBreakdown.feeBpsApplied ?? null,
        organizerPlatformFeeFixedCents: parsedBreakdown.feeFixedApplied ?? null,
        platformDefaultFeeBps: parsedBreakdown.feeBpsApplied ?? 0,
        platformDefaultFeeFixedCents: parsedBreakdown.feeFixedApplied ?? 0,
    });

    if (recalculated.totalCents !== intent.amount_received) {
        console.warn("[fulfillPayment] Divergência entre breakdown.totalCents e amount_received", {
            intentId: intent.id,
            breakdownTotal: parsedBreakdown.totalCents,
            recalculatedTotal: recalculated.totalCents,
            amountReceived: intent.amount_received,
        });
    }
}

// Tentar obter a fee real do Stripe via balance_transaction
let stripeFeeCents: number | null = null;
try {
    if (intent.latest_charge) {
        const charge = await stripe.charges.retrieve(intent.latest_charge as string, {
            expand: ["balance_transaction"],
        });
        const balanceTx = charge.balance_transaction as Stripe.BalanceTransaction | null;
        if (balanceTx?.fee != null) stripeFeeCents = balanceTx.fee;
    }
} catch (err) {
    console.warn("[fulfillPayment] Não foi possível obter balance_transaction; a usar estimativa", err);
}

```

```

// eslint-disable-next-line @typescript-eslint/no-unused-vars
const stripeFeeForIntentValue =
  stripeFeeCents ??
  estimateStripeFee(parsedBreakdown?.totalCents ?? intent.amount_received ?? intent.amount ?? 0);

await prisma.$transaction(async (tx) => {
  // Persistir breakdown se existir
  if (parsedBreakdown) {
    const promoIds = new Set<number>();
    parsedBreakdown.lines.forEach((l) => {
      if (l.promoCodeId) promoIds.add(l.promoCodeId);
    });
    if (promoCodeId && Number.isFinite(Number(promoCodeId))) {
      promoIds.add(Number(promoCodeId));
    }
    const promoSnapshots =
      promoIds.size > 0 ? await loadPromoSnapshots(Array.from(promoIds)) : new Map<number, PromoSnapshot>();
    const summaryPromo =
      promoCodeId && Number.isFinite(Number(promoCodeId))
        ? promoSnapshots.get(Number(promoCodeId)) ?? null
        : null;
    try {
      const feeMode = parsedBreakdown.feeMode as FeeMode | undefined;
      const stripeFee = stripeFeeCents ?? estimateStripeFee(parsedBreakdown.totalCents ?? 0);
      const netCents = Math.max(
        0,
        (parsedBreakdown.totalCents ?? 0) - (parsedBreakdown.platformFeeCents ?? 0) - stripeFee,
      );
      const summary = await tx.saleSummary.upsert({
        where: { paymentIntentId: intent.id },
        update: {
          eventId: eventRecord.id,
          userId: ownerUserId ?? null,
          ownerUserId: ownerUserId ?? null,
          ownerIdentityId: ownerIdentityId ?? null,
          purchaseId: purchaseAnchor,
          promoCodeId: promoCodeId ? Number(promoCodeId) : null,
          promoCodeSnapshot: summaryPromo?.code ?? null,
          promoLabelSnapshot: summaryPromo?.label ?? summaryPromo?.code ?? null,
          promoTypeSnapshot: summaryPromo?.type ?? null,
          promoValueSnapshot: summaryPromo?.value ?? null,
          subtotalCents: parsedBreakdown.subtotalCents,
          discountCents: parsedBreakdown.discountCents,
          platformFeeCents: parsedBreakdown.platformFeeCents,
          stripeFeeCents: stripeFee,
          totalCents: parsedBreakdown.totalCents,
          netCents,
          feeMode: feeMode,
          currency: parsedBreakdown.currency ?? "EUR",
        },
        create: {
          paymentIntentId: intent.id,
          eventId: eventRecord.id,
          userId: ownerUserId ?? null,
          ownerUserId: ownerUserId ?? null,
          ownerIdentityId: ownerIdentityId ?? null,
          purchaseId: purchaseAnchor,
          promoCodeId: promoCodeId ? Number(promoCodeId) : null,
          promoCodeSnapshot: summaryPromo?.code ?? null,
          promoLabelSnapshot: summaryPromo?.label ?? summaryPromo?.code ?? null,
          promoTypeSnapshot: summaryPromo?.type ?? null,
          promoValueSnapshot: summaryPromo?.value ?? null,
          subtotalCents: parsedBreakdown.subtotalCents,
          discountCents: parsedBreakdown.discountCents,
          platformFeeCents: parsedBreakdown.platformFeeCents,
          stripeFeeCents: stripeFee,
          totalCents: parsedBreakdown.totalCents,
          netCents,
          feeMode: feeMode,
          currency: parsedBreakdown.currency ?? "EUR",
        },
      });
      saleSummaryId = summary.id;
    } catch (err) {
      console.error(`Error persisting breakdown: ${err.message}`);
    }
  }
  // limpar linhas anteriores e regravar
  await tx.saleLine.deleteMany({ where: { saleSummaryId: summary.id } });
  for (const line of parsedBreakdown.lines) {
    const lineCopy = { ...line };
    lineCopy.saleSummaryId = saleSummaryId;
    await tx.saleLine.create({ data: lineCopy });
  }
});

```

```

const linePromo = line.promoCodeId ? promoSnapshots.get(line.promoCodeId) ?? null : summaryPromo;
const computed = computedLineMap.get(line.ticketTypeId);
const grossCents =
  computed?.grossCents ??
  line.lineGrossCents ??
  line.lineTotalCents ??
  (line.unitPriceCents != null && line.quantity != null ? line.unitPriceCents * line.quantity : 0);
const netCents =
  computed?.netCents ??
  (grossCents != null && parsedBreakdown.discountCents != null && parsedBreakdown.subtotalCents
    ? Math.max(
      0,
      grossCents -
        Math.floor(
          (grossCents * parsedBreakdown.discountCents) /
            Math.max(1, parsedBreakdown.subtotalCents),
        ),
    )
    : grossCents);
await tx.saleLine.create({
  data: {
    saleSummaryId: summary.id,
    eventId: eventRecord.id,
    ticketTypeId: line.ticketTypeId,
    promoCodeId: line.promoCodeId ?? (promoCodeId ? Number(promoCodeId) : null),
    promoCodeSnapshot: linePromo?.code ?? null,
    promoLabelSnapshot: linePromo?.label ?? linePromo?.code ?? null,
    promoTypeSnapshot: linePromo?.type ?? null,
    promoValueSnapshot: linePromo?.value ?? null,
    quantity: line.quantity,
    unitPriceCents: computed?.unitPriceCents ?? line.unitPriceCents,
    discountPerUnitCents: computed?.discountPerUnitCents ?? line.discountPerUnitCents ?? 0,
    grossCents: grossCents,
    netCents: netCents,
    platformFeeCents: computed?.platformFeeCents ?? line.platformFeeCents ?? 0,
  },
}, );
}
} catch (err) {
  console.warn("[fulfillPayment] Falha ao persistir saleSummary/saleLines", err);
}
}

if (promoCodeId && saleSummaryId) {
try {
  const promo = await tx.promoCode.findUnique({
    where: { id: Number(promoCodeId) },
    select: { id: true, maxUses: true, perUserLimit: true },
  });
  // Re-check limites em transação (best-effort contra race)
  const totalUses = await tx.promoRedemption.count({ where: { promoCodeId: Number(promoCodeId) } });
  const userUses =
    ownerUserId || guestEmail
    ? await tx.promoRedemption.count({
      where: {
        promoCodeId: Number(promoCodeId),
        OR: [{ userId: ownerUserId ?? undefined }, { guestEmail: guestEmail || undefined }],
      },
    })
    : 0;
  const exceedsGlobal = promo?.maxUses != null && totalUses >= promo.maxUses;
  const exceedsUser = promo?.perUserLimit != null && userUses >= promo.perUserLimit;
  if (!exceedsGlobal && !exceedsUser) {
    try {
      await tx.promoRedemption.upsert({
        where: {
          purchaseId_promoCodeId: {
            purchaseId: purchaseAnchor ?? undefined,
            promoCodeId: Number(promoCodeId),
          },
        },
        update: {
          userId: ownerUserId ?? null,
          guestEmail: guestEmail || null,
        },
        create: {
          promoCodeId: Number(promoCodeId),
          userId: ownerUserId ?? null,
        },
      });
    }
  }
}
}

```

```

        guestEmail: guestEmail || null,
        purchaseId: purchaseAnchor ?? null,
    },
});
} catch (err) {
    const isUnique =
        err &&
        typeof err === "object" &&
        "code" in err &&
        (err as { code: string }).code === "P2002";
    if (!isUnique) throw err;
    console.warn("[fulfillPayment] promoRedemption unique conflict ignorado", {
        promoCodeId,
        purchaseId: purchaseAnchor ?? null,
        userId: ownerUserId ?? null,
        guestEmail,
    });
}
} else {
    console.warn("[fulfillPayment] promoRedemption não criada por limite atingido", {
        promoCodeId,
        totalUses,
        userUses,
    });
}
} catch (err) {
    console.warn("[fulfillPayment] Não foi possível registar promo redemption (tx)", err);
}
}

for (const item of items) {
    const ticketType = eventRecord.ticketTypes.find((t) => t.id === item.ticketTypeId);
    if (!ticketType) {
        console.warn("[fulfillPayment] TicketType not found:", item.ticketTypeId);
        continue;
    }

    const qty = Math.max(1, Number(item.quantity ?? 0));
    if (!qty) continue;

    if (
        ticketType.totalQuantity !== null &&
        ticketType.totalQuantity !== undefined
    ) {
        const remaining = ticketType.totalQuantity - ticketType.soldQuantity;
        if (remaining <= 0 || qty > remaining) {
            console.warn("[fulfillPayment] Insufficient stock for:", {
                ticketTypeId: ticketType.id,
                remaining,
                requested: qty,
            });
            continue;
        }
    }

    const existingTickets = await tx.ticket.findMany({
        where: { purchaseId: purchaseAnchor, ticketTypeId: ticketType.id },
        select: { emissionIndex: true },
    });
    const existingIndexes = new Set<number>(existingTickets.map((t) => t.emissionIndex ?? 0));

    for (let i = 0; i < qty; i++) {
        if (existingIndexes.has(i)) {
            continue;
        }
        // Gerar QR seguro para cada bilhete
        const token = crypto.randomUUID();
        const feeForThisTicket =
            perTicketPlatformFee + (feeRemainder > 0 ? 1 : 0);
        if (feeRemainder > 0) feeRemainder -= 1;

        const lineFromBreakdown = parsedBreakdown?.lines.find(
            (l) => l.ticketTypeId === ticketType.id,
        );
        const computedLine = computedLineMap.get(ticketType.id);
        const lineQty = computedLine?.quantity ?? lineFromBreakdown?.quantity ?? qty;
        const lineNetCents =
            computedLine?.netCents ??

```

```

    lineFromBreakdown?.lineNetCents ??
    lineFromBreakdown?.lineTotalCents ??
    ticketType.price * lineQty;
  const pricePerTicketCents = Math.round(
    lineNetCents / Math.max(1, lineQty),
  );

  const ticket = await tx.ticket.create({
    data: {
      userId: ownerUserId ?? null,
      ownerUserId: ownerUserId ?? null,
      ownerIdentityId: ownerIdentityId ?? null,
      eventId: eventRecord.id,
      ticketTypeId: ticketType.id,
      status: "ACTIVE",
      purchasedAt: new Date(),
      qrSecret: token,
      pricePaid: pricePerTicketCents,
      currency: ticketType.currency,
      platformFeeCents: feeForThisTicket,
      totalPaidCents: pricePerTicketCents + feeForThisTicket,
      stripePaymentIntentId: intent.id,
      purchaseId: purchaseAnchor ?? intent.id,
      saleSummaryId: saleSummaryId ?? null,
      emissionIndex: i,
    },
  });
}

if (!ownerUserId && guestEmail) {
  await tx.guestTicketLink.upsert({
    where: { ticketId: ticket.id },
    update: {
      guestEmail,
      guestName: guestName || "Convidado",
      guestPhone: guestPhone || null,
    },
    create: {
      ticketId: ticket.id,
      guestEmail,
      guestName: guestName || "Convidado",
      guestPhone: guestPhone || null,
    },
  });
}

createdTicketsCount += 1;
}

await tx.ticketType.update({
  where: { id: ticketType.id },
  data: {
    soldQuantity: { increment: qty },
  },
});
}

if (createdTicketsCount === 0) {
  // Nada criado, marcamos paymentEvent como erro
  await tx.paymentEvent.updateMany({
    where: { stripePaymentIntentId: intent.id },
    data: {
      status: "ERROR",
      errorMessage: "Nenhum bilhete processado (stock/itens inválidos).",
      purchaseId: purchaseAnchor,
      stripeEventId: stripeEventId ?? undefined,
      source: PaymentEventSource.WEBHOOK,
      dedupeKey: purchaseAnchor ?? stripeEventId ?? intent.id,
      attempt: { increment: 1 },
      updatedAt: new Date(),
    },
  });
  return;
}

// Reservas apenas se a compra foi feita com sessão (guest não cria reservas)
if (ownerUserId) {
  await tx.ticketReservation.updateMany({
    where: {

```

```

        eventId: eventRecord.id,
        userId: ownerUserId,
        status: "ACTIVE",
    },
    data: { status: "COMPLETED" },
);
}

await tx.paymentEvent.updateMany({
    where: { stripePaymentIntentId: intent.id },
    data: {
        status: "OK",
        updatedAt: new Date(),
        errorMessage: null,
        purchaseId: purchaseAnchor,
        stripeEventId: stripeEventId ?? undefined,
        source: PaymentEventSource.WEBHOOK,
        dedupeKey: purchaseAnchor ?? stripeEventId ?? intent.id,
        attempt: { increment: 1 },
    },
});
});

if (createdTicketsCount === 0) {
    console.warn("[fulfillPayment] No valid items to process");
    return;
}

console.log("[fulfillPayment] OK, items processados:", {
    intentId: intent.id,
    userId,
    items,
});

// Enviar email de confirmação (best-effort)
const targetEmail = userId ? await fetchUserEmail(userId) : guestEmail || null;
if (targetEmail) {
    const baseUrl =
        process.env.NEXT_PUBLIC_BASE_URL ??
        process.env.NEXT_PUBLIC_APP_URL ??
        "https://app.orya.pt";

    try {
        await sendPurchaseConfirmationEmail({
            to: targetEmail,
            eventTitle: eventRecord.title,
            eventSlug: eventRecord.slug,
            startsAt: eventRecord.startsAt?.toISOString() ?? null,
            endsAt: eventRecord.endsAt?.toISOString() ?? null,
            locationName: eventRecord.locationName ?? null,
            ticketsCount: createdTicketsCount,
            ticketUrl: userId ? `${baseUrl}/me/tickets` : `${baseUrl}/`,
        });
        console.log("[fulfillPayment] Email de confirmação enviado para", targetEmail);
    } catch (emailErr) {
        console.error("[fulfillPayment] Falha ao enviar email de confirmação", emailErr);
    }
} else {
    console.warn("[fulfillPayment] Email do comprador não encontrado para envio de recibo");
}

return;
}

async function fetchUserEmail(userId: string) {
    try {
        const { data, error } = await supabaseAdmin.auth.admin.getUserById(userId);
        if (error) {
            console.warn("[fetchUserEmail] erro ao obter user", error);
            return null;
        }
        return data.user?.email ?? null;
    } catch (err) {
        console.warn("[fetchUserEmail] erro inesperado", err);
        return null;
    }
}

```

```

async function handlePadelSplitPayment(intent: Stripe.PaymentIntent) {
  const meta = intent.metadata ?? {};
  const pairingId = Number(meta.pairingId);
  const slotId = Number(meta.slotId);
  const ticketTypeId = Number(meta.ticketTypeId);
  const eventId = Number(meta.eventId);
  const userId = typeof meta.userId === "string" ? meta.userId : null;
  const purchaseId =
    typeof meta.purchaseId === "string" && meta.purchaseId.trim() !== ""
    ? meta.purchaseId.trim()
    : null;

  if (!Number.isFinite(pairingId) || !Number.isFinite(slotId) || !Number.isFinite(ticketTypeId) || !Number.isFinite(eventId)) {
    console.warn("[handlePadelSplitPayment] metadata incompleta", meta);
    return;
  }

  const ticketType = await prisma.ticketType.findUnique({
    where: { id: ticketTypeId },
    select: { id: true, price: true, currency: true, soldQuantity: true, eventId: true },
  });
  if (!ticketType || ticketType.eventId !== eventId) {
    console.warn("[handlePadelSplitPayment] ticketType inválido", { ticketTypeId, eventId });
    return;
  }

  const qrSecret = crypto.randomUUID();
  const rotatingSeed = crypto.randomUUID();
  await prisma.$transaction(async (tx) => {
    const pairing = await tx.padelPairing.findUnique({
      where: { id: pairingId },
      include: { slots: true },
    });
    if (!pairing || pairing.paymentMode !== PadelPaymentMode.SPLIT) {
      throw new Error("PAIRING_NOT_SPLIT");
    }
    if (pairing.pairingStatus === "CANCELLED") {
      throw new Error("PAIRING_CANCELLED");
    }
    const slot = pairing.slots.find((s) => s.id === slotId);
    if (!slot) throw new Error("SLOT_NOT_FOUND");
    if (slot.paymentStatus === PadelPairingPaymentStatus.PAID) {
      // já processado
      return;
    }

    // cria ticket para o slot
    const ticket = await tx.ticket.create({
      data: {
        eventId,
        ticketTypeId,
        pricePaid: ticketType.price,
        totalPaidCents: intent.amount,
        currency: ticketType.currency || intent.currency.toUpperCase(),
        stripePaymentIntentId: intent.id,
        status: "ACTIVE",
        qrSecret,
        rotatingSeed,
        userId: userId ?? undefined,
        ownerUserId: userId ?? null,
        ownerIdentityId: null,
        pairingId,
        padelSplitShareCents: ticketType.price,
      },
    });

    await tx.ticketType.update({
      where: { id: ticketTypeId },
      data: { soldQuantity: ticketType.soldQuantity + 1 },
    });

    const updated = await tx.padelPairing.update({
      where: { id: pairingId },
      data: {
        slots: {
          update: {
            where: { id: slotId },
            data: {
              soldQuantity: slot.soldQuantity + 1
            }
          }
        }
      }
    });
  });
}

```

```

        ticketId: ticket.id,
        profileId: userId ?? undefined,
        paymentStatus: PadelPairingPaymentStatus.PAID,
        slotStatus: userId ? PadelPairingSlotStatus.FILLED : slot.slotStatus,
    },
},
},
},
include: { slots: true },
});

const stillPending = updated.slots.some((s) => s.slotStatus === "PENDING" || s.paymentStatus === "UNPAID");
if (!stillPending && updated.pairingStatus !== "COMPLETE") {
    const confirmed = await tx.padelPairing.update({
        where: { id: pairingId },
        data: { pairingStatus: "COMPLETE" },
        select: { id: true, player1UserId: true, player2UserId: true },
    });
    await ensureEntriesForConfirmedPairing(confirmed.id);
    const captainUserId = confirmed.player1UserId ?? userId ?? undefined;
    if (captainUserId) {
        await queuePartnerPaid(pairingId, captainUserId, userId ?? undefined);
    }
}

// log payment_event
await tx.paymentEvent.upsert({
    where: { stripePaymentIntentId: intent.id },
    update: {
        status: "OK",
        amountCents: intent.amount,
        eventId,
        userId: userId ?? undefined,
        updatedAt: new Date(),
        errorMessage: null,
        mode: intent.livemode ? "LIVE" : "TEST",
        isTest: !intent.livemode,
        stripeFeeCents: stripeFeeForIntentValue,
        purchaseId: purchaseId ?? intent.id,
        source: PaymentEventSource.WEBHOOK,
        dedupeKey: purchaseId ?? intent.id,
        attempt: { increment: 1 },
    },
    create: {
        stripePaymentIntentId: intent.id,
        status: "OK",
        amountCents: intent.amount,
        eventId,
        userId: userId ?? undefined,
        mode: intent.livemode ? "LIVE" : "TEST",
        isTest: !intent.livemode,
        stripeFeeCents: stripeFeeForIntentValue,
        purchaseId: purchaseId ?? intent.id,
        source: PaymentEventSource.WEBHOOK,
        dedupeKey: purchaseId ?? intent.id,
        attempt: 1,
    },
});
};

async function handleSecondCharge(intent: Stripe.PaymentIntent) {
    const meta = intent.metadata ?? {};
    const pairingId = Number(meta.pairingId);
    if (!Number.isFinite(pairingId)) {
        console.warn("[handleSecondCharge] pairingId ausente no metadata", meta);
        return;
    }
    const now = new Date();

    if (intent.status === "succeeded") {
        await prisma.$transaction(async (tx) => {
            await tx.padelPairingSlot.updateMany({
                where: { pairingId, slotStatus: { in: ["PENDING", "FILLED"] } },
                data: { paymentStatus: PadelPairingPaymentStatus.PAID },
            });
            const confirmed = await tx.padelPairing.update({
                where: { id: pairingId },
            });
        });
    }
}

```

```

    data: {
      lifecycleStatus: PadelPairingLifecycleStatus.CONFIRMED_CAPTAIN_FULL,
      pairingStatus: PadelPairingStatus.COMPLETE,
      guaranteeStatus: "SUCCEEDED",
      secondChargePaymentIntentId: intent.id,
      captainSecondChargedAt: now,
      partnerPaidAt: now,
      graceUntilAt: null,
      partnerInviteToken: null,
      partnerLinkToken: null,
      partnerLinkExpiresAt: null,
    },
  });
  await ensureEntriesForConfirmedPairing(confirmed.id);
  await tx.padelPairingHold.updateMany({
    where: { pairingId, status: "ACTIVE" },
    data: { status: "CANCELLED" },
  });
  await tx.paymentEvent.upsert({
    where: { stripePaymentIntentId: intent.id },
    update: {
      status: "OK",
      updatedAt: now,
      amountCents: intent.amount,
      purchaseId: (meta as Record<string, unknown>)??.purchaseId as string | undefined ?? intent.id,
      stripeFeeCents: 0,
      source: PaymentEventSource.WEBHOOK,
      dedupeKey: (meta as Record<string, unknown>)??.purchaseId as string | undefined ?? intent.id,
      attempt: { increment: 1 },
    },
    create: {
      stripePaymentIntentId: intent.id,
      status: "OK",
      amountCents: intent.amount,
      eventId: Number(meta.eventId) || undefined,
      userId: typeof meta.userId === "string" ? meta.userId : undefined,
      purchaseId: (meta as Record<string, unknown>)??.purchaseId as string | undefined ?? intent.id,
      source: PaymentEventSource.WEBHOOK,
      dedupeKey: (meta as Record<string, unknown>)??.purchaseId as string | undefined ?? intent.id,
      attempt: 1,
      stripeFeeCents: 0,
      mode: intent.livemode ? "LIVE" : "TEST",
      isTest: !intent.livemode,
    },
  });
});
return;
}

if (intent.status === "requires_action") {
  await prisma.padelPairing.update({
    where: { id: pairingId },
    data: {
      guaranteeStatus: "REQUIRES_ACTION",
      graceUntilAt: computeGraceUntil(now),
      secondChargePaymentIntentId: intent.id,
    },
  });
  // Notificar capitão que precisa de ação (SCA)
  const pairing = await prisma.padelPairing.findUnique({ where: { id: pairingId }, select: { player1UserId: true, player2UserId: true } });
  const targets = [pairing?.player1UserId, pairing?.player2UserId].filter(Boolean) as string[];
  if (targets.length) {
    await queueOffsessionActionRequired(pairingId, targets);
  }
  return;
}

if (intent.status === "requires_payment_method" || intent.status === "canceled") {
  await prisma.$transaction(async (tx) => {
    await tx.padelPairing.update({
      where: { id: pairingId },
      data: {
        guaranteeStatus: "FAILED",
        lifecycleStatus: PadelPairingLifecycleStatus.CANCELLED_INCOMPLETE,
        pairingStatus: PadelPairingStatus.CANCELLED,
        graceUntilAt: null,
      },
    });
  });
}

```

```

    });
    await tx.padelPairingHold.updateMany({
      where: { pairingId, status: "ACTIVE" },
      data: { status: "CANCELLED" },
    });
  });

  const pairing = await prisma.padelPairing.findUnique({ where: { id: pairingId }, select: { player1UserId: true, player2UserId: true } });
  const targets = [pairing?.player1UserId, pairing?.player2UserId].filter(Boolean) as string[];
  if (targets.length) {
    await queueDeadlineExpired(pairingId, targets);
  }
  return;
}
}

async function handlePadelFullPayment(intent: Stripe.PaymentIntent) {
  const meta = intent.metadata ?? {};
  const pairingId = Number(meta.pairingId);
  const ticketTypeId = Number(meta.ticketTypeId);
  const eventId = Number(meta.eventId);
  const userId = typeof meta.userId === "string" ? meta.userId : null;

  if (!Number.isFinite(pairingId) || !Number.isFinite(ticketTypeId) || !Number.isFinite(eventId)) {
    console.warn("[handlePadelFullPayment] metadata incompleta", meta);
    return;
  }

  const ticketType = await prisma.ticketType.findUnique({
    where: { id: ticketTypeId },
    select: { id: true, price: true, currency: true, soldQuantity: true, eventId: true },
  });
  if (!ticketType || ticketType.eventId !== eventId) {
    console.warn("[handlePadelFullPayment] ticketType inválido", { ticketTypeId, eventId });
    return;
  }

  const qr1 = crypto.randomUUID();
  const qr2 = crypto.randomUUID();
  const rot1 = crypto.randomUUID();
  const rot2 = crypto.randomUUID();

  await prisma.$transaction(async (tx) => {
    const pairing = await tx.padelPairing.findUnique({
      where: { id: pairingId },
      include: { slots: true },
    });
    if (!pairing || pairing.paymentMode !== PadelPaymentMode.FULL) {
      throw new Error("PAIRING_NOT_FULL");
    }
    if (pairing.pairingStatus === "CANCELLED") {
      throw new Error("PAIRING_CANCELLED");
    }

    const captainSlot = pairing.slots.find((s) => s.slotRole === "CAPTAIN");
    const partnerSlot = pairing.slots.find((s) => s.slotRole === "PARTNER");
    if (!captainSlot || !partnerSlot) throw new Error("SLOTS_INVALID");

    // Cria 2 tickets (capitão e parceiro vazio)
    const ticketCaptain = await tx.ticket.create({
      data: {
        eventId,
        ticketTypeId,
        pricePaid: ticketType.price,
        totalPaidCents: ticketType.price,
        currency: ticketType.currency || intent.currency.toUpperCase(),
        stripePaymentIntentId: intent.id,
        status: "ACTIVE",
        qrSecret: qr1,
        rotatingSeed: rot1,
        userId: userId ?? undefined,
        ownerUserId: userId ?? null,
        ownerIdentityId: null,
        pairingId,
        padelSplitShareCents: ticketType.price,
      },
    });
  });
}

```

```

const ticketPartner = await tx.ticket.create({
  data: {
    eventId,
    ticketTypeId,
    pricePaid: ticketType.price,
    totalPaidCents: ticketType.price,
    currency: ticketType.currency || intent.currency.toUpperCase(),
    stripePaymentIntentId: intent.id,
    status: "ACTIVE",
    qrSecret: qr2,
    rotatingSeed: rot2,
    pairingId,
    padelSplitShareCents: ticketType.price,
    ownerUserId: userId ?? null,
    ownerIdentityId: null,
  },
});

await tx.ticketType.update({
  where: { id: ticketTypeId },
  data: { soldQuantity: ticketType.soldQuantity + 2 },
});

await tx.padelPairing.update({
  where: { id: pairingId },
  data: {
    pairingStatus: "INCOMPLETE",
    slots: {
      update: [
        {
          where: { id: captainSlot.id },
          data: {
            ticketId: ticketCaptain.id,
            profileId: userId ?? captainSlot.profileId ?? undefined,
            paymentStatus: PadelPairingPaymentStatus.PAID,
            slotStatus: PadelPairingSlotStatus.FILLED,
          },
        },
        {
          where: { id: partnerSlot.id },
          data: {
            ticketId: ticketPartner.id,
            paymentStatus: PadelPairingPaymentStatus.PAID,
            slotStatus: PadelPairingSlotStatus.PENDING,
          },
        },
      ],
    },
  },
});

await tx.paymentEvent.upsert({
  where: { stripePaymentIntentId: intent.id },
  update: {
    status: "OK",
    amountCents: intent.amount,
    eventId,
    userId: userId ?? undefined,
    updatedAt: new Date(),
    errorMessage: null,
    mode: intent.livemode ? "LIVE" : "TEST",
    isTest: !intent.livemode,
    purchaseId: purchaseId ?? intent.id,
    source: PaymentEventSource.WEBHOOK,
    dedupeKey: purchaseId ?? intent.id,
    attempt: { increment: 1 },
  },
  create: {
    stripePaymentIntentId: intent.id,
    status: "OK",
    amountCents: intent.amount,
    eventId,
    userId: userId ?? undefined,
    mode: intent.livemode ? "LIVE" : "TEST",
    isTest: !intent.livemode,
    purchaseId: purchaseId ?? intent.id,
    source: PaymentEventSource.WEBHOOK,
    dedupeKey: purchaseId ?? intent.id,
  },
});

```

```

        attempt: 1,
    },
});
});
}
}

export async function handleRefund(charge: Stripe.Charge) {
    const paymentIntentId =
        typeof charge.payment_intent === "string"
            ? charge.payment_intent
            : charge.payment_intent?.id;

    if (!paymentIntentId) {
        console.warn("[handleRefund] charge.refunded sem payment_intent");
        return;
    }

    // Obter metadata do payment intent para identificar PADEL_SPLIT
    const intent = await stripe.paymentIntents.retrieve(paymentIntentId, { expand: ["latest_charge"] }).catch(() => null);

    const paymentScenario = normalizePaymentScenario(
        typeof intent?.metadata?.paymentScenario === "string" ? intent?.metadata?.paymentScenario : null,
    );
    const isPadelSplit = paymentScenario === "GROUP_SPLIT";
    const tickets = await prisma.ticket.findMany({
        where: { stripePaymentIntentId: paymentIntentId },
        select: { id: true, ticketTypeId: true, eventId: true, status: true, pairingId: true },
    });

    if (!tickets.length) {
        console.warn("[handleRefund] Nenhum ticket associado ao payment_intent", paymentIntentId);
        return;
    }

    if (isPadelSplit) {
        await handlePadelSplitRefund(paymentIntentId, tickets, charge.livemode);
        return;
    }

    const byType = tickets.reduce<Record<number, number>>((acc, t) => {
        acc[t.ticketTypeId] = (acc[t.ticketTypeId] ?? 0) + 1;
        return acc;
    }, {});

    const ticketTypeIds = Object.keys(byType).map((id) => Number(id));
    const saleSummary = await prisma.saleSummary.findUnique({
        where: { paymentIntentId },
        select: { id: true, promoCodeId: true },
    });
    const ticketTypes = await prisma.ticketType.findMany({
        where: { id: { in: ticketTypeIds } },
        select: { id: true, soldQuantity: true },
    });

    const stockUpdates = ticketTypes.map((tt) => {
        const decrementBy = byType[tt.id] ?? 0;
        const newSold = Math.max(0, tt.soldQuantity - decrementBy);
        return prisma.ticketType.update({
            where: { id: tt.id },
            data: { soldQuantity: newSold },
        });
    });

    const ticketIds = tickets.map((t) => t.id);

    await prisma.$transaction([
        prisma.ticket.updateMany({
            where: { id: { in: ticketIds } },
            data: { status: "REFUNDED" },
        }),
        ...stockUpdates,
        prisma.paymentEvent.updateMany({
            where: { stripePaymentIntentId: paymentIntentId },
            data: {
                status: "REFUNDED",
                errorMessage: null,
                updatedAt: new Date(),
                mode: charge.livemode ? "LIVE" : "TEST",
            }
        })
    ]);
}

```

```

        isTest: !charge.livemode,
    },
}),
...(saleSummary?.id
? [
    prisma.promoRedemption.updateMany({
        where: { saleSummaryId: saleSummary.id },
        data: { cancelledAt: new Date() },
    }),
    prisma.saleSummary.update({
        where: { id: saleSummary.id },
        data: { status: SaleSummaryStatus.REFUNDED, updatedAt: new Date() },
    }),
]
:[],
[]),
]);
}

console.log("[handleRefund] Tickets marcados como REFUNDED", {
    paymentIntentId,
    ticketCount: tickets.length,
});
}

async function handlePadelSplitRefund(
    paymentIntentId: string,
    tickets: Array<{ id: string; pairingId: number | null }>,
    livemode: boolean,
) {
    const saleSummary = await prisma.saleSummary.findUnique({
        where: { paymentIntentId },
        select: { id: true },
    });
    const pairingIds = Array.from(new Set(tickets.map((t) => t.pairingId).filter(Boolean))) as number[];
    if (!pairingIds.length) return;

    await prisma.$transaction(async (tx) => {
        await tx.ticket.updateMany({
            where: { stripePaymentIntentId: paymentIntentId },
            data: { status: TicketStatus.REFUNDED },
        });

        for (const pairingId of pairingIds) {
            const pairing = await tx.padelPairing.findUnique({
                where: { id: pairingId },
                include: { slots: true },
            });
            if (!pairing) continue;

            const affectedSlots = pairing.slots.filter((s) => tickets.some((t) => t.id === s.ticketId));
            for (const slot of affectedSlots) {
                await tx.padelPairingSlot.update({
                    where: { id: slot.id },
                    data: { slotStatus: PadelPairingSlotStatus.CANCELLED, paymentStatus: PadelPairingPaymentStatus.UNPAID, ticketId: null
                },
            });
        }

        const hasPaid = pairing.slots.some(
            (s) => s.paymentStatus === PadelPairingPaymentStatus.PAID && !tickets.some((t) => t.id === s.ticketId),
        );
        const newStatus = hasPaid ? "INCOMPLETE" : "CANCELLED";

        await tx.padelPairing.update({
            where: { id: pairingId },
            data: {
                pairingStatus: newStatus,
                partnerInviteToken: newStatus === "CANCELLED" ? null : pairing.partnerInviteToken,
                partnerLinkToken: newStatus === "CANCELLED" ? null : pairing.partnerLinkToken,
                partnerLinkExpiresAt: newStatus === "CANCELLED" ? null : pairing.partnerLinkExpiresAt,
            },
        });
    });

    await tx.paymentEvent.updateMany({
        where: { stripePaymentIntentId: paymentIntentId },
        data: {
            status: "REFUNDED",
            updatedAt: new Date(),
        }
    })
}

```

```

        errorMessage: null,
        mode: livemode ? "LIVE" : "TEST",
        isTest: !livemode,
    },
});

if (saleSummary?.id) {
    await tx.saleSummary.update({
        where: { id: saleSummary.id },
        data: { status: SaleSummaryStatus.REFUNDED, updatedAt: new Date() },
    });
}
});
}

```

app/api/tickets/migrate-guest/route.ts

```

// app/api/tickets/migrate-guest/route.ts
export const runtime = "nodejs";

import { NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

function normalizeEmail(email: string | null | undefined) {
    return typeof email === "string" ? email.trim().toLowerCase() : "";
}

export async function POST() {
    try {
        const supabase = await createSupabaseServer();
        const { data: userData, error: authError } = await supabase.auth.getUser();

        if (authError || !userData?.user) {
            return NextResponse.json({ ok: false, error: "NOT_AUTHENTICATED" }, { status: 401 });
        }

        const userId = userData.user.id;
        const userEmail = normalizeEmail(userData.user.email);

        if (!userEmail) {
            return NextResponse.json(
                { ok: false, error: "USER_EMAIL_MISSING" },
                { status: 400 },
            );
        }

        const links = await prisma.guestTicketLink.findMany({
            where: {
                guestEmail: { equals: userEmail, mode: "insensitive" },
                OR: [{ migratedToUserId: null }, { migratedToUserId: { not: userId } }],
            },
            select: { ticketId: true },
        });

        if (links.length === 0) {
            return NextResponse.json({ ok: true, migrated: 0 });
        }

        const ticketIds = links.map((l) => l.ticketId);

        const result = await prisma.$transaction([
            prisma.ticket.updateMany({
                where: { id: { in: ticketIds } },
                data: { userId },
            }),
            prisma.guestTicketLink.updateMany({
                where: { ticketId: { in: ticketIds } },
                data: { migratedToUserId: userId, migratedAt: new Date() },
            }),
        ]);

        const migratedCount = result[0].count;

        return NextResponse.json({ ok: true, migrated: migratedCount });
    } catch (err) {

```

```

    console.error("[migrate-guest] error", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/tickets/resale/cancel/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { TicketStatus, ResaleStatus } from "@prisma/client";

/** 
 * F5-8 – Cancelar revenda
 * Body esperado: { resaleId: string }
 */
export async function POST(req: NextRequest) {
  try {
    // 1. Auth – garantir utilizador autenticado
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error: authError,
    } = await supabase.auth.getUser();

    if (authError) {
      console.error("Error getting user in resale/cancel:", authError);
    }

    if (!user) {
      return NextResponse.json(
        { ok: false, error: "UNAUTHENTICATED" },
        { status: 401 }
      );
    }
    const profile = await prisma.profile.findUnique({
      where: { id: user.id },
      select: { roles: true },
    });
    const roles = Array.isArray(profile?.roles) ? (profile?.roles as string[]) : [];
    const isAdmin = roles.some((r) => r.toLowerCase() === "admin");
    if (!isAdmin) {
      return NextResponse.json(
        { ok: false, error: "FORBIDDEN" },
        { status: 403 },
      );
    }
  }

  const body = (await req.json().catch(() => null)) as
    | { resaleId?: string }
    | null;

  if (!body || typeof body !== "object" || !body.resaleId) {
    return NextResponse.json(
      { ok: false, error: "INVALID_BODY" },
      { status: 400 }
    );
  }

  const { resaleId } = body;
  const userId = user.id;

  // 2. Buscar revenda e garantir que pertence ao utilizador e está LISTED
  const resale = await prisma.ticketResale.findUnique({
    where: { id: resaleId },
    include: {
      ticket: true,
    },
  });

  if (!resale) {
    return NextResponse.json(
      { ok: false, error: "RESALE_NOT_FOUND" },
      { status: 404 }
    );
  }
}

```

```

if (resale.sellerUserId !== userId) {
  return NextResponse.json(
    { ok: false, error: "NOT_RESALE_OWNER" },
    { status: 403 }
  );
}

if (resale.status !== ResaleStatus.LISTED) {
  return NextResponse.json(
    { ok: false, error: "RESALE_NOT_LISTED" },
    { status: 400 }
  );
}

if (!resale.ticket) {
  return NextResponse.json(
    { ok: false, error: "TICKET_NOT_FOUND_FOR_RESALE" },
    { status: 404 }
  );
}

// 3. Transaction – marcar revenda como CANCELLED
//     e (opcional) voltar o ticket a ACTIVE
await prisma.$transaction(async (tx) => {
  await tx.ticketResale.update({
    where: { id: resale.id },
    data: {
      status: ResaleStatus.CANCELLED,
      completedAt: new Date(),
    },
  });

  // Se o bilhete estiver com algum estado específico de revenda,
  // garantimos que volta a ACTIVE. Mesmo que já estivesse ACTIVE,
  // isto não é problemático.
  await tx.ticket.update({
    where: { id: resale.ticketId },
    data: {
      status: TicketStatus.ACTIVE,
    },
  });
});

return NextResponse.json({ ok: true }, { status: 200 });
} catch (error) {
  console.error("Error in /api/tickets/resale/cancel:", error);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 }
  );
}
}

```

app/api/tickets/resale/list/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";

/**
 * F5-7 – Criar revenda (listar bilhete)
 *
 * Body esperado:
 * {
 *   ticketId: string;
 *   price: number; // em cêntimos
 * }
 */
export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error: authError,
    } = await supabase.auth.getUser();
  
```

```

if (authError) {
  console.error("Error getting user in resale/list:", authError);
}

if (!user) {
  return NextResponse.json(
    { ok: false, error: "UNAUTHENTICATED" },
    { status: 401 }
  );
}

const profile = await prisma.profile.findUnique({
  where: { id: user.id },
  select: { roles: true },
});

const roles = Array.isArray(profile?.roles) ? (profile?.roles as string[]) : [];
const isAdmin = roles.some((r) => r?.toLowerCase() === "admin");
if (!isAdmin) {
  return NextResponse.json(
    { ok: false, error: "FORBIDDEN" },
    { status: 403 },
  );
}

const body = (await req.json()).catch(() => null) as
| { ticketId?: string; price?: number }
| null;

if (!body || typeof body !== "object") {
  return NextResponse.json(
    { ok: false, error: "INVALID_BODY" },
    { status: 400 }
  );
}

const { ticketId, price } = body;

if (!ticketId || typeof ticketId !== "string") {
  return NextResponse.json(
    { ok: false, error: "MISSING_TICKET_ID" },
    { status: 400 }
  );
}

if (
  typeof price !== "number" ||
  !Number.isFinite(price) ||
  price <= 0
) {
  return NextResponse.json(
    { ok: false, error: "INVALID_PRICE" },
    { status: 400 }
  );
}

const userId = user.id;

// 1. Validar que o bilhete pertence ao utilizador atual e está ACTIVE
const ticket = await prisma.ticket.findFirst({
  where: {
    id: ticketId,
    userId,
    status: "ACTIVE",
  },
  include: {
    event: {
      include: {
        ticketTypes: true,
      },
    },
  },
});

if (!ticket) {
  return NextResponse.json(
    { ok: false, error: "TICKET_NOT_FOUND_OR_NOT_ACTIVE" },
    { status: 404 }
  );
}

```

```

}

// 2. Verificar se já existe transferência PENDING para este bilhete
const existingPendingTransfer = await prisma.ticketTransfer.findFirst({
  where: {
    ticketId: ticket.id,
    status: "PENDING",
  },
});

if (existingPendingTransfer) {
  return NextResponse.json(
    { ok: false, error: "TRANSFER_ALREADY_PENDING" },
    { status: 400 }
  );
}

// 3. Verificar se o bilhete já está em revenda LISTED
const existingResale = await prisma.ticketResale.findFirst({
  where: {
    ticketId: ticket.id,
    status: "LISTED",
  },
});

if (existingResale) {
  return NextResponse.json(
    { ok: false, error: "TICKET_ALREADY_IN_RESALE" },
    { status: 400 }
  );
}

// 4. Validar configuração de revenda ao nível do evento
const event = ticket.event;
if (event) {
  const resaleMode =
    (event as { resaleMode?: string }).resaleMode ?? "ALWAYS";

  if (resaleMode === "DISABLED") {
    return NextResponse.json(
      { ok: false, error: "RESALE_DISABLED_FOR_EVENT" },
      { status: 400 }
    );
  }

  if (resaleMode === "AFTER SOLD OUT") {
    const ticketTypes = event.ticketTypes ?? [];
    const hasUnlimited = ticketTypes.some(
      (tt) => tt.totalQuantity === null || tt.totalQuantity === undefined,
    );
    const soldOut =
      !hasUnlimited &&
      ticketTypes.length > 0 &&
      ticketTypes.every((tt) => {
        if (tt.totalQuantity === null || tt.totalQuantity === undefined)
          return false;
        return tt.soldQuantity >= tt.totalQuantity;
      });

    if (!soldOut) {
      return NextResponse.json(
        { ok: false, error: "RESALE_ONLY_AFTER SOLD OUT" },
        { status: 400 }
      );
    }
  }
}

// 5. Criar registo em ticket_resales com status LISTED
const resale = await prisma.ticketResale.create({
  data: {
    ticketId: ticket.id,
    sellerUserId: userId,
    price,
    currency: ticket.currency ?? "EUR",
    status: "LISTED",
  },
});

```

```

// (Opcional) Se no futuro adicionares um estado específico no TicketStatus
// para bilhetes em revenda (ex.: RESALE_LISTED), podes atualizar aqui o ticket.

return NextResponse.json(
  {
    ok: true,
    resaleId: resale.id,
    status: resale.status,
  },
  { status: 200 }
);
} catch (error) {
  console.error("Error in /api/tickets/resale/list:", error);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 }
  );
}
}
}

```

app/api/tickets/resale/route.ts

app/api/tickets/scan/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { TicketStatus } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { canScanTickets } from "@/lib/organizerAccess";

type ScanResponseStatus = "OK" | "ALREADY_USED" | "CANCELLED" | "REFUNDED" | "INVALID" | "WRONG_EVENT";

function buildResponse(
  status: ScanResponseStatus,
  message: string,
  ticket?: {
    id: string;
    holderName: string | null;
    ticketTypeName: string | null;
    checkins: number;
    maxCheckins: number;
  },
  timestamps?: { checkedInAt?: Date | null; firstCheckinAt?: Date | null },
  httpStatus = 200,
) {
  return NextResponse.json(
    {
      status,
      message,
      ticket: ticket ?? null,
      checkedInAt: timestamps?.checkedInAt ?? null,
      firstCheckinAt: timestamps?.firstCheckinAt ?? null,
    },
    { status: httpStatus },
  );
}

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error,
    } = await supabase.auth.getUser();

    if (error || !user) {
      return NextResponse.json({ ok: false, error: "UNAUTHENTICATED" }, { status: 401 });
    }

    const body = await req.json().catch(() => null);
    const eventId = Number(body?.eventId);
  }
}

```

```

const ticketCode = typeof body?.ticketCode === "string" ? body.ticketCode.trim() : null;
const deviceId = typeof body?.deviceId === "string" ? body.deviceId.slice(0, 120) : null;

if (!eventId || Number.isNaN(eventId) || !ticketCode) {
  return NextResponse.json({ ok: false, error: "INVALID_PAYLOAD" }, { status: 400 });
}

const access = await canScanTickets(user.id, eventId);
if (!access.allowed) {
  return NextResponse.json({ ok: false, error: "NO_SCAN_PERMISSION", reason: access.reason }, { status: 403 });
}

const ticket = await prisma.ticket.findUnique({
  where: { qrSecret: ticketCode },
  include: {
    ticketType: { select: { name: true } },
    event: { select: { id: true, title: true } },
  },
});

if (!ticket) {
  return buildResponse("INVALID", "Bilhete não encontrado.");
}

if (ticket.eventId !== eventId) {
  return buildResponse("WRONG_EVENT", "O bilhete não pertence a este evento.");
}

if (ticket.status === TicketStatus.REFUNDED) {
  return buildResponse("REFUNDED", "Bilhete reembolsado – entrada não permitida.");
}
if (ticket.status === TicketStatus.RESALE_LISTED || ticket.status === TicketStatus.TRANSFERRED) {
  return buildResponse("INVALID", "Bilhete em transferência/revenda – confirme o estado antes de aceitar.");
}
if (ticket.status === TicketStatus.USED && ticket.usedAt) {
  return buildResponse(
    "ALREADY_USED",
    "Este bilhete já está marcado como usado.",
    {
      id: ticket.id,
      holderName: ticket.user?.fullName ?? null,
      ticketTypeName: ticket.ticketType?.name ?? null,
      checkins: 1,
      maxCheckins: 1,
    },
    { firstCheckinAt: ticket.usedAt, checkedInAt: ticket.usedAt },
  );
}

if (ticket.usedAt) {
  return buildResponse(
    "ALREADY_USED",
    "Este bilhete já foi usado.",
    {
      id: ticket.id,
      holderName: null,
      ticketTypeName: ticket.ticketType?.name ?? null,
      checkins: 1,
      maxCheckins: 1,
    },
    { firstCheckinAt: ticket.usedAt, checkedInAt: ticket.usedAt },
  );
}

const updated = await prisma.ticket.update({
  where: { id: ticket.id },
  data: {
    status: ticket.status === TicketStatus.ACTIVE ? TicketStatus.USED : ticket.status,
    usedAt: new Date(),
  },
});

return buildResponse(
  "OK",
  "Entrada validada.",
  {
    id: ticket.id,
    holderName: null,
  }
);

```

```

        ticketTypeName: ticket.ticketType?.name ?? null,
        checkins: 1,
        maxCheckins: 1,
    },
    { checkedInAt: updated.usedAt, firstCheckinAt: updated.usedAt },
);
} catch (err) {
    console.error("[tickets/scan][POST]", err);
    return NextResponse.json({ ok: false, error: "INTERNAL_ERROR" }, { status: 500 });
}
}

```

app/api/tickets/transfer/respond/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { NotificationType, TransferStatus, TicketStatus } from "@prisma/client";
import { createNotification, shouldNotify } from "@/lib/notifications";

/** 
 * F5-3 – Responder à transferência (aceitar / recusar)
 * Body esperado: { transferId: string, action: "ACCEPT" | "DECLINE" }
 */
export async function POST(req: NextRequest) {
    try {
        const supabase = await createSupabaseServer();
        const {
            data: { user },
            error: authError,
        } = await supabase.auth.getUser();

        if (authError) {
            console.error("Error getting user in transfer/respond:", authError);
        }

        if (!user) {
            return NextResponse.json(
                { ok: false, error: "UNAUTHENTICATED" },
                { status: 401 }
            );
        }

        const profile = await prisma.profile.findUnique({
            where: { id: user.id },
            select: { roles: true, username: true, fullName: true },
        });

        const roles = Array.isArray(profile?.roles) ? (profile?.roles as string[]) : [];
        const isAdmin = roles.some((r) => r?.toLowerCase() === "admin");
        if (!isAdmin) {
            return NextResponse.json(
                { ok: false, error: "FORBIDDEN" },
                { status: 403 },
            );
        }

        const body = await req.json().catch(() => null) as
            | { transferId?: string; action?: "ACCEPT" | "DECLINE" }
            | null;

        if (
            !body ||
            typeof body !== "object" ||
            !body.transferId ||
            (body.action !== "ACCEPT" && body.action !== "DECLINE")
        ) {
            return NextResponse.json(
                { ok: false, error: "INVALID_BODY" },
                { status: 400 }
            );
        }

        const { transferId, action } = body;
        const userId = user.id;

        // 1. Carregar TicketTransfer PENDING para o to_user_id = auth.uid()
        const transfer = await prisma.ticketTransfer.findUnique({

```

```

    where: { id: transferId },
    include: {
      ticket: { include: { event: { select: { id: true, title: true, organizerId: true } } } },
    },
  });

  if (!transfer) {
    return NextResponse.json(
      { ok: false, error: "TRANSFER_NOT_FOUND" },
      { status: 404 }
    );
  }

  if (transfer.toUserId !== userId) {
    return NextResponse.json(
      { ok: false, error: "NOT_TRANSFER_TARGET" },
      { status: 403 }
    );
  }

  if (transfer.status !== TransferStatus.PENDING) {
    return NextResponse.json(
      { ok: false, error: "TRANSFER_NOT_PENDING" },
      { status: 400 }
    );
  }

  // 2. Se for DECLINE -> marcar DECLINED e terminar
  if (action === "DECLINE") {
    const updated = await prisma.ticketTransfer.update({
      where: { id: transfer.id },
      data: {
        status: TransferStatus.CANCELLED,
        completedAt: new Date(),
      },
    });

    if (await shouldNotify(transfer.fromUserId, NotificationType.TICKET_TRANSFER_DECLINED)) {
      const actorName = profile?.username || profile?.fullName || "Um utilizador";
      await createNotification({
        userId: transfer.fromUserId,
        fromUserId: userId,
        organizerId: transfer.ticket.event?.organizerId ?? null,
        eventId: transfer.ticket.event?.id ?? null,
        ticketId: transfer.ticketId,
        type: NotificationType.TICKET_TRANSFER_DECLINED,
        title: "Transferência recusada",
        body: `${actorName} recusou o teu bilhete.`,
        payload: { ticketId: transfer.ticketId, eventId: transfer.ticket.event?.id },
      }).catch((err) => console.warn("[notification][transfer_declined] falhou", err));
    }
  }

  return NextResponse.json(
    {
      ok: true,
      status: updated.status,
    },
    { status: 200 }
  );
}

// 3. Se for ACCEPT -> mudar dono do ticket e marcar ACCEPTED
const ticket = transfer.ticket;

if (!ticket) {
  return NextResponse.json(
    { ok: false, error: "TICKET_NOT_FOUND_FOR_TRANSFER" },
    { status: 404 }
  );
}

if (ticket.status !== TicketStatus.ACTIVE) {
  return NextResponse.json(
    { ok: false, error: "TICKET_NOT_ACTIVE" },
    { status: 400 }
  );
}

```

```

if (ticket.userId !== transfer.fromUserId) {
  // Algo está inconsistente: o dono atual já não é o fromUserId
  return NextResponse.json(
    { ok: false, error: "TICKET_OWNER_MISMATCH" },
    { status: 409 }
  );
}

// Transaction: atualizar ticket + transfer de forma atómica
const result = await prisma.$transaction(async (tx) => {
  const updatedTicket = await tx.ticket.update({
    where: { id: ticket.id },
    data: {
      userId, // novo dono
      status: TicketStatus.ACTIVE, // garante que fica novamente ACTIVE, caso tenhas usado um estado intermédio
    },
  });

  const updatedTransfer = await tx.ticketTransfer.update({
    where: { id: transfer.id },
    data: {
      status: TransferStatus.ACCEPTED,
      completedAt: new Date(),
    },
  });

  return { updatedTicket, updatedTransfer };
});

if (await shouldNotify(transfer.fromUserId, NotificationType.TICKET_TRANSFER_ACCEPTED)) {
  const actorName = profile?.username || profile?.fullName || "Um utilizador";
  await createNotification({
    userId: transfer.fromUserId,
    fromUserId: userId,
    organizerId: transfer.ticket.event?.organizerId ?? null,
    eventId: transfer.ticket.event?.id ?? null,
    ticketId: transfer.ticketId,
    type: NotificationType.TICKET_TRANSFER_ACCEPTED,
    title: "Transferência aceite",
    body: `${actorName} aceitou o bilhete que enviaste.`,
    payload: { ticketId: transfer.ticketId, eventId: transfer.ticket.event?.id },
  }).catch((err) => console.warn("[notification][transfer_accepted] falhou", err));
}

return NextResponse.json(
{
  ok: true,
  status: result.updatedTransfer.status,
  ticketId: result.updatedTicket.id,
},
{ status: 200 }
);
} catch (error) {
  console.error("Error in /api/tickets/transfer/respond:", error);
  return NextResponse.json(
    { ok: false, error: "INTERNAL_ERROR" },
    { status: 500 }
  );
}
}
}

```

app/api/tickets/transfer/start/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { TicketStatus, TransferStatus, NotificationType } from "@prisma/client";
import { createNotification, shouldNotify } from "@/lib/notifications";

export async function POST(req: NextRequest) {
  try {
    const supabase = await createSupabaseServer();
    const {
      data: { user },
      error: authError,
    } = await supabase.auth.getUser();
  
```

```

if (authError || !user) {
  return NextResponse.json(
    { ok: false, error: "NOT_AUTHENTICATED" },
    { status: 401 }
  );
}

const profile = await prisma.profile.findUnique({
  where: { id: user.id },
  select: { roles: true, username: true, fullName: true },
});

const roles = Array.isArray(profile?.roles) ? (profile?.roles as string[]) : [];
const isAdmin = roles.some((r) => r?.toLowerCase() === "admin");
if (!isAdmin) {
  return NextResponse.json(
    { ok: false, error: "FORBIDDEN" },
    { status: 403 }
  );
}

const body = (await req.json().catch(() => null)) as
  | { ticketId?: string; targetIdentifier?: string }
  | null;

if (!body || typeof body !== "object") {
  return NextResponse.json(
    { ok: false, error: "INVALID_BODY" },
    { status: 400 }
  );
}

const { ticketId, targetIdentifier } = body;

if (!ticketId || typeof ticketId !== "string") {
  return NextResponse.json(
    { ok: false, error: "MISSING_TICKET_ID" },
    { status: 400 }
  );
}

if (!targetIdentifier || typeof targetIdentifier !== "string") {
  return NextResponse.json(
    { ok: false, error: "MISSING_TARGET_IDENTIFIER" },
    { status: 400 }
  );
}

const userId = user.id;

// 1. Validar que o bilhete pertence ao utilizador atual e está ACTIVE
const ticket = await prisma.ticket.findFirst({
  where: {
    id: ticketId,
    userId,
    status: TicketStatus.ACTIVE,
  },
  include: {
    event: true,
  },
});

if (!ticket) {
  return NextResponse.json(
    { ok: false, error: "TICKET_NOT_FOUND_OR_NOT_ACTIVE" },
    { status: 404 }
  );
}

// 2. Verificar se já existe transferência PENDING para este bilhete
const existingPendingTransfer = await prisma.ticketTransfer.findFirst({
  where: {
    ticketId: ticket.id,
    status: TransferStatus.PENDING,
  },
});

if (existingPendingTransfer) {
  return NextResponse.json(

```

```

        { ok: false, error: "TRANSFER_ALREADY_PENDING" },
        { status: 400 }
    );
}

// 3. Verificar se o bilhete está listado em revenda
const existingResale = await prisma.ticketResale.findFirst({
    where: {
        ticketId: ticket.id,
        status: "LISTED", // aqui podes manter string se o enum tiver outro nome
    },
});

if (existingResale) {
    return NextResponse.json(
        { ok: false, error: "TICKET_ALREADY_IN_RESALE" },
        { status: 400 }
    );
}

// 4. Resolver o target pelo username ORYA (profiles.username)
const targetProfile = await prisma.profile.findUnique({
    where: {
        username: targetIdentifier,
    },
});

if (!targetProfile) {
    return NextResponse.json(
        { ok: false, error: "TARGET_NOT_FOUND" },
        { status: 404 }
    );
}

if (targetProfile.id === userId) {
    return NextResponse.json(
        { ok: false, error: "CANNOT_TRANSFER_TO_SELF" },
        { status: 400 }
    );
}

// TODO(followers): quando existir tabela de seguidores, garantir follow mútuo
// Exemplo esperado:
// const isFriend = await isMutualFollower(userId, targetProfile.id);
// if (!isFriend) return NextResponse.json({ ok: false, error: "NOT_FRIEND" }, { status: 403 });
// Por agora, esta regra fica desativada até termos o modelo de followers.

// 5. Criar registo em ticket_transfers com status PENDING
const transfer = await prisma.ticketTransfer.create({
    data: {
        ticketId: ticket.id,
        fromUserId: userId,
        toUserId: targetProfile.id,
        status: TransferStatus.PENDING,
    },
});

if (await shouldNotify(targetProfile.id, NotificationType.TICKET_TRANSFER RECEIVED)) {
    await createNotification({
        userId: targetProfile.id,
        fromUserId: userId,
        organizerId: ticket.event?.organizerId ?? null,
        eventId: ticket.eventId,
        ticketId: ticket.id,
        type: NotificationType.TICKET_TRANSFER RECEIVED,
        title: "Tens um bilhete à tua espera",
        body: `Aceita o bilhete para ${ticket.event?.title ?? "este evento"}.`,
        payload: {
            ticketId: ticket.id,
            eventId: ticket.eventId,
            actor: { id: user.id, username: profile?.username, fullName: profile?.fullName },
        },
    }).catch((err) => console.warn("[notification][ticket_transfer_received] falhou", err));
}

return NextResponse.json(
    {
        ok: true,
    }
);

```

```

        transferId: transfer.id,
        status: transfer.status,
    },
    { status: 200 }
);
} catch (error) {
    console.error("Error in /api/tickets/transfer/start:", error);
    return NextResponse.json(
        { ok: false, error: "INTERNAL_ERROR" },
        { status: 500 }
    );
}
}
}

```

app/api/tournaments/[id]/live/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { prisma } from "@/lib/prisma";
import { getTournamentStructure, summarizeMatchStatus, computeStandingsForGroup } from "@/domain/tournaments/structure";

export async function GET(req: NextRequest, { params }: { params: { id: string } }) {
    const slug = params?.id;
    if (!slug) return NextResponse.json({ ok: false, error: "INVALID_SLUG" }, { status: 400 });

    const supabase = await createSupabaseServer();
    const { data: authData } = await supabase.auth.getUser();
    const userId = authData?.user?.id ?? null;

    const event = await prisma.event.findUnique({
        where: { slug },
        select: { id: true, title: true, startsAt: true, endsAt: true, status: true, tournament: { select: { id: true, format: true, generationSeed: true, tieBreakRules: true } } },
    });
    if (!event?.tournament) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

    const structure = await getTournamentStructure(event.tournament.id);
    if (!structure) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

    let userPairingId: number | null = null;
    if (userId) {
        const pairing = await prisma.padelPairing.findFirst({
            where: { eventId: event.id, OR: [{ player1UserId: userId }, { player2UserId: userId }] },
            select: { id: true },
        });
        userPairingId = pairing?.id ?? null;
    }

    const tieBreakRules = Array.isArray(structure.tieBreakRules)
        ? (structure.tieBreakRules as string[])
        : ["WINS", "SET_DIFF", "GAME_DIFF", "HEAD_TO_HEAD", "RANDOM"];

    const stages = structure.stages.map((s) => ({
        id: s.id,
        name: s.name,
        stageType: s.stageType,
        groups: s.groups.map((g) => ({
            id: g.id,
            name: g.name,
            standings: computeStandingsForGroup(g.matches, tieBreakRules, structure.generationSeed || undefined),
            matches: g.matches.map((m) => ({
                id: m.id,
                pairing1Id: m.pairing1Id,
                pairing2Id: m.pairing2Id,
                round: m.round,
                startAt: m.startAt,
                status: m.status,
                statusLabel: summarizeMatchStatus(m.status),
            })),
        })),
        matches: s.matches
            .filter((m) => !m.groupId)
            .map((m) => ({
                id: m.id,
                pairing1Id: m.pairing1Id,
                pairing2Id: m.pairing2Id,
            })),
    }));
}

```

```

        round: m.round,
        startAt: m.startAt,
        status: m.status,
        statusLabel: summarizeMatchStatus(m.status),
    })),
}),
};

const flat = stages.flatMap((s) => [...s.matches, ...s.groups.flatMap((g) => g.matches)]);
const nextMatch =
    userPairingId !== null
    ? flat
        .filter((m) => (m.pairing1Id === userPairingId || m.pairing2Id === userPairingId) && m.status !== "DONE")
        .sort((a, b) => (a.startAt && b.startAt ? new Date(a.startAt).getTime() - new Date(b.startAt).getTime() : 0))[0] ??
null
    : null;
const lastMatch =
    userPairingId !== null
    ? flat
        .filter((m) => (m.pairing1Id === userPairingId || m.pairing2Id === userPairingId) && m.status === "DONE")
        .sort((a, b) => (a.startAt && b.startAt ? new Date(b.startAt).getTime() - new Date(a.startAt).getTime() : 0))[0] ??
null
    : null;

const res = NextResponse.json(
{
    ok: true,
    event: { id: event.id, title: event.title, startsAt: event.startsAt, endsAt: event.endsAt, status: event.status },
    tournament: {
        id: structure.id,
        format: structure.format,
        stages,
        userPairingId,
        nextMatch,
        lastMatch,
    },
},
{ status: 200 },
);
res.headers.set("Cache-Control", "public, max-age=10");
return res;
}

```

app/api/tournaments/[id]/monitor/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { getTournamentStructure, summarizeMatchStatus, computeStandingsForGroup } from "@/domain/tournaments/structure";

export async function GET(_req: NextRequest, { params }: { params: { id: string } }) {
    const id = Number(params.id);
    if (!Number.isFinite(id)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

    const data = await getTournamentStructure(id);
    if (!data) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

    const tieBreakRules = Array.isArray(data.tieBreakRules)
        ? (data.tieBreakRules as string[])
        : ["WINS", "SET_DIFF", "GAME_DIFF", "HEAD_TO_HEAD", "RANDOM"];

    const stages = data.stages.map((s) => ({
        id: s.id,
        name: s.name,
        stageType: s.stageType,
        groups: s.groups.map((g) => ({
            id: g.id,
            name: g.name,
            standings: computeStandingsForGroup(g.matches, tieBreakRules, data.generationSeed || undefined),
            matches: g.matches.map((m) => ({
                id: m.id,
                pairing1Id: m.pairing1Id,
                pairing2Id: m.pairing2Id,
                round: m.round,
                startAt: m.startAt,
                status: m.status,
                statusLabel: summarizeMatchStatus(m.status),
            })),
        })),
    }));
}

```

```

    matches: s.matches
      .filter((m) => !m.groupId)
      .map((m) => ({
        id: m.id,
        pairing1Id: m.pairing1Id,
        pairing2Id: m.pairing2Id,
        round: m.round,
        startAt: m.startAt,
        status: m.status,
        statusLabel: summarizeMatchStatus(m.status),
      })),
    )),
  ));

const res = NextResponse.json(
{
  ok: true,
  tournament: {
    id: data.id,
    event: data.event,
    format: data.format,
    stages,
  },
},
{ status: 200 },
);
res.headers.set("Cache-Control", "public, max-age=10");
return res;
}

```

app/api/tournaments/[id]/structure/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { getTournamentStructure, summarizeMatchStatus, computeStandingsForGroup } from "@/domain/tournaments/structure";

export async function GET(_req: NextRequest, { params }: { params: { id: string } }) {
  const id = Number(params.id);
  if (!Number.isFinite(id)) return NextResponse.json({ ok: false, error: "INVALID_ID" }, { status: 400 });

  const data = await getTournamentStructure(id);
  if (!data) return NextResponse.json({ ok: false, error: "NOT_FOUND" }, { status: 404 });

  const tieBreakRules = Array.isArray(data.tieBreakRules)
    ? (data.tieBreakRules as string[])
    : ["WINS", "SET_DIFF", "GAME_DIFF", "HEAD_TO_HEAD", "RANDOM"];

  const payload = {
    id: data.id,
    event: data.event,
    format: data.format,
    stages: data.stages.map((s) => ({
      id: s.id,
      name: s.name,
      stageType: s.stageType,
      groups: s.groups.map((g) => ({
        id: g.id,
        name: g.name,
        standings: computeStandingsForGroup(g.matches, tieBreakRules),
        matches: g.matches.map((m) => ({
          id: m.id,
          pairing1Id: m.pairing1Id,
          pairing2Id: m.pairing2Id,
          round: m.round,
          startAt: m.startAt,
          status: m.status,
          statusLabel: summarizeMatchStatus(m.status),
        })),
      })),
      matches: s.matches
        .filter((m) => !m.groupId)
        .map((m) => ({
          id: m.id,
          pairing1Id: m.pairing1Id,
          pairing2Id: m.pairing2Id,
          round: m.round,
          startAt: m.startAt,
          status: m.status,
        })),
    })),
  };
}

export default function handler(req: NextRequest, res: NextResponse) {
  return handleRequest(req, res, getTournamentStructure);
}

```

```

        statusLabel: summarizeMatchStatus(m.status),
    )),
}),
};

const res = NextResponse.json({ ok: true, tournament: payload }, { status: 200 });
res.headers.set("Cache-Control", "public, max-age=10");
return res;
}

```

app/api/upload/route.ts

```

// app/api/upload/route.ts
import { NextRequest, NextResponse } from "next/server";
import { promises as fs } from "fs";
import path from "path";
import crypto from "crypto";

export async function POST(req: NextRequest) {
  try {
    const formData = await req.formData();
    const file = formData.get("file") as File | null;

    if (!file) {
      return NextResponse.json(
        { error: "Nenhum ficheiro enviado." },
        { status: 400 }
      );
    }

    if (!file.type.startsWith("image/")) {
      return NextResponse.json(
        { error: "Só são permitidas imagens." },
        { status: 400 }
      );
    }

    const bytes = await file.arrayBuffer();
    const buffer = Buffer.from(bytes);

    const uploadDir = path.join(process.cwd(), "public", "uploads");
    await fs.mkdir(uploadDir, { recursive: true });

    const ext = file.type.split("/")[1] || "png";
    const randomName = crypto.randomBytes(16).toString("hex");
    const filename = `${Date.now()}-${randomName}.${ext}`;

    const filepath = path.join(uploadDir, filename);
    await fs.writeFile(filepath, buffer);

    const url = `/uploads/${filename}`; // acessível em http://localhost:3000/uploads/ficheiro.png

    return NextResponse.json({ url }, { status: 201 });
  } catch (err) {
    console.error("[POST /api/upload]", err);
    return NextResponse.json(
      { error: "Erro ao fazer upload da imagem." },
      { status: 500 }
    );
  }
}

```

app/api/username/check/route.ts

```

import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/prisma";
import { normalizeAndValidateUsername } from "@/lib/globalUsernames";

export async function GET(req: NextRequest) {
  const { searchParams } = new URL(req.url);
  const usernameParam = searchParams.get("username") ?? "";

  const validated = normalizeAndValidateUsername(usernameParam);
  if (!validated.ok) {

```

```

    return NextResponse.json({ ok: false, error: validated.error }, { status: 400 });
}

const existing = await prisma.globalUsername.findUnique({
  where: { username: validated.username },
  select: { ownerType: true, ownerId: true },
});

return NextResponse.json(
{
  ok: true,
  username: validated.username,
  available: !existing,
  takenBy: existing ?? null,
},
{ status: 200 },
);
}

```

app/auth/callback/page.tsx

```

"use client";

import { Suspense, useEffect, useState } from "react";
import { useRouter, useSearchParams } from "next/navigation";
import { supabaseBrowser } from "@/lib/supabaseBrowser";

function AuthCallbackInner() {
  const router = useRouter();
  const searchParams = useSearchParams();
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    let cancelled = false;
    async function handleCallback() {
      try {
        // Garante que a sessão está escrita no browser
        await supabaseBrowser.auth.getSession();

        let target =
          searchParams.get("redirect") ||
          (typeof window !== "undefined"
            ? window.localStorage.getItem("orya_post_auth_redirect") || "/"
            : "/");
      }

      if (!target || target === "") target = "/";

      // Migração de bilhetes guest para user recém autenticado (best-effort, assíncrono)
      fetch("/api/tickets/migrate-guest", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
      }).catch((mErr) => console.warn("[auth/callback] migrate-guest falhou", mErr));

      if (cancelled) return;
      router.replace(target);
    } catch (err) {
      console.error("[auth/callback] erro", err);
      if (!cancelled) {
        setError("Não foi possível concluir a autenticação. Tenta novamente.");
      }
    } finally {
      if (typeof window !== "undefined") {
        try {
          window.localStorage.removeItem("orya_post_auth_redirect");
        } catch {}
      }
    }
  });

  handleCallback();
  return () => {
    cancelled = true;
  };
}, [router, searchParams]);

if (error) {

```

```

    return (
      <main className="min-h-screen flex items-center justify-center text-white">
        <p className="text-sm text-white/80">{error}</p>
      </main>
    );
  }

  return (
    <main className="min-h-screen flex items-center justify-center text-white">
      <p className="text-sm text-white/60">A concluir login...</p>
    </main>
  );
}

export default function AuthCallbackPage() {
  return (
    <Suspense
      fallback={
        <main className="min-h-screen flex items-center justify-center text-white">
          <p className="text-sm text-white/60">A concluir login...</p>
        </main>
      }
    >
      <AuthCallbackInner />
    </Suspense>
  );
}

```

app/bilhete/[id]/page.tsx

```

// app/bilhete/[id]/page.tsx
import { notFound, redirect } from "next/navigation";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import TicketLiveQr from "@/app/components/tickets/TicketLiveQr";

export default async function TicketPage({ params }: { params: Promise<{ id: string }> }) {
  const { id } = await params;

  const supabase = await createSupabaseServer();
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    redirect("/login?redirectTo=/me/tickets");
  }

  const ticket = await prisma.ticket.findFirst({
    where: { id, userId: user.id },
    include: {
      event: {
        select: {
          title: true,
          slug: true,
          startsAt: true,
          locationCity: true,
          locationName: true,
        },
      },
      ticketType: {
        select: {
          name: true,
        },
      },
    },
  });

  if (!ticket) {
    notFound();
  }

  const event = ticket.event;
  const dateLabel = event?.startsAt
    ? new Date(event.startsAt).toLocaleString("pt-PT", {

```

```

        weekday: "short",
        day: "2-digit",
        month: "short",
        hour: "2-digit",
        minute: "2-digit",
    })
: "Data a confirmar";

return (
<main className="min-h-screen orya-body-bg text-white">
<section className="max-w-4xl mx-auto px-5 py-8 space-y-6">
<div className="flex items-center justify-between">
<div>
<p className="text-xs text-white/60 uppercase tracking-[0.2em]">Bilhete ORYA</p>
<h1 className="text-2xl font-semibold">{event?.title ?? "Evento ORYA"}`</h1>
<p className="text-sm text-white/70">
    {dateLabel} · {event?.locationName || "Local a anunciar"}
    {event?.locationCity ? ` , ${event.locationCity}` : ""}
</p>
{ticket.ticketType?.name && (
    <p className="text-xs text-white/60 mt-1">Tipo: {ticket.ticketType.name}</p>
)}
</div>
<a
    href={event?.slug ? `/eventos/${event.slug}` : "/me/tickets"}
    className="text-[11px] text-white/70 hover:text-white underline underline-offset-4"
>
    Voltar
</a>
</div>
<div className="rounded-3xl border border-white/10 bg-white/[0.04] p-6 shadow-[0_18px_60px_rgba(0,0,0,0.8)] flex flex-col items-center gap-4">
    {ticket.qrSecret ? (
        <TicketLiveQr qrToken={ticket.qrSecret} />
    ) : (
        <p className="text-sm text-white/70">QR code ainda não está disponível para este bilhete.</p>
    )
</div>

<p className="text-[11px] text-white/50 text-center">
    Mostra este QR à entrada. Atualiza a cada 15 segundos para tua segurança.
</p>
</section>
</main>
);
}

```

app/components/charts/SalesLineChart.tsx

```

"use client";

import { useId, useMemo, useState } from "react";

type InputPoint = {
    date: string | Date;
    value: number;
    grossCents?: number;
    discountCents?: number;
    platformFeeCents?: number;
    netCents?: number;
};

type Point = {
    date: Date;
    value: number;
    grossCents?: number;
    discountCents?: number;
    platformFeeCents?: number;
    netCents?: number;
};

type Props = {
    data: InputPoint[];
    unit?: "eur" | "tickets";
    periodLabel?: string;
    metricLabel?: string;
}

```

```

accentColor?: string;
className?: string;
rangeDays?: number | "all";
startDate?: Date;
endDate?: Date;
};

/** 
 * Gráfico de linha/área responsivo, sem overflow fora do card.
 * Usa viewBox normalizada (0..100) e preserveAspectRatio default para manter proporções dentro do container.
 */
export function SalesLineChart({
  data,
  unit = "eur",
  periodLabel,
  metricLabel,
  accentColor = "#6BFFFF",
  className,
  rangeDays,
  startDate,
  endDate,
}: Props) {
  const gradientId = useId();
  const [hoverIdx, setHoverIdx] = useState<number | null>(null);

  const processed: Point[] = useMemo(() => {
    return data.map((d) => ({
      date: typeof d.date === "string" ? new Date(d.date) : d.date,
      value: Number.isFinite(d.value) ? d.value : 0,
      grossCents: d.grossCents ?? undefined,
      discountCents: d.discountCents ?? undefined,
      platformFeeCents: d.platformFeeCents ?? undefined,
      netCents: d.netCents ?? undefined,
    }));
  }, [data]);

  // Preencher datas em falta para ter linha contínua desde o início do período
  const derivedEnd =
    endDate ??
    (processed.length ? processed.reduce((latest, p) => (p.date > latest ? p.date : latest), processed[0].date) : new Date());
  const derivedStart = () => {
    if (startDate) return startDate;
    if (rangeDays && rangeDays !== "all") {
      const d = new Date(derivedEnd);
      d.setHours(0, 0, 0);
      d.setDate(d.getDate() - (rangeDays - 1));
      return d;
    }
    if (processed.length) {
      return processed.reduce((earliest, p) => (p.date < earliest ? p.date : earliest), processed[0].date);
    }
    const d = new Date(derivedEnd);
    d.setDate(d.getDate() - 6);
    return d;
  }();
}

const dateMap = new Map<
  string,
  {
    value: number;
    grossCents?: number;
    discountCents?: number;
    platformFeeCents?: number;
    netCents?: number;
  }
>();
processed.forEach((p) => {
  const key = p.date.toISOString().slice(0, 10);
  const prev = dateMap.get(key) ?? { value: 0, grossCents: 0, discountCents: 0, platformFeeCents: 0, netCents: 0 };
  dateMap.set(key, {
    value: prev.value + p.value,
    grossCents: (prev.grossCents ?? 0) + (p.grossCents ?? 0),
    discountCents: (prev.discountCents ?? 0) + (p.discountCents ?? 0),
    platformFeeCents: (prev.platformFeeCents ?? 0) + (p.platformFeeCents ?? 0),
    netCents: (prev.netCents ?? 0) + (p.netCents ?? p.value),
  });
});

```

```

const filled: Point[] = [];
const cursor = new Date(derivedStart);
cursor.setHours(0, 0, 0, 0);
const endCursor = new Date(derivedEnd);
endCursor.setHours(0, 0, 0, 0);
while (cursor <= endCursor) {
  const key = cursor.toISOString().slice(0, 10);
  const entry = dateMap.get(key);
  filled.push({
    date: new Date(cursor),
    value: entry?.value ?? 0,
    grossCents: entry?.grossCents ?? 0,
    discountCents: entry?.discountCents ?? 0,
    platformFeeCents: entry?.platformFeeCents ?? 0,
    netCents: entry?.netCents ?? entry?.value ?? 0,
  });
  cursor.setDate(cursor.getDate() + 1);
}

const values = filled.map((p) => p.value);
const maxVal = values.length ? Math.max(...values, 0) : 0;
const paddedMax = maxVal <= 0 ? 10 : Math.ceil(maxVal * 1.2) + 1; // 20% headroom
const minVal = 0;

// viewBox mais achatado para evitar distorção de texto/linha
const viewW = 100;
const viewH = 50;
const padX = 8;
const padY = 8;
const innerW = viewW - padX * 2;
const innerH = viewH - padY * 2;

const minTime = derivedStart.getTime();
const maxTime = endCursor.getTime() === minTime ? endCursor.getTime() + 24 * 3600 * 1000 : endCursor.getTime();

const toXY = (p: Point, index: number) => {
  const t = p.date.getTime();
  const ratioX = (t - minTime) / (maxTime - minTime || 1);
  const x = padX + ratioX * innerW;
  const ratio = paddedMax === minVal ? 0 : (p.value - minVal) / (paddedMax - minVal);
  const y = viewH - padY - ratio * innerH;
  return { x, y };
};

const path = filled.length
? filled
  .map((p, i) => {
    const { x, y } = toXY(p, i);
    return `${i === 0 ? "M" : "L"} ${x.toFixed(2)} ${y.toFixed(2)} `;
  })
  .join(" ")
: "";

const baseXStart = padX;
const baseXEnd = padX + innerW;
const baseY = viewH - padY;
const firstPoint = filled.length ? toXY(filled[0], 0) : { x: baseXStart, y: baseY };
const lastPoint = filled.length ? toXY(filled[filled.length - 1], filled.length - 1) : { x: baseXEnd, y: baseY };
const areaPath = filled.length
? `${path} L ${lastPoint.x.toFixed(2)} ${baseY.toFixed(2)} L ${firstPoint.x.toFixed(2)} ${baseY.toFixed(2)} Z`
: "";

const last = filled[filled.length - 1];
const lastLabel = () => {
  if (!last) return "-";
  if (unit === "tickets") return `${last.value} bilhetes`;
  return `${last.value.toFixed(2)} €`;
}();

const formatDate = (d: Date) =>
d.toLocaleDateString("pt-PT", {
  day: "2-digit",
  month: "short",
});

const positions = useMemo(() => {
  return filled.map((p, idx) => {
    const { x, y } = toXY(p, idx);
  });
}

```

```

        return { x, y };
    });
}, [filled]);
}

const handleMove = (evt: React.MouseEvent<SVGSVGELEMENT, MouseEvent>) => {
    const rect = (evt.currentTarget as SVGSVGELEMENT).getBoundingClientRect();
    const relX = ((evt.clientX - rect.left) / rect.width) * viewW;
    if (!positions.length) return;
    let nearestIdx = 0;
    let nearestDist = Math.abs(relX - positions[0].x);
    for (let i = 1; i < positions.length; i++) {
        const dist = Math.abs(relX - positions[i].x);
        if (dist < nearestDist) {
            nearestDist = dist;
            nearestIdx = i;
        }
    }
    setHoverIdx(nearestIdx);
};

const handleLeave = () => setHoverIdx(null);

const activePoint = hoverIdx != null ? filled[hoverIdx] : filled[filled.length - 1];
const activePos = hoverIdx != null ? positions[hoverIdx] : positions[positions.length - 1];
const toEuros = (cents?: number) => `${(cents ?? 0) / 100}.toFixed(2)} €`;

return (
    <div className={`relative w-full h-full overflow-hidden ${className ?? ""}`}>
        <svg
            className="absolute inset-0 h-full w-full"
            viewBox={`0 0 ${viewW} ${viewH}`}
            preserveAspectRatio="none"
            role="img"
            aria-label={`${metricLabel ?? "Gráfico"} ${periodLabel ?? ""} · Último ${lastLabel}`}
            onMouseMove={handleMove}
            onMouseLeave={handleLeave}
        >
            <defs>
                <linearGradient id={`${gradientId}-fill`} x1="0" y1="0" x2="0" y2="1">
                    <stop offset="0%" stopColor={accentColor} stopOpacity="0.22" />
                    <stop offset="100%" stopColor={accentColor} stopOpacity="0.02" />
                </linearGradient>
            </defs>
            {filled.length > 0 && (
                <>
                    <path d={areaPath} fill={`url(#${gradientId}-fill)`} stroke="none" />
                    <path d={path} fill="none" stroke={accentColor} strokeWidth={1} strokeLinecap="round" />
                {filled.length > 0 && (
                    () => {
                        const lastIdx = filled.length - 1;
                        const { x, y } = toXY(filled[lastIdx], lastIdx);
                        return (
                            <g>
                                <circle cx={x} cy={y} r={1} fill={accentColor} opacity={0.2} />
                                <circle cx={x} cy={y} r={1} fill={accentColor} stroke={accentColor} strokeWidth={0.4} />
                            </g>
                        );
                    }
                )()
            )}
            </>
        )
    </>
)
}

{filled.length >= 1 && (
    <g fontSize="2.4" fontFamily="Inter, system-ui, sans-serif" fill="#ffffff">
        {filled.map((p, idx) => {
            const total = filled.length;
            const step = total > 14 ? Math.ceil(total / 8) : 1;
            if (total > 14 && idx % step !== 0 && idx !== total - 1 && idx !== 0) return null;
            const { x } = toXY(p, idx);
            return (
                <g key={`${p.date.toISOString()}-tick`}>
                    <line x1={x} x2={x} y1={viewH - 6} y2={viewH - 3} stroke="rgba(255,255,255,0.3)" strokeWidth="0.3" />
                    <text x={x} y={viewH - 1} textAnchor="middle" fill="#ffffff">
                        {formatDate(p.date)}
                    </text>
                </g>
            );
        });
    </g>
)
}

```

```

        })}
      </g>
    )}
</svg>
{activePoint && activePos && (
  <div
    className="pointer-events-none absolute rounded-xl border border-white/10 bg-black/80 px-3 py-2 text-[11px] text-white/80 shadow-lg backdrop-blur"
    style={{
      left: `${(activePos.x / viewW) * 100}%`,
      top: `${(activePos.y / viewH) * 100}%`,
      transform: "translate(-50%, -110%)",
      minWidth: 140,
    }}
  >
  <div className="text-white font-semibold">
    {formatDate(activePoint.date)} · {unit === "tickets" ? `${activePoint.value} bilhetes` :
`${activePoint.value.toFixed(2)} €`}
  </div>
  {unit !== "tickets" && (
    <div className="mt-1 space-y-0.5">
      <div className="flex justify-between gap-3">
        <span>Bruto</span>
        <span>{toEuros(activePoint.grossCents)}</span>
      </div>
      <div className="flex justify-between gap-3">
        <span>Desconto</span>
        <span>{toEuros(activePoint.discountCents)}</span>
      </div>
      <div className="flex justify-between gap-3">
        <span>Taxes</span>
        <span>{toEuros(activePoint.platformFeeCents)}</span>
      </div>
      <div className="flex justify-between gap-3 text-white">
        <span>Líquido</span>
        <span>{toEuros(activePoint.netCents ?? activePoint.value * 100)}</span>
      </div>
    </div>
  )}
</div>
);
}
}

```

app/components/checkout/contextoCheckout.tsx

```

"use client";

import React, { createContext, useContext, useMemo, useState, useCallback } from "react";

export type DadosCheckout = {
  slug: string;
  ticketId: string;
  quantity: number;
  price: number | null;
  ticketName: string | null;
  eventId: string | null;
  userId: string | null;
  waves?: unknown[];
  additional?: Record<string, unknown>;
  paymentScenario?: string | null;
};

export type CheckoutBreakdown = {
  lines: {
    ticketTypeId: number;
    name: string;
    quantity: number;
    unitPriceCents: number;
    currency: string;
    lineTotalCents: number;
  }[];
  subtotalCents: number;
  feeMode: string | null;
  platformFeeCents: number;
};

```

```

totalCents: number;
currency: string;
discountCents?: number;
feeBpsApplied?: number;
feeFixedApplied?: number;
} | null;

type CheckoutContextType = {
  isOpen: boolean;
  passo: 1 | 2 | 3;
  dados: DadosCheckout | null;
  breakdown: CheckoutBreakdown;
  abrirCheckout: (params: {
    slug: string;
    ticketId: string;
    quantity?: number;
    price?: number | null;
    ticketName?: string | null;
    eventId?: string | null;
    userId?: string | null;
    waves?: unknown[];
  }) => void;
  fecharCheckout: () => void;
  irParaPasso: (passo: 1 | 2 | 3) => void;
  atualizarDados: (patch: Partial<DadosCheckout>) => void;
  setBreakdown: (b: CheckoutContextType["breakdown"]) => void;
};

const CheckoutContext = createContext<CheckoutContextType | undefined>(
  undefined,
);

export function CheckoutProvider({ children }: { children: React.ReactNode }) {
  const [isOpen, setIsOpen] = useState(false);
  const [passo, setPasso] = useState<1 | 2 | 3>(1);
  const [dados, setDados] = useState<DadosCheckout | null>(null);
  const [breakdown, setBreakdown] = useState<CheckoutBreakdown>(null);

  const abrirCheckout = useCallback(
    ({
      slug,
      ticketId,
      quantity = 1,
      price = null,
      ticketName = null,
      eventId = null,
      userId = null,
      waves,
    }: {
      slug: string;
      ticketId: string;
      quantity?: number;
      price?: number | null;
      ticketName?: string | null;
      eventId?: string | null;
      userId?: string | null;
      waves?: unknown[];
    }) => {
      setDados((prev) => {
        const safeWaves =
          Array.isArray(waves)
            ? waves
            : prev?.waves && Array.isArray(prev.waves)
            ? prev.waves
            : [];
        return {
          slug,
          ticketId,
          quantity,
          price,
          ticketName,
          eventId,
          userId,
          waves: safeWaves,
          additional: prev?.additional ?? {},
        };
      });
    },
  );
}

```

```

        setPasso(1);
        setIsOpen(true);
    },
    [],
);
};

const fecharCheckout = useCallback(() => {
    setIsOpen(false);
    setPasso(1);
    setDados(null);
    setBreakdown(null);
}, []);

const irParaPasso = useCallback((novoPasso: 1 | 2 | 3) => {
    setPasso(novoPasso);
}, []);

const atualizarDados = useCallback((patch: Partial<DadosCheckout>) => {
    setDados((prev) => (prev ? { ...prev, ...patch } : prev));
}, []);

const value = useMemo(
() => ({
    isOpen,
    passo,
    dados,
    breakdown,
    setBreakdown,
    abrirCheckout,
    fecharCheckout,
    irParaPasso,
    atualizarDados,
}),
[isOpen, passo, dados, breakdown, abrirCheckout, fecharCheckout, irParaPasso, atualizarDados],
);

return (
    <CheckoutContext.Provider value={value}>
        {children}
    </CheckoutContext.Provider>
);
}

export function useCheckout() {
    const ctx = useContext(CheckoutContext);
    if (!ctx) {
        throw new Error("useCheckout deve ser usado dentro de um CheckoutProvider");
    }
    return ctx;
}

```

app/components/checkout/ModalCheckout.tsx

```

"use client";

import { motion, AnimatePresence } from "framer-motion";
import { ReactNode, useEffect } from "react";
import { useCheckout } from "./contextoCheckout";
import Step1Bilhete from "./Step1Bilhete";
import Step2Pagamento from "./Step2Pagamento";
import Step3Sucesso from "./Step3Sucesso";

type ModalCheckoutProps = {
    children: ReactNode;
};

export default function ModalCheckout() {
    const { isOpen, passo, fecharCheckout, irParaPasso } = useCheckout();

    useEffect(() => {
        if (isOpen) {
            document.body.style.overflow = "hidden";
        } else {
            document.body.style.overflow = "";
        }
    })
}

```

```

    return () => {
      document.body.style.overflow = "";
    };
  }, [isOpen]);
}

// 🔥 Escutar evento vindo do WavesSectionClient e forçar passo 1
useEffect(() => {
  function forceStep1() {
    irParaPasso(1);
  }
  window.addEventListener("ORYA_CHECKOUT_FORCE_STEP1", forceStep1);
  return () => window.removeEventListener("ORYA_CHECKOUT_FORCE_STEP1", forceStep1);
}, [irParaPasso]);

return (
  <AnimatePresence>
    {isOpen && (
      <>
        {/* Overlay */}
        <motion.div
          className="fixed inset-0 bg-black/70 backdrop-blur-xl z-[200]"
          initial={{ opacity: 0 }}
          animate={{ opacity: 1 }}
          exit={{ opacity: 0 }}
          onClick={fecharCheckout}
        />

        {/* Modal */}
        <motion.div
          className="fixed inset-0 z-[210] flex items-center justify-center p-4 overflow-hidden"
          initial={{ opacity: 0, y: 40 }}
          animate={{ opacity: 1, y: 0 }}
          exit={{ opacity: 0, y: 30 }}
        >
          <div className="w-full max-w-lg max-h-[85vh] rounded-3xl border border-white/15 bg-gradient-to-br from-[#020617ee] via-[#020617f8] to-[#020617ee] shadow-[0_24px_80px_rgba(0,0,0,0.95)] text-white overflow-hidden">
            <div className="flex items-center justify-between px-5 py-3 border-b border-white/10">
              {passo > 1 && passo !== 3 ? (
                <button
                  type="button"
                  onClick={() => irParaPasso(Math.max(1, passo - 1) as 1 | 2 | 3)}
                  className="text-[12px] text-white/70 hover:text-white"
                >
                  ← Voltar
                </button>
              ) : (
                <span className="text-[12px] text-white/50">
                  {passo === 3 ? "Pagamento concluído" : "Checkout"}
                </span>
              )}
              <button
                type="button"
                onClick={fecharCheckout}
                className="h-8 w-8 inline-flex items-center justify-center rounded-full bg-white/10 hover:bg-white/20 text-white/80"
                aria-label="Fazer checkout"
              >
                ✕
              </button>
            </div>
            <div className="p-6 overflow-y-auto max-h-[78vh]">
              <StepController />
            </div>
          </div>
        </motion.div>
      </>
    )
  );
}

function StepController() {
  const { passo, irParaPasso } = useCheckout();

  useEffect(() => {
    function handler() {
      irParaPasso(1);
    }
    window.addEventListener("ORYA_CHECKOUT_FORCE_STEP1", handler);
  });
}

```

```

        }
        window.addEventListener("ORYA_CHECKOUT_FORCE_STEP1", handler);
        return () => window.removeEventListener("ORYA_CHECKOUT_FORCE_STEP1", handler);
    }, [irParaPasso]);
}

return (
<>
{passo === 1 && <Step1Bilhete />}
{passo === 2 && <Step2Pagamento />}
{passo === 3 && <Step3Sucesso />}
</>
);
}

```

app/components/checkout/Step1Bilhete.tsx

```

"use client";

import { useState } from "react";
import { useCheckout } from "./contextoCheckout";

type Wave = {
    id: string;
    name: string;
    price: number;
    description?: string | null;
    quantity?: number | null;
    status?: string;
    remaining?: number | null;
};

type CheckoutData = {
    waves?: Wave[];
    additional?: Record<string, unknown>;
    paymentScenario?: string | null;
};

export default function Step1Bilhete() {
    const { dados, irParaPasso, fecharCheckout, atualizarDados } = useCheckout();

    const safeDados: CheckoutData =
        dados && typeof dados === "object"
            ? (dados as CheckoutData)
            : { waves: [], additional: {} };

    const normalizeStatus = (status?: string) =>
        (status || "on_sale").toLowerCase();

    const stableWaves: Wave[] = Array.isArray(safeDados.waves)
        ? [...safeDados.waves].map((w) => ({
            ...w,
            status: normalizeStatus(w.status),
        }))
        : [];
    const cheapestAvailable = [...stableWaves]
        .filter((w) => {
            const st = normalizeStatus(w.status);
            return st !== "sold_out" && st !== "closed";
        })
        .sort((a, b) => (a.price ?? 0) - (b.price ?? 0));
    const hasWaves = stableWaves.length > 0;

    // 🚫 Quantidades iniciais por wave (memoizado para não recriar em cada render)
    const initialQuantidades: Record<string, number> = {};
    for (const w of stableWaves) {
        const rawQty =
            typeof w.quantity === "number" && w.quantity > 0 ? w.quantity : 0;
        const remaining =
            typeof w.remaining === "number" && w.remaining >= 0
                ? w.remaining
                : null;
        const maxForWave =
            remaining === null ? Number.MAX_SAFE_INTEGER : Math.max(0, remaining);
        initialQuantidades[w.id] = Math.min(rawQty, maxForWave);
    }
    const variant =

```

```

(safeDados.additional?.checkoutUiVariant as string) ?? "EVENT_DEFAULT";

const [quantidades, setQuantidades] = useState<Record<string, number>>(
  initialQuantidades,
);
const [padelSelection, setPadelSelection] = useState<
  "INDIVIDUAL" | "DUO_SPLIT" | "DUO_FULL"
>("INDIVIDUAL");
const [padelJoinMode, setPadelJoinMode] = useState<"INVITE_PARTNER" | "LOOKING_FOR_PARTNER">("INVITE_PARTNER");

// Qual wave está expandida (tipo acordeão)
const [aberto, setAberto] = useState<string | null>(
  cheapestAvailable?.id ?? stableWaves[0]?.id ?? null,
);

// 📈 Totais para mostrar apenas (backend recalcula sempre)
const { total, selectedQty } = stableWaves.reduce(
  (acc: { total: number; selectedQty: number }, w: Wave) => {
    const q = quantidades[w.id] ?? 0;
    const price = typeof w.price === "number" ? w.price : 0;
    return { total: acc.total + q * price, selectedQty: acc.selectedQty + q };
  },
  { total: 0, selectedQty: 0 },
);

function toggleWave(id: string) {
  setAberto((prev) => (prev === id ? null : id));
}

function getMaxForWave(waveId: string) {
  const wave = stableWaves.find((w) => w.id === waveId);
  if (!wave) return Number.MAX_SAFE_INTEGER;
  const remaining =
    typeof wave.remaining === "number" && wave.remaining >= 0
      ? wave.remaining
      : null;
  return remaining === null ? Number.MAX_SAFE_INTEGER : Math.max(0, remaining);
}

function handleIncrement(id: string) {
  const maxAllowed = getMaxForWave(id);
  setQuantidades((prev) => {
    const current = prev[id] ?? 0;
    if (current >= maxAllowed) return prev;
    return {
      ...prev,
      [id]: current + 1,
    };
  });
}

function handleDecrement(id: string) {
  setQuantidades((prev) => ({
    ...prev,
    [id]: Math.max(0, (prev[id] ?? 0) - 1),
  }));
}

function handleContinuar() {
  if (variant === "PADEL_TOURNAMENT") {
    const target = stableWaves.find((w) => normalizeStatus(w.status) !== "sold_out" && normalizeStatus(w.status) !== "closed");
    if (!target) return;

    const scenario =
      padelSelection === "DUO_SPLIT"
        ? "GROUP_SPLIT"
        : padelSelection === "DUO_FULL"
        ? "GROUP_FULL"
        : "SINGLE";
  }

  const nextQuantidades: Record<string, number> = { [target.id]: padelSelection === "DUO_FULL" ? 2 : 1 };

  atualizarDados({
    paymentScenario: scenario,
    additional: {
      ...(safeDados.additional && typeof safeDados.additional === "object" ? safeDados.additional : {}),
      quantidades: nextQuantidades,
    }
  });
}

```

```

        total: (target.price ?? 0) * (scenario === "GROUP_FULL" ? 2 : 1),
        padelJoinMode: padelJoinMode,
        checkoutUiVariant: variant,
    },
});

irParaPasso(2);
return;
}

// Permitir avançar mesmo que aparente 0€ – backend decide se é FREE/PAID.
if (selectedQty <= 0) return;

// Guardar info deste step no contexto (quantidades + total)
atualizarDados({
    paymentScenario: "SINGLE",
    additional: {
        ... (safeDados.additional && typeof safeDados.additional === "object" ? safeDados.additional : {}),
        quantidades,
        total,
        checkoutUiVariant: variant,
    },
});

irParaPasso(2);
}

if (!hasWaves) {
    return (
        <div className="p-6 text-sm text-white/70">
            A carregar bilhetes... Se isto persistir, volta atrás e tenta novamente.
        </div>
    );
}

if (variant === "PADEL_TOURNAMENT") {
    const baseWave = stableWaves.find((w) => normalizeStatus(w.status) !== "sold_out" && normalizeStatus(w.status) !== "closed")
    ?? stableWaves[0];
    const basePrice = baseWave?.price ?? 0;
    return (
        <div className="flex flex-col gap-6 text-white max-h-[80vh] overflow-hidden">
            <header className="flex items-start justify-between gap-3">
                <div className="space-y-1">
                    <p className="text-[11px] uppercase tracking-[0.18em] text-white/55">
                        Passo 1 de 3
                    </p>
                    <h2 className="text-xl font-semibold leading-tight">Escolhe como queres jogar</h2>
                    <p className="text-[11px] text-white/60 max-w-sm">
                        Padel: inscrição individual ou como dupla. Pagas já a tua parte ou a dupla completa.
                    </p>
                </div>
                <button
                    type="button"
                    onClick={fecharCheckout}
                    className="text-[11px] rounded-full border border-white/15 px-3 py-1 text-white/65 hover:text-white hover:border-white/40 transition-colors"
                >
                    Fechar
                </button>
            </header>
            <div className="h-1 w-full rounded-full bg-white/10 overflow-hidden">
                <div className="h-full w-1/3 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5]" />
            </div>
            <div className="grid gap-3 md:grid-cols-3">
                <button
                    type="button"
                    onClick={() => {
                        setPadelSelection("INDIVIDUAL");
                        setPadelJoinMode("INVITE_PARTNER");
                    }}
                    className={`rounded-2xl border px-4 py-4 text-left transition shadow ${
                        padelSelection === "INDIVIDUAL"
                            ? "border-[#6BFFFF] bg-white/10 shadow-[0_0_24px_rgba(107,255,255,0.35)]"
                            : "border-white/12 bg-white/[0.03] hover:border-white/25"
                    }`}
                >
                    <p className="text-sm font-semibold">Inscrição individual</p>

```

```

<p className="text-[11px] text-white/65 mt-1">1 lugar. Pode entrar em matchmaking.</p>
<p className="mt-3 text-lg font-semibold">{basePrice.toFixed(2)} €</p>
<div className="mt-3 space-y-2 text-[11px] text-white/70">
  <label className="flex items-center gap-2">
    <input
      type="radio"
      checked={padelJoinMode === "INVITE_PARTNER" && padelSelection === "INDIVIDUAL"}
      onChange={() => setPadelJoinMode("INVITE_PARTNER")}
    />
    Já tenho parceiro (convite)
  </label>
  <label className="flex items-center gap-2">
    <input
      type="radio"
      checked={padelJoinMode === "LOOKING_FOR_PARTNER" && padelSelection === "INDIVIDUAL"}
      onChange={() => setPadelJoinMode("LOOKING_FOR_PARTNER")}
    />
    Estou à procura de parceiro
  </label>
</div>
</button>

<button
  type="button"
  onClick={() => {
    setPadelSelection("DUO_SPLIT");
    setPadelJoinMode("INVITE_PARTNER");
  }}
  className={`rounded-2xl border px-4 py-4 text-left transition shadow ${
    padelSelection === "DUO_SPLIT"
      ? "border-[#6BFFFF] bg-white/10 shadow-[0_0_24px_rgba(107,255,255,0.35)]"
      : "border-white/12 bg-white/[0.03] hover:border-white/25"
  }`}
>
  <p className="text-sm font-semibold">Dupla · pagar só a minha parte</p>
  <p className="text-[11px] text-white/65 mt-1">1 lugar pago. O parceiro paga o dele.</p>
  <p className="mt-3 text-lg font-semibold">{basePrice.toFixed(2)} €</p>
</button>

<button
  type="button"
  onClick={() => {
    setPadelSelection("DUO_FULL");
    setPadelJoinMode("INVITE_PARTNER");
  }}
  className={`rounded-2xl border px-4 py-4 text-left transition shadow ${
    padelSelection === "DUO_FULL"
      ? "border-[#6BFFFF] bg-white/10 shadow-[0_0_24px_rgba(107,255,255,0.35)]"
      : "border-white/12 bg-white/[0.03] hover:border-white/25"
  }`}
>
  <p className="text-sm font-semibold">Dupla · pagar os dois lugares</p>
  <p className="text-[11px] text-white/65 mt-1">2 lugares pagos já garantidos.</p>
  <p className="mt-3 text-lg font-semibold">{(basePrice * 2).toFixed(2)} €</p>
</button>
</div>

<div className="flex items-center justify-between gap-3 border-t border-white/10 pt-3">
  <div className="text-[11px] text-white/70">
    Seleciona uma opção para avançar.
  </div>
  <button
    type="button"
    onClick={handleContinuar}
    className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-5 py-2.5 text-xs font-semibold text-black shadow-[0_0_26px_rgba(107,255,255,0.55)] hover:scale-[1.02] active:scale-95 transition-transform"
  >
    Continuar
  </button>
</div>
</div>
);
}

return (
<div className="flex flex-col gap-6 text-white max-h-[80vh] overflow-hidden">
  /* Header */
<header className="flex items-start justify-between gap-3">

```

```

<div className="space-y-1">
  <p className="text-[11px] uppercase tracking-[0.18em] text-white/55">
    Passo 1 de 3
  </p>
  <h2 className="text-xl font-semibold leading-tight">
    Escolhe o teu bilhete
  </h2>
  <p className="text-[11px] text-white/60 max-w-xs">
    Seleciona a wave, ajusta quantidades e revê antes de pagar.
  </p>
</div>
<button
  type="button"
  onClick={fecharCheckout}
  className="text-[11px] rounded-full border border-white/15 px-3 py-1 text-white/65 hover:text-white hover:border-white/40 transition-colors">
  >
  Fechar
</button>
</header>

{/* Barra de progresso */}
<div className="h-1 w-full rounded-full bg-white/10 overflow-hidden">
  <div className="h-full w-1/3 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5]" />
</div>

{/* Lista de waves com scroll interno */}
<div className="flex-1 overflow-y-auto pr-2 space-y-3">
  {stableWaves.map((wave) => {
    const q = quantidades[wave.id] ?? 0;
    const isOpen = aberto === wave.id;
    const status = normalizeStatus(wave.status);
    const isSoldOut = status === "sold_out" || status === "closed";
    const maxForWave = getMaxForWave(wave.id);
    const badge =
      status === "upcoming"
        ? "Em breve"
        : isSoldOut
        ? "Venda terminada"
        : "Disponível";
    const badgeClass = isSoldOut
      ? "border-red-400/50 bg-red-500/20 text-red-50"
      : status === "upcoming"
      ? "border-amber-300/50 bg-amber-400/20 text-amber-50"
      : "border-emerald-300/50 bg-emerald-500/18 text-emerald-50";

    return (
      <div
        key={wave.id}
        className="rounded-2xl border border-white/12 bg-white/[0.04] shadow-[0_6px_20px_rgba(0,0,0,0.55)]">
        >
        {/* Header Wave */}
        <button
          type="button"
          onClick={() => toggleWave(wave.id)}
          className="w-full flex items-center justify-between px-4 py-3"
          disabled={isSoldOut}
        >
        <div className="text-left">
          <p className="text-sm font-semibold">{wave.name}</p>
          <p className="text-[11px] text-white/50">
            {typeof wave.price === "number"
              ? `${wave.price.toFixed(2)} €`
              : "Preço indisponível"}
          </p>
        </div>

        <div className="flex items-center gap-2">
          <span
            className={`rounded-full border px-2 py-0.5 text-[10px] text-white/80 ${isSoldOut
              ? "border-red-400/40 bg-red-500/10"
              : "border-emerald-300/30 bg-emerald-400/10"
            }`}>
          <span>
            {isSoldOut ? "Esgotado" : "Disponível"}
          </span>
          <span>

```

```

        className={`flex h-7 w-7 items-center justify-center rounded-full border ${

          q > 0
            ? "border-emerald-400/50 bg-emerald-400/15 text-emerald-100"
            : "border-white/20 bg-white/10 text-white/80"
        }`}
      >
      {q > 0 ? q : isOpen ? "-" : "+"}
    </span>
  </div>
</button>

/* Conteúdo Wave */
{isOpen && (
  <div className="px-4 pb-4 flex flex-col gap-3">
    <p className="text-[11px] text-white/60">
      {wave.description ?? "Sem descrição disponível."}
    </p>

    {isSoldOut && (
      <div className="rounded-xl border border-white/10 bg-white/[0.03] px-3 py-2 text-[11px] text-white/70">
        Venda terminada. Escolhe outra wave ou volta mais tarde.
      </div>
    )}
  </div>
  <div className="inline-flex items-center gap-2 rounded-full bg-black/60 border border-white/15 px-2 py-1.5
shadow-md">

    <button
      type="button"
      onClick={() => handleDecrement(wave.id)}
      className="flex h-7 w-7 items-center justify-center rounded-full bg-white/10 hover:bg-white/20
disabled:opacity-50"
      disabled={isSoldOut}
    >
      -
    </button>

    <span className="w-9 text-center text-sm font-semibold">
      {q}
    </span>

    <button
      type="button"
      onClick={() => handleIncrement(wave.id)}
      className="flex h-7 w-7 items-center justify-center rounded-full bg-white text-black hover:bg-zinc-100
disabled:opacity-50"
      disabled={
        isSoldOut || (quantidades[wave.id] ?? 0) >= maxForWave
      }
    >
      +
    </button>
  </div>
</div>
)
);
})
</div>

/* Total + CTA */
<div className="border-t border-white/10 pt-3 flex flex-col sm:flex-row sm:items-center sm:justify-between">
  <p className="text-[12px] text-white/70">
    Total:{" "}
    <span className="font-semibold text-white text-base">
      {total.toFixed(2)} €
    </span>
  </p>
  <button
    type="button"
    disabled={total === 0}
    onClick={handleContinuar}
    className="mt-3 sm:mt-0 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-5 py-2.5 text-xs
font-semibold text-black shadow-[0_0_26px_rgba(107,255,255,0.55)] disabled:opacity-50 hover:scale-[1.02] active:scale-95
transition-transform"
  >
    Continuar para pagamento
  </button>
</div>

```

```
    </div>
);
}
```

app/components/checkout/Step2Pagamento.tsx

```
"use client";

import { useEffect, useMemo, useRef, useState } from "react";
import {
  Elements,
  PaymentElement,
  useStripe,
  useElements,
} from "@stripe/react-stripe-js";
import {
  loadStripe,
  type Appearance,
  type StripeElementsOptions,
} from "@stripe/stripe-js";
import { type CheckoutBreakdown, useCheckout } from "./contextoCheckout";
import { supabaseBrowser } from "@/lib/supabaseBrowser";
import { isValidPhone, sanitizePhone } from "@/lib/phone";
import { sanitizeUsername, validateUsername } from "@/lib/username";
import { createPurchaseId } from "@/lib/checkoutSchemas";

function isValidEmail(email: string) {
  return /^[^@]+@[^@]+\.[^@]+$/ .test(email);
}

function formatMoney(cents: number, currency = "EUR") {
  return new Intl.NumberFormat("pt-PT", {
    style: "currency",
    currency,
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  }).format(cents / 100);
}

function buildClientFingerprint(input: unknown) {
  try {
    return JSON.stringify(input);
  } catch {
    // Fallback muito raro (ex.: objeto circular) - usamos um valor que força refresh.
    return `fp_${Date.now()}`;
  }
}

type CheckoutItem = {
  ticketId: number;
  quantity: number;
};

const scenarioCopy: Record<string, string> = {
  GROUP_SPLIT: "Estás a pagar apenas a tua parte desta dupla.",
  GROUP_FULL: "Estás a comprar 2 lugares (tu + parceiro).",
  RESALE: "Estás a comprar um bilhete em revenda.",
  FREE_CHECKOUT: "Evento gratuito – só para utilizadores com conta e username.",
};

type CheckoutWave = {
  id: number | string;
  price: number;
};

type CheckoutData = {
  slug?: string;
  waves?: CheckoutWave[];
  additional?: {
    quantidades?: Record<string, number>;
    total?: number;
    guestName?: string;
    guestEmail?: string;
    guestPhone?: string | null;
    idempotencyKey?: string;
    purchaseId?: string | null;
    requiresAuth?: boolean;
  };
};
```

```

};

paymentScenario?: string | null;
};

type GuestInfo = {
  name: string;
  email: string;
  phone?: string;
};

const FREE_PLACEHOLDER_INTENT_ID = "FREE_CHECKOUT";

export default function Step2Pagamento() {
  const { dados, irParaPasso, atualizarDados, breakdown, setBreakdown } = useCheckout();

  const [clientSecret, setClientSecret] = useState<string | null>(null);
  const [serverAmount, setServerAmount] = useState<number | null>(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  // 🗝️ Auth state
  const [authChecked, setAuthChecked] = useState(false);
  const [userId, setIdUser] = useState<string | null>(null);
  const [authChecking, setAuthChecking] = useState(true);
  // Preferimos convidado por defeito para reduzir fricção
  const [purchaseMode, setPurchaseMode] = useState("auth" | "guest")("guest");
  const [authInfo, setAuthInfo] = useState<string | null>(null);
  const [guestErrors, setGuestErrors] = useState<{ name?: string; email?: string; phone?: string }>({});

  // 🧑 Guest form state
  const [guestName, setGuestName] = useState("");
  const [guestEmail, setGuestEmail] = useState("");
  const [guestEmailConfirm, setGuestEmailConfirm] = useState("");
  const [guestPhone, setGuestPhone] = useState("");
  const [guestSubmitVersion, setGuestSubmitVersion] = useState(0);
  const [promoInput, setPromoInput] = useState("");
  const [promoCode, setPromoCode] = useState("");
  const [appliedDiscount, setAppliedDiscount] = useState<number>(0);
  const [promoWarning, setPromoWarning] = useState<string | null>(null);
  const [appliedPromoLabel, setAppliedPromoLabel] = useState<string | null>(null);
  const lastIntentKeyRef = useRef<string | null>(null);
  const inFlightIntentRef = useRef<string | null>(null);
  const ensuredIdemKeyRef = useRef(false);
  const lastClearedFingerprintRef = useRef<string | null>(null);
  const idempotencyMismatchCountRef = useRef(0);
  const [cachedIntent, setCachedIntent] = useState<{
    key: string;
    clientSecret: string | null;
    amount: number | null;
    breakdown: CheckoutBreakdown;
    discount: number;
    freeCheckout: boolean;
    paymentScenario?: string | null;
    promoLabel?: string | null;
    autoAppliedPromo?: boolean;
    purchaseId?: string | null;
  } | null>(null);

  const safeDados: CheckoutData | null =
    dados && typeof dados === "object" ? (dados as CheckoutData) : null;
  const scenario = safeDados?.paymentScenario ?? cachedIntent?.paymentScenario ?? null;
  const isFreeScenario = scenario === "FREE_CHECKOUT";
  const needsStripe = !isFreeScenario;

  const additionalForRules =
    safeDados?.additional && typeof safeDados.additional === "object"
      ? safeDados.additional
      : {};

  // Regras de acesso ao checkout:
  // - FREE_CHECKOUT: sempre com conta
  // - GROUP_SPLIT: por defeito exige conta (capitão a pagar a sua parte)
  // - Podemos forçar via additional.requiresAuth (SSOT no futuro)
  const requiresAuth =
    Boolean((additionalForRules as Record<string, unknown>)? .requiresAuth) ||
    isFreeScenario ||
    scenario === "GROUP_SPLIT";
}

```

```

useEffect(() => {
  if (!safeDados) return;
  if (ensuredIdemKeyRef.current) return;

  const additional =
    safeDados.additional && typeof safeDados.additional === "object"
    ? (safeDados.additional as Record<string, unknown>)
    : {};

  const existing =
    typeof additional.idempotencyKey === "string" && additional.idempotencyKey.trim()
    ? additional.idempotencyKey.trim()
    : null;

  if (!existing) {
    ensuredIdemKeyRef.current = true;
    try {
      atualizarDados({
        additional: {
          ... (safeDados?.additional ?? {}),
          idempotencyKey: crypto.randomUUID(),
        },
      });
    } catch {
      // Se por algum motivo falhar (ambiente sem crypto), não bloqueamos o checkout
    }
    return;
  }

  ensuredIdemKeyRef.current = true;
}, [safeDados, atualizarDados]);

useEffect(() => {
  if (!promoCode.trim()) {
    setAppliedDiscount(0);
  }
}, [promoCode]);

useEffect(() => {
  if (requiresAuth && purchaseMode !== "auth") {
    setPurchaseMode("auth");
    setClientSecret(null);
    setServerAmount(null);
  }
}, [requiresAuth, purchaseMode]);

const stripePromise = useMemo(() => {
  // FREE_CHECKOUT não precisa de Stripe no cliente.
  if (!needsStripe) return null;
  const key = process.env.NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY;
  if (!key) return null;
  return loadStripe(key);
}, [needsStripe]);

useEffect(() => {
  if (needsStripe && !stripePromise) {
    setError("Configuração de pagamentos indisponível. Tenta novamente mais tarde.");
    setLoading(false);
  }
}, [stripePromise, needsStripe]);

// Primeiro: verificar se há sessão Supabase no browser e ficar a escutar mudanças de auth
useEffect(() => {
  let cancelled = false;

  async function checkAuthOnce() {
    try {
      setAuthChecking(true);
      const { data, error } = await supabaseBrowser.auth.getUser();

      if (cancelled) return;

      if (error || !data?.user) {
        setId(null);
        if (error) {
          setAuthInfo("Sessão não encontrada. Inicia sessão ou continua como convidado.");
        }
      }
    }
  }
}

```

```

        } else {
          setId(data.user.id);
          setAuthInfo(null);
        }
      } catch (err) {
        console.error("[Step2Pagamento] Erro ao verificar auth inicial:", err);
        if (!cancelled) {
          setId(null);
          setAuthInfo("Sessão não encontrada. Inicia sessão ou continua como convidado.");
        }
      } finally {
        if (!cancelled) {
          setAuthChecked(true);
          setAuthChecking(false);
        }
      }
    }
  }

checkAuthOnce();

const {
  data: { subscription },
} = supabaseBrowser.auth.onAuthStateChange(_event, session) => {
  if (cancelled) return;

  if (session?.user) {
    setId(session.user.id);
    setAuthChecked(true);
    setAuthChecking(false);
    setAuthInfo(null);
  } else {
    setId(null);
    setAuthChecked(true);
    setAuthChecking(false);
    setAuthInfo("Sessão não encontrada. Inicia sessão ou continua como convidado.");
  }
};

return () => {
  cancelled = true;
  subscription.unsubscribe();
};
}, [ ]);

// Se vierem dados pré-preenchidos (ex.: voltar atrás), sincronizamos com o estado local do guest
useEffect(() => {
  const additional =
    safeDados?.additional && typeof safeDados.additional === "object"
    ? safeDados.additional
    : {};
  if (typeof additional.guestName === "string") {
    setGuestName(additional.guestName);
  }
  if (typeof additional.guestEmail === "string") {
    setGuestEmail(additional.guestEmail);
  }
  if (typeof (additional as Record<string, unknown>)?.guestEmailConfirm === "string") {
    setGuestEmailConfirm((additional as Record<string, string>).guestEmailConfirm);
  }
  if (typeof additional.guestPhone === "string") {
    setGuestPhone(additional.guestPhone);
  }
}, [safeDados]);

const payload = useMemo(() => {
  if (!safeDados) return null;

  const waves = Array.isArray(safeDados.waves) ? safeDados.waves : [];
  const additional =
    safeDados.additional && typeof safeDados.additional === "object"
    ? safeDados.additional
    : {};
  const quantidades: Record<string, number> =
    (additional.quantidades as Record<string, number> | undefined) ?? {};

  if (!safeDados.slug || waves.length === 0) return null;
}

```

```

const items: CheckoutItem[] = (waves
  .map((w) => {
    const qty = quantidades[w.id] ?? 0;
    if (!qty || qty <= 0) return null;
    const ticketId = Number(w.id);
    if (!Number.isFinite(ticketId)) return null;
    return { ticketId, quantity: qty };
  })
  .filter(Boolean) as CheckoutItem[])
  .sort((a, b) => a.ticketId - b.ticketId);

if (items.length === 0) return null;

const totalFromStep1 =
  typeof additional.total === "number" ? additional.total : null;

// Não enviamos âncoras (purchaseId/idempotencyKey); o backend calcula dedupe (purchaseId) de forma determinística.
const idemKey = undefined;
const purchaseId = undefined;

return {
  slug: safeDados.slug,
  items,
  total: totalFromStep1,
  promoCode: promoCode.trim() || undefined,
  paymentScenario: safeDados.paymentScenario ?? undefined,
  requiresAuth,
  idempotencyKey: idemKey,
  purchaseId: purchaseId || undefined,
};
, [safeDados, promoCode, requiresAuth]);

const checkUsernameAvailability = async (value: string) => {
  const cleaned = sanitizeUsername(value);
  const validation = validateUsername(cleaned);
  if (!validation.valid) {
    setError(validation.error);
    return { ok: false, username: cleaned };
  }
  try {
    const res = await fetch(`'/api/username/check?username=${encodeURIComponent(cleaned)}'`);
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.available === false) {
      setError("Esse @ já está a ser usado.");
      return { ok: false, username: cleaned };
    }
    return { ok: true, username: validation.normalized };
  } catch (err) {
    console.error("[Step2Pagamento] erro a verificar username", err);
    setError("Não foi possível verificar o username. Tenta novamente.");
    return { ok: false, username: cleaned };
  }
};

useEffect(() => {
  // Se não houver dados de checkout, mandamos de volta
  if (!payload) {
    irParaPasso(1);
    return;
  }

  if (needsStripe && !stripePromise) return;
  // Enquanto não sabemos se está logado, não fazemos nada
  if (!authChecked) return;

  const isGuestFlow = purchaseMode === "guest";
  const hasGuestSubmission = guestSubmitVersion > 0;
  const guestNameClean = guestName.trim();
  const guestEmailClean = guestEmail.trim();
  const guestPhoneClean = guestPhone.trim();
  const guestReady =
    isGuestFlow && hasGuestSubmission && guestNameClean !== "" && guestEmailClean !== "";

  // Se não está logado e ainda não escolheu convidado, mostramos UI e não chamamos a API
  if (!userId && !guestReady) {
    setLoading(false);
    setClientSecret(null);
    setServerAmount(null);
  }
}

```

```

        return;
    }

    const guestPayload: GuestInfo | null = guestReady
    ? {
        name: guestNameClean,
        email: guestEmailClean,
        phone: guestPhoneClean || undefined,
    }
    : null;

    const clientFingerprint = buildClientFingerprint({
        slug: payload.slug,
        items: payload.items,
        total: payload.total ?? null,
        promoCode: payload.promoCode ?? null,
        paymentScenario: payload.paymentScenario ?? null,
        requiresAuth,
        mode: purchaseMode,
        userId: userId ?? null,
        guest: guestPayload
    ? {
        name: guestPayload.name,
        email: guestPayload.email,
        phone: guestPayload.phone ?? null,
    }
    : null,
});

const additionalObj =
safeDados?.additional && typeof safeDados.additional === "object"
? (safeDados.additional as Record<string, unknown>)
: {};

const existingClientFingerprint =
typeof (additionalObj as any).clientFingerprint === "string"
? String((additionalObj as any).clientFingerprint)
: null;

const existingIntentFingerprint =
typeof (additionalObj as any).intentFingerprint === "string"
? String((additionalObj as any).intentFingerprint)
: null;

const hasExistingPurchaseState = Boolean(
(additionalObj as any).purchaseId || (additionalObj as any).paymentIntentId || (additionalObj as any).freeCheckout,
);

// Se o utilizador alterou o checkout (items/promo/guest	mode/etc.) mas ainda temos um purchaseId antigo,
// limpamos o estado para não reutilizar o PaymentIntent errado (caso clássico: aplicar/remover promo e não recalcular).
if (
hasExistingPurchaseState &&
existingClientFingerprint &&
existingClientFingerprint !== clientFingerprint &&
lastClearedFingerprintRef.current !== clientFingerprint
) {
lastClearedFingerprintRef.current = clientFingerprint;

// Limpeza local imediata para evitar UI/states inconsistentes
setCachedIntent(null);
setClientSecret(null);
setServerAmount(null);
setBreakdown(null);
lastIntentKeyRef.current = null;
inFlightIntentRef.current = null;

let nextIdemKey: string | undefined;
try {
nextIdemKey = crypto.randomUUID();
} catch {
nextIdemKey = undefined;
}

atualizarDados({
additional: {
...(safeDados?.additional ?? {}),
purchaseId: null,
paymentIntentId: undefined,
}
});
}

```

```

        freeCheckout: undefined,
        appliedPromoLabel: undefined,
        // clientFingerprint é só para o FE detetar mudanças
        clientFingerprint,
        // intentFingerprint é do BE (hash). Ao mudar seleção, limpamos.
        intentFingerprint: undefined,
        idempotencyKey: nextIdemKey ?? (additionalObj as any).idempotencyKey,
    },
});

 setLoading(false);
return;
}

// Backfill: se já existe purchaseId mas ainda não guardámos fingerprint, guardamos para futuras comparações.
if (hasExistingPurchaseState && !existingClientFingerprint) {
    atualizarDados({
        additional: {
            ... (safeDados?.additional ?? {}),
            clientFingerprint,
        },
    });
}

// Chave estável para não recriar PaymentIntent sem necessidade
const intentKey = JSON.stringify({
    payload,
    guest: guestPayload,
    userId: userId ?? "guest",
    mode: purchaseMode,
    scenario: safeDados?.paymentScenario ?? null,
    clientFingerprint,
    idempotencyKey:
        (safeDados?.additional as Record<string, unknown> | undefined)?.idempotencyKey ??
        (payload as any)?.idempotencyKey ??
        null,
    purchaseId:
        (payload as any)?.purchaseId ??
        (safeDados?.additional as any)?.purchaseId ??
        null,
}),
);

// Se já temos um intent em cache com a mesma key, reaproveitamos
if (cachedIntent?.key === intentKey) {
    setClientSecret(cachedIntent.clientSecret);
    setServerAmount(cachedIntent.amount);
    setBreakdown(cachedIntent.breakdown);
    setAppliedDiscount(cachedIntent.discount);
    setAppliedPromoLabel(cachedIntent.promoLabel ?? null);
    lastIntentKeyRef.current = intentKey;
    setLoading(false);
    if (cachedIntent.freeCheckout) {
        irParaPasso(3);
        return;
    }
    return;
}

// Se já temos clientSecret para o mesmo payload, não refazemos
if (clientSecret && lastIntentKeyRef.current === intentKey) {
    setLoading(false);
    return;
}

// Evita requests paralelos com o mesmo payload; se detectarmos que estamos presos
// (ex.: Strict Mode cancela a primeira run e deixa loading a true), limpamos o ref
// para voltar a tentar.
if (inFlightIntentRef.current === intentKey) {
    const stuck =
        loading &&
        !clientSecret &&
        lastIntentKeyRef.current !== intentKey;
    if (!stuck) {
        return;
    }
    inFlightIntentRef.current = null;
}

```

```

let cancelled = false;

async function createIntent() {
  try {
    inFlightIntentRef.current = intentKey;
    setLoading(true);
    setError(null);
    setBreakdown(null);

    console.log(
      "[Step2Pagamento] A enviar payload para /api/payments/intent:",
      payload,
    );
  }

  const idem =
    (safeDados?.additional as Record<string, unknown> | undefined)?.idempotencyKey ??
    (payload as any)?.idempotencyKey ??
    null;

  let attempt = 0;
  // Não enviamos purchaseId/idempotencyKey; o backend calcula anchors determinísticas.
  let currentPayload = { ...payload };
  delete (currentPayload as any).purchaseId;
  delete (currentPayload as any).idempotencyKey;
  let currentIntentFingerprint = undefined;
  let res: Response | null = null;
  let data: any = null;
  let handled409 = false;

  while (attempt < 2) {
    res = await fetch("/api/payments/intent", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        ...currentPayload,
        guest: guestPayload ?? undefined,
        requiresAuth,
        purchaseId: null,
        idempotencyKey: null,
        intentFingerprint: currentIntentFingerprint ?? undefined,
      }),
    });
  }

  data = await res.json().catch(() => null);

  if (res.status === 409) {
    handled409 = true;
    attempt += 1;
    // Reset total: limpar caches e fazer um retry sem anchors próprias.
    currentPayload = { ...payload };
    delete (currentPayload as any).purchaseId;
    delete (currentPayload as any).idempotencyKey;
    currentIntentFingerprint = undefined;

    if (!cancelled) {
      setCachedIntent(null);
      setClientSecret(null);
      setServerAmount(null);
      setBreakdown(null);
      lastIntentKeyRef.current = null;
      inFlightIntentRef.current = null;
      try {
        atualizarDados({
          additional: {
            ...(safeDados?.additional ?? {}),
            purchaseId: null,
            paymentIntentId: undefined,
            freeCheckout: undefined,
            appliedPromoLabel: undefined,
            clientFingerprint,
            intentFingerprint: undefined,
            idempotencyKey: undefined,
          },
        });
      } catch {}
    }
  }
}

```

```

        continue;
    }

    // status != 409 → sair do loop
    break;
}

if (!data || typeof data !== "object") {
    if (!cancelled) {
        setBreakdown(null);
        setError("Resposta inválida do servidor. Tenta novamente.");
    }
    return;
}
console.log("[Step2Pagamento] Resposta de /api/payments/intent:", {
    status: res.status,
    ok: res.ok,
    data,
});

// 409 → idempotencyKey reutilizada com payload diferente (proteção contra intents errados/duplicados)
if (res.status === 409) {
    if (!cancelled) {
        // Reset total para forçar recalcular (idempotencyKey + purchaseId + PI state)
        setCachedIntent(null);
        setClientSecret(null);
        setServerAmount(null);
        setBreakdown(null);
        lastIntentKeyRef.current = null;
        inFlightIntentRef.current = null;

        let nextIdemKey: string | undefined;
        try {
            nextIdemKey = crypto.randomUUID();
        } catch {
            nextIdemKey = undefined;
        }

        try {
            atualizarDados({
                additional: {
                    ... (safeDados?.additional ?? {}),
                    purchaseId: null,
                    paymentIntentId: undefined,
                    freeCheckout: undefined,
                    appliedPromoLabel: undefined,
                    clientFingerprint,
                    intentFingerprint: undefined,
                    idempotencyKey: nextIdemKey,
                },
            });
        } catch {}
    }
}

// Força novo ciclo de preparação (especialmente útil em guest flow)
setGuestSubmitVersion((v) => v + 1);
setPromoWarning(
    typeof (data as any)?.error === "string"
    ? (data as any).error
    : "O seu checkout mudou e estamos a recalcular o pagamento..",
);
setError(null);
}

return;
}

// Se a API disser 401 → sessão ausente/expirada OU cenário que exige auth
if (res.status === 401) {
    const respCode = typeof data?.code === "string" ? data.code : null;
    const mustAuth =
        requiresAuth ||
        respCode === "AUTH_REQUIRED_FOR_GROUP_SPLIT" ||
        respCode === "AUTH_REQUIRED";

    if (!cancelled) {
        // Garantimos estado limpo para permitir retry correto
        setClientSecret(null);
        setServerAmount(null);
        setBreakdown(null);
    }
}

```

```

        setCachedIntent(null);
    }

    if (mustAuth) {
        if (!cancelled) {
            setPurchaseMode("auth");
            setGuestSubmitVersion(0);
            setGuestErrors({});
            setError(null);
            setAuthInfo(
                respCode === "AUTH_REQUIRED_FOR_GROUP_SPLIT"
                    ? "Para pagar apenas a tua parte tens de iniciar sessão."
                    : "Este tipo de checkout requer sessão iniciada."
            );
        }
        return;
    }

    // Guest permitido, mas algo correu mal (ex.: backend rejeitou por sessão)
    if (!cancelled) {
        setError(
            typeof data?.error === "string"
                ? data.error
                : "Sessão expirada. Tenta novamente."
        );
    }
    return;
}

if (!res.ok || !data?.ok) {
    setBreakdown(null);

    const respCode = typeof data?.code === "string" ? data.code : null;
    const retryable = typeof data?.retryable === "boolean" ? data.retryable : null;
    const nextAction =
        typeof data?.nextAction === "string" && data.nextAction
            ? data.nextAction
            : null;

    if (respCode === "IDEMPOTENCY_KEY_PAYLOAD_MISMATCH") {
        if (!cancelled) {
            // Limita a 1 tentativa local para evitar loops de 409
            if (idempotencyMismatchCountRef.current >= 1) {
                setError(
                    typeof data?.error === "string"
                        ? data.error
                        : "O checkout mudou noutro separador. Volta ao passo anterior ou recarrega a página e tenta de novo.",
                );
                setSubmitting(false);
                setLoading(false);
                return;
            }
            idempotencyMismatchCountRef.current += 1;
        }
        // Limpa estado local para forçar novo intent (sem reusar idempotency antiga)
        setClientSecret(null);
        setServerAmount(null);
        setBreakdown(null);
        setCachedIntent(null);
        lastIntentKeyRef.current = null;
        inFlightIntentRef.current = null;

        try {
            atualizarDados({
                additional: {
                    ...(safeDados?.additional ?? {}),
                    purchaseId: null,
                    paymentIntentId: undefined,
                    idempotencyKey: undefined,
                    intentFingerprint: undefined,
                    clientFingerprint,
                    freeCheckout: undefined,
                    appliedPromoLabel: undefined,
                },
            });
        } catch {}
    }
    setError(

```

```

        typeof data?.error === "string"
        ? data.error
        : "O checkout foi aberto noutro separador. Recriámos o pagamento; volta a clicar em Pagar.",
    );
    // Não re-submete automaticamente; utilizador volta a clicar
}
return;
}

if (respCode === "USERNAME_REQUIRED_FOR_FREE") {
    if (!cancelled) {
        setError("Este evento gratuito requer sessão com username definido.");
        setPurchaseMode("auth");
        setAuthInfo(
            "Inicia sessão e define um username para concluir a inscrição gratuita.",
        );
    }
    return;
}

const msg =
    respCode === "PRICE_CHANGED"
        ? "Os preços foram atualizados. Revê a seleção e tenta novamente."
        : respCode === "INSUFFICIENT_STOCK"
        ? "Stock insuficiente para um dos bilhetes."
        : typeof data?.error === "string"
        ? data.error
        : "Não foi possível preparar o pagamento.";

if (respCode === "ORGANIZER_STRIPE_NOT_CONNECTED") {
    if (!cancelled) {
        setError(
            data?.message ||
            "Pagamentos desativados para este evento enquanto o organizador não ligar a Stripe.",
        );
        setAuthInfo("Liga a Stripe em Finanças & Payouts para ativar pagamentos.");
    }
    return;
}

const promoFail =
    payload?.promoCode &&
    typeof data?.error === "string" &&
    data.error.toLowerCase().includes("código");

if (promoFail && !cancelled) {
    setPromoWarning("Código não aplicado. Continua sem desconto.");
    setPromoCode("");
    setAppliedDiscount(0);
    setAppliedPromoLabel(null);
    setError(null);
    setBreakdown(null);
    return;
}

if (!cancelled) {
    setError(
        respCode === "PRICE_CHANGED"
            ? "Os preços mudaram. Volta ao passo anterior e revê a seleção."
            : respCode === "INSUFFICIENT_STOCK"
            ? "Stock insuficiente. Remove itens esgotados e tenta novamente."
            : nextAction === "PAY_NOW" && retryable
            ? "Precisamos de novo pagamento para continuar."
            : msg,
    );

    if (respCode === "PRICE_CHANGED" || respCode === "INSUFFICIENT_STOCK") {
        setPromoWarning(null);
        setBreakdown(null);
        setClientSecret(null);
        setServerAmount(null);
    }
}
return;
}

if (!cancelled) {
    const paymentScenarioResponse =

```

```

    typeof data?.paymentScenario === "string"
      ? data.paymentScenario
      : safeDados?.paymentScenario ?? null;
const purchaseIdFromServer = typeof data?.purchaseId === "string" ? data.purchaseId : undefined;
const responseIntentFingerprint =
  typeof data?.intentFingerprint === "string" ? data.intentFingerprint : null;
const responseIdemKey =
  typeof data?.idempotencyKey === "string" && data.idempotencyKey.trim()
  ? data.idempotencyKey.trim()
  : null;

const effectiveIntentFingerprint = responseIntentFingerprint ?? existingIntentFingerprint ?? undefined;

// Só fazemos backfill da idempotencyKey se por algum motivo ainda não existir localmente.
const localIdem =
  typeof (safeDados?.additional as Record<string, unknown> | undefined)?.idempotencyKey === "string" &&
  String((safeDados?.additional as Record<string, unknown>).idempotencyKey).trim()
  ? String((safeDados?.additional as Record<string, unknown>).idempotencyKey).trim()
  : null;
const effectiveIdemKey = localIdem ?? responseIdemKey;

const promoLabel =
  promoCode?.trim()
  ? promoCode.trim()
  : data.discountCents && data.discountCents > 0
  ? "Promo automática"
  : null;
const isAutoAppliedPromo = !promoCode?.trim() && Boolean(promoLabel);
const breakdownFromResponse =
  data.breakdown && typeof data.breakdown === "object"
  ? (data.breakdown as CheckoutBreakdown)
  : null;
const discountCentsNumber =
  typeof data.discountCents === "number"
  ? data.discountCents
  : typeof breakdownFromResponse?.discountCents === "number"
  ? breakdownFromResponse.discountCents
  : 0;
const subtotalCentsNumber =
  typeof breakdownFromResponse?.subtotalCents === "number"
  ? breakdownFromResponse.subtotalCents
  : null;
const platformFeeCentsNumber =
  typeof breakdownFromResponse?.platformFeeCents === "number"
  ? breakdownFromResponse.platformFeeCents
  : null;
const totalCentsNumber =
  typeof breakdownFromResponse?.totalCents === "number"
  ? breakdownFromResponse.totalCents
  : typeof data.amount === "number"
  ? data.amount
  : null;
const currencyFromResponse =
  typeof breakdownFromResponse?.currency === "string" ? breakdownFromResponse.currency : undefined;

const statusFromResponse =
  typeof data?.status === "string" ? data.status.toUpperCase() : null;
const nextActionFromResponse =
  typeof data?.nextAction === "string" ? data.nextAction : null;
const paymentIntentIdFromResponse =
  typeof data?.paymentIntentId === "string" ? data.paymentIntentId : null;

if (statusFromResponse === "FAILED") {
  setClientSecret(null);
  setServerAmount(null);
  setBreakdown(null);
  setError(typeof data?.error === "string" ? data.error : "Pagamento falhou.");
  return;
}

if (data.freeCheckout || data.isFreeCheckout || statusFromResponse === "PAID") {
  const totalCents = totalCentsNumber ?? 0;
  setBreakdown(breakdownFromResponse);
  setAppliedDiscount(discountCentsNumber > 0 ? discountCentsNumber / 100 : 0);
  setAppliedPromoLabel(promoLabel);
  setClientSecret(null);
  setServerAmount(0);
  atualizarDados({
    ...
  });
}

```

```

        additional: {
          ... (safeDados?.additional ?? {}),
          paymentIntentId: paymentIntentIdFromResponse ?? FREE_PLACEHOLDER_INTENT_ID,
          purchaseId: purchaseIdFromServer,
          subtotalCents: subtotalCentsNumber ?? undefined,
          discountCents: discountCentsNumber ?? undefined,
          platformFeeCents: platformFeeCentsNumber ?? undefined,
          totalCents: totalCents,
          currency: currencyFromResponse ?? undefined,
          total: totalCents / 100,
          freeCheckout: true,
          clientFingerprint,
          intentFingerprint: effectiveIntentFingerprint,
          idempotencyKey: effectiveIdemKey ?? undefined,
          promoCode: payload?.promoCode,
          promoCodeRaw: payload?.promoCode,
          appliedPromoLabel: promoLabel ?? undefined,
          paymentScenario: paymentScenarioResponse ?? undefined,
        },
      });
      lastIntentKeyRef.current = intentKey;
      irParaPasso(3);
      return;
    }

    setClientSecret(data.clientSecret as string);
    setServerAmount(typeof data.amount === "number" ? data.amount : null);
    setBreakdown(breakdownFromResponse);
    setAppliedDiscount(discountCentsNumber > 0 ? discountCentsNumber / 100 : 0);
    setAppliedPromoLabel(promoLabel);
    atualizarDados({
      paymentScenario: paymentScenarioResponse ?? undefined,
      additional: {
        ... (safeDados?.additional ?? {}),
        clientFingerprint,
        intentFingerprint: effectiveIntentFingerprint,
        idempotencyKey: effectiveIdemKey ?? undefined,
        purchaseId:
          purchaseIdFromServer ??
          (safeDados?.additional as Record<string, unknown> | undefined)?.purchaseId ??
          (payload as any)?.purchaseId,
        paymentIntentId:
          paymentIntentIdFromResponse ??
          safeDados?.additional?.paymentIntentId,
        subtotalCents: subtotalCentsNumber ?? undefined,
        discountCents: discountCentsNumber ?? undefined,
        platformFeeCents: platformFeeCentsNumber ?? undefined,
        totalCents: totalCentsNumber ?? undefined,
        currency: currencyFromResponse ?? undefined,
        promoCode: payload?.promoCode,
        promoCodeRaw: payload?.promoCode,
        appliedPromoLabel: promoLabel ?? undefined,
      },
    });
    lastIntentKeyRef.current = intentKey;
    setCachedIntent({
      key: intentKey,
      clientSecret: data.clientSecret as string,
      amount: typeof data.amount === "number" ? data.amount : null,
      breakdown: breakdownFromResponse,
      discount: discountCentsNumber > 0 ? discountCentsNumber / 100 : 0,
      freeCheckout: false,
      paymentScenario: paymentScenarioResponse,
      promoLabel,
      autoAppliedPromo: isAutoAppliedPromo,
      purchaseId: purchaseIdFromServer ?? null,
    });
  }
}

} catch (err) {
  console.error("Erro ao criar PaymentIntent:", err);
  if (!cancelled) {
    setError("Erro inesperado ao preparar o pagamento.");
  }
} finally {
  if (!cancelled) setLoading(false);
  if (inFlightIntentRef.current === intentKey) {
    inFlightIntentRef.current = null;
  }
}

```

```

        }

    }

    createIntent();

    return () => {
      cancelled = true;
      inFlightIntentRef.current = null;
    };
}, [
  payload,
  irParaPasso,
  authChecked,
  userId,
  stripePromise,
  purchaseMode,
  guestSubmitVersion,
  cachedIntent,
]);
}

if (!safeDados) {
  return (
    <div className="p-6 text-sm text-white/70">
      Ocorreu um problema com os dados do checkout. Volta atrás e tenta de novo.
    </div>
  );
}

const additional =
  safeDados.additional && typeof safeDados.additional === "object"
    ? safeDados.additional
    : {};
const totalFromContext = typeof additional.total === "number" ? additional.total : null;

const breakdownTotal =
  breakdown && typeof breakdown.totalCents === "number" ? breakdown.totalCents / 100 : null;

const total =
  breakdownTotal !== null
    ? breakdownTotal
    : totalFromContext !== null
      ? totalFromContext
      : serverAmount !== null
        ? serverAmount / 100
        : null;

const appearance: Appearance = {
  theme: "night",
  variables: {
    colorPrimary: "#FF00C8",
    colorBackground: "#0A0A0F",
    colorText: "#F5F5F5",
    colorDanger: "#FF4242",
    fontFamily:
      "Inter, system-ui, -apple-system, BlinkMacSystemFont, sans-serif",
    borderRadius: "16px",
  },
  rules: {
    ".Input": {
      padding: "14px",
      backgroundColor: "#11131A",
      border: "1px solid rgba(255,255,255,0.08)",
    },
  },
};

const options: StripeElementsOptions | undefined = clientSecret
? {
  clientSecret,
  appearance,
  // Nota: a lista de métodos permitidos é definida no PaymentIntent (backend).
}
: undefined;

const loadErrorCountRef = useRef(0);

const handlePaymentElementError = () => {

```

```

// Evita loop infinito: só tentamos regenerar 1x automaticamente
if (loadErrorCountRef.current >= 1) return;
loadErrorCountRef.current += 1;

setCachedIntent(null);
setClientSecret(null);
setServerAmount(null);
setBreakdown(null);
lastIntentKeyRef.current = null;
inFlightIntentRef.current = null;

let nextIdemKey: string | undefined;
try {
  nextIdemKey = crypto.randomUUID();
} catch {
  nextIdemKey = undefined;
}

atualizarDados({
  additional: {
    ... (safeDados?.additional ?? {}),
    purchaseId: null,
    paymentIntentId: undefined,
    freeCheckout: undefined,
    appliedPromoLabel: undefined,
    intentFingerprint: undefined,
    idempotencyKey: nextIdemKey ?? (additionalObj as any).idempotencyKey,
  },
});
};

// Callback chamado pelo AuthWall quando o utilizador faz login/cria conta com sucesso
const handleAuthenticated = async (newUserId: string) => {
  setUserId(newUserId);
  setAuthChecked(true);
  setAuthChecking(false);
  setPurchaseMode("auth");

  // Tentar migrar bilhetes de guest para este user (best-effort)
  try {
    await fetch("/api/tickets/migrate-guest", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
    });
  } catch (err) {
    console.warn("[Step2Pagamento] Falha ao migrar bilhetes de convidado", err);
  }
};

// Callback para continuar como convidado
const handleGuestContinue = () => {
  if (requiresAuth) {
    setError("Este tipo de checkout requer sessão iniciada.");
    setPurchaseMode("auth");
    setAuthInfo("Inicia sessão para continuar.");
    return;
  }
  setError(null);
  const localErrors: { name?: string; email?: string; phone?: string } = {};
  if (!guestName.trim()) {
    localErrors.name = "Nome é obrigatório para emitir o bilhete.";
  }
  if (!guestEmail.trim()) {
    localErrors.email = "Email é obrigatório para enviar os bilhetes.";
  } else if (!isValidEmail(guestEmail.trim())) {
    localErrors.email = "Email inválido. Confirma o formato (ex: nome@dominio.com).";
  } else if (guestEmailConfirm.trim() && guestEmailConfirm.trim() !== guestEmail.trim()) {
    localErrors.email = "Email e confirmação não coincidem.";
  }

  const phoneNormalized = sanitizePhone(guestPhone);
  if (phoneNormalized) {
    if (!isValidPhone(phoneNormalized)) {
      localErrors.phone = "Telemóvel inválido. Usa apenas dígitos e opcional + no início.";
    }
  }
  setGuestErrors(localErrors);
};

```

```

if (localErrors.name || localErrors.email || localErrors.phone) {
  setError("Revê os dados para continuar como convidado.");
  return;
}

atualizarDados({
  additional: {
    ...(safeDados?.additional ?? {}),
    guestName: guestName.trim(),
    guestEmail: guestEmail.trim(),
    guestEmailConfirm: guestEmailConfirm.trim(),
    guestPhone: phoneNormalized || undefined,
    // Reset de estado para evitar reutilização de intents/purchase antigos
    purchaseId: null,
    paymentIntentId: undefined,
    freeCheckout: undefined,
    appliedPromoLabel: undefined,
    clientFingerprint: undefined,
    intentFingerprint: undefined,
    idempotencyKey: crypto.randomUUID(),
  },
});
};

setPurchaseMode("guest");
setClientSecret(null);
setServerAmount(null);
setGuestSubmitVersion((v) => v + 1);
};

const showPaymentUI =
  (!authChecking && Boolean(userId)) ||
  (!isFreeScenario && purchaseMode === "guest" && guestSubmitVersion > 0);

const handleRemovePromo = () => {
  setPromoCode("");
  setPromoInput("");
  setAppliedDiscount(0);
  setAppliedPromoLabel(null);
  setPromoWarning(null);
  setError(null);
  setBreakdown(null);
  setClientSecret(null);
  setServerAmount(null);
  setGuestSubmitVersion((v) => v + 1);

  try {
    atualizarDados({
      additional: {
        ...(safeDados?.additional ?? {}),
        purchaseId: null,
        paymentIntentId: undefined,
        freeCheckout: undefined,
        appliedPromoLabel: undefined,
        clientFingerprint: undefined,
        intentFingerprint: undefined,
        idempotencyKey: crypto.randomUUID(),
      },
    });
  } catch {}
};

lastIntentKeyRef.current = null;
inFlightIntentRef.current = null;
};

return (
  <div className="flex flex-col gap-6 text-white">
    <header className="flex items-start justify-between gap-3">
      <div className="space-y-1">
        <p className="text-[11px] uppercase tracking-[0.18em] text-white/55">
          Passo 2 de 3
        </p>
        <h2 className="text-xl font-semibold leading-tight">
          {isFreeScenario ? "Inscrição gratuita" : "Pagamento"}
        </h2>
        <p className="text-[11px] text-white/60 max-w-xs">
          {isFreeScenario
            ? "Confirma a tua inscrição. Requer sessão iniciada e username definido."
            : "Pagamento seguro processado pela Stripe."}
        </p>
      </div>
    </header>
  </div>
);

```

```

        </p>
        {scenario && scenarioCopy[scenario] &&
         <p className="text-[11px] text-white/75 max-w-sm">{scenarioCopy[scenario]}</p>
        )}
      </div>
    </header>

    <div className="h-1 w-full rounded-full bg-white/10 overflow-hidden">
      <div className="h-full w-2/3 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5]" />
    </div>

    {/* 🛡 Se ainda estamos a verificar auth */}
    {authChecking && (
      <div className="flex-1 rounded-2xl border border-white/10 bg-white/[0.03] px-6 py-12 flex flex-col justify-center items-center text-center shadow-[0_0_40px_rgba(255,0,200,0.25)]">
        <div className="relative mb-6">
          <div className="h-14 w-14 rounded-full border-2 border-white/20 border-t-transparent animate-spin" />
          <div className="absolute inset-0 h-14 w-14 animate-pulse rounded-full border border-[#6BFFFF]/20" />
        </div>
        <h3 className="text-sm font-semibold mb-1 animate-pulse">
          A verificar sessão...
        </h3>
        <p className="text-[11px] text-white/65 max-w-xs leading-relaxed">
          Estamos a confirmar se já tens sessão iniciada na ORYA.
        </p>
      </div>
    )}

    {/* 💳 UI de pagamento (user logado OU convidado depois de validar form) */}
    {!authChecking && showPaymentUI ? (
      <>
        {error || (needsStripe && !stripePromise) ? (
          <div className="flex-1 rounded-2xl border border-red-500/30 bg-red-500/10 px-5 py-6 text-sm text-red-100 shadow-[0_0_30px_rgba(255,0,0,0.35)]">
            <p className="font-semibold mb-1 flex items-center gap-2">
              <span className="text-lg" style="color: red;">⚠ Ocorreu um problema
            </p>
            <p className="text-[12px] mb-4 leading-relaxed">
              {error ?? "Configuração de pagamentos indisponível. Tenta novamente mais tarde."}
            </p>
            <button
              type="button"
              onClick={() => window.location.reload()}
              className="rounded-full bg-white text-red-700 px-5 py-1.5 text-[11px] font-semibold shadow hover:bg-white/90 transition">
              Tentar novamente
            </button>
          </div>
        ) : loading || (needsStripe && (!clientSecret || !options)) ? (
          <div className="flex-1 rounded-2xl border border-white/10 bg-white/[0.03] px-6 py-12 flex flex-col justify-center items-center text-center shadow-[0_0_40px_rgba(255,0,200,0.25)]">
            <div className="relative mb-6">
              <div className="h-14 w-14 rounded-full border-2 border-white/20 border-t-transparent animate-spin" />
              <div className="absolute inset-0 h-14 w-14 animate-pulse rounded-full border border-[#6BFFFF]/20" />
            </div>
            <h3 className="text-sm font-semibold mb-1 animate-pulse">
              A preparar o teu pagamento...
            </h3>
            <p className="text-[11px] text-white/65 max-w-xs leading-relaxed">
              Estamos a ligar-te à Stripe para criar uma transação segura.
            </p>
          </div>
        ) : error ? (
          <div className="flex-1 rounded-2xl border border-red-500/30 bg-red-500/10 px-5 py-6 text-sm text-red-100 shadow-[0_0_30px_rgba(255,0,0,0.35)]">
            <p className="font-semibold mb-1 flex items-center gap-2">
              <span className="text-lg" style="color: red;">⚠ Ocorreu um problema
            </p>
            <p className="text-[12px] mb-4 leading-relaxed">{error}</p>
            <button
              type="button"
              onClick={() => window.location.reload()}
              className="rounded-full bg-white text-red-700 px-5 py-1.5 text-[11px] font-semibold shadow hover:bg-white/90 transition">
              Tentar novamente
            </button>
          </div>
        )
      </>
    )} 
  
```

```

        </div>
    ) : (
        <div className="flex-1 rounded-2xl border border-white/12 bg-white/[0.05] px-6 py-6 shadow-[0_0_60px_rgba(0,0,0,0.6)] overflow-y-auto max-h-[65vh] space-y-4">
            {promoWarning && (
                <div className="rounded-md border border-amber-400/40 bg-amber-400/10 px-3 py-2 text-sm text-amber-100">
                    {promoWarning}
                </div>
            )}
            {appliedPromoLabel === "Promo automática" && appliedDiscount > 0 && (
                <div className="rounded-md border border-emerald-400/40 bg-emerald-500/10 px-3 py-2 text-xs text-emerald-100">
                    Desconto aplicado automaticamente ✨
                </div>
            )}
            <div className="rounded-2xl border border-white/10 bg-black/30 p-4 space-y-2">
                <label className="text-xs text-white/70">Tens um código promocional?</label>
                <div className="flex flex-col gap-2 sm:flex-row">
                    <input
                        type="text"
                        value={promoInput}
                        onChange={(e) => setPromoInput(e.target.value)}
                        placeholder="Insere o código"
                        className="flex-1 rounded-xl bg-black/50 border border-white/15 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]"
                    />
                    <button
                        type="button"
                        onClick={() => {
                            setPromoWarning(null);
                            setError(null);
                            if (!promoInput.trim()) {
                                setPromoWarning("Escreve um código antes de aplicar.");
                                return;
                            }
                            // Promo altera o cálculo: limpamos purchase/payment state para obrigar a recalcular intent.
                            try {
                                atualizarDados({
                                    additional: {
                                        ... (safeDados?.additional ?? {}),
                                        purchaseId: null,
                                        paymentIntentId: undefined,
                                        freeCheckout: undefined,
                                        appliedPromoLabel: undefined,
                                        intentFingerprint: undefined,
                                        idempotencyKey: crypto.randomUUID(),
                                    },
                                });
                            } catch {}
                            setCachedIntent(null);
                            setClientSecret(null);
                            setServerAmount(null);
                            setBreakdown(null);
                            lastIntentKeyRef.current = null;
                            inFlightIntentRef.current = null;
                            setPromoCode(promoInput.trim());
                            setGuestSubmitVersion((v) => v + 1);
                        }}
                    >
                        Aplicar
                    </button>
                </div>
            <div className="flex flex-wrap items-center gap-2 rounded-lg border border-emerald-400/30 bg-emerald-500/10 px-3 py-2 text-xs text-emerald-100">
                <span>
                    {appliedPromoLabel
                    ? `Desconto ${appliedPromoLabel}: -${appliedDiscount.toFixed(2)} €`
                    : `Desconto aplicado: -${appliedDiscount.toFixed(2)} €`}
                </span>
                <button
                    type="button"
                    onClick={handleRemovePromo}
                    className="rounded-full border border-emerald-300/40 px-2 py-0.5 text-[11px] text-emerald-50 hover:bg-emerald-500/20"
                >
                    Remover
                </button>
            </div>
        </div>
    )
}

```



```

        onChangeEmailConfirm={setGuestEmailConfirm}
        onChangePhone={setGuestPhone}
        onContinue={handleGuestContinue}
      />
    ) : (
      <AuthWall onAuthenticated={handleAuthenticated} />
    )
  </div>
)
</div>
);
}

type PaymentFormProps = {
  total: number | null;
  discount?: number;
  breakdown?: CheckoutBreakdown;
  clientSecret: string | null;
  onLoadError?: () => void;
};

function PaymentForm({ total, discount = 0, breakdown, clientSecret, onLoadError }: PaymentFormProps) {
  const stripe = useStripe();
  const elements = useElements();
  const { irParaPasso, atualizarDados, dados } = useCheckout();
  const [submitting, setSubmitting] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [elementReady, setElementReady] = useState(false);
  const currency = breakdown?.currency ?? "EUR";
  const discountCents = Math.max(0, Math.round(discount * 100));
  const hasInvoice = Boolean(breakdown?.lines?.length);
  const platformFeeCents =
    breakdown && breakdown.feeMode === "ADDED" ? breakdown.platformFeeCents : 0;
  const subtotalCents = breakdown?.subtotalCents ?? 0;
  const baseSubtotalCents =
    hasInvoice && discountCents > 0 ? subtotalCents + discountCents : subtotalCents;
  const promoApplied = discountCents > 0;

  useEffect(() => {
    // sempre que o clientSecret muda, obrigamos o PaymentElement a fazer ready novamente
    setElementReady(false);
  }, [clientSecret]);

  // Se o utilizador for redirecionado pela Stripe (ex.: 3DS), o URL volta com
  // `payment_intent_client_secret` e `redirect_status`. Aqui recuperamos o PI
  // e avançamos automaticamente para o passo 3 quando o pagamento fica concluído.
  useEffect(() => {
    if (!stripe) return;
    if (typeof window === "undefined") return;

    const params = new URLSearchParams(window.location.search);
    const clientSecretFromUrl = params.get("payment_intent_client_secret");

    if (!clientSecretFromUrl) return;

    let cancelled = false;

    (async () => {
      try {
        // Limpamos o erro antigo para não confundir o utilizador.
        setError(null);

        const result = await stripe.retrievePaymentIntent(clientSecretFromUrl);

        if (cancelled) return;

        if (result.error) {
          setError(result.error.message ?? "Não foi possível confirmar o estado do pagamento.");
          return;
        }

        const paymentIntent = result.paymentIntent;
        if (!paymentIntent) {
          setError("Não foi possível confirmar o estado do pagamento.");
          return;
        }

        if (paymentIntent.status === "succeeded") {

```

```

atualizarDados({
  additional: {
    ...dados?.additional ?? {},
    paymentIntentId: paymentIntent.id,
  },
});

try {
  const { trackEvent } = await import("@/lib/analytics");
  trackEvent("checkout_payment_confirmed", {
    eventId: dados?.eventId,
    promoApplied: discountCents > 0,
    currency,
    totalCents: total ? Math.round(total * 100) : null,
    viaRedirect: true,
  });
} catch (err) {
  console.warn("[trackEvent] checkout_payment_confirmed (redirect) falhou", err);
}

irParaPasso(3);
return;
}

if (paymentIntent.status === "processing") {
  setError("Pagamento em processamento. Aguarda uns segundos e verifica o teu email.");
  return;
}

if (paymentIntent.status === "requires_payment_method") {
  setError("O pagamento não foi concluído. Tenta novamente ou usa outro método.");
  return;
}

// Para outros estados, mostramos uma mensagem genérica.
setError("O pagamento não ficou concluído. Tenta novamente.");
} catch (err) {
  console.error("[PaymentForm] Erro a recuperar PaymentIntent do redirect:", err);
  if (!cancelled) setError("Não foi possível confirmar o estado do pagamento.");
} finally {
  // Removemos os query params de redirect para evitar loops se o utilizador fizer refresh.
  try {
    const url = new URL(window.location.href);
    url.searchParams.delete("payment_intent");
    url.searchParams.delete("payment_intent_client_secret");
    url.searchParams.delete("redirect_status");
    window.history.replaceState({}, "", url.toString());
  } catch {}
}
})();

return () => {
  cancelled = true;
};

[stripe, atualizarDados, dados?.additional, dados?.eventId, irParaPasso, currency, total, discountCents]);
}

async function handleSubmit(e: React.FormEvent) {
  e.preventDefault();
  if (!stripe || !elements || !elementReady) return;

  setSubmitting(true);
  setError(null);

  try {
    const returnUrl =
      typeof window !== "undefined" ? window.location.href : undefined;

    const { error, paymentIntent } = await stripe.confirmPayment({
      elements,
      confirmParams: returnUrl ? { return_url: returnUrl } : undefined,
      redirect: "if_required",
    });

    if (error) {
      setError(error.message ?? "O pagamento não foi concluído.");
      return;
    }
  }
}

```

```

    if (paymentIntent && paymentIntent.status === "succeeded") {
      atualizarDados({
        additional: {
          ...(dados?.additional ?? {}),
          paymentIntentId: paymentIntent.id,
        },
      });
      try {
        const { trackEvent } = await import("@/lib/analytics");
        trackEvent("checkout_payment_confirmed", {
          eventId: dados?.eventId,
          promoApplied,
          currency,
          totalCents: total ? Math.round(total * 100) : null,
        });
      } catch (err) {
        console.warn("[trackEvent] checkout_payment_confirmed falhou", err);
      }
      irParaPasso(3);
    }
  } catch (err) {
    console.error("Erro ao confirmar pagamento:", err);
    setError("Erro inesperado ao confirmar o pagamento.");
  } finally {
    setSubmitting(false);
  }
}

return (
  <form onSubmit={handleSubmit} className="flex flex-col gap-5">
    {(hasInvoice || total !== null) &&
      <div className="rounded-2xl border border-white/10 bg-black/40 px-5 py-4 shadow-inner shadow-black/40 space-y-3">
        <div className="flex items-center justify-between text-xs text-white/70">
          <span className="uppercase tracking-[0.14em]">Resumo</span>
          <span className="inline-flex items-center gap-1 rounded-full bg-white/5 px-3 py-1 border border-white/10 text-[11px] text-white/70">
             Pagamento seguro
          </span>
        </div>
        {hasInvoice && (
          <div className="space-y-2">
            {breakdown?.lines?.map((line) => (
              <div key={`${line.ticketTypeId}-${line.name}-${line.quantity}`}
                className="flex items-center justify-between text-sm text-white/80">
                <div className="flex flex-col">
                  <span className="font-medium">{line.name}</span>
                  <span className="text-[11px] text-white/55">x{line.quantity}</span>
                </div>
                <div className="text-right">
                  <p className="text-[13px] font-semibold">
                    {formatMoney(line.lineTotalCents, line.currency || currency)}
                  </p>
                  <p className="text-[11px] text-white/45">
                    {formatMoney(line.unitPriceCents, line.currency || currency)} / bilhete
                  </p>
                </div>
              </div>
            )));
        )
      <div className="h-px w-full bg-white/10" />
      {discountCents > 0 && (
        <>
          <div className="flex items-center justify-between text-sm text-white/70">
            <span>Subtotal (antes de desconto)</span>
            <span className="font-semibold">
              {formatMoney(baseSubtotalCents, currency)}
            </span>
          </div>
          <div className="flex items-center justify-between text-sm text-emerald-300">
            <span>Desconto aplicado</span>
            <span>-{formatMoney(discountCents, currency)}</span>
          </div>
        </>
      )})
    }
  
```

```

<div className="flex items-center justify-between text-sm text-white/80">
  <span>Subtotal</span>
  <span className="font-semibold">
    {formatMoney(subtotalCents, currency)}
  </span>
</div>

{platformFeeCents > 0 && (
  <div className="flex items-center justify-between text-sm text-white/70">
    <span>Taxas de serviço</span>
    <span>{formatMoney(platformFeeCents, currency)}</span>
  </div>
)
</div>
)}

{total !== null && (
  <div className="flex items-center justify-between rounded-xl bg-white/5 px-4 py-3 border border-white/10">
    <div className="flex flex-col text-white/80">
      <span className="text-[12px]">Total a pagar</span>
      {breakdown?.feeMode === "INCLUDED" &&
        <span className="text-[11px] text-white/55">
          Taxas já incluídas
        </span>
      }
    </div>
    <span className="text-xl font-semibold text-white">
      {formatMoney(Math.round(total * 100), currency)}
    </span>
  </div>
)
</div>
)}

<div className="rounded-xl bg-black/40 px-3 py-3 text-sm min-h-[320px] max-h-[400px] overflow-y-auto pr-1 payment-scroll">
  <PaymentElement
    // key força remount quando o clientSecret muda para evitar usar intents antigos
    key={clientSecret ?? "payment-element"}
    options={{
      // Ordem preferida; apenas métodos autorizados (Stripe: Card/Link/MB WAY – Apple Pay vem via Card)
      paymentMethodOrder: ["card", "link", "mb_way"],
    }}
    onReady={() => setElementReady(true)}
    onLoadError={(err) => {
      console.error("[PaymentElement] loaderror", err);
      setElementReady(false);
      setError("Não foi possível carregar o formulário de pagamento. Tenta novamente.");
      if (onLoadError) onLoadError();
    }}
  />
</div>

{error && (
  <p className="text-[11px] text-red-300 mt-1 leading-snug">{error}</p>
)}

<button
  type="submit"
  disabled={submitting || !stripe || !elements || !elementReady}
  className="mt-3 inline-flex w-full items-center justify-center rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFF] to-[#1646F5] px-6 py-3 text-xs font-semibold text-black shadow-[0_0_32px_rgba(107,255,255,0.55)] disabled:opacity-40 disabled:cursor-not-allowed hover:scale-[1.03] active:scale-95 transition-transform">
  >
  {submitting ? "A processar..." : "Pagar agora"}
</button>

<p className="mt-2 text-[10px] text-white/40 text-center leading-snug">
  Pagamento seguro processado pela Stripe. A ORYA nunca guarda dados do
  teu cartão.
</p>
</form>
);
}

type AuthWallProps = {
  onAuthenticated?: (userId: string) => void;
};

```

```

function delay(ms: number) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}

function AuthWall({ onAuthenticated }: AuthWallProps) {
  const [mode, setMode] = useState<"login" | "signup" | "verify">("login");
  const [identifier, setIdentifier] = useState("");
  const [password, setPassword] = useState("");
  const [confirmPassword, setConfirmPassword] = useState("");
  const [fullName, setFullName] = useState("");
  const [username, setUsername] = useState("");
  const [submitting, setSubmitting] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [otp, setOtp] = useState("");
  const [authOtpCooldown, setAuthOtpCooldown] = useState(0);
  const [authOtpResending, setAuthOtpResending] = useState(false);

  function isUnconfirmedError(err: unknown) {
    if (!err) return false;
    const anyErr = err as { message?: string; status?: number; error_description?: string };
    const msg = (anyErr.message || anyErr.error_description || "").toLowerCase();
    return (
      msg.includes("not confirmed") ||
      msg.includes("confirm your email") ||
      msg.includes("email_not_confirmed")
    );
  }

  useEffect(() => {
    if (authOtpCooldown <= 0) return;
    const t = setInterval(() => {
      setAuthOtpCooldown((prev) => (prev > 0 ? prev - 1 : 0));
    }, 1000);
    return () => clearInterval(t);
  }, [authOtpCooldown]);

  async function triggerResendOtp(email: string) {
    setError(null);
    setAuthOtpResending(true);
    setAuthOtpCooldown(60);
    try {
      const res = await fetch("/api/auth/resend-otp", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email }),
      });
      const data = await res.json().catch(() => null);
      if (!res.ok) {
        setError(data?.error ?? "Não foi possível reenviar o código.");
        setAuthOtpCooldown(0);
      }
    } catch (err) {
      console.error("[AuthWall] resend OTP error:", err);
      setError("Não foi possível reenviar o código.");
      setAuthOtpCooldown(0);
    } finally {
      setAuthOtpResending(false);
    }
  }

  async function handleGoogle() {
    setSubmitting(true);
    setError(null);
    try {
      const redirectTo =
        typeof window !== "undefined"
          ? `${window.location.origin}/auth/callback?redirect=${encodeURIComponent(window.location.href)}`
          : undefined;
      if (typeof window !== "undefined") {
        try {
          localStorage.setItem("orya_post_auth_redirect", window.location.href);
        } catch {}
      }
      const { error } = await supabaseBrowser.auth.signInWithOAuth({
        provider: "google",
        options: { redirectTo },
      });
    }
  }
}

```

```

        if (error) {
            setError(error.message ?? "Não foi possível iniciar sessão com Google.");
        }
    } catch (err) {
        console.error("[AuthWall] Google OAuth error:", err);
        setError("Não foi possível iniciar sessão com Google.");
    } finally {
        setSubmitting(false);
    }
}

async function handleSubmit(e: React.FormEvent) {
    e.preventDefault();
    setSubmitting(true);
    setError(null);

    try {
        if (mode === "verify") {
            if (!identifier || !otp.trim()) {
                setError("Indica o email e o código recebido.");
                return;
            }
            const emailToUse = identifier.trim().toLowerCase();
            const token = otp.trim();
            const { error: verifyErr } = await supabaseBrowser.auth.verifyOtp({
                type: "signup",
                email: emailToUse,
                token,
            });
            if (verifyErr) {
                setError(verifyErr.message || "Código inválido ou expirado.");
                setAuthOtpCooldown(0);
                return;
            }
            await delay(400);
            const { data: userData } = await supabaseBrowser.auth.getUser();
            if (userData?.user) onAuthenticated?(userData.user.id);
            return;
        }

        if (!identifier || !password) {
            setError("Preenche o email e a palavra-passe.");
            return;
        }

        let emailToUse = identifier.trim().toLowerCase();
        if (!identifier.includes("@")) {
            const res = await fetch("/api/auth/resolve-identifier", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ identifier }),
            });
            const data = await res.json().catch(() => null);
            if (!res.ok || !data?.ok || !data?.email) {
                setError("Credenciais inválidas. Confirma username/email e password.");
                return;
            }
            emailToUse = data.email;
        }

        if (mode === "login") {
            const { error } = await supabaseBrowser.auth.signInWithEmailAndPassword({
                email: emailToUse,
                password,
            });
            if (error) {
                if (isUnconfirmedError(error)) {
                    setMode("verify");
                    setIdentifier(emailToUse);
                    setError("Email ainda não confirmado. Reenviei-te um novo código.");
                    await triggerResendOtp(emailToUse);
                    return;
                }
                setError(error.message ?? "Não foi possível iniciar sessão.");
                return;
            }
        } else {
            if (password.length < 6) {

```

```

        setError("A password deve ter pelo menos 6 caracteres.");
        return;
    }
    if (password !== confirmPassword) {
        setError("As passwords não coincidem.");
        return;
    }
    if (!fullName.trim()) {
        setError("Nome é obrigatório para criar conta.");
        return;
    }
    const usernameCheck = await checkUsernameAvailability(username);
    if (!usernameCheck.ok) {
        setSubmitting(false);
        return;
    }

    const res = await fetch("/api/auth/send-otp", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
            email: emailToUse,
            password,
            username: usernameCheck.username,
            fullName: fullName.trim(),
        }),
    });
    const data = await res.json().catch(() => null);
    if (!res.ok || !data?.ok) {
        const message = data?.error ?? "Não foi possível enviar o código de verificação.";
        setError(message);
        return;
    }

    setMode("verify");
    setIdentifier(emailToUse);
    setError("Enviámos um código para confirmar o email. Introduz para continuares.");
    setAuthOtpCooldown(60);
    return;
}

// Pequeno delay para garantir que a sessão foi escrita nos cookies
await delay(600);

// Depois do delay, garantimos que a sessão está disponível e notificamos o Step2
try {
    const { data: userData } = await supabaseBrowser.auth.getUser();
    if (userData?.user) {
        onAuthenticated?(userData.user.id);
    }
} catch (e: unknown) {
    console.warn("[AuthWall] Não foi possível ler user após login:", e);
}
} catch (err) {
    console.error("[AuthWall] Erro:", err);
    setError("Ocorreu um erro. Tenta novamente.");
} finally {
    setSubmitting(false);
}

return (
    <div className="flex-1 rounded-2xl border border-white/12 bg-white/[0.05] px-6 py-6 shadow-[0_0_-40px_rgba(0,0,0,0.6)] flex flex-col gap-4">
        <div className="flex items-start justify-between gap-4">
            <div>
                <h3 className="text-sm font-semibold mb-1">
                    Inicia sessão para continuar
                </h3>
                <p className="text-[11px] text-white/60 max-w-sm leading-relaxed">
                    Para associar os bilhetes à tua conta ORYA e evitar problemas no
                    check-in, tens de estar com a sessão iniciada antes de pagar.
                </p>
            </div>
            <span className="text-[20px]">🔗</span>
        </div>
        <form onSubmit={handleSubmit} className="flex flex-col gap-3 mt-2">

```

```

{mode !== "verify" && (
  <div className="flex gap-2 text-[11px] bg-black/40 rounded-full p-1 border border-white/10 w-fit">
    <button
      type="button"
      onClick={() => setMode("login")}
      className={`${px-3 py-1 rounded-full ${mode === "login" ? "bg-white text-black font-semibold" : "text-white/70"}`}>
      Já tenho conta
    </button>
    <button
      type="button"
      onClick={() => setMode("signup")}
      className={`${px-3 py-1 rounded-full ${mode === "signup" ? "bg-white text-black font-semibold" : "text-white/70"}`}>
      Criar conta
    </button>
  </div>
)})

<div className="flex flex-col gap-2 text-[12px]">
  {mode === "verify" ? (
    <>
      <div className="flex flex-col gap-1">
        <label className="text-white/70">Email</label>
        <input
          type="email"
          className="w-full rounded-xl bg-black/60 border border-white/15 px-3 py-2 text-[12px] outline-none
focus:border-[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]"
          placeholder="nome@exemplo.com"
          value={identifier}
          onChange={(e) => setIdentifier(e.target.value)}
          autoComplete="email"
        />
      </div>
      <div className="flex flex-col gap-1">
        <label className="text-white/70">Palavra-passe</label>
        <input
          type="password"
          className="w-full rounded-xl bg-black/60 border border-white/15 px-3 py-2 text-[12px] outline-none
focus:border-[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]"
          placeholder="*****"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          autoComplete={mode === "login" ? "current-password" : "new-password"}
        />
      </div>
    </>
  ) : mode === "signup" && (
    <>
      <div className="flex flex-col gap-1">
        <label className="text-white/70">Confirmar palavra-passe</label>
        <input
          type="password"
          className="w-full rounded-xl bg-black/60 border border-white/15 px-3 py-2 text-[12px] outline-none
focus:border-[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]"
          placeholder="*****"
          value={confirmPassword}
          onChange={(e) => setConfirmPassword(e.target.value)}
          autoComplete="new-password"
        />
      </div>
      <div className="flex flex-col gap-1">
        <label className="text-white/70">Nome completo</label>
        <input
          type="text"
          className="w-full rounded-xl bg-black/60 border border-white/15 px-3 py-2 text-[12px] outline-none
focus:border-[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]"
          placeholder="O teu nome"
          value={fullName}
          onChange={(e) => setFullName(e.target.value)}
          autoComplete="name"
        />
      </div>
      <div className="flex flex-col gap-1">
        <label className="text-white/70">Username</label>
        <input
          type="text"
          className="w-full rounded-xl bg-black/60 border border-white/15 px-3 py-2 text-[12px] outline-none
focus:border-[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]"
        />
      </div>
    </>
  ) : null
</div>
)

```

```

        placeholder="@teuuser"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
        autoComplete="username"
      />
    </div>
  </>
)
</>
) : (
<>
  <p className="text-[12px] text-white/70">
    Enviámos um código de confirmação para <strong>{identifier}</strong>. Introduz abaixo ou pede novo código.
  </p>
  <div className="flex flex-col gap-1">
    <label className="text-white/70">Código</label>
    <input
      type="text"
      maxLength={8}
      className="w-full rounded-xl bg-black/60 border border-white/15 px-3 py-2 text-[12px] outline-none
focus:border-[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]"
      placeholder="87612097"
      value={otp}
      onChange={(e) => setOtp(e.target.value)}
      autoComplete="one-time-code"
    />
  </div>
  <div className="text-[11px] text-white/65 flex items-center justify-between">
    <span>
      Não chegou? {authOtpCooldown > 0 ? `Podes reenviar em ${authOtpCooldown}s.` : "Reenvia um novo código."}
    </span>
    <button
      type="button"
      onClick={() => identifier && triggerResendOtp(identifier)}
      disabled={authOtpCooldown > 0 || authOtpResending || !identifier}
      className="text-[#6BFFFF] hover:text-white transition disabled:opacity-50"
    >
      Reenviar código
    </button>
  </div>
</>
)
</div>

(error && (
  <p className="text-[11px] text-red-300 mt-1 leading-snug">{error}</p>
))
(authOtpCooldown > 0 && mode === "verify" && (
  <p className="text-[11px] text-white/60">Podes reenviar código em {authOtpCooldown}s.</p>
))

<button
  type="button"
  onClick={handleGoogle}
  disabled={submitting}
  className="inline-flex w-full items-center justify-center gap-2 rounded-full border border-white/20 bg-black/50 px-6
py-2.5 text-xs font-semibold text-white shadow hover:border-white/40 hover:bg-black/60 transition-colors disabled:opacity-50"
>
  <span>Continuar com Google</span>
</button>

<button
  type="submit"
  disabled={submitting}
  className="mt-2 inline-flex w-full items-center justify-center rounded-full bg-gradient-to-r from-[#FF00C8] via-
[#6BFFFF] to-[#1646F5] px-6 py-2.5 text-xs font-semibold text-black shadow-[0_0_24px_rgba(107,255,255,0.55)] disabled:opacity-40
disabled:cursor-not-allowed hover:scale-[1.02] active:scale-95 transition-transform"
>
  {mode === "verify"
    ? submitting
    ? "A confirmar..."
    : "Confirmar código e continuar"
  : mode === "login"
    ? submitting
    ? "A entrar..."
    : "Iniciar sessão e continuar"
  : submitting
  ? "A enviar código..."

```

```

        : "Criar conta e enviar código"
    </button>
</form>
</div>
);
}

type GuestCheckoutCardProps = {
  guestName: string;
  guestEmail: string;
  guestEmailConfirm: string;
  guestPhone: string;
  guestErrors: { name?: string; email?: string; phone?: string };
  onChangeName: (v: string) => void;
  onChangeEmail: (v: string) => void;
  onChangeEmailConfirm: (v: string) => void;
  onChangePhone: (v: string) => void;
  onContinue: () => void;
};

function GuestCheckoutCard({
  guestName,
  guestEmail,
  guestEmailConfirm,
  guestPhone,
  guestErrors,
  onChangeName,
  onChangeEmail,
  onChangeEmailConfirm,
  onChangePhone,
  onContinue,
}: GuestCheckoutCardProps) {
  return (
    <div className="flex-1 rounded-2xl border border-white/12 bg-white/[0.05] px-6 py-6 shadow-[0_0_40px_rgba(0,0,0,0.6)] flex flex-col gap-4">
      <div className="flex items-start justify-between gap-4">
        <div>
          <h3 className="text-sm font-semibold mb-1">Continuar como convidado</h3>
          <p className="text-[11px] text-white/60 max-w-sm leading-relaxed">
            Compra em 30 segundos. Guardamos os teus bilhetes pelo email e podes
            criar conta depois para os ligar ao teu perfil.
          </p>
          <div className="mt-2 space-y-1 text-[11px] text-white/55">
            <p>• Email é usado para entregar bilhetes e recibo.</p>
            <p>• Telefone ajuda no contacto no dia do evento (opcional).</p>
          </div>
        </div>
        <span className="text-[20px]">➡</span>
      </div>

      <div className="flex flex-col gap-3 text-[12px]">
        <div className="flex flex-col gap-1">
          <label className="text-white/70">Nome completo</label>
          <input
            type="text"
            className={`w-full rounded-xl bg-black/60 border px-3 py-2 text-[12px] outline-none focus:border-[#6BFFFF]
            focus:ring-1 focus:ring-[#6BFFFF] ${guestErrors.name ? "border-red-400/70" : "border-white/15"
            }`}
            placeholder="Como queres que apareça no bilhete"
            value={guestName}
            onChange={(e) => onChangeName(e.target.value)}
            autoComplete="name"
          />
          {guestErrors.name &&
            <span className="text-[11px] text-red-300">{guestErrors.name}</span>
          }
        </div>
        <div className="flex flex-col gap-1">
          <label className="text-white/70">Email</label>
          <input
            type="email"
            className={`w-full rounded-xl bg-black/60 border px-3 py-2 text-[12px] outline-none focus:border-[#6BFFFF]
            focus:ring-1 focus:ring-[#6BFFFF] ${guestErrors.email ? "border-red-400/70" : "border-white/15"
            }`}
            placeholder="nome@exemplo.com"
            value={guestEmail}
          />
        </div>
      </div>
    </div>
  );
}

```

```

        onChange={(e) => onChangeEmail(e.target.value)}
        autoComplete="email"
      />
      {guestErrors.email && (
        <span className="text-[11px] text-red-300">{guestErrors.email}</span>
      )}
    </div>
    <div className="flex flex-col gap-1">
      <label className="text-white/70">Confirmar email</label>
      <input
        type="email"
        className={`w-full rounded-xl bg-black/60 border px-3 py-2 text-[12px] outline-none focus:border-[#6BFFFF]
focus:ring-1 focus:ring-[#6BFFFF] ${guestErrors.email ? "border-red-400/70" : "border-white/15"}`}
        placeholder="repete o teu email"
        value={guestEmailConfirm}
        onChange={(e) => onChangeEmailConfirm(e.target.value)}
        autoComplete="email"
      />
    </div>
    <div className="flex flex-col gap-1">
      <label className="text-white/70">Telemóvel (opcional)</label>
      <input
        type="tel"
        inputMode="tel"
        className={`w-full rounded-xl bg-black/60 border px-3 py-2 text-[12px] outline-none focus:border-[#6BFFFF]
focus:ring-1 focus:ring-[#6BFFFF] ${guestErrors.phone ? "border-red-400/70" : "border-white/15"}`}
        placeholder="+351 ..."
        value={guestPhone}
        onChange={(e) => {
          const sanitized = sanitizePhone(e.target.value);
          onChangePhone(sanitized);
        }}
        autoComplete="tel"
      />
      {guestErrors.phone && (
        <span className="text-[11px] text-red-300">{guestErrors.phone}</span>
      )}
    </div>
  </div>

  <button
    type="button"
    onClick={onContinue}
    className="mt-1 inline-flex w-full items-center justify-center rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-6 py-2.5 text-xs font-semibold text-black shadow-[0_0_24px_rgba(107,255,255,0.55)] hover:scale-[1.02] active:scale-95 transition-transform"
  >
    Continuar como convidado
  </button>

```

<p className="mt-1 text-[10px] text-white/40 leading-snug">
Vamos enviar os bilhetes para este email. Depois podes criar conta e
migrar todos os bilhetes para o teu perfil.
</p>

</div>

);

app/components/checkout/Step3Sucesso.tsx

```

"use client";

import { useEffect, useState } from "react";
import { useRouter } from "next/navigation";
import { useCheckout } from "./contextoCheckout";
import { formatMoney } from "@lib/money";

const FREE_PLACEHOLDER_INTENT_ID = "FREE_CHECKOUT";

const scenarioCopy: Record<string, string> = {
  GROUP_SPLIT: "Pagaste apenas a tua parte desta dupla.",
  GROUP_FULL: "Pagaste 2 lugares (tu + parceiro).",
}

```

```

RESALE: "Compra de bilhete em revenda.",
FREE_CHECKOUT: "Inscrição gratuita concluída.",
};

function normalizeCheckoutStatus(raw: unknown): "PROCESSING" | "PAID" | "FAILED" {
  const v = typeof raw === "string" ? raw.trim().toUpperCase() : "";
  if (!["PAID", "OK", "SUCCEEDED", "SUCCESS", "COMPLETED", "CONFIRMED"].includes(v)) return "PAID";
  if (!["FAILED", "ERROR", "CANCELED", "CANCELLED", "REQUIRES_PAYMENT_METHOD"].includes(v)) return "FAILED";
  return "PROCESSING";
}

function numberFromUnknown(v: unknown): number | null {
  return typeof v === "number" && Number.isFinite(v) ? v : null;
}

function eurosToCents(v: number): number {
  return Math.max(0, Math.round(v * 100));
}

function centsFromAdditional(additional: Record<string, unknown>, key: string): number | null {
  // Convention used in this checkout:
  // - `*Cents` fields are cents
  // - `total` (without suffix) has historically been stored as euros
  const centsKey = `${key}Cents`;
  const cents = numberFromUnknown(additional[centsKey]);
  if (cents !== null) return cents;

  if (key === "total") {
    const totalEuros = numberFromUnknown(additional.total);
    if (totalEuros !== null) return eurosToCents(totalEuros);
  }

  return null;
}

export default function Step3Sucesso() {
  const { dados, fecharCheckout, breakdown: checkoutBreakdown } = useCheckout();
  const router = useRouter();
  const [statusError, setStatusError] = useState<string | null>(null);

  const additional:
    | Record<string, unknown>
    | undefined =
  dados?.additional && typeof dados.additional === "object"
    ? (dados.additional as Record<string, unknown>)
    : undefined;

  const scenario =
    (dados?.paymentScenario as string | null | undefined) ??
    (additional && typeof additional.paymentScenario === "string"
      ? (additional.paymentScenario as string)
      : null);

  const isFreeScenario = scenario === "FREE_CHECKOUT";

  const paymentIntentId =
    additional && typeof additional.paymentIntentId === "string"
      ? additional.paymentIntentId
      : null;

  const fallbackPurchaseId =
    paymentIntentId && paymentIntentId !== FREE_PLACEHOLDER_INTENT_ID ? paymentIntentId : null;

  const purchaseId =
    additional && typeof additional.purchaseId === "string"
      ? additional.purchaseId
      : fallbackPurchaseId;

  useEffect(() => {
    if (dados && !purchaseId && !isFreeScenario) {
      router.replace("/explorar");
    }
  }, [dados, router, purchaseId, isFreeScenario]);

  // Revalidar bilhetes após sucesso (traz novos bilhetes mais depressa)
  useEffect(() => {
    async function revalidateTickets() {
      try {

```

```

        await fetch("/api/me/tickets", { method: "GET", cache: "no-store" });
    } catch (err) {
        console.warn("[Step3Sucesso] Falha ao revalidar /api/me/tickets", err);
    }
}
revalidateTickets();
}, []);

const guestEmail =
    additional && typeof additional.guestEmail === "string" ? additional.guestEmail : null;

const breakdown = () => {
    const add = additional ?? {};
}

const subtotalFromContext =
    typeof checkoutBreakdown?.subtotalCents === "number" ? checkoutBreakdown.subtotalCents : null;

const subtotalFromLines =
    checkoutBreakdown?.lines?.reduce((sum, line) => sum + Number(line.lineTotalCents ?? 0), 0) ?? null;

// If we lost the context breakdown (refresh), we fallback to additional.
// NOTE: `additional.total` is stored as euros in Step2, so we convert to cents.
const subtotalCentsRaw =
    subtotalFromContext ??
    numberFromUnknown(add.subtotalCents) ??
    numberFromUnknown(add.totalCents) ??
    centsFromAdditional(add, "total") ??
    0;

const subtotalCents =
    subtotalCentsRaw && subtotalCentsRaw > 0
    ? subtotalCentsRaw
    : subtotalFromLines && subtotalFromLines > 0
    ? subtotalFromLines
    : 0;

const discountCents =
    typeof checkoutBreakdown?.discountCents === "number"
    ? checkoutBreakdown.discountCents
    : numberFromUnknown(add.discountCents) ?? 0;

const platformFeeCents =
    typeof checkoutBreakdown?.platformFeeCents === "number"
    ? checkoutBreakdown.platformFeeCents
    : numberFromUnknown(add.platformFeeCents) ?? 0;

const totalCentsFromContext =
    typeof checkoutBreakdown?.totalCents === "number" ? checkoutBreakdown.totalCents : null;

const totalCentsFromAdditional =
    numberFromUnknown(add.totalCents) ?? centsFromAdditional(add, "total");

const computedTotalFallback = Math.max(0, subtotalCents - discountCents + platformFeeCents);

const totalCents = totalCentsFromContext ?? totalCentsFromAdditional ?? computedTotalFallback;

// fallback: if discount did not come but total < subtotal (+fees), infer it
const inferredDiscount =
    discountCents > 0
    ? discountCents
    : subtotalCents > 0 && totalCents >= 0
    ? Math.max(0, subtotalCents + platformFeeCents - totalCents)
    : 0;

const code =
    typeof add.appliedPromoLabel === "string"
    ? add.appliedPromoLabel
    : typeof add.promoCodeRaw === "string"
    ? add.promoCodeRaw
    : typeof add.promoCode === "string"
    ? add.promoCode
    : null;

const currency =
    typeof checkoutBreakdown?.currency === "string"
    ? checkoutBreakdown.currency
    : typeof add.currency === "string"
    ? add.currency
    : null;

```

```

        : "EUR";

    if (
      Number.isNaN(subtotalCents) &&
      Number.isNaN(inferredDiscount) &&
      Number.isNaN(platformFeeCents) &&
      Number.isNaN(totalCents)
    ) {
      return null;
    }

    return {
      subtotalCents,
      discountCents: inferredDiscount,
      platformFeeCents,
      totalCents,
      code,
      currency,
    };
  })();
}

const subtotalEur = breakdown ? breakdown.subtotalCents / 100 : null;
const discountEur = breakdown ? breakdown.discountCents / 100 : null;
const platformFeeEur = breakdown ? breakdown.platformFeeCents / 100 : null;
const totalEur = breakdown ? breakdown.totalCents / 100 : null;

const initialStatus: "PROCESSING" | "PAID" | "FAILED" =
  isFreeScenario ? "PAID" : purchaseId ? "PROCESSING" : "PROCESSING";
const [status, setStatus] = useState<"PROCESSING" | "PAID" | "FAILED">(initialStatus);

useEffect(() => {
  if (isFreeScenario) setStatus("PAID");
}, [isFreeScenario]);

useEffect(() => {
  if (!purchaseId || isFreeScenario) return;

  let cancelled = false;
  let interval: ReturnType<typeof setInterval> | null = null;

  const poll = async () => {
    try {
      const url = new URL("/api/checkout/status", window.location.origin);
      url.searchParams.set("purchaseId", purchaseId);
      if (paymentIntentId && paymentIntentId !== purchaseId) {
        url.searchParams.set("paymentIntentId", paymentIntentId);
      }
      const res = await fetch(url.toString(), { cache: "no-store" });
      const data = await res.json().catch(() => null);
      const mapped = normalizeCheckoutStatus(data?.status);
      if (cancelled) return;

      if (mapped === "PAID") {
        setStatus("PAID");
        setStatusError(null);
        if (interval) clearInterval(interval);
        // revalidate once more after confirmed
        try {
          await fetch("/api/me/tickets", { method: "GET", cache: "no-store" });
        } catch {}
        return;
      }

      if (mapped === "FAILED") {
        setStatus("FAILED");
        setStatusError(typeof data?.error === "string" ? data.error : null);
        if (interval) clearInterval(interval);
        return;
      }

      setStatus("PROCESSING");
      setStatusError(null);
    } catch (err) {
      console.warn("[Step3Sucesso] Poll status falhou", err);
      if (!cancelled) setStatusError(null);
    }
  };
  poll();
}

```

```

interval = setInterval(poll, 3000);

return () => {
  cancelled = true;
  if (interval) clearInterval(interval);
};

}, [purchaseId, isFreeScenario];

if (!dados) {
  return (
    <div className="text-center space-y-4">
      <h2 className="text-xl font-semibold text-black">Algo correu mal 😵</h2>
      <p className="text-sm text-black/70">
        Não encontrámos os dados do bilhete. Fecha esta janela e tenta novamente.
      </p>
      <button
        onClick={fecharCheckout}
        className="mt-4 w-full rounded-xl bg-black text-white py-2 text-sm font-medium hover:bg-black/80"
      >
        Fechar
      </button>
    </div>
  );
}

if (!purchaseId && !isFreeScenario) {
  return null;
}

return (
  <div className="flex flex-col items-center text-center gap-8 py-6 px-4 text-white">

    /* Título */
    <div className="space-y-1">
      <h2 className="text-3xl font-semibold bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] bg-clip-text text-transparent">
        {status === "PAID"
          ? isFreeScenario
            ? "Inscrição confirmada 🎉"
            : "Compra Confirmada 🎉"
          : status === "FAILED"
            ? "Pagamento não confirmado"
            : "A confirmar pagamento..."}
      </h2>
      <p className="text-sm text-white/70">
        {status === "FAILED"
          ? statusError ?? "Não conseguimos confirmar o pagamento. Tenta novamente ou contacta suporte."
          : status === "PAID"
            ? guestEmail
              ? `Obrigado! Enviámos os teus bilhetes para ${guestEmail}.`
              : isFreeScenario
                ? "A tua inscrição gratuita está confirmada."
                : "Compra confirmada. Já podes ver os teus bilhetes."
            : guestEmail
              ? `Estamos a confirmar o pagamento. Assim que estiver confirmado, vais receber os bilhetes em ${guestEmail}.`
              : "Estamos a confirmar o pagamento. Mantém esta página aberta."}
      </p>
    </div>

    /* Card principal estilo Apple Wallet */
    <div className="w-full rounded-3xl bg-white/[0.03] backdrop-blur-xl border border-white/10 px-6 py-8 shadow-[0_0_40px_rgba(0,0,0,0.45)] space-y-6">

      /* Evento */
      <div className="space-y-1">
        <p className="text-[11px] uppercase tracking-widest text-white/50">Evento</p>
        <p className="text-xl font-semibold">
          {dados.ticketName ?? "Bilhete"}
        </p>
        {scenario && scenarioCopy[scenario] && (
          <p className="text-[11px] text-white/70">{scenarioCopy[scenario]}</p>
        )}
      </div>

      /* Breakdown */
      {status === "PAID" && breakdown && (
        <div className="space-y-2 text-sm text-white/80">
          <div className="flex items-center justify-between border-b border-white/10 pb-2">

```

```

<span className="text-white/60 text-[11px] uppercase tracking-widest">Total dos bilhetes</span>
<span className="font-semibold">
{formatMoney(subtotalEur)}
</span>
</div>
{breakdown.discountCents > 0 && (
<div className="flex items-center justify-between">
<span className="text-white/60">Desconto {breakdown.code} ? `(${breakdown.code})` : ""</span>
<span className="text-emerald-300">{formatMoney(discountEur)}</span>
</div>
)}
{breakdown.platformFeeCents > 0 && (
<div className="flex items-center justify-between">
<span className="text-white/60">Taxa da plataforma</span>
<span className="text-orange-200">{formatMoney(platformFeeEur)}</span>
</div>
)}
<div className="flex items-center justify-between border-t border-white/10 pt-2">
<span className="text-white text-[12px] font-semibold uppercase tracking-widest">Total Pago</span>
<span className="text-xl font-semibold">
{formatMoney(totalEur)}
</span>
</div>
</div>
)}

/* Info */
<div className="space-y-1 text-sm text-white/60">
<p>
{status === "PAID"
? guestEmail
? "Guarda o email com os bilhetes. Podes criar conta e ligar estes bilhetes mais tarde."
: "A tua compra foi concluída com sucesso."
: status === "FAILED"
? "O pagamento não ficou confirmado. Se o teu banco debitou, contacta suporte${purchaseId} com o ID de compra: ${purchaseId}."
: "Estamos a confirmar o pagamento. Se demorares mais de alguns minutos, fecha e volta a abrir o checkout."
</p>
</div>

/* Botão ver bilhetes */
{status === "PAID" ? (
<button
onClick={() => (guestEmail ? router.push("/login") : router.push("/me"))}
className="w-full rounded-full bg-white text-black py-3 text-sm font-semibold shadow-[0_0_25px_rgba(255,255,255,0.35)] hover:scale-[1.03] active:scale-95 transition-transform"
>
{guestEmail ? "Criar conta e ligar bilhetes" : "Ver os teus bilhetes"}
</button>
) : status === "FAILED" ? (
<div className="w-full rounded-2xl border border-red-500/40 bg-red-500/10 text-sm text-red-100 py-3 text-center">
Pagamento não confirmado. Verifica o método de pagamento ou tenta novamente.
</div>
) : (
<div className="w-full rounded-full bg-white/10 text-white text-sm font-semibold py-3 text-center">
A confirmar...
</div>
)}
</div>

/* Fechar */
<button
onClick={fecharCheckout}
className="w-full rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold py-2.5 text-sm shadow-[0_0_25px_rgba(107,255,255,0.45)] hover:scale-[1.02] active:scale-95 transition-transform"
>
Fechar
</button>
</div>
);
}

```

app/components/ConfirmDestructiveActionDialog.tsx

```
"use client";
```

```

import { ReactNode, useEffect } from "react";
import { createPortal } from "react-dom";

type Props = {
  open: boolean;
  title: string;
  description?: ReactNode;
  consequences?: string[];
  confirmLabel?: string;
  cancelLabel?: string;
  dangerLevel?: "medium" | "high";
  onConfirm: () => void;
  onClose: () => void;
};

export function ConfirmDestructiveActionDialog({
  open,
  title,
  description,
  consequences,
  confirmLabel = "Confirmar",
  cancelLabel = "Cancelar",
  dangerLevel = "medium",
  onConfirm,
  onClose,
}: Props) {
  useEffect(() => {
    if (!open) return;
    const original = document.body.style.overflow;
    document.body.style.overflow = "hidden";
    return () => {
      document.body.style.overflow = original;
    };
  }, [open]);

  if (!open) return null;

  const tone =
    dangerLevel === "high"
      ? "from-[#7F1D1D] via-[#991B1B] to-[#111827]"
      : "from-[#92400E] via-[#78350F] to-[#0F172A]";

  return createPortal(
    <div
      className="fixed inset-0 z-[120] flex items-center justify-center bg-black/70 backdrop-blur-sm px-4"
      role="dialog"
      aria-modal="true"
      onClick={(e) => {
        if (e.target === e.currentTarget) onClose();
      }}
    >
      <div className="w-full max-w-md rounded-2xl border border-white/12 bg-[#050915]/95 p-5 shadow-[0_28px_80px_rgba(0,0,0,0.75)] text-white">
        <div className={`rounded-xl border border-white/10 bg-gradient-to-br px-4 py-3 ${tone}`}>
          <h3 className="text-lg font-semibold leading-tight">{title}</h3>
          {description && <p className="mt-1 text-sm text-white/80">{description}</p>}
        </div>
        {Array.isArray(consequences) && consequences.length > 0 && (
          <ul className="mt-4 space-y-1 text-[13px] text-white/70">
            {consequences.map((c) => (
              <li key={c} className="flex items-start gap-2">
                <span className="mt-1 h-1.5 w-1.5 rounded-full bg-white/60" />
                <span>{c}</span>
              </li>
            )))
          </ul>
        )}
        <div className="mt-5 flex items-center justify-end gap-2">
          <button
            type="button"
            onClick={onClose}
            className="rounded-full border border-white/15 px-3 py-2 text-white/75 hover:border-white/30 hover:bg-white/5"
          >
            {cancelLabel}
          </button>
        </div>
      </div>
    </div>
  );
}

```

```

<button
  type="button"
  onClick={onConfirm}
  className={`rounded-full px-4 py-2 text-[12px] font-semibold shadow-[0_0_22px_rgba(107,255,255,0.35)] ${dangerLevel === "high"
    ? "bg-gradient-to-r from-[#F87171] via-[#EF4444] to-[#B91C1C] text-white hover:brightness-110"
    : "bg-gradient-to-r from-[#FBBF24] via-[#F59E0B] to-[#D97706] text-black hover:brightness-110"
  }`}
>
  {confirmLabel}
</button>
</div>
</div>
</div>,  

document.body,  

);
}

```

app/components/EventCard.tsx

```

// app/components/EventCard.tsx
import Link from 'next/link';

type EventTicket = { price: number };

type EventForCard = {
  slug: string;
  title: string;
  startDate: string;
  endDate: string;
  timezone: string;
  locationName: string;
  address: string;
  isFree: boolean;
  tickets?: EventTicket[];
  interestedCount: number;
  goingCount: number;
};

type Props = {
  event: EventForCard;
};

function formatPrice(tickets: EventTicket[] | undefined, isFree: boolean) {
  if (isFree) return 'Grátis';

  const list = tickets || []; // <- Evita undefined

  if (!list.length) return 'Preço a definir';

  const min = Math.min(...list.map(t => t.price));
  const max = Math.max(...list.map(t => t.price));

  if (min === 0 && max === 0) return 'Grátis';
  if (min === max) return `${min.toFixed(2)} €`;

  return `${min.toFixed(2)} - ${max.toFixed(2)} €`;
}

function formatDateRange(startDate: string, endDate: string, timezone: string) {
  const start = new Date(startDate);
  const end = new Date(endDate);

  const optsDay: Intl.DateTimeFormatOptions = {
    weekday: 'short',
    day: '2-digit',
    month: 'short',
  };

  const optsTime: Intl.DateTimeFormatOptions = {
    hour: '2-digit',
    minute: '2-digit',
  };

  const dayStr = new Intl.DateTimeFormat('pt-PT', { ...optsDay, timeZone: timezone }).format(start);
  const startTimeStr = new Intl.DateTimeFormat('pt-PT', { ...optsTime, timeZone: timezone }).format(start);

```

```

const endTimeStr = new Intl.DateTimeFormat('pt-PT', { ...optsTime, timeZone: timezone }).format(end);

return `${dayStr} · ${startTimeStr} - ${endTimeStr}`;
}

export default function EventCard({ event }: Props) {
  const priceLabel = formatPrice(event.tickets, event.isFree);
  const dateLabel = formatDateRange(event.startDate, event.endDate, event.timezone);

  return (
    <Link href={`/eventos/${event.slug}`}
      className="group relative overflow-hidden rounded-2xl border border-white/10 bg-white/5 hover:bg-white/[0.08] transition-all duration-300 flex flex-col"
    >
      {/* IMAGE */}
      <div className="relative h-44 w-full overflow-hidden">
        {/* Aqui depois ligamos a cover real (Image do Next) */}
        <div className="absolute inset-0 bg-gradient-to-tr from-[#FF00C8]/40 via-[#6BFFFF]/20 to-transparent" />
        <div className="absolute inset-0 bg-[url('/images/placeholder-event.jpg')] bg-cover bg-center opacity-60 group-hover:scale-105 transition-transform duration-500" />
      </div>

      {/* CONTENT */}
      <div className="flex-1 p-4 flex flex-col justify-between">
        <div>
          <div className="text-xs uppercase tracking-wide text-[#6BFFFF]/80 mb-1">
            {dateLabel}
          </div>
          <h3 className="text-lg font-semibold leading-snug mb-1">
            {event.title}
          </h3>
          <p className="text-sm text-white/60 line-clamp-2">
            {event.locationName} · {event.address}
          </p>
        </div>

        <div className="mt-4 flex items-center justify-between text-sm">
          <span className="inline-flex items-center rounded-full bg-white/10 px-3 py-1 text-xs font-medium text-[#6BFFFF]">
            {priceLabel}
          </span>

          <div className="flex items-center gap-3 text-white/50">
            <span className="inline-flex items-center gap-1">
              <span className="text-xs">❤</span>
              <span>{event.interestedCount}</span>
            </span>
            <span className="inline-flex items-center gap-1">
              <span className="text-xs">想去</span>
              <span>{event.goingCount}</span>
            </span>
          </div>
        </div>
      </div>
    </Link>
  );
}

```

app/components/flows/FlowStepper.tsx

```

"use client";

type StepStatus = "active" | "done" | "locked";

export type FlowStep = {
  key: string;
  title: string;
  subtitle?: string;
  status: StepStatus;
  onSelect?: () => void;
};

type FlowStepperProps = {
  steps: FlowStep[];
  className?: string;
  variant?: "default" | "compact";
};

```

```

};

export function FlowStepper({ steps, className, variant = "default" }: FlowStepperProps) {
  if (variant === "compact") {
    return (
      <div className={`flex gap-2 overflow-x-auto pb-1 ${className ?? ""}`}>
        {steps.map((step, idx) => {
          const isActive = step.status === "active";
          const isDone = step.status === "done";
          const clickable = step.status !== "locked" && Boolean(step.onSelect);
          const statusLabel = isActive ? "Em curso" : isDone ? "Completo" : "Por fazer";
          return (
            <button
              key={step.key}
              type="button"
              onClick={() => step.onSelect?.()}
              disabled={!clickable}
              className={`${flex} min-w-[160px] flex-1 items-center gap-2 rounded-xl border px-3 py-2 text-left text-xs transition`}
            ${{
              isActive
                ? "border-white/50 bg-white/[0.08] shadow-[0_10px_30px_rgba(0,0,0,0.45)]"
                : isDone
                  ? "border-emerald-300/35 bg-emerald-400/10 shadow-[0_8px_24px_rgba(0,0,0,0.35)]"
                  : "border-white/12 bg-black/40"
            }} ${clickable ? "hover:-translate-y-0.5 hover:border-white/30" : "cursor-default opacity-85"}>
            >
            <span
              className={`${flex h-8 w-8 items-center justify-center rounded-full border text-xs font-semibold ${
                isActive
                  ? "border-white bg-white text-black shadow-[0_0_6px_rgba(255,255,255,0.1)]"
                  : isDone
                    ? "border-emerald-300/70 bg-emerald-300/25 text-emerald-50"
                    : "border-white/30 text-white/75"
              }}`}
              aria-hidden
            >
              {isDone ? "✓" : idx + 1}
            </span>
            <div className="space-y-0.5">
              <p className={`${font-semibold ${isActive ? "text-white" : isDone ? "text-emerald-50" : "text-white/80"}`}>
                {step.title}
              </p>
              <p className="text-[11px] text-white/60">{statusLabel}</p>
            </div>
            </button>
          );
        })}
      </div>
    );
  }

  return (
    <div className={`${grid gap-3 sm:grid-cols-2 lg:grid-cols-3 ${className ?? ""}`}>
      {steps.map((step, idx) => {
        const isActive = step.status === "active";
        const isDone = step.status === "done";
        const clickable = step.status !== "locked" && Boolean(step.onSelect);
        return (
          <button
            key={step.key}
            type="button"
            onClick={() => step.onSelect?.()}
            disabled={!clickable}
            className={`${group relative overflow-hidden rounded-2xl border px-4 py-3 text-left transition-all duration-200 ${
              isActive
                ? "border-white/50 bg-white/[0.06] shadow-[0_18px_50px_rgba(0,0,0,0.45)]"
                : isDone
                  ? "border-emerald-300/25 bg-emerald-400/5 shadow-[0_12px_30px_rgba(0,0,0,0.35)]"
                  : "border-white/10 bg-black/30"
            }} ${clickable ? "hover:-translate-y-0.5 hover:border-white/30" : "cursor-default opacity-90"}>
          >
          <div
            className={`${absolute inset-0 opacity-0 transition duration-200 ${
              isActive
                ? "bg-gradient-to-br from-[#FF00C8]/15 via-[#6BFFFF]/10 to-[#1646F5]/20 opacity-100"
                : ""
            }}`}
          />
        );
      })}
    </div>
  );
}

```

```


<div
    className={`${mt-0.5 flex h-10 w-10 items-center justify-center rounded-full border text-sm font-semibold transition-all duration-200 ${isDone ? "border-white bg-white text-black shadow-[0_0_8px_rgba(255,255,255,0.08)]" : "border-emerald-300/70 bg-emerald-300/25 text-emerald-50" : "border-white/30 text-white/70"}`}
    isActive
    ? "border-white bg-white text-black shadow-[0_0_8px_rgba(255,255,255,0.08)]"
    : isDone
    ? "border-emerald-300/70 bg-emerald-300/25 text-emerald-50"
    : "border-white/30 text-white/70"
  >
  {isDone ? "✓" : idx + 1}
</div>
<div className="space-y-1">
  <p
    className={`text-sm font-semibold transition-colors ${isActive ? "text-white" : isDone ? "text-emerald-50" : "text-white/80"}`}
  >
    {step.title}
  </p>
  {step.subtitle && (
    <p className="text-[12px] leading-snug text-white/65">{step.subtitle}</p>
  )}
<div className="flex items-center gap-2 text-[11px] uppercase tracking-[0.16em] text-white/45">
  <span
    className="h-1.5 w-1.5 rounded-full ${
      isActive ? "bg-white shadow-[0_0_5px_rgba(255,255,255,0.08)]" : "bg-white/40"
    }"
  >
    <span>{isActive ? "Em curso" : isDone ? "Completo" : "Bloqueado"}</span>
  </div>
</div>
</div>
</button>
);
);
);
</div>
);
}


```

app/components/flows/FlowStickyFooter.tsx

```

"use client";

type FlowStickyFooterProps = {
  backLabel?: string;
  nextLabel: string;
  helper?: string;
  disabledReason?: string | null;
  loading?: boolean;
  loadingLabel?: string;
  showLoadingHint?: boolean;
  disableBack?: boolean;
  disableNext?: boolean;
  onBack?: () => void;
  onNext?: () => void;
};

export function FlowStickyFooter({
  backLabel = "Voltar",
  nextLabel,
  helper,
  disabledReason,
  loading,
  loadingLabel,
  showLoadingHint,
  disableBack,
  disableNext,
  onBack,
  onNext,
}: FlowStickyFooterProps) {
  const nextIsDisabled = disableNext || Boolean(disabledReason);
  return (
    <div className="sticky bottom-0 left-0 right-0 z-[var(--z-footer)] pt-4">

```

```

<div className="relative overflow-hidden border-t border-white/10 bg-black/30 px-4 py-3 md:px-5 md:py-4 backdrop-blur-xl shadow-[_0_-18px_45px_rgba(0,0,0,0.45)]">
  {loading && showLoadingHint && (
    <div className="absolute left-0 right-0 top-0 h-[3px] overflow-hidden">
      <div className="h-full w-full animate-pulse bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5]" />
    </div>
  )}
  <div className="relative flex flex-col gap-3 sm:flex-row sm:items-center sm:justify-between">
    <div className="text-[12px] text-white/70 leading-snug">
      {helper ? <p>{helper}</p> : <p>Guarda e revê no final. Navega sem perder contexto.</p>}
      {disabledReason && <p className="text-white/55">{disabledReason}</p>}
    </div>
    <div className="flex flex-wrap gap-2">
      <button
        type="button"
        onClick={onBack}
        disabled={disableBack}
        className="rounded-full border border-white/12 bg-white/5 px-4 py-2 text-sm font-semibold text-white/80 transition hover:border-white/22 hover:bg-white/8 disabled:opacity-55"
      >
        {backLabel}
      </button>
      <button
        type="button"
        onClick={onNext}
        disabled={nextIsDisabled || loading}
        className="btn-orya px-6 text-sm font-semibold shadow-none"
        title={disabledReason ?? ""}
      >
        {loading ? loadingLabel || "A processar..." : nextLabel}
      </button>
    </div>
  </div>
</div>
</div>
);
}

```

app/components/forms/InlineDatePicker.tsx

```

"use client";

import { useEffect, useMemo, useState } from "react";
import { createPortal } from "react-dom";

type Props = {
  label: string;
  value: string;
  onChange: (value: string) => void;
  minDateTime?: Date;
  required?: boolean;
};

function formatToLocalInput(date: Date) {
  const pad = (n: number) => n.toString().padStart(2, "0");
  return `${date.getFullYear()}-${pad(date.getMonth()) + 1}-${pad(date.getDate())T${pad(date.getHours())}:${pad(date.getMinutes())}}`;
}

function startOfDay(date: Date) {
  const d = new Date(date);
  d.setHours(0, 0, 0, 0);
  return d;
}

function isSameDay(a: Date, b: Date) {
  return (
    a.getFullYear() === b.getFullYear() &&
    a.getMonth() === b.getMonth() &&
    a.getDate() === b.getDate()
  );
}

function roundUpToQuarter(date: Date) {
  const d = new Date(date);
  const minutes = d.getMinutes();

```

```

const rounded = Math.ceil(minutes / 15) * 15;
d.setMinutes(rounded, 0, 0);
return d;
}

export function InlineDateTimePicker({
  label,
  value,
  onChange,
  minDateTime,
  required,
}: Props) {
  const parsedValue = useMemo(() => {
    if (!value) return null;
    const parsed = new Date(value);
    return Number.isNaN(parsed.getTime()) ? null : parsed;
  }, [value]);

  const minDate = useMemo(() => startOfDay(minDateTime ?? new Date()), [minDateTime]);
  const minDateTimeRounded = useMemo(
    () => (minDateTime ? roundUpToQuarter(minDateTime) : roundUpToQuarter(new Date())),
    [minDateTime],
  );

  const [open, setOpen] = useState(false);
  const [viewMonth, setViewMonth] = useState<Date>(parsedValue ?? new Date());
  const [selectedDate, setSelectedDate] = useState<Date | null>(parsedValue ?? null);
  const [selectedTime, setSelectedTime] = useState<string>(() => {
    if (!parsedValue) return "";
    const pad = (n: number) => n.toString().padStart(2, "0");
    return `${pad(parsedValue.getHours())}:${pad(parsedValue.getMinutes())}`;
  });
  const [mounted, setMounted] = useState(false);

  useEffect(() => {
    setMounted(true);
  }, []);

  /* eslint-disable react-hooks/set-state-in-effect */
  useEffect(() => {
    if (parsedValue) {
      setSelectedDate(parsedValue);
      const pad = (n: number) => n.toString().padStart(2, "0");
      setSelectedTime(`${pad(parsedValue.getHours())}:${pad(parsedValue.getMinutes())}`);
      setViewMonth(parsedValue);
    }
  }, [parsedValue]);
  /* eslint-enable react-hooks/set-state-in-effect */

  const days = useMemo(() => {
    const firstOfMonth = new Date(viewMonth.getFullYear(), viewMonth.getMonth(), 1);
    const startWeekday = firstOfMonth.getDay(); // 0-6
    const daysInMonth = new Date(viewMonth.getFullYear(), viewMonth.getMonth() + 1, 0).getDate();
    const list: { date: Date; disabled: boolean }[] = [];
    for (let i = 0; i < startWeekday; i++) {
      list.push({ date: new Date(NaN), disabled: true });
    }
    for (let d = 1; d <= daysInMonth; d++) {
      const dayDate = new Date(viewMonth.getFullYear(), viewMonth.getMonth(), d);
      const disabled = dayDate < minDate;
      list.push({ date: dayDate, disabled });
    }
    return list;
  }, [viewMonth, minDate]);

  const timeOptions = useMemo(() => {
    const options: { label: string; value: string; disabled: boolean }[] = [];
    const baseDate = selectedDate ?? minDate;
    for (let h = 0; h < 24; h++) {
      for (let m = 0; m < 60; m += 15) {
        const label = `${h.toString().padStart(2, "0")}:${m.toString().padStart(2, "0")}`;
        const optionDate = new Date(baseDate);
        optionDate.setHours(h, m, 0, 0);
        let disabled = false;
        if (minDateTimeRounded && selectedDate && isSameDay(selectedDate, minDateTimeRounded)) {
          disabled = optionDate < minDateTimeRounded;
        } else if (!selectedDate && minDateTimeRounded) {
          disabled = optionDate < minDateTimeRounded;
        }
        options.push({ label, value: label, disabled });
      }
    }
    return options;
  }, [selectedDate, minDateTimeRounded]);
}

```

```

        }
        options.push({ label, value: label, disabled });
    }
}

return options;
}, [selectedDate, minDateTimeRounded, minDate]);

useEffect(() => {
    if (!selectedDate || !selectedTime) return;
    const [h, m] = selectedTime.split(":").map((v) => Number(v));
    const next = new Date(selectedDate);
    next.setHours(h || 0, m || 0, 0, 0);
    if (minDateTimeRounded && next < minDateTimeRounded && isSameDay(next, minDateTimeRounded)) {
        return;
    }
    const nextValue = formatToLocalInput(next);
    if (nextValue === value) return;
    onChange(nextValue);
}, [selectedDate, selectedTime, minDateTimeRounded, onChange, value]);

const monthLabel = viewMonth.toLocaleString("pt-PT", { month: "long", year: "numeric" });

return (
    <div className="space-y-1">
        <label className="block text-sm font-medium mb-1">{label}</label>
        <button
            type="button"
            onClick={() => setOpen(true)}
            className="flex w-full items-center justify-between rounded-md border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]/60"
        >
            <span>{parsedValue ? parsedValue.toLocaleString("pt-PT") : "Escolher data e hora"}</span>
            <span className="text-[11px] text-white/60">▼</span>
        </button>
        {open && mounted && createPortal(
            <div
                className="fixed inset-0 z-[var(--z-modal)] flex items-center justify-center bg-black/70 px-4"
                onClick={() => setOpen(false)}
            >
                <div
                    className="w-full max-w-3xl rounded-2xl border border-white/15 bg-[#040712]/95 p-4 shadow-[0_20px_60px_rgba(0,0,0,0.7)] space-y-4"
                    onClick={(e) => e.stopPropagation()}
                >
                    <div className="flex items-center justify-between text-sm text-white/80">
                        <button
                            type="button"
                            onClick={() => {
                                const prev = new Date(viewMonth);
                                prev.setMonth(prev.getMonth() - 1);
                                if (startOfDay(prev) < minDate) return;
                                setViewMonth(prev);
                            }}
                            className="rounded-full px-2 py-1 text-xs hover:bg-white/10 disabled:opacity-40"
                        >
                            ←
                        </button>
                        <span className="font-semibold capitalize">{monthLabel}</span>
                    <button
                        type="button"
                        onClick={() => {
                            const next = new Date(viewMonth);
                            next.setMonth(next.getMonth() + 1);
                            setViewMonth(next);
                        }}
                        className="rounded-full px-2 py-1 text-xs hover:bg-white/10"
                    >
                        →
                    </button>
                </div>
                <div className="grid grid-cols-7 gap-1 text-[11px] text-white/60">
                    {"D", "S", "T", "O", "S", "S"}.map((d, idx) => (
                        <span key={`${d}-${idx}`} className="text-center py-1">
                            {d}
                        </span>
                    )))
                    {days.map((d, idx) => {
                        if (Number.isNaN(d.date.getTime())) {

```

```

        return <span key={`blank-${idx}`}>;
    }
    const isSelected = selectedDate ? isSameDay(selectedDate, d.date) : false;
    return (
      <button
        key={d.date.toISOString()}
        type="button"
        disabled={d.disabled}
        onClick={() => {
          setSelectedDate(d.date);
        }}
        className={`${`h-9 w-9 rounded-full text-[12px] ${d.disabled
          ? "text-white/25 cursor-not-allowed"
          : isSelected
          ? "bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold shadow-[0_0_14px_rgba(107,255,255,0.6)]"
          : "text-white/80 hover:bg-white/10"
        )}`}
      >
        {d.date.getDate()}
      </button>
    );
  )}
</div>

<div className="space-y-2">
  <p className="text-[11px] text-white/60">Hora (15 em 15 min)</p>
  <div className="grid grid-cols-4 gap-2 max-h-40 overflow-y-auto pr-1">
    {timeOptions.map((opt) => (
      <button
        key={opt.value}
        type="button"
        disabled={opt.disabled}
        onClick={() => setSelectedTime(opt.value)}
        className={`${`rounded-lg px-2 py-1 text-xs ${opt.disabled
          ? "text-white/30 cursor-not-allowed"
          : selectedTime === opt.value
          ? "bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold shadow-[0_0_12px_rgba(107,255,255,0.5)]"
          : "bg-white/5 text-white/80 hover:bg-white/10"
        )}`}
      >
        {opt.label}
      </button>
    )));
  </div>
</div>
<div className="flex justify-end">
  <button
    type="button"
    onClick={() => setOpen(false)}
    className="rounded-full border border-white/20 px-4 py-2 text-[12px] text-white hover:bg-white/10"
  >
    Confirmar
  </button>
</div>
</div>,
document.body,
)
}
{required && !value && <p className="text-xs text-red-400">Obrigatório</p>}
</div>
);
}

```

app/components/MobileBottomNav.tsx

```

"use client";

import { useMemo } from "react";
import { useRouter } from "next/navigation";

type MobileBottomNavProps = {
  pathname: string;

```

```

isSearchOpen: boolean;
onOpenSearch: () => void;
onCloseSearch: () => void;
};

type Item = {
  label: string;
  icon: string;
  path: string;
  active: (path: string) => boolean;
};

export default function MobileBottomNav({
  pathname,
  isSearchOpen,
  onOpenSearch,
  onCloseSearch,
}: MobileBottomNavProps) {
  const router = useRouter();
  const homeActive = pathname === "/" || pathname === "";
  const derivedTab = () => {
    if (isSearchOpen) return "search";
    if (pathname.startsWith("/explorar")) return "explorar";
    if (pathname.startsWith("/buscar")) return "search";
    if (pathname.startsWith("/me/tickets")) return "tickets";
    if (pathname.startsWith("/me/compras")) return "purchases";
    if (pathname.startsWith("/me")) return "profile";
    return "home";
  }();
}

const itemExplorar: Item = useMemo(
() => ({
  label: "Explorar",
  icon: "🔍",
  path: "/explorar",
  active: (p) => p.startsWith("/explorar"),
}),
[],
);

const itemProcurar: Item = useMemo(
() => ({
  label: "Procurar",
  icon: "🔍",
  path: "/buscar",
  active: (_p) => isSearchOpen,
}),
[isSearchOpen],
);

const itemBilhetes: Item = useMemo(
() => ({
  label: "Bilhetes",
  icon: "🎫",
  path: "/me/tickets",
  active: (p) => p.startsWith("/me/tickets"),
}),
[],
);

const itemCompras: Item = useMemo(
() => ({
  label: "Compras",
  icon: "🛍",
  path: "/me/compras",
  active: (p) => p.startsWith("/me/compras"),
}),
[],
);

const itemPerfil: Item = useMemo(
() => ({
  label: "Perfil",
  icon: "👤",
  path: "/me",
  active: (p) => p.startsWith("/me"),
}),
[],
);

```

```

);

const go = (item: Item) => {
  onCloseSearch();
  router.push(item.path);
};

return (
  <nav
    className="fixed bottom-0 left-0 right-0 z-[70] text-white md:hidden"
    style={{ paddingBottom: "calc(14px + env(safe-area-inset-bottom, 14px))" }}
  >
    <div className="mx-auto max-w-3xl px-3">
      <div className="relative h-[86px] flex justify-center">
        {/* Fundo glass + blur */}
        <div className="absolute inset-0 rounded-3xl border border-white/12 bg-black/38 backdrop-blur-[30px] shadow-[0_-32px_80px_rgba(0,0,0,0.78)]"/>
        <div className="absolute inset-0 rounded-3xl bg-gradient-to-r from-[#0a1120]/80 via-[#0b0f1c]/82 to-[#0a1120]/80 opacity-95" />

        {/* Content */}
        <div className="relative z-10 h-full px-3 pb-3">
          <div className="grid h-full grid-cols-[1fr_1fr_76px_1fr_1fr] items-center text-center gap-2">
            <NavItem item={itemExplorar} isActive={derivedTab === "explorar"} onClick={go} />
            <NavItem
              item={itemProcurar}
              isActive={derivedTab === "search"}
              onClick={() => onOpenSearch()}
            />
            <div />
            <NavItem item={itemBilhetes} isActive={derivedTab === "tickets"} onClick={go} />
            <NavItem item={itemCompras} isActive={derivedTab === "purchases"} onClick={go} />
            <NavItem item={itemPerfil} isActive={derivedTab === "profile"} onClick={go} />
          </div>
        
```

```

    >
      <span className="text-[18px] leading-none">{item.icon}</span>
      <span className="leading-none">{item.label}</span>
    </button>
  );
}

```

app/components/Navbar.tsx

```

"use client";

import { useEffect, useRef, useState } from "react";
import { usePathname, useRouter } from "next/navigation";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";
import { useUser } from "@/app/hooks/useUser";
import Link from "next/link";
import MobileBottomNav from "./MobileBottomNav";
import NotificationBell from "./notifications/NotificationBell";
import { featureFlags } from "@/lib/flags";
import { supabaseBrowser } from "@/lib/supabaseBrowser";
import { OryaPortal } from "./OryaPortal";

type Suggestion = {
  id: number;
  type: "EVENT" | "EXPERIENCE";
  slug: string;
  title: string;
  startsAt: string | null;
  locationName: string | null;
  locationCity: string | null;
  coverImageUrl: string | null;
};

export function Navbar() {
  const router = useRouter();
  const rawPathname = usePathname();

  const { openModal: openAuthModal, isOpen: isAuthOpen } = useAuthModal();
  const { user, profile, roles, isLoading } = useUser();

  const [isVisible, setIsVisible] = useState(true);
  const [isAtTop, setIsAtTop] = useState(true);
  const [isSearchOpen, setIsSearchOpen] = useState(false);
  const [searchQuery, setSearchQuery] = useState("");
  const [suggestions, setSuggestions] = useState<Suggestion[]>([]);
  const [isSuggestLoading, setIsSuggestLoading] = useState(false);
  const [isProfileMenuOpen, setIsProfileMenuOpen] = useState(false);
  const [hasMounted, setHasMounted] = useState(false);
  const [hydratedPathname, setHydratedPathname] = useState<string | null>(null);
  const [lastOrganizerUsername, setLastOrganizerUsername] = useState<string | null>(null);
  const profileMenuRef = useRef<HTMLDivElement | null>(null);
  const lastScrollYRef = useRef(0);
  const pathname = hydratedPathname ?? "";
  const useNewNavbar = featureFlags.NEW_NAVBAR();
  const shouldHide = rawPathname?.startsWith("/organizador");

  const LogoNew = () => {
    const [logoState, setLogoState] = useState<"idle" | "hover" | "press">("idle");
    return (
      <button
        type="button"
        onClick={() => router.push("")}
        onMouseEnter={() => setLogoState("hover")}
        onMouseLeave={() => setLogoState("idle")}
        onMouseDown={() => setLogoState("press")}
        onMouseUp={() => setLogoState("hover")}
        className="flex items-center gap-2"
        aria-label="Voltar à homepage ORYA"
      >
        <OryaPortal size={44} state={logoState} variant="full" />
        <span className="hidden text-sm font-semibold tracking-wide text-white sm:inline">ORYA</span>
      </button>
    );
  };

  useEffect(() => {

```

```

    setHasMounted(true);
}, []);

useEffect(() => {
  // Garantir pathname estável só depois de montar para evitar mismatch
  if (typeof window !== "undefined") {
    setHydratedPathname(rawPathname ?? window.location.pathname);
  }
}, [rawPathname]);

useEffect(() => {
  if (typeof document === "undefined") return;
  if (shouldHide) {
    document.body.dataset.navHidden = "true";
  } else {
    delete document.body.dataset.navHidden;
  }
}, [shouldHide]);

useEffect(() => {
  if (typeof window === "undefined") return;

  const handleScroll = () => {
    const currentY = window.scrollY || 0;
    const atTop = currentY < 10;
    setIsAtTop(atTop);

    const prevY = lastScrollYRef.current;

    if (atTop) {
      // No topo: navbar sempre visível
      setIsVisible(true);
    } else {
      // A descer esconde, a subir mostra
      if (currentY > prevY + 12) {
        setIsVisible(false);
      } else if (currentY < prevY - 12) {
        setIsVisible(true);
      }
    }
  }

  lastScrollYRef.current = currentY;
};

// Inicializa logo o estado correto com a posição atual do scroll
handleScroll();

window.addEventListener("scroll", handleScroll, { passive: true });

return () => {
  window.removeEventListener("scroll", handleScroll);
};
}, []);

useEffect(() => {
  if (typeof document === "undefined") return;

  const originalOverflow = document.body.style.overflow;

  if (isSearchOpen) {
    document.body.style.overflow = "hidden";
  } else {
    document.body.style.overflow = originalOverflow;
  }

  return () => {
    document.body.style.overflow = originalOverflow;
  };
}, [isSearchOpen]);

useEffect(() => {
  if (typeof document === "undefined") return;

  const handleClickOutside = (event: MouseEvent) => {
    if (
      profileMenuRef.current &&
      !profileMenuRef.current.contains(event.target as Node)
    ) {

```

```

        setIsProfileMenuOpen(false);
    }
};

const handleKeyDown = (event: KeyboardEvent) => {
    if (event.key === "Escape") {
        setIsSearchOpen(false);
        setIsProfileMenuOpen(false);
    }
};

document.addEventListener("mousedown", handleClickOutside);
document.addEventListener("keydown", handleKeyDown);

return () => {
    document.removeEventListener("mousedown", handleClickOutside);
    document.removeEventListener("keydown", handleKeyDown);
};
}, []);

const inAuthPage =
    pathname === "/login" || pathname === "/signup" || pathname === "/auth/callback";

const handleSubmitSearch = (e: React.FormEvent) => {
    e.preventDefault();
    const query = searchQuery.trim();
    if (!query) return;

    setIsSearchOpen(false);
    router.push(`/explorar?query=${encodeURIComponent(query)}`);
};

const handleQuickSearch = (value: string) => {
    setIsSearchOpen(false);
    router.push(`/explorar?query=${encodeURIComponent(value)}`);
};

const handleLogout = async () => {
    try {
        await supabaseBrowser.auth.signOut();
    } catch (err) {
        console.warn("[navbar] signOut falhou", err);
    } finally {
        setIsProfileMenuOpen(false);
        router.push("/");
        router.refresh();
    }
};

const buildSlug = (item: Pick<Suggestion, "type" | "slug">) =>
    item.type === "EXPERIENCE" ? `/experiencias/${item.slug}` : `/eventos/${item.slug}`;

// Sugestões ao digitar (tipo DICE)
useEffect(() => {
    let active = true;
    const controller = new AbortController();

    async function load() {
        const q = searchQuery.trim();
        if (q.length < 2) {
            setSuggestions([]);
            return;
        }
        try {
            setIsSuggestLoading(true);
            const res = await fetch(`/_api/explorar/list?q=${encodeURIComponent(q)}&limit=6`, {
                cache: "no-store",
                signal: controller.signal,
            });
            if (!res.ok) throw new Error("erro sugestões");
            const data = await res.json();
            if (!active) return;
            const items = Array.isArray(data?.items)
                ? (data.items as Array<{
                    id: number;
                    type: "EVENT" | "EXPERIENCE";
                    slug: string;
                    title: string;
                }>)
                : [];
            setSuggestions(items);
        } catch (err) {
            console.error(err);
        }
    }
    load();
    controller.signal.addEventListener("abort", () => {
        active = false;
    });
});

```

```

        startsAt?: string | null;
        location?: { name?: string | null; city?: string | null };
        coverImageUrl?: string | null;
    }>
);
: [];
setSuggestions(
    items.map((it) => ({
        id: it.id,
        type: it.type,
        slug: it.slug,
        title: it.title,
        startsAt: it.startsAt ?? null,
        locationName: it.location?.name ?? null,
        locationCity: it.location?.city ?? null,
        coverImageUrl: it.coverImageUrl ?? null,
    })),
);
} catch (err) {
    if (err instanceof DOMException && err.name === "AbortError") return;
    if (active) setSuggestions([]);
} finally {
    if (active) setIsSuggestLoading(false);
}
}

const handle = setTimeout(load, 220);
return () => {
    active = false;
    controller.abort();
    clearTimeout(handle);
};
}, [searchQuery]);

// Forçar onboarding: se autenticado e perfil incompleto, abre modal e impede fechar
useEffect(() => {
    if (user && profile && !profile.onboardingDone && !isAuthOpen && !inAuthPage) {
        openAuthModal({
            mode: "onboarding",
            redirectTo: pathname || "/",
        });
    }
}, [user, profile, pathname, openAuthModal, isAuthOpen, inAuthPage]);

const isAuthenticated = !!user;
const isOrganizer = roles?.includes("organizer");
const userLabel =
    profile?.username ||
    profile?.fullName ||
    (typeof user?.email === "string" ? user.email : "");
const userInitial =
    (userLabel || "0").trim().charAt(0).toUpperCase() || "0";

useEffect(() => {
    try {
        const stored = sessionStorage.getItem("orya_last_organizer_username");
        if (stored) setLastOrganizerUsername(stored);
    } catch {
        // ignore storage issues
    }
}, []);

return (
    <>
        <header
            className={`fixed inset-x-0 top-0 z-50 transition-transform duration-300 ease-out ${
                isVisible ? "translate-y-0" : "-translate-y-full"
            } ${shouldHide ? "hidden" : ""}`}
        >
            <div
                className={`flex w-full items-center gap-4 px-4 md:px-6 lg:px-8 transition-all duration-300 ${
                    isAtTop && !isSearchOpen
                    ? "py-4 md:py-5 border-b border-white/5 bg-[#050915]/60 backdrop-blur-2xl"
                    : "py-3.5 md:py-4 border-b border-white/10 bg-[#060a16]/85 backdrop-blur-2xl shadow-[0_14px_50px_rgba(0,0,0,0.65)]"
                }`}
            >
                {/* Logo + link explorar */}
                <div className="flex flex-1 items-center gap-3">

```

```

{useNewNavbar ? (
  <LogoNew />
) : (
  <button
    type="button"
    onClick={() => router.push("")}
    className="flex items-center gap-2"
  >
    <div className="relative flex h-9 w-9 items-center justify-center rounded-2xl bg-gradient-to-br from-[#0f172a] via-[#111827] to-[#0b1224] text-xs font-black tracking-[0.2em] shadow-[0_0_18px_rgba(107,255,255,0.25)]">
      <span className="absolute inset-0 rounded-2xl border border-white/10" />
      <span className="absolute inset-0 rounded-2xl bg-gradient-to-tr from-[#FF00C8]/35 via-[#6BFFFF]/20 to-transparent animate-[spin_9s_linear_infinite]" />
      <span className="relative z-10 bg-gradient-to-r from-[#FF9CF2] to-[#6BFFFF] bg-clip-text text-transparent">
        OY
      </span>
    </div>
    <span className="hidden text-sm font-semibold uppercase tracking-[0.22em] text-zinc-100 sm:inline">
      ORYA
    </span>
  </button>
) {}

<nav className="hidden items-center gap-3 text-xs text-zinc-300 md:flex">
  <button
    type="button"
    onClick={() => router.push("/explorar")}
    className={`rounded-full px-4 py-2 text-sm font-semibold transition-colors ${ pathname?.startsWith("/explorar") ? "bg-white/12 text-white border border-white/25 shadow-[0_0_18px_rgba(255,255,255,0.18)]" : "text-zinc-200 hover:bg-white/5 hover:text-white border border-white/10"}`}
  >`>
    Explorar
  </button>
  <button
    type="button"
    onClick={() => router.push("/organizador")}
    className={`rounded-full px-4 py-2 text-sm font-semibold transition-colors ${ pathname?.startsWith("/organizador") ? "bg-white/12 text-white border border-white/25 shadow-[0_0_18px_rgba(107,255,255,0.25)]" : "text-zinc-200 hover:bg-white/5 hover:text-white border border-white/10"}`}
  >`>
    Organizar
  </button>
</nav>
</div>

/* Barra de pesquisa central */
<div className="hidden md:flex flex-[1.2] justify-center">
  <button
    type="button"
    onClick={() => setIsSearchOpen(true)}
    className="group flex w-full max-w-xl items-center gap-3 rounded-full border border-white/12 bg-white/5 px-4 py-2 text-left text-[13px] text-white/75 hover:border-white/40 hover:bg-white/10 transition-colors shadow-[0_16px_36px_rgba(0,0,0,0.45)]"
  >
    <span className="flex h-5 w-5 items-center justify-center rounded-full border border-white/30 text-[10px] text-white/70">
      >
    </span>
    <span className="flex-1 truncate text-[12px]">
      Procurar por evento, local ou cidade
    </span>
    <span className="hidden rounded-full border border-white/20 px-2.5 py-1 text-[10px] text-white/50 md:inline">
      Pesquisar
    </span>
  </button>
</div>

/* Lado direito: auth/profile */
<div className="flex flex-1 items-center justify-end gap-2 md:gap-3">
  {isLoading ? (
    <div className="flex items-center gap-2 rounded-full border border-white/10 bg-white/5 px-3 py-1.5 text-[11px] text-white/60 animate-pulse">
      <div className="h-7 w-7 rounded-full bg-white/20" />
      <div className="h-3 w-20 rounded-full bg-white/15" />
    </div>
  ) : (
    <div className="flex items-center gap-2 rounded-full border border-white/10 bg-white/5 px-3 py-1.5 text-[11px] text-white/60" >
      <div className="h-7 w-7 rounded-full bg-white/20" />
      <div className="h-3 w-20 rounded-full bg-white/15" />
    </div>
  )}
</div>

```

```

</div>
) : !isAuthenticated || inAuthPage ? (
  <button
    type="button"
    onClick={() => {
      const redirect = pathname && pathname !== "/" ? pathname : "/";
      openAuthModal({ mode: "login", redirectTo: redirect });
    }}
    className="inline-flex items-center justify-center rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFF] to-[#1646F5] px-3.5 py-1.5 text-[11px] font-semibold text-black shadow-[0_0_26px_rgba(107,255,255,0.65)] hover:brightness-110"
  >
    Entrar / Registrar
  </button>
) : (
  <div className="relative flex items-center gap-2" ref={profileMenuRef}>
    {useNewNavbar && <NotificationBell />}
    <Link
      href="/me"
      className="flex items-center gap-2 rounded-full border border-white/15 bg-white/10 px-2.5 py-1 text-[11px] text-white/85 hover:bg-white/15"
      aria-label="Ir para a tua conta"
    >
      <div className="relative h-9 w-9 overflow-hidden rounded-full border border-white/20 bg-gradient-to-br from-[#0f172a] via-[#111827] to-[#0b1224] text-[11px] font-bold text-white shadow-[0_0_22px_rgba(107,255,255,0.55)]">
        <span className="pointer-events-none absolute inset-0 rounded-full border border-white/10" />
        <span className="pointer-events-none absolute inset-0 rounded-full bg-gradient-to-tr from-[#FF00C8]/35 via-[#6BFFF]/25 to-transparent animate-[spin_14s_linear_infinite]" />
        <span className="relative z-10 flex h-full w-full items-center justify-center bg-gradient-to-r from-[#FF9CF2] to-[#6BFFF] bg-clip-text text-transparent">
          {userInitial}
        </span>
      </div>
      <span className="hidden max-w-[120px] truncate text-[11px] sm:inline">
        {userLabel} || Conta ORYA
      </span>
    </Link>
    <button
      type="button"
      onClick={() => setIsProfileMenuOpen((open) => !open)}
      className="rounded-full border border-white/15 bg-white/10 px-2 py-1 text-[11px] text-white/80 hover:bg-white/15"
      aria-haspopup="menu"
      aria-expanded={isProfileMenuOpen}
      aria-label="Abrir menu de conta"
    >
      ▾
    </button>
  </div>
  {isProfileMenuOpen && (
    <div
      className="absolute right-0 top-full mt-2 w-56 origin-top-right rounded-2xl border border-white/14 bg-black/85 p-2 text-[11px] text-white/80 shadow-[0_22px_60px_rgba(0,0,0,0.85)] backdrop-blur-2xl"
      role="menu"
      aria-label="Menu de conta ORYA"
    >
      <Link
        href="/me"
        onClick={() => setIsProfileMenuOpen(false)}
        className="flex w-full items-center justify-between rounded-xl px-2.5 py-1.5 text-left hover:bg-white/8"
      >
        <span>A minha conta</span>
      </Link>
      <Link
        href="/me/tickets"
        onClick={() => setIsProfileMenuOpen(false)}
        className="flex w-full items-center justify-between rounded-xl px-2.5 py-1.5 text-left hover:bg-white/8"
      >
        <span>Os meus bilhetes</span>
      </Link>
      <Link
        href="/me/compras"
        onClick={() => setIsProfileMenuOpen(false)}
        className="flex w-full items-center justify-between rounded-xl px-2.5 py-1.5 text-left hover:bg-white/8"
      >
        <span>Minhas compras</span>
      </Link>
      <Link
        href="/me/settings"
      >
    
```

```

    onClick={() => setIsProfileMenuOpen(false)}
    className="flex w-full items-center justify-between rounded-xl px-2.5 py-1.5 text-left hover:bg-white/8"
  >
  <span>Definições</span>
</Link>
<Link
  href="/me/experiencias"
  onClick={() => setIsProfileMenuOpen(false)}
  className="flex w-full items-center justify-between rounded-xl px-2.5 py-1.5 text-left hover:bg-white/8"
>
  <span>Minhas experiências</span>
</Link>
{lastOrganizerUsername && (
  <Link
    href={`/o/${lastOrganizerUsername}`}
    onClick={() => setIsProfileMenuOpen(false)}
    className="flex w-full items-center justify-between rounded-xl px-2.5 py-1.5 text-left hover:bg-white/8"
  >
    <span>Ver página pública</span>
  </Link>
)}
{pathname?.startsWith("/organizador") && (
  <Link
    href="/me"
    onClick={() => setIsProfileMenuOpen(false)}
    className="flex w-full items-center justify-between rounded-xl px-2.5 py-1.5 text-left hover:bg-white/8"
  >
    <span>Voltar a utilizador</span>
  </Link>
)}
<button
  type="button"
  onClick={() => {
    setIsProfileMenuOpen(false);
    router.push("/organizador");
  }}
  className="mt-1 flex w-full items-center justify-between rounded-xl px-2.5 py-1.5 text-left hover:bg-
white/8"
>
  <span>{isOrganizer ? "Dashboard de organizador" : "Tornar-me organizador"}</span>
  {!isOrganizer && (
    <span className="text-[10px] text-[#FFCC66]">
      Em breve
    </span>
  )}
</button>
<div className="my-1 h-px w-full bg-white/10" />
<button
  type="button"
  onClick={handleLogout}
  className="mt-1 w-full rounded-xl bg-white/10 px-3 py-2 text-left text-red-100 hover:bg-white/15"
>
  Terminar sessão
</button>
</div>
)}
</div>
</div>
</header>
{/* Overlay de pesquisa estilo full-screen, com sugestões */}
{isSearchOpen && (
  <div
    className="fixed inset-0 z-40 bg-black/75 backdrop-blur-2xl"
    role="dialog"
    aria-modal="true"
    onClick={(e) => {
      if (e.target === e.currentTarget) {
        setIsSearchOpen(false);
      }
    }}
  >
    <div className="mx-auto mt-20 max-w-3xl px-4">
      <div
        className="rounded-3xl border border-white/18 bg-[#050915]/90 p-4 shadow-[0_32px_90px_rgba(0,0,0,0.9)]"
        aria-label="Pesquisa de eventos ORYA"
      >

```

```

<form
  onSubmit={handleSubmitSearch}
  className="flex items-center gap-3 rounded-2xl border border-white/20 bg-black/60 px-4 py-2.5"
>
  <span className="flex h-6 w-6 items-center justify-center rounded-full border border-white/30 text-[12px] text-white/80">
    .
  </span>
  <input
    value={searchQuery}
    onChange={(e) => setSearchQuery(e.target.value)}
    placeholder="O que queres fazer hoje?"
    className="flex-1 bg-transparent text-base text-white placeholder:text-white/45 focus:outline-none"
  />
  <button
    type="button"
    onClick={() => setIsSearchOpen(false)}
    className="text-[11px] text-white/60 hover:text-white"
  >
    Fechar
  </button>
</form>

<div className="mt-4 grid gap-4 md:grid-cols-3">
  <div className="md:col-span-3 rounded-2xl border border-white/8 bg-white/5 p-3">
    <div className="flex items-center justify-between text-[11px] text-white/60">
      <span>Resultados</span>
      {isSuggestLoading && <span className="animate-pulse text-white/50">a carregar...</span>}
    </div>
    <div className="mt-2 space-y-2">
      {suggestions.length === 0 && !isSuggestLoading && (
        <p className="text-[11px] text-white/55">
          Começa a escrever para ver eventos, locais e cidades.
        </p>
      )}
      {suggestions.map((item) => (
        <button
          key={`${item.type}-${item.id}`}
          type="button"
          onClick={() => {
            setIsSearchOpen(false);
            router.push(buildSlug(item));
          }}
          className="w-full rounded-xl border border-white/8 bg-black/50 p-2.5 text-left hover:border-white/20
          hover:bg-white/5 transition flex gap-3"
        >
          <div className="h-14 w-14 overflow-hidden rounded-lg bg-gradient-to-br from-[#111827]/70 to-
[#0f172a]/60">
            {item.coverImageUrl ? (
              // eslint-disable-next-line @next/next/no-img-element
              <img
                src={item.coverImageUrl}
                alt={item.title}
                className="h-full w-full object-cover"
              />
            ) : null}
          </div>
          <div className="flex-1">
            <p className="text-[12px] font-semibold text-white line-clamp-1">
              {item.title}
            </p>
            <p className="text-[10px] text-white/65 line-clamp-1">
              {item.locationName || item.locationCity || "Local a anunciar"}
            </p>
            <p className="text-[10px] text-white/55">
              {item.startsAt
                ? new Date(item.startsAt).toLocaleString("pt-PT", {
                  weekday: "short",
                  day: "2-digit",
                  month: "short",
                  hour: "2-digit",
                  minute: "2-digit",
                })
                : "Data a anunciar"}
            </p>
          </div>
          <span className="self-center rounded-full border border-white/15 bg-white/5 px-2 py-0.5 text-[10px]
text-white/70">

```

app/components/notifications/NotificationBell.tsx

```
"use client";

import { useEffect, useMemo, useRef, useState } from "react";
import useSWR from "swr";
import { useUser } from "@/app/hooks/useUser";
import Link from "next/link";
import { formatDistanceToNow } from "date-fns";
import { pt } from "date-fns/locale";

type NotificationDto = {
  id: string;
  type: string;
  title: string;
  body: string;
  ctaUrl?: string | null;
  ctaLabel?: string | null;
  priority?: "LOW" | "NORMAL" | "HIGH";
  readAt?: string | null;
  isRead?: boolean;
  createdAt: string;
};

const fetcher = (url: string) => fetch(url).then((r) => r.json());

const TYPE_LABEL: Record<string, string> = {
  ORGANIZER_INVITE: "Convite de organização",
  STAFF_INVITE: "Convite de staff",
  EVENT_SALE: "Venda",
  STRIPE_STATUS: "Stripe",
  EVENT_RemINDER: "Lembrete",
  FRIEND_REQUEST: "Pedido de amizade",
  FRIEND_ACCEPT: "Amigo aceitou",
  MARKETING_PROMO_ALERT: "Marketing",
  SYSTEM_ANNOUNCE: "Sistema",
};

export function NotificationBell() {
  const { user } = useUser();
  const [open, setOpen] = useState(false);
  const [filter, setFilter] = useState("all" | "sales" | "invites" | "system" | "social">("all");
  const panelRef = useRef<HTMLDivElement | null>(null);
  const queryTypes =
    filter === "sales"
    ? "EVENT_SALE"
    : filter === "invites"
    ? "ORGANIZER_INVITE,STAFF_INVITE"
    : filter === "system"
    ? "STRIPE_STATUS,MARKETING_PROMO_ALERT,SYSTEM_ANNOUNCE"
    : filter === "social"
    ? "FRIEND_REQUEST,FRIEND_ACCEPT"
    : undefined;
}
```

```

const query = user
? `/api/notifications?status=all${queryTypes ? `&types=${encodeURIComponent(queryTypes)}` : ""}`
: null;

const { data, mutate } = useSWR(
  query,
  fetcher,
  { refreshInterval: 30000, revalidateOnFocus: true },
);

const items: NotificationDto[] = useMemo(() => data?.items ?? [], [data]);
const unreadCount = useMemo(
  () => items.filter((n) => n.isRead === false || (!n.isRead && !n.readAt)).length,
  [items],
);

useEffect(() => {
  const onClick = (e: MouseEvent) => {
    if (panelRef.current && !panelRef.current.contains(e.target as Node)) {
      setOpen(false);
    }
  };
  if (open) document.addEventListener("mousedown", onClick);
  return () => document.removeEventListener("mousedown", onClick);
}, [open]);

const markAll = async () => {
  await fetch("/api/notifications/mark-read", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ markAll: true }),
  });
  mutate();
};

const grouped = useMemo(() => {
  const groups: Record<string, NotificationDto[]> = {};
  for (const n of items) {
    const date = new Date(n.createdAt);
    const key = date.toLocaleDateString("pt-PT");
    groups[key] = groups[key] ? [...groups[key], n] : [n];
  }
  return groups;
}, [items]);

return (
  <div className="relative">
    <button
      type="button"
      onClick={() => setOpen((v) => !v)}
      className="relative rounded-full border border-white/15 bg-white/5 p-2 text-white/80 hover:bg-white/10 transition"
      aria-label="Notificações"
    >
      <span>🔔</span>
      {unreadCount > 0 && (
        <span className="absolute -right-1 -top-1 min-w-[18px] rounded-full bg-emerald-500 px-1 text-[11px] font-semibold text-black text-center">
          {unreadCount}
        </span>
      )}
    </button>
    {open && (
      <div
        ref={panelRef}
        className="absolute right-0 mt-2 w-80 rounded-2xl border border-white/10 bg-[#0b0f18]/95 shadow-[0_20px_80px_rgba(0,0,0,.6)] backdrop-blur-xl p-3 text-white/80 z-50"
      >
        <div className="mb-2 flex items-center justify-between text-xs">
          <span className="font-semibold text-white">Notificações</span>
          <button
            type="button"
            className="text-[11px] text-white/60 hover:text-white"
            onClick={markAll}
          >
            Marcar todas como lidas
          </button>
        </div>
    
```

```

<div className="mb-3 flex flex-wrap gap-2 text-[11px]">
  {[ 
    { key: "all", label: "Todas" },
    { key: "sales", label: "Vendas" },
    { key: "invites", label: "Convites" },
    { key: "system", label: "Sistema" },
    { key: "social", label: "Social" },
  ].map((item) => (
    <button
      key={item.key}
      type="button"
      onClick={() => setFilter(item.key as typeof filter)}
      className={`rounded-full border px-2.5 py-1 ${
        filter === item.key
          ? "border-emerald-400/50 bg-emerald-500/15 text-emerald-100"
          : "border-white/15 bg-white/5 text-white/70 hover:border-white/30"
      }`}
    >
      {item.label}
    </button>
  )));
</div>

{items.length === 0 && (
  <div className="rounded-xl border border-dashed border-white/15 bg-white/5 p-3 text-xs text-white/60">
    Sem notificações ainda.
  </div>
)}

{Object.entries(grouped).map(([day, list]) => (
  <div key={day} className="mb-3">
    <p className="text-[11px] uppercase tracking-[0.18em] text-white/50 mb-1">{day}</p>
    <div className="space-y-1.5">
      {list.map((n) => (
        <div
          key={n.id}
          className={`rounded-xl border px-3 py-2 text-xs ${
            n.readAt
              ? "border-white/10 bg-white/3"
              : "border-emerald-400/30 bg-emerald-500/8"
          }`}
        >
          <div className="flex items-center justify-between gap-2">
            <div className="flex items-center gap-2">
              <span className="text-[11px] text-white/60">
                {TYPE_LABEL[n.type] ?? "Atualização"}
              </span>
              {(!n.isRead === false || (!n.isRead && !n.readAt)) && (
                <span className="h-2 w-2 rounded-full bg-emerald-400" />
              )}
            </div>
            <span className="text-[11px] text-white/45">
              {formatDistanceToNow(new Date(n.createdAt), { locale: pt, addSuffix: true })}
            </span>
          </div>
          <p className="mt-1 text-[13px] font-semibold text-white">{n.title}</p>
          <p className="text-white/70">{n.body}</p>
          {n.ctaUrl && n.ctaLabel && (
            <Link
              href={n.ctaUrl}
              className="mt-2 inline-flex text-[11px] text-[#6BFFFF] hover:underline"
            >
              {n.ctaLabel}
            </Link>
          )}
        </div>
      )));
    </div>
  </div>
));
</div>
);
}

```

app/components/OryaLogo.tsx

```
"use client";

import { useEffect, useRef } from "react";

export default function OryaLogo({ small = false }) {
  return (
    <div
      className={`relative flex items-center justify-center ${small ? "w-10 h-10" : "w-40 h-40"}`}
    >
      {/* Canvas do anel */}
      <canvas
        id={small ? "oryaLogoSmall" : "oryaLogoBig"}
        className="absolute inset-0"
      />

      {/* Texto ORYA */}
      <span
        className={`font-bold tracking-tight bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#FF00C8] bg-clip-text text-transparent ${small ? "text-sm" : "text-4xl"}`}
      >
        ORYA
      </span>
    </div>
  );
}
```

app/components/OryaPortal.tsx

```
"use client";

import type { CSSProperties } from "react";

type PortalState = "idle" | "hover" | "press" | "loader" | "empty";
type PortalVariant = "full" | "eyes";

type Props = {
  size?: number;
  state?: PortalState;
  variant?: PortalVariant;
};

export function OryaPortal({ size = 44, state = "idle", variant = "full" }: Props) {
  const headLiftMap: Record<PortalState, number> = {
    idle: 8,
    hover: 12,
    press: 14,
    loader: 10,
    empty: 6,
  };

  const glowMap: Record<PortalState, number> = {
    idle: 1,
    hover: 1.2,
    press: 1.35,
    loader: 1.3,
    empty: 0.9,
  };

  const showOrbs = state === "loader" || state === "hover" || state === "press";
  const eyesOnly = variant === "eyes";

  const styleVars: CSSProperties = {
    "--portal-size": `${size}px`,
    "--head-lift": `${headLiftMap[state]}px`,
    "--glow-scale": glowMap[state],
  } as CSSProperties;

  return (
    <div
      style={styleVars}
    >
```

```

<div className="relative inline-block" style={styleVars} aria-hidden="true">
  <div className="portal-shell">
    <div className={`portal-ring ${state !== "empty" ? "animate-breathe" : ""}` />
    <div className="portal-inner" />

    {!eyesOnly && (
      <div className={`character ${state !== "empty" ? "animate-bob" : ""}`}>
        <div className="head">
          <span className="eye eye-left" />
          <span className="eye eye-right" />
          <span className="hand hand-left" />
          <span className="hand hand-right" />
        </div>
      </div>
    )}

    {eyesOnly && (
      <div className="eyes-only">
        <span className="eye eye-left small" />
        <span className="eye eye-right small" />
      </div>
    )}

    {showOrbs && (
      <>
        <span className="orb orb-a" />
        <span className="orb orb-b" />
      </>
    )}
  </div>

  <style jsx>{
    .portal-shell {
      position: relative;
      width: var(--portal-size);
      height: var(--portal-size);
    }

    .portal-ring {
      position: absolute;
      inset: 0;
      border-radius: 999px;
      background: conic-gradient(
        from 90deg,
        #ff00c8,
        #7300ff,
        #6bffff,
        #1646f5,
        #ff00c8
      );
      box-shadow: 0 0 24px rgba(107, 255, 255, 0.35);
      -webkit-mask: radial-gradient(circle at center, transparent 60%, black 60%);
      mask: radial-gradient(circle at center, transparent 60%, black 60%);
      transform: scale(var(--glow-scale));
      transition: transform 240ms ease, box-shadow 240ms ease;
    }

    .portal-inner {
      position: absolute;
      inset: 4px;
      border-radius: 999px;
      background: radial-gradient(circle at 30% 30%, rgba(40, 60, 90, 0.4), rgba(6, 10, 18, 0.95));
      box-shadow: inset 0 0 18px rgba(0, 0, 0, 0.6);
    }

    .character {
      position: absolute;
      left: 50%;
      bottom: 8px;
      transform: translate(-50%, calc(-1 * var(--head-lift)));
      width: 62%;
      height: 62%;
    }

    .head {
      position: absolute;
      inset: 0;
      border-radius: 999px;
    }
  }
}

```

```

background: radial-gradient(circle at 30% 30%, rgba(255, 255, 255, 0.08), rgba(12, 18, 32, 0.9));
box-shadow: inset 0 -6px 12px rgba(0, 0, 0, 0.35);
}

.eye {
  position: absolute;
  top: 44%;
  width: 16%;
  height: 16%;
  border-radius: 999px;
  background: linear-gradient(120deg, #e5f9ff, #9ee7ff);
  box-shadow: 0 0 6px rgba(107, 255, 255, 0.5);
  animation: blink 7s ease-in-out infinite;
}
.eye-left {
  left: 33%;
  animation-delay: 0.6s;
}
.eye-right {
  right: 33%;
  animation-delay: 1.1s;
}
.eye.small {
  top: 47%;
  width: 14%;
  height: 14%;
}

.hand {
  position: absolute;
  bottom: 10%;
  width: 18%;
  height: 10%;
  border-radius: 999px;
  background: linear-gradient(90deg, rgba(255, 255, 255, 0.08), rgba(255, 255, 255, 0.02));
  opacity: 0.6;
}
.hand-left {
  left: 22%;
  transform: rotate(-8deg);
}
.hand-right {
  right: 22%;
  transform: rotate(8deg);
}

.eyes-only {
  position: absolute;
  inset: 0;
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 16%;
}

.orb {
  position: absolute;
  width: 16%;
  height: 16%;
  border-radius: 999px;
  background: radial-gradient(circle, rgba(255, 255, 255, 0.95), rgba(255, 255, 255, 0));
  box-shadow: 0 0 12px rgba(255, 255, 255, 0.45);
  top: 50%;
  left: 50%;
  transform-origin: -220% center;
  animation: orbitA 8s linear infinite;
  opacity: 0.8;
}
.orb-b {
  width: 12%;
  height: 12%;
  transform-origin: 200% center;
  animation: orbitB 10s linear infinite;
  background: radial-gradient(circle, rgba(107, 255, 255, 0.9), rgba(107, 255, 255, 0));
}

@keyframes orbitA {
  from {

```

```

        transform: rotate(0deg) translateX(-50%) translateY(-50%);
    }
    to {
        transform: rotate(360deg) translateX(-50%) translateY(-50%);
    }
}
@keyframes orbitB {
    from {
        transform: rotate(360deg) translateX(-50%) translateY(-50%);
    }
    to {
        transform: rotate(0deg) translateX(-50%) translateY(-50%);
    }
}

@keyframes breathe {
    0%,
    100% {
        box-shadow: 0 0 18px rgba(107, 255, 255, 0.28);
    }
    50% {
        box-shadow: 0 0 28px rgba(107, 255, 255, 0.46);
    }
}
.animate-breathe {
    animation: breathe 8s ease-in-out infinite;
}

@keyframes bob {
    0%,
    100% {
        transform: translate(-50%, calc(-1 * var(--head-lift)));
    }
    50% {
        transform: translate(-50%, calc(-1 * var(--head-lift) - 2px));
    }
}
.animate-bob {
    animation: bob 6s ease-in-out infinite;
}

@keyframes blink {
    0%,
    92%,
    100% {
        transform: scaleY(1);
    }
    94% {
        transform: scaleY(0.25);
    }
    96% {
        transform: scaleY(1);
    }
}

```

@media (prefers-reduced-motion: reduce) {

- .animate-breathe,**
- .animate-bob,**
- .orb {**
- animation: none;**
- }**
- .orb {**
- display: none;**
- }**
- }**

}

</style>

</div>

};

}

app/components/profile/ProfileHeader.tsx

```

"use client";

import Link from "next/link";

```

```

export type ProfileHeaderProps = {
  /** Se é o próprio utilizador a ver o seu perfil */
  isOwner: boolean;
  /** Nome completo do utilizador (ex: "Nuno Lopes") */
  name?: string | null;
  /** Username público (ex: "nuno") */
  username?: string | null;
  /** URL do avatar (pode ser null) */
  avatarUrl?: string | null;
  /** Pequena descrição ou bio (opcional, para o futuro) */
  bio?: string | null;
  /** Data de criação da conta em ISO (opcional) */
  createdAt?: string | null;
};

function formatJoinedDate(createdAt?: string | null): string | null {
  if (!createdAt) return null;
  const d = new Date(createdAt);
  if (Number.isNaN(d.getTime())) return null;

  return new Intl.DateTimeFormat("pt-PT", {
    month: "long",
    year: "numeric",
  }).format(d);
}

export default function ProfileHeader({
  isOwner,
  name,
  username,
  avatarUrl,
  bio,
  createdAt,
}: ProfileHeaderProps) {
  const displayName = name?.trim() || "Utilizador ORYA";
  const handle = username?.trim() || undefined;
  const joinedLabel = formatJoinedDate(createdAt);

  const ownerHasPublicProfile = Boolean(handle);

  const safeAvatarUrl = avatarUrl && avatarUrl.trim().length > 0
    ? avatarUrl
    : undefined;

  const publicProfileHref = ownerHasPublicProfile ? `/${handle}` : null;

  return (
    <section className="w-full rounded-3xl border border-white/10 bg-gradient-to-b from-white/5 via-black/80 to-black/95 px-4 py-5 sm:px-6 sm:py-6 shadow-[#_28px_80px_rgba(0,0,0,0.9)]">
      <div className="flex flex-col gap-4 sm:flex-row sm:items-center sm:justify-between">
        {/* Esquerda: avatar + info principal */}
        <div className="flex items-start gap-4 sm:gap-5">
          {/* Avatar com aro em gradiente ORYA */}
          <div className="shrink-0">
            <div className="relative inline-flex h-20 w-20 sm:h-24 sm:w-24 items-center justify-center rounded-full bg-gradient-to-tr from-[#FF00C8] via-[#0BFFFF] to-[#1646F5] p-[2px] shadow-[#0_0_40px_rgba(255,0,200,0.45)]">
              <div className="flex h-full w-full items-center justify-center rounded-full bg-black/90 overflow-hidden">
                {safeAvatarUrl ? (
                  // eslint-disable-next-line @next/next/no-img-element
                  <img
                    src={safeAvatarUrl}
                    alt={displayName}
                    className="h-full w-full object-cover"
                  />
                ) : (
                  <span className="text-xs font-semibold uppercase tracking-[0.2em] text-white/60">
                    {displayName
                      .split(" ")
                      .map((part) => part[0])
                      .join("")
                      .slice(0, 3)}
                  </span>
                )}
              </div>
            </div>
          </div>
        </div>
      </div>
    </section>
    {/* Nome + username + badges */}
  )
}

```

```

<div className="flex flex-col gap-2 min-w-0">
  <div>
    <h1 className="text-xl sm:text-2xl font-semibold tracking-tight text-white truncate">
      {displayName}
    </h1>
    <div className="mt-0.5 flex flex-wrap items-center gap-2 text-[11px] text-white/60">
      {handle && (
        <span className="rounded-full bg-white/5 px-2 py-0.5 font-medium text-white/75">
          @{handle}
        </span>
      )}
      {!handle && isOwner && (
        <span className="rounded-full border border-dashed border-white/25 px-2 py-0.5 text-[10px] text-white/70">
          Define um @username para ativares o teu perfil público
        </span>
      )}
      {joinedLabel && (
        <span className="inline-flex items-center gap-1 rounded-full bg-white/3 px-2 py-0.5 text-[10px] text-
white/65">
          <span className="inline-block h-1.5 w-1.5 rounded-full bg-emerald-400" />
          Na ORYA desde {joinedLabel}
        </span>
      )}
    </div>
  </div>

  {/* Badges de perfil (placeholders por agora) */}
  <div className="mt-1" />

  {/* Bio / descrição curta */}
  {bio && (
    <p className="mt-1 max-w-xl text-xs text-white/75 leading-relaxed">
      {bio}
    </p>
  )}
</div>
</div>

  {/* Direita: botões de ação (diferentes para owner vs público) */}
  <div className="flex flex-wrap items-center justify-start gap-2 sm:justify-end text-[11px]">
    {isOwner ? (
      <>
        {publicProfileHref && (
          <Link
            href={publicProfileHref}
            className="inline-flex items-center gap-1 rounded-full border border-white/25 bg-black/40 px-3 py-1.5 text-
white/80 hover:bg-white/10 transition-colors"
          >
            Ver perfil público
          </Link>
        )}
        <Link
          href="/me/edit"
          className="inline-flex items-center gap-1 rounded-full bg-white text-black px-3 py-1.5 font-semibold shadow-
[0_0_22px_rgba(255,255,255,0.35)] hover:scale-[1.02] active:scale-95 transition-transform"
        >
          Editar perfil
        </Link>
      {(!ownerHasPublicProfile && (
        <Link
          href="/me/edit"
          className="inline-flex items-center gap-1 rounded-full border border-dashed border-white/25 bg-black/30 px-3
py-1.5 text-white/75 hover:bg-white/5 transition-colors"
          >
            Ativar perfil público
          </Link>
      )) : (
        <Link
          href="/me/settings"
          className="inline-flex items-center gap-1 rounded-full border border-white/10 px-3 py-1.5 text-white/70
hover:bg-white/10 transition-colors"
          >
            Definições
          </Link>
        </>
      ) : (
        <>
      )}
    </>
  </div>

```

```

        <>
        <button
          type="button"
          className="inline-flex items-center gap-1 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-3 py-1.5 font-semibold text-black shadow-[0_0_26px_rgba(107,255,255,0.7)] hover:scale-[1.02] active:scale-95 transition-transform"
        >
          Seguir
        </button>

        <button
          type="button"
          className="inline-flex items-center gap-1 rounded-full border border-white/25 bg-black/40 px-3 py-1.5 text-white/85 hover:bg-white/10 transition-colors"
        >
          Mensagem
        </button>

        <button
          type="button"
          className="inline-flex items-center gap-1 rounded-full border border-white/20 bg-black/30 px-3 py-1.5 text-white/80 hover:bg-white/10 transition-colors"
        >
          Partilhar perfil
        </button>
      </>
    )}
  </div>
</div>
</section>
);
}
}

```

app/components/profile/ProfileStatsBar.tsx

```

import React from "react";
import clsx from "clsx";

type ProfileStatsBarProps = {
  /**
   * Indica se é o próprio utilizador a ver o perfil.
   * Se true → mostra "Total investido".
   * Se false → mostra "Desde ..." (se existir).
   */
  isOwner?: boolean;

  /**
   * Número total de eventos com bilhete ligados a esta conta.
   */
  totalTickets: number;

  /**
   * Número de eventos futuros (data >= hoje).
   */
  upcomingTickets: number;

  /**
   * Número de eventos passados (data < hoje).
   */
  pastTickets: number;

  /**
   * Total estimado gasto em bilhetes ORYA (em euros, por agora).
   * Apenas mostrado quando isOwner = true.
   */
  totalSpentEuros?: number | null;

  /**
   * Texto amigável para "membro desde" (ex: "2024", "Jan 2025").
   * Usado em modo público (isOwner = false).
   */
  memberSinceLabel?: string | null;
}

```

```

    * Classe extra opcional para ajustar margem/padding onde for usado.
    */
    className?: string;
};

function formatPlural(count: number, singular: string, plural: string) {
  if (!Number.isFinite(count)) return `0 ${plural}`;
  if (count === 1) return `1 ${singular}`;
  if (count < 0) return `0 ${plural}`;
  return `${count} ${plural}`;
}

export function ProfileStatsBar({
  isOwner = false,
  totalTickets,
  upcomingTickets,
  pastTickets,
  totalSpentEuros,
  memberSinceLabel,
  className,
}: ProfileStatsBarProps) {
  const hasSpentInfo =
    isOwner && totalSpentEuros != null && Number.isFinite(totalSpentEuros);

  const memberSinceText =
    !isOwner && memberSinceLabel
    ? `Na ORYA desde ${memberSinceLabel}`
    : null;

  return (
    <section
      aria-label="Estatísticas de eventos"
      className={clsx(
        "w-full rounded-2xl border border-white/10 bg-black/60 px-4 py-3 md:px-6 md:py-4 backdrop-blur-xl shadow-[0_18px_60px_rgba(0,0,0,0.8)]",
        className,
      )}
    >
    <div className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between">
      /* Título + subtítulo */
      <div className="space-y-1">
        <p className="text-[11px] uppercase tracking-[0.18em] text-white/55">
          Resumo da tua vida ORYA
        </p>
        <p className="text-sm text-white/80">
          Uma visão rápida dos eventos onde já estiveste e dos próximos que
          vêm aí.
        </p>
      </div>
    </div>

    /* Cards de estatísticas */
    <div className="grid grid-cols-2 gap-2 text-[11px] md:flex md:flex-row md:items-stretch md:gap-3">
      /* Total de eventos com bilhete */
      <div className="flex flex-1 flex-col justify-between rounded-xl border border-white/15 bg-white/5 px-3 py-2.5">
        <p className="text-[10px] uppercase tracking-[0.16em] text-white/55">
          Eventos com bilhete
        </p>
        <p className="mt-1 text-lg font-semibold text-white">
          {totalTickets}
        </p>
        <p className="mt-0.5 text-[10px] text-white/55">
          {formatPlural(totalTickets, "evento", "eventos")}
        </p>
      </div>

      /* Próximos eventos */
      <div className="flex flex-1 flex-col justify-between rounded-xl border border-emerald-400/40 bg-emerald-500/10 px-3
py-2.5">
        <p className="text-[10px] uppercase tracking-[0.16em] text-emerald-100/80">
          Próximos
        </p>
        <p className="mt-1 text-lg font-semibold text-emerald-100">
          {upcomingTickets}
        </p>
        <p className="mt-0.5 text-[10px] text-emerald-100/70">
          {upcomingTickets === 0
            ? "Ainda não tens nada marcado"
            : formatPlural(upcomingTickets, "evento marcado", "eventos marcados")}
        </p>
      </div>
    </div>
  
```

```

        </p>
    </div>

    /* Eventos já vividos */
    <div className="flex flex-1 flex-col justify-between rounded-xl border border-white/15 bg-white/3 px-3 py-2.5">
        <p className="text-[10px] uppercase tracking-[0.16em] text-white">
            Já vividos
        </p>
        <p className="mt-1 text-lg font-semibold text-white">
            {pastTickets}
        </p>
        <p className="mt-0.5 text-[10px] text-white/55">
            {pastTickets === 0
                ? "Tudo ainda por acontecer"
                : formatPlural(pastTickets, "evento passado", "eventos passados")}
        </p>
    </div>

    /* Total investido (privado) OU "Na ORYA desde" (público) */
    <div className="flex flex-1 flex-col justify-between rounded-xl border border-white/15 bg-gradient-to-r from-[#FF00C8]/10 via-[#BFFF00]/10 to-[#1646F5]/15 px-3 py-2.5">
        {isOwner ? (
            <>
                <p className="text-[10px] uppercase tracking-[0.16em] text-white/65">
                    Total investido
                </p>
                <p className="mt-1 text-lg font-semibold text-white">
                    {hasSpentInfo
                        ? `${totalSpentEuros!.toFixed(2)} €`
                        : "-"}
                </p>
                <p className="mt-0.5 text-[10px] text-white/60">
                    {hasSpentInfo
                        ? "Valor aproximado em bilhetes ORYA."
                        : "Assim que começares a comprar bilhetes, este valor aparece aqui."}
                </p>
            </>
        ) : (
            <>
                <p className="text-[10px] uppercase tracking-[0.16em] text-white/65">
                    Perfil público
                </p>
                <p className="mt-1 text-lg font-semibold text-white">
                    {memberSinceLabel ?? "ORYA Explorer"}
                </p>
                <p className="mt-0.5 text-[10px] text-white/60">
                    {memberSinceText ??
                        "Explorador ORYA – presença em eventos reais."}
                </p>
            </>
        )}
    </div>
    </div>
</section>
);
}

export default ProfileStatsBar;

```

app/components/profile/ProfileTicketsStrip.tsx

```

import Link from "next/link";

// Tipo de bilhete simplificado para uso no perfil
// Mantém compatibilidade com a estrutura vinda de /me e /me/tickets
export type ProfileTicketItem = {
    id: string;
    quantity?: number | null;
    createdAt?: string | null;
    event?: {
        slug?: string | null;
        title?: string | null;
        startDate?: string | null;
        locationName?: string | null;
        coverImageUrl?: string | null;
    }
}

```

```

} | null;
ticket?: {
  name?: string | null;
} | null;
};

export type ProfileTicketsStripProps = {
  tickets: ProfileTicketItem[];
  /**
   * Se for o próprio dono do perfil (vista privada em /me)
   * → mostra copy "Os teus próximos eventos" + CTA "Ver todos os bilhetes".
   * Caso contrário (perfil público /u/[username])
   * → copy diferente e sem botão de abrir bilhete.
   */
  isOwner?: boolean;
  username?: string | null;
  className?: string;
};

function formatDate(dateStr?: string | null) {
  if (!dateStr) return "";
  const d = new Date(dateStr);
  if (Number.isNaN(d.getTime())) return "";
  return new Intl.DateTimeFormat("pt-PT", {
    day: "2-digit",
    month: "short",
    hour: "2-digit",
    minute: "2-digit",
  }).format(d);
}

function buildStatusLabel(startDate?: string | null) {
  if (!startDate) {
    return { label: "Confirmado", className: "border-emerald-400/50 bg-emerald-500/10 text-emerald-200" };
  }

  const now = new Date();
  const d = new Date(startDate);
  if (Number.isNaN(d.getTime())) {
    return { label: "Confirmado", className: "border-emerald-400/50 bg-emerald-500/10 text-emerald-200" };
  }

  const isPast = d.getTime() < now.getTime();
  const isSameDay =
    d.getFullYear() === now.getFullYear() &&
    d.getMonth() === now.getMonth() &&
    d.getDate() === now.getDate();

  if (isPast) {
    return {
      label: "Já aconteceu",
      className: "border-white/25 bg-white/5 text-white/80",
    };
  }

  if (isSameDay) {
    return {
      label: "É hoje",
      className: "border-sky-400/60 bg-sky-500/10 text-sky-100",
    };
  }

  return {
    label: "Em breve",
    className: "border-[#6BFFFF]/60 bg-[#6BFFFF]/10 text-[#6BFFFF]",
  };
}

export default function ProfileTicketsStrip({
  tickets,
  isOwner = false,
  username,
  className,
}: ProfileTicketsStripProps) {
  // Filtrar e ordenar: focar nos próximos eventos
  const now = new Date();
  const upcoming = [...tickets]
    .filter((t) => {

```

```

    const dateStr = t.event?.startDate ?? t.createdAt ?? null;
    if (!dateStr) return false;
    const d = new Date(dateStr);
    if (Number.isNaN(d.getTime())) return false;
    // Consideramos "próximo" tudo o que ainda não aconteceu
    return d.getTime() >= now.getTime();
})
.sort((a, b) => {
    const da = a.event?.startDate ?? a.createdAt ?? "";
    const db = b.event?.startDate ?? b.createdAt ?? "";
    const daNum = new Date(da).getTime();
    const dbNum = new Date(db).getTime();
    return daNum - dbNum;
})
.slice(0, 4); // só 3-4 itens

if (upcoming.length === 0) {
    // Se não há eventos próximos, podemos simplesmente não mostrar a secção
    return null;
}

const title = isOwner
? "Os teus próximos eventos"
: username
? `Eventos onde ${username} vai estar`
: "Eventos onde este utilizador vai estar";

const subtitle = isOwner
? "Um resumo rápido dos próximos eventos para os quais já tens bilhete."
: "Uma amostra dos eventos públicos onde esta pessoa planeia estar presente.';

return (
<section
    aria-label={title}
    className={[
        "mt-6 space-y-4 rounded-2xl border border-white/10 bg-black/60 px-4 py-4 md:px-5 md:py-5 backdrop-blur-xl shadow-[0_18px_60px_rgba(0,0,0,0.8)]",
        className ?? ""
    ]}
    .filter(Boolean)
    .join(" ")
>
<div className="flex items-center justify-between gap-3">
    <div className="space-y-1">
        <h2 className="text-sm md:text-base font-semibold text-white/90">
            {title}
        </h2>
        <p className="text-[11px] md:text-xs text-white/60 max-w-md">
            {subtitle}
        </p>
    </div>
    {isOwner ? (
        <Link
            href="/me/tickets"
            className="hidden sm:inline-flex items-center gap-1 rounded-full border border-white/20 bg-white/5 px-3 py-1.5 text-[11px] font-medium text-white/80 hover:bg-white/10 hover:text-white transition-colors"
        >
            Ver todos os bilhetes
            <span aria-hidden>&lt;/span>
        </Link>
    ) : (
        <Link
            href="/explorar"
            className="hidden sm:inline-flex items-center gap-1 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-[11px] font-medium text-white/80 hover:bg-white/10 hover:text-white transition-colors"
        >
            Ver eventos na ORYA
            <span aria-hidden>&lt;/span>
        </Link>
    )}
</div>

<div className="mt-3 flex gap-3 overflow-x-auto pb-2 md:grid md:grid-cols-3 md:gap-4 md:overflow-visible">
    {upcoming.map((t) => {
        const event = t.event ?? {};
        const ticket = t.ticket ?? {};
        const dateLabel = formatDate(event.startDate ?? t.createdAt ?? undefined);
    })
}

```

```

const { label: statusLabel, className: statusClass } = buildStatusLabel(
  event.startDate ?? t.createdAt ?? undefined,
);

const quantity = t.quantity ?? 1;

return (
  <article
    key={t.id}
    className="group relative flex min-w-[230px] flex-col overflow-hidden rounded-2xl border border-white/12 bg-gradient-to-b from-white/5 via-black/80 to-black/95 shadow-[0_14px_45px_rgba(0,0,0,0.85)] hover:border-[#6BFFFF]/70 transition-colors"
  >
  <div className="relative aspect-[4/3] w-full overflow-hidden">
    {event.coverImageUrl ? (
      // eslint-disable-next-line @next/next/no-img-element
      <img
        src={event.coverImageUrl}
        alt={event.title ?? "Evento ORYA"}
        className="h-full w-full object-cover transition-transform duration-500 group-hover:scale-[1.03]"
      />
    ) : (
      <div className="flex h-full w-full items-center justify-center bg-gradient-to-br from-[#1b1b2f] via-black to-[#141421] text-[11px] text-white/40">
        Evento ORYA
      </div>
    )}
  </div>

  {/* Overlay + badges */}
  <div className="pointer-events-none absolute inset-0 bg-gradient-to-t from-black/90 via-black/40 to-transparent" />

  <div className="absolute inset-x-2 top-2 flex items-center justify-between gap-2">
    <span
      className={`inline-flex items-center rounded-full border px-2 py-0.5 text-[10px] font-medium ${statusClass}`}
    >
      {statusLabel}
    </span>

    <span className="inline-flex items-center rounded-full bg-black/70 px-2 py-0.5 text-[10px] font-semibold text-white/85">
      {quantity > 1 ? `${quantity} bilhetes` : "1 bilhete"}
    </span>
  </div>

  <div className="absolute inset-x-2 bottom-2 space-y-1">
    <p className="text-[10px] uppercase tracking-[0.16em] text-white/60">
      Bilhete ORYA
    </p>
    <h3 className="text-sm font-semibold leading-snug line-clamp-2">
      {event.title ?? "Evento ORYA"}
    </h3>
    {event.locationName && (
      <p className="text-[11px] text-white/70 line-clamp-1">
        {event.locationName}
      </p>
    )}
    {dateLabel && (
      <p className="text-[11px] text-white/80">{dateLabel}</p>
    )}
  </div>
</div>

<div className="border-t border-white/10 bg-black/85 px-3 py-3 space-y-2">
  <div className="flex items-center justify-between gap-2 text-[11px]">
    <div className="space-y-1">
      <p className="text-white/50">Tipo de bilhete</p>
      <p className="text-white/90 line-clamp-1">
        {ticket.name ?? "Geral / Wave"}
      </p>
    </div>
    {isOwner && event.slug && (
      <a href={`/bilhete/${t.id}`}
        className="inline-flex items-center gap-1 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-3 py-1 font-semibold text-[11px] text-black shadow-[0_0_20px_rgba(107,255,255,0.7)] hover:scale-[1.03] active:scale-95 transition-transform"
      >
    )}
  </div>
</div>

```

```

        >
          Abrir bilhete
        </Link>
      )}
</div>

{isOwner && event.slug && (
  <button
    type="button"
    className="inline-flex items-center gap-1 text-[10px] text-white/45 hover:text-white/70 transition-colors"
    // No futuro podes ligar a uma ação de partilha específica
  >
    <span aria-hidden>🔗</span> Partilhar com amigos (em breve)
  </button>
)}
</div>
</article>
);
)}
</div>
</div>

/* CTA mobile */
<div className="mt-3 flex justify-end sm:hidden">
  {isOwner ? (
    <Link
      href="/me/tickets"
      className="inline-flex items-center gap-1 rounded-full border border-white/20 bg-white/5 px-3 py-1.5 text-[11px]
font-medium text-white/80 hover:bg-white/10 hover:text-white transition-colors"
    >
      Ver todos os bilhetes
      <span aria-hidden>→</span>
    </Link>
  ) : (
    <Link
      href="/explorar"
      className="inline-flex items-center gap-1 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-[11px]
font-medium text-white/80 hover:bg-white/10 hover:text-white transition-colors"
    >
      Ver eventos na ORYA
      <span aria-hidden>→</span>
    </Link>
  )
}
</div>
</section>
);
}

```

app/components/Ring.tsx

```

"use client";

import { useRef, useEffect } from "react";

interface RingProps {
  size?: number;
  speed?: number;
  glow?: boolean;
  bloom?: boolean;
}

export default function Ring({
  size = 120,
  speed = 1,
  glow = true,
  bloom = true,
}: RingProps) {
  const canvasRef = useRef<HTMLCanvasElement | null>(null);

  useEffect(() => {
    const canvas = canvasRef.current!;
    const ctx = canvas.getContext("2d", { alpha: true })!;
    let frame = 0;

    const DPR = Math.min(window.devicePixelRatio || 1, 2);
    canvas.width = size * DPR;
    canvas.height = size * DPR;
  }, [size, speed, glow, bloom]);
}

```

```

const CX = canvas.width / 2;
const CY = canvas.height / 2;
const R = (size * 0.42) * DPR;
const THICK = (size * 0.11) * DPR;

const COLORS = ["#6BFFFF", "#45E5FF", "#7300FF", "#FF00C8", "#6BFFFF"];

function conicGradient(angle = 0) {
  const g = ctx.createConicGradient(angle, CX, CY);
  const step = 1 / (COLORS.length - 1);
  COLORS.forEach((c, i) => g.addColorStop(i * step, c));
  return g;
}

function draw() {
  frame += 0.0025 * speed;

  ctx.clearRect(0, 0, canvas.width, canvas.height);

  ctx.lineWidth = THICK;
  ctx.lineCap = "round";

  // ANEL PRINCIPAL
  ctx.strokeStyle = conicGradient(frame);
  ctx.beginPath();
  ctx.arc(CX, CY, R, 0, Math.PI * 2);
  ctx.stroke();

  // INNER CLEAR
  ctx.globalCompositeOperation = "destination-out";
  ctx.beginPath();
  ctx.arc(CX, CY, R - THICK * 0.55, 0, Math.PI * 2);
  ctx.fill();
  ctx.globalCompositeOperation = "source-over";

  requestAnimationFrame(draw);
}

draw();
}, []);

return (
<div
  className="relative flex items-center justify-center"
  style={{ width: size, height: size }}>
>

/* OUTER GLOW */
{glow && (
<div
  className="absolute rounded-full pointer-events-none"
  style={{
    width: size * 1.35,
    height: size * 1.35,
    filter: "blur(48px)",
    opacity: 0.7,
    background:
      "conic-gradient(from 0deg, #6BFFFF, #45E5FF, #7300FF, #FF00C8, #6BFFFF)",
    mask: "radial-gradient(circle, transparent 58%, black 75%)",
    WebkitMask: "radial-gradient(circle, transparent 58%, black 75%)",
  }}
></div>
)}

/* INNER BLOOM */
{bloom && (
<div
  className="absolute rounded-full pointer-events-none"
  style={{
    width: size * 0.9,
    height: size * 0.9,
    filter: "blur(32px)",
    opacity: 0.35,
    background:
      "radial-gradient(circle, rgba(255,0,200,0.18), rgba(80,0,120,0.08), transparent 70%)",
  }}
></div>
)}

```

```

        )}

        /* CANVAS */
        <canvas ref={canvasRef} className="relative z-10" />
    </div>
};

}

```

app/components/tickets/TicketCard.tsx

```

"use client";

import Link from "next/link";

export type TicketCardProps = {
    id: string;
    quantity: number;
    pricePaid: number; // total pago por esta compra
    currency: string;
    createdAt?: string;
    event: {
        slug?: string;
        title: string;
        startDate?: string;
        locationName?: string;
        coverImageUrl?: string | null;
    };
    ticket: {
        name: string; // wave / tipo de bilhete
        description?: string | null;
    };
    qrToken?: string | null;
};

function formatDate(dateStr?: string) {
    if (!dateStr) return "";
    const d = new Date(dateStr);
    if (Number.isNaN(d.getTime())) return "";
    return new Intl.DateTimeFormat("pt-PT", {
        day: "2-digit",
        month: "short",
        year: "numeric",
        hour: "2-digit",
        minute: "2-digit",
    }).format(d);
}

function formatPrice(amount: number, currency: string) {
    if (!Number.isFinite(amount)) return "";
    return `${amount.toFixed(2)} ${currency || "EUR"}`;
}

export function TicketCard(props: TicketCardProps) {
    const { id, quantity, pricePaid, currency, event, ticket, qrToken } = props;

    const dateLabel = formatDate(event.startDate ?? props.createdAt);
    const totalLabel = formatPrice(pricePaid, currency);
    const unitPrice = quantity > 0 ? pricePaid / quantity : pricePaid;
    const unitPriceLabel = formatPrice(unitPrice, currency);

    const eventDateObj = event.startDate ? new Date(event.startDate) : null;
    const now = new Date();

    let statusLabel = "Confirmado";
    let statusClass =
        "border-emerald-400/50 bg-emerald-500/10 text-emerald-200";

    if (eventDateObj && !Number.isNaN(eventDateObj.getTime())) {
        const isPast = eventDateObj.getTime() < now.getTime();

        const isSameDay =
            eventDateObj.getFullYear() === now.getFullYear() &&
            eventDateObj.getMonth() === now.getMonth() &&
            eventDateObj.getDate() === now.getDate();

        if (isPast) {

```

```

        statusLabel = "Já aconteceu";
        statusClass = "border-white/25 bg-white/5 text-white/80";
    } else if (isSameDay) {
        statusLabel = "É hoje";
        statusClass = "border-sky-400/60 bg-sky-500/10 text-sky-100";
    }
}

const hasCover = Boolean(event.coverImageUrl);

return (
    <article className="group relative overflow-hidden rounded-2xl border border-white/10 bg-gradient-to-b from-white/5 via-black/80 to-black/95 hover:border-[#6BFFFF]/70 transition-colors shadow-[0_16px_45px_rgba(0,0,0,0.75)]">
        {/* Poster visual */}
        <div className="relative w-full overflow-hidden">
            <div className="relative aspect-[3/4] w-full">
                {hasCover ? (
                    // eslint-disable-next-line @next/next/no-img-element
                    <img
                        src={event.coverImageUrl as string}
                        alt={event.title}
                        className="h-full w-full object-cover transition-transform duration-500 group-hover:scale-[1.03]"
                    />
                ) : (
                    <div className="flex h-full w-full items-center justify-center bg-gradient-to-br from-[#1b1b2f] via-black to-[#141421] text-[11px] text-white/40">
                        Sem imagem de capa
                    </div>
                )}
            </div>
        </div>

        {/* Overlay gradient */}
        <div className="pointer-events-none absolute inset-0 bg-gradient-to-t from-black/90 via-black/40 to-transparent" />

        {/* Badges no topo */}
        <div className="absolute left-2 right-2 top-2 flex items-center justify-between gap-2">
            <span
                className={`${'inline-flex items-center rounded-full border px-2 py-0.5 text-[10px] font-medium ${statusClass}'}`}
            >
                {statusLabel}
            </span>
            <span className="inline-flex items-center rounded-full bg-black/70 px-2 py-0.5 text-[10px] font-semibold text-white/85">
                {quantity > 1 ? `${quantity} bilhetes` : "1 bilhete"}
            </span>
        </div>

        {/* Título + local + data na base do poster */}
        <div className="absolute inset-x-2 bottom-2 space-y-1">
            <p className="text-[11px] uppercase tracking-[0.16em] text-white/60">
                Bilhete ORYA
            </p>
            <h3 className="text-sm font-semibold leading-snug line-clamp-2">
                {event.title}
            </h3>
            {event.locationName && (
                <p className="text-[11px] text-white/70 line-clamp-1">
                    {event.locationName}
                </p>
            )}
            {dateLabel && (
                <p className="text-[11px] text-white/80">{dateLabel}</p>
            )}
        </div>
    </div>
</div>

        {/* Zona de detalhes por baixo do poster */}
        <div className="relative border-t border-white/10 bg-black/80 px-4 py-3 space-y-3">
            <div className="grid grid-cols-2 gap-2 text-[11px]">
                <div className="space-y-1">
                    <p className="text-white/50">Total pago</p>
                    <p className="text-white/90">{totalLabel}</p>
                </div>
                <div className="space-y-1">
                    <p className="text-white/50">Preço unitário</p>
                    <p className="text-white/90">{unitPriceLabel}</p>
                </div>
            <div className="space-y-1">

```

```

        <p className="text-white/50">Tipo de bilhete</p>
        <p className="text-white/90 line-clamp-1">{ticket.name}</p>
    </div>
    <div className="space-y-1">
        <p className="text-white/50">Referência</p>
        <p className="text-white/80 text-[10px] break-all">{id}</p>
    </div>
</div>

<p className="mt-1 text-[10px] text-white/40">
    QR code disponível neste bilhete. Em breve vais poder transferir
    bilhetes, revender e gerir tudo diretamente nesta página.
</p>

{/* Preview do QR Code pequeno, no canto inferior direito */}
<div className="mt-3 flex items-center justify-between gap-3">
    <div className="flex flex-col gap-1 text-[11px] text-white/70">
        <span className="inline-flex rounded-full bg-white/5 px-2 py-0.5 text-[10px] font-medium text-white/75">
            {quantity > 1 ? `${quantity} entradas` : "Entrada única"}
        </span>
        {ticket.description && (
            <span className="line-clamp-2 text-[10px] text-white/55">
                {ticket.description}
            </span>
        )}
    </div>

    {qrToken ? (
        <div className="shrink-0 rounded-2xl bg-white p-2 shadow-[0_0_28px_rgba(255,0,200,0.35)]">
            {/* eslint-disable-next-line @next/next/no-img-element */}
            <img
                src={`/api/qr/${qrToken}?theme=dark`}
                alt="QR Code do bilhete ORYA"
                className="h-20 w-20 object-contain"
            />
        </div>
    ) : (
        <div className="shrink-0 rounded-2xl border border-dashed border-white/25 bg-black/40 px-3 py-2 text-[10px] text-white/60 text-center max-w-[140px]">
            A preparar o QR deste bilhete...
        </div>
    )}
</div>

<div className="mt-3 flex items-center justify-between gap-2 text-[11px]">
    <button
        type="button"
        className="inline-flex items-center gap-1 rounded-full border border-white/25 px-3 py-1 text-white/80 hover:bg-white/10 transition-colors"
    >
        <Link href={event.slug ? `/eventos/${event.slug}` : "/explorar"}>
            Ver evento
        </Link>
    </button>
    <Link
        href={`/bilhete/${id}`}
        className="inline-flex items-center gap-1 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-3 py-1 font-semibold text-black shadow-[0_0_20px_rgba(107,255,255,0.7)] hover:scale-[1.03] active:scale-95 transition-transform"
    >
        Abrir bilhete
    </Link>
</div>
</div>
</article>
);
}

```

app/components/tickets/TicketLiveQr.tsx

```

"use client";

import { useEffect, useState } from "react";

type TicketLiveQrProps = {
    qrToken: string;
}

```

```

};

export default function TicketLiveQr({ qrToken }: TicketLiveQrProps) {
  const [refreshKey, setRefreshKey] = useState(() => Date.now());
  const [loadedAt, setLoadedAt] = useState<number | null>(null);
  const [mounted, setMounted] = useState(false);

  // Evitar hydration mismatch
  useEffect(() => {
    // Montamos o componente depois da hidratação para evitar desencontros entre SSR e cliente
    // eslint-disable-next-line react-hooks/set-state-in-effect
    setMounted(true);
  }, []);

  // Auto-refresh
  useEffect(() => {
    if (!mounted) return;

    const interval = setInterval(() => {
      setLoadedAt(null);
      setRefreshKey(Date.now());
    }, 15000);

    return () => clearInterval(interval);
  }, [mounted]);

  // Antes de montar no cliente → placeholder estático (SSR-safe)
  if (!mounted) {
    return (
      <div className="w-64 h-64 rounded-xl bg-white/10 animate-pulse" />
    );
  }

  return (
    <div className="flex flex-col items-center gap-2">
      <img
        key={refreshKey}
        src={`/api/qr/${qrToken}?r=${refreshKey}`}
        alt="QR Code ORYA"
        aria-label="Código QR do bilhete ORYA"
        loading="eager"
        className="w-64 h-64 rounded-xl bg-white p-4 transition-opacity duration-500 ${loadedAt ? "opacity-100" : "opacity-0"}"
        onLoad={() => setLoadedAt(Date.now())}
      />

      <p className="text-[11px] text-white/50 font-medium">
        QR atualiza a cada 15s
      </p>
    </div>
  );
}

```

app/components/tickets/TicketQrBox.tsx

```

"use client";

import { useEffect, useState } from "react";

type TicketQrBoxProps = {
  qrToken: string | null;
  purchaseId: string;
};

const REFRESH_INTERVAL_MS = 15_000; // 15s

export default function TicketQrBox({ qrToken, purchaseId }: TicketQrBoxProps) {
  const [refreshKey, setRefreshKey] = useState(() => Date.now());

  useEffect(() => {
    if (!qrToken) return;

    const id = setInterval(() => {
      setRefreshKey(Date.now());
    }, REFRESH_INTERVAL_MS);
  }, []);
}

```

```

    return () => clearInterval(id);
}, [qrToken]);

if (!qrToken) {
  return (
    <div
      className="flex h-48 w-48 flex-col items-center justify-center gap-2 rounded-2xl border border-white/20 bg-white/5
animate-pulse"
      aria-busy="true"
      aria-label="A carregar o QR do bilhete ORYA"
    >
      <div className="h-24 w-24 rounded-lg bg-white/15" />
      <p className="px-3 text-center text-[10px] text-white/70">
        A gerar o QR code deste bilhete... mantém esta página aberta.
      </p>
    </div>
  );
}

const qrSrc = `/api/qr/${qrToken}?t=${purchaseId}&r=${refreshKey}`;

return (
  <div className="animate-[fadeIn_0.4s_ease-out]">
    /* eslint-disable-next-line @next/next/no-img-element */
    <img
      src={qrSrc}
      alt="QR Code ORYA"
      className="mx-auto h-64 w-64 rounded-2xl bg-white p-4 shadow-[0_0_45px_rgba(107,255,255,0.35)] border border-white/20
object-contain"
    />
  </div>
);
}

```

app/em-breve/page.tsx

```

export const runtime = "nodejs";

import Link from "next/link";

export default function EmBrevePage() {
  return (
    <div className="min-h-screen bg-[#060711] text-white flex items-center justify-center px-4">
      <div className="max-w-md w-full rounded-3xl border border-white/10 bg-white/[0.04] p-8 shadow-[0_25px_80px_rgba(0,0,0,0.45)] text-center space-y-4">
        <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">ORYA</p>
        <h1 className="text-2xl font-semibold">Em breve</h1>
        <p className="text-sm text-white/70">
          Esta área está temporariamente disponível apenas para administradores. Estamos a preparar a melhor
          experiência para ti.
        </p>
        <div className="flex flex-col gap-3 pt-2">
          <Link
            href="/explorar"
            className="w-full rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-5 py-2.5 text-sm font-
semibold text-black shadow-[0_0_30px_rgba(107,255,255,0.22)] transition hover:brightness-110"
          >
            Voltar a explorar eventos
          </Link>
          <Link
            href="/"
            className="w-full rounded-full border border-white/20 px-5 py-2.5 text-sm font-semibold text-white hover:bg-white/10
transition"
          >
            Ir para a página inicial
          </Link>
        </div>
      </div>
    );
}

```

app/error.tsx

```

"use client";

import Link from "next/link";
import { useEffect } from "react";

type ErrorProps = {
  error: Error & { digest?: string };
  reset: () => void;
};

export default function Error({ error, reset }: ErrorProps) {
  useEffect(() => {
    // Se quiseres, podes enviar o erro para um serviço externo (Sentry, etc.)
    console.error("[ORYA error boundary]", error);
  }, [error]);

  return (
    <main className="min-h-screen bg-[radial-gradient(circle_at_top,_#1a1030_0,_#050509_45%,_#02020a_100%)] text-white flex items-center justify-center px-4">
      <div className="max-w-lg w-full rounded-3xl border border-white/10 bg-black/55 backdrop-blur-2xl px-6 py-8 md:px-8 md:py-10 shadow-[0_24px_80px_rgba(0,0,0,0.8)] space-y-6">
        <div className="flex items-center gap-3">
          <span className="inline-flex h-9 w-9 items-center justify-center rounded-2xl bg-gradient-to-tr from-[#FF00C8] via-[#6BFFF] to-[#1646F5] text-[11px] font-extrabold tracking-[0.18em]">
            OR
          </span>
          <div className="space-y-1">
            <p className="text-[11px] uppercase tracking-[0.22em] text-white/55">
              ORYA
            </p>
            <p className="text-sm text-white/80">Ups, algo correu mal.</p>
          </div>
        </div>

        <div className="space-y-4">
          <h1 className="text-2xl md:text-3xl font-semibold tracking-tight">
            Ocorreu um erro inesperado.
          </h1>
          <p className="text-sm text-white/65">
            Isto pode ter sido causado por um problema temporário na ligação,
            por dados inválidos ou por um bug que ainda estamos a caçar. Não te
            preocipes – nada foi cobrado nem perdido sem confirmação clara.
          </p>
        </div>

        <div className="space-y-3">
          <button
            type="button"
            onClick={reset}
            className="inline-flex w-full items-center justify-center gap-2 rounded-xl bg-gradient-to-r from-[#FF00C8] via-[#6BFFF] to-[#1646F5] px-4 py-2.5 text-xs font-semibold text-black shadow-[0_0_32px_rgba(107,255,255,0.6)] transition-transform hover:scale-[1.02] active:scale-95">
            Tentar novamente
            <span className="text-[14px]">v</span>
          </button>
          <Link
            href="/explorar"
            className="inline-flex w-full items-center justify-center rounded-xl border border-white/15 bg-white/5 px-4 py-2.5 text-[11px] font-medium text-white/80 hover:bg-white/10 transition-colors">
            Voltar a explorar eventos
          </Link>
        </div>

        {process.env.NODE_ENV === "development" && error?.digest && (
          <p className="text-[10px] text-white/40 text-center">
            Código do erro: <span className="font-mono">{error.digest}</span>
          </p>
        )}
      </div>
    </main>
  );
}

```

app/eventos/[slug]/EventPageClient.tsx

```
"use client";

import ModalCheckout from "@/app/components/checkout/ModalCheckout";

type EventPageClientProps = {
  cover: string | null;
  event: Record<string, unknown>;
  uiTickets: Record<string, unknown>[];
  currentUserId: string | null;
};

export default function EventPageClient(props: EventPageClientProps) {
  void props;
  return <ModalCheckout />;
}
```

app/eventos/[slug]/PadelSignupInline.tsx

```
"use client";

import { useState } from "react";
import { useRouter } from "next/navigation";

type Props = {
  eventId: number;
  organizerId: number | null;
  ticketTypeId: number | null;
  categoryId?: number | null;
  padelV2Enabled: boolean;
  templateType?: string | null;
};

export default function PadelSignupInline({
  eventId,
  organizerId,
  ticketTypeId,
  categoryId,
  padelV2Enabled,
  templateType,
}: Props) {
  const router = useRouter();
  const [loadingFull, setLoadingFull] = useState(false);
  const [loadingSplit, setLoadingSplit] = useState(false);

  const isPadelV2 = templateType === "PADEL" && padelV2Enabled;
  const canProceed = isPadelV2 && organizerId && ticketTypeId;

  const createPairing = async (mode: "FULL" | "SPLIT") => {
    if (!canProceed) {
      alert("Sem configuração válida para inscrição Padel.");
      return;
    }
    const setLoading = mode === "FULL" ? setLoadingFull : setLoadingSplit;
    setLoading(true);
    try {
      const res = await fetch("/api/padel/pairings", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          eventId,
          organizerId,
          categoryId: categoryId ?? undefined,
          paymentMode: mode,
        }),
      });
      const json = await res.json();
      if (!res.ok || !json?.ok || !json?.pairing?.id) {
        throw new Error(json?.error || "Não foi possível iniciar inscrição Padel.");
      }
      const pairingId = json.pairing.id as number;
      if (mode === "SPLIT") {
        router.push(`'/eventos/${slug}?pairingId=${pairingId}`);
      }
    } catch (error) {
      console.error(error);
    }
  };
}
```

```

    } else {
      router.push(`/eventos/${slug}?pairingId=${pairingId}&mode=full`);
    }
  } catch (err) {
    console.error("[PadelSignupInline] erro", err);
    alert(err instanceof Error ? err.message : "Erro ao iniciar inscrição.");
  } finally {
    setLoading(false);
  }
};

if (!isPadelV2 || !ticketTypeId) return null;

return (
  <div className="space-y-3 rounded-xl border border-white/15 bg-black/45 p-4">
    <div className="flex items-center justify-between gap-2">
      <div>
        <p className="text-xs uppercase tracking-[0.14em] text-white/60">Padel (duas)</p>
        <h4 className="text-lg font-semibold text-white">Escolhe como queres pagar</h4>
      </div>
    </div>
    <div className="space-y-2">
      <button
        type="button"
        disabled={!canProceed || loadingFull}
        onClick={() => createPairing("FULL")}
        className="w-full rounded-full bg-white text-black px-4 py-2 font-semibold shadow hover:brightness-105
disabled:opacity-60"
      >
        {loadingFull ? "A preparar..." : "Pagar dupla inteira"}
      </button>
      <button
        type="button"
        disabled={!canProceed || loadingSplit}
        onClick={() => createPairing("SPLIT")}
        className="w-full rounded-full border border-white/20 px-4 py-2 font-semibold text-white hover:bg-white/10
disabled:opacity-60"
      >
        {loadingSplit ? "A preparar..." : "Pagar só o meu lugar"}
      </button>
      {!canProceed && (
        <p className="text-[12px] text-amber-200">Bilhetes indisponíveis ou configuração Padel v2 desligada.</p>
      )}
    </div>
  </div>
);
}

```

app/eventos/[slug]/page.tsx

```

// app/eventos/[slug]/page.tsx
import { prisma } from "@lib/prisma";
import { CheckoutProvider } from "@app/components/checkout/contextoCheckout";
import { notFound } from "next/navigation";
import { headers } from "next/headers";
import WavesSectionClient, { type WaveTicket, type WaveStatus } from "./WavesSectionClient";
import Link from "next/link";
import EventPageClient from "./EventPageClient";
import { createSupabaseServer } from "@lib/supabaseServer";
import type { Metadata } from "next";
import type { Prisma } from "@prisma/client";
import Image from "next/image";
import { defaultBlurDataURL, optimizeImageUrl } from "@lib/image";
import PadelSignupInline from "./PadelSignupInline";
import { buildPadelEventSnapshot } from "@lib/padel/eventSnapshot";

type EventPageParams = { slug: string };
type EventPageParamsInput = EventPageParams | Promise<EventPageParams>;

export async function generateMetadata(
  { params }: { params: EventPageParamsInput },
): Promise<Metadata> {
  const resolved = await params;
  const slug = resolved?.slug;

  if (!slug) {

```

```

        return {
          title: "Evento | ORYA",
          description: "Explora eventos na ORYA.",
        };
      }

      const event = await prisma.event.findUnique({
        where: { slug },
        select: {
          title: true,
          description: true,
          locationName: true,
        },
      });

      if (!event) {
        return {
          title: "Evento não encontrado | ORYA",
          description: "Este evento já não está disponível.",
        };
      }

      const location = event.locationName || "ORYA";
      const baseTitle = event.title || "Evento ORYA";

      return {
        title: `${baseTitle} | ORYA`,
        description:
          event.description && event.description.trim().length > 0
            ? event.description
            : `Descobre o evento ${baseTitle} em ${location} na ORYA.`,
      };
    }

    type EventResale = {
      id: string;
      ticketId: string;
      price: number;
      currency: string;
      seller?: {
        username: string | null;
        fullName: string | null;
      } | null;
      ticketTypeName?: string | null;
    };

    function getWaveStatus(ticket: {
      startsAt: Date | null;
      endsAt: Date | null;
      totalQuantity: number | null;
      soldQuantity: number;
    }) {
      const now = new Date();

      if (
        ticket.totalQuantity !== null &&
        ticket.totalQuantity !== undefined &&
        ticket.soldQuantity >= ticket.totalQuantity
      ) {
        return "sold_out" as const;
      }

      if (ticket.startsAt && now < ticket.startsAt) {
        return "upcoming" as const;
      }

      if (ticket.endsAt && now > ticket.endsAt) {
        return "closed" as const;
      }

      return "on_sale" as const;
    }

    export default async function EventPage({ params }: { params: EventPageParamsInput }) {
      const { slug } = await params;

      if (!slug) {
        return notFound();
      }
    }
  
```

```

}

type EventWithTickets = Prisma.EventGetPayload<{
  include: { ticketTypes: true };
}>;

type TicketTypeWithVisibility =
  EventWithTickets["ticketTypes"][number] & { isVisible?: boolean | null };

const supabase = await createSupabaseServer();
const {
  data: { user },
} = await supabase.auth.getUser();
const profile = user
  ? await prisma.profile.findUnique({ where: { id: user.id } })
  : null;
const isAdmin = Array.isArray(profile?.roles) ? profile.roles.includes("admin") : false;

const event = await prisma.event.findUnique({
  where: { slug },
  include: { ticketTypes: true, padelTournamentConfig: true },
});
if (!event) {
  notFound();
}
if (event.isTest && !isAdmin) {
  notFound();
}
const isPadel = event.templateType === "PADEL";
const padelSnapshot = isPadel ? await buildPadelEventSnapshot(event.id) : null;

// Buscar bilhetes ligados a este evento (para contagem de pessoas)
const safeLocationName = event.locationName || "Local a anunciar";
const safeTimezone = event.timezone || "Europe/Lisbon";
const safeOrganizer = "ORYA";

// Nota: no modelo atual, não determinamos o utilizador autenticado neste
// Server Component para evitar erros de escrita de cookies.
// A verificação de "já tens bilhete" pode ser feita no cliente.
const userId: string | null = null;
const currentUserHasTicket = false;

const startDateObj = event.startsAt;
const endDateObj = event.endsAt ?? event.startsAt;

const dateFormatter = new Intl.DateTimeFormat("pt-PT", {
  weekday: "long",
  day: "numeric",
  month: "long",
  timeZone: safeTimezone,
});

const timeFormatter = new Intl.DateTimeFormat("pt-PT", {
  hour: "2-digit",
  minute: "2-digit",
  timeZone: safeTimezone,
});

const date = dateFormatter.format(startDateObj);
const time = timeFormatter.format(startDateObj);
const endTime = timeFormatter.format(endDateObj);

const rawCover =
  event.coverImageUrl && event.coverImageUrl.trim().length > 0
    ? event.coverImageUrl
    : "https://images.unsplash.com/photo-1541987392829-5937c1069305?q=80&w=1600";
const cover = optimizeImageUrl(rawCover, 1200, 72, "webp");
// versão ultra-leve apenas para o blur de fundo (mantém o efeito mas evita puxar MBs)
const blurredCover = optimizeImageUrl(rawCover, 120, 20, "webp");

const nowDate = new Date();
const eventEnded = endDateObj < nowDate;

const ticketTypesWithVisibility = event.ticketTypes as TicketTypeWithVisibility[];

const orderedTickets = ticketTypesWithVisibility
  .filter((t) => t.isVisible ?? true)
  .sort((a, b) => {

```

```

    const ao = a.sortOrder ?? 0;
    const bo = b.sortOrder ?? 0;
    if (ao !== bo) return ao - bo;
    return a.price - b.price;
});

const uiTickets: WaveTicket[] = orderedTickets.map((t, index) => {
  const rawStatus = String(t.status || "").toUpperCase();
  const remaining =
    t.totalQuantity === null || t.totalQuantity === undefined
      ? null
      : t.totalQuantity - t.soldQuantity;

  const statusFromEnum =
    rawStatus === "CLOSED" || rawStatus === "ENDED" || rawStatus === "OFF_SALE"
      ? "closed"
      : rawStatus === "SOLD_OUT"
        ? "sold_out"
        : rawStatus === "UPCOMING"
          ? "upcoming"
          : "on_sale";

  // Override: if remaining is 0, this wave é sold_out (mesmo com status)
  const finalStatus: WaveStatus =
    remaining !== null && remaining <= 0
      ? "sold_out"
      : statusFromEnum !== "on_sale"
        ? (statusFromEnum as WaveStatus)
        : getWaveStatus({
            startsAt: t.startsAt,
            endsAt: t.endsAt,
            totalQuantity: t.totalQuantity,
            soldQuantity: t.soldQuantity,
          });

  return {
    id: String(t.id),
    name: t.name?.trim() || `Wave ${index + 1}`,
    price: (t.price ?? 0) / 100,
    currency: t.currency,
    totalQuantity: t.totalQuantity,
    soldQuantity: t.soldQuantity,
    remaining,
    status: finalStatus as WaveStatus,
    startsAt: t.startsAt ? t.startsAt.toISOString() : null,
    endsAt: t.endsAt ? t.endsAt.toISOString() : null,
    available:
      finalStatus === "on_sale"
        ? remaining === null
          ? true
          : remaining > 0 && !eventEnded
        : false,
    isVisible: t.isVisible ?? true,
  };
});

const minTicketPrice =
  uiTickets.length > 0
    ? uiTickets.reduce(
        (min, t) => (t.price < min ? t.price : min),
        uiTickets[0].price,
      )
    : null;

const displayPriceFrom = minTicketPrice;
const anyOnSale = uiTickets.some((t) => t.status === "on_sale");
const anyUpcoming = uiTickets.some((t) => t.status === "upcoming");
const allClosed = uiTickets.length > 0 && uiTickets.every((t) => t.status === "closed");
const allSoldOut = uiTickets.length > 0 && uiTickets.every((t) => t.status === "sold_out");

// Carregar revendas deste evento via API F5-9
let resales: EventResale[] = [];
try {
  const headersList = await headers();
  const protocol = headersList.get("x-forwarded-proto") ?? "http";
  const host = headersList.get("host");

  if (host) {

```

```

const baseUrl = `${protocol}://${host}`;
const res = await fetch(
  `${baseUrl}/api/eventos/${encodeURIComponent(slug)}/resales`,
  { cache: "no-store" }
);

if (res.ok) {
  const data = (await res.json().catch(() => null)) as
    | { ok?: boolean; resales?: EventResale[] } |
    | null;

  if (data?.ok && Array.isArray(data.resales)) {
    resales = data.resales;
  }
} else {
  console.error(
    "Falha ao carregar revendas para o evento",
    slug,
    res.status,
  );
}

} catch (err) {
  console.error("Erro ao carregar revendas para o evento", slug, err);
}

const showPriceFrom = !event.isFree && minTicketPrice !== null;

const padelV2Enabled = Boolean(event.padelTournamentConfig?.padelV2Enabled);
const defaultPadelTicketId = () => {
  const eligible = orderedTickets.filter((t) => {
    const remaining =
      t.totalQuantity === null || t.totalQuantity === undefined
        ? null
        : t.totalQuantity - t.soldQuantity;
    const onSale = String(t.status || "").toUpperCase() === "ON_SALE";
    const hasStock = remaining === null ? true : remaining > 0;
    return onSale && hasStock;
  });
  if (!eligible.length) return null;
  const cheapest = eligible.reduce((min, cur) => (cur.price < min.price ? cur : min), eligible[0]);
  return cheapest.id ?? null;
}();

return (
<main className="relative orya-body-bg min-h-screen w-full text-white">
  <CheckoutProvider>
    {/* BG: blur da capa a cobrir o topo da página com transição super suave para o fundo ORYA */}
    <div
      className="pointer-events-none absolute inset-x-0 top-0 h-[160vh] overflow-hidden"
      aria-hidden="true"
    >
      {/* camada principal: cover blur com máscara para fazer o fade vertical muito suave */}
      <div
        className="h-full w-full scale-[1.25]"
        style={{
          backgroundImage: `url(${blurredCover})`,
          backgroundSize: "cover",
          backgroundPosition: "center",
          filter: "blur(40px)",
          WebkitFilter: "blur(40px)",
          transform: "scale(1.25)",
          WebkitTransform: "scale(1.25)",
          WebkitMaskImage:
            "linear-gradient(to bottom, rgba(0,0,0,1) 0%, rgba(0,0,0,0.95) 18%, rgba(0,0,0,0.8) 35%, rgba(0,0,0,0.55) 55%, rgba(0,0,0,0.3) 75%, rgba(0,0,0,0) 100%)",
          maskImage:
            "linear-gradient(to bottom, rgba(0,0,0,1) 0%, rgba(0,0,0,0.95) 18%, rgba(0,0,0,0.8) 35%, rgba(0,0,0,0.55) 55%, rgba(0,0,0,0.3) 75%, rgba(0,0,0,0) 100%)",
        }}
      />
      {/* overlay extra para garantir legibilidade no topo da hero e uma transição ainda mais orgânica */}
      <div className="absolute inset-0 bg-gradient-to-b from-black/55 via-black/35 to-transparent" />
      {/* fade tardio para preto para unir com o fundo */}
      <div
        className="absolute inset-0"
        style={{
          background:

```

```

        "linear-gradient(to bottom, rgba(0,0,0,0) 0%, rgba(0,0,0,0) 70%, rgba(0,0,0,0.45) 82%, rgba(0,0,0,0.8) 92%,  

    rgba(0,0,0,1) 100%)",  

    })  

    />  

</div>  
  

{/* ===== HERO ===== */}  

<section className="relative z-10 w-full pt-24 pb-10 md:pt-28 md:pb-12">  

    <div className="mx-auto mb-4 flex w-full max-w-6xl items-center px-4 md:px-8">  

        <Link  

            href="/explorar"  

            className="inline-flex items-center gap-2 text-xs font-medium text-white/75 transition hover:text-white"  

        >  

            <span className="text-lg leading-none"></span>  

            <span>Voltar a explorar</span>  

        </Link>  

    </div>  

    <div className="mx-auto flex w-full max-w-6xl items-end px-4 md:px-8">  

        <div className="flex w-full flex-col gap-6 md:flex-row md:items-stretch">  

            /* CARTÃO VIDRO - INFO DO EVENTO */  

            <div className="inline-flex max-w-3xl flex-1 flex-col gap-4 rounded-3xl border border-white/18 bg-gradient-to-br  

from-[#FF8AD908] via-[#9BE7FFF] to-[#020617e6] px-6 py-5 shadow-[_0_24px_60px_rgba(0,0,0.75)] backdrop-blur-2xl md:px-8 md:py-  

7">  

                <p className="mb-1 text-xs uppercase tracking-[0.22em] text-white/70">  

                    Evento ORYA · {safeLocationName}  

                </p>  
  

                /* Badges / info rápida */  

                <div className="flex flex-wrap items-center gap-3 text-xs text-white/85 md:text-sm">  

                    <span className="rounded-full border border-white/25 bg-black/35 px-3 py-1">  

                        {date.charAt(0).toUpperCase() + date.slice(1)} · {time} -{" "}  

                        {endTime}  

                    </span>  

                    <span className="rounded-full border border-white/25 bg-black/35 px-3 py-1">  

                        {safeLocationName}  

                    </span>  

                    {event.isFree ?  

                        <span className="rounded-full border border-emerald-400/50 bg-emerald-500/18 px-3 py-1 text-emerald-100">  

                            Entrada gratuita  

                        </span>  

                    ) : showPriceFrom ?  

                        <span className="rounded-full border border-fuchsia-400/50 bg-fuchsia-500/18 px-3 py-1 text-fuchsia-100">  

                            Desde {(displayPriceFrom ?? 0).toFixed(2)}€  

                        </span>  

                    ) : (  

                        <span className="rounded-full border border-fuchsia-400/40 bg-fuchsia-500/12 px-3 py-1 text-fuchsia-100/90">  

                            Preço a anunciar  

                        </span>  

                    )  

                </div>  

            </div>  
  

            /* Titulo */  

            <h1 className="bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] bg-clip-text text-3xl font-extrabold  

leading-tight text-transparent md:text-5xl lg:text-6xl md:leading-[1.03]">  

            {event.title}  

</h1>  
  

            /* badge "Já tens bilhete" */  

            {currentUserHasTicket && (  

                <div className="inline-flex items-center gap-2 rounded-full border border-emerald-400/70 bg-emerald-500/18 px-3  

py-1 text-xs text-emerald-100">  

                    <span className="text-sm">✓</span>  

                    <span>Já tens bilhete para este evento</span>  

                </div>  

            )}  
  

            <div className="mt-3 flex flex-wrap items-center gap-3 text-sm">  

                {!eventEnded && (  

                    <a  

                        href="#bilhetes"  

                        className="inline-flex items-center gap-2 rounded-full bg-white px-4 py-1.5 text-xs font-semibold text-black  

shadow-[_0_0_30px_rgba(255,255,255,0.25)] transition-transform hover:scale-105 active:scale-95 md:text-sm"  

                    >  

                        {event.isFree ? "Garantir lugar" : "Ver bilhetes"}  

                        <span className="text-xs">!</span>  

                    </a>  

                )}  

                {!eventEnded && showPriceFrom && (

```

```

        <span className="text-xs text-white/80 md:text-sm">
          A partir de(" ")
          <span className="font-semibold text-white">
            {(displayPriceFrom ?? 0).toFixed(2)} €
          </span>
        </span>
      )}
    </div>
</div>

/* POSTER / CAPA DO EVENTO À DIREITA (maior, alinhado em altura com o card) */
<div className="md:w-[280px] lg:w-[320px] flex-shrink-0 flex">
  <div className="relative mt-4 h-60 w-full overflow-hidden rounded-3xl border border-white/20 bg-white/5 shadow-[0_22px_50px_rgba(0,0,0,0.9)] backdrop-blur-2xl md:mt-0 md:h-full min-h-[230px]">
    <Image
      src={cover}
      alt={`Capa do evento ${event.title}`}
      fill
      priority
      fetchPriority="high"
      sizes="(max-width: 768px) 90vw, (max-width: 1200px) 40vw, 480px"
      className="object-cover"
      placeholder="blur"
      blurDataURL={defaultBlurDataURL}
    />
    <div className="pointer-events-none absolute inset-0 bg-gradient-to-t from-black/35 via-transparent to-white/0" />
  </div>
</div>
</div>
</div>

<div className="mx-auto mt-2 max-w-6xl px-4 md:px-8">
  <p className="text-xs text-white/70">
    Capa oficial do evento · {safeOrganizer}
  </p>
</div>
</section>

/* ====== CONTENT AREA ====== */
<section className="relative z-10 mx-auto grid max-w-6xl grid-cols-1 gap-16 px-6 pb-20 pt-8 md:grid-cols-3 md:px-10 md:pt-10">

  /* LEFT SIDE - Info + Descrição */
  <div className="space-y-10 md:col-span-2">
    <div>
      <h2 className="mb-4 text-2xl font-semibold">Sobre o evento</h2>
      <p className="whitespace-pre-line leading-relaxed text-white/80">
        {event.description &&
        event.description.trim().length > 0
        ? event.description
        : "A descrição deste evento será atualizada em breve."}
      </p>
    </div>
  </div>

  /* RIGHT SIDE - CARD DE INFORMAÇÕES / TICKETS */
  <div className="space-y-6 rounded-2xl border border-white/15 bg-gradient-to-br from-[#FF8AD908] via-[#9BE7FF14] to-[#020617E6] p-6 shadow-[0_22px_50px_rgba(0,0,0,0.85)] backdrop-blur-2xl">
    <div>
      <h3 className="mb-2 text-xl font-semibold">Data & Hora</h3>
      <p className="text-white/80">
        {date.charAt(0).toUpperCase() + date.slice(1)} · {time} -{" "}
        {endTime}
      </p>
    </div>

    <div>
      <h3 className="mb-2 text-xl font-semibold">Local</h3>
      <p className="text-white/80">{safeLocationName}</p>
      {event.address && (
        <p className="text-sm text-white/60">{event.address}</p>
      )}
    </div>

    {!eventEnded ? (
      <div
        id="bilhetes"
    
```

```

    className="scroll-mt-24 space-y-4 border-t border-white/12 pt-4"
  >
  <div className="flex items-center justify-between gap-2">
    <h3 className="text-xl font-semibold">Bilhetes</h3>
    {!event.isFree && showPriceFrom && (
      <span className="text-xs text-white/75">
        A partir de{" "}
        <span className="font-semibold text-white">
          {(displayPriceFrom ?? 0).toFixed(2)} €
        </span>
      </span>
    )}
  </div>

{event.isFree ? (
  <div className="flex items-center justify-between gap-3 rounded-xl border border-emerald-500/40 bg-emerald-500/15 px-3.5 py-2.5 text-sm text-emerald-100">
    <div>
      <p className="font-semibold">Entrada gratuita</p>
      <p className="text-[11px] text-emerald-100/85">
        Basta garantir o teu lugar – não há custo de bilhete.
      </p>
    </div>
  </div>
) : uiTickets.length === 0 ? (
  <div className="rounded-xl border border-white/12 bg-black/45 px-3.5 py-2.5 text-sm text-white/80">
    Ainda não há waves configuradas para este evento.
  </div>
) : allSoldOut ? (
  <div className="rounded-xl border border-orange-400/40 bg-orange-500/15 px-3.5 py-2.5 text-sm text-orange-100">
    <div>
      <p className="font-semibold">Evento esgotado</p>
      <p className="text-[11px] text-orange-100/85">
        Não há mais bilhetes disponíveis para este evento.
      </p>
    </div>
  </div>
) : !anyOnSale && anyUpcoming ? (
  <div className="rounded-xl border border-yellow-400/40 bg-yellow-500/15 px-3.5 py-2.5 text-sm text-yellow-100">
    <div>
      <p className="font-semibold">Vendas ainda não abriram</p>
      <p className="text-[11px] text-yellow-100/85">
        As vendas de bilhetes para este evento ainda não abriram. Volta mais tarde!
      </p>
    </div>
  </div>
) : allClosed ? (
  <div className="rounded-xl border border-white/12 bg-black/45 px-3.5 py-2.5 text-sm text-white/80">
    <div>
      <p className="font-semibold">Vendas encerradas</p>
      <p className="text-[11px] text-white/70">
        As vendas para este evento já encerraram.
      </p>
    </div>
  </div>
) : (
  <WavesSectionClient
    slug={event.slug}
    tickets={uiTickets}
    isFreeEvent={event.isFree}
    checkoutUiVariant={isPadel ? "PADEL_TOURNAMENT" : "EVENT_DEFAULT"}>
  />
)
}

{resales.length > 0 && (
  <div className="mt-6 border-t border-white/15 pt-4 space-y-3">
    <div className="flex items-center justify-between gap-2">
      <h3 className="text-base font-semibold">
        Bilhetes entre utilizadores
      </h3>
      <span className="text-xs text-white/70">
        {resales.length} oferta
        {resales.length === 1 ? "" : "s"} de revenda
      </span>
    </div>
    <p className="text-xs text-white/65">
      Estes bilhetes são vendidos por outros utilizadores da ORYA.
    </p>
  </div>
)
}

```

```

O pagamento é feito de forma segura através da plataforma.
</p>

<div className="space-y-3">
  {resales.map((r) => (
    <div
      key={r.id}
      className="flex items-center justify-between rounded-xl border border-white/15 bg-black/40 px-3.5 py-2.5
text-sm"
    >
      <div className="flex flex-col gap-0.5">
        <div className="flex flex-wrap items-center gap-2">
          <span className="font-medium">
            {r.ticketTypeName ?? "Bilhete ORYA"}
          </span>
          {r.seller && (
            <span className="text-xs text-white/60">
              por{" "}
              {r.seller.username
                ? `#${r.seller.username}`
                : r.seller.fullName ?? "utilizador ORYA"}
            </span>
          )}
        </div>
        <span className="text-xs text-white/65">
          Preço pedido:{" "}
          <span className="font-semibold text-white">
            {(r.price / 100).toFixed(2)} €
          </span>
        </span>
      </div>
      <Link
        href={`/resale/${r.id}`}
        className="inline-flex items-center rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-
[#1646F5] px-3 py-1.5 text-xs font-semibold text-black shadow-[0_0_18px_rgba(107,255,255,0.65)] hover:scale-[1.01] active:scale-
95 transition-transform"
      >
        Comprar agora
      </Link>
    </div>
  ))}
</div>
</div>
)}
</div>
) : (
  <div className="mt-4 rounded-xl border border-white/15 bg-black/60 px-4 py-3 text-sm text-white/85">
    Este evento já terminou. Bilhetes e inscrições deixaram de estar
    disponíveis.
  </div>
)
}

{padelSnapshot && (
  <div className="mt-4 space-y-3 rounded-xl border border-white/15 bg-black/40 px-4 py-3 text-white">
    <div className="flex flex-wrap items-center justify-between gap-2">
      <div>
        <p className="text-[11px] uppercase tracking-[0.16em] text-white/60">Padel</p>
        <h3 className="text-base font-semibold">{padelSnapshot.title}</h3>
        <p className="text-[12px] text-white/65">
          {padelSnapshot.clubName || "Clube a anunciar"} · {padelSnapshot.clubCity || "Cidade em breve"}
        </p>
      </div>
      <span className="rounded-full border border-white/20 bg-white/10 px-2 py-1 text-[11px] text-white/75">
        Estado: {padelSnapshot.status}
      </span>
    </div>
    <div className="grid gap-2 sm:grid-cols-2 lg:grid-cols-4">
      {padelSnapshot.timeline.map((step) => (
        <div
          key={step.key}
          className={` rounded-lg border px-3 py-2 text-sm ${{
            step.state === "done"
              ? "border-emerald-400/40 bg-emerald-500/10 text-emerald-50"
              : step.state === "active"
                ? "border-[#6BFFFF]/60 bg-[#0b1224] text-white"
                : "border-white/15 bg-white/5 text-white/70"
            }}`})
      )}
    </div>
  </div>
)
}

```

```

        >
          <p className="font-semibold">{step.label}</p>
          <p className="text-[12px] opacity-80">{step.cancelled ? "Cancelado" : step.date ? dateFormatter.format(new Date(step.date)) : "Data a anunciar"}</p>
        </div>
      )}
    </div>
  )}
<div className="grid gap-2 md:grid-cols-2 text-[13px]">
  <div className="rounded-lg border border-white/10 bg-white/5 p-3">
    <p className="text-[11px] uppercase tracking-[0.12em] text-white/60">Clubes</p>
    <p className="text-white/80">
      Principal: <span className="font-semibold text-white">{padelSnapshot.clubName || "A anunciar"}</span>
    </p>
    <p className="text-white/70">
      Parceiros:< " ">
      {padelSnapshot.partnerClubs && padelSnapshot.partnerClubs.length > 0
       ? padelSnapshot.partnerClubs.map((c) => c.name || `Clube ${c.id}`).join(" · ")
       : "-"}
    </p>
  </div>
  <div className="rounded-lg border border-white/10 bg-white/5 p-3">
    <p className="text-[11px] uppercase tracking-[0.12em] text-white/60">Courts</p>
    {padelSnapshot.courts && padelSnapshot.courts.length > 0 ? (
      <div className="flex flex-wrap gap-2">
        {padelSnapshot.courts.map((c, idx) => (
          <span key={`${c.name}-${idx}`} className="rounded-full border border-white/15 bg-black/30 px-2 py-1 text-[12px]">
            {c.name} {c.clubName ? `· ${c.clubName}` : ""} {c.indoor ? `· Indoor` : ""}
          </span>
        )))
      </div>
    ) : (
      <p className="text-white/70 text-[12px]">Courts a definir.</p>
    )}
  </div>
</div>
</div>
)
}
{event.templateType === "PADEL" && padelV2Enabled && defaultPadelTicketId && (
  <div className="border-t border-white/15 pt-6">
    <PadelSignupInline
      eventId={event.id}
      organizerId={event.organizerId ?? null}
      ticketTypeId={defaultPadelTicketId as number}
      categoryId={event.padelTournamentConfig?.defaultCategoryId ?? null}
      padelV2Enabled={padelV2Enabled}
      templateType={event.templateType}
    />
  </div>
)
}
</div>
</section>
<EventPageClient
  event={event}
  uiTickets={uiTickets}
  cover={cover}
  currentUserID={userId}
/>
</CheckoutProvider>
</main>
);
}
export const dynamic = "force-dynamic";
export const revalidate = 0;

```

app/eventos/[slug]/WavesSectionClient.tsx

```

"use client";

import { useState } from "react";
import { useCheckout } from "@app/components/checkout/contextoCheckout";

export type WaveStatus = "on_sale" | "upcoming" | "closed" | "sold_out";

export type WaveTicket = {

```

```

id: string;
name: string;
price: number;
currency: string;
remaining: number | null; // null = stock ilimitado
status: WaveStatus;
startsAt: string | null;
endsAt: string | null;
available: boolean;
isVisible: boolean;
// info extra de stock (opcional, vindo da API)
totalQuantity?: number | null;
soldQuantity?: number;
};

type WavesSectionClientProps = {
  slug: string;
  tickets: WaveTicket[];
  // para sabermos se devemos ir para checkout ou fazer "join" direto
  isFreeEvent?: boolean;
  checkoutUiVariant?: "EVENT_DEFAULT" | "PADEL_TOURNAMENT";
};

type FeedbackType = "success" | "error";

type FeedbackState = {
  [ticketId: string]: {
    type: FeedbackType;
    message: string;
  };
};

function formatDateTime(value: string | null) {
  if (!value) return null;
  const d = new Date(value);
  if (Number.isNaN(d.getTime())) return null;

  return d.toLocaleString("pt-PT", {
    day: "2-digit",
    month: "short",
    hour: "2-digit",
    minute: "2-digit",
  });
}

function computeRemainingAndStatus(
  prev: WaveTicket,
  updated: { totalQuantity?: number | null; soldQuantity?: number },
): { remaining: number | null; status: WaveStatus } {
  const total =
    updated.totalQuantity !== undefined
      ? updated.totalQuantity
      : prev.totalQuantity ?? null;

  const sold =
    updated.soldQuantity !== undefined
      ? updated.soldQuantity
      : prev.soldQuantity ?? 0;

  let remaining: number | null = null;

  if (total === null || total === undefined) {
    remaining = null; // ilimitado
  } else {
    const diff = total - sold;
    remaining = diff < 0 ? 0 : diff;
  }

  let status: WaveStatus = prev.status;

  if (total !== null && total !== undefined && sold >= total) {
    status = "sold_out";
  } else if (status === "sold_out") {
    status = "sold_out";
  }
}

// Se estiver marcada como indisponível/oculta, tratamos como encerrada visualmente
if (!prev.available || !prev.isVisible) {

```

```

        status = "closed";
    }

    return { remaining, status };
}

export default function WavesSectionClient({
    slug,
    tickets: initialTickets,
    isFreeEvent,
    checkoutUiVariant = "EVENT_DEFAULT",
}: WavesSectionClientProps) {
    const { abrirCheckout, atualizarDados } = useCheckout();
    const [tickets, setTickets] = useState<WaveTicket[]>(initialTickets);
    const [loadingId, setLoadingId] = useState<string | null>(null);
    const [feedback, setFeedback] = useState<FeedbackState>({});

    async function handlePurchase(ticketId: string) {
        const selectedTicket = tickets.find((t) => t.id === ticketId);
        if (!selectedTicket) return;

        // limpar feedback antigo
        setFeedback((prev) => {
            const clone = { ...prev };
            delete clone[ticketId];
            return clone;
        });

        setLoadingId(ticketId);

        try {
            // Todo o checkout (pago ou grátis) passa pelo modal/core único.
            abrirCheckout({
                slug,
                ticketId,
                price: selectedTicket.price,
                ticketName: selectedTicket.name,
                additional: {
                    checkoutUiVariant,
                },
                waves: tickets,
            });
        } catch (err) {
            console.error(err);
            setFeedback((prev) => ({
                ...prev,
                [ticketId]: {
                    type: "error",
                    message: "Erro inesperado ao processar a ação. Tenta outra vez.",
                },
            }));
        } finally {
            setLoadingId(null);
        }
    }

    const visibleTickets = tickets.filter((t) => t.isVisible);
    const purchasableTickets = visibleTickets.filter(
        (t) => t.status === "on_sale" || t.status === "upcoming",
    );

    // 🔥 Calcular preço mínimo (defensivo para o caso de não haver bilhetes visíveis)
    const minPrice =
        purchasableTickets.length > 0
            ? Math.min(...purchasableTickets.map((t) => t.price))
            : null;

    return (
        <div className="mt-6 w-full">
            <div className="rounded-2xl bg-white/5 border border-white/10 px-5 py-4 backdrop-blur-xl flex flex-col gap-3 shadow-[0_0_30px_rgba(0,0,0,0.45)]">
                <p className="text-white/80 text-sm">
                    {minPrice !== null ? (
                        <>
                            A partir de{" "}
                            <span className="text-white font-semibold">
                                {minPrice.toFixed(2)}€
                            </span>
                    ) : null}
                </p>
            </div>
        </div>
    );
}

```

```

        </>
    ) : (
      <span className="text-white/60">Sem bilhetes disponíveis</span>
    )}
</p>

<button
  type="button"
  disabled={purchasableTickets.length === 0}
  onClick={() => {
    if (purchasableTickets.length === 0) return;

    atualizarDados({
      slug,
      waves: visibleTickets,
      additional: {
        checkoutUiVariant,
      },
    });
  }}

  const defaultTicket = purchasableTickets[0];

  abrirCheckout({
    slug,
    ticketId: defaultTicket.id,
    price: defaultTicket.price,
    ticketName: defaultTicket.name,
    additional: {
      checkoutUiVariant,
    },
    waves: visibleTickets,
  });

  setTimeout(() => {
    try {
      const evt = new Event("orya-force-step1");
      window.dispatchEvent(evt);
    } catch {}
  }, 10);
}

  className="w-full rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold py-3 shadow-[0_0_20px_rgba(107,255,255,0.45)] hover:scale-[1.02] active:scale-95 transition-transform text-sm disabled:opacity-50 disabled:cursor-not-allowed"
>
  {purchasableTickets.length === 0 ? "Esgotado" : "Comprar agora"}
</button>
</div>
</div>
);
}

```

app/eventos/page.tsx

```

"use client";

import { useEffect, useState } from "react";
import Link from "next/link";

type EventCard = {
  id: number;
  slug: string;
  title: string;
  description: string | null;
  startsAt: string;
  endsAt: string | null;
  locationName: string | null;
  locationCity: string | null;
  isFree: boolean;
  priceFrom: number | null;
};

export default function EventosFeedPage() {
  const [events, setEvents] = useState<EventCard[]>([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState<string | null>(null);

```

```

const [search, setSearch] = useState("");

useEffect(() => {
  fetchEvents();
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, []);

async function fetchEvents() {
  setLoading(true);
  setError(null);

  try {
    const params = new URLSearchParams();
    if (search.trim()) params.set("q", search.trim());

    const res = await fetch(`api/eventos/list?${params.toString()}`);
    if (!res.ok) {
      throw new Error("Falha ao carregar eventos");
    }

    const data = await res.json();
    const items: EventCard[] = Array.isArray(data.events)
      ? data.events.map((ev: any) => ({
          id: ev.id,
          slug: ev.slug,
          title: ev.title,
          description: ev.shortDescription ?? ev.description ?? null,
          startsAt: ev.startDate ?? ev.startsAt ?? "",
          endsAt: ev.endDate ?? null,
          locationName: ev.venue?.name ?? ev.locationName ?? null,
          locationCity: ev.venue?.city ?? ev.locationCity ?? null,
          isFree: Boolean(ev.isFree),
          priceFrom: ev.priceFrom ?? null,
        }))
      : [];
    setEvents(items);
  } catch (err: unknown) {
    console.error(err);
    const message =
      err instanceof Error ? err.message : "Erro ao carregar eventos.";
    setError(message);
  } finally {
    setLoading(false);
  }
}

function handleSearchSubmit(e: React.FormEvent) {
  e.preventDefault();
  fetchEvents();
}

return (
  <main className="min-h-screen w-full bg-[radial-gradient(circle_at_top,_#140b2a_0,#050509_45%,#02020a_100%] text-white">
    {/* Top bar */}
    <header className="border-b border-white/10 bg-black/30 backdrop-blur-xl">
      <div className="max-w-6xl mx-auto px-10 md:px-10 py-4 flex items-center justify-between gap-4">
        <div className="flex items-center gap-3">
          <span className="inline-flex h-8 w-8 items-center justify-center rounded-xl bg-gradient-to-tr from-[#FF00C8] via-[#6BFFF] to-[#1646F5] text-xs font-extrabold tracking-[0.15em]">
            OR
          </span>
        <div>
          <p className="text-xs uppercase tracking-[0.2em] text-white/60">
            Explorar eventos
          </p>
          <p className="text-sm text-white/80">
            Descobre o que está a acontecer perto de ti (e onde devias
            estar).
          </p>
        </div>
      </div>
    </div>

    <Link
      href="/organizador/eventos/novo"
      className="hidden sm:inline-flex items-center gap-2 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFF] to-[#1646F5] px-4 py-1.5 text-xs font-semibold text-black shadow-[0_0_22px_rgba(107,255,255,0.45)] hover:scale-105 active:scale-95 transition">
    </Link>
  </main>
)

```

```

        + Criar evento
      </Link>
    </div>
</header>

<section className="max-w-6xl mx-auto px-6 md:px-10 py-8 md:py-10 space-y-6">
  /* Search */
  <div className="space-y-4">
    <form
      onSubmit={handleSearchSubmit}
      className="flex flex-col md:flex-row gap-3 md:items-center"
    >
      <div className="relative flex-1">
        <span className="pointer-events-none absolute left-3 top-1/2 -translate-y-1/2 text-xs text-white/40">
          <img alt="Search icon" />
        </span>
        <input
          value={search}
          onChange={(e) => setSearch(e.target.value)}
          placeholder="Pesquisar por evento, local ou vibe..." 
          className="w-full rounded-full bg-black/60 border border-white/15 pl-8 pr-24 py-2 text-xs outline-none
focus:border-[#6BFFF] focus:ring-1 focus:ring-[#6BFFF]/60 transition"
        />
        <button
          type="submit"
          className="absolute right-1 top-1/2 -translate-y-1/2 px-3 py-1 rounded-full bg-white/90 text-[11px] font-medium
text-black hover:bg-white"
        >
          Pesquisar
        </button>
      </div>
    </form>
  </div>
  {/* Estado de loading / erro */}
  {error && (
    <div className="rounded-xl border border-red-500/40 bg-red-500/10 px-4 py-3 text-xs text-red-100">
      {error}
    </div>
  )}
  {/* Grid de eventos */}
  <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4">
    {loading && !events.length ? (
      Array.from({ length: 6 }).map((_, i) => (
        <div
          key={i}
          className="h-64 rounded-2xl border border-white/10 bg-white/5 animate-pulse"
        /))
    ) : events.length === 0 ? (
      <p className="text-sm text-white/60 col-span-full">
        Ainda não há eventos. Tenta ajustar a pesquisa ou criar o primeiro
        evento.
      </p>
    ) : (
      events.map((ev) => (
        <Link
          key={ev.id}
          href={`/eventos/${ev.slug}`}
          className="group rounded-2xl border border-white/10 bg-gradient-to-b from-white/5 via-black/70 to-black/90
overflow-hidden shadow-[_18px_40px_rgba(0,0,0,0.7)] hover:border-[#6BFFF]/60 hover:shadow-[0_0_40px_rgba(107,255,255,0.35)]
transition"
        >
          <div className="relative h-32 overflow-hidden">
            <div className="h-full w-full bg-[radial-gradient(circle_at_top,_#FF00C8_0,_#02020a_65%)] flex items-center
justify-center text-xs text-white/60">
              ORYA • Evento
            </div>
            {ev.isFree && (
              <span className="absolute bottom-2 left-2 rounded-full bg-black/80 px-2 py-0.5 text-[10px] font-semibold
text-[#6BFFF] border border-[#6BFFF]/40">
                Evento gratuito
              </span>
            )}
          </div>
        <div className="p-3.5 space-y-2">

```

```

<div className="flex items-center justify-between gap-2">
  <p className="text-[11px] text-white/60">
    {formatDate(ev.startsAt)}
  </p>
  {ev.priceFrom !== null && !ev.isFree && (
    <p className="text-[11px] text-white">
      desde{" "}
      <span className="font-semibold">
        {ev.priceFrom} €{" "}
      </span>
    </p>
  )}
</div>

<h2 className="text-sm font-semibold line-clamp-2">
  {ev.title}
</h2>

<p className="text-[11px] text-white/60 line-clamp-2">
  {ev.description || "Sem descrição adicionada."}
</p>

<div className="flex items-center justify-between gap-2 mt-2">
  <div className="flex flex-col gap-0.5">
    <p className="text-[11px] text-white/70 line-clamp-1">
      {ev.locationName || "Local a definir"}
    </p>
    {ev.locationCity && (
      <p className="text-[10px] text-white/40 line-clamp-1">
        {ev.locationCity}
      </p>
    )}
  </div>
  </div>
</div>
</Link>
))
)
</div>
</section>
</main>
);
}

// Helpers
function formatDate(dateStr: string) {
  const d = new Date(dateStr);
  return d.toLocaleString("pt-PT", {
    weekday: "short",
    day: "2-digit",
    month: "short",
    hour: "2-digit",
    minute: "2-digit",
  });
}

```

app/experiencias/[slug]/page.tsx

```

"use client";

import { useEffect, useState } from "react";
import { useParams, useRouter } from "next/navigation";
import { useUser } from "@/app/hooks/useUser";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";

type ExperienceItem = {
  id: number;
  slug: string;
  title: string;
  description: string;
  startsAt: string;
  endsAt: string;
  locationName: string;
  locationCity: string;
  isFree: boolean;
  host?: {

```

```

    id: string;
    username: string;
    fullName: string;
    avatarUrl: string;
};

};

export default function ExperiencePage() {
  const params = useParams();
  const slug = params.slug as string;

  const { user, profile, isLoading } = useUser();
  const { openModal } = useAuthModal();

  const [experience, setExperience] = useState<ExperienceItem | null>(null);
  const [loading, setLoading] = useState(true);
  const [joining, setJoining] = useState(false);
  const [joinError, setJoinError] = useState<string | null>(null);
  const [joined, setJoined] = useState(false);

  async function checkJoined(eventId: number) {
    if (!user) return;
    try {
      const res = await fetch(`'/api/experiencias/join/status?eventId=${eventId}` , {
        cache: "no-store",
      });
      const json = await res.json();
      if (json.joined) setJoined(true);
    } catch (err) {
      console.error("Erro a verificar join:", err);
    }
  }

  async function loadExperience() {
    setLoading(true);
    setExperience(null);

    try {
      const res = await fetch(
        "/api/experiencias/list?slug=" + encodeURIComponent(slug) + "&limit=1",
      );
      const json = await res.json();
      setExperience(json.items?.[0] || null);
      if (json.items?.[0]?.id) {
        await checkJoined(json.items[0].id);
      }
    } catch {
      setExperience(null);
    } finally {
      setLoading(false);
    }
  }

  useEffect(() => {
    if (!slug) return;
    loadExperience();
  }, [slug]);
}

async function handleJoin() {
  if (!user) {
    openModal({ mode: "login", redirectTo: "/experiencias/" + slug, showGoogle: true });
    return;
  }
  if (!experience) return;

  setJoining(true);
  setJoinError(null);

  try {
    const res = await fetch("/api/experiencias/join", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ eventId: experience.id }),
    });
    if (!res.ok) {
      const data = await res.json();
      setJoinError(data?.error || "Erro ao juntar-se à experiência.");
    }
  } catch {
    setJoinError("Erro ao juntar-se à experiência.");
  }
}

```

```

    } else {
      setJoined(true);
      await loadExperience();
    }
  } catch {
    setJoinError("Erro ao juntar-se à experiência.");
  } finally {
    setJoining(false);
  }
}

if (loading) {
  return (
    <div className="container mx-auto max-w-2xl py-10">
      <p>Carregando...</p>
    </div>
  );
}

if (!experience) {
  return (
    <div className="container mx-auto max-w-2xl py-10">
      <p>Experiência não encontrada.</p>
    </div>
  );
}

return (
  <div className="container mx-auto max-w-2xl py-10">
    <h1 className="text-xl font-semibold">{experience.title}</h1>
    {experience.host?.username && (
      <p className="mt-2 text-sm text-gray-600">
        Criado por{" "}
        <a href={`/ ${experience.host.username}`}>{experience.host.username}</a>
      </p>
    )}
    <p className="mt-4">{experience.description}</p>
    <p className="mt-4 text-sm text-gray-700">
      Início: {new Date(experience.startsAt).toLocaleString()}
    </p>
    <p className="mt-1 text-sm text-gray-700">
      Local: {experience.locationName}, {experience.locationCity}
    </p>
    {experience.isFree && (
      <p className="mt-1 text-sm font-semibold text-green-600">Gratuita</p>
    )}
    <div className="mt-6">
      <button
        onClick={handleJoin}
        disabled={joining || joined}
        className={`${`px-4 py-2 rounded ${joined ? "bg-gray-400 cursor-not-allowed" : "bg-blue-600 text-white hover:bg-blue-700"}`}`}
      >
        {joined ? "Já vais" : joining ? "A juntar-me..." : "Juntar-me"}
      </button>
      {joinError && <p className="mt-2 text-red-600">{joinError}</p>}
    </div>
  </div>
);
}

```

app/experiencias/nova/page.tsx

```

"use client";

import { Suspense, useEffect, useState, FormEvent, useRef } from "react";
import { useRouter, useSearchParams } from "next/navigation";
import { useUser } from "@/app/hooks/useUser";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";
import { InlineDateTimePicker } from "@/app/components/forms/InlineDateTimePicker";

```

```

const TEMPLATE_TYPES = [
  { value: "PARTY", label: "Festa" },
  { value: "SPORT", label: "Desporto" },
  { value: "VOLUNTEERING", label: "Voluntariado" },
  { value: "TALK", label: "Palestra / Talk" },
  { value: "OTHER", label: "Outro" },
] as const;

const CATEGORY_OPTIONS = [
  { value: "FESTA", label: "Festa", accent: "from-[#FF00C8] to-[#FF8AD9]" },
  { value: "DESPORTO", label: "Desporto", accent: "from-[#6BFFFF] to-[#4ADE80]" },
  { value: "CONCERTO", label: "Concerto", accent: "from-[#9B8cff] to-[#6BFFFF]" },
  { value: "PALESTRA", label: "Palestra", accent: "from-[#FDE68A] to-[#F472B6]" },
  { value: "ARTE", label: "Arte", accent: "from-[#F472B6] to-[#A855F7]" },
  { value: "COMIDA", label: "Comida", accent: "from-[#F97316] to-[#FACC15]" },
  { value: "DRINKS", label: "Drinks", accent: "from-[#34D399] to-[#6BFFFF]" },
] as const;

type TemplateType = (typeof TEMPLATE_TYPES)[number]["value"];

type FormState = {
  title: string;
  description: string;
  date: string; // datetime-local
  endDate: string; // datetime-local opcional
  locationName: string;
  locationCity: string;
  templateType: TemplateType;
  address: string;
  categories: string[];
  coverUrl: string | null;
};

function NovaExperienciaContent() {
  const router = useRouter();
  const searchParams = useSearchParams();
  const { user, isLoading } = useUser();
  const { openModal } = useAuthModal();

  const [form, setForm] = useState<FormState>({
    title: "",
    description: "",
    date: "",
    endDate: "",
    locationName: "",
    locationCity: "",
    templateType: "OTHER",
    address: "",
    categories: [],
    coverUrl: null,
  });

  const [isSubmitting, setIsSubmitting] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [uploadingCover, setUploadingCover] = useState(false);

  // Evitar abrir o modal em loop
  const authGuardChecked = useRef(false);

  useEffect(() => {
    if (isLoading || authGuardChecked.current) return;

    if (!user) {
      openModal({
        mode: "login",
        redirectTo: "/experiencias/nova",
      });
    }
  }, [authGuardChecked.current = true, isLoading, user, openModal]);

  function handleChange<K extends keyof FormState>(field: K, value: FormState[K]) {
    setForm((prev) => ({ ...prev, [field]: value }));
  }

  const handleCoverUpload = async (file: File | null) => {
    if (!file) return;
  }
}

```

```

setUploadingCover(true);
setError(null);
try {
  const formData = new FormData();
  formData.append("file", file);
  const res = await fetch("/api/upload", {
    method: "POST",
    body: formData,
  });
  const json = await res.json();
  if (!res.ok || !json?.url) {
    throw new Error(json?.error || "Erro ao carregar imagem.");
  }
  setForm((prev) => ({ ...prev, coverUrl: json.url as string }));
} catch (err) {
  console.error("Erro no upload de capa:", err);
  setError("Não foi possível carregar a imagem de capa.");
} finally {
  setUploadingCover(false);
}
};

const handleSubmit = async (e: FormEvent) => {
  e.preventDefault();

  if (!user) {
    openModal({ mode: "login", redirectTo: "/experiencias/nova", showGoogle: true });
    return;
  }

  setIsSubmitting(true);
  setError(null);

  if (!form.categories.length) {
    setError("Escolhe pelo menos uma categoria.");
    setIsSubmitting(false);
    return;
  }

  try {
    const startsAt = form.date;
    const endsAt = form.endDate || undefined;

    const res = await fetch("/api/experiencias/create", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        title: form.title,
        description: form.description,
        startsAt,
        endsAt,
        locationName: form.locationName,
        locationCity: form.locationCity,
        templateType: form.templateType,
        address: form.address,
        categories: form.categories,
        coverImageUrl: form.coverUrl,
      }),
    });

    const json = await res.json();

    if (!res.ok || !json.ok) {
      setError(json.error || "Não foi possível criar a experiência.");
      setIsSubmitting(false);
      return;
    }

    const slug: string | undefined = json.event?.slug;
    if (slug) {
      router.push(`/experiencias/${slug}`);
    } else {
      router.push("/experiencias");
    }
  } catch (err) {
    console.error("Erro ao criar experiência:", err);
  }
};

```

```

        setError("Erro inesperado ao criar experiência.");
        setIsSubmitting(false);
    }

};

const redirectFromQuery = searchParams.get("redirectTo");

return (
    <div className="max-w-2xl mx-auto px-4 py-8">
        <h1 className="text-2xl font-semibold mb-2">Criar experiência</h1>
        <p className="text-sm text-neutral-500 mb-6">
            Cria um momento simples: um jogo de padel, um jantar, um café, uma caminhada, o que quiseres.
        </p>

        {redirectFromQuery && (
            <p className="text-xs text-neutral-500 mb-4">
                Depois de criar, vamos levar-te para: {redirectFromQuery}
            </p>
        )}

        {!isLoading && !user && (
            <div className="mb-4 rounded-md border border-orange-300 bg-orange-50 px-3 py-2 text-sm text-orange-800">
                Precisas de entrar na tua conta para crieares uma experiência.
            </div>
        )}
    </div>
)

<form onSubmit={handleSubmit} className="space-y-4">
    <div>
        <label className="block text-sm font-medium mb-1">Imagen de capa</label>
        <div className="flex gap-3 items-start">
            <div className="h-28 w-40 rounded-xl border border-neutral-700 bg-neutral-900/40 overflow-hidden flex items-center justify-center text-[11px] text-neutral-400">
                {form.coverUrl ? (
                    // eslint-disable-next-line @next/next/no-img-element
                    <img src={form.coverUrl} alt="Capa" className="h-full w-full object-cover" />
                ) : (
                    "Sem imagem"
                )}
            </div>
            <div className="space-y-2 text-[12px] text-neutral-300">
                <input
                    type="file"
                    accept="image/*"
                    onChange={(e) => handleCoverUpload(e.target.files?.[0] ?? null)}
                />
                <div className="flex items-center gap-2">
                    <button
                        type="button"
                        onClick={() => handleChange("coverUrl", null)}
                        className="rounded-full border border-neutral-600 px-3 py-1 text-xs hover:border-white"
                    >
                        Remover
                    </button>
                    {uploadingCover && <span className="text-xs text-neutral-400">A carregar...</span>}
                </div>
            </div>
        </div>
    </div>
    <div>
        <label className="block text-sm font-medium mb-1">Titolo *</label>
        <input
            type="text"
            className="w-full rounded-md border border-neutral-300 bg-neutral-900/20 px-3 py-2 text-sm outline-none focus:border-pink-500 focus:ring-1 focus:ring-pink-500"
            placeholder="Jogo de padel às 18h, Café e conversa, etc."
            value={form.title}
            onChange={(e) => handleChange("title", e.target.value)}
            required
        />
    </div>
    <div>
        <label className="block text-sm font-medium mb-1">Descrição</label>
        <textarea
            className="w-full rounded-md border border-neutral-300 bg-neutral-900/20 px-3 py-2 text-sm outline-none focus:border-pink-500 focus:ring-1 focus:ring-pink-500"

```

```

    rows={4}
    placeholder="Explica rapidamente o que vai acontecer, para quem é, nível, etc."
    value={form.description}
    onChange={(e) => handleChange("description", e.target.value)}
  />
</div>

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">
  <InlineDatePicker
    label="Data e hora de início *"
    value={form.date}
    onChange={(v) => handleChange("date", v)}
    minDateTime={new Date()}
    required
  />
  <InlineDatePicker
    label="Data e hora de fim (opcional)"
    value={form.endDate}
    onChange={(v) => handleChange("endDate", v)}
    minDateTime={form.date ? new Date(form.date) : new Date()}
  />
</div>

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">
  <div>
    <label className="block text-sm font-medium mb-1">Local *</label>
    <input
      type="text"
      className="w-full rounded-md border border-neutral-300 bg-neutral-900/20 px-3 py-2 text-sm outline-none
focus:border-pink-500 focus:ring-1 focus:ring-pink-500"
      placeholder="Nome do sítio (ex.: Parque da Cidade)"
      value={form.locationName}
      onChange={(e) => handleChange("locationName", e.target.value)}
      required
    />
  </div>
  <div>
    <label className="block text-sm font-medium mb-1">Cidade *</label>
    <input
      type="text"
      className="w-full rounded-md border border-neutral-300 bg-neutral-900/20 px-3 py-2 text-sm outline-none
focus:border-pink-500 focus:ring-1 focus:ring-pink-500"
      placeholder="Porto, Lisboa, Braga..."
      value={form.locationCity}
      onChange={(e) => handleChange("locationCity", e.target.value)}
      required
    />
  </div>
</div>

<div>
  <label className="block text-sm font-medium mb-1">
    Rua / morada (opcional)
  </label>
  <input
    type="text"
    className="w-full rounded-md border border-neutral-300 bg-neutral-900/20 px-3 py-2 text-sm outline-none
focus:border-pink-500 focus:ring-1 focus:ring-pink-500"
    placeholder="Ex.: Rua de exemplo, 123 (TODO: ligar a Mapbox Search)"
    value={form.address}
    onChange={(e) => handleChange("address", e.target.value)}
  />
</div>

<div>
  <label className="block text-sm font-medium mb-1">Tipo de experiência</label>
  <select
    className="w-full rounded-md border border-neutral-300 bg-neutral-900/20 px-3 py-2 text-sm outline-none
focus:border-pink-500 focus:ring-1 focus:ring-pink-500"
    value={form.templateType}
    onChange={(e) => handleChange("templateType", e.target.value as TemplateType)}
  >
    {TEMPLATE_TYPES.map((t) => (
      <option key={t.value} value={t.value}>
        {t.label}
      </option>
    )))
  </select>
</div>

```

```

        </select>
    </div>

    <div>
        <label className="block text-sm font-medium mb-1">
            Categorias (obrigatório)
        </label>
        <div className="grid grid-cols-2 sm:grid-cols-3 gap-2">
            {CATEGORY_OPTIONS.map((cat) => {
                const checked = form.categories.includes(cat.value);
                return (
                    <label
                        key={cat.value}
                        className={`inline-flex items-center gap-2 rounded-2xl border px-3 py-2 text-sm transition ${checked
                            ? "bg-white text-black border-white shadow-[0_0_18px_rgba(255,255,255,0.35)]"
                            : "bg-black/30 border-white/15 text-white"
                        }`}
                    >
                        <input
                            type="checkbox"
                            className="sr-only"
                            checked={checked}
                            onChange={(e) => {
                                const next = e.target.checked
                                ? [...form.categories, cat.value]
                                : form.categories.filter((c) => c !== cat.value);
                                handleChange("categories", next);
                            }}
                        />
                        <span className="flex items-center gap-2">
                            <span
                                className={`h-2 w-2 rounded-full bg-gradient-to-r ${cat.accent} shadow-[0_0_10px_rgba(255,255,255,0.4)]`}
                            />
                            {cat.label}
                        </span>
                    </label>
                );
            ))}
        </div>
        <p className="mt-1 text-xs text-neutral-400">
            Escolhe pelo menos uma categoria para ajudar na descoberta.
        </p>
    </div>

    {error && (
        <p className="text-sm text-red-500 mt-1">{error}</p>
    )}

    <button
        type="submit"
        disabled={isSubmitting}
        className="mt-4 inline-flex items-center justify-center rounded-md bg-pink-600 px-4 py-2 text-sm font-medium text-white hover:bg-pink-700 disabled:opacity-60"
    >
        {isSubmitting ? "A criar..." : "Criar experiência"}
    </button>
</form>
</div>
);
}

export default function NovaExperienciaPage() {
    return (
        <Suspense fallback={null}>
            <NovaExperienciaContent />
        </Suspense>
    );
}

```

app/experiencias/page.tsx

```
"use client";
```

```

import { useEffect, useState } from "react";
import Link from "next/link";
import { useRouter } from "next/navigation";
import { useUser } from "@/app/hooks/useUser";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";

type ExperienceListItem = {
  id: number;
  slug: string;
  title: string;
  description: string | null;
  startsAt: string | null;
  endsAt: string | null;
  locationName: string | null;
  locationCity: string | null;
  isFree: boolean;
  host: {
    id: string;
    username: string | null;
    fullName: string | null;
    avatarUrl: string | null;
  } | null;
};

function formatDateTime(value: string | null) {
  if (!value) return "Data a definir";
  const d = new Date(value);
  if (Number.isNaN(d.getTime())) return "Data a definir";
  return d.toLocaleString("pt-PT", {
    day: "2-digit",
    month: "short",
    hour: "2-digit",
    minute: "2-digit",
  });
}

export default function ExperiencesPage() {
  const [items, setItems] = useState<ExperienceListItem[]>([]);
  const [isLoadingList, setIsLoadingList] = useState(true);
  const [error, setError] = useState<string | null>(null);

  const { user, isLoading: isUserLoading } = useUser();
  const { openModal } = useAuthModal();
  const router = useRouter();

  useEffect(() => {
    let cancelled = false;

    async function load() {
      try {
        setIsLoadingList(true);
        setError(null);
        const res = await fetch("/api/experiencias/list");
        if (!res.ok) {
          throw new Error("Falha ao carregar experiências.");
        }
        const data = (await res.json()) as { items: ExperienceListItem[] };
        if (!cancelled) {
          setItems(data.items || []);
        }
      } catch (err) {
        if (!cancelled) {
          console.error("Falha a carregar experiências:", err);
          setError("Não foi possível carregar as experiências.");
        }
      } finally {
        if (!cancelled) {
          setIsLoadingList(false);
        }
      }
    }
    load();
  });

  return () => {
    cancelled = true;
  };
}, []);

```

```

function handleCreateClick() {
  if (isUserLoading) return;

  if (!user) {
    openModal({ mode: "login", redirectTo: "/experiencias/nova", showGoogle: true });
    return;
  }

  router.push("/experiencias/nova");
}

return (
  <div className="max-w-5xl mx-auto px-4 py-8 space-y-6">
    <div className="flex flex-col sm:flex-row sm:items-center sm:justify-between gap-4">
      <div>
        <h1 className="text-2xl font-semibold">Experiências</h1>
        <p className="text-sm text-neutral-500">
          Explora experiências criadas por outros utilizadores ORYA ou cria a tua.
        </p>
      </div>

      <button
        type="button"
        onClick={handleCreateClick}
        className="inline-flex items-center justify-center rounded-full px-4 py-2 text-sm font-medium bg-white/10 border border-white/20 hover:bg-white/20 transition"
      >
        Criar experiência
      </button>
    </div>

    {error && (
      <div className="rounded-lg border border-red-500/40 bg-red-500/10 px-4 py-3 text-sm text-red-100">
        {error}
      </div>
    )}

    {isLoadingList ? (
      <div className="text-sm text-neutral-400">A carregar experiências...</div>
    ) : items.length === 0 ? (
      <div className="text-sm text-neutral-400">
        Ainda não existem experiências. Sê o primeiro a criar uma.
      </div>
    ) : (
      <div className="grid gap-4 md:grid-cols-2">
        {items.map((exp) => (
          <Link
            key={exp.id}
            href={`/experiencias/${exp.slug}`}
            className="group rounded-xl border border-white/10 bg-white/5 px-4 py-3 hover:bg-white/10 transition flex flex-col gap-2"
          >
            <div className="flex items-center justify-between gap-2">
              <h2 className="text-base font-semibold line-clamp-1 group-hover:underline">
                {exp.title}
              </h2>
              <span className="text-[11px] px-2 py-1 rounded-full border border-white/20 text-neutral-200">
                {exp.isFree ? "Gratuita" : "Com custo"}
              </span>
            </div>

            <p className="text-xs text-neutral-400 line-clamp-2">
              {exp.description || "Sem descrição."}
            </p>

            <div className="mt-1 flex flex-wrap items-center gap-2 text-[11px] text-neutral-300">
              <span>{formatDateTime(exp.startsAt)}</span>
              {exp.locationName && (
                <span>• {exp.locationName}</span>
              )}
              {exp.locationCity && (
                <span>• {exp.locationCity}</span>
              )}
            </div>
          </Link>
        ))
      </div>
    )}
  </div>
)

```

```

        Criado por {" "}
        <span className="font-medium">@{exp.host.username} || "utilizador"</span>
      </div>
    )}
</Link>
))}
</div>
)}
</div>
);
}

```

app/explorar/page.tsx

```

"use client";

import { Suspense, useEffect, useMemo, useRef, useState } from "react";
import { useSearchParams } from "next/navigation";
import Link from "next/link";
import { useRouter } from "next/navigation";
import Image from "next/image";
import { defaultBlurDataURL, optimizeImageUrl } from "@/lib/image";
import { PORTUGAL_CITIES } from "@/config/cities";
import { clampWithGap } from "@/lib/filters";
import { trackEvent } from "@/lib/analytics";

type ExploreItem = {
  id: number;
  type: "EVENT" | "EXPERIENCE";
  slug: string;
  title: string;
  shortDescription: string | null;
  startsAt: string;
  endsAt: string;
  location: {
    name: string | null;
    city: string | null;
    lat: number | null;
    lng: number | null;
  };
  coverImageUrl: string | null;
  isFree: boolean;
  priceFrom: number | null;
  categories: string[];
  hostName: string | null;
  hostUsername: string | null;
  status: "ACTIVE" | "CANCELLED" | "PAST" | "DRAFT";
  isHighlighted: boolean;
};

type ApiResponse = {
  items: ExploreItem[];
  pagination: {
    nextCursor: number | null;
    hasMore: boolean;
  };
};

type DateFilter = "all" | "today" | "weekend" | "custom";
type TypeFilter = "all" | "event" | "experience";

const DATE_FILTER_OPTIONS = [
  { value: "all", label: "Todas as datas" },
  { value: "today", label: "Hoje" },
  { value: "weekend", label: "Este fim de semana" },
] as const;

const TYPE_OPTIONS: { value: TypeFilter; label: string }[] = [
  { value: "all", label: "Tudo" },
  { value: "event", label: "Eventos" },
  { value: "experience", label: "Experiências" },
];

const CATEGORY_OPTIONS = [
  { value: "DESPORTO", label: "Desporto", accent: "from-[#6BFFFF] to-[#4ADE80]" },
  { value: "GERAL", label: "Eventos gerais", accent: "from-[#FF00C8] via-[#9B8cff] to-[#1646F5]" },
];

```

```

] as const;

function formatDateRange(start: string, end: string) {
  const startDate = new Date(start);
  const endDate = new Date(end);

  const sameDay = startDate.toDateString() === endDate.toDateString();

  const base0pts: Intl.DateTimeFormatOptions = {
    weekday: "short",
    day: "2-digit",
    month: "short",
    hour: "2-digit",
    minute: "2-digit",
  };

  const startStr = startDate.toLocaleString("pt-PT", base0pts);

  if (sameDay) {
    const endTime = endDate.toLocaleTimeString("pt-PT", {
      hour: "2-digit",
      minute: "2-digit",
    });
    return `${startStr} · ${endTime}`;
  }

  const endStr = endDate.toLocaleString("pt-PT", base0pts);
  return `${startStr} → ${endStr}`;
}

function statusTag(status: ExploreItem["status"]) {
  if (status === "CANCELLED") return { text: "Cancelado", className: "text-red-200" };
  if (status === "PAST") return { text: "Já aconteceu", className: "text-white/55" };
  if (status === "DRAFT") return { text: "Rascunho", className: "text-white/60" };
  return { text: "Em breve", className: "text-[#6BFFFF]" };
}

function buildSlug(type: ExploreItem["type"], slug: string) {
  return type === "EXPERIENCE" ? `/experiencias/${slug}` : `/eventos/${slug}`;
}

function ExplorarContent() {
  const [items, setItems] = useState<ExploreItem[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  const [searchInput, setSearchInput] = useState("");
  const [search, setSearch] = useState("");

  const [dateFilter, setDateFilter] = useState<DateFilter>("all");
  const [customDate, setCustomDate] = useState("");
  const [typeFilter, setTypeFilter] = useState<TypeFilter>("all");
  const [selectedCategories, setSelectedCategories] = useState<string[]>([]);
  const [cityInput, setCityInput] = useState("");
  const [city, setCity] = useState("");
  const [citySearch, setCitySearch] = useState("");
  const [priceMin, setPriceMin] = useState(0);
  const [priceMax, setPriceMax] = useState(100); // 100 = "sem limite" (100+)
  const [filtersTick, setFiltersTick] = useState(0);
  const [isMobile, setIsMobile] = useState(false);
  const [calendarMonth, setCalendarMonth] = useState<Date>(() => {
    const today = new Date();
    today.setHours(0, 0, 0, 0);
    return today;
  });

  const [nextCursor, setNextCursor] = useState<number | null>(null);
  const [hasMore, setHasMore] = useState(false);
  const [isLoadingMore, setIsLoadingMore] = useState(false);

  const [isCityOpen, setIsCityOpen] = useState(false);
  const [isDateOpen, setIsDateOpen] = useState(false);
  const [isPriceOpen, setIsPriceOpen] = useState(false);
  const cityRef = useRef<HTMLDivElement | null>(null);
  const dateRef = useRef<HTMLDivElement | null>(null);
  const priceRef = useRef<HTMLDivElement | null>(null);

  const [likedItems, setLikedItems] = useState<number[]>([]);
}

```

```

const searchParams = useSearchParams();
const requestController = useRef<AbortController | null>(null);
const [hydratedFromParams, setHydratedFromParams] = useState(false);

// City via geolocation + Mapbox (opcional)
// Geolocation desativada para evitar preencher com valores inválidos

const effectiveMaxParam = priceMax >= 100 ? null : priceMax;
const filteredCities = useMemo(() => {
  const needle = citySearch.trim().toLowerCase();
  if (!needle) return PORTUGAL_CITIES;
  return PORTUGAL_CITIES.filter((c) => c.toLowerCase().includes(needle));
}, [citySearch]);

const hasActiveFilters = useMemo(
() =>
  search.trim().length > 0 ||
  dateFilter === "all" ||
  (dateFilter === "custom" && !customDate) ||
  typeFilter === "all" ||
  selectedCategories.length > 0 ||
  city.trim().length > 0 ||
  priceMin > 0 ||
  effectiveMaxParam !== null,
[city, customDate, dateFilter, effectiveMaxParam, priceMin, search, selectedCategories.length, typeFilter],
);

function toggleLike(id: number) {
  setLikedItems((prev) => (prev.includes(id) ? prev.filter((v) => v !== id) : [...prev, id]));
}

const CityPanel = () => (
<>
  {isMobile && (
    <div className="mb-2">
      <input
        value={citySearch}
        onChange={(e) => setCitySearch(e.target.value)}
        placeholder="Procurar cidade"
        className="w-full rounded-lg border border-white/15 bg-white/5 px-3 py-2 text-[12px] text-white placeholder:text-white/40 focus:border-white/35 focus:outline-none"
      />
    </div>
  )}
  <div className="max-h-52 overflow-auto space-y-1.5 pr-1">
    {filteredCities.map((c) => {
      const active = city === c;
      return (
        <button
          key={c}
          type="button"
          onClick={() => {
            setCity(c);
            setCityInput(c);
            setIsCityOpen(false);
          }}
          className={`flex w-full items-center justify-between rounded-lg border px-3 py-2 text-left text-[12px] ${active ? "border-[#6BFFFF]/70 bg-[#6BFFFF]/15 text-white" : "border-white/12 bg-white/5 text-white/80 hover:border-white/30"}`}
        >
          {c}
          {active && <span className="text-[10px] text-[#E5FFFF]">Selecionada</span>}
        </button>
      );
    ))}
  </div>
  <div className="flex items-center justify-between pt-2">
    <button
      type="button"
      onClick={() => setIsCityOpen(false)}
      className="text-[11px] text-white/70 hover:text-white/90"
    >
      Fechar
    </button>
    <button
      type="button"
    >

```

```

    onClick={() => {
      setCityInput("");
      setCity("");
      setIsCityOpen(false);
    }}
    className="px-2.5 py-1 rounded-full bg-transparent border border-white/15 text-[11px] text-white/70 hover:bg-white/5"
  >
  Limpar cidade
</button>
</div>
</>
);
};

const calendarDays = useMemo(() => {
  const firstOfMonth = new Date(calendarMonth.getFullYear(), calendarMonth.getMonth(), 1);
  const startWeekday = firstOfMonth.getDay(); // domingo = 0
  const daysInMonth = new Date(calendarMonth.getFullYear(), calendarMonth.getMonth() + 1, 0).getDate();
  const list: Array<{ date: Date | null }> = [];
  for (let i = 0; i < startWeekday; i++) {
    list.push({ date: null });
  }
  for (let d = 1; d <= daysInMonth; d++) {
    list.push({ date: new Date(calendarMonth.getFullYear(), calendarMonth.getMonth(), d) });
  }
  return list;
}, [calendarMonth]);

const monthLabel = useMemo(
() =>
  calendarMonth.toLocaleString("pt-PT", {
    month: "long",
    year: "numeric",
  }),
  [calendarMonth],
);

const DatePanel = () => (
<>
  <p className="text-[11px] text-white/60 mb-1.5">Quando queres sair?</p>
  <div className="flex flex-wrap gap-1.5">
    {DATE_FILTER_OPTIONS.map((opt) => (
      <button
        key={opt.value}
        type="button"
        onClick={() => {
          setCustomDate("");
          setDateFilter(opt.value as DateFilter);
        }}
        className={`px-2.5 py-1 rounded-full text-[11px] ${dateFilter === opt.value && !customDate
          ? "bg-[#6BFFFF]/25 border border-[#6BFFFF]/70 text-[#EFFFFF]"
          : "bg-white/5 border border-white/18 text-white/75 hover:bg-white/10"}`}
      >
        {opt.label}
      </button>
    )));
  </div>
<div className="mt-3 rounded-2xl border border-white/12 bg-white/5 p-3 shadow-[_16px_40px_rgba(0,0,0,0.4)]">
  <div className="mb-2 flex items-center justify-between text-[12px] text-white/80">
    <button
      type="button"
      onClick={() => {
        const prev = new Date(calendarMonth);
        prev.setMonth(prev.getMonth() - 1);
        setCalendarMonth(prev);
      }}
      className="rounded-full px-2 py-1 hover:bg-white/10"
    >
      ←
    </button>
    <span className="font-semibold capitalize">{monthLabel}</span>
    <button
      type="button"
      onClick={() => {
        const next = new Date(calendarMonth);
        next.setMonth(next.getMonth() + 1);
        setCalendarMonth(next);
      }}
    >

```

```

        }
        className="rounded-full px-2 py-1 hover:bg-white/10"
    >
    -->
</button>
</div>
<div className="mb-1 grid grid-cols-7 gap-1 text-[10px] text-white/45">
  ["D", "S", "T", "Q", "O", "S", "S"].map((d, idx) => (
    <span key={`${d}-${idx}`} className="text-center uppercase tracking-wide">
      {d}
    </span>
  )));
</div>
<div className="grid grid-cols-7 gap-1 text-[12px]">
  {calendarDays.map((d, idx) => {
    if (!d.date) return <span key={`blank-${idx}`}> />;
    const iso = d.date.toISOString().slice(0, 10);
    const isSelected = customDate ? iso === customDate : false;
    return (
      <button
        key={iso}
        type="button"
        onClick={() => {
          setCustomDate(iso);
          setDateFilter("custom");
        }}
        className={`${'h-9 w-9 rounded-full transition ${
          isSelected
            ? "bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold shadow-[0_0_16px_rgba(107,255,255,0.55)]"
            : "text-white/80 hover:bg-white/10"
        }}`}
      >
        {d.date.getDate()}
      </button>
    );
  )});
</div>
<p className="mt-2 text-[10px] text-white/45">Sem input livre: escolhe pelo calendário ou chips.</p>
</div>
<div className="flex items-center justify-between pt-1">
  <button
    type="button"
    onClick={() => setIsDateOpen(false)}
    className="text-[11px] text-white/70 hover:text-white/90"
  >
    Fechar
  </button>
  <button
    type="button"
    onClick={() => {
      setCustomDate("");
      setDateFilter("all");
      setIsDateOpen(false);
    }}
    className="px-2.5 py-1 rounded-full bg-transparent border border-white/15 text-[11px] text-white/70 hover:bg-white/5"
  >
    Limpar datas
  </button>
</div>
</>
);

const PricePanel = () => (
  <>
    <p className="text-[11px] text-white/60">Intervalo de preço</p>
    <div className="rounded-2xl border border-white/10 bg-gradient-to-br from-[#0b162c] via-[#0a1227] to-[#060b18] px-3 py-4 space-y-3 shadow-[0_12px_40px_rgba(0,0,0,0.5)]">
      <DoubleRange
        min={0}
        max={100}
        step={5}
        valueMin={priceMin}
        valueMax={priceMax}
        onChange={({min, max}) => {
          setPriceMin(min);
          setPriceMax(max);
        }}
      >
    </div>
  </>
);

```

```

        />
      <div className="flex items-center justify-between text-[11px] text-white/80">
        <span>Min: € ${priceMin}</span>
        <span>Máx: ${priceMax} >= 100 ? "100+ (sem limite)" : `€ ${priceMax}`</span>
      </div>
      <div className="flex items-center justify-between text-[11px] text-white/60">
        <button
          type="button"
          onClick={() => {
            setPriceMin(0);
            setPriceMax(100);
          }}
          className="rounded-full border border-white/15 px-3 py-1 text-white/75 hover:border-white/35 hover:bg-white/5"
        >
          Limpar filtro de preço
        </button>
        <span className="text-[10px] text-white/50">Debounce 250ms</span>
      </div>
    </div>
    <button
      type="button"
      onClick={() => {
        setPriceMin(0);
        setPriceMax(100);
        setIsPriceOpen(false);
      }}
      className="text-[11px] text-white/60 hover:text-white/90"
    >
      Limpar filtro de preço
    </button>
  </>
);

async function fetchItems(opts?: { append?: boolean; cursor?: number | null }) {
  const append = opts?.append ?? false;
  const cursorToUse = opts?.cursor ?? null;

  // Cancelar pedidos anteriores para evitar estados inconsistentes
  if (requestController.current) {
    requestController.current.abort();
  }
  const controller = new AbortController();
  requestController.current = controller;
  const timeoutId = setTimeout(() => controller.abort(), 4500);
  const currentRequest = controller;

  try {
    if (!append) {
      setLoading(true);
      setError(null);
    } else {
      setIsLoadingMore(true);
    }

    const params = new URLSearchParams();
    if (search.trim()) params.set("q", search.trim());
    if (dateFilter === "custom" && customDate) {
      params.set("date", "day");
      params.set("day", customDate);
    } else if (dateFilter !== "all") {
      params.set("date", dateFilter);
    }
    if (typeFilter !== "all") params.set("type", typeFilter);
    if (selectedCategories.length > 0) params.set("categories", selectedCategories.join(","));
    if (city.trim()) params.set("city", city.trim());
    if (priceMin > 0) params.set("priceMin", String(priceMin));
    if (effectiveMaxParam !== null) params.set("priceMax", String(effectiveMaxParam));
    if (cursorToUse !== null) params.set("cursor", String(cursorToUse));

    const res = await fetch(`~/api/explorar/list?${params.toString()}`, {
      cache: "no-store",
      signal: controller.signal,
    });
    if (!res.ok) {
      const text = await res.text();
      if (process.env.NODE_ENV !== "production") {
        console.error("Erro a carregar explorar:", text);
      } else {
        ...
      }
    }
  } catch (error) {
    ...
  }
}

```

```

        console.warn("Erro a carregar explorar");
    }
    throw new Error("Erro ao carregar explorar");
}

const data: ApiResponse = await res.json();
const nextItems = Array.isArray(data?.items) ? data.items : [];

if (requestController.current === currentRequest) {
    if (append) {
        setItems((prev) => [...(Array.isArray(prev) ? prev : []), ...nextItems]);
    } else {
        setItems(nextItems);
    }

    setNextCursor(data.pagination.nextCursor);
    setHasMore(data.pagination.hasMore);
}
} catch (err) {
    if (requestController.current !== currentRequest) return;
    const isAbort = (err as Error | undefined)?.name === "AbortError";
    if (!isAbort && process.env.NODE_ENV !== "production") {
        console.error(err);
    }
    setError(
        isAbort
            ? "Demorou demasiado a responder. Tenta novamente."
            : "Não conseguimos carregar. Tenta outra vez.",
    );
} finally {
    clearTimeout(timeoutId);
    if (requestController.current === currentRequest) {
        setLoading(false);
        setIsLoadingMore(false);
    }
}
}

useEffect(() => {
    const handle = setTimeout(() => setFiltersTick((v) => v + 1), 250);
    return () => clearTimeout(handle);
}, [search, dateFilter, customDate, typeFilter, selectedCategories, city, priceMin, effectiveMaxParam]);

useEffect(() => {
    trackEvent("explore_filter_price_changed", {
        min: priceMin,
        max: effectiveMaxParam ?? "100_plus",
    });
}, [priceMin, effectiveMaxParam]);

useEffect(() => {
    trackEvent("explore_filter_date_changed", {
        dateFilter,
        customDate: customDate || null,
    });
}, [dateFilter, customDate]);

useEffect(() => {
    if (!city && !cityInput) return;
    trackEvent("explore_filter_location_changed", { city: city || cityInput });
}, [city, cityInput]);

useEffect(() => {
    fetchItems({ append: false, cursor: null });
    // eslint-disable-next-line react-hooks/exhaustive-deps
}, [filtersTick]);

useEffect(() => {
    const handle = setTimeout(() => setSearch(searchInput.trim()), 350);
    return () => clearTimeout(handle);
}, [searchInput]);

useEffect(() => {
    if (!customDate) return;
    const parsed = new Date(customDate);
    if (Number.isNaN(parsed.getTime())) return;
    parsed.setHours(0, 0, 0, 0);
    setCalendarMonth(parsed);
}
)

```

```

}, [customDate]);

useEffect(() => {
  const updateIsMobile = () => setIsMobile(typeof window !== "undefined" ? window.innerWidth < 640 : false);
  updateIsMobile();
  window.addEventListener("resize", updateIsMobile);
  return () => window.removeEventListener("resize", updateIsMobile);
}, []);

useEffect(() => {
  const handle = setTimeout(() => {
    const normalized = cityInput.trim();
    if (!normalized) {
      setCity("");
      return;
    }
    const match = PORTUGAL_CITIES.find((c) => c.toLowerCase() === normalized.toLowerCase());
    setCity(match ?? "");
  }, 300);
  return () => clearTimeout(handle);
}, [cityInput]);

// Atualiza pesquisa ao entrar via query da Navbar (/explorar?query=)
useEffect(() => {
  if (hydratedFromParams) return;
  const qp = searchParams.get("query") ?? searchParams.get("q") ?? "";
  const cityQ = searchParams.get("city") ?? "";
  const priceMinQ = searchParams.get("priceMin");
  const priceMaxQ = searchParams.get("priceMax");
  const dateQ = searchParams.get("date");
  const dayQ = searchParams.get("day");
  const typeQ = searchParams.get("type") as TypeFilter | null;
  const catsQ = searchParams.get("categories");

  if (qp) {
    setSearchInput(qp);
    setSearch(qp);
  }
  if (cityQ) {
    setCityInput(cityQ);
    setCity(cityQ);
  }
  if (priceMinQ) setPriceMin(Math.max(0, Number(priceMinQ)));
  if (priceMaxQ) {
    const maxVal = Math.max(0, Number(priceMaxQ));
    setPriceMax(Number.isFinite(maxVal) ? maxVal : 100);
  }
  if (dateQ === "today" || dateQ === "upcoming" || dateQ === "weekend") {
    setDateFilter(dateQ);
  } else if (dateQ === "day" && dayQ) {
    setDateFilter("custom");
    setCustomDate(dayQ);
  }
  if (typeQ === "event" || typeQ === "experience") {
    setTypeFilter(typeQ);
  }
  if (catsQ) {
    const arr = catsQ
      .split(",")
      .map((c) => c.trim())
      .filter(Boolean);
    setSelectedCategories(arr);
  }
  setHydratedFromParams(true);
}, [searchParams, hydratedFromParams]);

useEffect(() => {
  const handle = setTimeout(() => setCity(cityInput.trim()), 350);
  return () => clearTimeout(handle);
}, [cityInput]);

useEffect(() => {
  const handleClickOutside = (event: MouseEvent | TouchEvent) => {
    const target = event.target as Node | null;
    if (!target) return;
    if (isCityOpen && cityRef.current && !cityRef.current.contains(target)) {
      setIsCityOpen(false);
    }
  }
}, []);

```

```

    }
    if (isDateOpen && dateRef.current && !dateRef.current.contains(target)) {
      setIsDateOpen(false);
    }
    if (isPriceOpen && priceRef.current && !priceRef.current.contains(target)) {
      setIsPriceOpen(false);
    }
  };
  document.addEventListener("mousedown", handleClickOutside);
  document.addEventListener("touchstart", handleClickOutside);
  return () => {
    document.removeEventListener("mousedown", handleClickOutside);
    document.removeEventListener("touchstart", handleClickOutside);
  };
}, [isCityOpen, isDateOpen, isPriceOpen]);

const headingCity = city.trim() || "Portugal";
const dateLabel =
  dateFilter === "custom" && customDate
    ? new Date(customDate).toLocaleDateString("pt-PT", { day: "2-digit", month: "short" })
    : DATE_FILTER_OPTIONS.find((d) => d.value === dateFilter)?.label;
const resultsLabel = items.length === 1 ? "1 resultado" : `${items.length} resultados`;
const showSkeleton = loading || (error && items.length === 0);

return (
  <main className="orya-body-bg min-h-screen w-full text-white">
    <section className="max-w-6xl mx-auto px-6 md:px-10 py-6 md:py-8 space-y-6">
      {/* TOPO - FILTROS PRINCIPAIS */}
      <div className="flex flex-col gap-4">
        <div className="flex flex-wrap items-center gap-3">
          {/* Localização */}
          <div className="relative" ref={cityRef}>
            <button
              type="button"
              onClick={() => {
                setIsCityOpen((v) => !v);
                setIsDateOpen(false);
                setIsPriceOpen(false);
              }}
            >
              {headingCity}
            </button>
            {isCityOpen &&
              <>
                {isMobile &&
                  <div className="fixed inset-0 z-50 bg-black/80 backdrop-blur-sm flex justify-center p-4">
                    <div className="w-full max-w-sm rounded-2xl border border-white/15 bg-black/90 p-4 space-y-3 shadow-2xl">
                      <div className="flex items-center justify-between">
                        <h3 className="text-sm font-semibold text-white">Escolhe cidade</h3>
                        <button
                          type="button"
                          onClick={() => setIsCityOpen(false)}
                          className="text-white/60 hover:text-white"
                        >
                          ×
                        </button>
                      </div>
                      <CityPanel />
                    </div>
                  </div>
                }
                {!isMobile &&
                  <div className="mt-2 w-full rounded-2xl border border-white/15 bg-black/90 p-3 backdrop-blur md: absolute
md:w-72 md:shadow-2xl md:z-50">
                    <CityPanel />
                  </div>
                }
              </>
            }
          </div>
        </div>
      </div>
      {/* Data */}
      <div className="relative" ref={dateRef}>
        <button
          type="button"

```

```

    onClick={() => {
      setIsDateOpen((v) => !v);
      setIsCityOpen(false);
      setIsPriceOpen(false);
    }}
    className="inline-flex items-center gap-2 rounded-2xl border border-white/18 bg-black/40 px-3.5 py-2 text-xs
md:text-sm hover:border-[#6BFFFF]/70 hover:bg-black/60 transition"
  >
  <span className="text-base">setDataLabel</span>
  <span className="font-medium">{dateLabel}</span>
</button>
{isDateOpen && (
  <>
    {isMobile && (
      <div className="fixed inset-0 z-50 bg-black/80 backdrop-blur-sm flex justify-center p-4">
        <div className="w-full max-w-sm rounded-2xl border border-white/15 bg-black/90 p-4 space-y-3 shadow-2xl">
          <DatePanel />
        </div>
      </div>
    )}
    {!isMobile && (
      <div className="mt-2 w-full rounded-2xl border border-white/15 bg-black/90 p-3 backdrop-blur space-y-3
md:absolute md:w-72 md:shadow-2xl md:z-50">
        <DatePanel />
      </div>
    )}
  </>
)}
</div>

/* Preço */
<div className="relative" ref={priceRef}>
  <button
    type="button"
    onClick={() => {
      setIsPriceOpen((v) => !v);
      setIsCityOpen(false);
      setIsDateOpen(false);
    }}
    className="inline-flex items-center gap-2 rounded-2xl border border-white/18 bg-black/40 px-3.5 py-2 text-xs
md:text-sm hover:border-[#6BFFFF]/70 hover:bg-black/60 transition"
  >
  <span className="text-base">setPriceRef</span>
  <span className="font-medium">
    {priceMin === 0 && effectiveMaxParam === null
      ? "Qualquer preço"
      : `${priceMin} - ${effectiveMaxParam === null ? "100+" : effectiveMaxParam}`}
  </span>
  </button>
{isPriceOpen && (
  <>
    {isMobile && (
      <div className="fixed inset-0 z-50 bg-black/80 backdrop-blur-sm flex justify-center p-4">
        <div className="w-full max-w-sm rounded-2xl border border-white/15 bg-black/90 p-4 space-y-3 shadow-2xl">
          <PricePanel />
        </div>
      </div>
    )}
    {!isMobile && (
      <div className="mt-2 w-full rounded-2xl border border-white/15 bg-black/90 p-3 backdrop-blur space-y-3
md:absolute md:w-80 md:shadow-2xl md:z-50">
        <PricePanel />
      </div>
    )}
  </>
)}
</div>

/* Tipo */
<div className="flex flex-wrap items-center gap-2 ml-auto text-[11px]">
  <div className="flex items-center gap-1 rounded-full bg-white/5 border border-white/15 px-1 py-1">
    {TYPE_OPTIONS.map((opt) => {
      const isActive = typeFilter === opt.value;
      return (
        <button
          key={opt.value}
          type="button"
          onClick={() => setTypeFilter(opt.value)}
        >

```

```

        className={`px-3 py-1 rounded-full transition ${
          isActive
            ? "bg-white text-black shadow-[0_0_15px_rgba(255,255,255,0.35)]"
            : "text-white/75 hover:bg-white/10"
        }`}
      >
      {opt.label}
    </button>
  );
}
);
</div>

{hasActiveFilters && (
  <button
    type="button"
    onClick={() => {
      setSearch("");
      setSearchInput("");
      setDateFilter("all");
      setTypeFilter("all");
      setSelectedCategories([]);
      setCity("");
      setCityInput("");
      setPriceMin(0);
      setPriceMax(100);
    }}
    className="text-[11px] text-white/55 hover:text-white/90"
  >
    Limpar tudo
  </button>
)
)
</div>
</div>

/* categorias */
<div className="flex w-full items-center gap-2 overflow-x-auto pb-1">
  <span className="text-[10px] text-white/50 shrink-0">Categorias:</span>
  <div className="flex gap-1.5">
    {CATEGORY_OPTIONS.map((cat) => {
      const isActive = selectedCategories.includes(cat.value);
      return (
        <button
          key={cat.value}
          type="button"
          onClick={() => {
            setSelectedCategories((prev) =>
              prev.includes(cat.value)
                ? prev.filter((c) => c !== cat.value)
                : [...prev, cat.value],
            );
          }}
        >
          className={`inline-flex items-center gap-1.5 rounded-2xl border px-3 py-1.5 text-[11px] whitespace nowrap
transition ${

          isActive
            ? "bg-white text-black border-white shadow-[0_0_18px_rgba(255,255,255,0.35)]"
            : "bg-white/5 border-white/18 text-white/80 hover:bg-white/10"
        }`}
          aria-pressed={isActive}
          aria-label={`${cat.label} - categoria`}
        >
          <span
            className={`${`h-2 w-2 rounded-full bg-gradient-to-r ${cat.accent} shadow-[0_0_12px_rgba(255,255,255,0.45)]`}`}
          />
          <span>{cat.label}</span>
        </button>
      );
    ))}
  </div>
</div>

/* resumo */
<div className="flex items-center justify-between text-[11px] text-white/60">
  <span>
    Encontrados{" "}
    <span className="font-semibold text-white/90">
      {resultsLabel}
    </span>
  </span>
</div>

```

```

  <span className="text-white/45">
    Eventos e experiências em{" "}
    <span className="text-white/85">{headingCity}</span>
  </span>
</div>
</div>

/* LOADING */
{showSkeleton && (
  <div className="mt-4 grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4" aria-hidden>
    {Array.from({ length: 8 }).map(_, i) => (
      <div
        key={i}
        className="rounded-3xl border border-white/10 bg-white/5 p-3 animate-pulse space-y-3"
      >
        <div className="rounded-2xl bg-white/10 aspect-square" />
        <div className="h-3 w-3/4 rounded bg-white/10" />
        <div className="h-3 w-1/2 rounded bg-white/8" />
      </div>
    )))
  </div>
)}

/* ERRO */
{error && (
  <div className="mt-4 max-w-xl mx-auto rounded-xl border border-red-500/30 bg-red-500/10 px-4 py-3 text-sm text-red-100 shadow-[0_12px_30px_rgba(0,0,0,0.55)]">
    <p className="font-semibold mb-1">Não foi possível carregar.</p>
    <p className="text-[12px] text-red-50/85 leading-relaxed">{error}</p>
    <div className="mt-3 flex gap-2">
      <button
        type="button"
        onClick={() => fetchItems({ append: false, cursor: null })}
        className="rounded-full bg-white text-red-700 px-4 py-1.5 text-[11px] font-semibold shadow hover:bg-white/90 transition"
      >
        Tentar novamente
      </button>
      <button
        type="button"
        onClick={() => {
          setSearch("");
          setSearchInput("");
          setDateFilter("all");
          setTypeFilter("all");
          setSelectedCategories([]);
          setCity("");
          setCityInput("");
          setPriceMin(0);
          setPriceMax(100);
          fetchItems({ append: false, cursor: null });
        }}
        className="rounded-full border border-white/25 text-white/85 px-4 py-1.5 text-[11px] hover:bg-white/5 transition"
      >
        Limpar filtros
      </button>
    </div>
  </div>
)}

/* SEM RESULTADOS */
{!loading && !error && items.length === 0 && (
  <div className="mt-10 flex flex-col items-center text-center gap-2 text-sm text-white/60">
    <p>Não encontrámos eventos ou experiências com estes filtros.</p>
    <p className="text-xs text-white/40 max-w-sm">
      Ajusta a cidade, categorias ou preço - ou volta mais tarde. A cidade está sempre a
      mexer.
    </p>
    <button
      type="button"
      onClick={() => {
        setSearch("");
        setSearchInput("");
        setDateFilter("all");
        setTypeFilter("all");
        setSelectedCategories([]);
        setCity("");
      }}
      className="rounded-full border border-white/25 text-white/85 px-4 py-1.5 text-[11px] hover:bg-white/5 transition"
    >
      Limpar filtros
    </button>
  </div>
)
}

```

```

        setCityInput("");
        setPriceMin(0);
        setPriceMax(100);
    )}
    className="mt-2 px-4 py-1.5 rounded-full bg-white/5 border border-white/20 text-xs text-white/80 hover:bg-white/10"
white/10"
    >
    Limpar filtros e voltar ao inicio
    </button>
</div>
)}

/* LISTA */
{!loading && items.length > 0 && (
<>
<div className="mt-4 grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-5">
    {items.map((item) =>
        item.type === "EVENT" ? (
            <EventCard
                key={`${item.type}-${item.id}`}
                item={item}
                onLike={toggleLike}
                liked={likedItems.includes(item.id)}
            />
        ) : (
            <ExperienceCard
                key={`${item.type}-${item.id}`}
                item={item}
                onLike={toggleLike}
                liked={likedItems.includes(item.id)}
            />
        ),
    )
)
</div>

{hasMore && (
    <div className="mt-6 flex justify-center">
        <button
            type="button"
            onClick={() => fetchItems({ append: true, cursor: nextCursor })}
            disabled={isLoadingMore}
            className="px-5 py-2 rounded-full bg-white/5 border border-white/20 text-xs text-white/80 hover:bg-white/10
disabled:opacity-60"
            >
            {isLoadingMore ? "A carregar mais..." : "Ver mais"}
        </button>
    </div>
)
)
</section>
</main>
);
}

export default function ExplorarPage() {
    return (
        <Suspense fallback={null}>
            <ExplorarContent />
        </Suspense>
    );
}

type CardProps = {
    item: ExploreItem;
    liked: boolean;
    onLike: (id: number) => void;
    neonClass?: string;
};

function PriceBadge({ item }: { item: ExploreItem }) {
    if (item.isFree) return <span className="text-emerald-200">Grátis</span>;
    if (item.priceFrom !== null) return <span>Desde {item.priceFrom.toFixed(2)} €</span>;
    return <span>Preço a anunciar</span>;
}

type DoubleRangeProps = {
    min: number;
}

```

```

max: number;
step: number;
valueMin: number;
valueMax: number;
onChange: (min: number, max: number) => void;
};

// Estilo inline para evitar que clicar no track move os thumbs
const doubleRangeStyles = `
.price-range-thumb::-webkit-slider-runnable-track { pointer-events: none; }
.price-range-thumb::-moz-range-track { pointer-events: none; }
.price-range-thumb::-ms-track { pointer-events: none; }
.price-range-thumb::-webkit-slider-thumb {
  pointer-events: auto;
  height: 18px;
  width: 18px;
  border-radius: 50%;
  background: radial-gradient(circle at 30% 30%, #E5FFFF, #6BFFFF 55%, #1646F5 100%);
  box-shadow: 0 0 12px rgba(107,255,255,0.6);
  border: 2px solid rgba(255,255,255,0.6);
  margin-top: -8px;
}
.price-range-thumb::-moz-range-thumb {
  pointer-events: auto;
  height: 18px;
  width: 18px;
  border-radius: 50%;
  background: radial-gradient(circle at 30% 30%, #E5FFFF, #6BFFFF 55%, #1646F5 100%);
  box-shadow: 0 0 12px rgba(107,255,255,0.6);
  border: 2px solid rgba(255,255,255,0.6);
}
.price-range-thumb::-ms-thumb {
  pointer-events: auto;
  height: 18px;
  width: 18px;
  border-radius: 50%;
  background: radial-gradient(circle at 30% 30%, #E5FFFF, #6BFFFF 55%, #1646F5 100%);
  box-shadow: 0 0 12px rgba(107,255,255,0.6);
  border: 2px solid rgba(255,255,255,0.6);
}
`;

if (typeof document !== "undefined") {
  const existing = document.getElementById("double-range-styles");
  if (!existing) {
    const style = document.createElement("style");
    style.id = "double-range-styles";
    style.innerHTML = doubleRangeStyles;
    document.head.appendChild(style);
  }
}

function DoubleRange({ min, max, step, valueMin, valueMax, onChange }: DoubleRangeProps) {
  const gap = 5;
  const bounds = { min, max };

  return (
    <div className="space-y-3">
      <div className="relative h-2 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#9B8cff] to-[#6BFFFF]">
        <div
          className="absolute h-full rounded-full bg-black/30"
          style={{
            left: `${(valueMin / max) * 100}%`,
            right: `${100 - (valueMax / max) * 100}%`,
          }}
        />
        <input
          type="range"
          min={min}
          max={max}
          step={step}
          value={valueMin}
          onChange={(e) => {
            const next = Number(e.target.value);
            const { min: cMin, max: cMax } = clampWithGap(next, valueMax, step, gap, bounds);
            onChange(cMin, cMax);
          }}
        />
      </div>
    </div>
  );
}

```

```

    className="price-range-thumb pointer-events-auto absolute -top-3 h-7 w-full appearance-none bg-transparent z-20"
    style={{ touchAction: "none" }}
  />
<input
  type="range"
  min={min + gap}
  max={max}
  step={step}
  value={valueMax}
  onChange={(e) => {
    const next = Number(e.target.value);
    const { min: cMin, max: cMax } = clampWithGap(valueMin, next, step, gap, bounds);
    onChange(cMin, cMax);
  }}
  className="price-range-thumb pointer-events-auto absolute -top-3 h-7 w-full appearance-none bg-transparent z-30"
  style={{ touchAction: "none" }}
/>
</div>
<div className="flex justify-between text-[10px] text-white/60">
  <span>{min}</span>
  <span>{max}</span>
</div>
</div>
);
}
}

function BaseCard({
  item,
  liked,
  onLike,
  accentClass,
  badge,
  neonClass,
}: CardProps & { accentClass: string; badge: string }) {
  const router = useRouter();
  const status = statusTag(item.status);
  const dateLabel = formatDateRange(item.startsAt, item.endsAt);
  const venueLabel = item.location.name || item.location.city || "Local a anunciar";
  const isEvent = badge === "Evento";
  const badgeGrad = isEvent
    ? "from-white/12 via-white/9 to-white/6"
    : "from-white/10 via-white/8 to-white/5";
  const badgeDot = isEvent
    ? "from-[#FF66E0]/70 via-[#8DEFFF]/70 to-[#5270FF]/70"
    : "from-[#6EE7FF]/70 via-[#34d399]/70 to-[#3b82f6]/70";
  const badgeIcon = isEvent ? "💡" : "📅";
  return (
    <Link href={buildSlug(item.type, item.slug)}
      className={`group rounded-3xl border border-white/10 bg-white/[0.02] overflow-hidden flex flex-col transition-all
      hover:border-white/16 hover:-translate-y-[6px] ${neonClass ?? ""}`}
    >
      <div className="relative overflow-hidden">
        <div className="aspect-square w-full">
          {item.coverImageUrl ? (
            <Image
              src={optimizeImageUrl(item.coverImageUrl, 900, 72)}
              alt={item.title}
              fill
              sizes="(max-width: 640px) 100vw, (max-width: 1024px) 50vw, 33vw"
              className="object-cover transform transition-transform duration-300 group-hover:scale-[1.04]"
              placeholder="blur"
              blurDataURL={defaultBlurDataURL}
            />
          ) : (
            <div className={`h-full w-full bg-gradient-to-br ${accentClass}`} />
          )}
        </div>
        <div
          className={`absolute top-2 left-2 flex items-center gap-2 rounded-2xl border border-white/16 px-3 py-1 text-[11px]
          font-semibold text-white/90 backdrop-blur-lg bg-gradient-to-r ${badgeGrad}`}
        >
          <span className="flex h-5 w-5 items-center justify-center rounded-full bg-white/10 text-[11px]">
            {badgeIcon}
          </span>
          <span>

```

```

    className={`${`h-1.5 w-6 rounded-full bg-gradient-to-r ${badgeDot}`}`}
  />
  <span className="tracking-wide leading-none">{badge}</span>
</div>

<button
  type="button"
  onClick={(e) => {
    e.preventDefault();
    e.stopPropagation();
    onLike(item.id);
  }}
  className="absolute top-2 right-2 flex h-9 w-9 items-center justify-center rounded-full bg-black/60 border border-white/30 shadow-[0_0_15px_rgba(0,0,0,0.6)] text-base opacity-0 group-hover:opacity-100 group-hover:scale-105 transition-all hover:bg-black/80"
  aria-label={liked ? "Remover interesse" : "Marcar interesse"}
>
  <span
    className={`${`transition-transform duration-150 ${{
      liked ? "scale-110 text-[#FF00C8]" : "scale-100 text-white"
    }}`}`}
  >
    {liked ? "♥" : "♡"}
  </span>
</button>

<div className="pointer-events-none absolute inset-x-0 bottom-0 h-16 bg-gradient-to-t from-black/70 via-black/20 to-transparent" />
</div>

<div className="p-3 flex flex-col gap-1.5">
  <div className="flex items-center justify-between text-[11px] text-white/75">
    {item.hostUsername ? (
      <button
        type="button"
        onClick={(e) => {
          e.preventDefault();
          e.stopPropagation();
          router.push(`/${item.hostUsername}`);
        }}
        className="truncate text-left hover:text-[#6BFFFF]"
      >
        {item.hostName || `@$ {item.hostUsername}`}
      </button>
    ) : (
      <span className="truncate">{item.hostName || "Organizador ORYA"}</span>
    )}
    <span className="rounded-full bg-white/5 px-2 py-0.5 border border-white/10">
      <PriceBadge item={item} />
    </span>
  </div>
</div>

<h2 className="text-[14px] md:text-[15px] font-semibold leading-snug text-white line-clamp-2">
  {item.title}
</h2>

<p className="text-[11px] text-white/80 line-clamp-2">{dateLabel}</p>
<p className="text-[11px] text-white/70">{venueLabel}</p>

<div className="flex flex-wrap gap-1.5 mt-2">
  {item.categories.map((c) => {
    const catLabel = CATEGORY_OPTIONS.find((opt) => opt.value === c)?.label ?? c;
    return (
      <span
        key={c}
        className="text-[10px] rounded-full bg-white/5 border border-white/10 px-2 py-0.5 text-white/75"
      >
        {catLabel}
      </span>
    );
  ))}
</div>

<div className="mt-2 flex items-center justify-between text-[11px]">
  <span className="px-2 py-0.5 rounded-full bg-black/75 border border-white/22 text-white font-medium">
    {item.isFree ? "Entrada gratuita" : "Bilhetes disponíveis"}
  </span>
  <span className={status.className}>{status.text}</span>
</div>

```

```

        </div>
        </div>
    </Link>
);
}

function EventCard(props: CardProps) {
    return (
        <BaseCard
            {...props}
            accentClass="from-[#FF00C8]/45 via-[#6BFFFF]/25 to-[#1646F5]/45"
            badge="Evento"
            neonClass="shadow-[0_14px_32px_rgba(0,0,0,0.45)]"
        />
);
}

function ExperienceCard(props: CardProps) {
    return (
        <BaseCard
            {...props}
            accentClass="from-[#22d3ee]/45 via-[#34d399]/25 to-[#0ea5e9]/45"
            badge="Experiência"
            neonClass="shadow-[0_14px_32px_rgba(0,0,0,0.42)]"
        />
);
}

```

app/globals.css

```

@import "tailwindcss";

/* ===== CORES BASE ===== */
:root {
    --background: #05060a;
    --foreground: #f9fafb;

    --orya-pink: #ff00c8;
    --orya-cyan: #6bffff;
    --orya-blue: #1646f5;
    --orya-purple: #7300ff;

    --glass-bg: rgba(255, 255, 255, 0.06);
    --glass-border: rgba(255, 255, 255, 0.1);
    --glass-border-strong: rgba(255, 255, 255, 0.18);
    --text-strong: rgba(255, 255, 255, 0.92);
    --text-muted: rgba(255, 255, 255, 0.62);
    --divider: rgba(255, 255, 255, 0.08);

    /* Z-index tokens */
    --z-footer: 60;
    --z-popover: 120;
    --z-modal: 200;
}

@media (prefers-color-scheme: dark) {
    :root {
        --background: #05060a;
        --foreground: #f9fafb;
    }
}

body {
    background: var(--background);
    color: var(--foreground);
    font-family: system-ui, -apple-system, BlinkMacSystemFont, "SF Pro Text",
        "SF Pro Display", -apple-system, sans-serif;
}

/* Remover padding top do main quando a navbar pública está escondida (organizador) */
body[data-nav-hidden="true"] .main-shell {
    padding-top: 0 !important;
    padding-bottom: 0;
}

/* Padding default do main quando a navbar está visível (páginas públicas) */

```

```

body:not([data-nav-hidden="true"]) .main-shell {
  padding-top: 4rem;
  padding-bottom: 120px;
}

/* Scrollbars discretos (checkout stripe) */
.payment-scroll {
  scrollbar-width: thin;
  scrollbar-color: rgba(255, 255, 255, 0.18) transparent;
}
.payment-scroll::-webkit-scrollbar {
  width: 6px;
}
.payment-scroll::-webkit-scrollbar-track {
  background: transparent;
}
.payment-scroll::-webkit-scrollbar-thumb {
  background: rgba(255, 255, 255, 0.18);
  border-radius: 9999px;
}
.payment-scroll::-webkit-scrollbar-thumb:hover {
  background: rgba(255, 255, 255, 0.28);
}

/* ===== ORYA LOGO ===== */

/* Glow subtil à volta do anel */
.orya-logo-glow {
  width: 240px;
  height: 240px;
  border-radius: 50%;
  filter: blur(35px);
  opacity: 0.55;
  background: conic-gradient(
    var(--orya-cyan),
    #45e5ff,
    var(--orya-purple),
    var(--orya-pink),
    var(--orya-cyan)
  );
  mask-image: radial-gradient(circle, transparent 58%, black 74%);
  -webkit-mask-image: radial-gradient(circle, transparent 58%, black 74%);
}

/* Canvas do anel */
.orya-logo-canvas {
  background: transparent !important;
  pointer-events: none;
}

/* ===== ANIMAÇÃO DO ANEL ORYA (rotação devagar e cinematográfica) ===== */
@keyframes orya-spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

.animate-orya-spin-slow {
  animation: orya-spin 7s linear infinite;
}

/* ===== ORYA - React Datepicker dark neon theme ===== */

.orya-datepicker-popper {
  z-index: 40;
}

/* Container principal do calendário */
.orya-datepicker-calendar.react-datepicker {
  background: radial-gradient(
    circle at top,
    #17152b 0,

```

```

        #05051a 45%,
        #040315 100%
    );
    border-radius: 16px;
    border: 1px solid rgba(255, 255, 255, 0.14);
    box-shadow: 0 18px 45px rgba(0, 0, 0, 0.75);
    color: #f9fafb;
    overflow: hidden;
    font-size: 11px;
}

/* Header (m  s, setas, etc.) */
.orya-datepicker-calendar .react-datepicker__header {
    background: linear-gradient(
        135deg,
        rgba(255, 0, 200, 0.18),
        rgba(107, 255, 255, 0.12)
    );
    border-bottom: 1px solid rgba(255, 255, 255, 0.16);
}

.orya-datepicker-calendar .react-datepicker__current-month,
.orya-datepicker-calendar .react-datepicker-time__header,
.orya-datepicker-calendar .react-datepicker-year__header {
    color: #f9fafb;
    font-size: 12px;
}

/* Dias e labels */
.orya-datepicker-calendar .react-datepicker__day-name,
.orya-datepicker-calendar .react-datepicker__day,
.orya-datepicker-calendar .react-datepicker__time-name {
    color: rgba(249, 250, 251, 0.85);
}

/* Hover */
.orya-datepicker-calendar .react-datepicker__day:hover,
.orya-datepicker-calendar .react-datepicker__time-list-item:hover {
    background: rgba(107, 255, 255, 0.16);
    color: #f9fafb;
}

/* Dia selecionado */
.orya-datepicker-calendar .react-datepicker__day--selected,
.orya-datepicker-calendar .react-datepicker__day--keyboard-selected {
    background: linear-gradient(135deg, #ff00c8, #6bffff);
    color: #020617;
}

/* Hoje */
.orya-datepicker-calendar .react-datepicker__day--today {
    border-radius: 9999px;
    border: 1px solid rgba(107, 255, 255, 0.8);
}

/* Coluna de horas */
.orya-datepicker-calendar .react-datepicker__time-container {
    border-left: 1px solid rgba(255, 255, 255, 0.12);
}

.orya-datepicker-calendar .react-datepicker__time-list {
    background: rgba(1, 3, 17, 0.95);
}

.orya-datepicker-calendar .react-datepicker__time-list-item {
    padding: 4px 6px;
}

/* Input wrapper */
.orya-datepicker-input.react-datepicker-wrapper,
.orya-datepicker-input .react-datepicker__input-container input {
    width: 100%;
}

.orya-datepicker-input::placeholder,
.orya-datepicker-input .react-datepicker__input-container input::placeholder {
    color: rgba(148, 163, 184, 0.6);
}

```

```

/* ===== FUNDO GLOBAL ORYA - neutro + detalhe ===== */
.orya-body-bg {
  min-height: 100vh;
  color: #f9fafb;
  background-color: #050814;
  /* Camadas grandes e suaves, pensadas para o hero + scroll */
  background-image:
    radial-gradient(1200px 800px at 50% -200px, rgba(107, 255, 255, 0.18), transparent 60%),
    radial-gradient(900px 700px at -220px 45%, rgba(255, 0, 200, 0.22), transparent 65%),
    radial-gradient(1000px 750px at 115% 65%, rgba(22, 70, 245, 0.25), transparent 70%),
    linear-gradient(180deg, #050814 0%, #05040f 45%, #020308 100%);
  background-attachment: fixed;
  background-repeat: no-repeat;
  background-size: cover;
  position: relative;
  overflow-x: hidden;
}

/* Overlay com "textura" leve (grid + glow suave) */
.orya-body-bg::before {
  content: "";
  position: fixed;
  inset: -40px;
  pointer-events: none;
  z-index: -1;
  background-image:
    radial-gradient(circle at 0 0, rgba(255, 255, 255, 0.08) 0, transparent 50%),
    radial-gradient(circle at 100% 100%, rgba(255, 255, 255, 0.04) 0, transparent 55%),
    repeating-linear-gradient(
      90deg,
      rgba(255, 255, 255, 0.03) 0,
      rgba(255, 255, 255, 0.03) 1px,
      transparent 1px,
      transparent 14px
    ),
    repeating-linear-gradient(
      0deg,
      rgba(255, 255, 255, 0.02) 0,
      rgba(255, 255, 255, 0.02) 1px,
      transparent 1px,
      transparent 14px
    );
  opacity: 0.42;
  mix-blend-mode: soft-light;
}

@keyframes subtle-fade-slide {
  0% {
    opacity: 0;
    transform: translateY(8px);
  }
  100% {
    opacity: 1;
    transform: translateY(0);
  }
}

.animate-fade-slide {
  animation: subtle-fade-slide 0.24s ease;
}

@keyframes step-pop {
  0% {
    opacity: 0;
    transform: scale(0.97);
  }
  100% {
    opacity: 1;
    transform: scale(1);
  }
}

.animate-step-pop {
  animation: step-pop 0.22s ease;
}

```

```

@keyframes wizardInRight {
  from {
    opacity: 0;
    transform: translateX(10px);
  }
  to {
    opacity: 1;
    transform: translateX(0);
  }
}

@keyframes wizardInLeft {
  from {
    opacity: 0;
    transform: translateX(-10px);
  }
  to {
    opacity: 1;
    transform: translateX(0);
  }
}

.wizard-step-in-right {
  animation: wizardInRight 230ms cubic-bezier(0.2, 0.8, 0.2, 1);
}

.wizard-step-in-left {
  animation: wizardInLeft 230ms cubic-bezier(0.2, 0.8, 0.2, 1);
}

.btn-orya {
  position: relative;
  border-radius: 9999px;
  padding: 12px 18px;
  background: rgba(255, 255, 255, 0.06);
  border: 1px solid rgba(255, 255, 255, 0.14);
  backdrop-filter: blur(12px);
  color: rgba(255, 255, 255, 0.92);
  transition: transform 0.15s ease, box-shadow 0.2s ease, border-color 0.2s ease;
}

.btn-orya::before {
  content: "";
  position: absolute;
  inset: -1px;
  border-radius: 9999px;
  padding: 1px;
  background: linear-gradient(90deg, #7300ff, #1646f5, #6bffff, #ff00c8);
  -webkit-mask: linear-gradient(#000 0 0) content-box, linear-gradient(#000 0 0);
  -webkit-mask-composite: xor;
  mask-composite: exclude;
  opacity: 0.55;
}

.btn-orya:hover {
  transform: translateY(-1px);
  box-shadow: 0 0 0 1px rgba(255, 255, 255, 0.1), 0 0 32px rgba(107, 255, 255, 0.14);
  border-color: rgba(255, 255, 255, 0.22);
}

.btn-orya:active {
  transform: translateY(1px);
  box-shadow: 0 0 0 1px rgba(255, 255, 255, 0.08), 0 0 18px rgba(107, 255, 255, 0.12);
}

.btn-orya:disabled {
  opacity: 0.45;
  cursor: not-allowed;
  transform: none;
  box-shadow: none;
}

.btn-ghost {
  border: 1px solid var(--glass-border);
  background: rgba(255, 255, 255, 0.04);
  color: var(--text-strong);
  border-radius: 9999px;
  padding: 10px 14px;
}

```

```

    transition: border-color 0.15s ease, background 0.15s ease, transform 0.12s ease;
}

.btn-ghost:hover {
  border-color: var(--glass-border-strong);
  background: rgba(255, 255, 255, 0.06);
  transform: translateY(-1px);
}

.btn-ghost:disabled {
  opacity: 0.55;
  cursor: not-allowed;
  transform: none;
}

.btn-chip {
  border: 1px solid var(--glass-border);
  background: rgba(255, 255, 255, 0.04);
  color: var(--text-strong);
  border-radius: 9999px;
  padding: 6px 10px;
  font-size: 12px;
  transition: border-color 0.15s ease, background 0.15s ease;
}

.btn-chip:hover {
  border-color: var(--glass-border-strong);
  background: rgba(255, 255, 255, 0.08);
}

@keyframes popover-in {
  from {
    opacity: 0;
    transform: translateY(4px) scale(0.98);
  }
  to {
    opacity: 1;
    transform: translateY(0) scale(1);
  }
}

.animate-popover {
  animation: popover-in 0.18s ease-out;
}

```

app/hooks/useAuthRequired.ts

```

"use client";

import { useEffect, useRef, useState } from "react";
import useSWR from "swr";
import { supabaseBrowser } from "@/lib/supabaseBrowser";

type Role = "user" | "organizer" | "admin" | string;

type ApiMeResponse = {
  user: {
    id: string;
    email: string | null;
    emailConfirmed?: boolean;
    emailConfirmedAt?: string | null;
  } | null;
  profile: {
    id: string;
    username: string | null;
    fullName: string | null;
    avatarUrl: string | null;
    bio: string | null;
    city: string | null;
    favouriteCategories: string[];
  }
}

```

```

onboardingDone: boolean;
roles: Role[];
visibility: string;
allowEmailNotifications: boolean;
allowEventReminders: boolean;
allowFriendRequests: boolean;
profileVisibility: "PUBLIC" | "PRIVATE";
} | null;
};

const fetcher = async (url: string) => {
  const res = await fetch(url, { credentials: "include", cache: "no-store" });
  if (res.status === 401) {
    // Sem sessão válida → devolve user/profile null sem erro
    return { user: null, profile: null };
  }
  if (!res.ok) {
    throw new Error("Falha ao carregar user");
  }
  return (await res.json()) as ApiMeResponse;
};

export function useUser() {
  const { data, error, isLoading, mutate } = useSWR<ApiMeResponse>(
    "/api/auth/me",
    fetcher
  );
  const [migrated, setMigrated] = useState(false);
  const migratedRef = useRef(false);

  // Forçar refresh quando o estado de auth muda (sign in/out) para evitar estado preso
  useEffect(() => {
    const {
      data: { subscription },
    } = supabaseBrowser.auth.onAuthStateChange(() => {
      mutate();
    });
    return () => {
      subscription.unsubscribe();
    };
  }, [migrate]);

  // Migrar bilhetes de guest após login (best-effort)
  useEffect(() => {
    const migrate = async () => {
      try {
        await fetch("/api/tickets/migrate-guest", { method: "POST" });
      } catch (err) {
        console.warn("[useUser] migrate-guest falhou", err);
      }
    };
    if (data?.user && !migratedRef.current) {
      migratedRef.current = true;
      migrate();
    }
  }, [data?.user]);

  return {
    user: data?.user ?? null,
    profile: data?.profile ?? null,
    roles: data?.profile?.roles ?? [],
    isLoading,
    isLoggedIn: !!data?.user,
    error,
    mutate,
    refetch: mutate,
  };
}

```

app/layout.tsx

```

import type { Metadata } from "next";
import "./globals.css";
import { Navbar } from "./components/Navbar";
import { AuthModalProvider } from "./components/autenticação/AuthModalContext";
import AuthModal from "./components/autenticação/AuthModal";

```

```

export const metadata: Metadata = {
  title: "ORYA",
  description: "O centro da tua vida social em Portugal.",
};

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="pt-PT" className="h-full suppressHydrationWarning>
      <body
        className="orya-body-bg antialiased min-h-screen flex flex-col font-sans"
      >
        <AuthModalProvider>
          <Navbar />
          <main className="main-shell flex-1">
            {children}
          </main>
          <AuthModal />
        </AuthModalProvider>
      </body>
    </html>
  );
}

```

app/live/[id]/monitor/page.tsx

```

// app/live/[id]/monitor/page.tsx
import { notFound } from "next/navigation";

type PageProps = { params: Promise<{ id: string }> };

async function getStructure(id: number) {
  const res = await fetch(`#${process.env.NEXT_PUBLIC_SITE_URL || ""}/api/tournaments/${id}/monitor`, {
    cache: "force-cache",
    next: { revalidate: 10 },
  });
  if (!res.ok) return null;
  return res.json();
}

export default async function MonitorPage({ params }: PageProps) {
  const resolved = await params;
  const tournamentId = Number(resolved.id);
  if (!Number.isFinite(tournamentId)) notFound();

  const data = await getStructure(tournamentId);
  if (!data?.ok) notFound();

  const tournament = data.tournament;
  const matches = tournament.stages.flatMap((s: any) => [...s.matches, ...s.groups.flatMap((g: any) => g.matches)]);
  const now = new Date();

  const upcoming = matches
    .filter((m: any) => m.startAt && new Date(m.startAt) > now && m.status !== "DONE")
    .sort((a: any, b: any) => new Date(a.startAt).getTime() - new Date(b.startAt).getTime())
    .slice(0, 8);
  const live = matches.filter((m: any) => m.status === "IN_PROGRESS").slice(0, 6);

  return (
    <div className="min-h-screen bg-black text-white p-4 space-y-4">
      <div className="flex items-center justify-between">
        <div>
          <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Monitor</p>
          <h1 className="text-2xl font-semibold">{tournament?.event?.title ?? "Torneio"}</h1>
          <p className="text-white/60 text-sm">Formato: {tournament.format}</p>
        </div>
        <p className="text-white/60 text-sm">Atualiza a cada ~10s</p>
      </div>
      <div className="grid gap-3 md:grid-cols-2">
        <section className="rounded-2xl border border-white/10 bg-white/5 p-3 space-y-2">
          <div className="flex items-center justify-between">

```

```

<h2 className="text-lg font-semibold">Em curso</h2>
<span className="text-white/60 text-sm">{live.length} jogo(s)</span>
</div>
{live.length === 0 && <p className="text-white/60 text-sm">Sem jogos em curso.</p>}
{live.map((m: any) => (
  <div key={m.id} className="rounded-lg border border-green-400/30 bg-green-500/10 px-3 py-2 space-y-1">
    <div className="flex items-center justify-between">
      <span className="text-white">#{m.id} · {m.pairing1Id ?? "?"} vs {m.pairing2Id ?? "?"}</span>
      <span className="text-[11px] text-white/70">{mStatusLabel}</span>
    </div>
    <p className="text-[11px] text-white/60">Court {m.courtId ?? "-"} · {m.startAt ? new Date(m.startAt).toLocaleTimeString("pt-PT") : "-"}</p>
  </div>
))
)}
</section>

<section className="rounded-2xl border border-white/10 bg-white/5 p-3 space-y-2">
  <div className="flex items-center justify-between">
    <h2 className="text-lg font-semibold">Próximos</h2>
    <span className="text-white/60 text-sm">{upcoming.length} agendados</span>
  </div>
  {upcoming.length === 0 && <p className="text-white/60 text-sm">Sem jogos agendados.</p>}
  {upcoming.map((m: any) => (
    <div key={m.id} className="rounded-lg border border-white/10 bg-black/30 px-3 py-2 space-y-1">
      <div className="flex items-center justify-between">
        <span className="text-white">#{m.id} · {m.pairing1Id ?? "?"} vs {m.pairing2Id ?? "?"}</span>
        <span className="text-[11px] text-white/70">{mStatusLabel}</span>
      </div>
      <p className="text-[11px] text-white/60">Court {m.courtId ?? "-"} · {m.startAt ? new Date(m.startAt).toLocaleTimeString("pt-PT") : "-"}</p>
    </div>
  )))
}
</section>
</div>

<section className="rounded-2xl border border-white/10 bg-white/5 p-3 space-y-2">
  <div className="flex items-center justify-between">
    <h2 className="text-lg font-semibold">Chave / Grupos</h2>
    <span className="text-white/60 text-sm">Resumo</span>
  </div>
  <div className="grid gap-3 md:grid-cols-2 lg:grid-cols-3">
    {tournament.stages.map((stage: any) => (
      <div key={stage.id} className="rounded-xl border border-white/10 bg-black/30 p-2 space-y-1">
        <p className="text-white font-semibold">{stage.name || stage.stageType}</p>
        {stage.groups.map((g: any) => (
          <div key={g.id} className="rounded-lg border border-white/10 bg-white/5 p-2 space-y-1 text-[12px] text-white/80">
            <p className="text-white text-sm">{g.name}</p>
            <p className="text-white/60 text-[11px]">Jogos: {g.matches.length}</p>
            {g.standings?.length ? (
              <div className="space-y-1">
                {g.standings.slice(0, 4).map((s: any, idx: number) => (
                  <div key={s.pairingId} className="flex items-center justify-between rounded border border-white/10 bg-white/5 px-2 py-1">
                    <span className="text-white">{idx + 1}º · Dupla # {s.pairingId}</span>
                    <span className="text-white/60">V {s.wins} · Sets {s.setDiff}</span>
                  </div>
                )))
              )
            ) : (
              <p className="text-white/60">Sem standings.</p>
            )}
          </div>
        )))
      <div>
        {stage.matches.length > 0 && (
          <div className="rounded-lg border border-white/10 bg-white/5 p-2 text-[12px] text-white/80 space-y-1">
            <p className="text-white text-sm">Playoffs</p>
            {stage.matches.slice(0, 4).map((m: any) => (
              <div key={m.id} className="flex items-center justify-between rounded border border-white/10 bg-black/20 px-2 py-1">
                <span className="text-white">#{m.id} · {m.pairing1Id ?? "?"} vs {m.pairing2Id ?? "?"}</span>
                <span className="text-white/60 text-[11px]">{mStatusLabel}</span>
              </div>
            )))
          )
        )}
      </div>
    )))
  </div>
</section>

```

```

        </div>
    </section>
</div>
);
}
}


```

app/login/page.tsx

```

"use client";
import { Suspense, useEffect, useState } from "react";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";
import { useSearchParams, useRouter } from "next/navigation";
import { supabaseBrowser } from "@/lib/supabaseBrowser";
import { PendingDeleteBanner } from "./pending-delete-banner";

function LoginContent() {
    const { openModal, isOpen } = useAuthModal();
    const searchParams = useSearchParams();
    const router = useRouter();
    const redirectTo = searchParams.get("redirectTo") ?? "/";
    const [checked, setChecked] = useState(false);
    const [openedOnce, setOpenedOnce] = useState(false);
    const [fallback, setFallback] = useState<string>("/");
    const [mounted, setMounted] = useState(false);
    const [pendingEmail, setPendingEmail] = useState<string | null>(null);

    useEffect(() => {
        // eslint-disable-next-line react-hooks/set-state-in-effect
        setMounted(true);
        if (typeof window !== "undefined") {
            const pending = window.localStorage.getItem("orya_pending_email");
            if (pending) setPendingEmail(pending);
        }
    }, []);

    useEffect(() => {
        // Escolhe melhor fallback: query ? redirectTo : same-origin referrer : "/"
        if (typeof window !== "undefined") {
            const ref = document.referrer;
            const sameOriginRef =
                ref && new URL(ref).origin === window.location.origin
                    ? new URL(ref).pathname + new URL(ref).search
                    : null;
            // eslint-disable-next-line react-hooks/set-state-in-effect
            setFallback(redirectTo || sameOriginRef || "/");
        }
    }, [redirectTo]);

    useEffect(() => {
        let cancelled = false;
        async function checkSession() {
            const { data, error } = await supabaseBrowser.auth.getUser();
            if (cancelled) return;
            if (!error && data?.user) {
                router.replace(redirectTo);
                return;
            }
            if (!openedOnce && !isOpen) {
                if (pendingEmail) {
                    openModal({ mode: "verify", email: pendingEmail, redirectTo });
                } else {
                    openModal({ mode: "login", redirectTo, showGoogle: true });
                }
                setOpenedOnce(true);
            }
            setChecked(true);
        }
        checkSession();
        return () => {
            cancelled = true;
        };
    }, [openModal, redirectTo, router, isOpen, openedOnce, pendingEmail]);

    // Se o modal foi aberto e entretanto está fechado, redireciona para o fallback
    useEffect(() => {
        if (openedOnce && !isOpen) {

```

```

        router.replace(fallback);
    }
}, [openedOnce, isOpen, router, fallback]);

if (!mounted) {
    return null;
}

return (
    <main className="min-h-screen flex items-center justify-center text-white">
        <div className="w-full max-w-md text-center space-y-4 px-4">
            <PendingDeleteBanner />
            <p>(checked ? "Se o modal não abriu," : "A preparar sessão...")</p>
            {checked && (
                <div className="flex items-center justify-center gap-3">
                    <button
                        onClick={() =>
                            openModal({ mode: "login", redirectTo, showGoogle: true })
                        }
                        className="underline"
                    >
                        abrir modal
                    </button>
                    <button
                        onClick={() => router.replace(redirectTo)}
                        className="underline"
                    >
                        sair e voltar
                    </button>
                </div>
            )}
        </div>
    </main>
);
}

export default function LoginRedirectPage() {
    return (
        <Suspense
            fallback={
                <main className="min-h-screen flex items-center justify-center text-white">
                    <p className="text-sm text-white/60">A preparar sessão...</p>
                </main>
            }
        >
            <LoginContent />
        </Suspense>
    );
}
}

```

app/login/pending-delete-banner.tsx

```

export function PendingDeleteBanner() {
    const params = new URLSearchParams(typeof window !== "undefined" ? window.location.search : "");
    const pending = params.get("pending_delete");
    if (!pending) return null;
    return (
        <div className="mb-4 rounded-xl border border-amber-400/40 bg-amber-500/15 px-4 py-3 text-sm text-amber-100">
            A tua conta está marcada para eliminação. Faz login para a reativar dentro do prazo.
        </div>
    );
}

```

app/mapa/page.tsx

```

"use client";

export default function MapaPage() {
    return (
        <main className="orya-body-bg min-h-screen text-white flex items-center justify-center px-6 py-10">
            <div className="rounded-3xl border border-white/15 bg-white/5 px-6 py-4 shadow">
                <h1 className="text-lg font-semibold">Mapa</h1>
                <p className="text-sm text-white/70">Em breve vais poder explorar eventos no mapa.</p>
            </div>
        </main>
    );
}

```

```
</main>
);
}
```

app/me/carteira/page.tsx

```
"use client";

import { useEffect, useMemo, useState } from "react";

type TicketItem = {
  id: string;
  badge: string;
  paymentStatusLabel?: string;
  nextAction?: string;
  event: { title: string; slug: string };
  isTournament?: boolean;
  liveLink?: string | null;
};

type InscricaoItem = {
  id: string;
  badge: string;
  paymentStatusLabel?: string;
  nextAction?: string;
  event: { title: string; slug: string } | null;
  isCaptain: boolean;
  ctaUrl?: string | null;
  liveLink?: string | null;
};

type FilterKey = "ALL" | "TICKETS" | "TOURNAMENTS" | "FREE" | "RESALE";

export default function CarteiraPage() {
  const [tickets, setTickets] = useState<TicketItem[]>([]);
  const [inscricoes, setInscricoes] = useState<InscricaoItem[]>([]);
  const [filter, setFilter] = useState<FilterKey>("ALL");
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    // Best-effort claim automático (se email verificado)
    fetch("/api/email/verified", { method: "POST" }).catch(() => undefined);

    const load = async () => {
      setLoading(true);
      try {
        const [tRes, iRes] = await Promise.all([fetch("/api/me/tickets"), fetch("/api/me/inscricoes")]);
        const tJson = await tRes.json();
        const iJson = await iRes.json();
        if (tJson?.tickets) setTickets(tJson.tickets);
        if (iJson?.items) setInscricoes(iJson.items);
      } catch (err) {
        console.warn("Erro ao carregar carteira", err);
      } finally {
        setLoading(false);
      }
    };
    void load();
  }, []);

  const list = useMemo(() => {
    const ticketItems =
      tickets?.map((t) => ({
        id: t.id,
        kind: t.isTournament ? "TORNEIO" : "BILHETE",
        title: t.event?.title ?? "Evento",
        slug: t.event?.slug ?? "",
        badge: t.badge,
        nextAction: t.nextAction,
        paymentStatusLabel: t.paymentStatusLabel,
        liveLink: t.liveLink ?? (t.event?.slug ? `/torneios/${t.event.slug}/live` : null),
      })) ?? [];
    const inscricaoItems =
      inscricoes?.map((i) => ({
        id: `insc-${i.id}`,
        kind: "TORNEIO",
      }));
  }, [tickets, inscricoes]);
}
```

```

    title: i.event?.title ?? "Torneio",
    slug: i.event?.slug ?? "",
    badge: i.badge,
    nextAction: i.nextAction,
    paymentStatusLabel: i.paymentStatusLabel,
    ctaUrl: i.ctaUrl,
    liveLink: i.liveLink,
  })) ?? [];
const combined = [...ticketItems, ...inscricaoItems];
return combined.filter((item) => {
  if (filter === "ALL") return true;
  if (filter === "TICKETS") return item.kind === "BILHETE";
  if (filter === "TOURNAMENTS") return item.kind === "TORNEIO";
  if (filter === "FREE") return item.badge === "FREE";
  if (filter === "RESALE") return item.badge === "RESALE";
  return true;
});
}, [tickets, inscricoes, filter]);
}

return (
<main className="mx-auto max-w-4xl px-4 py-8 space-y-4">
<header className="space-y-2">
  <h1 className="text-2xl font-semibold">Carteira</h1>
  <p className="text-sm text-gray-500">Bilhetes e inscrições num só sítio.</p>
  <div className="flex gap-2 flex-wrap">
    {[{
      { key: "ALL", label: "Tudo" },
      { key: "TICKETS", label: "Bilhetes" },
      { key: "TOURNAMENTS", label: "Torneios" },
      { key: "FREE", label: "Gratuitos" },
      { key: "RESALE", label: "Revenda" },
    ] satisfies { key: FilterKey; label: string }[]}.map((f) => (
      <button
        key={f.key}
        onClick={() => setFilter(f.key)}
        className={`rounded-full px-3 py-1 text-sm border ${filter === f.key ? "bg-black text-white" : "bg-white text-black"}`}
      >
        {f.label}
      </button>
    )));
  </div>
</header>

{loading && <p className="text-sm text-gray-500">A carregar...</p>}

<ul className="space-y-3">
  {list.map((item) => (
    <li key={item.id} className="rounded-lg border p-3 flex items-center justify-between gap-3">
      <div className="flex gap-2 items-center">
        <span className="text-xs rounded-full bg-gray-100 px-2 py-0.5">{item.kind}</span>
        <span className="text-xs rounded-full bg-blue-100 px-2 py-0.5">{item.badge}</span>
        {item.paymentStatusLabel && (
          <span className="text-xs text-gray-500">{item.paymentStatusLabel}</span>
        )}
      </div>
      <div className="text-sm font-medium">{item.title}</div>
    </div>
    <div className="flex gap-2">
      {item.nextAction === "CONFIRM_GUARANTEE" && (
        <a
          href={item.ctaUrl ?? "#"}
          className="bg-black text-white px-3 py-1 rounded text-sm"
        >
          Confirmar garantia
        </a>
      )}
      {item.nextAction === "PAY_PARTNER" && (
        <a href={item.ctaUrl ?? "#"} className="border px-3 py-1 rounded text-sm">
          Pagar a minha parte
        </a>
      )}
      {item.nextAction === "VIEW_LIVE" && (
        <a href={item.liveLink ?? `/torneios/${item.slug}/live`} className="underline text-sm">
          Ver ao vivo
        </a>
      )}
    </div>
  </li>
))
</ul>

```

```

        )}
        {!item.nextAction && item.liveLink && (
          <a href={item.liveLink} className="underline text-sm">
            Ver ao vivo
          </a>
        )}
      </div>
    </li>
  )));
  {!loading && list.length === 0 && <p className="text-sm text-gray-500">Nada para mostrar ainda.</p>}
</ul>
</main>
);
}

```

app/me/compras/page.tsx

```

"use client";

import { useEffect, useState } from "react";

type PurchaseItem = {
  id: number;
  purchaseId: string | null;
  totalCents: number;
  currency: string;
  createdAt: string;
  badge: string;
  status: string;
  timeline: { id: number; status: string; createdAt: string; source: string; errorMessage?: string | null }[];
  lines: { id: number; eventTitle: string; eventSlug: string; ticketTypeName: string }[];
};

export default function PurchasesPage() {
  const [items, setItems] = useState<PurchaseItem[]>([]);
  const [loading, setLoading] = useState(false);
  const [statusFilter, setStatusFilter] = useState<string | null>(null);
  const [includeFree, setIncludeFree] = useState(false);

  useEffect(() => {
    const load = async () => {
      setLoading(true);
      try {
        const params = new URLSearchParams();
        if (includeFree) params.set("includeFree", "true");
        if (statusFilter) params.set("status", statusFilter);
        const res = await fetch(`/api/me/purchases?${params.toString()}`);
        const json = await res.json();
        if (json?.items) setItems(json.items);
      } catch (err) {
        console.warn("Erro a carregar compras", err);
      } finally {
        setLoading(false);
      }
    };
    void load();
  }, [statusFilter, includeFree]);

  const statusOptions = ["PAID", "PROCESSING", "REFUNDED", "DISPUTED", "FAILED"];

  return (
    <main className="mx-auto max-w-4xl px-4 py-8 space-y-4">
      <header className="space-y-2">
        <h1 className="text-2xl font-semibold">Minhas Compras</h1>
        <div className="flex gap-2 items-center flex-wrap">
          <label className="text-sm flex items-center gap-1">
            <input type="checkbox" checked={includeFree} onChange={(e) => setIncludeFree(e.target.checked)} />
            Incluir gratuitos
          </label>
          <div className="flex gap-1 items-center">
            <span className="text-sm text-gray-500">Status:</span>
            <select
              value={statusFilter ?? ""}
              onChange={(e) => setStatusFilter(e.target.value || null)}
              className="border rounded px-2 py-1 text-sm"
            >

```

```

<option value="">Todos</option>
{statusOptions.map((s) => (
  <option key={s} value={s}>
    {s}
  </option>
))
</select>
</div>
</div>
</header>

{loading && <p className="text-sm text-gray-500">A carregar...</p>

<ul className="space-y-3">
  {items.map((item) => (
    <li key={item.id} className="rounded-lg border p-3 space-y-2">
      <div className="flex items-center gap-2">
        <span className="text-xs rounded-full bg-gray-100 px-2 py-0.5">{item.badge}</span>
        <span className="text-xs rounded-full bg-blue-100 px-2 py-0.5">{item.status}</span>
        <span className="text-xs text-gray-500">
          {new Date(item.createdAt).toLocaleString()} · {item.totalCents / 100} {item.currency}
        </span>
        {item.purchaseId && <span className="text-xs text-gray-400">ID: {item.purchaseId}</span>}
      </div>
      <div className="space-y-1">
        {item.lines.map((l) => (
          <div key={l.id} className="text-sm">
            {l.eventTitle} - {l.ticketTypeName}
          </div>
        ))
      </div>
      <div className="space-y-1">
        <p className="text-xs text-gray-500">Timeline:</p>
        <div className="flex flex-wrap gap-2 text-xs">
          {item.timeline.map((t) => (
            <span key={t.id} className="px-2 py-0.5 rounded bg-gray-100">
              {t.status} ({t.source}) {new Date(t.createdAt).toLocaleString()}
            </span>
          ))
        </div>
      </div>
    </li>
  )));
  {!loading && !items.length && <p className="text-sm text-gray-500">Sem compras.</p>}
</ul>
</main>
);
}

```

app/me/compras/purchases-client.tsx

```

"use client";

import useSWR from "swr";
import Link from "next/link";
import { formatMoney, centsToEuro } from "@/lib/money";
import { useUser } from "@app/hooks/useUser";

const fetcher = (url: string) => fetch(url).then((r) => r.json());

type Badge = "FREE" | "RESALE" | "SPLIT" | "FULL" | "SINGLE";
type NextAction = "NONE" | "PAY_PARTNER" | "CONFIRM_GUARANTEE";

type Purchase = {
  id: number;
  paymentIntentId: string | null;
  purchaseId?: string | null;
  badge?: Badge;
  nextAction?: NextAction;
  event: { id: number; title: string; slug: string; startsAt?: string | null; endsAt?: string | null } | null;
  subtotalCents: number;
  discountCents: number;
  platformFeeCents: number;
  totalCents: number;
  netCents: number;
  feeMode?: string | null;
}

```

```

createdAt: string;
lines: { ticketTypeId: number; quantity: number; unitPriceCents: number; discountPerUnitCents: number; grossCents: number }[];
};

type PurchasesResponse = { ok: true; items: Purchase[] } | { ok: false; error?: string };

function PurchaseCard({ purchase }: { purchase: Purchase }) {
  const eventTitle = purchase.event?.title ?? "Evento";
  const startsAt = purchase.event?.startsAt ? new Date(purchase.event.startsAt) : null;
  const dateStr = startsAt ? startsAt.toLocaleString("pt-PT", { day: "2-digit", month: "short", hour: "2-digit", minute: "2-digit" }) : null;
  const badgeLabel =
    purchase.badge === "FREE"
      ? "Gratuito"
      : purchase.badge === "RESALE"
        ? "Revenda"
        : purchase.badge === "SPLIT"
          ? "Split"
          : purchase.badge === "FULL"
            ? "Full"
            : "Compra";
  const nextActionLabel =
    purchase.nextAction === "PAY_PARTNER"
      ? "A tua parte está paga, falta o parceiro."
      : purchase.nextAction === "CONFIRM_GUARANTEE"
        ? "Ação necessária: confirmar a garantia."
        : null;
  return (
    <div className="rounded-2xl border border-white/12 bg-white/5/50 p-4 shadow-[0_16px_50px_rgba(0,0,0,0.35)] backdrop-blur">
      <div className="flex flex-col gap-2 sm:flex-row sm:items-start sm:justify-between">
        <div className="space-y-1 text-sm">
          <p className="font-semibold text-white">{eventTitle}</p>
          {dateStr && <p className="text-[12px] text-white/60">{dateStr}</p>}
          <p className="text-[12px] text-white/50">
            Compra #{purchase.id} {purchase.purchaseId} ` · ${purchase.purchaseId}` : ""
          </p>
        </div>
        <div className="mt-1 flex flex-wrap items-center gap-2">
          <span className="rounded-full bg-white/10 px-2 py-[2px] text-[11px] text-white/80">
            {badgeLabel}
          </span>
          {nextActionLabel && (
            <span className="rounded-full bg-yellow-300/15 px-2 py-[2px] text-[11px] text-yellow-100">
              {nextActionLabel}
            </span>
          )}
        </div>
      </div>
      <div className="text-right text-sm text-white/80">
        <p className="text-lg font-semibold text-white">{formatMoney(centsToEuro(purchase.totalCents) ?? 0)}</p>
        <p className="text-[11px] text-white/60">{new Date(purchase.createdAt).toLocaleDateString("pt-PT")}</p>
      </div>
    </div>
    {nextActionLabel && (
      <div className="mt-2 rounded-xl border border-yellow-300/20 bg-yellow-400/10 px-3 py-2 text-[12px] text-yellow-100">
        {nextActionLabel}
      </div>
    )}
    <div className="mt-2 space-y-1 rounded-xl border border-white/10 bg-black/30 p-3 text-[13px] text-white/80">
      {purchase.lines.map((line) => (
        <div key={`${purchase.id}-${line.ticketTypeId}-${line.quantity}`} className="flex items-center justify-between text-[13px] text-white/70">
          <span>Bilhete #${line.ticketTypeId} x ${line.quantity}</span>
          <span>{formatMoney(centsToEuro(line.grossCents) ?? 0)}</span>
        </div>
      )))
    </div>
    <div className="mt-2 flex flex-wrap items-center justify-between gap-2 text-[12px] text-white/60">
      {purchase.discountCents > 0 && <span>Desconto: -{formatMoney(centsToEuro(purchase.discountCents) ?? 0)}</span>}
      {purchase.platformFeeCents > 0 && <span>Taxas: {formatMoney(centsToEuro(purchase.platformFeeCents) ?? 0)}</span>}
    </div>
    {purchase.event?.slug && (
      <Link href={`/eventos/${purchase.event.slug}`} className="mt-3 inline-flex text-[12px] text-[#6BFFFF] hover:underline">
        Ver evento
      </Link>
    )}
  </div>
);

```

```

}

export default function PurchasesClient() {
  const { user, profile } = useUser();
  const { data, isLoading } = useSWR<PurchasesResponse>("/api/me/purchases", fetcher, {
    revalidateOnFocus: false,
  });

  return (
    <div className="min-h-screen bg-[radial-gradient(circle_at_10%_20%,rgba(131,58,180,0.18),transparent_30%),radial-gradient(circle_at_80%_10%,rgba(45,156,219,0.18),transparent_25%),radial-gradient(circle_at_50%_80%,rgba(99,102,241,0.14),transparent_35%)] text-white">
      <div className="mx-auto flex max-w-5xl flex-col gap-px-4 pb-14 pt-10 sm:px-6 lg:px-8">
        <div className="rounded-3xl border border-white/10 bg-black/50 p-6 shadow-[0_24px_70px_rgba(0,0,0,0.65)] backdrop-blur">
          <div className="flex flex-col gap-3 sm:flex-row sm:items-center sm:justify-between">
            <div className="space-y-1">
              <p className="text-[11px] uppercase tracking-[0.28em] text-white/60">Área pessoal</p>
              <h1 className="text-3xl font-semibold">As minhas compras</h1>
              <p className="text-sm text-white/65">Recibos e históricos dos teus bilhetes, num só lugar.</p>
            </div>
            {user && (
              <div className="flex items-center gap-3 rounded-2xl border border-white/10 bg-white/5 px-3 py-2 text-[12px] text-white/75">
                <div className="flex h-10 w-10 items-center justify-center rounded-full border border-white/15 bg-gradient-to-br from-[#0f172a] via-[#111827] to-[#0b1224] text-sm font-semibold">
                  {(profile?.username ?? user.email ?? "U")[0].toUpperCase()}
                </div>
                <div className="leading-tight">
                  <p className="font-semibold text-white">{profile?.fullName || profile?.username || user.email}</p>
                  <p className="text-[11px] text-white/60">Conta ORYA</p>
                </div>
              </div>
            )}
            </div>
          </div>
        </div>
      </div>
    </div>
    {isLoading && (
      <div className="rounded-3xl border border-white/10 bg-black/40 p-6 text-white/70">A carregar compras...</div>
    )}
    {!isLoading && (!data || data.ok === false) && (
      <div className="rounded-3xl border border-red-400/30 bg-red-900/20 p-6 text-sm text-red-100">
        <p className="text-lg font-semibold text-white">Não foi possível carregar as compras.</p>
        <p className="text-white/80">Tenta novamente mais tarde.</p>
      </div>
    )}
    {!isLoading && data && data.ok && data.items.length === 0 && (
      <div className="flex flex-col items-center justify-center gap-3 rounded-3xl border border-white/10 bg-black/40 p-8 text-center text-white/80 shadow-[0_18px_60px_rgba(0,0,0,0.45)]">
        <div className="h-12 w-12 rounded-full border border-white/15 bg-white/5 text-xl flex items-center justify-center">
          <div>
            <p className="text-lg font-semibold text-white">Ainda não tens compras.</p>
            <p className="text-sm text-white/65">Quando comprares bilhetes, aparecem aqui com recibo.</p>
          </div>
          <Link href="/explorar"
            className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow hover:scale-[1.02]"
          >
            Explorar eventos
          </Link>
        </div>
      </div>
    )}
    {!isLoading && data && data.ok && data.items.length > 0 && (
      <div className="space-y-3">
        {data.items.map((p) => (
          <PurchaseCard key={p.id} purchase={p} />
        )))
      </div>
    )}
    </div>
  </div>
);
}

```

app/me/edit/page.tsx

```
// app/me/edit/page.tsx
"use client";

import { useEffect, useState, FormEvent } from "react";
import { useRouter } from "next/navigation";
import { useUser } from "@/app/hooks/useUser";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";
import { sanitizeUsername, validateUsername, USERNAME_RULES_HINT } from "@/lib/username";

type SaveBasicResponse = {
  ok: boolean;
  profile?: {
    id: string;
    username: string | null;
    fullName: string | null;
    onboardingDone: boolean;
  } | null;
  error?: string;
};

export default function EditProfilePage() {
  const router = useRouter();
  const { user, profile, isLoading, mutate } = useUser();
  const { openModal } = useAuthModal();

  const [fullName, setFullName] = useState("");
  const [username, setUsername] = useState("");
  const [avatarUrl, setAvatarUrl] = useState<string | null>(null);
  const [avatarUploading, setAvatarUploading] = useState(false);
  const [profileVisibility, setProfileVisibility] = useState<"PUBLIC" | "PRIVATE">("PUBLIC");
  const [allowEmailNotifications, setAllowEmailNotifications] = useState(true);
  const [allowEventReminders, setAllowEventReminders] = useState(true);
  const [allowFriendRequests, setAllowFriendRequests] = useState(true);
  const [allowSalesAlerts, setAllowSalesAlerts] = useState(true);
  const [allowSystemAnnouncements, setAllowSystemAnnouncements] = useState(true);

  const [checkingUsername, setCheckingUsername] = useState(false);
  const [usernameAvailable, setUsernameAvailable] = useState<boolean | null>(
    null,
  );
  const [usernameHint, setUsernameHint] = useState<string | null>(null);

  const [saving, setSaving] = useState(false);
  const [errorMsg, setErrorMsg] = useState<string | null>(null);
  const [successMsg, setSuccessMsg] = useState<string | null>(null);
  const [uploadError, setUploadError] = useState<string | null>(null);

  async function handleAvatarUpload(file: File | null) {
    if (!file) return;
    setUploadError(null);
    setAvatarUploading(true);
    try {
      const formData = new FormData();
      formData.append("file", file);
      const res = await fetch("/api/upload", { method: "POST", body: formData });
      const json = await res.json().catch(() => null);
      if (!res.ok || !json?.url) {
        throw new Error(json?.error || "Falha no upload.");
      }
      setAvatarUrl(json.url);
      setSuccessMsg("Foto atualizada. Não te esqueças de guardar.");
    } catch (err) {
      console.error("[me/edit] upload avatar", err);
      setUploadError("Não foi possível enviar a foto. Tenta outra vez.");
    } finally {
      setAvatarUploading(false);
    }
  }

  // Preenche o formulário quando o perfil estiver carregado
  useEffect(() => {
    if (!isLoading && !user) {
      // Se alguém chegar aqui sem sessão, mandamos para login
      openModal({ mode: "login", redirectTo: "/me/edit", showGoogle: true });
    }
  });
}
```

```

    router.push("/");
    return;
}

if (profile) {
  setFullName(profile.fullName ?? "");
  setUsername(sanitizeUsername(profile.username ?? ""));
  setAvatarUrl(profile.avatarUrl ?? null);
  setProfileVisibility(profile.profileVisibility === "PRIVATE" ? "PRIVATE" : "PUBLIC");
  setAllowEmailNotifications(Boolean(profile.allowEmailNotifications));
  setAllowEventReminders(Boolean(profile.allowEventReminders));
  setAllowFriendRequests(Boolean(profile.allowFriendRequests));
}
}, [isLoading, user, profile, openModal, router]);

useEffect(() => {
  let cancelled = false;
  const loadPrefs = async () => {
    try {
      const res = await fetch("/api/notifications/prefs");
      const json = await res.json().catch(() => null);
      if (!res.ok || !json?.prefs || cancelled) return;
      setAllowEmailNotifications(Boolean(json.prefs.allowEmailNotifications));
      setAllowEventReminders(Boolean(json.prefs.allowEventReminders));
      setAllowFriendRequests(Boolean(json.prefs.allowFriendRequests));
      setAllowSalesAlerts(Boolean(json.prefs.allowSalesAlerts));
      setAllowSystemAnnouncements(Boolean(json.prefs.allowSystemAnnouncements));
    } catch (err) {
      console.warn("[me/edit] prefs fetch failed", err);
    }
  };
  loadPrefs();
  return () => {
    cancelled = true;
  };
}, []);

async function checkUsernameAvailability(value: string) {
  const trimmed = sanitizeUsername(value);

  if (!trimmed) {
    setUsernameAvailable(null);
    setUsernameHint(USER_NAME_RULES_HINT);
    return false;
  }

  const validation = validateUsername(trimmed);
  if (!validation.valid) {
    setUsernameHint(validation.error);
    setUsernameAvailable(false);
    return false;
  }

  // Se não mudou em relação ao username atual, consideramos disponível
  if (trimmed === (profile?.username ?? "")) {
    setUsernameAvailable(true);
    setUsernameHint(null);
    return true;
  }

  try {
    setCheckingUsername(true);
    const res = await fetch(` /api/username/check?username=${encodeURIComponent(trimmed)}`);

    if (!res.ok) {
      setUsernameAvailable(null);
      return false;
    }

    const json = (await res.json()) as { available: boolean };
    setUsernameAvailable(json.available);
    setUsernameHint(json.available ? null : "Esse username já existe.");
    return json.available;
  } catch (err) {
    console.error("Erro ao verificar username:", err);
    setUsernameAvailable(null);
    return false;
  } finally {
}

```

```

        setCheckingUsername(false);
    }
}

async function handleSubmit(e: FormEvent) {
    e.preventDefault();

    setErrorMsg(null);
    setSuccessMsg(null);

    const trimmedUsername = sanitizeUsername(username);
    const validation = validateUsername(trimmedUsername);

    if (!validation.valid) {
        setErrorMsg(validation.error);
        return;
    }

    if (usernameAvailable === false) {
        setErrorMsg("Esse username já está a ser usado.");
        return;
    }

    if (usernameAvailable === null) {
        const ok = await checkUsernameAvailability(trimmedUsername);
        if (!ok) {
            setErrorMsg("Esse username já está a ser usado.");
            return;
        }
    }

    try {
        setSaving(true);

        const res = await fetch("/api/profiles/save-basic", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                fullName: fullName.trim() || null,
                username: validation.normalized,
                avatarUrl: avatarUrl ?? null,
                visibility: profileVisibility,
                allowEmailNotifications,
                allowEventReminders,
                allowFriendRequests,
            }),
        });

        if (!res.ok) {
            setErrorMsg("Erro a guardar o perfil.");
            setSaving(false);
            return;
        }

        const json = (await res.json()) as SaveBasicResponse;

        if (!json.ok) {
            setErrorMsg(json.error || "Erro a guardar o perfil.");
            setSaving(false);
            return;
        }

        await fetch("/api/notifications/prefs", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                allowEmailNotifications,
                allowEventReminders,
                allowFriendRequests,
                allowSalesAlerts,
                allowSystemAnnouncements,
            }),
        }).catch(() => null);

        setSuccessMsg("Perfil atualizado!");

        // Atualiza o cache do /api/auth/me
        await mutate();
    }
}

```

```

        setTimeout(() => {
          router.push("/me");
        }, 800);
      } catch (err) {
        console.error("Erro inesperado ao guardar perfil:", err);
        setErrorMsg("Erro inesperado ao guardar o perfil.");
      } finally {
        setSaving(false);
      }
    }

    if (isLoading) {
      return (
        <main className="min-h-screen orya-body-bg text-white flex items-center justify-center">
          <p className="text-white/60 text-sm">A carregar o teu perfil...</p>
        </main>
      );
    }

    return (
      <main className="min-h-screen orya-body-bg text-white pb-24">
        <div className="max-w-xl mx-auto px-5 pt-6">
          {/* Header */}
          <header className="flex items-center justify-between gap-3 mb-6">
            <div>
              <h1 className="text-2xl md:text-3xl font-semibold tracking-tight">
                Editar perfil
              </h1>
              <p className="mt-1 text-sm text-white/65">
                Ajusta o teu nome e username dentro da ORYA.
              </p>
            </div>

            <button
              type="button"
              onClick={() => router.push("/me")}
              className="hidden sm:inline-flex items-center gap-2 text-[11px] text-white/55 hover:text-white/85"
            >
              ← Voltar à conta
            </button>
          </header>

          {/* Mensagens */}
          {errorMsg && (
            <div className="mb-4 p-3 border border-red-500/40 bg-red-500/10 rounded-lg text-red-200 text-sm">
              {errorMsg}
            </div>
          )}
          {successMsg && (
            <div className="mb-4 p-3 border border-emerald-500/40 bg-emerald-500/10 rounded-lg text-emerald-200 text-sm">
              {successMsg}
            </div>
          )}
        </div>

        <section className="rounded-2xl border border-white/15 bg-gradient-to-br from-white/[0.04] via-slate-950/85 to-slate-950
background-blur-xl p-6 shadow-[0_14px_34px_rgba(15,23,42,0.8)] space-y-4">
          <div className="flex items-center gap-4">
            <div className="relative h-16 w-16 rounded-full border border-white/15 bg-white/5 overflow-hidden flex items-center
justify-center shadow-[0_0_18px_rgba(107,255,255,0.35)]">
              {avatarUrl ? (
                // eslint-disable-next-line @next/next/no-img-element
                <img src={avatarUrl} alt="Avatar" className="h-full w-full object-cover" />
              ) : (
                <span className="text-lg font-semibold text-white/70">
                  {fullName?.[0]?.toUpperCase() || "?"}
                </span>
              )}
              <span className="pointer-events-none absolute inset-0 rounded-full ring-1 ring-white/10" />
            </div>
            <div className="flex flex-col gap-2 text-sm">
              <div className="flex gap-2 flex-wrap">
                <label className="cursor-pointer rounded-full border border-white/20 bg-white/10 px-3 py-1.5 text-xs font-medium
text-white/85 hover:border-white/35">
                  {avatarUploading ? "A enviar..." : "Carregar nova foto"}
                <input
                  type="file"
                  accept="image/*"
                </input>
              </div>
            </div>
          </div>
        </section>
      </main>
    );
  }
}

```

```

        className="hidden"
        onChange={(e) => handleAvatarUpload(e.target.files?[0] ?? null)}
        disabled={avatarUploading}
      />
    </label>
    <button
      type="button"
      onClick={() => setAvatarUrl(null)}
      className="rounded-full border border-white/20 px-3 py-1.5 text-xs text-white/75 hover:border-red-400
      hover:text-red-200"
      >
      Remover foto
    </button>
  </div>
  <p className="text-[11px] text-white/55">Foto circular, idealmente quadrada (ex.: 800x800).</p>
  {uploadError && <p className="text-[11px] text-red-300">{uploadError}</p>}
</div>
</div>

<form className="space-y-4" onSubmit={handleSubmit}>
  <div className="flex flex-col gap-1">
    <label className="text-white/70 text-sm">Nome público</label>
    <input
      type="text"
      value={fullName}
      onChange={(e) => setFullName(e.target.value)}
      className="bg-black/40 border border-white/15 rounded-lg px-3 py-2 text-white outline-none focus:border-
      [#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]/60 text-sm"
      placeholder="Como te chamas?"
    />
  </div>

  <div className="flex flex-col gap-1">
    <label className="text-white/70 text-sm">Username</label>
    <input
      type="text"
      inputMode="text"
      pattern="[A-Za-z0-9._]{0,30}"
      value={username}
      onChange={(e) => {
        const raw = e.target.value;
        const cleaned = sanitizeUsername(raw);
        setUsername(cleaned);
        const validation = validateUsername(cleaned);
        setUsernameHint(validation.valid ? null : validation.error);
        setUsernameAvailable(null);
      }}
      onBlur={(e) => checkUsernameAvailability(e.target.value)}
      maxLength={30}
      className="bg-black/40 border border-white/15 rounded-lg px-3 py-2 text-white outline-none focus:border-
      [#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]/60 text-sm"
      placeholder="Escolhe o teu @username"
    />
    {usernameHint && (
      <p className="text-[11px] text-amber-300 mt-1">{usernameHint}</p>
    )}
    <p className="text-[11px] text-white/55 mt-1">
      Este será o link do teu perfil público (ex.: orya.app/@teuusername).
    </p>
    {checkingUsername && (
      <p className="text-[11px] text-white/60 mt-1">
        A verificar disponibilidade..
      </p>
    )}
    {usernameAvailable === false && !checkingUsername && (
      <p className="text-[11px] text-red-300 mt-1">
        Esse username já está usado.
      </p>
    )}
    {usernameAvailable && !checkingUsername && (
      <p className="text-[11px] text-emerald-300 mt-1">
        Username disponível ✓
      </p>
    )}
  </div>

<div className="grid grid-cols-1 gap-4 rounded-xl border border-white/10 bg-black/30 p-4 sm:grid-cols-2">
  <div className="space-y-2">

```

```

<p className="text-sm font-semibold text-white">Visibilidade do perfil</p>
<div className="space-y-2 text-sm text-white/75">
  {
    {
      key: "PUBLIC" as const,
      title: "Público",
      desc: "Visível em listas públicas, convites e páginas de evento.",
    },
    {
      key: "PRIVATE" as const,
      title: "Privado",
      desc: "Mostra só avatar, nome e username. Sem email/telefone.",
    },
  ].map((opt) => (
  <label
    key={opt.key}
    className={`flex cursor-pointer flex-col gap-1 rounded-lg border px-3 py-2 transition ${profileVisibility === opt.key ? "border-white/60 bg-white/10" : "border-white/15 bg-black/30 hover:border-white/35"}`}
  >
    <div className="flex items-center justify-between gap-2">
      <span className="font-semibold">{opt.title}</span>
      <input
        type="radio"
        name="visibility"
        className="accent-white"
        checked={profileVisibility === opt.key}
        onChange={() => setProfileVisibility(opt.key)}
      />
    </div>
    <p className="text-xs text-white/65">{opt.desc}</p>
  </label>
))
</div>
</div>

<div className="space-y-2">
  <p className="text-sm font-semibold text-white">Notificações</p>
  <div className="space-y-2 text-sm text-white/75">
    {
      {
        key: "allowEmailNotifications" as const,
        label: "Emails de novidades e segurança",
        value: allowEmailNotifications,
        setter: setAllowEmailNotifications,
      },
      {
        key: "allowEventReminders" as const,
        label: "Lembretes de eventos/experiências",
        value: allowEventReminders,
        setter: setAllowEventReminders,
      },
      {
        key: "allowFriendRequests" as const,
        label: "Pedidos de amizade / convites",
        value: allowFriendRequests,
        setter: setAllowFriendRequests,
      },
      {
        key: "allowSalesAlerts" as const,
        label: "Alertas de vendas/promoções",
        value: allowSalesAlerts,
        setter: setAllowSalesAlerts,
      },
      {
        key: "allowSystemAnnouncements" as const,
        label: "Anúncios do sistema",
        value: allowSystemAnnouncements,
        setter: setAllowSystemAnnouncements,
      },
    ].map((opt) => (
    <label
      key={opt.key}
      className="flex items-center justify-between rounded-lg border border-white/10 bg-black/20 px-3 py-2 text-xs"
    >

```

```

<span className="text-white/80">{opt.label}</span>
<button
  type="button"
  onClick=={() => opt.setter(!opt.value)}
  className=`rounded-full px-3 py-1 text-[11px] font-semibold transition ${
    opt.value ? "bg-white text-black" : "border border-white/25 text-white/70 hover:border-white/60"
  }`}
>
  {opt.value ? "On" : "Off"}
</button>
</label>
)}
</div>
</div>
</div>

<div className="flex flex-col sm:flex-row sm:items-center sm:justify-between gap-3 pt-2"
  <button
    type="button"
    onClick=={() => router.push("/me")}
    className="inline-flex justify-center px-4 py-2 rounded-xl border border-white/20 bg-white/5 text-xs font-medium text-white/80 hover:border-white/40 hover:bg-white/10 transition"
  >
    Cancelar
</button>
<button
  type="submit"
  disabled={saving}
  className="w-full sm:w-auto px-6 py-3 rounded-2xl text-sm font-semibold bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] shadow-[0_0_22px_#1646F577] disabled:opacity-60 disabled:cursor-not-allowed"
  >
    {saving ? "A guardar..." : "Guardar alterações"}
</button>
</div>
</form>
</section>
</div>
</main>
);
}

```

app/me/experiencias/page.tsx

```

// app/me/experiencias/page.tsx
"use client";

import Link from "next/link";
import { useEffect, useState } from "react";

type ExperienceCard = {
  id: number;
  title: string;
  slug: string;
  startsAt: string | null;
  locationName: string | null;
  coverImageUrl: string | null;
};

export default function MinhasExperienciasPage() {
  const [items, setItems] = useState<ExperienceCard[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    let active = true;
    async function load() {
      try {
        const res = await fetch("/api/me/experiencias", { cache: "no-store" });
        if (res.status === 404) {
          setItems([]);
          return;
        }
        if (!res.ok) throw new Error("Erro ao carregar experiências");
        const data = await res.json();
        if (!active) return;
        setItems(Array.isArray(data?.items) ? data.items : []);
      }
    }
    load();
  }, []);
}

```

```

    } catch (err) {
      console.error(err);
      if (active) setError("Não foi possível carregar as experiências.");
    } finally {
      if (active) setLoading(false);
    }
  }
  load();
  return () => {
    active = false;
  };
}, []);
}

return (
<div className="mx-auto flex max-w-5xl flex-col gap-6 px-4 py-10 text-white">
  <div className="flex items-center justify-between">
    <div>
      <p className="text-xs uppercase tracking-[0.18em] text-white/60">Área pessoal</p>
      <h1 className="text-2xl font-semibold">Minhas experiências</h1>
      <p className="text-sm text-white/70">Experiências que compraste ou reservaste.</p>
    </div>
    <Link
      href="/explorar"
      className="rounded-full bg-white px-2 text-sm font-semibold text-black shadow-lg hover:brightness-105"
    >
      Explorar experiências
    </Link>
  </div>
  <div>
    {loading && (
      <div className="rounded-2xl border border-white/10 bg-white/[0.04] p-4">
        <div className="h-4 w-32 animate-pulse rounded-full bg-white/10" />
        <div className="mt-3 space-y-2">
          {[1, 2, 3].map((v) => (
            <div key={v} className="flex gap-3">
              <div className="h-16 w-16 animate-pulse rounded-xl bg-white/10" />
              <div className="flex-1 space-y-2">
                <div className="h-4 w-40 animate-pulse rounded-full bg-white/10" />
                <div className="h-3 w-24 animate-pulse rounded-full bg-white/10" />
              </div>
            </div>
          )));
        </div>
      </div>
    )}

    {!loading && error && (
      <div className="rounded-2xl border border-red-500/30 bg-red-500/10 px-4 py-3 text-sm text-red-200">
        {error}
      </div>
    )}
  </div>
  {!loading && !error && items.length === 0 && (
    <div className="rounded-2xl border border-white/10 bg-white/[0.04] p-4 text-sm text-white/75">
      Ainda não tens experiências. Quando comprares ou fores convidado para uma experiência, ela aparece aqui.
    </div>
  )}
  {!loading && !error && items.length > 0 && (
    <div className="grid gap-4 md:grid-cols-2">
      {items.map((exp) => (
        <Link
          key={exp.id}
          href={`/experiencias/${exp.slug}`}
          className="group flex gap-3 rounded-2xl border border-white/10 bg-white/[0.04] p-3 transition hover:border-#6BFFFF/80 hover:-translate-y-[2px]"
        >
          <div className="h-20 w-20 overflow-hidden rounded-xl bg-gradient-to-br from-[#22d3ee]/30 via-[#34d399]/20 to-[#0ea5e9]/30">
            {exp.coverImageUrl ? (
              // eslint-disable-next-line @next/next/no-img-element
              <img
                src={exp.coverImageUrl}
                alt={exp.title}
                className="h-full w-full object-cover transition-transform duration-200 group-hover:scale-105"
              />
            ) : null}
          </div>
        </Link>
      ))}
    </div>
  )}
</div>

```

```

        </div>
      <div className="flex-1 space-y-1">
        <p className="text-[13px] font-semibold leading-tight group-hover:text-[#6BFFFF]">
          {exp.title}
        </p>
        <p className="text-xs text-white/65">
          {exp.startsAt ? new Date(exp.startsAt).toLocaleString("pt-PT") : "Data a definir"}
        </p>
        <p className="text-xs text-white/60">{exp.locationName || "Local a anunciar"}</p>
      </div>
    </Link>
  )));
</div>
)
</div>
);
}

```

app/me/page.tsx

```

"use client";

import { useEffect, useState } from "react";
import Image from "next/image";
import Link from "next/link";
import ProfileHeader from "@/app/components/profile/ProfileHeader";
import { useUser } from "@/app/hooks/useUser";
import { defaultBlurDataURL, optimizeImageUrl } from "@/lib/image";

type UserTicket = {
  id: string;
  qrToken: string | null;
  eventId: number;
  eventSlug: string;
  eventTitle: string;
  eventStartDate: string;
  eventLocationName: string | null;
  eventCoverImageUrl: string | null;
  ticketName: string;
  quantity: number;
  pricePaid: number; // em céntimos
  currency: string;
  createdAt: string; // data de compra
};

type TicketsApiResponse =
| {
  success?: boolean;
  tickets: UserTicket[];
  error?: string;
}
| {
  success?: boolean;
  error: string;
  tickets?: UserTicket[];
};

function parseDate(value?: string | null): Date | null {
  if (!value) return null;
  const d = new Date(value);
  if (Number.isNaN(d.getTime())) return null;
  return d;
}

export default function MePage() {
  const { user, profile, isLoading: meLoading } = useUser();

  const [ticketsLoading, setTicketsLoading] = useState(true);
  const [ticketsError, setTicketsError] = useState<string | null>(null);
  const [tickets, setTickets] = useState<UserTicket[]>([]);

  const isAdmin = Array.isArray(profile?.roles) && profile.roles.includes("admin");

  const fetchTickets = async () => {
    setTicketsLoading(true);
    setTicketsError(null);
  }
}

```

```

try {
  const res = await fetch("/api/me/tickets", {
    credentials: "include",
    cache: "no-store",
  });

  if (!res.ok) {
    if (res.status === 401) {
      setTicketsError("Precisas de iniciar sessão para ver os teus bilhetes.");
      setTickets([]);
      return;
    }

    const text = await res.text();
    console.error("Erro /api/me/tickets:", text);
    setTicketsError("Erro ao carregar os teus bilhetes.");
    return;
  }
}

const data: TicketsApiResponse = await res.json();

if ("success" in data && data.success === false) || "error" in data) {
  setTicketsError("Erro ao carregar os teus bilhetes.");
  setTickets([]);
  return;
}

const mappedTickets =
  (data as { tickets?: any[] }).tickets?.map((t) => ({
    id: t.id,
    qrToken: t.qrToken ?? null,
    eventId: t.event?.id ?? 0,
    eventSlug: t.event?.slug ?? "",
    eventTitle: t.event?.title ?? "Evento",
    eventStartDate: t.event?.startDate ?? t.purchasedAt,
    eventLocationName: t.event?.locationName ?? null,
    eventCoverImageUrl: t.event?.coverImageUrl ?? null,
    ticketName: t.ticket?.name ?? "Bilhete",
    quantity: t.quantity ?? 1,
    pricePaid: t.pricePaid ?? 0,
    currency: t.currency ?? "EUR",
    createdAt: t.purchasedAt ?? new Date().toISOString(),
  })) ?? [];

setTickets(mappedTickets);
} catch (err) {
  console.error("Erro inesperado em /api/me/tickets:", err);
  setTicketsError("Erro inesperado ao carregar os teus bilhetes.");
} finally {
  setTicketsLoading(false);
}
};

useEffect(() => {
  if (!user) {
    setTickets([]);
    setTicketsLoading(false);
    return;
  }

  fetchTickets();
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, [user?.id]);

useEffect(() => {
  const onFocus = () => {
    if (user) fetchTickets();
  };
  window.addEventListener("focus", onFocus);
  return () => window.removeEventListener("focus", onFocus);
}, [user]);

const displayName =
  profile?.fullName ||
  profile?.username ||
  user?.email?.split("@")[0] ||
  "Utilizador ORYA";

```

```

const displayInitial =
  (profile?.fullName || profile?.username || user?.email || "0")
    .trim()
    .charAt(0)
    .toUpperCase() || "0";

const now = new Date();

const upcomingTickets = tickets.filter((t) => {
  const d = parseDate(t.eventStartDate);
  return d !== null && d.getTime() >= now.getTime();
});

const pastTickets = tickets.filter((t) => {
  const d = parseDate(t.eventStartDate);
  return d !== null && d.getTime() < now.getTime();
});

const totalEvents = tickets.length;
const totalUpcoming = upcomingTickets.length;
const totalPast = pastTickets.length;

const totalSpentCents = tickets.reduce((sum, t) => sum + (t.pricePaid || 0), 0);
const totalSpentEuros = (totalSpentCents / 100).toFixed(2);

let levelLabel = "Explorador ORYA";
let levelDescription =
  "Começo perfeito. Em breve vais desbloquear novas badges com mais eventos.";

if (totalEvents >= 10) {
  levelLabel = "Lenda dos eventos";
  levelDescription =
    "Já estás em modo ORYA total. Continuar assim e vamos ter de inventar um nível novo só para ti.";
} else if (totalEvents >= 5) {
  levelLabel = "Insider da noite";
  levelDescription =
    "Já és cliente recorrente. Os melhores spots da tua cidade começam a ser a tua segunda casa.";
} else if (totalEvents >= 1) {
  levelLabel = "Primeiros passos";
  levelDescription =
    "Já tens os teus primeiros bilhetes ORYA. Bora continuar a construir a tua timeline.";
}

const hasTickets = !ticketsLoading && tickets.length > 0;
const hasActivity = !ticketsLoading && tickets.length > 0;

// Bilhetes mais recentes para preview no perfil (3-4)
const sortedTickets = [...tickets].sort((a, b) => {
  const d1 = parseDate(a.eventStartDate)?.getTime() ?? 0;
  const d2 = parseDate(b.eventStartDate)?.getTime() ?? 0;
  return d2 - d1;
});

const recentTickets = sortedTickets.slice(0, 4);

return (
  <main className="min-h-screen bg-gradient-to-b from-slate-950 via-slate-900 to-black text-white">
    <section className="mx-auto flex max-w-6xl flex-col gap-6 px-4 py-8">
      <ProfileHeader
        displayName={displayName}
        handle={profile?.username || user?.email || "user"}
        avatarUrl={profile?.avatarUrl}
        stats={{
          totalEvents,
          totalUpcoming,
          totalPast,
          totalSpentEuros,
        }}
        levelLabel={levelLabel}
        levelDescription={levelDescription}
        isLoading={meLoading}
        isAdmin={isAdmin}
      />
      {/* STATUS */}
      <section className="rounded-2xl border border-white/8 bg-white/[0.02] p-5 shadow-[0_18px_50px_rgba(0,0,0,0.65)] backdrop-blur-xl">

```

```

<div className="grid gap-3 sm:grid-cols-2 lg:grid-cols-4">
  <div className="rounded-xl border border-white/10 bg-gradient-to-br from-[#0f172a] via-slate-950 to-slate-950 p-3">
    <p className="text-[11px] uppercase tracking-[0.16em] text-white/65">
      Eventos com bilhete
    </p>
    <p className="mt-1 text-2xl font-semibold text-white">{totalEvents}</p>
    <p className="text-[12px] text-white/60">Timeline ORYA.</p>
  </div>

  <div className="rounded-xl border border-emerald-400/20 bg-emerald-500/10 p-3">
    <p className="text-[11px] uppercase tracking-[0.16em] text-emerald-50/80">
      Próximos
    </p>
    <p className="mt-1 text-2xl font-semibold text-emerald-50">{totalUpcoming}</p>
    <p className="text-[12px] text-emerald-50/80">0 que vem ai.</p>
  </div>

  <div className="rounded-xl border border-cyan-400/20 bg-cyan-500/10 p-3">
    <p className="text-[11px] uppercase tracking-[0.16em] text-cyan-50/80">
      Passados
    </p>
    <p className="mt-1 text-2xl font-semibold text-cyan-50">{totalPast}</p>
    <p className="text-[12px] text-cyan-50/80">Memórias.</p>
  </div>

  <div className="rounded-xl border border-purple-400/25 bg-purple-500/10 p-3">
    <p className="text-[11px] uppercase tracking-[0.16em] text-purple-50/80">
      Total investido
    </p>
    <p className="mt-1 text-2xl font-semibold text-purple-50">{totalSpentEuros} €</p>
    <p className="text-[12px] text-purple-50/80">Bruto - taxas.</p>
  </div>
</div>
</section>

/* BILHETES */
<section className="rounded-2xl border border-[#6BFFFF]/30 bg-gradient-to-br from-[#020617f2] via-slate-950 to-slate-950 backdrop-blur-xl p-5 space-y-4 shadow-[0_16px_40px_rgba(15,23,42,0.7)] min-h-[320px]">
  <div className="flex items-center justify-between gap-3 flex-wrap">
    <div>
      <h2 className="text-sm font-semibold text-white/95">Os meus bilhetes</h2>
      <p className="text-[11px] text-white/65">
        Tudo o que já reservaste na ORYA. Próximos eventos à frente, memórias logo atrás.
      </p>
    </div>
    <Link href="/me/carteira" className="inline-flex items-center gap-1 rounded-full bg-white text-black text-[11px] font-semibold px-3.5 py-1.5 shadow-[0_0_18px_rgba(255,255,255,0.3)] hover:scale-[1.02] active:scale-95 transition-transform">
      <span>Ver carteira</span>
    </Link>
  </div>
  {ticketsLoading && (
    <div className="space-y-2">
      <div className="h-24 rounded-xl bg-white/5 border border-white/10 animate-pulse" />
      <div className="h-24 rounded-xl bg-white/5 border border-white/10 animate-pulse" />
    </div>
  )}

  {!ticketsLoading && ticketsError && (
    <div className="rounded-xl border border-red-400/40 bg-red-500/10 px-3 py-2.5 text-[11px] text-red-50">
      {ticketsError}
    </div>
  )}

  {!ticketsLoading && !ticketsError && tickets.length === 0 && (
    <div className="rounded-3xl border border-white/10 bg-gradient-to-br from-[#0f172a]/70 via-[#020617]/60 to-black/70 backdrop-blur-2xl p-8 text-center flex flex-col items-center gap-4 shadow-[0_28px_80px_rgba(15,23,42,0.95)]">
      <div className="h-20 w-20 rounded-2xl bg-gradient-to-br from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] flex items-center justify-center shadow-[0_0_35px_rgba(107,255,255,0.5)] text-black text-3xl font-bold">
        <span>Ainda não tens bilhetes ORYA</span>
      </div>
      <h3 className="text-lg font-semibold text-white/95">Ainda não tens bilhetes ORYA</h3>
      <p className="text-[12px] text-white/70 max-w-sm">
        A tua jornada na ORYA começa aqui. Explora eventos, música, festas e experiências novas – e guarda os teus
      </p>
    </div>
  )}

```

```

bilhetes com estilo.
    </p>
    <Link
      href="/explorar"
      className="inline-flex mt-2 px-4 py-2.5 rounded-xl bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5]
text-xs font-semibold text-black shadow-[0_0_28px_rgba(107,255,255,0.5)] hover:scale-[1.05] active:scale-95 transition-
transform">
    >
      Explorar eventos
    </Link>
  </div>
)}

{!ticketsLoading && !ticketsError && tickets.length > 0 && (
  <div className="space-y-3">
    <p className="text-[11px] text-white/60">
      Últimos bilhetes comprados. Em breve vais conseguir ver aqui a tua timeline completa de eventos.
    </p>
    <div className="flex gap-3 overflow-x-auto pb-1 pt-0.5">
      {recentTickets.map((t) => {
        const eventDate = parseDate(t.eventStartDate) ?? parseDate(t.createdAt);
        const dateLabel = eventDate
          ? eventDate.toLocaleString("pt-PT", {
              day: "2-digit",
              month: "short",
              hour: "2-digit",
              minute: "2-digit",
            })
          : "Data a anunciar";

        const quantityLabel = t.quantity > 1 ? `${t.quantity} bilhetes` : "1 bilhete";
        const totalEuros = (t.pricePaid / 100).toFixed(2);

        return (
          <article
            key={t.id}
            className="group flex min-w-[260px] max-w-[280px] flex-col overflow-hidden rounded-2xl border border-
white/15 bg-white/[0.03] backdrop-blur-xl shadow-[0_16px_45px_rgba(0,0,0,0.85)]">
            >
              <div className="relative h-32 w-full overflow-hidden">
                {t.eventCoverImageUrl ? (
                  <Image
                    src={optimizeImageUrl(t.eventCoverImageUrl, 600, 70)}
                    alt={t.eventTitle}
                    fill
                    sizes="(max-width: 640px) 90vw, 320px"
                    className="object-cover transition-transform duration-500 group-hover:scale-[1.03]"
                    placeholder="blur"
                    blurDataURL={defaultBlurDataURL}
                  />
                ) : (
                  <div className="flex h-full w-full items-center justify-center bg-gradient-to-br from-[#FF00C8] via-
[#6BFFFF] to-[#1646F5] text-[11px] font-semibold text-black/80">
                    Bilhete ORYA
                  </div>
                )}
              <div className="pointer-events-none absolute inset-0 bg-gradient-to-t from-black/85 via-black/30 to-
transparent" />
              <div className="absolute inset-x-3 bottom-2 space-y-0.5">
                <p className="text-[10px] uppercase tracking-[0.16em] text-white/65">Bilhete ORYA</p>
                <h3 className="text-sm font-semibold leading-snug line-clamp-2">{t.eventTitle}</h3>
                <p className="text-[11px] text-white/80">{dateLabel}</p>
              </div>
            </div>
          </article>
        )
      )
    </div>
    <div className="flex items-center justify-between gap-3 px-3 py-2.5 text-[11px]">
      <div className="space-y-0.5">
        <p className="text-white/65">{quantityLabel}</p>
        <p className="text-white font-medium">{totalEuros} €</p>
        <span className="inline-flex mt-0.5 items-center rounded-full bg-white/5 px-2 py-0.5 text-
white/75">
          {t.ticketName}
        </span>
      </div>
      {t.qrToken && (
        <Image
          src={`/api/qr/${t.qrToken}`}
          alt="QR Code do bilhete ORYA"
        </Image>
      )}
    </div>
  </div>
)
}

```

```

        width={48}
        height={48}
        className="h-12 w-12 rounded-lg bg-black/20 p-1 object-cover"
        sizes="48px"
        placeholder="blur"
        blurDataURL={defaultBlurDataURL}
      />
    )}
</div>
</article>
);
)}
</div>
</div>
)
}
</section>
</section>
</main>
);
}
}

```

app/me/settings/page.tsx

```

"use client";

import Link from "next/link";
import { useRouter } from "next/navigation";
import { useUser } from "@/app/hooks/useUser";
import { useState, useEffect } from "react";

export default function SettingsPage() {
  const { user, profile, isLoading, error, mutate } = useUser();
  const router = useRouter();

  const [email, setEmail] = useState("");
  const [visibility, setVisibility] = useState<"PUBLIC" | "PRIVATE">("PUBLIC");
  const [allowEmailNotifications, setAllowEmailNotifications] = useState(true);
  const [allowEventReminders, setAllowEventReminders] = useState(true);
  const [allowFriendRequests, setAllowFriendRequests] = useState(true);
  const [allowSalesAlerts, setAllowSalesAlerts] = useState(true);
  const [allowSystemAnnouncements, setAllowSystemAnnouncements] = useState(true);

  const [savingSettings, setSavingSettings] = useState(false);
  const [savingEmail, setSavingEmail] = useState(false);
  const [deleting, setDeleting] = useState(false);
  const [logoutLoading, setLogoutLoading] = useState(false);
  const [feedback, setFeedback] = useState<string | null>(null);
  const [errorMsg, setErrorMsg] = useState<string | null>(null);
  const [showDeleteConfirm, setShowDeleteConfirm] = useState(false);
  const [deleteConfirmText, setDeleteConfirmText] = useState("");

  useEffect(() => {
    if (user?.email) setEmail(user.email);
    if (profile?.visibility) setVisibility(profile.visibility as "PUBLIC" | "PRIVATE");
    if (typeof profile?.allowEmailNotifications === "boolean") {
      setAllowEmailNotifications(profile.allowEmailNotifications);
    }
    if (typeof profile?.allowEventReminders === "boolean") {
      setAllowEventReminders(profile.allowEventReminders);
    }
    if (typeof profile?.allowFriendRequests === "boolean") {
      setAllowFriendRequests(profile.allowFriendRequests);
    }
  }, [profile?.allowEmailNotifications, profile?.allowEventReminders, profile?.allowFriendRequests, profile?.visibility, user?.email]);

  useEffect(() => {
    let cancelled = false;
    async function loadPrefs() {
      try {
        const res = await fetch("/api/notifications/prefs");
        const json = await res.json().catch(() => null);
        if (!cancelled && res.ok && json?.prefs) {
          setAllowEmailNotifications(Boolean(json.prefs.allowEmailNotifications));
          setAllowEventReminders(Boolean(json.prefs.allowEventReminders));
          setAllowFriendRequests(Boolean(json.prefs.allowFriendRequests));
        }
      } catch {}
    }
  });
}

```

```

        setAllowSalesAlerts(Boolean(json.prefs.allowSalesAlerts));
        setAllowSystemAnnouncements(Boolean(json.prefs.allowSystemAnnouncements));
    }
} catch (err) {
    console.warn("[settings] load prefs failed", err);
}
}

if (user) loadPrefs();
return () => {
    cancelled = true;
};

}, [user]);

async function handleSaveSettings() {
    if (!user) return;
    setSavingSettings(true);
    setFeedback(null);
    setErrorMsg(null);

    try {
        const res = await fetch("/api/me/settings/save", {
            method: "PATCH",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                visibility,
                allowEmailNotifications,
                allowEventReminders,
                allowFriendRequests,
                allowSalesAlerts,
                allowSystemAnnouncements,
            }),
        });
        const json = await res.json();
        if (!res.ok || !json.ok) {
            throw new Error(json.error || "Erro ao guardar definições.");
        }
        // Sincronizar prefs de notificações
        await fetch("/api/notifications/prefs", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                allowEmailNotifications,
                allowEventReminders,
                allowFriendRequests,
                allowSalesAlerts,
                allowSystemAnnouncements,
            }),
        });
    }

    setFeedback("Definições guardadas.");
    await mutate();
} catch (err) {
    console.error(err);
    setErrorMsg("Não foi possível guardar as definições.");
} finally {
    setSavingSettings(false);
}

}

async function handleEmailUpdate() {
    if (!email.trim()) {
        setErrorMsg("Indica um email válido.");
        return;
    }

    setSavingEmail(true);
    setFeedback(null);
    setErrorMsg(null);

    try {
        const res = await fetch("/api/me/settings/email", {
            method: "PATCH",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ email }),
        });
        const json = await res.json();
        if (!res.ok || !json.ok) {
            throw new Error(json.error || "Erro ao atualizar email.");
        }
    }
}

```

```

        }
        setFeedback(json.message || "Email atualizado.");
        await mutate();
    } catch (err) {
        console.error(err);
        setErrorMsg(err instanceof Error ? err.message : "Não foi possível atualizar o email.");
    } finally {
        setSavingEmail(false);
    }
}

async function handleLogout() {
    setLogoutLoading(true);
    setFeedback(null);
    setErrorMsg(null);
    try {
        await fetch("/api/auth/logout", { method: "POST" });
    } catch (err) {
        console.error("Erro no logout:", err);
    } finally {
        setLogoutLoading(false);
        router.push("/login");
    }
}

async function handleDeleteAccount() {
    if (deleteConfirmText.trim().toUpperCase() !== "APAGAR CONTA") {
        setErrorMsg("Para confirmar, escreve: APAGAR CONTA");
        return;
    }
    setDeleting(true);
    setErrorMsg(null);
    setFeedback(null);
    try {
        const res = await fetch("/api/me/settings/delete", { method: "POST" });
        const json = await res.json();
        if (!res.ok || !json.ok) {
            throw new Error(json?.error || "Erro ao apagar conta.");
        }
        setFeedback(json.message || "Conta marcada para eliminação. Podes reverter dentro de 30 dias.");
        setShowDeleteConfirm(false);
        router.push("/login?pending_delete=1");
    } catch (err) {
        console.error(err);
        setErrorMsg("Não foi possível marcar a eliminação. Tenta mais tarde.");
    } finally {
        setDeleting(false);
    }
}

if (isLoading) {
    return (
        <main className="orya-body-bg min-h-screen text-white">
            <div className="max-w-4xl mx-auto px-5 py-10">
                <div className="h-36 rounded-2xl border border-white/10 bg-white/5 animate-pulse blur-[0.2px]" />
            </div>
        </main>
    );
}

if (error || !user) {
    return (
        <main className="orya-body-bg min-h-screen text-white">
            <div className="max-w-3xl mx-auto px-5 py-10 space-y-4">
                <h1 className="text-xl font-semibold">Definições</h1>
                <p className="text-sm text-white/70">
                    Precisas de iniciar sessão para aceder às definições da conta.
                </p>
                <Link href="/login?redirectTo=/me/settings"
                    className="inline-flex items-center gap-2 rounded-full bg-white text-black px-4 py-2 text-sm font-semibold"
                >
                    Entrar
                </Link>
            </div>
        </main>
    );
}

```

```

return (
  <main className="orya-body-bg min-h-screen text-white">
    <section className="max-w-4xl mx-auto px-5 py-10 space-y-6">
      <header className="flex flex-col gap-2">
        <h1 className="text-2xl font-semibold">Definições</h1>
        <p className="text-sm text-white/70">
          Zona segura para gerir email, privacidade, notificações e terminar sessão.
        </p>
      </header>

      {errorMsg && (
        <div className="rounded-xl border border-red-500/40 bg-red-500/10 px-4 py-3 text-sm text-red-100">
          {errorMsg}
        </div>
      )}
      {feedback && (
        <div className="rounded-xl border border-emerald-500/40 bg-emerald-500/10 px-4 py-3 text-sm text-emerald-100">
          {feedback}
        </div>
      )}
    </section>

    <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
      <section className="rounded-2xl border border-white/12 bg-white/[0.04] backdrop-blur-xl p-5 space-y-3">
        <div className="space-y-1">
          <h2 className="text-sm font-semibold text-white/90">Email</h2>
          <p className="text-xs text-white/60">
            Atualiza o email de login. O Supabase pode pedir confirmação.
          </p>
        </div>
        <div className="space-y-2">
          <input
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="w-full rounded-xl bg-black/40 border border-white/15 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]/60"
          />
          <button
            type="button"
            onClick={handleEmailUpdate}
            disabled={savingEmail}
            className="inline-flex items-center justify-center rounded-full bg-white text-black px-4 py-2 text-sm font-semibold shadow-[0_0_20px_rgba(255,255,255,0.3)] hover:scale-[1.01] active:scale-[0.99] transition disabled:opacity-60"
          >
            {savingEmail ? "A atualizar..." : "Atualizar email"}
          </button>
        </div>
      </section>

      <section className="rounded-2xl border border-white/12 bg-white/[0.04] backdrop-blur-xl p-5 space-y-4">
        <div className="space-y-1">
          <h2 className="text-sm font-semibold text-white/90">Privacidade</h2>
          <p className="text-xs text-white/60">Controla quem pode ver o teu perfil.</p>
        </div>
        <div className="flex flex-col gap-2 text-sm text-white/80">
          <label className="inline-flex items-center gap-2">
            <input
              type="radio"
              name="visibility"
              value="PUBLIC"
              checked={visibility === "PUBLIC"}
              onChange={() => setVisibility("PUBLIC")}
              className="h-3 w-3 accent-[#6BFFFF]"
            />
            <span>Perfil público</span>
          </label>
          <label className="inline-flex items-center gap-2">
            <input
              type="radio"
              name="visibility"
              value="PRIVATE"
              checked={visibility === "PRIVATE"}
              onChange={() => setVisibility("PRIVATE")}
              className="h-3 w-3 accent-[#FF00C8]"
            />
            <span>Perfil privado (mostra só avatar, nome e username)</span>
          </label>
        </div>
      </section>
    </div>
  </main>
)

```

```

        </div>
    </section>
</div>

<section className="rounded-2xl border border-white/12 bg-white/[0.04] backdrop-blur-xl p-5 space-y-3">
    <div className="space-y-1">
        <h2 className="text-sm font-semibold text-white/90">Notificações</h2>
        <p className="text-xs text-white/60">Controla emails e alertas relevantes.</p>
    </div>
    <div className="flex flex-col gap-2 text-sm text-white/80">
        <label className="inline-flex items-center gap-2">
            <input
                type="checkbox"
                checked={allowEmailNotifications}
                onChange={(e) => setAllowEmailNotifications(e.target.checked)}
                className="h-3 w-3 accent-[#6BFFFF]" />
            <span>Email de novidades e segurança</span>
        </label>
        <label className="inline-flex items-center gap-2">
            <input
                type="checkbox"
                checked={allowEventReminders}
                onChange={(e) => setAllowEventReminders(e.target.checked)}
                className="h-3 w-3 accent-[#6BFFFF]" />
            <span>Lembretes de eventos / experiências</span>
        </label>
        <label className="inline-flex items-center gap-2">
            <input
                type="checkbox"
                checked={allowFriendRequests}
                onChange={(e) => setAllowFriendRequests(e.target.checked)}
                className="h-3 w-3 accent-[#6BFFFF]" />
            <span>Pedidos de amizade / convites</span>
        </label>
        <label className="inline-flex items-center gap-2">
            <input
                type="checkbox"
                checked={allowSalesAlerts}
                onChange={(e) => setAllowSalesAlerts(e.target.checked)}
                className="h-3 w-3 accent-[#6BFFFF]" />
            <span>Alertas de vendas / estado Stripe</span>
        </label>
        <label className="inline-flex items-center gap-2">
            <input
                type="checkbox"
                checked={allowSystemAnnouncements}
                onChange={(e) => setAllowSystemAnnouncements(e.target.checked)}
                className="h-3 w-3 accent-[#6BFFFF]" />
            <span>Anúncios do sistema / updates críticos</span>
        </label>
    </div>
    <button
        type="button"
        onClick={handleSaveSettings}
        disabled={savingSettings}
        className="mt-3 inline-flex items-center justify-center rounded-full bg-white text-black px-4 py-2 text-sm font-semibold shadow-[0_0_20px_rgba(255,255,255,0.3)] hover:scale-[1.01] active:scale-[0.99] transition disabled:opacity-60">
        >
        {savingSettings ? "A guardar..." : "Guardar definições"}
    </button>
</section>

<section className="rounded-2xl border border-white/12 bg-white/[0.04] backdrop-blur-xl p-5 space-y-3">
    <div className="space-y-1">
        <h2 className="text-sm font-semibold text-white/90">Sessão e conta</h2>
        <p className="text-xs text-white/60">
            Termina sessão ou marca a tua conta para eliminação. Tens 30 dias para reverter; depois desse prazo, a conta é anonimizada.
        </p>
    </div>
    <div className="flex flex-wrap gap-2">
        <button
            type="button"

```

```

        onClick={handleLogout}
        disabled={logoutLoading}
        className="inline-flex items-center gap-2 rounded-full border border-white/20 px-4 py-2 text-sm text-white/80
        hover:bg-white/10 transition disabled:opacity-60"
      >
    {logoutLoading ? "A terminar sessão..." : "Terminar sessão"}
  </button>
  <button
    type="button"
    onClick={() => {
      setShowDeleteConfirm(true);
      setDeleteConfirmText("");
      setFeedback(null);
      setErrorMsg(null);
    }}
    disabled={deleting}
    className="inline-flex items-center gap-2 rounded-full border border-red-400/40 bg-red-500/10 px-4 py-2 text-sm
    text-red-100 hover:bg-red-500/20 transition disabled:opacity-60"
  >
    {deleting ? "A apagar..." : "Apagar conta"}
  </button>
</div>
</div>
</section>

<div className="flex items-center gap-2">
  <Link
    href="/me"
    className="inline-flex items-center gap-2 rounded-full border border-white/20 px-4 py-2 text-sm text-white/80
    hover:bg-white/10 transition"
  >
    ← Voltar à conta
  </Link>
</div>
</div>
</section>

{showDeleteConfirm && (
  <div className="fixed inset-0 z-50 flex items-center justify-center px-4">
    <div className="absolute inset-0 bg-black/70 backdrop-blur-md" />
    <div className="relative w-full max-w-md rounded-2xl border border-white/12 bg-gradient-to-b from-[#0b0b13]/95 to-
    [&#0b0b13]/90 p-6 shadow-2xl">
      <div className="flex items-start justify-between gap-3">
        <div>
          <p className="text-xs uppercase tracking-[0.2em] text-white/40">Zona segura</p>
          <h3 className="mt-1 text-lg font-semibold text-white">Apagar conta ORYA</h3>
        </div>
        <button
          type="button"
          className="h-8 w-8 rounded-full border border-white/10 text-white/70 hover:bg-white/10 transition"
          onClick={() => {
            setShowDeleteConfirm(false);
            setDeleteConfirmText("");
          }}
        >
          {deleting}
          aria-label="Fechar confirmação"
        </button>
        <button
          type="button"
          className="h-8 w-8 rounded-full border border-white/10 text-white/70 hover:bg-white/10 transition"
          onClick={() => {
            setShowDeleteConfirm(false);
            setDeleteConfirmText("");
          }}
        >
          X
        </button>
      </div>
    </div>
  </div>
  <p className="mt-3 text-sm text-white/70 leading-relaxed">
    Vamos marcar a tua conta para eliminação e desativá-la de imediato. Tens 30 dias para reativar fazendo login ou
    clicando no link do email de cancelamento. Após esse prazo, os dados pessoais são anonimizados.
    Para continuares, escreve <span className="font-semibold text-white">APAGAR CONTA</span> e confirma.
  </p>
  <div className="mt-4 space-y-2">
    <label className="text-xs text-white/60" htmlFor="delete-confirm-input">
      Confirmação
    </label>
    <input
      id="delete-confirm-input"
      value={deleteConfirmText}
      onChange={(e) => setDeleteConfirmText(e.target.value)}
      placeholder="APAGAR CONTA"
      className="w-full rounded-xl bg-black/40 border border-white/15 px-3 py-2 text-sm outline-none focus:border-
      [&#FF4D8F] focus:ring-1 focus:ring-[#FF4D8F]/50 text-white placeholder:text-white/30"
      disabled={deleting}
    /&gt;
  &lt;/div&gt;
)
</pre>
</div>
```

```

        {deleteConfirmText.length > 0 && deleteConfirmText.trim().toUpperCase() !== "APAGAR CONTA" && (
            <p className="text-xs text-red-300">Escreve exatamente: APAGAR CONTA.</p>
        )}
    </div>

    <div className="mt-5 flex flex-col-reverse gap-2 sm:flex-row sm:justify-end">
        <button
            type="button"
            className="w-full sm:w-auto rounded-full border border-white/15 px-4 py-2 text-sm text-white/80 hover:bg-white/10 transition disabled:opacity-50"
            onClick={() => {
                setShowDeleteConfirm(false);
                setDeleteConfirmText("");
            }}
            disabled={deleting}
        >
            Cancelar
        </button>
        <button
            type="button"
            onClick={handleDeleteAccount}
            disabled={deleting || deleteConfirmText.trim().toUpperCase() !== "APAGAR CONTA"}
            className="w-full sm:w-auto inline-flex items-center justify-center gap-2 rounded-full border border-red-400/50 bg-gradient-to-r from-[#FF2A6F] to-[#FF6B2A] px-4 py-2 text-sm font-semibold text-white shadow-[0_0_30px_rgba(255,77,143,0.35)] hover:translate-y-[-1px] active:translate-y-[0px] transition disabled:opacity-60"
        >
            {deleting ? "A apagar..." : "Confirmar eliminação"}
        </button>
    </div>
</div>
</div>
</main>
);
}
}

```

app/me/tickets/page.tsx

```

"use client";

import { useEffect, useState } from "react";
import { useRouter } from "next/navigation";
import Link from "next/link";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";
import { useUser } from "@/app/hooks/useUser";

type ResaleMode = "ALWAYS" | "AFTER SOLD_OUT" | "DISABLED";

type TicketFromApi = {
    id: string;
    pricePaid?: number | null; // céntimos
    grossCents?: number | null;
    discountCents?: number | null;
    platformFeeCents?: number | null;
    netCents?: number | null;
    currency?: string | null;
    purchasedAt: string;
    badge?: "FREE" | "RESALE" | "SPLIT" | "FULL" | "SINGLE";
    nextAction?: "NONE" | "PAY_PARTNER" | "CONFIRM_GUARANTEE";
    event?: {
        id?: number | null;
        slug?: string | null;
        title?: string | null;
        startDate?: string | null;
        locationName?: string | null;
        coverImageUrl?: string | null;
        resaleMode?: ResaleMode | null;
        isSoldOut?: boolean | null;
    } | null;
    ticket?: {
        id?: string | null;
        name?: string | null;
        description?: string | null;
    } | null;
    qrToken?: string | null;
    resaleId?: string | null;
}

```

```

resaleStatus?: "LISTED" | "SOLD" | "CANCELLED" | null;
resalePrice?: number | null; // céntimos
resaleCurrency?: string | null;
};

type TicketsApiResponse = {
  success: boolean;
  tickets: TicketFromApi[];
};

type UITicket = {
  id: string;
  eventId: number;
  ticketTypeId: string;
  priceCents: number;
  grossCents: number;
  discountCents: number;
  platformFeeCents: number;
  netCents: number;
  priceEur: number;
  promoLabel?: string | null;
  currency: string;
  createdAt: string;
  badge: TicketFromApi["badge"];
  nextAction: TicketFromApi["nextAction"];
  qrToken: string | null;
  resaleId?: string | null;
  resaleStatus?: "LISTED" | "SOLD" | "CANCELLED" | null;
  resalePriceCents?: number | null;
  resaleCurrency?: string | null;
};

type UITicketGroup = {
  key: string;
  quantity: number;
  totalPaidEur: number;
  totalGrossEur: number;
  totalDiscountEur: number;
  promoLabel?: string | null;
  currency: string;
  badge?: TicketFromApi["badge"];
  nextAction?: TicketFromApi["nextAction"];
  event: {
    id: number;
    slug: string;
    title: string;
    startDate: string;
    locationName: string;
    coverImageUrl?: string | null;
    resaleMode: ResaleMode;
    isSoldOut: boolean;
  };
  ticket: {
    id: string;
    name: string;
    description?: string | null;
  };
  tickets: UITicket[];
};

export default function MyTicketsPage() {
  const { profile } = useUser();
  const router = useRouter();
  const { openModal } = useAuthModal();

  const [loading, setLoading] = useState(true);
  const [errorMsg, setErrorMsg] = useState<string | null>(null);
  const [ticketGroups, setTicketGroups] = useState<UITicketGroup[]>([]);
  const [expandedGroupKeys, setExpandedGroupKeys] = useState<Set<string>>(new Set());
  const [activeTab, setActiveTab] = useState<"upcoming" | "past" | "all">("upcoming");

  const [transferModal, setTransferModal] = useState<{
    open: boolean;
    ticketId: string | null;
    ticketTitle: string | null;
  }>({
    open: false,
    ticketId: null,
  });
}

```

```

    ticketTitle: null,
  });

const [transferUsername, setTransferUsername] = useState("");
const [transferLoading, setTransferLoading] = useState(false);
const [transferError, setTransferError] = useState<string | null>(null);
const [transferSuccess, setTransferSuccess] = useState<string | null>(null);

const [resaleModal, setResaleModal] = useState<{
  open: boolean;
  ticketId: string | null;
  ticketTitle: string | null;
}>({
  open: false,
  ticketId: null,
  ticketTitle: null,
});

const [resalePrice, setResalePrice] = useState("");
const [resaleLoading, setResaleLoading] = useState(false);
const [resaleError, setResaleError] = useState<string | null>(null);
const [resaleSuccess, setResaleSuccess] = useState<string | null>(null);
const [toasts, setToasts] = useState<{
  id: number;
  type: "error" | "success";
  message: string
}[]>([]);

const isAdmin = Array.isArray(profile?.roles) && profile.roles.includes("admin");
useEffect(() => {
  setExpandedGroupKeys((prev) => {
    if (!prev.size) return prev;
    const valid = new Set<string>();
    ticketGroups.forEach((g) => {
      if (prev.has(g.key)) valid.add(g.key);
    });
    if (valid.size === prev.size && Array.from(prev).every((k) => valid.has(k))) {
      return prev;
    }
    return valid;
  });
}, [ticketGroups]);

function pushToast(message: string, type: "error" | "success" = "error") {
  const id = Date.now() + Math.random();
  setToasts((prev) => [...prev, { id, type, message }]);
  setTimeout(() => {
    setToasts((prev) => prev.filter((t) => t.id !== id));
  }, 4000);
}

useEffect(() => {
  let cancelled = false;

  async function loadTickets() {
    try {
      setLoading(true);
      setErrorMsg(null);

      const res = await fetch("/api/me/tickets", {
        method: "GET",
        headers: {
          "Content-Type": "application/json",
        },
        cache: "no-store",
      });

      if (res.status === 401) {
        // Não autenticado → abre modal de autenticação e define redirect para esta página
        openModal({ mode: "login", redirectTo: "/me/tickets", showGoogle: true });
        return;
      }

      if (!res.ok) {
        const text = await res.text();
        console.error(`Erro ao carregar bilhetes: ${text}`);
        if (!cancelled) {
          setErrorMsg(
            "Não foi possível carregar os teus bilhetes. Tenta novamente em alguns segundos."
          );
        }
      }
    }
  }
}

```

```

}

const json = (await res.json()) as TicketsApiResponse;

if (!json.success || !Array.isArray(json.tickets)) {
  if (!cancelled) {
    setErrorMsg("Resposta inesperada ao tentar carregar os bilhetes.");
  }
  return;
}

const apiTicketMap = new Map<string, TicketFromApi>(
  json.tickets.map((t: TicketFromApi) => [t.id, t])
);

// Normalizar tickets individuais (1 entrada = 1 registo)
const singles: UITicket[] = json.tickets.map((p: TicketFromApi) => {
  const priceCents = Number(p.pricePaid ?? 0);
  const grossCents = Number(p.grossCents ?? priceCents);
  const discountCents = Number(p.discountCents ?? 0);
  const platformFeeCents = Number(p.platformFeeCents ?? 0);
  const netCents = Number(p.netCents ?? priceCents);
  const eventId = Number(p.event?.id ?? -1);
  const ticketTypeId = (p.ticket?.id ?? "").toString();

  return {
    id: p.id,
    eventId,
    ticketTypeId,
    priceCents,
    grossCents,
    discountCents,
    platformFeeCents,
    netCents,
    priceEur: priceCents / 100,
    currency: p.currency ?? "EUR",
    createdAt: p.purchasedAt,
    badge: p.badge,
    nextAction: p.nextAction,
    qrToken: p.qrToken ?? null,
    resaleId: p.resaleId ?? null,
    resaleStatus: (p.resaleStatus as "LISTED" | "SOLD" | "CANCELLED" | null) ?? null,
    resalePriceCents: p.resalePrice ?? null,
    resaleCurrency: p.resaleCurrency ?? null,
  };
});

// Agrupar por evento + tipo de bilhete para o cartão principal,
// mas manter cada QR individual dentro do grupo.
const groupsMap = new Map<string, UITicketGroup>();

for (const single of singles) {
  const source = apiTicketMap.get(single.id) as TicketFromApi | undefined;
  const event = source?.event ?? {};
  const ticket = source?.ticket ?? {};
  const badge = source?.badge;
  const nextAction = source?.nextAction;

  const eventData = {
    id: Number(event.id ?? single.eventId ?? -1),
    slug: event.slug ?? "",
    title: event.title ?? "Evento ORYA",
    startDate: event.startDate ?? single.createdAt,
    locationName: event.locationName ?? "Local a anunciar",
    coverImageUrl: event.coverImageUrl ?? null,
    resaleMode: (event.resaleMode as ResaleMode | undefined) ?? "ALWAYS",
    isSoldOut: Boolean(event.isSoldOut),
  };

  const ticketData = {
    id: ticket.id ? String(ticket.id) : "wave",
    name: ticket.name ?? "Bilhete",
    description: ticket.description ?? null,
  };

  const eventKey = eventData.id > 0 ? String(eventData.id) : eventData.slug;
  const key = `${eventKey}-${ticketData.id}`;
}

```

```

    if (!groupsMap.has(key)) {
      groupsMap.set(key, {
        key,
        quantity: 1,
        totalPaidEur: single.priceEur,
        totalGrossEur: single.grossCents / 100,
        totalDiscountEur: single.discountCents / 100,
        promoLabel: single.promoLabel ?? null,
        currency: single.currency,
        badge,
        nextAction,
        event: eventData,
        ticket: ticketData,
        tickets: [single],
      });
    } else {
      const existing = groupsMap.get(key)!";
      existing.quantity += 1;
      existing.totalPaidEur += single.priceEur;
      existing.totalGrossEur += single.grossCents / 100;
      existing.totalDiscountEur += single.discountCents / 100;
      if (!existing.promoLabel && single.promoLabel) existing.promoLabel = single.promoLabel;
      if (!existing.badge && badge) existing.badge = badge;
      if (existing.nextAction === undefined && nextAction) existing.nextAction = nextAction;
      existing.tickets.push(single);
    }
  }

  const mapped = Array.from(groupsMap.values());

  // Ordenar por data do evento (mais próximo primeiro)
  mapped.sort((a, b) => {
    const da = new Date(a.event.startDate).getTime();
    const db = new Date(b.event.startDate).getTime();
    return da - db;
  });

  if (!cancelled) {
    setTicketGroups(mapped);
  }
} catch (err) {
  console.error("Erro inesperado ao carregar bilhetes:", err);
  if (!cancelled) {
    setErrorMsg(
      "Ocorreu um problema inesperado. Tenta novamente dentro de instantes."
    );
  }
} finally {
  if (!cancelled) {
    setLoading(false);
  }
}
}

loadTickets();

return () => {
  cancelled = true;
};

}, [router, openModal]);

const hasTickets = ticketGroups.length > 0;

const now = new Date();
const rotationWindow = Math.floor(now.getTime() / 15000);

const totalCount = ticketGroups.reduce((sum, group) => sum + group.quantity, 0);

const upcomingCount = ticketGroups.reduce((sum, group) => {
  const date = group.event?.startDate ? new Date(group.event.startDate) : null;
  if (!date || Number.isNaN(date.getTime())) return sum;
  return date >= now ? sum + group.quantity : sum;
}, 0);

const pastCount = ticketGroups.reduce((sum, group) => {
  const date = group.event?.startDate ? new Date(group.event.startDate) : null;
  if (!date || Number.isNaN(date.getTime())) return sum;
  return date < now ? sum + group.quantity : sum;
}

```

```

}, 0);

const filteredGroups = ticketGroups.filter((group) => {
  const date = group.event?.startDate ? new Date(group.event.startDate) : null;

  if (!date || Number.isNaN(date.getTime())) {
    return activeTab === "all";
  }

  if (activeTab === "upcoming") {
    return date >= now;
  }

  if (activeTab === "past") {
    return date < now;
  }

  if (activeTab === "all") return true;
  return true;
});

function formatDate(dateStr: string) {
  if (!dateStr) return "";
  const d = new Date(dateStr);
  if (Number.isNaN(d.getTime())) return "";
  return new Intl.DateTimeFormat("pt-PT", {
    day: "2-digit",
    month: "short",
    year: "numeric",
    hour: "2-digit",
    minute: "2-digit",
  }).format(d);
}

function formatPrice(amount: number, currency: string) {
  const safeCurrency = currency || "EUR";
  return new Intl.NumberFormat("pt-PT", {
    style: "currency",
    currency: safeCurrency,
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  }).format(amount);
}

function openTransferModalForTicket(ticketId: string, ticketTitle: string) {
  setTransferModal({
    open: true,
    ticketId,
    ticketTitle,
  });
  setTransferUsername("");
  setTransferError(null);
  setTransferSuccess(null);
}

async function handleConfirmTransfer() {
  if (!transferModal.ticketId) return;

  const username = transferUsername.trim();
  if (!username) {
    setTransferError("Indica o username ORYA do teu amigo.");
    setTransferSuccess(null);
    return;
  }

  try {
    setTransferLoading(true);
    setTransferError(null);
    setTransferSuccess(null);

    const res = await fetch("/api/tickets/transfer/start", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        ticketId: transferModal.ticketId,
        targetIdentifier: username,
      })
    });
    if (res.ok) {
      setTransferSuccess("Transferência realizada com sucesso!");
      setTransferError(null);
      setTransferLoading(false);
    } else {
      const error = await res.json();
      setTransferError(error.message);
      setTransferSuccess(null);
      setTransferLoading(false);
    }
  } catch (error) {
    console.error("Erro ao confirmar transferência:", error);
    setTransferError("Ocorreu um erro ao confirmar a transferência.");
    setTransferSuccess(null);
  }
}

```

```

    }),

    const data = await res.json().catch(() => null);

    if (!res.ok || !data?.ok) {
      const code = (data?.error || data?.reason || "").toString();
      const friendly =
        code === "CANNOT_TRANSFER_TO_SELF"
          ? "Não podes enviar um bilhete para ti próprio."
          : code === "TARGET_NOT_FOUND"
          ? "Não encontrámos esse username. Confirma o @ do teu amigo."
          : code === "TRANSFER_ALREADY_PENDING"
          ? "Já tens uma transferência pendente para este bilhete."
          : "Não foi possível iniciar a transferência. Tenta novamente.";
      const reason = friendly;
      setTransferError(reason);
      pushToast(reason, "error");
      setTransferSuccess(null);
      return;
    }

    setTransferSuccess(
      `Transferência enviada para @${username}. O teu amigo vai poder aceitar ou recusar.`);
    pushToast(`Transferência enviada para @${username}.`, "success");
  } catch (err) {
    console.error("Erro ao iniciar transferência:", err);
    setTransferError(
      "Ocorreu um erro inesperado ao iniciar a transferência. Tenta novamente dentro de instantes.");
  }
  setTransferSuccess(null);
  pushToast(
    "Ocorreu um erro inesperado ao iniciar a transferência. Tenta novamente.",
    "error"
  );
} finally {
  setTransferLoading(false);
}

function openResaleModalForTicket(ticketId: string, ticketTitle: string) {
  setResaleModal({
    open: true,
    ticketId,
    ticketTitle,
  });
  setResalePrice("");
  setResaleError(null);
  setResaleSuccess(null);
}

async function handleConfirmResale() {
  if (!resaleModal.ticketId) return;

  const targetGroup = ticketGroups.find((g) =>
    g.tickets.some((t) => t.id === resaleModal.ticketId)
  );
  if (targetGroup) {
    const resaleMode = targetGroup.event.resaleMode ?? "ALWAYS";
    const allowed =
      resaleMode === "ALWAYS" ||
      (resaleMode === "AFTER SOLD OUT" && targetGroup.event.isSoldOut);
    if (!allowed) {
      setResaleError(
        resaleMode === "DISABLED"
          ? "Este evento não permite revendas."
          : "A revenda fica disponível quando o evento estiver esgotado."
      );
      setResaleSuccess(null);
      return;
    }
  }

  const raw = resalePrice.replace(",", ".").trim();
  const value = Number(raw);
  if (!value || Number.isNaN(value) || value <= 0) {
    setResaleError("Indica um preço válido em euros (ex: 15 ou 15,50).");
  }
}

```

```

        setResaleSuccess(null);
        return;
    }

    const cents = Math.round(value * 100);

    try {
        setResaleLoading(true);
        setResaleError(null);
        setResaleSuccess(null);

        const res = await fetch("/api/tickets/resale/list", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({
                ticketId: resaleModal.ticketId,
                price: cents,
                priceCents: cents,
            }),
        });

        const data = await res.json().catch(() => null);

        if (!res.ok || !data?.ok) {
            const code = (data?.error || data?.reason || "").toString();
            const friendly =
                code === "RESALE_DISABLED_FOR_EVENT"
                    ? "Este evento não permite revendas."
                    : code === "RESALE_ONLY_AFTER SOLD_OUT"
                    ? "Só podes revender depois de o evento esgotar."
                    : code === "TICKET_ALREADY_IN_RESALE"
                    ? "Este bilhete já está listado."
                    : code === "TRANSFER_ALREADY_PENDING"
                    ? "Tens uma transferência pendente para este bilhete."
                    : "Não foi possível listar este bilhete à venda. Tenta novamente.";
            setResaleError(friendly);
            pushToast(friendly, "error");
            setResaleSuccess(null);
            return;
        }

        setResaleSuccess(
            "Bilhete colocado em revenda. Outros utilizadores já o podem comprar."
        );
        pushToast("Bilhete colocado em revenda.", "success");

        setTicketGroups((prev) =>
            prev.map((group) => {
                if (!group.tickets.some((t) => t.id === resaleModal.ticketId)) {
                    return group;
                }

                return {
                    ...group,
                    tickets: group.tickets.map((t) =>
                        t.id === resaleModal.ticketId
                            ? {
                                ...t,
                                resaleStatus: "LISTED",
                                resaleId: (data.resaleId as string | undefined) ?? t.resaleId ?? null,
                                resalePriceCents: cents,
                                resaleCurrency: t.currency,
                            }
                            : t
                    ),
                };
            })
        );
    } catch (err) {
        console.error("Erro ao criar revenda:", err);
        setResaleError(
            "Ocorreu um erro inesperado ao listar o bilhete. Tenta novamente dentro de instantes."
        );
        setResaleSuccess(null);
        pushToast(
            "Ocorreu um erro inesperado ao listar o bilhete. Tenta novamente dentro de instantes."
        );
    }
}

```

```

        "error"
    );
} finally {
    setResaleLoading(false);
}
}

async function handleCancelResale(resaleId: string, ticketId: string) {
    try {
        const res = await fetch("/api/tickets/resale/cancel", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ resaleId }),
        });

        const data = await res.json().catch(() => null);

        if (!res.ok || !data?.ok) {
            console.error("Erro ao cancelar revenda:", data);
            alert("Não foi possível cancelar esta revenda. Tenta novamente.");
            pushToast("Não foi possível cancelar esta revenda. Tenta novamente.", "error");
            return;
        }

        setTicketGroups((prev) =>
            prev.map((group) => {
                if (!group.tickets.some((t) => t.id === ticketId)) return group;

                return {
                    ...group,
                    tickets: group.tickets.map((t) =>
                        t.id === ticketId
                            ? { ...t, resaleStatus: null, resaleId: null, resalePriceCents: null }
                            : t
                    ),
                };
            });
        );
        pushToast("Revenda cancelada com sucesso.", "success");
    } catch (err) {
        console.error("Erro ao cancelar revenda:", err);
        alert(
            "Ocorreu um erro inesperado ao cancelar a revenda. Tenta novamente dentro de instantes."
        );
        pushToast(
            "Ocorreu um erro inesperado ao cancelar a revenda. Tenta novamente.",
            "error"
        );
    }
}

return (
    <>
    <div className="fixed top-4 right-4 z-[60] space-y-2">
        {toasts.map((toast) => (
            <div
                key={toast.id}
                className={`${`min-w-[240px] rounded-xl px-3 py-2 text-[11px] shadow-lg backdrop-blur`} ${toast.type === "success"
                    ? "bg-emerald-500/20 border border-emerald-400/50 text-emerald-50"
                    : "bg-red-500/15 border border-red-400/50 text-red-50"
                }`}
            >
                {toast.message}
            </div>
        )));
    </div>
<main
    aria-labelledby="my-tickets-title"
    className="orya-body-bg min-h-screen w-full text-white pb-16"
>
    /* Top bar */
<header className="border-b border-white/10 bg-black/40 backdrop-blur-xl">
    <div className="max-w-5xl mx-auto px-5 py-4 flex items-center justify-between">
        <div className="flex items-center gap-3">

```

```



```

```

        <div className="h-3 w-32 rounded-full bg-white/10" />
        <div className="grid grid-cols-2 gap-2">
          <div className="h-3 w-full rounded-full bg-white/10" />
          <div className="h-3 w-full rounded-full bg-white/10" />
        </div>
      </div>
    </div>
  </div>
}

{errorMsg && !loading && (
  <div className="mt-6 rounded-xl border border-red-500/40 bg-red-500/10 px-4 py-3 text-red-100 flex items-start gap-2" role="alert" aria-live="assertive">
    <span className="mt-[2px] text-sm">⚠</span>
    <div className="space-y-1">
      <p className="font-medium text-red-100">Não foi possível carregar</p>
      <p className="text-[11px] text-red-100/80">{errorMsg}</p>
    </div>
  </div>
)
}

 {!loading && !errorMsg && !hasTickets && (
  <div className="mt-8 rounded-2xl border border-dashed border-white/15 bg-white/3 px-6 py-8 text-center space-y-3">
    <p className="text-lg font-medium">Ainda não tens bilhetes</p>
    <p className="text-sm text-white/65 max-w-md mx-auto">
      Assim que comprares um bilhete através da ORYA, ele vai aparecer
      aqui – com acesso rápido ao evento, à informação e ao teu QR code.
    </p>
  </div>
)
}

{hasTickets && (
  <div className="mt-6 space-y-3">
    <div className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between">
      <div className="space-y-2 md:max-w-md">
        <h2 className="text-sm font-semibold text-white/85">
          Eventos a que vais com bilhetes ORYA
        </h2>
        <p className="text-[11px] text-white/60">
          Agrupados por próximos, passados e todos os bilhetes ligados à tua conta ORYA, para encontrares tudo em
        segundos.
        </p>
        <div className="flex flex-wrap gap-3 text-[10px] text-white/55">
          <span>
            <span className="font-semibold text-white/80">{upcomingCount}</span>{" "}
            próximos
          </span>
          <span>
            <span className="font-semibold text-white/80">{pastCount}</span>{" "}
            passados
          </span>
          <span>
            <span className="font-semibold text-white/80">{totalCount}</span>{" "}
            no total
          </span>
        </div>
      </div>
    </div>
  </div>
  <div className="inline-flex items-center rounded-full bg-white/5 p-1 text-[11px]">
    <button
      type="button"
      onClick={() => setActiveTab("upcoming")}
      className={`px-3 py-1.5 rounded-full font-medium transition-colors ${
        activeTab === "upcoming"
          ? "bg-white text-black"
          : "text-white/70 hover:text-white"
      }`}>
      Próximos ({upcomingCount})
    </button>
    <button
      type="button"
      onClick={() => setActiveTab("past")}
      className={`px-3 py-1.5 rounded-full font-medium transition-colors ${
        activeTab === "past"
          ? "bg-white text-black"
          : "text-white/70 hover:text-white"
      }`}>
      Passados ({pastCount})
    </button>
  </div>
)
}

```

```

        : "text-white/70 hover:text-white"
    }`}
>
    Passados ({pastCount})
</button>
<button
    type="button"
    onClick={() => setActiveTab("all")}
    className={`px-3 py-1.5 rounded-full font-medium transition-colors ${activeTab === "all" ? "bg-white text-black" : "text-white/70 hover:text-white"}`}
>
    Todos ({totalCount})
</button>
</div>
</div>

{filteredGroups.length === 0 ? (
    <p className="mt-2 text-xs text-white/60">
        Não há bilhetes nesta secção. Experimenta outra aba acima.
    </p>
) : (
    <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
        {filteredGroups.map((group) => {
            const event = group.event;
            const ticket = group.ticket;
            const dateLabel = formatDate(event.startDate);
            const totalLabel = formatPrice(group.totalPaidEur, group.currency);
            const unitGross =
                group.quantity > 0 && group.totalGrossEur !== undefined
                    ? group.totalGrossEur / group.quantity
                    : 0;
            const unitDiscount =
                group.quantity > 0 && group.totalDiscountEur !== undefined
                    ? group.totalDiscountEur / group.quantity
                    : 0;
            const eventDateObj = event.startDate ? new Date(event.startDate) : null;
            let statusLabel = "Confirmado";
            let statusClass =
                "border-emerald-400/50 bg-emerald-500/10 text-emerald-200";

            if (eventDateObj && !Number.isNaN(eventDateObj.getTime())) {
                const isPast = eventDateObj.getTime() < now.getTime();

                const isSameDay =
                    eventDateObj.getFullYear() === now.getFullYear() &&
                    eventDateObj.getMonth() === now.getMonth() &&
                    eventDateObj.getDate() === now.getDate();

                if (isPast) {
                    statusLabel = "Já aconteceu";
                    statusClass =
                        "border-white/25 bg-white/5 text-white/80";
                } else if (isSameDay) {
                    statusLabel = "É hoje";
                    statusClass =
                        "border-sky-400/60 bg-sky-500/10 text-sky-100";
                }
            }

            const isExpanded = expandedGroupKeys.has(group.key);
            const resaleMode = event.resaleMode ?? "ALWAYS";
            const resaleAllowed =
                resaleMode === "ALWAYS" ||
                (resaleMode === "AFTER SOLD_OUT" && event.isSoldOut);
            const resaleDisabledReason =
                resaleMode === "DISABLED"
                    ? "Revenda desativada pelo organizador."
                    : "A revenda fica disponível quando o evento estiver esgotado.";

            return (
                <div
                    key={group.key}
                    className="group relative overflow-hidden rounded-2xl border border-white/10 bg-gradient-to-b from-white/5 via-black/80 to-black/95 hover:border-[#6BFFFF]/70 transition-colors shadow-[0_16px_45px_rgba(0,0,0,0.75)]"
                >

```

```


<div className="h-12 w-12 rounded-lg bg-black/30 border border-white/10 flex items-center justify-center text-[11px] text-white/70 font-semibold">
    {group.quantity}
  </div>
  <div className="flex flex-col">
    <p className="text-white font-medium leading-tight">{event.title}</p>
    <p className="text-xs text-white/60 leading-tight">
      {ticket.name}
    </p>
  </div>
  <div className="ml-auto inline-flex items-center gap-1 text-[10px] text-white/60">
    <span
      className={`rounded-full border px-2 py-0.5 ${statusClass}`}
    >
      {statusLabel}
    </span>
    <span className="rounded-full border border-white/15 px-2 py-0.5 bg-white/5">
      {resaleMode === "DISABLED"
        ? "Revenda OFF"
        : resaleMode === "AFTER_SOLD_OUT"
        ? event.isSoldOut
        ? "Revenda ON (esgotado)"
        : "Revenda após esgotar"
        : "Revenda ON"}
    </span>
  </div>
</div>

/* Poster visual */
<div className="relative w-full overflow-hidden">
  <div className="relative aspect-[3/4] w-full">
    {event.coverImageUrl ? (
      // eslint-disable-next-line @next/next/no-img-element
      <img
        src={event.coverImageUrl}
        alt={event.title}
        className="h-full w-full object-cover transition-transform duration-500 group-hover:scale-[1.03]"
      />
    ) : (
      <div className="flex h-full w-full items-center justify-center bg-gradient-to-br from-[#1b1b2f] via-black to-[#141421] text-[11px] text-white/40">
        Sem imagem de capa
      </div>
    )}
  </div>
</div>

/* Overlay gradient + basic info sobre o poster */
<div className="pointer-events-none absolute inset-0 bg-gradient-to-t from-black/90 via-black/40 to-transparent" />

/* Badges no topo */
<div className="absolute left-2 right-2 top-2 flex items-center justify-between gap-2">
  <span
    className={`inline-flex items-center rounded-full border px-2 py-0.5 text-[10px] font-medium ${statusClass}`}
  >
    {statusLabel}
  </span>
  <span className="inline-flex items-center rounded-full bg-black/70 px-2 py-0.5 text-[10px] font-semibold text-white/85">
    {group.quantity > 1
      ? `${group.quantity} bilhetes`
      : "1 bilhete"}
  </span>
</div>

/* Título + data na parte de baixo do poster */
<div className="absolute inset-x-2 bottom-2 space-y-1">
  <p className="text-[11px] uppercase tracking-[0.16em] text-white/60">
    Bilhetes ORYA
  </p>
  <h3 className="text-sm font-semibold leading-snug line-clamp-2">
    {event.title}
  </h3>
  {event.locationName && (
    <p className="text-[11px] text-white/70 line-clamp-1">
      {event.locationName}
    </p>
  )}
</div>


```

```

        )}
        {dateLabel && (
            <p className="text-[11px] text-white/80">
                {dateLabel}
            </p>
        )}
    </div>
</div>
</div>

/* Zona de detalhes por baixo do poster */
<div className="border-t border-white/10 bg-black/80 px-4 py-3 space-y-2">
    <div className="grid grid-cols-2 gap-2 text-[11px]">
        <div className="space-y-1">
            <p className="text-white/50">Total pago</p>
            <p className="text-white/90">{totalLabel}</p>
        </div>
        <div className="space-y-1">
            <p className="text-white/50">Preço bilhete (bruto)</p>
            <p className="text-white/90">
                {formatPrice(unitGross, group.currency)}
            </p>
        </div>
        {unitDiscount > 0 && (
            <div className="space-y-1 col-span-2">
                <p className="text-white/50">Desconto aplicado {group.promoLabel} ? `(${group.promoLabel})` : "">
                <p className="text-emerald-300">
                    -{formatPrice(unitDiscount, group.currency)} por bilhete
                </p>
            </div>
        )}
        <div className="space-y-1">
            <p className="text-white/50">Tipo de bilhete</p>
            <p className="text-white/90 line-clamp-1">
                {ticket.name}
            </p>
        </div>
        <div className="space-y-1">
            <p className="text-white/50">Referência</p>
            <p className="text-white/80 text-[10px] break-all">
                {group.tickets[0].id}
                {group.quantity > 1 ? " (" + ")" : ""}
            </p>
        </div>
    </div>
</div>

<p className="mt-1 text-[10px] text-white/45">
    Os QR codes aparecem apenas ao abrir este cartão. Cada bilhete tem o seu QR único.
</p>

<div className="mt-3 flex flex-wrap items-center justify-between gap-2 text-[11px]">
    <div className="flex flex-wrap gap-2">
        <button
            type="button"
            onClick={() =>
                event.slug ? router.push(`/eventos/${event.slug}`) : null
            }
            className="inline-flex items-center gap-1 rounded-full border border-white/25 px-3 py-1 text-white/80 hover:bg-white/10 transition-colors"
        >
            Ver evento
        </button>
    </div>
    <button
        type="button"
        onClick={() => {
            setExpandedGroupKeys((prev) => {
                const next = new Set(prev);
                if (next.has(group.key)) {
                    next.delete(group.key);
                } else {
                    next.add(group.key);
                }
                return next;
            });
        }}
        className="inline-flex items-center gap-1 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF]"
    >

```

```

to-[#1646F5] px-3 py-1 font-semibold text-black shadow-[0_0_20px_rgba(107,255,255,0.7)] hover:scale-[1.03] active:scale-95
transition-transform"
    >
        {isExpanded ? "Fechar bilhetes" : `Ver bilhetes (${group.quantity})`}
    </button>
</div>

{isExpanded && (
<div className="mt-3 space-y-3">
    {group.tickets.map((t, idx) => {
        const isListed = t.resaleStatus === "LISTED" && t.resaleId;
        const ticketLabel = `${event.title} - ${ticket.name} (#${idx + 1})`;

        return (
            <div
                key={t.id}
                className="rounded-xl border border-white/12 bg-black/60 p-3 space-y-3"
            >
                <div className="flex items-center justify-between gap-2">
                    <div className="flex flex-col">
                        <p className="text-sm font-medium text-white">
                            Bilhete #{idx + 1}
                        </p>
                        <p className="text-[10px] text-white/60 break-all">
                            Ref: #${t.id}
                        </p>
                    </div>
                    <span className="inline-flex items-center rounded-full border px-2 py-0.5 text-[10px] text-white/80">
                        {isListed ? "À venda" : "Na tua carteira"}
                    </span>
                </div>
            </div>
            <div className="flex items-start gap-3">
                {t.qrToken ? (
                    <div className="shrink-0 rounded-2xl bg-white p-2 shadow-[0_0_28px_rgba(255,0,200,0.35)]">
                        /* eslint-disable-next-line @next/next/no-img-element */
                        <img
                            src={`/api/qr/${t.qrToken}?r=${rotationWindow}`}
                            alt="QR Code do bilhete ORYA"
                            className="h-20 w-20 object-contain"
                        />
                    </div>
                ) : (
                    <div className="shrink-0 rounded-2xl border border-dashed border-white/25 bg-black/40 px-3 py-2 text-[10px] text-white/60 text-center max-w-[120px]">
                        A preparar o OR...
                    </div>
                )}
            </div>
        )
    )
    <div className="flex-1 space-y-2 text-[11px]">
        <p className="text-white/80">
            Preço pago: {formatPrice(t.priceEur, t.currency)}
        </p>
        <div className="space-y-1 text-white/60">
            <div className="flex gap-2 text-[10px]">
                <span className="w-24 text-white/50">Bruto</span>
                <span>{formatPrice(t.grossCents / 100, t.currency)}</span>
            </div>
            {t.discountCents > 0 && (
                <div className="flex gap-2 text-[10px] text-emerald-300">
                    <span className="w-24 text-white/50">Desconto</span>
                    <span>{formatPrice(t.discountCents / 100, t.currency)}</span>
                </div>
            )}
            {t.platformFeeCents > 0 && (
                <div className="flex gap-2 text-[10px] text-orange-200">
                    <span className="w-24 text-white/50">Taxa</span>
                    <span>{formatPrice(t.platformFeeCents / 100, t.currency)}</span>
                </div>
            )}
            <div className="flex gap-2 text-[10px] text-white">
                <span className="w-24 text-white/50">Líquido</span>
                <span>{formatPrice(t.netCents / 100, t.currency)}</span>
            </div>
        </div>
        <div className="flex flex-wrap gap-2">

```



```

[0_20px_60px_rgba(0,0,0,0.85)]">
    <div className="flex items-start justify-between gap-3">
        <div>
            <p className="text-[11px] uppercase tracking-[0.16em] text-white/50">
                Transferir bilhete
            </p>
            <h2 className="mt-1 text-sm font-semibold text-white">
                Enviar a um amigo
            </h2>
            {transferModal.ticketTitle && (
                <p className="mt-1 text-[11px] text-white/65">
                    Evento: <span className="font-medium">{transferModal.ticketTitle}</span>
                </p>
            )}
        </div>
        <button
            type="button"
            onClick={() =>
                setTransferModal({ open: false, ticketId: null, ticketTitle: null })
            }
            className="text-[11px] text-white/50 hover:text-white"
        >
            Fechar
        </button>
    </div>
<div className="mt-4 space-y-2">
    <label className="block text-[11px] text-white/70">
        Username ORYA do teu amigo
    </label>
    <input
        type="text"
        value={transferUsername}
        onChange={(e) => setTransferUsername(e.target.value)}
        placeholder="@username"
        className="mt-1 w-full rounded-xl border border-white/15 bg-white/5 px-3 py-2 text-sm text-white
placeholder:text-white/40 focus:outline-none focus:ring-2 focus:ring-[#6BFFFF]/70"
    />
    {transferError && (
        <p className="mt-2 text-[11px] text-red-300">
            {transferError}
        </p>
    )}
    {transferSuccess && (
        <p className="mt-2 text-[11px] text-emerald-300">
            {transferSuccess}
        </p>
    )}
</div>
<div className="mt-5 flex justify-end gap-2 text-[11px]">
    <button
        type="button"
        onClick={() =>
            setTransferModal({ open: false, ticketId: null, ticketTitle: null })
        }
        className="px-3 py-1.5 rounded-full border border-white/20 text-white/75 hover:bg-white/5 transition-colors"
    >
        Cancelar
    </button>
    <button
        type="button"
        onClick={handleConfirmTransfer}
        disabled={transferLoading}
        className="px-3 py-1.5 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] font-semibold
text-black shadow-[0_0_20px_rgba(107,255,255,0.7)] hover:scale-[1.02] active:scale-95 transition-transform disabled:opacity-60
disabled:hover:scale-100"
    >
        {transferLoading ? "A enviar..." : "Confirmar transferência"}
    </button>
</div>
</div>
</div>
)
{resaleModal.open && (
    <div className="fixed inset-0 z-50 flex items-center justify-center bg-black/70 backdrop-blur-md">

```

```

<div className="w-full max-w-sm rounded-2xl border border-white/15 bg-black/90 p-5 shadow-[0_20px_60px_rgba(0,0,0,0.85)]">
  <div className="flex items-start justify-between gap-3">
    <div>
      <p className="text-[11px] uppercase tracking-[0.16em] text-white/50">
        Revender bilhete
      </p>
      <h2 className="mt-1 text-sm font-semibold text-white">
        Colocar à venda
      </h2>
    {resaleModal.ticketTitle && (
      <p className="mt-1 text-[11px] text-white/65">
        Evento:{" "}
        <span className="font-medium">
          {resaleModal.ticketTitle}
        </span>
      </p>
    )}
  </div>
  <button
    type="button"
    onClick={() =>
      setResaleModal({
        open: false,
        ticketId: null,
        ticketTitle: null,
      })
    }
    className="text-[11px] text-white/50 hover:text-white"
  >
    Fechar
  </button>
</div>

<div className="mt-4 space-y-2">
  <label className="block text-[11px] text-white/70">
    Preço de revenda (em €)
  </label>
  <input
    type="text"
    value={resalePrice}
    onChange={(e) => setResalePrice(e.target.value)}
    placeholder="ex: 15 ou 15,50"
    className="mt-1 w-full rounded-xl border border-white/15 bg-white/5 px-3 py-2 text-sm text-white
placeholder:text-white/40 focus:outline-none focus:ring-2 focus:ring-[#6BFFFF]/70"
  />
  {resaleError && (
    <p className="mt-2 text-[11px] text-red-300">
      {resaleError}
    </p>
  )}
  {resaleSuccess && (
    <p className="mt-2 text-[11px] text-emerald-300">
      {resaleSuccess}
    </p>
  )}
</div>

<div className="mt-5 flex justify-end gap-2 text-[11px]">
  <button
    type="button"
    onClick={() =>
      setResaleModal({
        open: false,
        ticketId: null,
        ticketTitle: null,
      })
    }
    className="px-3 py-1.5 rounded-full border border-white/20 text-white/75 hover:bg-white/5 transition-colors"
  >
    Cancelar
  </button>
  <button
    type="button"
    onClick={handleConfirmResale}
    disabled={resaleLoading}
  >
    Confirmar
  </button>
</div>

```

```

        className="px-3 py-1.5 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] font-semibold
text-black shadow-[0_0_20px_rgba(107,255,255,0.7)] hover:scale-[1.02] active:scale-95 transition-transform disabled:opacity-60
disabled:hover:scale-100"
      >
      {resaleLoading ? "A listar..." : "Confirmar revenda"}
    </button>
  </div>
</div>
</div>
</main>
</>
);
}
}

```

app/me/tickets/transfers/page.tsx

```

"use client";

import { useEffect, useState } from "react";
import Link from "next/link";
import { useRouter } from "next/navigation";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";

type TransferUser = {
  id: string;
  username?: string | null;
  fullName?: string | null;
};

type TransferEvent = {
  slug?: string | null;
  title?: string | null;
  startDate?: string | null;
  locationName?: string | null;
};

type TransferTicketType = {
  name?: string | null;
};

type TransferTicket = {
  id: string;
  event?: TransferEvent | null;
  ticketType?: TransferTicketType | null;
};

type TransferItem = {
  id: string;
  status: "PENDING" | "ACCEPTED" | "CANCELLED" | "EXPIRED" | string;
  createdAt: string;
  completedAt?: string | null;
  ticket?: TransferTicket | null;
  fromUser?: TransferUser | null;
  toUser?: TransferUser | null;
};

type TransfersApiResponse = {
  ok: boolean;
  incoming: TransferItem[];
  outgoing: TransferItem[];
  // se a API devolver { ok: false, error }, tratamos pelo status HTTP
};

export default function TicketTransfersPage() {
  const router = useRouter();
  const { openModal } = useAuthModal();

  const [loading, setLoading] = useState(true);
  const [errorMsg, setErrorMsg] = useState<string | null>(null);
  const [incoming, setIncoming] = useState<TransferItem[]>([]);
  const [outgoing, setOutgoing] = useState<TransferItem[]>([]);
  const [respondingId, setRespondingId] = useState<string | null>(null);

  useEffect(() => {
    let cancelled = false;

```

```

async function loadTransfers() {
  try {
    setLoading(true);
    setErrorMsg(null);

    const res = await fetch("/api/me/tickets/transfers", {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
      },
      cache: "no-store",
    });

    if (res.status === 401) {
      // não autenticado → abrir modal de login e redirecionar de volta para esta página
      openModal({ mode: "login", redirectTo: "/me/tickets/transfers", showGoogle: true });
      return;
    }

    if (!res.ok) {
      const text = await res.text().catch(() => "");
      console.error("Erro ao carregar transferências:", text);
      if (!cancelled) {
        setErrorMsg(
          "Não foi possível carregar as tuas transferências. Tenta novamente em alguns segundos."
        );
      }
      return;
    }
  }

  const json = (await res.json()).catch(() => null) as TransfersApiResponse | null;

  if (!json || !json.ok) {
    if (!cancelled) {
      setErrorMsg("Resposta inesperada ao tentar carregar as transferências.");
    }
    return;
  }

  if (!cancelled) {
    setIncoming(Array.isArray(json.incoming) ? json.incoming : []);
    setOutgoing(Array.isArray(json.outgoing) ? json.outgoing : []);
  }
} catch (err) {
  console.error("Erro inesperado ao carregar transferências:", err);
  if (!cancelled) {
    setErrorMsg(
      "Ocorreu um problema inesperado. Tenta novamente dentro de instantes."
    );
  }
} finally {
  if (!cancelled) {
    setLoading(false);
  }
}
}

loadTransfers();

return () => {
  cancelled = true;
};

}, [openModal]);

function formatDate(dateStr?: string | null) {
  if (!dateStr) return "";
  const d = new Date(dateStr);
  if (Number.isNaN(d.getTime())) return "";
  return new Intl.DateTimeFormat("pt-PT", {
    day: "2-digit",
    month: "short",
    year: "numeric",
    hour: "2-digit",
    minute: "2-digit",
  }).format(d);
}

```

```

function formatUser(user?: TransferUser | null) {
  if (!user) return "Utilizador ORYA";
  if (user.username) return `${user.username}`;
  if (user.fullName) return user.fullName;
  return "Utilizador ORYA";
}

function statusLabel(status: string) {
  switch (status) {
    case "PENDING":
      return "Pendente";
    case "ACCEPTED":
      return "Aceite";
    case "CANCELLED":
      return "Cancelada";
    case "EXPIRED":
      return "Expirada";
    default:
      return status;
  }
}

function statusClasses(status: string) {
  switch (status) {
    case "PENDING":
      return "border-amber-400/50 bg-amber-500/10 text-amber-100";
    case "ACCEPTED":
      return "border-emerald-400/50 bg-emerald-500/10 text-emerald-100";
    case "CANCELLED":
    case "EXPIRED":
      return "border-white/25 bg-white/5 text-white/80";
    default:
      return "border-white/25 bg-white/5 text-white/80";
  }
}

async function handleRespond(transferId: string, action: "ACCEPT" | "DECLINE") {
  try {
    setRespondingId(transferId);

    const res = await fetch("/api/tickets/transfer/respond", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ transferId, action }),
    });

    const data = await res.json().catch(() => null);

    if (!res.ok || !data?.ok) {
      console.error("Erro ao responder à transferência:", data);
      alert("Não foi possível atualizar esta transferência. Tenta novamente.");
      return;
    }

    // Atualizar estado local: a resposta do backend já mudou o dono / status;
    // aqui só precisamos de refletir o novo estado na lista de incoming.
    setIncoming((prev) =>
      prev.map((t) =>
        t.id === transferId
          ? {
              ...t,
              status: action === "ACCEPT" ? "ACCEPTED" : "CANCELLED",
              completedAt: new Date().toISOString(),
            }
          : t
      )
    );
  } catch (err) {
    console.error("Erro ao responder à transferência:", err);
    alert("Ocorreu um erro inesperado. Tenta novamente dentro de instantes.");
  } finally {
    setRespondingId(null);
  }
}

const hasIncoming = incoming.length > 0;

```

```

const hasOutgoing = outgoing.length > 0;

return (
  <main className="orya-body-bg min-h-screen w-full text-white pb-16">
    {/* Top bar */}
    <header className="border-b border-white/10 bg-black/40 backdrop-blur-xl">
      <div className="max-w-5xl mx-auto px-5 py-4 flex items-center justify-between">
        <div className="flex items-center gap-3">
          <span className="inline-flex h-8 w-8 items-center justify-center rounded-xl bg-gradient-to-tr from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-xs font-extrabold tracking-[0.15em]">
            OR
          </span>
        <div>
          <p className="text-xs uppercase tracking-[0.18em] text-white/60">
            A minha conta
          </p>
          <p className="text-sm text-white/85">
            Transferências de bilhetes ligadas à tua conta ORYA.
          </p>
        </div>
      </div>
    </div>

    <div className="flex items-center gap-2">
      <Link href="/me/tickets" className="hidden sm:inline-flex text-[11px] px-3 py-1.5 rounded-xl border border-white/15 text-white/75 hover:bg-white/5 transition-colors">
        &larr; Voltar aos bilhetes
      </Link>
      <Link href="/me" className="hidden sm:inline-flex text-[11px] px-3 py-1.5 rounded-xl border border-white/15 text-white/75 hover:bg-white/5 transition-colors">
        Conta
      </Link>
    </div>
  </header>

  <section className="max-w-5xl mx-auto px-5 pt-8 md:pt-10 space-y-6">
    <div className="flex flex-col md:flex-row md:items-center md:justify-between gap-3">
      <div>
        <h1 className="text-2xl md:text-3xl font-semibold tracking-tight">
          Transferências de bilhetes
        </h1>
        <p className="mt-1 text-sm text-white/70 max-w-xl">
          Aqui vais conseguir ver convites de bilhetes que te enviaram, as transferências que fizeste para amigos e o estado de cada uma.
        </p>
      </div>
      <div className="flex flex-wrap gap-2 text-[11px]">
        <Link href="/me/tickets" className="px-3 py-1.5 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold hover:scale-105 active:scale-95 transition-transform shadow-[0_0_26px_rgba(107,255,255,0.45)]">
          Ver os meus bilhetes
        </Link>
      </div>
    </div>
    {loading && (
      <div className="mt-6 space-y-4" role="status" aria-live="polite">
        <div className="rounded-xl border border-white/10 bg-white/5 px-4 py-3 text-sm text-white/70">
          A carregar as tuas transferências...
        </div>
      </div>
    )}
    {errorMsg && !loading && (
      <div className="mt-6 rounded-xl border border-red-500/40 bg-red-500/10 px-4 py-3 text-xs text-red-100 flex items-start gap-2" role="alert">
        <div>

```

```

<span className="mt-[2px] text-sm">⚠</span>
<div className="space-y-1">
  <p className="font-medium text-red-100">Não foi possível carregar</p>
  <p className="text-[11px] text-red-100/80">{errorMsg}</p>
</div>
</div>
) {}

{!loading && !errorMsg && !hasIncoming && !hasOutgoing && (
<div className="mt-8 rounded-2xl border border-dashed border-white/15 bg-white/5 px-6 py-8 text-center space-y-3">
  <p className="text-lg font-medium">Ainda não tens transferências</p>
  <p className="text-sm text-white/65 max-w-md mx-auto">
    Quando envias um bilhete a um amigo – ou alguém te enviar um a
    ti – esse pedido vai aparecer aqui, com o estado da transferência.
  </p>
  <Link href="/me/tickets"
    className="inline-flex mt-2 px-4 py-2.5 rounded-xl bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5]"
    text-xs font-semibold text-black hover:scale-105 active:scale-95 transition-transform shadow-[0_0_28px_rgba(107,255,255,0.5)]">
    >
    Voltar aos bilhetes
  </Link>
</div>
) {}

{!loading && !errorMsg && (hasIncoming || hasOutgoing) && (
<div className="space-y-8">
  /* Convites recebidos */
  <section className="space-y-3">
    <div className="flex items-center justify-between gap-2">
      <h2 className="text-sm font-semibold text-white/85">
        Convites recebidos
      </h2>
      <span className="text-[11px] text-white/60">
        {incoming.length} transferência(s)
      </span>
    </div>

    {incoming.length === 0 ? (
      <p className="text-[11px] text-white/55">
        Neste momento não tens convites de bilhetes por aceitar.
      </p>
    ) : (
      <div className="space-y-3">
        {incoming.map(t => {
          const event = t.ticket?.event;
          const ticketType = t.ticket?.ticketType;
          const from = t.fromUser;
          const isPending = t.status === "PENDING";

          return (
            <div
              key={t.id}
              className="rounded-2xl border border-white/12 bg-black/70 px-4 py-3 flex flex-col gap-2 md:flex-row
md:items-center md:justify-between"
            >
              <div className="space-y-1 text-[11px]">
                <div className="flex flex-wrap items-center gap-2">
                  <span className="text-white/80 font-semibold">
                    {event?.title || "Evento ORYA"}
                  </span>
                  {ticketType?.name && (
                    <span className="rounded-full border border-white/15 px-2 py-[2px] text-[10px] text-white/70">
                      Wave {ticketType.name}
                    </span>
                  )}
                </div>
                <p className="text-white/60">
                  Enviado por{" "}
                  <span className="font-medium text-white/85">
                    {formatUser(from)}
                  </span>
                </p>
                {event?.startDate && (
                  <p className="text-white/55">
                    Data do evento:{" "}
                    <span className="text-white/80">
                      {formatDate(event.startDate)}
                    </span>
                  </p>
                )}
              </div>
            </div>
          )
        )}
      </div>
    )
  </section>
</div>
) {}}

```

```

        </span>
        </p>
    )}
<div className="flex flex-wrap items-center gap-2">
    <span
        className={`inline-flex items-center rounded-full border px-2 py-[2px] text-[10px] font-medium
${statusClasses(
            t.status
        )}`}>
    <span>
        {statusLabel(t.status)}
    </span>
    <span className="text-[10px] text-white/50">
        Pedido criado em {formatDate(t.createdAt)}
    </span>
    {t.completedAt && (
        <span className="text-[10px] text-white/45">
            · Atualizado em {formatDate(t.completedAt)}
        </span>
    )}
</div>
</div>

<div className="flex flex-wrap gap-2 justify-end mt-2 md:mt-0 text-[11px]">
    {event?.slug && (
        <button
            type="button"
            onClick={() => router.push(`/eventos/${event.slug}`)}
            className="px-3 py-1.5 rounded-full border border-white/20 text-white/80 hover:bg-white/10
transition-colors"
        >
            Ver evento
        </button>
    )}
    {isPending && (
        <>
            <button
                type="button"
                onClick={() => handleRespond(t.id, "DECLINE")}
                disabled={respondingId === t.id}
                className="px-3 py-1.5 rounded-full border border-white/20 text-white/75 hover:bg-white/8
transition-colors disabled:opacity-60"
            >
                {respondingId === t.id && "A atualizar..."}
                {respondingId !== t.id && "Recusar"}
            </button>
            <button
                type="button"
                onClick={() => handleRespond(t.id, "ACCEPT")}
                disabled={respondingId === t.id}
                className="px-3 py-1.5 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5]
font-semibold text-black shadow-[0_0_20px_rgba(107,255,255,0.7)] hover:scale-[1.02] active:scale-95 transition-transform
disabled:opacity-60 disabled:hover:scale-100"
            >
                {respondingId === t.id ? "A aceitar..." : "Aceitar bilhete"}
            </button>
        </>
    )}
    </div>
    </div>
);
}]}
</div>
</section>

/* Transferências enviadas */
<section className="space-y-3">
    <div className="flex items-center justify-between gap-2">
        <h2 className="text-sm font-semibold text-white/85">
            Transferências enviadas
        </h2>
        <span className="text-[11px] text-white/60">
            {outgoing.length} transferência(s)
        </span>
    </div>

```

```

{outgoing.length === 0 ? (
  <p className="text-[11px] text-white/55">
    Ainda não enviaste nenhum bilhete para amigos. Podes começar a
    partir da página dos teus bilhetes.
  </p>
) : (
  <div className="space-y-3">
    {outgoing.map((t) => {
      const event = t.ticket?.event;
      const ticketType = t.ticket?.ticketType;
      const to = t.toUser;

      return (
        <div
          key={t.id}
          className="rounded-2xl border border-white/12 bg-black/70 px-4 py-3 flex flex-col gap-2 md:flex-row
md:items-center md:justify-between"
        >
          <div className="space-y-1 text-[11px]">
            <div className="flex flex-wrap items-center gap-2">
              <span className="text-white/80 font-semibold">
                {event?.title || "Evento ORYA"}
              </span>
              {ticketType?.name && (
                <span className="rounded-full border border-white/15 px-2 py-[2px] text-[10px] text-white/70">
                  Wave {ticketType.name}
                </span>
              )}
            </div>
          <p className="text-white/60">
            Enviado para{" "}
            <span className="font-medium text-white/85">
              {formatUser(to)}
            </span>
          </p>
          {event?.startDate && (
            <p className="text-white/55">
              Data do evento:{" "}
              <span className="text-white/80">
                {formatDate(event.startDate)}
              </span>
            </p>
          )}
        <div className="flex flex-wrap items-center gap-2">
          <span
            className={`inline-flex items-center rounded-full border px-2 py-[2px] text-[10px] font-medium
${statusClasses(
  t.status
)}`}>
            {statusLabel(t.status)}
          </span>
          <span className="text-[10px] text-white/50">
            Pedido criado em {formatDate(t.createdAt)}
          </span>
          {t.completedAt && (
            <span className="text-[10px] text-white/45">
              · Atualizado em {formatDate(t.completedAt)}
            </span>
          )}
        </div>
      </div>
    );
  )});

```

```

        </div>
    )}
</section>
</div>
)}
</section>
</main>
);
}

```

app/not-found.tsx

```

import Link from "next/link";

export default function NotFound() {
  return (
    <main className="min-h-screen bg-[radial-gradient(circle_at_top,_#1a1030_0,_#050509_45%,_#02020a_100%)] text-white flex items-center justify-center px-4">
      <div className="max-w-lg w-full rounded-3xl border border-white/10 bg-black/50 backdrop-blur-2xl px-6 py-8 md:px-8 md:py-10 shadow-[0_24px_80px_rgba(0,0,0,0.75)] space-y-6">
        <div className="flex items-center gap-3">
          <span className="inline-flex h-9 w-9 items-center justify-center rounded-2xl bg-gradient-to-tr from-[#FF00C8] via-[#6BFFF] to-[#1646F5] text-[11px] font-extrabold tracking-[0.18em]">
            OR
          </span>
          <div className="space-y-1">
            <p className="text-[11px] uppercase tracking-[0.22em] text-white/55">
              ORYA
            </p>
            <p className="text-sm text-white/80">
              Página não encontrada
            </p>
          </div>
        </div>
      </div>

      <div className="space-y-4">
        <h1 className="text-2xl md:text-3xl font-semibold tracking-tight">
          Não encontrámos nada aqui.
        </h1>
        <p className="text-sm text-white/65">
          A página que procuraste pode ter mudado, deixado de existir ou o link pode estar incorreto. Mas a cidade continua cheia de eventos à tua espera.
        </p>
      </div>

      <div className="space-y-3">
        <Link href="/explorar"
              className="inline-flex w-full items-center justify-center gap-2 rounded-xl bg-gradient-to-r from-[#FF00C8] via-[#6BFFF] to-[#1646F5] px-4 py-2.5 text-xs font-semibold text-black shadow-[0_0_32px_rgba(107,255,255,0.6)] transition-transform hover:scale-[1.02] active:scale-95">
          Voltar a explorar eventos
          <span className="text-[14px]">v</span>
        </Link>
        <Link href="/"
              className="inline-flex w-full items-center justify-center rounded-xl border border-white/15 bg-white/5 px-4 py-2.5 text-[11px] font-medium text-white/80 hover:bg-white/10 transition-colors">
          Ir para a página inicial
        </Link>
      </div>

      <p className="text-[10px] text-white/45 text-center">
        Se achares que isto é um erro, tenta voltar atrás ou usar a pesquisa na página inicial para encontrar o evento certo.
      </p>
    </div>
  );
}

```

app/onboarding/perfil/page.tsx

```
"use client";

import { Suspense, useEffect, useState, FormEvent } from "react";
import { useRouter, useSearchParams } from "next/navigation";
import { useUser } from "@app/hooks/useUser";
import { sanitizeUsername, validateUsername, USERNAME_RULES_HINT } from "@lib/username";

function OnboardingPerfilContent() {
  const router = useRouter();
  const searchParams = useSearchParams();
  const redirectTo = searchParams.get("redirectTo") || "/";

  const { user, profile, isLoading, refetch } = useUser();

  // Se não estiver autenticado e não estiver a carregar, manda embora
  useEffect(() => {
    if (!isLoading && !user) {
      router.replace("/");
    }
  }, [isLoading, user, router]);

  if (isLoading || !user || !profile) {
    return (
      <div className="min-h-[60vh] flex items-center justify-center">
        <p className="text-sm text-gray-500">A carregar o teu perfil...</p>
      </div>
    );
  }

  return (
    <ProfileForm
      initialFullName={profile.fullName ?? ""}
      initialUsername={profile.username ?? ""}
      onSaved={async () => {
        await refetch();
        router.push(redirectTo || "/");
      }}
    />
  );
}

export default function OnboardingPerfilPage() {
  return (
    <Suspense fallback={null}>
      <OnboardingPerfilContent />
    </Suspense>
  );
}

type ProfileFormProps = {
  initialFullName: string;
  initialUsername: string;
  onSaved: () => Promise<void>;
};

function ProfileForm({
  initialFullName,
  initialUsername,
  onSaved,
}: ProfileFormProps) {
  const [fullName, setFullName] = useState(initialFullName);
  const [username, setUsername] = useState(sanitizeUsername(initialUsername));
  const [usernameHint, setUsernameHint] = useState<string | null>(null);
  const [usernameStatus, setUsernameStatus] = useState<
    "idle" | "checking" | "available" | "taken" | "error"
  >("idle");
  const [error, setError] = useState<string | null>(null);
  const [isSubmitting, setIsSubmitting] = useState(false);

  async function checkUsernameAvailability(currentUsername: string) {
    const trimmed = sanitizeUsername(currentUsername);
    if (!trimmed) {
      setUsernameHint(USERNAME_RULES_HINT);
      setUsernameStatus("idle");
    }
  }
}
```

```

        return false;
    }

    const validation = validateUsername(trimmed);
    if (!validation.valid) {
        setUsernameHint(validation.error);
        setUsernameStatus("error");
        return false;
    }

    setUsernameHint(null);
    setUsernameStatus("checking");
    try {
        const res = await fetch(`~/api/username/check?username=${encodeURIComponent(trimmed)}`);

        if (!res.ok) {
            setUsernameStatus("error");
            return false;
        }

        const data = (await res.json()) as { available: boolean };
        const available = data.available;
        setUsernameStatus(available ? "available" : "taken");
        return available;
    } catch (e) {
        console.error("Erro ao verificar username:", e);
        setUsernameStatus("error");
        return false;
    }
}

async function handleSubmit(e: FormEvent) {
    e.preventDefault();
    setError(null);

    const trimmedName = fullName.trim();
    const trimmedUsername = sanitizeUsername(username);
    const validation = validateUsername(trimmedUsername);

    if (!trimmedName || !validation.valid) {
        setError(validation.valid ? "Preenche o nome e o username." : validation.error);
        return;
    }

    setIsSubmitting(true);

    const available = await checkUsernameAvailability(trimmedUsername);
    if (!available) {
        setIsSubmitting(false);
        setError("Este nome já está em uso – escolha outro.");
        return;
    }

    try {
        const res = await fetch("/api/profiles/save-basic", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ fullName: trimmedName, username: validation.normalized }),
        });

        if (!res.ok) {
            const data = await res.json().catch(() => null);
            const message = data?.error || "Não foi possível salvar o perfil.";
            setError(message);
            setIsSubmitting(false);
            return;
        }

        await onSave();
    } catch (err) {
        console.error("Erro ao salvar perfil:", err);
        setError("Ocorreu um erro ao salvar. Tente novamente.");
        setIsSubmitting(false);
        return;
    }

    setIsSubmitting(false);
}

```

```

return (
  <div className="min-h-[70vh] flex items-center justify-center px-4">
    <div className="w-full max-w-md rounded-2xl border border-gray-200 bg-white/80 p-6 shadow-sm">
      <h1 className="text-xl font-semibold mb-1">Completa o teu perfil</h1>
      <p className="text-sm text-gray-500 mb-6">
        Só precisas de definir o teu nome e um username. Depois disto já estás
        pronto para criar experiências e juntar-te a eventos.
      </p>

      <form onSubmit={handleSubmit} className="space-y-4">
        <div className="space-y-1">
          <label className="text-sm font-medium" htmlFor="fullName">
            Nome completo
          </label>
          <input
            id="fullName"
            type="text"
            value={fullName}
            onChange={(e) => setFullName(e.target.value)}
            className="w-full rounded-lg border border-gray-300 px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-black/80"
            placeholder="Como os teus amigos te conhecem"
          />
        </div>

        <div className="space-y-1">
          <label className="text-sm font-medium" htmlFor="username">
            Username
          </label>
          <div className="relative">
            <span className="pointer-events-none absolute left-3 top-1/2 -translate-y-1/2 text-gray-400 text-sm">
              @
            </span>
            <input
              id="username"
              type="text"
              inputMode="text"
              pattern="[A-Za-z0-9._]{0,30}"
              value={username}
              onChange={(e) => {
                const raw = e.target.value;
                const cleaned = sanitizeUsername(raw);
                setUsername(cleaned);
                const validation = validateUsername(cleaned);
                setUsernameHint(validation.valid ? null : validation.error);
                setUsernameStatus("idle");
              }}
              onBlur={() => checkUsernameAvailability(username)}
              maxLength={30}
              className="w-full rounded-lg border border-gray-300 pl-7 pr-3 py-2 text-sm focus:outline-none focus:ring-2
focus:ring-black/80"
              placeholder="teu.nome_ou_marca"
            />
          </div>
          <p className="text-xs text-gray-400">
            Este será o link público do teu perfil: orya.app/{username} || "teu.username"
          </p>
        {usernameHint && (
          <p className="text-xs text-amber-600">{usernameHint}</p>
        )}
        {usernameStatus === "checking" && (
          <p className="text-xs text-gray-500">A verificar disponibilidade...</p>
        )}
        {usernameStatus === "available" && username && (
          <p className="text-xs text-green-600">Este username está disponível.</p>
        )}
        {usernameStatus === "taken" && (
          <p className="text-xs text-red-600">Este username já existe, escolhe outro.</p>
        )}
        {usernameStatus === "error" && (
          <p className="text-xs text-red-600">Não foi possível verificar o username.</p>
        )}
      </div>

      {error && <p className="text-sm text-red-600">{error}</p>

      <button

```

```

        type="submit"
        disabled={isSubmitting}
        className="mt-2 inline-flex w-full items-center justify-center rounded-lg bg-black px-4 py-2 text-sm font-medium
text-white hover:bg-black/90 disabled:opacity-60"
      >
    {isSubmitting ? "A guardar..." : "Guardar e continuar"}
  </button>
</form>
</div>
</div>
);
}

```

app/org/[username]/page.tsx

```

import { notFound } from "next/navigation";
import type { CSSProperties } from "react";
import { prisma } from "@/lib/prisma";

type Params = { username: string };

export default async function PublicOrganizerPage({ params }: { params: Promise<Params> }) {
  const { username } = await params;

  if (!username) notFound();

  const organizer = await prisma.organizer.findFirst({
    where: { username: { equals: username, mode: "insensitive" } },
    select: {
      id: true,
      displayName: true,
      publicName: true,
      businessName: true,
      username: true,
      city: true,
      address: true,
      showAddressPublicly: true,
      publicListingEnabled: true,
      brandingAvatarUrl: true,
      brandingPrimaryColor: true,
      brandingSecondaryColor: true,
    },
  });

  if (!organizer || organizer.publicListingEnabled === false) {
    notFound();
  }

  const events = await prisma.event.findMany({
    where: { organizerId: organizer.id, status: "PUBLISHED" },
    select: {
      id: true,
      slug: true,
      title: true,
      templateType: true,
      startsAt: true,
      locationName: true,
      coverImageUrl: true,
      isFree: true,
    },
    orderBy: { startsAt: "asc" },
    take: 50,
  });

  const primary = organizer.brandingPrimaryColor || "#6BFFFF";
  const secondary = organizer.brandingSecondaryColor || "#0b1224";
  const displayName = organizer.publicName || organizer.displayName || organizer.businessName || organizer.username || "Organizador";

  return (
    <main
      className="min-h-screen text-white"
      style={{
        "--brand-primary": primary,
        "--brand-secondary": secondary,
      }}
    >
      <OrganizerPage {...organizer} events={events} />
    </main>
  );
}

```

```

        } as CSSProperties
    }
>
<section className="mx-auto max-w-5xl px-5 py-10 space-y-6">
    <div className="flex items-center gap-3">
        <div className="h-16 w-16 rounded-2xl border border-white/15 bg-[var(--brand-secondary)] shadow-[0_10px_30px_rgba(0,0,0,0.45)] overflow-hidden">
            {organizer.brandingAvatarUrl ? (
                // eslint-disable-next-line @next/next/no-img-element
                <img src={organizer.brandingAvatarUrl} alt={displayName} className="h-full w-full object-cover" />
            ) : (
                <div className="flex h-full w-full items-center justify-center text-lg font-semibold text-white/80">
                    {displayName.charAt(0).toUpperCase()}
                </div>
            )
        )}
    </div>
    <div className="space-y-1">
        <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Organizador</p>
        <h1 className="text-3xl font-bold">{displayName}</h1>
        {organizer.city && <p className="text-sm text-white/70">{organizer.city}</p>}
        {organizer.showAddressPublicly && organizer.address && (
            <p className="text-sm text-white/60">{organizer.address}</p>
        )}
    </div>
</div>

```

```

<div className="rounded-3xl border border-white/10 bg-gradient-to-br from-[var(--brand-secondary)] via-[#0b1224] to-black p-5 shadow-[0_20px_60px_rgba(0,0,0,0.55)]">
    <div className="flex items-center justify-between gap-3">
        <div>
            <h2 className="text-lg font-semibold">Eventos</h2>
            <p className="text-[12px] text-white/65">
                Agenda pública deste organizador. Apenas eventos listados publicamente são mostrados.
            </p>
        </div>
        <span className="rounded-full border border-white/15 bg-white/5 px-3 py-1 text-[12px] text-white/70">
            {events.length} evento{events.length === 1 ? "" : "s"}
        </span>
    </div>

```

```

{events.length === 0 ? (
    <div className="mt-4 rounded-xl border border-dashed border-white/15 bg-white/5 p-4 text-sm text-white/70">
        Ainda não há eventos publicados por esta organização.
    </div>
) : (
    <div className="mt-4 grid gap-4 sm:grid-cols-2 lg:grid-cols-3">
        {events.map((ev) => (
            <a
                key={ev.id}
                href={`/eventos/${ev.slug}`}
                className="group rounded-2xl border border-white/10 bg-white/[0.04] overflow-hidden hover:border-white/18
                hover:-translate-y-[4px] transition block"
            >
                <div className="h-32 w-full bg-gradient-to-br from-[var(--brand-secondary)]/80 to-black/60 overflow-hidden">
                    {ev.coverImageUrl ? (
                        // eslint-disable-next-line @next/next/no-img-element
                        <img
                            src={ev.coverImageUrl}
                            alt={ev.title}
                            className="h-full w-full object-cover transition-transform duration-200 group-hover:scale-[1.03]"
                        />
                    ) : null}
                </div>
                <div className="p-3 space-y-1.5">
                    <p className="text-[13px] font-semibold text-white line-clamp-2">{ev.title}</p>
                    <p className="text-[11px] text-white/70">
                        {ev.startsAt
                            ? new Date(ev.startsAt).toLocaleString("pt-PT", {
                                weekday: "short",
                                day: "2-digit",
                                month: "short",
                                hour: "2-digit",
                                minute: "2-digit",
                            })
                            : "Data a anunciar"}
                    </p>
                    <p className="text-[11px] text-white/60 line-clamp-1">{ev.locationName || "Local a anunciar"}</p>
                    <span className="inline-flex rounded-full border border-white/12 bg-[var(--brand-primary)]/15 px-2 py-0.5

```

```

text-[10px] text-white/80">
    {ev.templateType || "Evento"}
    </span>
    </div>
    </a>
    )}
    </div>
)
</div>
</section>
</main>
);
}

```

app/organizador/(dashboard)/categorias/padel/page.tsx

```

import { redirect } from "next/navigation";

export const runtime = "nodejs";

export default function PadelCategoryPage() {
  redirect("/organizador?tab=padel");
}

```

app/organizador/(dashboard)/categorias/page.tsx

```

"use client";

import useSWR from "swr";
import Link from "next/link";
import { useMemo } from "react";
import { useUser } from "@/app/hooks/useUser";

type EventsResponse = {
  ok: boolean;
  items: {
    id: number;
    title: string;
    startsAt: string | null;
    status: string;
    templateType: string | null;
    categories?: string[] | null;
  }[];
};

const fetcher = (url: string) => fetch(url).then((res) => res.json());

const CATEGORY_CARDS = [
  {
    key: "padel",
    title: "Torneios de Padel",
    desc: "Quadros, equipas, rankings. Em breve UI dedicada.",
    template: "SPORT",
    preset: "padel",
  },
  {
    key: "restaurantes",
    title: "Restaurantes & Jantares",
    desc: "Menus fixos, reservas por slot, grupos.",
    template: "COMIDA",
    preset: "restaurante",
  },
  {
    key: "solidario",
    title: "Solidário / Voluntariado",
    desc: "Angariação de fundos e inscrições de voluntários.",
    template: "VOLUNTEERING",
    preset: "solidario",
  },
  {
    key: "festas",
    title: "Festas & Noite",
    desc: "Guest lists, packs e consumo mínimo.",
    template: "PARTY",
  }
];

```

```

        preset: "party",
    },
];

export default function OrganizerCategoriesPage() {
    const { user, isLoading } = useUser();
    const { data, isLoading } = useSWR<EventsResponse>(
        user ? "/api/organizador/events/list" : null,
        fetcher,
        { revalidateOnFocus: false }
    );

    const events = data?.items ?? [];

    const cards = useMemo(() => {
        return CATEGORY_CARDS.map((cat) => {
            const filtered = events.filter((ev) => {
                const matchTemplate = ev.templateType === cat.template;
                const matchCategory = (ev.categories ?? []).includes(cat.template);
                return matchTemplate || matchCategory;
            });
            const future = filtered.filter((ev) => {
                const start = ev.startsAt ? new Date(ev.startsAt) : null;
                return start && start.getTime() > Date.now();
            });
            const active = filtered.filter((ev) => ev.status === "PUBLISHED");
            return { ...cat, filtered, future, active };
        });
    }), [events]);

    if (userLoading || !user) {
        return (
            <div className="mx-auto max-w-6xl px-4 py-10 space-y-4 md:px-6 lg:px-8 text-white">
                {userLoading ? "A carregar..." : "Precisas de iniciar sessão para veres as categorias."}
            </div>
        );
    }

    return (
        <div className="mx-auto max-w-6xl px-4 py-10 space-y-6 md:px-6 lg:px-8 text-white">
            <div className="space-y-2">
                <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Categorias</p>
                <h1 className="text-3xl font-semibold">Escolhe o modo de trabalho</h1>
                <p className="text-sm text-white/65">
                    Atalhos para criar eventos com a categoria certa e filtrar a tua lista. Em breve cada categoria terá UI própria.
                </p>
            </div>

            <div className="grid gap-3 md:grid-cols-2">
                {cards.map((cat) => (
                    <div
                        key={cat.key}
                        className="rounded-2xl border border-white/10 bg-white/5 p-4 space-y-3 shadow-[0_16px_50px_rgba(0,0,0,0.45)]"
                    >
                        <div className="flex items-center justify-between">
                            <div>
                                <h3 className="text-lg font-semibold">{cat.title}</h3>
                                <p className="text-[12px] text-white/65">{cat.desc}</p>
                            </div>
                            <div className="flex flex-col text-[11px] text-white/60 items-end">
                                <span>Ativos: {cat.active.length}</span>
                                <span>Futuros: {cat.future.length}</span>
                            </div>
                        </div>
                        <div className="flex flex-wrap gap-2 text-[11px] text-white/70">
                            {cat.filtered.slice(0, 3).map((ev) => (
                                <span key={ev.id} className="rounded-full border border-white/15 bg-white/5 px-2 py-0.5">
                                    {ev.title}
                                </span>
                            )))
                            {cat.filtered.length === 0 && <span className="text-white/50">Sem eventos nesta categoria.</span>}
                        </div>
                    <div className="flex gap-2">
                        <Link
                            href={`/organizador?tab=events&type=${cat.template}`}
                            className="flex-1 rounded-full border border-white/20 px-3 py-1.5 text-white/80 hover:bg-white/10 text-center"
                        >

```

```

        Ver eventos
    </Link>
    <Link
        href={`/organizador/(dashboard)/eventos/novo?preset=${cat.preset}`}
        className="flex-1 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-3 py-1.5 text-[12px] font-semibold text-black shadow text-center"
    >
        Criar {cat.preset === "restaurante" ? "jantar" : cat.preset === "padel" ? "torneio" : "evento"}
    </Link>
</div>
</div>
        ))}
    </div>
</div>
);
}
}

```

app/organizador/(dashboard)/eventos/[id]/edit/page.tsx

```

// app/organizador/(dashboard)/eventos/[id]/edit/page.tsx
import { notFound, redirect } from "next/navigation";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { EventEditClient } from "@app/organizador/(dashboard)/eventos/EventEditClient";

type PageProps = {
    params: Promise<{ id: string }>;
};

export default async function OrganizerEventEditPage({ params }: PageProps) {
    const { id } = await params;
    const eventId = Number(id);
    if (!Number.isFinite(eventId)) notFound();

    const supabase = await createSupabaseServer();
    const { data, error } = await supabase.auth.getUser();
    if (error || !data?.user) {
        redirect("/login?redirectTo=/organizador/eventos");
    }

    const organizer = await prisma.organizer.findFirst({
        where: { userId: data.user.id },
    });
    if (!organizer) {
        redirect("/organizador");
    }

    const event = await prisma.event.findFirst({
        where: { id: eventId, organizerId: organizer.id },
        include: {
            ticketTypes: true,
        },
    });

    if (!event) notFound();

    const tickets = event.ticketTypes.map((t) => ({
        id: t.id,
        name: t.name,
        description: t.description,
        price: t.price,
        currency: t.currency,
        totalQuantity: t.totalQuantity,
        soldQuantity: t.soldQuantity,
        status: t.status,
        startsAt: t.startsAt ? t.startsAt.toISOString() : null,
        endsAt: t.endsAt ? t.endsAt.toISOString() : null,
    }));
}

return (
    <div className="mx-auto max-w-5xl px-4 py-10 space-y-6 text-white md:px-6 lg:px-8">
        <div className="flex items-center justify-between gap-3">
            <div>
                <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Editar evento</p>
                <h1 className="text-2xl font-semibold">{event.title}</h1>
                <p className="text-sm text-white/60">ID {event.id} · {event.slug}</p>
            </div>
        </div>
    </div>
)

```

```

</div>
<a
  href={`/eventos/${event.slug}`}
  className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-3 py-1.5 text-[11px] font-semibold text-black shadow"
>
  Ver página pública
</a>
</div>

<EventEditClient
  event={{
    id: event.id,
    title: event.title,
    description: event.description,
    startsAt: event.startsAt.toISOString(),
    endsAt: event.endsAt.toISOString(),
    locationName: event.locationName,
    locationCity: event.locationCity,
    address: event.address,
    templateType: event.templateType,
    isFree: event.isFree,
    coverImageUrl: event.coverImageUrl,
    feeModeOverride: event.feeModeOverride,
    platformFeeBpsOverride: event.platformFeeBpsOverride,
    platformFeeFixedCentsOverride: event.platformFeeFixedCentsOverride,
    payoutMode: event.payoutMode,
  }}
  tickets={tickets}
/>
</div>
);
}
}

```

app/organizador/(dashboard)/eventos/[id]/PadelTournamentSection.tsx

```

"use client";

import { useEffect, useState } from "react";

type Player = {
  id: number;
  fullName: string;
  level: string | null;
};

type Team = {
  id: number;
  player1?: Player | null;
  player2?: Player | null;
};

type Match = {
  id: number;
  status: string;
  teamA?: Team | null;
  teamB?: Team | null;
  score: any;
};

type Props = {
  eventId: number;
  organizerId: number | null;
};

export default function PadelTournamentSection({ eventId, organizerId }: Props) {
  const [teams, setTeams] = useState<Team>([]);
  const [matches, setMatches] = useState<Match>([]);
  const [rankings, setRankings] = useState<{ position: number; points: number; player: Player }>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [players, setPlayers] = useState<Player[]>([]);
  const [teamForm, setTeamForm] = useState({ p1: "", p2: "", p1Id: "", p2Id: "" });
  const [format, setFormat] = useState<"TODOS_CONTRA_TODOS" | "QUADRO_ELIMINATORIO">("TODOS_CONTRA_TODOS");

  async function fetchTeams() {

```

```

const res = await fetch(`/api/padel/teams?eventId=${eventId}`);
const json = await res.json();
if (!res.ok || !json?.ok) throw new Error(json?.error || "Erro ao carregar equipas");
setTeams(json.items);
}

async function fetchPlayers() {
  if (!organizerId) return;
  const res = await fetch(`/api/padel/players?organizerId=${organizerId}`);
  const json = await res.json();
  if (!res.ok || !json?.ok) throw new Error(json?.error || "Erro ao carregar jogadores");
  setPlayers(json.items);
}

async function fetchMatches() {
  const res = await fetch(`/api/padel/matches?eventId=${eventId}`);
  const json = await res.json();
  if (!res.ok || !json?.ok) throw new Error(json?.error || "Erro ao carregar jogos");
  setMatches(json.items);
}

async function fetchRankings() {
  const res = await fetch(`/api/padel/rankings?eventId=${eventId}`);
  const json = await res.json();
  if (!res.ok || !json?.ok) throw new Error(json?.error || "Erro ao carregar ranking");
  setRankings(json.items);
}

useEffect(() => {
  (async () => {
    try {
      setLoading(true);
      await Promise.all([fetchTeams(), fetchMatches(), fetchPlayers(), fetchRankings()]);
    } catch (err) {
      console.error(err);
      setError("Erro ao carregar torneio.");
    } finally {
      setLoading(false);
    }
  })();
}, [eventId, organizerId]);

const addTeam = async () => {
  if (!teamForm.p1.trim() && !teamForm.p1Id) return setError("Indica o jogador 1.");
  if (!teamForm.p2.trim() && !teamForm.p2Id) return setError("Indica o jogador 2.");
  setError(null);
  const res = await fetch("/api/padel/teams", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      eventId,
      player1Name: teamForm.p1,
      player2Name: teamForm.p2,
      player1Id: teamForm.p1Id || undefined,
      player2Id: teamForm.p2Id || undefined,
    }),
  });
  const json = await res.json();
  if (!res.ok || !json?.ok) {
    setError(json?.error || "Erro ao criar equipa");
    return;
  }
  setTeams((prev) => [json.team, ...prev]);
  setTeamForm({ p1: "", p2: "", p1Id: "", p2Id: "" });
};

const generateMatches = async () => {
  setError(null);
  const res = await fetch("/api/padel/matches/generate", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ eventId, format }),
  });
  const json = await res.json();
  if (!res.ok || !json?.ok) {
    setError(json?.error || "Erro ao gerar jogos");
    return;
  }
  setMatches(json.matches);
}

```

```

};

const updateMatchStatus = async (matchId: number, status: string) => {
  const res = await fetch("/api/padel/matches", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ id: matchId, status }),
  });
  const json = await res.json();
  if (!res.ok || !json?.ok) {
    setError(json?.error || "Erro ao atualizar jogo");
    return;
  }
  setMatches((prev) => prev.map((m) => (m.id === matchId ? json.match : m)));
};

const generateRanking = async () => {
  setError(null);
  const res = await fetch("/api/padel/rankings", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ eventId }),
  });
  const json = await res.json();
  if (!res.ok || !json?.ok) {
    setError(json?.error || "Erro ao gerar ranking");
    return;
  }
  await fetchRankings();
};

return (
  <div className="space-y-4 rounded-2xl border border-white/10 bg-black/40 p-4">
    <div className="flex items-center justify-between">
      <div>
        <p>Torneio (MVP)</p>
        <h3>Equipes & Jogos</h3>
      </div>
      <select value={format}>
        <option value="TODOS_CONTRA_TODOS">Todos contra todos</option>
        <option value="QUADRO_ELIMINATORIO">Quadro eliminatório</option>
      </select>
    </div>
    {error && <p>{error}</p>}
    {loading && <p>A carregar torneio...</p>}

    <div className="grid gap-3 md:grid-cols-[1.1fr_1fr]">
      <div className="space-y-3 rounded-xl border border-white/10 bg-white/5/20 p-3">
        <p>Criar equipa</p>
        <div className="grid grid-cols-1 gap-2 sm:grid-cols-2">
          <input value={teamForm.p1} onChange={(e) => setTeamForm((p) => ({ ...p, p1: e.target.value }))} placeholder="Jogador 1 (novo)" className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none" />
          <select value={teamForm.p1Id} onChange={(e) => setTeamForm((p) => ({ ...p, p1Id: e.target.value }))} className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none" >
            <option value="">Selecionar jogador existente</option>
            {players.map((pl) => (
              <option key={pl.id} value={pl.id}>
                {pl.fullName} {pl.level ? `(${pl.level})` : ""}</option>
            ))}
          </select>
          <input value={teamForm.p2} onChange={(e) => setTeamForm((p) => ({ ...p, p2: e.target.value }))} placeholder="Jogador 2 (novo)" className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none" />
        </div>
      </div>
    </div>
  </div>
);

```

```

/>
<select
  value={teamForm.p2Id}
  onChange={(e) => setTeamForm((p) => ({ ...p, p2Id: e.target.value }))}
  className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none"
>
  <option value="">Seleccionar jugador existente</option>
  {players.map((pl) => (
    <option key={pl.id} value={pl.id}>
      {pl.fullName} {pl.level ? `(${pl.level})` : ""}
    </option>
  )))
</select>
</div>
<button
  onClick={addTeam}
  className="w-fit rounded-full bg-white/10 px-4 py-2 text-sm text-white hover:bg-white/15"
>
  Guardar equipa
</button>
<div className="space-y-2 max-h-56 overflow-auto text-sm">
  {teams.length === 0 && <p className="text-white/60">Sem equipas ainda.</p>}
  {teams.map((t) => (
    <div key={t.id} className="rounded-lg border border-white/10 bg-black/30 px-3 py-2">
      <p className="font-semibold text-white">
        {t.player1?.fullName ?? "Jogador 1"} & {t.player2?.fullName ?? "Jogador 2"}
      </p>
    </div>
  )))
</div>
</div>

<div className="space-y-3 rounded-xl border border-white/10 bg-white/5/20 p-3">
<div className="flex items-center justify-between">
  <p className="text-[12px] uppercase tracking-[0.2em] text-white/60">Jogos</p>
  <button
    onClick={generateMatches}
    className="rounded-full bg-white/10 px-3 py-1.5 text-[12px] text-white hover:bg-white/20"
  >
    Gerar jogos
  </button>
</div>
<div className="space-y-2 max-h-60 overflow-auto text-sm">
  {matches.length === 0 && <p className="text-white/60">Sem jogos ainda.</p>}
  {matches.map((m) => (
    <div key={m.id} className="rounded-lg border border-white/10 bg-black/30 px-3 py-2">
      <div className="flex items-center justify-between gap-2">
        <div className="space-y-1">
          <p className="font-semibold text-white">
            {(m.teamA?.player1?.fullName ?? "Equipa A")} vs {m.teamB?.player1?.fullName ?? "Equipa B"}
          </p>
          <p className="text-[11px] text-white/60">Estado: {m.status}</p>
        </div>
        <select
          value={m.status}
          onChange={(e) => updateMatchStatus(m.id, e.target.value)}
          className="rounded-full border border-white/15 bg-white/5 px-2 py-1 text-[11px] text-white/80"
        >
          <option value="PENDING">Pendente</option>
          <option value="IN_PROGRESS">Em jogo</option>
          <option value="DONE">Concluido</option>
          <option value="CANCELLED">Cancelado</option>
        </select>
      </div>
    </div>
  )))
</div>
<div className="flex justify-end">
  <button
    onClick={generateRanking}
    className="rounded-full border border-white/20 px-3 py-1.5 text-[12px] text-white/80 hover:bg-white/10"
  >
    Gerar ranking deste torneio
  </button>
</div>
<div className="space-y-2 rounded-xl border border-white/10 bg-black/30 p-3">
  <p className="text-[12px] uppercase tracking-[0.2em] text-white/60">Ranking do torneio</p>
  {rankings.length === 0 && <p className="text-white/60 text-sm">Sem ranking gerado.</p>}

```

```

        {rankings.map((r) => (
            <div key={r.player.id} className="flex items-center justify-between rounded-lg border border-white/10 bg-white/5
px-3 py-2">
                <div className="flex items-center gap-2">
                    <span className="text-[12px] text-white/60">{r.position}</span>
                    <div>
                        <p className="font-semibold text-white">{r.player.fullName}</p>
                        <p className="text-[11px] text-white/60">{r.player.level || "Nivel?"}</p>
                    </div>
                </div>
                <span className="text-[12px] font-semibold text-[#6BFFFF]">{r.points} pts</span>
            </div>
        )))
    </div>
</div>
</div>
</div>
);
}

```

app/organizador/(dashboard)/eventos/[id]/PadelTournamentTabs.tsx

```

"use client";

import { useState } from "react";
import useSWR from "swr";

type Pairing = {
    id: number;
    pairingStatus: string;
    paymentMode: string;
    categoryId?: number | null;
    slots: { id: number; slotRole: string; slotStatus: string; paymentStatus: string; playerProfile?: { displayName?: string | null; fullName?: string | null } | null }[];
    inviteToken?: string | null;
};

type Match = {
    id: number;
    status: string;
    pairingA?: Pairing | null;
    pairingB?: Pairing | null;
    scoreSets?: Array<{ teamA: number; teamB: number }> | null;
};

type Standings = Record<string, Array<{ pairingId: number; points: number; wins: number; losses: number; setsFor: number; setsAgainst: number }>>;
type CategoryMeta = { name?: string; categoryId?: number | null; capacity?: number | null; registrationType?: string | null };

const fetcher = (url: string) => fetch(url).then((r) => r.json());

function nameFromSlots(pairing?: Pairing | null) {
    if (!pairing) return "-";
    const names = pairing.slots
        .map((s) => s.playerProfile?.displayName || s.playerProfile?.fullName)
        .filter(Boolean) as string[];
    return names.length ? names.join(" / ") : "Dupla incompleta";
}

export default function PadelTournamentTabs({ eventId, categoriesMeta }: { eventId: number; categoriesMeta?: CategoryMeta[] }) {
    const [tab, setTab] = useState<"duplas" | "jogos" | "rankings">("duplas");

    const { data: pairingsRes } = useSWR(eventId ? `/api/padel/pairings?eventId=${eventId}` : null, fetcher);
    const { data: matchesRes, mutate: mutateMatches } = useSWR(eventId ? `/api/padel/matches?eventId=${eventId}` : null, fetcher);
    const { data: standingsRes } = useSWR(eventId ? `/api/padel/standings?eventId=${eventId}` : null, fetcher);

    const pairings: Pairing[] = pairingsRes?.pairings ?? [];
    const matches: Match[] = matchesRes?.items ?? [];
    const standings: Standings = standingsRes?.standings ?? {};

    const categoryStats = (() => {
        const metaMap = new Map<number | null, CategoryMeta>();
        (categoriesMeta || []).forEach((m) => {
            const key = Number.isFinite(m.categoryId as number) ? (m.categoryId as number) : null;
            metaMap.set(key, m);
        });
    })();

```

```

const counts = new Map<number | null, number>();
pairings.forEach((p) => {
  const key = Number.isFinite(p.categoryId as number) ? (p.categoryId as number) : null;
  counts.set(key, (counts.get(key) || 0) + 1);
});
const rows: Array<{ key: number | null; label: string; count: number; capacity: number | null }> = [];
const keys = new Set([...counts.keys(), ...metaMap.keys()]);
keys.forEach((key) => {
  const meta = metaMap.get(key);
  const label = meta?.name || (key === null ? "Categoria" : `Categoria ${key}`);
  const capacity = meta?.capacity ?? null;
  rows.push({ key, label, count: counts.get(key) || 0, capacity });
});
return rows;
}();

const matchesSummary = {
  pending: matches.filter((m) => m.status === "PENDING").length,
  live: matches.filter((m) => m.status === "LIVE").length,
  done: matches.filter((m) => m.status === "DONE").length,
};

async function submitResult(matchId: number, scoreText: string) {
  const sets = scoreText
    .split(",")
    .map((p) => p.trim())
    .filter(Boolean)
    .map((s) => s.split("-").map((v) => Number(v.trim())))
    .filter((arr) => arr.length === 2 && Number.isFinite(arr[0]) && Number.isFinite(arr[1]))
    .map(([a, b]) => ({ teamA: a, teamB: b }));
  await fetch(`/api/padel/matches`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ id: matchId, status: "DONE", score: { sets } }),
  });
  mutateMatches();
}

return (
  <section className="rounded-2xl border border-white/10 bg-black/40 p-4 space-y-4 mt-6">
    <div className="grid gap-3 md:grid-cols-3">
      <div className="rounded-xl border border-white/10 bg-white/5 p-3 space-y-1">
        <p className="text-[11px] uppercase tracking-[0.16em] text-white/60">Inscrições Padel</p>
        <p className="text-2xl font-semibold text-white">{pairings.length}</p>
        <p className="text-[12px] text-white/70">
          Completas: {pairings.filter((p) => p.pairingStatus === "COMPLETE").length} · Pendentes:{" "}
          {pairings.filter((p) => p.pairingStatus !== "COMPLETE").length}
        </p>
      </div>
      <div className="rounded-xl border border-white/10 bg-white/5 p-3 space-y-1">
        <p className="text-[11px] uppercase tracking-[0.16em] text-white/60">Jogos</p>
        <p className="text-2xl font-semibold text-white">{matches.length}</p>
        <p className="text-[12px] text-white/70">
          Pendentes {matchesSummary.pending} · Live {matchesSummary.live} · Terminados {matchesSummary.done}
        </p>
      </div>
      <div className="rounded-xl border border-white/10 bg-white/5 p-3 space-y-2">
        <p className="text-[11px] uppercase tracking-[0.16em] text-white/60">Categorias</p>
        <div className="space-y-1 text-[12px] text-white/75">
          {categoryStats.length === 0 && <p className="text-white/60">Sem categorias definidas.</p>}
          {categoryStats.map((c) => {
            const occupancy = c.capacity ? Math.min(100, Math.round((c.count / c.capacity) * 100)) : null;
            return (
              <div key={`${c.key ?? "default"}`} className="flex items-center justify-between gap-2">
                <span className="text-white">{c.label}</span>
                <span className="text-white/70">
                  {c.count} equipa{c.count === 1 ? "" : "s"} {c.capacity ? `· ${occupancy}%` : ""}
                </span>
              </div>
            );
          ))}
        </div>
      </div>
    </div>
  </section>
  <div className="flex items-center gap-2 text-[12px]">
    {[...]
  
```

```

        { key: "duplas", label: "Duplas" },
        { key: "jogos", label: "Jogos" },
        { key: "rankings", label: "Rankings" },
    ].map((t) => (
        <button
            key={t.key}
            onClick={() => setTab(t.key as "duplas" | "jogos" | "rankings")}
            className={`rounded-full px-3 py-1 border ${tab === t.key ? "bg-white text-black font-semibold" : "border-white/20 text-white/75"}`}
        >
            {t.label}
        </button>
    )));
</div>

{tab === "duplas" && (
    <div className="space-y-2">
        {pairings.length === 0 && <p className="text-sm text-white/70">Ainda não há duplas.</p>}
        {pairings.map((p) => (
            <div key={p.id} className="rounded-xl border border-white/15 bg-white/5 p-3 text-sm flex items-center justify-between">
                <div>
                    <p className="font-semibold">{nameFromSlots(p)}</p>
                    <p className="text-[11px] text-white/60">{p.pairingStatus} · {p.paymentMode}</p>
                </div>
                {p.inviteToken && (
                    <button
                        type="button"
                        onClick={() => navigator.clipboard.writeText(` ${window.location.origin}/eventos/${event.slug}?token=${p.inviteToken}`)}
                        className="rounded-full border border-white/20 px-3 py-1 text-[12px] text-white/80 hover:bg-white/10"
                    >
                        Copiar convite
                    </button>
                )}
            </div>
        )))
    </div>
)
}

{tab === "jogos" && (
    <div className="space-y-3">
        {matches.length === 0 && <p className="text-sm text-white/70">Sem jogos gerados.</p>}
        {matches.map((m) => (
            <div key={m.id} className="rounded-xl border border-white/15 bg-white/5 p-3 text-sm space-y-2">
                <div className="flex items-center justify-between">
                    <p className="font-semibold">{nameFromSlots(m.pairingA as Pairing)} vs {nameFromSlots(m.pairingB as Pairing)}</p>
                </div>
                <span className="text-[11px] text-white/60">{m.status}</span>
            </div>
            <p className="text-[12px] text-white/70">Resultado: {m.scoreSets?.length ? m.scoreSets.map((s) => `${s.teamA}-${s.teamB}`).join(", ") : "-"}</p>
            <div className="flex items-center gap-2 text-[12px]">
                <input
                    type="text"
                    placeholder="6-3, 6-4"
                    className="flex-1 rounded-lg border border-white/15 bg-black/30 px-2 py-1"
                    onBlur={(e) => {
                        const v = e.target.value.trim();
                        if (v) submitResult(m.id, v);
                    }}
                />
                <span className="text-white/50">(guardar ao sair do campo)</span>
            </div>
        )))
    </div>
)
}

{tab === "rankings" && (
    <div className="space-y-3 text-sm">
        {Object.keys(standings).length === 0 && <p className="text-white/70">Sem standings.</p>}
        {Object.entries(standings).map(([groupKey, rows]) => (
            <div key={groupKey} className="rounded-xl border border-white/15 bg-white/5 p-3 space-y-2">
                <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">{groupKey}</p>
                <div className="space-y-1">
                    {rows.map((r, idx) => (
                        <div key={r.pairingId} className="flex items-center justify-between">
                            <div>
                                <p>{r.name}</p>
                                <p>{r.wins}-{r.losses}</p>
                            </div>
                            <div>
                                <p>{r.rank}</p>
                                <p>{r.rank}</p>
                            </div>
                        </div>
                    )))
                </div>
            </div>
        )))
    </div>
)
}

```

```

        <span className="flex items-center gap-2 text-white/85">
          <span className="text-[11px] text-white/60">#{idx + 1}</span>
          <span>Dupla {r.pairingId}</span>
        </span>
        <span className="text-white/70">Pts {r.points} · {r.wins}V{r.losses}D · Sets {r.setsFor}-{r.setsAgainst}
      </span>
    </div>
  )})
</div>
</div>
))
</div>
)
</section>
);
}

```

app/organizador/(dashboard)/eventos/[id]/page.tsx

```

// app/organizador/eventos/[id]/page.tsx
/* eslint-disable @next/next/no-html-link-for-pages */
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { notFound, redirect } from "next/navigation";
import PadelTournamentTabs from "./PadelTournamentTabs";
import type { Event, TicketType } from "@prisma/client";

type PageProps = {
  params: Promise<{
    id: string;
  }>;
};

type EventWithTickets = Event & {
  ticketTypes: TicketType[];
};

export default async function OrganizerEventDetailPage({ params }: PageProps) {
  const resolved = await params;

  // 1) Garante auth
  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();

  if (error || !data?.user) {
    redirect("/login");
  }

  const userId = data.user.id;

  const organizer = await prisma.organizer.findFirst({
    where: { userId },
  });

  if (!organizer) {
    redirect("/organizador");
  }

  const eventId = Number.parseInt(resolved.id, 10);
  if (!Number.isFinite(eventId)) {
    notFound();
  }

  // 2) Buscar evento + tipos de bilhete (waves)
  const event = (await prisma.event.findFirst({
    where: {
      id: eventId,
      organizerId: organizer.id,
    },
    include: {
      ticketTypes: {
        orderBy: {
          sortOrder: "asc",
        },
      },
      padelTournamentConfig: {
    
```

```

        include: { club: true },
    },
}),
)) as (EventWithTickets & { padelTournamentConfig: { numberOfCourts: number; club?: { name: string; city: string | null;
address: string | null } | null; partnerClubIds?: number[]; advancedSettings?: Record<string, unknown> | null } } | null;

if (!event) {
    notFound();
}

const now = new Date();

// 3) Métricas agregadas
const totalWaves = event.ticketTypes.length;
const totalTicketsSold = event.ticketTypes.reduce(
    (sum, t) => sum + t.soldQuantity,
    0,
);
const totalStock = event.ticketTypes.reduce(
    (sum, t) =>
        sum +
        (t.totalQuantity !== null && t.totalQuantity !== undefined
            ? t.totalQuantity
            : 0),
    0,
);
const overallOccupancy =
    totalStock > 0
        ? Math.min(100, Math.round((totalTicketsSold / totalStock) * 100))
        : null;

const totalRevenueCents = event.ticketTypes.reduce(
    (sum, t) => sum + t.soldQuantity * (t.price ?? 0),
    0,
);
const totalRevenue = (totalRevenueCents / 100).toFixed(2);

const cheapestWave = event.ticketTypes.length
    ? event.ticketTypes.reduce((min, t) =>
        ((t.price ?? 0) < (min.price ?? 0) ? t : min)
    )
    : null;

const formatDateTime = (d: Date | null | undefined) => {
    if (!d) return null;
    return new Date(d).toLocaleString("pt-PT", {
        day: "2-digit",
        month: "short",
        hour: "2-digit",
        minute: "2-digit",
    });
};

const formatMoney = (cents: number) =>
    `${cents / 100}.toFixed(2)` .replace(".", ",");

const startDateFormatted = formatDateTime(event.startsAt);
const endDateFormatted = formatDateTime(event.endsAt);

const tournamentState =
    event.status === "CANCELLED"
        ? "Cancelado"
        : event.status === "FINISHED"
        ? "Terminado"
        : event.status === "DRAFT"
        ? "Óctavo"
        : "Público";

const partnerClubs =
    event.padelTournamentConfig?.partnerClubIds?.length
        ? await prisma.padelClub.findMany({
            where: { id: { in: event.padelTournamentConfig.partnerClubIds as number[] } },
            select: { id: true, name: true, city: true },
        })
        : [];
const advancedSettings = event.padelTournamentConfig?.advancedSettings as
| {

```

```

maxEntriesTotal?: number | null;
waitlistEnabled?: boolean;
allowSecondCategory?: boolean;
allowCancelGames?: boolean;
gameDurationMinutes?: number | null;
courtsFromClubs?: Array<{ id?: number; clubId?: number | null; clubName?: string | null; name?: string | null; indoor?: boolean }>;
staffFromClubs?: Array<{ clubName?: string | null; email?: string | null; role?: string | null }>;
categoriesMeta?: Array<{ name?: string; categoryId?: number | null; capacity?: number | null; registrationType?: string | null }>;
}
| null;
const categoriesMeta = advancedSettings?.categoriesMeta ?? [];

const timeline = [
  { key: "OCULTO", label: "Ócrito", active: ["DRAFT"].includes(event.status), done: event.status !== "DRAFT" },
  { key: "INSCRICOES", label: "Inscrições", active: event.status === "PUBLISHED", done: ["PUBLISHED", "FINISHED", "CANCELLED"].includes(event.status) },
  { key: "PUBLICO", label: "Público", active: event.status === "PUBLISHED", done: ["PUBLISHED", "FINISHED", "CANCELLED"].includes(event.status) },
  { key: "TERMINADO", label: "Terminado", active: event.status === "FINISHED", done: event.status === "FINISHED" },
];

return (
  <div className="mx-auto max-w-6xl px-4 py-8 md:px-6 lg:px-8 space-y-7 text-white">
    <div className="flex flex-col gap-2 sm:flex-row sm:items-center sm:justify-between">
      <div className="space-y-1">
        <p>uppercase tracking-[0.3em] text-white/60</p>Gestão de evento</p>
        <h1>text-2xl font-semibold tracking-tight>Detalhes & waves</h1>
        <p>text-sm text-white/70 line-clamp-2</p>{event.title}</p>
      </div>
      <div className="flex flex-wrap gap-2 text-[11px]">
        <a href="/organizador/eventos" className="px-3 py-1.5 rounded-xl border border-white/20 bg-white/5 text-white/80 hover:bg-white/10 transition">
          ← Voltar à lista
        </a>
        <a href={`/eventos/${event.slug}`} className="px-3 py-1.5 rounded-xl bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] font-semibold text-black hover:scale-[1.03] active:scale-95 transition-transform shadow-[0_0_22px_rgba(107,255,255,0.7)]">
          Ver página pública
        </a>
      </div>
    </div>
  </div>

  <div className="grid grid-cols-1 gap-5 md:grid-cols-[minmax(0,1.7fr)_minmax(0,1.1fr)]">
    <div className="rounded-2xl border border-white/14 bg-black/45 backdrop-blur-xl p-5 space-y-3">
      <div>
        <h2>text-2xl font-semibold tracking-tight>{event.title}</h2>
        <p>mt-1 text-[11px] text-white/65</p>
        {startDateFormatted}
        {endDateFormatted ? ` - ${endDateFormatted}` : ""} •{" "}
        {event.locationName}
      </p>
      {event.address && (
        <p>text-[11px] text-white/45</p>
        {event.address}
      </p>
      )}
    </div>
    {event.coverImageUrl && (
      // eslint-disable-next-line @next/next/no-img-element
      <img src={event.coverImageUrl} alt={event.title} className="hidden md:block w-28 h-20 rounded-xl object-cover border border-white/20" />
    )}
  </div>

  <div className="mt-2 flex flex-wrap items-center gap-2 text-[11px]">
    {timeline.map((step, idx) => (

```

```

<div key={step.key} className="flex items-center gap-2">
  <span
    className={`inline-flex items-center gap-1 rounded-full border px-2 py-1 ${step.done
      ? "border-emerald-400/60 bg-emerald-400/15 text-emerald-100"
      : step.active
      ? "border-white/30 bg-white/10 text-white"
      : "border-white/15 bg-black/30 text-white/60"
    }`}
  >
    {step.label}
  </span>
  {idx < timeline.length - 1 && <span className="text-white/25">&lt;/span>}
</div>
)}
)
</div>

{cheapestWave && (
  <p className="mt-1 text-[11px] text-white/70">
    Preço a partir de{" "}
    <span className="font-semibold">
      {formatMoney(cheapestWave.price ?? 0)}
    </span>{" "}
    ({totalWaves} wave{totalWaves !== 1 ? "s" : ""})
  </p>
)

<p className="mt-1 text-[11px] text-white/60 line-clamp-3">
  {event.description}
</p>

<p className="mt-2 text-[10px] text-white/40 font-mono">
  ID: {event.id} • Slug: {event.slug}
</p>

{event.padelTournamentConfig && (
  <div className="mt-3 grid gap-2 rounded-xl border border-white/10 bg-white/5 p-3 text-sm">
    <div className="flex items-center justify-between">
      <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Torneio de Padel</p>
      <span className="rounded-full border border-white/20 bg-white/10 px-3 py-1 text-[12px]">
        {tournamentState}
      </span>
    </div>
    <p className="font-semibold">
      {event.padelTournamentConfig.club?.name ?? "Clube não definido"}
    </p>
    <p className="text-white/70">
      {event.padelTournamentConfig.club?.city ?? "Cidade -"} ·{" "}
      {event.padelTournamentConfig.club?.address ?? "Morada em falta"}
    </p>
    <p className="text-white/75">
      Courts usados: {event.padelTournamentConfig.numberOfCourts}
    </p>
    {partnerClubs.length > 0 && (
      <div className="text-[12px] text-white/70">
        <p className="text-[11px] uppercase tracking-[0.16em] text-white/55 mt-2">Clubes parceiros</p>
        <div className="flex flex-wrap gap-2">
          {partnerClubs.map((c) => (
            <span key={c.id} className="rounded-full border border-white/15 bg-white/10 px-2 py-1">
              {c.name} {c.city ? `· ${c.city}` : ""}
            </span>
          )))
        </div>
      </div>
    )}
    {advancedSettings && (
      <div className="text-[12px] text-white/70">
        <p className="text-[11px] uppercase tracking-[0.16em] text-white/55 mt-2">Opções avançadas</p>
        <p className="text-white/75">
          Limite total: {advancedSettings.maxEntriesTotal ?? "-"} · Waitlist:{" "}
          {advancedSettings.waitlistEnabled ? "on" : "off"} · 2ª categoria:{" "}
          {advancedSettings.allowSecondCategory ? "sim" : "não"} · Cancelar jogos:{" "}
          {advancedSettings.allowCancelGames ? "sim" : "não"} · Jogo padrão:{" "}
          {advancedSettings.gameDurationMinutes ?? "-"} min
        </p>
        {advancedSettings.courtsFromClubs?.length ? (
          <div className="mt-2 space-y-1">
            <p className="text-[11px] uppercase tracking-[0.14em] text-white/55">Courts incluídos</p>

```

```

<div className="flex flex-wrap gap-2">
  {advancedSettings.courtsFromClubs.map((c, idx) =>
    <span key={`${c.id}-${idx}`} className="rounded-full border border-white/15 bg-white/10 px-2 py-1">
      {c.name || "Court"} · {c.clubName || `Clube ${c.clubId ?? ""}`} {c.indoor ? "(Indoor)" : ""}
    </span>
  )))
</div>
</div>
) : null}
{advancedSettings.staffFromClubs?.length ? (
<div className="mt-2 space-y-1">
  <p className="text-[11px] uppercase tracking-[0.14em] text-white/55">Staff herdado</p>
  <div className="flex flex-wrap gap-2">
    {advancedSettings.staffFromClubs.map((s, idx) =>
      <span key={`${s.email}-${idx}`} className="rounded-full border border-white/15 bg-white/10 px-2 py-1">
        {s.email || s.role || "Staff"} · {s.role || "Role"} · {s.clubName || "Clube"}
      </span>
    )))
  </div>
</div>
) : null}
</div>
)}
</div>
)
)
</div>
</div>

<div className="grid grid-cols-1 gap-4 sm:grid-cols-2">
<div className="rounded-2xl border border-[#6BFFFF]/40 bg-[#02040b]/95 backdrop-blur-xl px-4 py-3.5">
  <p className="text-[11px] text-[#6BFFFF]/80">
    Bilhetes vendidos
  </p>
  <p className="mt-1 text-2xl font-semibold tracking-tight">
    {totalTicketsSold}
  </p>
  {overallOccupancy !== null && (
    <p className="mt-1 text-[11px] text-white/65">
      {overallOccupancy}% de ocupação (stock total {totalStock})
    </p>
  )}
  {overallOccupancy !== null && (
    <div className="mt-2 h-1.5 rounded-full bg-white/10 overflow-hidden">
      <div
        className="h-full rounded-full bg-gradient-to-r from-[#6BFFFF] to-[#FF00C8]"
        style={{ width: `${overallOccupancy}%` }}
      />
    </div>
  )}
</div>
</div>

<div className="rounded-2xl border border-white/14 bg-white/5 backdrop-blur-xl px-4 py-3.5">
  <p className="text-[11px] text-white/65">Receita bruta</p>
  <p className="mt-1 text-2xl font-semibold tracking-tight">
    {totalRevenue.replace(".", ",")}{` €`}
  </p>
  <p className="mt-1 text-[11px] text-white/55">
    Calculado com base em preço x bilhetes vendidos, por wave.
  </p>
  <p className="mt-1 text-[10px] text-white/40">
    Nota: valores em modo de teste – integrar com relatórios reais
    mais tarde.
  </p>
</div>
</div>
</div>

<section className="rounded-2xl border border-white/12 bg-black/40 backdrop-blur-xl p-5 space-y-4">
<div className="flex items-center justify-between gap-2">
  <div>
    <h2 className="text-sm font-semibold text-white/90">
      Waves & bilhetes
    </h2>
    <p className="text-[11px] text-white/65">
      Visão por wave: estado, stock, vendas e receita individual.
    </p>
  </div>
</div>
</div>

```

```

{event.ticketTypes.length === 0 && (
  <div className="mt-2 rounded-xl border border-dashed border-white/20 bg-white/5 px-4 py-4 text-[11px] text-white/70">
    Este evento ainda não tem waves configuradas. Usa o criador de
    eventos para adicionar bilhetes.
  </div>
)}


{event.ticketTypes.length > 0 && (
  <div className="mt-2 grid grid-cols-1 gap-4 md:grid-cols-2">
    {event.ticketTypes.map((ticket) => {
      const remaining =
        ticket.totalQuantity !== null &&
        ticket.totalQuantity !== undefined
        ? ticket.totalQuantity - ticket.soldQuantity
        : null;

      const occupancy =
        ticket.totalQuantity && ticket.totalQuantity > 0
        ? Math.min(
          100,
          Math.round(
            (ticket.soldQuantity / ticket.totalQuantity) * 100,
          ),
        )
        : null;
    });

    // Determinar estado da wave
    let statusLabel = "A vender";
    let statusBadgeClass =
      "bg-emerald-500/10 border-emerald-400/70 text-emerald-100";
    const nowTime = now.getTime();
    const startsAtTime = ticket.startsAt
      ? new Date(ticket.startsAt).getTime()
      : null;
    const endsAtTime = ticket.endsAt
      ? new Date(ticket.endsAt).getTime()
      : null;

    if (
      ticket.totalQuantity !== null &&
      ticket.totalQuantity !== undefined &&
      ticket.soldQuantity >= ticket.totalQuantity
    ) {
      statusLabel = "Esgotado";
      statusBadgeClass =
        "bg-red-500/10 border-red-400/70 text-red-100";
    } else if (startsAtTime && nowTime < startsAtTime) {
      statusLabel = "Em breve";
      statusBadgeClass =
        "bg-amber-500/10 border-amber-400/70 text-amber-100";
    } else if (endsAtTime && nowTime > endsAtTime) {
      statusLabel = "Encerrado";
      statusBadgeClass =
        "bg-white/8 border-white/30 text-white/75";
    }

    const startsAtLabel = formatDateTime(ticket.startsAt);
    const endsAtLabel = formatDateTime(ticket.endsAt);

    const revenueCents =
      ticket.soldQuantity * (ticket.price ?? 0);
    const revenue = (revenueCents / 100).toFixed(2);

    return (
      <article
        key={ticket.id}
        className="rounded-xl border border-white/14 bg-gradient-to-br from-white/5 via-black/80 to-black/95 px-4 py-4 flex flex-col gap-3">
        >
        <div className="flex items-start justify-between gap-3">
          <div>
            <h3 className="text-sm font-semibold text-white/95">
              {ticket.name}
            </h3>
            {ticket.description && (
              <p className="mt-0.5 text-[11px] text-white/60 line-clamp-2">
                {ticket.description}
              </p>
            )}
          </div>
        </div>
      </article>
    );
  });
)
}

```

```

        </p>
    )}
<p className="mt-1 text-[10px] text-white/45 font-mono">
    ID: {ticket.id}
</p>
</div>

<div className="flex flex-col items-end gap-1">
    <span
        className={`px-2 py-1 rounded-full border text-[10px] ${statusBadgeClass}`}
    >
        {statusLabel}
    </span>
    <span className="px-2 py-0.5 rounded-full bg-white/5 border border-white/20 text-[10px] text-white/80">
        {formatMoney(ticket.price ?? 0)}
    </span>
</div>
</div>

<div className="flex flex-wrap items-center gap-3 text-[10px] text-white/65">
    {startsAtLabel && (
        <span>
            ⚡ Abre:" "
            <span className="text-white/85">
                {startsAtLabel}
            </span>
        </span>
    )}
    {endsAtLabel && (
        <span>
            Fecha:" "
            <span className="text-white/85">{endsAtLabel}</span>
        </span>
    )}
    {!startsAtLabel && !endsAtLabel && (
        <span>Sem janela definida (sempre ativo).</span>
    )}
</div>

<div className="flex flex-wrap items-center justify-between gap-3 text-[11px] text-white/80">
<div className="flex flex-col gap-1">
    <div className="flex items-center gap-1.5">
        <span className="text-white/60">
            Vendidos / stock:
        </span>
        <span className="font-semibold">
            {ticket.soldQuantity}
            {ticket.totalQuantity
            ? ` / ${ticket.totalQuantity}`
            : " / ~"}`

        </span>
    <span>
        {remaining !== null && remaining >= 0 && (
            <span className="text-[10px] text-white/55">
                ({remaining} restantes)
            </span>
        )}
    </span>
</div>
    {occupancy !== null && (
        <div className="h-1.5 w-40 rounded-full bg-white/10 overflow-hidden">
            <div
                className="h-full rounded-full bg-gradient-to-r from-[#6BFFFF] to-[#FF00C8]"
                style={{ width: `${occupancy}%` }}
            />
        </div>
    )}
</div>

<div className="flex flex-col items-end gap-1 text-right">
    <span className="text-[10px] text-white/60">
        Receita estimada
    </span>
    <span className="text-sm font-semibold">
        {revenue.replace(".", ",")} €
    </span>
</div>
</div>

```

```

        <p className="mt-1 text-[10px] text-white/40">
          Funcionalidades avançadas como lista de compras por
          utilizador, links de promotores e tracking detalhado por
          wave podem ser geridas na área de gestão avançada do
          evento.
        </p>
      </article>
    );
  )}
</div>
)
</section>

{event.templateType === "PADEL" && (
  <PadelTournamentTabs eventId={event.id} categoriesMeta={categoriesMeta} />
)
</div>
);
}
}

```

app/organizador/(dashboard)/eventos/EventEditClient.tsx

```

"use client";

import { useEffect, useMemo, useRef, useState } from "react";
import Link from "next/link";
import useSWR from "swr";
import { InlineDateTimePicker } from "@app/components/forms/InlineDateTimePicker";
import { TicketTypeStatus } from "@prisma/client";
import { useUser } from "@app/hooks/useUser";

type ToastTone = "success" | "error";
type Toast = { id: number; message: string; tone: ToastTone };

type TicketTypeUI = {
  id: number;
  name: string;
  description: string | null;
  price: number;
  currency: string;
  totalQuantity: number | null;
  soldQuantity: number;
  status: TicketTypeStatus;
  startsAt: string | null;
  endsAt: string | null;
};

type EventEditClientProps = {
  event: {
    id: number;
    title: string;
    description: string | null;
    startsAt: string;
    endsAt: string;
    locationName: string | null;
    locationCity: string | null;
    address: string | null;
    templateType: string | null;
    isFree: boolean;
    coverImageUrl: string | null;
    feeModeOverride?: string | null;
    platformFeeBpsOverride?: number | null;
    platformFeeFixedCentsOverride?: number | null;
    payoutMode?: string | null;
  };
  tickets: TicketTypeUI[];
  eventHasTickets?: boolean;
};

const fetcher = (url: string) => fetch(url).then((res) => res.json());

export function EventEditClient({ event, tickets }: EventEditClientProps) {
  const { user, profile } = useUser();
  const { data: organizerStatus } = useSWR<{ paymentsStatus?: string }>(
    user ? "/api/organizador/me" : null,
    fetcher,
  );
}
```

```

    { revalidateOnFocus: false }
);
const [title, setTitle] = useState(event.title);
const [description, setDescription] = useState(event.description ?? "");
const [startsAt, setStartsAt] = useState(event.startsAt);
const [endsAt, setEndsAt] = useState(event.endsAt);
const [locationName, setLocationName] = useState(event.locationName ?? "");
const [locationCity, setLocationCity] = useState(event.locationCity ?? "");
const [address, setAddress] = useState(event.address ?? "");
const [templateType] = useState(event.templateType ?? "OTHER");
const [isFree] = useState(event.isFree);
const [coverUrl, setCoverUrl] = useState<string | null>(event.coverImageUrl);
const [uploadingCover, setUploadingCover] = useState(false);
const [ticketList, setTicketList] = useState<TicketTypeUI[]>([]);
const [currentStep, setCurrentStep] = useState(0);
const [fieldErrors, setFieldErrors] = useState<Partial<Record<"title" | "startsAt" | "endsAt" | "locationCity" | "locationName", string>>>({});

const [errorSummary, setErrorSummary] = useState<{ field: string; message: string }[]>([]);
const steps = useMemo(() =>
  isFree
    ? [
        { key: "base", label: "Essenciais", desc: "Imagen e localização" },
        { key: "dates", label: "Datas & Local", desc: "Inicio e fim" },
        { key: "summary", label: "Revisão", desc: "Confirmar e guardar" },
      ]
    : [
        { key: "base", label: "Essenciais", desc: "Imagen e localização" },
        { key: "dates", label: "Datas & Local", desc: "Inicio e fim" },
        { key: "tickets", label: "Bilhetes / Inscrições", desc: "Gestão e vendas" },
      ],
  [isFree],
);
const freeCapacity = useMemo(() => {
  if (!isFree) return null;
  const total = ticketList.reduce((sum, t) => {
    if (t.totalQuantity == null) return sum;
    return sum + t.totalQuantity;
  }, 0);
  return total > 0 ? total : null;
}, [isFree, ticketList]);

const [newTicket, setNewTicket] = useState({
  name: "",
  description: "",
  priceEuro: "",
  totalQuantity: "",
  startsAt: "",
  endsAt: ""
});

const [endingIds, setEndingIds] = useState<number[]>([]);
const [confirmId, setConfirmId] = useState<number | null>(null);
const [confirmText, setConfirmText] = useState("");
const [isSaving, setIsSaving] = useState(false);
const [message, setMessage] = useState<string | null>(null);
const [error, setError] = useState<string | null>(null);
const [stripeAlert, setStripeAlert] = useState<string | null>(null);
const [validationAlert, setValidationAlert] = useState<string | null>(null);
const [backendAlert, setBackendAlert] = useState<string | null>(null);
const ctaRef = useRef<HTMLDivElement | null>(null);
const titleRef = useRef<HTMLInputElement | null>(null);
const startsRef = useRef<HTMLDivElement | null>(null);
const endsRef = useRef<HTMLDivElement | null>(null);
const cityRef = useRef<HTMLInputElement | null>(null);
const locationNameRef = useRef<HTMLInputElement | null>(null);
const errorSummaryRef = useRef<HTMLDivElement | null>(null);
const [toasts, setToasts] = useState<Toast[]>([]);

const pushToast = (message: string, tone: ToastTone = "error") => {
  const id = Date.now() + Math.random();
  setToasts((prev) => [...prev, { id, message, tone }]);
  setTimeout(() => setToasts((prev) => prev.filter((t) => t.id !== id)), 4200);
};

const roles = Array.isArray(profile?.roles) ? (profile?.roles as string[]) : [];
const isAdmin = roles.some((r) => r?.toLowerCase() === "admin");
const payoutMode = (event.payoutMode ?? "ORGANIZER").toUpperCase();
const isPlatformPayout = payoutMode === "PLATFORM";

```

```

const paymentsStatusRaw = isAdmin ? "READY" : organizerStatus?.paymentsStatus ?? "NO_STRIPE";
const paymentsStatus = isPlatformPayout ? "READY" : paymentsStatusRaw;
const hasPaidTicket = useMemo(
  () =>
    ticketList.some((t) => t.price > 0 && t.status === TicketTypeStatus.CANCELLED) ||
    (newTicket.priceEuro && Number(newTicket.priceEuro.replace(",", ".") > 0),
    [ticketList, newTicket.priceEuro],
);
const FormAlert = ({  

  variant,  

  title,  

  message,  

}: {  

  variant: "error" | "warning" | "success";  

  title?: string;  

  message: string;  

}) => {  

  const tones =  

    variant === "error"  

      ? "border-red-500/40 bg-red-500/10 text-red-100"  

    : variant === "warning"  

      ? "border-amber-400/40 bg-amber-400/10 text-amber-100"  

    : "border-emerald-400/40 bg-emerald-500/10 text-emerald-50";  

  return (  

    <div className={`rounded-md border px-4 py-3 text-sm ${tones}`}>  

      {title && <p className="font-semibold">{title}</p>}  

      <p>{message}</p>  

    </div>
  );
};

const focusField = (field: string) => {
  const target =  

    field === "title"  

      ? titleRef.current  

    : field === "startsAt"  

      ? (startsRef.current?.querySelector("button") as HTMLElement | null)  

    : field === "endsAt"  

      ? (endsRef.current?.querySelector("button") as HTMLElement | null)  

    : field === "locationCity"  

      ? cityRef.current  

    : field === "locationName"  

      ? locationNameRef.current  

    : null;
  target?.scrollIntoView({ behavior: "smooth", block: "center" });
  target?.focus({ preventScroll: true });
};

const applyErrors = (issues: { field: string; message: string }[]) => {
  setFieldErrors((prev) => {
    const next = { ...prev };
    issues.forEach((issue) => {
      next[issue.field as keyof typeof next] = issue.message;
    });
    return next;
  });
  setErrorSummary(issues);
  if (issues.length > 0) {
    setTimeout(() => errorSummaryRef.current?.focus({ preventScroll: false }), 40);
  }
};

const clearErrorsForFields = (fields: string[]) => {
  setFieldErrors((prev) => {
    const next = { ...prev };
    fields.forEach((f) => delete next[f as keyof typeof next]);
    return next;
  });
  setErrorSummary((prev) => prev.filter((err) => !fields.includes(err.field)));
};

const collectErrors = (step: number | "all") => {
  const stepsToCheck = step === "all" ? [0, 1] : [step];
  const issues: { field: string; message: string }[] = [];

  stepsToCheck.forEach((idx) => {
    if (idx === 0) {
      if (!title.trim()) issues.push({ field: "title", message: "Título obrigatório." });
    }
  });
};

```

```

        if (!locationName.trim()) issues.push({ field: "locationName", message: "Local obrigatório." });
        if (!locationCity.trim()) issues.push({ field: "locationCity", message: "Cidade obrigatória." });
    }
    if (idx === 1) {
        if (!startsAt) issues.push({ field: "startsAt", message: "Data/hora de início obrigatória." });
        if (endsAt && startsAt && new Date(endsAt).getTime() < new Date(startsAt).getTime()) {
            issues.push({ field: "endsAt", message: "A data/hora de fim tem de ser depois do início." });
        }
    }
});

return issues;
};

const validateStep = (step: number) => {
    const issues = collectErrors(step);
    if (issues.length > 0) {
        applyErrors(issues);
        setValidationAlert("Revê os campos assinalados antes de continuar.");
        setError(issues[0]?.message ?? null);
        return false;
    }
    clearErrorsForFields(step === 0 ? ["title", "locationCity", "locationName"] : ["startsAt", "endsAt"]);
    setValidationAlert(null);
    setError(null);
    return true;
};

useEffect(() => {
    if (title.trim()) clearErrorsForFields(["title"]);
}, [title]);

useEffect(() => {
    if (locationName.trim()) clearErrorsForFields(["locationName"]);
}, [locationName]);

useEffect(() => {
    if (locationCity.trim()) clearErrorsForFields(["locationCity"]);
}, [locationCity]);

useEffect(() => {
    if (startsAt) clearErrorsForFields(["startsAt"]);
}, [startsAt]);

useEffect(() => {
    if (!endsAt) {
        clearErrorsForFields(["endsAt"]);
        return;
    }
    if (startsAt && new Date(endsAt).getTime() >= new Date(startsAt).getTime()) {
        clearErrorsForFields(["endsAt"]);
    }
}, [endsAt, startsAt]);

const goNext = () => {
    const ok = validateStep(currentStep);
    if (!ok) return;
    if (currentStep < steps.length - 1) {
        setValidationAlert(null);
        setError(null);
        setErrorSummary([]);
        setCurrentStep((s) => s + 1);
    } else {
        handleSave();
    }
};

const goPrev = () => {
    setValidationAlert(null);
    setError(null);
    setCurrentStep((s) => Math.max(0, s - 1));
};

const handleCoverUpload = async (file: File | null) => {
    if (!file) return;
    setUploadingCover(true);
    setError(null);
    try {

```

```

const formData = new FormData();
formData.append("file", file);
const res = await fetch("/api/upload", { method: "POST", body: formData });
const json = await res.json();
if (!res.ok || !json?.url) {
  throw new Error(json?.error || "Falha no upload da imagem.");
}
setCoverUrl(json.url as string);
} catch (err) {
  console.error("Erro upload cover", err);
  setError("Não foi possível carregar a imagem de capa.");
} finally {
  setUploadingCover(false);
}
};

const handleSave = async () => {
  setStripeAlert(null);
  setValidationAlert(null);
  setBackendAlert(null);
  setError(null);
  setMessage(null);

  const issues = collectErrors("all");
  if (issues.length > 0) {
    applyErrors(issues);
    setValidationAlert("Revê os campos assinalados antes de guardar o evento.");
    setError(issues[0]?.message ?? null);
    return;
  }
  clearErrorsForFields(["title", "locationCity", "locationName", "startsAt", "endsAt"]);

  if (hasPaidTicket && paymentsStatus !== "READY") {
    setStripeAlert("Podes gerir o evento, mas só vender bilhetes pagos depois de ligares o Stripe.");
    setError("Liga o Stripe em Finanças & Payouts para vender bilhetes pagos.");
    ctaRef.current?.scrollIntoView({ behavior: "smooth", block: "center" });
    return;
  }

  setIsSaving(true);
  try {
    const ticketTypeUpdates = endingIds.map((id) => ({
      id,
      status: TicketTypeStatus.CLOSED,
    }));
    const newTicketsPayload =
      newTicket.name.trim() && newTicket.priceEuro
      ? [
        {
          name: newTicket.name.trim(),
          description: newTicket.description?.trim() || null,
          price: Math.round(Number(newTicket.priceEuro.replace(",", ".") * 100)) || 0,
          totalQuantity: newTicket.totalQuantity
            ? Number(newTicket.totalQuantity)
            : null,
          startsAt: newTicket.startsAt || null,
          endsAt: newTicket.endsAt || null,
        },
      ]
      : [];

    const res = await fetch("/api/organizador/events/update", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        eventId: event.id,
        title,
        description,
        startsAt,
        endsAt,
        locationName,
        locationCity,
        address,
        templateType,
        isFree,
        coverImageUrl: coverUrl,
        feeModeOverride: null,
      })
    });
  }
}

```

```

        platformFeeBpsOverride: null,
        platformFeeFixedCentsOverride: null,
        ticketTypeUpdates,
        newTicketTypes: newTicketsPayload,
    }),
});

const json = await res.json().catch(() => null);
if (!res.ok || !json?.ok) {
    throw new Error(json?.error || "Erro ao atualizar evento.");
}

setMessage("Evento atualizado com sucesso.");
pushToast("Evento atualizado com sucesso.", "success");
setEndingIds([]);
if (ticketTypeUpdates.length > 0) {
    setTicketList((prev) =>
        prev.map((t) =>
            endingIds.includes(t.id) ? { ...t, status: TicketTypeStatus.CLOSED } : t
        )
    );
}
if (newTicketsPayload.length > 0) {
    // Não temos ID do novo ticket aqui, mas podemos forçar refresh manual ou deixar como está.
    // Para feedback imediato, adicionamos placeholder sem ID real.
    setTicketList((prev) => [
        ...prev,
        {
            id: Date.now(), // placeholder local
            name: newTicketsPayload[0].name,
            description: newTicketsPayload[0].description ?? null,
            price: newTicketsPayload[0].price,
            currency: "EUR",
            totalQuantity: newTicketsPayload[0].totalQuantity ?? null,
            soldQuantity: 0,
            status: TicketTypeStatus.ON_SALE,
            startsAt: newTicketsPayload[0].startsAt,
            endsAt: newTicketsPayload[0].endsAt,
        },
    ]);
}
setNewTicket({
    name: "",
    description: "",
    priceEuro: "",
    totalQuantity: "",
    startsAt: "",
    endsAt: "",
});
setErrorSummary([]);
setFieldErrors({});
setMessage("Evento atualizado com sucesso.");
} catch (err) {
    console.error("Erro ao atualizar evento", err);
    setBackendAlert(err instanceof Error ? err.message : "Erro ao atualizar evento.");
    pushToast(err instanceof Error ? err.message : "Erro ao atualizar evento.");
    ctaRef.current?.scrollIntoView({ behavior: "smooth", block: "center" });
} finally {
    setIsSaving(false);
}
};

const openConfirmEnd = (id: number) => {
    setConfirmId(id);
    setConfirmText("");
};

const confirmEnd = async () => {
    if (!confirmId) return;
    if (confirmText.trim().toUpperCase() !== "TERMINAR VENDA") {
        setError('Escreve "TERMINAR VENDA" para confirmar.');
        return;
    }
    setIsSaving(true);
    setError(null);
    try {
        const res = await fetch("/api/organizador/events/update", {
            method: "POST",

```

```

    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      eventId: event.id,
      ticketTypeUpdates: [{ id: confirmId, status: TicketTypeStatus.CLOSED }],
    }),
  );
  const json = await res.json().catch(() => null);
  if (!res.ok || !json?.ok) {
    throw new Error(json?.error || "Erro ao terminar venda.");
  }
  setTicketList((prev) =>
    prev.map((t) => (t.id === confirmId ? { ...t, status: TicketTypeStatus.CLOSED } : t)),
  );
  setMessage("Venda terminada para este bilhete.");
  pushToast("Venda terminada para este bilhete.", "success");
} catch (err) {
  console.error("Erro ao terminar venda", err);
  setError(err instanceof Error ? err.message : "Erro ao terminar venda.");
  pushToast(err instanceof Error ? err.message : "Erro ao terminar venda.");
} finally {
  setIsSaving(false);
  setConfirmId(null);
  setConfirmText("");
}
};

const progress = steps.length > 1 ? Math.min(100, (currentStep / (steps.length - 1)) * 100) : 100;

const renderStepContent = () => {
  const baseBlock = (
    <div className="space-y-4">
      <div className="space-y-2">
        <label className="text-sm font-medium">Imagen de capa</label>
        <div className="flex flex-col sm:flex-row gap-3 items-start">
          <div className="h-32 w-48 rounded-xl border border-white/15 bg-black/30 overflow-hidden flex items-center justify-center text-[11px] text-white/60">
            {coverUrl ? (
              // eslint-disable-next-line @next/next/no-img-element
              <img src={coverUrl} alt="Capa" className="h-full w-full object-cover" />
            ) : (
              <span>Sem imagem</span>
            )}
          </div>
          <div className="space-y-2">
            <div className="flex flex-wrap gap-2 text-[11px] text-white/60">
              <label className="inline-flex cursor-pointer items-center gap-2 rounded-full border border-white/20 px-3 py-1 hover:bg-white/10">
                {span}<span>{coverUrl ? "Substituir imagem" : "Adicionar imagem de capa"}</span>
                <input
                  type="file"
                  accept="image/*"
                  onChange={(e) => handleCoverUpload(e.target.files?.[0] ?? null)}
                  className="hidden"
                />
              </label>
              <button
                type="button"
                disabled={uploadingCover || !coverUrl}
                onClick={() => setCoverUrl(null)}
                className="inline-flex items-center rounded-full border border-white/20 px-3 py-1 hover:bg-white/10 disabled:opacity-60"
              >
                Remover imagem
              </button>
            </div>
            {uploadingCover && <span className="text-[11px] text-white/60">A carregar imagem...</span>}
          </div>
        </div>
      </div>
    </div>
  );
}

<div className="space-y-2">
  <label className="text-sm font-medium">Título *</label>
  <input
    value={title}
    onChange={(e) => setTitle(e.target.value)}
    ref={titleRef}
    aria-invalid={Boolean(fieldErrors.title)}
    className="w-full rounded-md border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none focus:border-

```

```

white/60"
    />
    {fieldErrors.title && (
      <p className="flex items-center gap-2 text-xs font-semibold text-amber-100">
        <span aria-hidden>⚠</span>
        {fieldErrors.title}
      </p>
    )}
</div>
<div className="space-y-2">
  <label className="text-sm font-medium">Descrição</label>
  <textarea
    value={description}
    onChange={(e) => setDescription(e.target.value)}
    rows={4}
    className="w-full rounded-md border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none focus:border-
white/60"
  />
</div>

<div className="grid grid-cols-1 gap-4 sm:grid-cols-2">
  <div className="space-y-1">
    <label className="text-sm font-medium">Local *</label>
    <input
      value={locationName}
      onChange={(e) => setLocationName(e.target.value)}
      ref={locationNameRef}
      aria-invalid={Boolean(fieldErrors.locationName)}
      className="w-full rounded-md border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none focus:border-
white/60"
    />
    {fieldErrors.locationName && (
      <p className="flex items-center gap-2 text-xs font-semibold text-amber-100">
        <span aria-hidden>⚠</span>
        {fieldErrors.locationName}
      </p>
    )}
  </div>
  <div className="space-y-1">
    <label className="text-sm font-medium">Cidade *</label>
    <input
      value={locationCity}
      onChange={(e) => setLocationCity(e.target.value)}
      ref={cityRef}
      aria-invalid={Boolean(fieldErrors.locationCity)}
      className="w-full rounded-md border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none focus:border-
white/60"
    />
    {fieldErrors.locationCity && (
      <p className="flex items-center gap-2 text-xs font-semibold text-amber-100">
        <span aria-hidden>⚠</span>
        {fieldErrors.locationCity}
      </p>
    )}
  </div>
</div>
<div className="space-y-1">
  <label className="text-sm font-medium">Morada</label>
  <input
    value={address}
    onChange={(e) => setAddress(e.target.value)}
    className="w-full rounded-md border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none focus:border-
white/60"
  />
</div>
<div className="space-y-1">
  <label className="text-sm font-medium">Template</label>
  <div className="rounded-md border border-white/15 bg-black/20 px-3 py-2 text-sm text-white/80">
    {templateType === "PARTY"
      ? "Festa"
      : templateType === "SPORT"
      ? "Desporto"
      : templateType === "VOLUNTEERING"
      ? "Voluntariado"
      : templateType === "TALK"
      ? "Palestra / Talk"
      : "Outro"}
  </div>
</div>

```

```

<p className="text-[11px] text-white/55">0 template não pode ser alterado depois de criar o evento.</p>
</div>
<div className="rounded-2xl border border-white/10 bg-white/5 px-3 py-3 text-sm text-white/75">
  <p className="font-semibold text-white">Taxas</p>
  <p className="text-[12px] text-white/65">
    As taxas são definidas pela ORYA. O organizador não altera fee mode nem valores (para orgs de plataforma, taxa ORYA é zero; apenas taxa Stripe aplica).
  </p>
</div>
<div className="rounded-2xl border border-white/10 bg-white/5 px-3 py-3 text-sm text-white/75">
  <p className="font-semibold text-white">Evento grátis</p>
  <p className="text-[12px] text-white/65">
    Só é possível definir se é grátis no momento da criação. Estado atual: {isFree ? "grátis" : "pago"}.
    {isFree && (
      <span className="block text-[12px] text-white/60 mt-1">
        Vagas/inSCRIções: {freeCapacity != null ? freeCapacity : "Sem limite definido"}.
      </span>
    )}
  </p>
</div>
</div>
);

const datesBlock = (
  <div className="space-y-4">
    <div className="grid grid-cols-1 gap-4 sm:grid-cols-2">
      <div ref={startsRef} className="space-y-1">
        <InlineDateTimePicker
          label="Data/hora início"
          value={startsAt}
          onChange={(v) => setStartsAt(v)}
        />
        {fieldErrors.startsAt && (
          <p className="flex items-center gap-2 text-xs font-semibold text-amber-100">
            <span aria-hidden>⚠</span>
            {fieldErrors.startsAt}
          </p>
        )}
      </div>
      <div ref={endsRef} className="space-y-1">
        <InlineDateTimePicker
          label="Data/hora fim"
          value={endsAt}
          onChange={(v) => setEndsAt(v)}
          minDateTime={startsAt ? new Date(startsAt) : undefined}
        />
        {fieldErrors.endsAt && (
          <p className="flex items-center gap-2 text-xs font-semibold text-amber-100">
            <span aria-hidden>⚠</span>
            {fieldErrors.endsAt}
          </p>
        )}
      </div>
    </div>
  </div>
);

const ticketsBlock = (
  <div className="space-y-3">
    <div className="flex items-center justify-between">
      <h2 className="text-sm font-semibold uppercase tracking-wide text-white/70">
        Bilhetes (não removemos, só terminamos venda)
      </h2>
      <Link href={`/organizador?tab=sales&eventId=${event.id}`} className="text-[11px] text-[#6BFFFF]">
        Ver vendas →
      </Link>
    </div>
    <div className="space-y-2">
      {ticketList.map((t) => {
        const price = (t.price / 100).toFixed(2);
        const remaining =
          t.totalQuantity === null && t.totalQuantity !== undefined
            ? t.totalQuantity - t.soldQuantity
            : null;
        const isEnding = endingIds.includes(t.id) || t.status === TicketTypeStatus.CLOSED;

        return (

```

```

<div
  key={t.id}
  className="rounded-xl border border-white/12 bg-black/30 p-3 flex flex-col gap-2"
>
  <div className="flex items-center justify-between gap-2">
    <div className="flex flex-col">
      <p className="font-semibold text-sm">{t.name}</p>
      <p className="text-[11px] text-white/60">
        {price} € • Vendidos: {t.soldQuantity}
        {remaining === null ? `• Stock restante: ${remaining}` : ""}
      </p>
    </div>
    <span className="text-[10px] rounded-full border border-white/20 px-2 py-0.5 text-white/75">
      {isEnding ? "Venda terminada" : t.status}
    </span>
  </div>
  <div className="flex flex-wrap gap-2 text-[11px]">
    <button
      type="button"
      onClick={() => openConfirmEnd(t.id)}
      disabled={t.status === TicketTypeStatus.CLOSED}
      className={`rounded-full px-3 py-1 border ${
        t.status === TicketTypeStatus.CLOSED
          ? "border-white/15 text-white/40 cursor-not-allowed"
          : "border-amber-300/60 text-amber-100 hover:bg-amber-500/10"
      }`}
    >
      Terminar venda
    </button>
  </div>
</div>
);
})}
</div>

<div className="rounded-xl border border-white/12 bg-black/25 p-3 space-y-2">
  <p className="text-[12px] font-semibold">Adicionar novo bilhete</p>
  <div className="grid grid-cols-1 md:grid-cols-2 gap-2">
    <input
      placeholder="Nome"
      value={newTicket.name}
      onChange={(e) => setNewTicket((p) => ({ ...p, name: e.target.value }))}
      className="rounded-md border border-white/15 bg-black/30 px-3 py-2 text-sm"
    />
    <input
      placeholder="Preço (euros)"
      value={newTicket.priceEuro}
      onChange={(e) => setNewTicket((p) => ({ ...p, priceEuro: e.target.value }))}
      className="rounded-md border border-white/15 bg-black/30 px-3 py-2 text-sm"
    />
    <input
      placeholder="Quantidade total"
      value={newTicket.totalQuantity}
      onChange={(e) => setNewTicket((p) => ({ ...p, totalQuantity: e.target.value }))}
      className="rounded-md border border-white/15 bg-black/30 px-3 py-2 text-sm"
    />
    <input
      placeholder="Descrição (opcional)"
      value={newTicket.description}
      onChange={(e) => setNewTicket((p) => ({ ...p, description: e.target.value }))}
      className="rounded-md border border-white/15 bg-black/30 px-3 py-2 text-sm"
    />
  </div>
  <div className="text-[11px] text-white/70">
    Início vendas
    <input
      type="datetime-local"
      value={newTicket.startsAt}
      onChange={(e) => setNewTicket((p) => ({ ...p, startsAt: e.target.value }))}
      className="mt-1 w-full rounded-md border border-white/15 bg-black/30 px-3 py-2 text-sm"
    />
  </div>
  <div className="text-[11px] text-white/70">
    Fim vendas
    <input
      type="datetime-local"
      value={newTicket.endsAt}
      onChange={(e) => setNewTicket((p) => ({ ...p, endsAt: e.target.value }))}
      className="mt-1 w-full rounded-md border border-white/15 bg-black/30 px-3 py-2 text-sm"
    />
  </div>

```

```

        />
      </div>
    </div>
    <p className="text-[11px] text-white/50">
      Novo bilhete fica ON_SALE por padrão. Não removemos bilhetes antigos para manter histórico.
    </p>
    </div>
  </div>
);

const summaryBlock = (
  <div className="space-y-4">
    <div className="rounded-2xl border border-white/10 bg-white/5 p-4">
      <p className="text-sm font-semibold text-white">Resumo rápido</p>
      <p className="text-white/70 text-sm mt-1">Confirma os detalhes antes de guardar.</p>
      <div className="mt-3 grid grid-cols-1 md:grid-cols-2 gap-3 text-sm text-white/80">
        <div className="rounded-xl border border-white/10 bg-black/20 p-3">
          <p className="text-[11px] uppercase tracking-wide text-white/60">Evento</p>
          <p className="font-semibold">{title || "Sem título"}</p>
          <p className="text-white/60 text-sm line-clamp-2">{description || "Sem descrição"}</p>
        </div>
        <div className="rounded-xl border border-white/10 bg-black/20 p-3 space-y-1">
          <p className="text-[11px] uppercase tracking-wide text-white/60">Local e datas</p>
          <p>{locationName || "Local a definir"}</p>
          <p className="text-white/70">{locationCity || "Cidade a definir"}</p>
          <p className="text-white/70">
            {startsAt ? new Date(startsAt).toLocaleString() : "Início por definir"}{" "}
            {endsAt ? `→ ${new Date(endsAt).toLocaleString()}` : ""}
          </p>
        </div>
      <div className="rounded-xl border border-white/10 bg-black/20 p-3 space-y-1">
        <p className="text-[11px] uppercase tracking-wide text-white/60">Estado</p>
        <p className="font-semibold">{isFree ? "Evento grátis" : "Evento pago"}</p>
        {isFree && (
          <p className="text-white/70">
            Vagas/inscrições: {freeCapacity != null ? freeCapacity : "Sem limite definido"}.
          </p>
        )}
      </div>
    </div>
  </div>
);

switch (steps[currentStep].key) {
  case "base":
    return baseBlock;
  case "dates":
    return datesBlock;
  case "tickets":
    return ticketsBlock;
  case "summary":
    return summaryBlock;
  default:
    return null;
}
};

return (
<>
  <div className="space-y-6">
    {confirmId && (
      <div className="fixed inset-0 z-40 flex items-center justify-center bg-black/70 backdrop-blur">
        <div className="w-full max-w-sm rounded-2xl border border-white/15 bg-black/90 p-5 shadow-[0_20px_60px_rgba(0,0,0,.85)] space-y-3">
          <h3 className="text-lg font-semibold">Terminar venda do bilhete?</h3>
          <p className="text-sm text-white/70">
            Esta ação é definitiva para este tipo de bilhete. Escreve{" "}
            <span className="font-semibold">TERMINAR VENDA</span> para confirmar.
          </p>
          <input
            value={confirmText}
            onChange={(e) => setConfirmText(e.target.value)}
            className="w-full rounded-md border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-white/50"
            placeholder="TERMINAR VENDA"
          />
        <div className="flex justify-end gap-2 text-[12px]">
    
```

```

        <button
          type="button"
          onClick={() => {
            setConfirmId(null);
            setConfirmText("");
          }}
          className="rounded-full border border-white/20 px-3 py-1 text-white/75 hover:bg-white/10"
        >
          Cancelar
        </button>
        <button
          type="button"
          onClick={confirmEnd}
          className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-3 py-1 font-semibold
text-black shadow"
        >
          Confirmar
        </button>
      </div>
    </div>
  </div>
}

<div className="rounded-2xl border border-white/10 bg-white/5 p-4 space-y-5">
  <div className="flex flex-col gap-2 sm:flex-row sm:items-center sm:justify-between">
    <div>
      <p className="text-[11px] uppercase tracking-wide text-white/60">Edição em passos</p>
      <p className="text-lg font-semibold text-white">Editar evento</p>
      <p className="text-sm text-white/60">
        Define o teu evento passo a passo. Podes guardar como rascunho em qualquer momento.
      </p>
    </div>
    <div className="text-right text-[12px] text-white/60">
      <p>Estado: {isFree ? "Grátis" : "Pago"}</p>
      <p>Template: {templateType}</p>
    </div>
  </div>
  <div>
    {errorSummary.length > 0 && (
      <div
        ref={errorSummaryRef}
        tabIndex={-1}
        className="rounded-xl border border-amber-400/40 bg-amber-500/10 p-3 text-sm text-amber-50 focus:outline-none
focus:ring-2 focus:ring-amber-200/70"
      >
        <div className="flex items-center gap-2 font-semibold">
          <span aria-hidden>⚠</span>
          <span>Revê estes campos antes de continuar</span>
        </div>
        <ul className="mt-2 space-y-1 text-[13px]">
          {errorSummary.map((err) => (
            <li key={`${err.field}-${err.message}`}>
              <button
                type="button"
                onClick={() => focusField(err.field)}
                className="inline-flex items-center gap-2 text-left font-semibold text-white underline decoration-amber-
200 underline-offset-4 hover:text-amber-50"
              >
                <span aria-hidden>✖</span>
                <span>{err.message}</span>
              </button>
            </li>
          )));
        </ul>
      </div>
    )}>
  </div>
  <div className="space-y-3">
    <div className="relative h-1 rounded-full bg-white/10">
      <div
        className="absolute left-0 top-0 h-1 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5]"
        style={{ width: `${progress}%` }}
      />
    </div>
    <div className="grid grid-cols-1 gap-3 sm:grid-cols-3 md:grid-cols-5">
      {steps.map((step, idx) => {
        const state = idx === currentStep ? "active" : idx < currentStep ? "done" : "future";
        const allowClick = idx < currentStep;
        <div
          key={idx}
          style={{ background: state === "active" ? "#1646F5" : "#E0E0E0", color: state === "active" ? "white" : "black", padding: "5px 10px", border: "1px solid #E0E0E0", border-radius: "10px", cursor: state === "active" ? "pointer" : "normal" }
        >
          {step}
        </div>
      )}>
    </div>
  </div>

```

```

return (
  <button
    key={step.key}
    type="button"
    onClick={() => allowClick && setCurrentStep(idx)}
    className={`flex flex-col items-start rounded-xl border px-3 py-3 text-left transition ${
      state === "active"
        ? "border-white/40 bg-white/10 shadow"
        : state === "done"
        ? "border-white/15 bg-white/5 text-white/80"
        : "border-white/10 bg-black/10 text-white/60"
    } ${!allowClick ? "cursor-default" : "hover:border-white/30 hover:bg-white/5"}`}
    disabled={!allowClick}
  >
  <div
    className={`mb-2 flex h-9 w-9 items-center justify-center rounded-full border ${
      state === "active"
        ? "border-white bg-white text-black shadow-[0_0_6px_rgba(255,255,255,0.08)]"
        : state === "done"
        ? "border-emerald-300/70 bg-emerald-400/20 text-emerald-100"
        : "border-white/30 text-white/70"
    }`}
  >
    {state === "done" ? "✓" : idx + 1}
  </div>
  <p className="text-sm font-semibold text-white">{step.label}</p>
  <p className="text-[12px] text-white/60">{step.desc}</p>
</button>
);
})
)}
</div>
</div>

<div className="rounded-2xl border border-white/10 bg-black/20 p-4">
  {renderStepContent()}
</div>

<div ref={ctaRef} className="space-y-3">
  {stripeAlert && (
    <FormAlert
      variant={hasPaidTicket ? "error" : "warning"}
      title="Stripe incompleto"
      message={stripeAlert}
    />
  )}
  {validationAlert && <FormAlert variant="warning" message={validationAlert} />}
  {error && <FormAlert variant="error" message={error} />}
  {backendAlert && (
    <FormAlert
      variant="error"
      title="Algo correu mal ao guardar o evento"
      message={backendAlert}
    />
  )}
  {message && <FormAlert variant="success" message={message} />}
</div>

<div className="flex flex-col gap-3 sm:flex-row sm:items-center sm:justify-between">
  <div className="flex gap-2 text-sm">
    <button
      type="button"
      onClick={goPrev}
      disabled={currentStep === 0 || isSaving}
      className="rounded-full border border-white/20 px-4 py-2 text-white/80 hover:bg-white/10 disabled:opacity-50"
    >
      Anterior
    </button>
    <Link
      href={`/organizador/eventos/${event.id}`}
      className="rounded-full border border-white/20 px-4 py-2 text-white/80 hover:bg-white/10"
    >
      Voltar
    </Link>
  </div>
  <button
    type="button"
    onClick={goNext}
    disabled={isSaving}
    className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-5 py-2 text-sm font-

```

```

semibold text-black shadow disabled:opacity-60"
      >
        {currentStep === steps.length - 1 ? (isSaving ? "A gravar..." : "Guardar alterações") : "Continuar"}
      </button>
    </div>
  </div>
</div>
{toasts.length > 0 && (
  <div className="pointer-events-none fixed bottom-6 right-6 z-40 flex flex-col gap-2">
    {toasts.map((toast) => (
      <div
        key={toast.id}
        className={`${pointerEventsAuto minW-[240px] rounded-lg border px-4 py-3 text-sm shadow-lg ${toast.tone === "success" ? "border-emerald-400/50 bg-emerald-500/15 text-emerald-50" : "border-red-400/50 bg-red-500/15 text-red-50"}`}
      >
        {toast.message}
      </div>
    )));
  </div>
)}
</>
);
}
}

```

app/organizador/(dashboard)/eventos/novo/page.tsx

```

"use client";

import { useCallback, useEffect, useMemo, useRef, useState } from "react";
import { useRouter, useSearchParams } from "next/navigation";
import Link from "next/link";
import useSWR from "swr";
import { InlineDateTimePicker } from "@app/components/forms/InlineDateTimePicker";
import { FlowStickyFooter } from "@app/components/flows/FlowStickyFooter";
import { useUser } from "@app/hooks/useUser";
import { useAuthModal } from "@app/components/autenticação/AuthModalContext";
import { StepperDots, type WizardStep } from "@components/organizador/eventos/wizard/StepperDots";
import { PT_CITIES, type PTCity } from "@lib/constants/ptCities";

type TicketTypeRow = {
  name: string;
  price: string;
  totalQuantity: string;
};

type ToastTone = "success" | "error";
type Toast = { id: number; message: string; tone: ToastTone };

const DRAFT_KEY = "orya-organizer-new-event-draft";

const CATEGORY_OPTIONS = [
  {
    key: "padel",
    value: "SPORT",
    label: "Padel / Torneio",
    accent: "from-[#6BFFFF] to-[#22c55e]",
    copy: "Setup rápido com courts, rankings e lógica de torneio.",
    categories: ["DESPORTO"],
  },
  {
    key: "default",
    value: "OTHER",
    label: "Evento padrão",
    accent: "from-[#9ca3af] to-[#6b7280]",
    copy: "Fluxo base sem extras – serve para qualquer formato simples.",
    categories: ["FESTA"],
  },
] as const;

const DEFAULT_PLATFORM_FEE_BPS = 800; // 8%
const DEFAULT_PLATFORM_FEE_FIXED_CENTS = 30; // €0.30
const DEFAULT_STRIPE_FEE_BPS = 140; // 1.4%

```

```

const DEFAULT_STRIPE_FEE_FIXED_CENTS = 25; // €0.25

type PlatformFeeResponse =
| {
  ok: true;
  orya: { feeBps: number; feeFixedCents: number };
  stripe: { feeBps: number; feeFixedCents: number; region: string };
}
| { ok: false; error?: string };

const fetcher = (url: string) => fetch(url).then((res) => res.json());

type StepKey = "preset" | "details" | "schedule" | "tickets" | "review";
type FieldKey =
| "preset"
| "title"
| "description"
| "startsAt"
| "endsAt"
| "locationName"
| "locationCity"
| "address"
| "tickets";

type RecentVenuesResponse = {
  ok: boolean;
  items?: Array<{ name: string; city?: string | null }>;
};

function computeFeePreview(
  priceEuro: number,
  mode: "ON_TOP" | "INCLUDED",
  platformFees: { feeBps: number; feeFixedCents: number },
  stripeFees: { feeBps: number; feeFixedCents: number },
) {
  const baseCents = Math.round(Math.max(0, priceEuro) * 100);
  const feeCents = Math.round((baseCents * platformFees.feeBps) / 10_000) + platformFees.feeFixedCents;

  if (mode === "ON_TOP") {
    const totalCliente = baseCents + feeCents;
    const stripeOnTotal = Math.round((totalCliente * stripeFees.feeBps) / 10_000) + stripeFees.feeFixedCents;
    const recebeOrganizador = Math.max(0, baseCents - stripeOnTotal);
    return { baseCents, feeCents, totalCliente, recebeOrganizador, stripeFeeCents: stripeOnTotal };
  }

  const totalCliente = baseCents;
  const stripeOnBase = Math.round((totalCliente * stripeFees.feeBps) / 10_000) + stripeFees.feeFixedCents;
  const recebeOrganizador = Math.max(0, baseCents - feeCents - stripeOnBase);
  return { baseCents, feeCents, totalCliente, recebeOrganizador, stripeFeeCents: stripeOnBase };
}

export default function NewOrganizerEventPage() {
  const router = useRouter();
  const searchParams = useSearchParams();
  const { user, profile, isLoading: isUserLoading } = useUser();
  const { openModal } = useAuthModal();
  const { data: platformFeeData } = useSWR<PlatformFeeResponse>("/api/platform/fees", fetcher, {
    revalidateOnFocus: false,
  });
  const { data: organizerStatus } = useSWR<{ paymentsStatus?: string; profileStatus?: string }>(
    user ? "/api/organizador/me" : null,
    fetcher,
    { revalidateOnFocus: false }
  );
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [startsAt, setStartsAt] = useState("");
  const [endsAt, setEndsAt] = useState("");
  const [locationName, setLocationName] = useState("");
  const [locationCity, setLocationCity] = useState<PTCity>(PT_CITIES[0]);
  const [address, setAddress] = useState("");
  const [ticketTypes, setTicketTypes] = useState<TicketTypeRow[]>([{ name: "Geral", price: "", totalQuantity: "" }]);
  const [feeMode, setFeeMode] = useState<"ON_TOP" | "INCLUDED">("ON_TOP");
  const [coverUrl, setCoverUrl] = useState<string | null>(null);
  const [uploadingCover, setUploadingCover] = useState(false);
  const [isTest, setIsTest] = useState(false);
  const [selectedPreset, setSelectedPreset] = useState<string | null>(null);
  const [isFreeEvent, setIsFreeEvent] = useState(false);
}

```

```

const [freeTicketName, setFreeTicketName] = useState("Inscrição");
const [freeCapacity, setFreeCapacity] = useState("");
const [currentStep, setCurrentStep] = useState(0);
const [maxStepReached, setMaxStepReached] = useState(0);

const [isSubmitting, setIsSubmitting] = useState(false);
const [showLoadingHint, setShowLoadingHint] = useState(false);
const [errorMessage, setErrorMessage] = useState<string | null>(null);
const [stripeAlert, setStripeAlert] = useState<string | null>(null);
const [validationAlert, setValidationAlert] = useState<string | null>(null);
const [backendAlert, setBackendAlert] = useState<string | null>(null);
const [draftLoaded, setDraftLoaded] = useState(false);
const [draftSavedAt, setDraftSavedAt] = useState<number | null>(null);
const [toasts, setToasts] = useState<Toast[]>([]);
const [fieldErrors, setFieldErrors] = useState<Partial<Record<FieldKey, string>>>({});

const [errorSummary, setErrorSummary] = useState<{ field: FieldKey; message: string }[]>([]);
const [showLocationSuggestions, setShowLocationSuggestions] = useState(false);
const [pendingFocusField, setPendingFocusField] = useState<FieldKey | null>(null);
const [creationSuccess, setCreationSuccess] = useState<{ eventId?: number; slug?: string } | null>(null);
const prevStepIndexRef = useRef(0);

const ctaAlertRef = useRef<HTMLDivElement | null>(null);
const errorSummaryRef = useRef<HTMLDivElement | null>(null);
const titleRef = useRef<HTMLInputElement | null>(null);
const startsRef = useRef<HTMLDivElement | null>(null);
const endsRef = useRef<HTMLDivElement | null>(null);
const locationNameRef = useRef<HTMLInputElement | null>(null);
const cityRef = useRef<HTMLInputElement | null>(null);
const ticketsRef = useRef<HTMLDivElement | null>(null);
const suggestionBlurTimeout = useRef<NodeJS.Timeout | null>(null);

const roles = Array.isArray(profile?.roles) ? (profile?.roles as string[]) : [];
const isOrganizer = roles.includes("organizer");
const isAdmin = roles.some((r) => r.toLowerCase() === "admin");
const paymentsStatus = isAdmin ? "READY" : organizerStatus?.paymentsStatus ?? "NO_STRIPE";
const platformFees =
  platformFeeData && platformFeeData.ok
    ? platformFeeData.orya
    : { feeBps: DEFAULT_PLATFORM_FEE_BPS, feeFixedCents: DEFAULT_PLATFORM_FEE_FIXED_CENTS };
const stripeFees =
  platformFeeData && platformFeeData.ok
    ? platformFeeData.stripe
    : { feeBps: DEFAULT_STRIPE_FEE_BPS, feeFixedCents: DEFAULT_STRIPE_FEE_FIXED_CENTS, region: "UE" };
const hasPaidTicket = useMemo(
  () => !isFreeEvent && ticketTypes.some((t) => Number(t.price.replace(",",".")) > 0),
  [isFreeEvent, ticketTypes],
);

const presetMap = useMemo(() => {
  const map = new Map<string, (typeof CATEGORY_OPTIONS)[number]>();
  CATEGORY_OPTIONS.forEach((opt) => map.set(opt.key, opt));
  return map;
}, []);

const { data: recentVenues } = useSWR<RecentVenuesResponse>(
  user ? `/api/organizador/venues/recent?${encodeURIComponent(locationName.trim())}` : null,
  fetcher,
  { revalidateOnFocus: false },
);

useEffect(() => {
  if (draftLoaded) return;
  if (typeof window === "undefined") return;
  const raw = window.localStorage.getItem(DRAFT_KEY);
  if (!raw) {
    setDraftLoaded(true);
    return;
  }
  try {
    const draft = JSON.parse(raw) as Partial<{
      title: string;
      description: string;
      startsAt: string;
      endsAt: string;
      locationName: string;
      locationCity: string;
      address: string;
      ticketTypes: TicketTypeRow[];
    }>;
  
```

```

    feeMode: "ON_TOP" | "INCLUDED";
    coverUrl: string | null;
    selectedPreset: string | null;
    isFreeEvent: boolean;
    freeTicketName: string;
    freeCapacity: string;
    currentStep: number;
    maxStepReached: number;
    savedAt: number;
  }>;
  setTitle(draft.title ?? "");
  setDescription(draft.description ?? "");
  setStartsAt(draft.startsAt ?? "");
  setEndsAt(draft.endsAt ?? "");
  setLocationName(draft.locationName ?? "");
  setLocationCity(
    draft.locationCity && PT_CITIES.includes(draft.locationCity as PTCity)
    ? (draft.locationCity as PTCity)
    : PT_CITIES[0],
  );
  setAddress(draft.address ?? "");
  setTicketTypes(
    Array.isArray(draft.ticketTypes) && draft.ticketTypes.length > 0
    ? draft.ticketTypes
    : [{ name: "Geral", price: "", totalQuantity: "" }],
  );
  setFeeMode(draft.feeMode ?? "ON_TOP");
  setCoverUrl(draft.coverUrl ?? null);
  setSelectedPreset(draft.selectedPreset ?? null);
  setIsFreeEvent(Boolean(draft.isFreeEvent));
  setFreeTicketName(draft.freeTicketName || "Inscrição");
  setFreeCapacity(draft.freeCapacity || "");
  const draftCurrentStep =
    typeof draft.currentStep === "number" && Number.isFinite(draft.currentStep) ? draft.currentStep : 0;
  const draftMaxStep =
    typeof draft.maxStepReached === "number" && Number.isFinite(draft.maxStepReached)
    ? draft.maxStepReached
    : draftCurrentStep;
  setCurrentStep(Math.min(draftCurrentStep, 4));
  setMaxStepReached(Math.min(draftMaxStep, 4));
  setDraftSavedAt(draft.savedAt ?? null);
} catch (err) {
  console.warn("Falha ao carregar rascunho local", err);
} finally {
  setDraftLoaded(true);
}
}, [draftLoaded]);

useEffect(() => {
  if (!draftLoaded) return;
  const typeParam = searchParams?.get("type");
  const keyParam = searchParams?.get("category") ?? searchParams?.get("preset");
  if (selectedPreset || (!typeParam && !keyParam)) return;
  const match = CATEGORY_OPTIONS.find((opt) => opt.value === typeParam || opt.key === keyParam);
  if (match) {
    setSelectedPreset(match.key);
  }
}, [draftLoaded, searchParams, selectedPreset]);

useEffect(() => {
  if (!isFreeEvent) return;
  setTicketTypes([
    {
      name: freeTicketName.trim() || "Inscrição",
      price: "0",
      totalQuantity: freeCapacity,
    },
  ]);
}, [isFreeEvent, freeTicketName, freeCapacity]);

useEffect(() => {
  clearErrorsForFields(["tickets"]);
  setStripeAlert(null);
}, [isFreeEvent]);

const stepOrder = useMemo<{ key: StepKey; title: string; subtitle: string }[]>(
() => [
  { key: "preset", title: "Formato", subtitle: "Escolhe o tipo de evento" },

```

```

{ key: "details", title: "Essenciais", subtitle: "Imagen, título e descrição" },
{ key: "schedule", title: "Datas", subtitle: "Início, fim e local" },
{
  key: "tickets",
  title: "Bilhetes",
  subtitle: isFreeEvent ? "Capacidade e vagas" : "Preços e stock",
},
{ key: "review", title: "Rever", subtitle: "Confirma & cria" },
],
[isFreeEvent],
);

const stepIndexMap = useMemo(() => {
  const map = new Map<StepKey, number>();
  stepOrder.forEach((step, idx) => map.set(step.key, idx));
  return map;
}, [stepOrder]);
const wizardSteps: WizardStep[] = useMemo(
() => [
  { id: "formato", title: "Formato" },
  { id: "essenciais", title: "Essenciais" },
  { id: "datas_local", title: "Datas" },
  { id: "bilhetes", title: "Bilhetes" },
  { id: "revisao", title: "Rever" },
],
[], []
);
const stepIdByKey: Record<StepKey, WizardStep["id"]> = {
  preset: "formato",
  details: "essenciais",
  schedule: "datas_local",
  tickets: "bilhetes",
  review: "revisao",
};
const baseInputClasses =
  "w-full rounded-md border border-white/12 bg-black/25 px-4 py-3 text-sm text-white/90 placeholder:text-white/45 outline-none transition focus:border-[var(--orya-cyan)] focus:ring-2 focus:ring-[rgba(107,255,255,0.35)] focus:ring-offset-0 focus:ring-offset-transparent";
const errorInputClasses =
  "border-[rgba(255,0,200,0.45)] focus:border-[rgba(255,0,200,0.6)] focus:ring-[rgba(255,0,200,0.4)]";
const inputClass = (errored?: boolean) => `${baseInputClasses} ${errored ? errorInputClasses : ""}`;
const labelClass =
  "text-[11px] font-semibold uppercase tracking-[0.18em] text-white/65 flex items-center gap-1";
const errorTextClass = "flex items-center gap-2 text-[12px] font-semibold text-pink-200";

const pushToast = (message: string, tone: ToastTone = "success") => {
  const id = Date.now() + Math.random();
  setToasts((prev) => [...prev, { id, message, tone }]);
  setTimeout(() => setToasts((prev) => prev.filter((t) => t.id !== id)), 3800);
};

const saveDraft = () => {
  if (typeof window === "undefined") return;
  const payload = {
    title,
    description,
    startsAt,
    endsAt,
    locationName,
    locationCity,
    address,
    ticketTypes,
    feeMode,
    coverUrl,
    selectedPreset,
    isFreeEvent,
    freeTicketName,
    freeCapacity,
    currentStep,
    maxStepReached,
    savedAt: Date.now(),
  };
  window.localStorage.setItem(DRAFT_KEY, JSON.stringify(payload));
  setDraftSavedAt(Date.now());
  pushToast("Rascunho guardado.", "success");
};

const handleRequireLogin = () => {

```

```

openModal({
  mode: "login",
  redirectTo: "/organizador/(dashboard)/eventos/novo",
});
};

const handleSelectPreset = (key: string) => {
  const preset = presetMap.get(key);
  if (!preset) return;
  setSelectedPreset(preset.key);
  setValidationAlert(null);
  setErrorMessage(null);
  clearErrorsForFields(["preset"]);
};

const handleSelectLocationSuggestion = (suggestion: { name: string; city?: string | null }) => {
  setLocationName(suggestion.name);
  if (suggestion.city && PT_CITIES.includes(suggestion.city as PTCity)) {
    setLocationCity(suggestion.city as PTCity);
  }
  clearErrorsForFields(["locationName", "locationCity"]);
  setShowLocationSuggestions(false);
};

const handleAddTicketType = () => {
  clearErrorsForFields(["tickets"]);
  setStripeAlert(null);
  setTicketTypes((prev) => [...prev, { name: "", price: "", totalQuantity: "" }]);
};

const handleRemoveTicketType = (index: number) => {
  clearErrorsForFields(["tickets"]);
  setStripeAlert(null);
  setTicketTypes((prev) => prev.filter(_ , i) => i !== index));
};

const handleTicketChange = (index: number, field: keyof TicketTypeRow, value: string) => {
  setTicketTypes((prev) => prev.map((row, i) => (i === index ? { ...row, [field]: value } : row)));
  clearErrorsForFields(["tickets"]);
  setStripeAlert(null);
};

const handleCoverUpload = async (file: File | null) => {
  if (!file) return;
  const formData = new FormData();
  formData.append("file", file);
  setUploadingCover(true);
  setErrorMessage(null);
  try {
    const res = await fetch("/api/upload", {
      method: "POST",
      body: formData,
    });
    const json = await res.json();
    if (!res.ok || !json?.url) {
      throw new Error(json?.error || "Falha no upload da imagem.");
    }
    setCoverUrl(json.url as string);
  } catch (err) {
    console.error("Erro no upload de capa", err);
    setErrorMessage("Não foi possível carregar a imagem de capa.");
  } finally {
    setUploadingCover(false);
  }
};

const buildTicketsPayload = () => {
  if (isFreeEvent) {
    const totalQuantityRaw = freeCapacity ? Number(freeCapacity) : null;
    const parsedQuantity =
      typeof totalQuantityRaw === "number" && Number.isFinite(totalQuantityRaw) && totalQuantityRaw > 0
        ? totalQuantityRaw
        : null;
    return [
      {
        name: freeTicketName.trim() || "Inscrição",
        price: 0,
        totalQuantity: parsedQuantity,
      }
    ];
  }
};

```

```

        },
    ];
}

return ticketTypes
.map((row) => ({
    name: row.name.trim(),
    price: Number(row.price.replace(",",".")) || 0,
    totalQuantity: row.totalQuantity ? Number(row.totalQuantity) : null,
}))
.filter((t) => t.name);
};

const fieldsByStep: Record<StepKey, FieldKey[]> = {
    preset: ["preset"],
    details: ["title", "description"],
    schedule: ["startsAt", "endsAt", "locationName", "locationCity", "address"],
    tickets: ["tickets"],
    review: [],
};

function collectStepErrors(stepKey: StepKey | "all") {
    const keys = stepKey === "all" ? (["preset", "details", "schedule", "tickets"] as StepKey[]) : [stepKey];
    const issues: { field: FieldKey; message: string }[] = [];
    keys.forEach((key) => {
        if (key === "preset" && !selectedPreset) {
            issues.push({ field: "preset", message: "Escolhe um formato." });
        }
        if (key === "details") {
            if (!title.trim()) {
                issues.push({ field: "title", message: "Título obrigatório." });
            }
        }
        if (key === "schedule") {
            if (!startsAt) issues.push({ field: "startsAt", message: "Data/hora de início obrigatória." });
            if (!endsAt) issues.push({ field: "endsAt", message: "Data/hora de fim obrigatória." });
            if (!locationName.trim()) issues.push({ field: "locationName", message: "Local obrigatório." });
            if (!locationCity.trim()) issues.push({ field: "locationCity", message: "Cidade obrigatória." });
            if (endsAt && startsAt && new Date(endsAt).getTime() <= new Date(startsAt).getTime()) {
                issues.push({ field: "endsAt", message: "A data/hora de fim tem de ser depois do inicio." });
            }
        }
        if (key === "tickets") {
            const preparedTickets = buildTicketsPayload();
            if (preparedTickets.length === 0) {
                issues.push({ field: "tickets", message: "Adiciona pelo menos um bilhete ou inscrição." });
            }
            if (!isFreeEvent && preparedTickets.some((t) => t.price < 0)) {
                issues.push({ field: "tickets", message: "Preço tem de ser positivo." });
            }
            if (!isFreeEvent && hasPaidTicket && paymentsStatus !== "READY") {
                issues.push({ field: "tickets", message: "Liga o Stripe em Finanças & Payouts para vender bilhetes pagos." });
            }
        }
    });
    return issues;
}

const fieldStepMap = useMemo<Record<FieldKey, StepKey>>(
() => ({
    preset: "preset",
    title: "details",
    description: "details",
    startsAt: "schedule",
    endsAt: "schedule",
    locationName: "schedule",
    locationCity: "schedule",
    address: "schedule",
    tickets: "tickets",
}),
[],
[]
);

function clearErrorsForFields(fields: FieldKey[]) {
    setFieldErrors((prev) => {
        const next = { ...prev };
        fields.forEach((field) => {
            delete next[field];
        });
        return next;
    });
}

```

```

    });
    return next;
});
setErrorSummary((prev) => prev.filter((err) => !fields.includes(err.field)));
}

function applyErrors(errors: { field: FieldKey; message: string }[], focusSummary = true) {
  if (errors.length === 0) {
    setErrorSummary([]);
  } else {
    setErrorSummary(errors);
  }
  setFieldErrors((prev) => {
    const next = { ...prev };
    errors.forEach((err) => {
      next[err.field] = err.message;
    });
    return next;
  });
  if (errors.length > 0 && focusSummary) {
    setTimeout(() => {
      errorSummaryRef.current?.focus({ preventScroll: false });
    }, 40);
  }
}

const focusField = useCallback(
  (field: FieldKey) => {
    const targetStep = fieldStepMap[field];
    const targetStepIndex = stepIndexMap.get(targetStep);
    if (typeof targetStepIndex === "number" && targetStepIndex !== currentStep) {
      setCurrentStep(targetStepIndex);
      setMaxStepReached((prev) => Math.max(prev, targetStepIndex));
      setPendingFocusField(field);
      return;
    }

    const focusable =
      field === "title"
        ? titleRef.current
        : field === "startsAt"
        ? (startsRef.current?.querySelector("button") as HTMLElement | null)
        : field === "endsAt"
        ? (endsRef.current?.querySelector("button") as HTMLElement | null)
        : field === "locationName"
        ? locationNameRef.current
        : field === "locationCity"
        ? cityRef.current
        : field === "tickets"
        ? (ticketsRef.current?.querySelector("input,button,select,textarea") as HTMLElement | null)
        : null;

    if (focusable) {
      focusable.scrollIntoView({ behavior: "smooth", block: "center" });
      focusable.focus({ preventScroll: true });
    }
  },
  [currentStep, fieldStepMap, stepIndexMap],
);

useEffect(() => {
  if (!pendingFocusField) return;
  const expectedStep = fieldStepMap[pendingFocusField];
  const targetStepIndex = stepIndexMap.get(expectedStep);
  if (typeof targetStepIndex === "number" && targetStepIndex !== currentStep) return;
  focusField(pendingFocusField);
  setPendingFocusField(null);
}, [pendingFocusField, currentStep, stepIndexMap, fieldStepMap, focusField]);

useEffect(() => {
  let timer: NodeJS.Timeout | null = null;
  if (isSubmitting) {
    timer = setTimeout(() => setShowLoadingHint(true), 750);
  } else {
    setShowLoadingHint(false);
  }
  return () => {
    if (timer) clearTimeout(timer);
  };
}

```

```

};

}, [isSubmitting];

const activeStepKey = stepOrder[currentStep]?.key ?? "preset";
const nextDisabledReason = () => {
  const issues =
    activeStepKey === "review" ? collectStepErrors("all") : collectStepErrors(activeStepKey as StepKey);
  if (isSubmitting) return "A criar evento..";
  return issues[0]?.message ?? null;
}();

const currentWizardStepId = stepIdByKey[activeStepKey];
const direction = currentStep >= prevStepIndexRef.current ? "right" : "left";
useEffect(() => {
  prevStepIndexRef.current = currentStep;
}, [currentStep]);

const filteredLocationSuggestions = useMemo(() => {
  const items = recentVenues?.ok && Array.isArray(recentVenues.items) ? recentVenues.items : [];
  if (items.length === 0) return [];
  const term = `${locationName} ${locationCity}`.toLowerCase().trim();
  const filtered = items.filter((s) => `${s.name} ${s.city ?? ""}`.toLowerCase().includes(term));
  return (term ? filtered : items).slice(0, 8);
}, [recentVenues, locationName, locationCity]);

useEffect(() => {
  if (title.trim()) clearErrorsForFields(["title"]);
}, [title]);

useEffect(() => {
  if (startsAt) clearErrorsForFields(["startsAt"]);
}, [startsAt]);

useEffect(() => {
  if (locationName.trim()) clearErrorsForFields(["locationName"]);
}, [locationName]);

useEffect(() => {
  if (locationCity.trim()) clearErrorsForFields(["locationCity"]);
}, [locationCity]);

useEffect(() => {
  if (endsAt && startsAt && new Date(endsAt).getTime() > new Date(startsAt).getTime()) {
    clearErrorsForFields(["endsAt"]);
  }
}, [endsAt, startsAt]);

const FormAlert = ({ variant,
  title: alertTitle,
  message,
  actionLabel,
  onAction,
}: {
  variant: "error" | "warning" | "success";
  title?: string;
  message: string;
  actionLabel?: string;
  onAction?: () => void;
}) => {
  const tones =
    variant === "error"
      ? "border-red-500/40 bg-red-500/10 text-red-100"
      : variant === "warning"
        ? "border-amber-400/40 bg-amber-400/10 text-amber-100"
        : "border-emerald-400/40 bg-emerald-500/10 text-emerald-50";
  return (
    <div className={`${`rounded-md border px-4 py-3 text-sm ${tones}`}>
      <div className="flex flex-wrap items-center justify-between gap-2">
        <div className="space-y-1">
          {alertTitle && <p className="font-semibold">{alertTitle}</p>}
          <p>{message}</p>
        </div>
        {actionLabel && onAction && (
          <button
            type="button"
            onClick={onAction}
            className="rounded-full border border-white/30 px-3 py-1 text-[11px] font-semibold hover:bg-white/10"
        )}
      </div>
    </div>
  )
}

```

```

        >
          {actionLabel}
        </button>
      )}
</div>
</div>
);
};

const goNext = () => {
  const activeKey = stepOrder[currentStep].key;
  if (!activeKey) return;
  const issues = activeKey === "review" ? collectStepErrors("all") : collectStepErrors(activeKey as StepKey);
  if (issues.length > 0) {
    applyErrors(issues);
    setValidationAlert("Revê os campos em falta antes de continuar.");
    setErrorMessage(issues[0]?.message ?? null);
    setStripeAlert(issues.find((err) => err.message.includes("Stripe")) ? "Liga o Stripe em Finanças & Payouts para vender bilhetes pagos." : null);
    return;
  }
  clearErrorsForFields(fieldsByStep[activeKey as StepKey]);
  setValidationAlert(null);
  setErrorMessage(null);
  setStripeAlert(null);
  setErrorSummary([]);
  if (currentStep >= stepOrder.length - 1) {
    handleSubmit();
    return;
  }
  setValidationAlert(null);
  setErrorMessage(null);
  setCurrentStep((s) => s + 1);
  setMaxStepReached((prev) => Math.max(prev, currentStep + 1));
};

const goPrev = () => {
  setValidationAlert(null);
  setErrorMessage(null);
  setErrorSummary([]);
  setStripeAlert(null);
  setCurrentStep((s) => Math.max(0, s - 1));
};

const handleSubmit = async () => {
  setStripeAlert(null);
  setValidationAlert(null);
  setBackendAlert(null);
  setErrorMessage(null);

  const issues = collectStepErrors("all");
  if (issues.length > 0) {
    applyErrors(issues);
    setValidationAlert("Revê os campos em falta antes de criar o evento.");
    setErrorMessage(issues[0]?.message ?? null);
    setStripeAlert(issues.find((err) => err.message.includes("Stripe")) ? "Liga o Stripe em Finanças & Payouts para vender bilhetes pagos." : null);
    return;
  }

  const preparedTickets = buildTicketsPayload();
  const scrollTo = (el?: HTMLElement | null) => el?.scrollIntoView({ behavior: "smooth", block: "center" });

  if (!user) {
    handleRequireLogin();
    return;
  }

  if (!isOrganizer) {
    setErrorMessage("Ainda não és organizador. Vai à área de organizador para ativares essa função.");
    return;
  }

  setIsSubmitting(true);

  try {
    const preset = selectedPreset ? presetMap.get(selectedPreset) : null;
    const categoriesToSend = preset?.categories ?? ["FESTA"];
  }

```

```

const templateToSend = preset?.value ?? "OTHER";
const payload = {
  title: title.trim(),
  description: description.trim() || null,
  startsAt,
  endsAt,
  locationName: locationName.trim() || null,
  locationCity: locationCity.trim() || null,
  templateType: templateToSend,
  address: address.trim() || null,
  categories: categoriesToSend,
  ticketTypes: preparedTickets,
  coverImageUrl: coverUrl,
  feeMode,
  isTest: isAdmin ? isTest : undefined,
};

const res = await fetch("/api/organizador/events/create", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify(payload),
});

const data = await res.json();

if (!res.ok || !data?.ok) {
  throw new Error(data?.error || "Erro ao criar evento.");
}

const event = data.event;
if (event?.id || event?.slug) {
  if (typeof window !== "undefined") {
    window.localStorage.removeItem(DRAFT_KEY);
  }
  setCreationSuccess({ eventId: event.id, slug: event.slug });
  setCurrentStep(stepOrder.length - 1);
  setMaxStepReached(stepOrder.length - 1);
  setErrorSummary([]);
  setFieldErrors({});
}
} catch (err) {
  console.error("Erro ao criar evento de organizador:", err);
  const message = err instanceof Error ? err.message : null;
  setBackendAlert(message || "Algo correu mal ao guardar o evento. Tenta novamente em segundos.");
  scrollTo(ctaAlertRef.current);
} finally {
  setIsSubmitting(false);
}
};

const resetForm = () => {
  setSelectedPreset(null);
  setTitle("");
  setDescription("");
  setStartsAt("");
  setEndsAt("");
  setLocationName("");
  setLocationCity(PT_CITIES[0]);
  setAddress("");
  setTicketTypes([{ name: "Geral", price: "", totalQuantity: "" }]);
  setIsFreeEvent(false);
  setFreeTicketName("Inscrição");
  setFreeCapacity("");
  setCoverUrl(null);
  setCreationSuccess(null);
  setCurrentStep(0);
  setMaxStepReached(0);
  setValidationAlert(null);
  setErrorMessage(null);
  setErrorSummary([]);
  setFieldErrors({});
  setStripeAlert(null);
  setBackendAlert(null);
  if (typeof window !== "undefined") {
    window.localStorage.removeItem(DRAFT_KEY);
  }
}

```

```

};

if (isUserLoading) {
  return (
    <div className="max-w-3xl mx-auto px-4 py-8">
      <p>A carregar a tua conta...</p>
    </div>
  );
}

if (!user) {
  return (
    <div className="max-w-3xl mx-auto px-4 py-8 space-y-4">
      <h1 className="text-2xl font-semibold">Criar novo evento</h1>
      <p>Precisas de iniciar sessão para criar eventos como organizador.</p>
      <button
        type="button"
        onClick={handleRequireLogin}
        className="inline-flex items-center rounded-md border border-white/10 bg-white/5 px-2 py-2 text-sm font-medium
        hover:bg-white/10"
      >
        Entrar
      </button>
    </div>
  );
}

if (!isOrganizer) {
  return (
    <div className="max-w-3xl mx-auto px-4 py-8 space-y-4">
      <h1 className="text-2xl font-semibold">Criar novo evento</h1>
      <p>ainda não é organizador. Vai à área de organizador para ativar essa função.</p>
      <Link
        href="/organizador"
        className="inline-flex items-center rounded-md border border-white/10 bg-white/5 px-2 py-2 text-sm font-medium
        hover:bg-white/10"
      >
        Ir para área de organizador
      </Link>
    </div>
  );
}

const renderPresetStep = () => (
  <div className="space-y-4 animate-fade-slide">
    <div className="flex flex-col gap-2">
      <p className="text-sm text-white/75">Escolhe o formato. Padel ativa o wizard dedicado; Evento padrão é neutro.</p>
      <p className="text-[12px] text-white/55">Todo segue a mesma linguagem visual.</p>
      {fieldErrors.preset && (
        <p className={errorTextClass}>
          <span aria-hidden>⚠</span>
          {fieldErrors.preset}
        </p>
      )}
    </div>
    <div className="grid gap-3 sm:grid-cols-2">
      {CATEGORY_OPTIONS.map((opt) => {
        const isActive = selectedPreset === opt.key;
        return (
          <button
            key={opt.key}
            type="button"
            onClick={() => handleSelectPreset(opt.key)}
            className={`group flex flex-col items-start gap-2 rounded-2xl border border-white/12 bg-black/40 p-4 text-left
            transition hover:border-white/30 hover:shadow-[0_10px_30px_rgba(0,0,0,0.4)] ${{
              isActive ? "ring-2 ring-[#6BFFFF]/40 border-white/30" : ""
            }}`}
          >
            <span
              className={`${` inline-flex items-center rounded-full bg-gradient-to-r ${opt.accent} px-3 py-1 text-[11px] font-
              semibold text-black shadow`}`}
            >
              {opt.label}
            </span>
            <p className="text-sm text-white/80">{opt.copy}</p>
          </div>
          {opt.categories.length === 0 ? (
            <span className="rounded-full border border-white/10 px-2 py-0.5">Personalizado</span>

```

```

        ) : (
          opt.categories.map((cat) => (
            <span key={cat} className="rounded-full border border-white/15 px-2 py-0.5">
              {cat}
            </span>
          ))
        )
      </div>
    </button>
  );
)
);
</div>
</div>
);

const renderDetailsStep = () => (
  <div className="space-y-4 animate-fade-slide">
    {isAdmin && (
      <label className="flex items-center gap-3 rounded-2xl border border-white/12 bg-black/30 px-3 py-2 text-sm">
        <input
          type="checkbox"
          checked={isTest}
          onChange={(e) => setIsTest(e.target.checked)}
          className="h-4 w-4 rounded border-white/40 bg-transparent"
        />
        <span className="text-white/80">Evento de teste (visível só para admin, não aparece em explorar)</span>
      </label>
    )}
    <div className="rounded-2xl border border-white/12 bg-[rgba(14,14,20,0.7)] p-4 space-y-3 shadow-[0_14px_36px_rgba(0,0,0,0.45)]">
      <div className="flex flex-col gap-1 sm:flex-row sm:items-start sm:justify-between">
        <div>
          <p className={labelClass}>Imagen de capa</p>
          <p className="text-[12px] text-white/65">Hero do evento – legível em mobile.</p>
        </div>
        <div className="flex items-center gap-2 text-[11px] text-white/60">
          {uploadingCover && <span className="animate-pulse text-white/70">A carregar...</span>}
          {coverUrl && (
            <button
              type="button"
              onClick={() => setCoverUrl(null)}
              className="rounded-full border border-white/20 px-3 py-1 text-white/75 hover:bg-white/10"
            >
              Remover
            </button>
          )}
        </div>
      </div>
      <div className="flex flex-col gap-3 sm:flex-row sm:items-center">
        <div className="h-36 w-56 rounded-xl border border-white/15 bg-gradient-to-br from-[#12121f] via-[#0b0b18] to-[#1f1f30] overflow-hidden flex items-center justify-center text-[11px] text-white/60">
          {coverUrl ? (
            // eslint-disable-next-line @next/next/no-img-element
            <img src={coverUrl} alt="Capa" className="h-full w-full object-cover" />
          ) : (
            <span className="text-white/55">Sem imagem</span>
          )}
        </div>
        <div className="flex flex-wrap gap-2 text-[12px] text-white/60">
          <label className="inline-flex cursor-pointer items-center gap-2 rounded-full border border-white/25 px-3 py-1 hover:bg-white/10">
            {coverUrl ? "Trocar imagem" : "Adicionar imagem"}<span>
            <input
              type="file"
              accept="image/*"
              onChange={(e) => handleCoverUpload(e.target.files?.[0] ?? null)}
              className="hidden"
            />
          </label>
          <span className="inline-flex items-center rounded-full border border-white/10 px-3 py-1 text-white/50">
            1200x630 recomendado
          </span>
        </div>
      </div>
    </div>
  </div>

  <div className="grid grid-cols-1 gap-4 md:grid-cols-2">

```

```

<div className="space-y-2 md:col-span-2">
  <label className={labelClass}>
    Título <span aria-hidden>*</span>
  </label>
  <input
    type="text"
    value={title}
    onChange={(e) => setTitle(e.target.value)}
    ref={titleRef}
    aria-invalid={Boolean(fieldErrors.title)}
    className={inputClass(Boolean(fieldErrors.title))}
    placeholder="Torneio Sunset Padel"
  />
  {fieldErrors.title && (
    <p className={errorTextClass}>
      <span aria-hidden>⚠</span>
      {fieldErrors.title}
    </p>
  )}
</div>

<div className="space-y-2 md:col-span-2">
  <label className={labelClass}>Descrição</label>
  <textarea
    value={description}
    onChange={(e) => setDescription(e.target.value)}
    rows={4}
    className={inputClass(false)}
    placeholder="Explica rapidamente o que torna o evento único."
  />
</div>
</div>
);

const renderScheduleStep = () => (
  <div className="space-y-4 animate-fade-slide">
    <div className="grid grid-cols-1 gap-4 sm:grid-cols-2">
      <div ref={startsRef} className="space-y-1">
        <InlineDateTimePicker
          label="Data/hora inicio *"
          value={startsAt}
          onChange={(v) => setStartsAt(v)}
          minDateTime={new Date()}
          required
        />
        {fieldErrors.startsAt && (
          <p className={errorTextClass}>
            <span aria-hidden>⚠</span>
            {fieldErrors.startsAt}
          </p>
        )}
      </div>
      <div ref={endsRef} className="space-y-1">
        <InlineDateTimePicker
          label="Data/hora fim *"
          value={endsAt}
          onChange={(v) => setEndsAt(v)}
          minDateTime={startsAt ? new Date(startsAt) : new Date()}
        />
        {fieldErrors.endsAt && (
          <p className={errorTextClass}>
            <span aria-hidden>⚠</span>
            {fieldErrors.endsAt}
          </p>
        )}
      </div>
    </div>
    <div className="grid grid-cols-1 gap-4 sm:grid-cols-2">
      <div className="space-y-1">
        <label className={labelClass}>
          Local <span aria-hidden>*</span>
        </label>
        <div className="relative overflow-visible">
          <input
            type="text"
            value={locationName}

```

```

    onChange={(e) => {
      setLocationName(e.target.value);
      setShowLocationSuggestions(true);
    }}
    onFocus={() => setShowLocationSuggestions(true)}
    onBlur={() => {
      if (suggestionBlurTimeout.current) clearTimeout(suggestionBlurTimeout.current);
      suggestionBlurTimeout.current = setTimeout(() => setShowLocationSuggestions(false), 120);
    }}
    ref={locationNameRef}
    aria-invalid={Boolean(fieldErrors.locationName)}
    className={inputClass(Boolean(fieldErrors.locationName))}
    placeholder="Clube, sala ou venue"
  />
{showLocationSuggestions && (
  <div className="absolute left-0 right-0 z-[70] mt-2 max-h-56 overflow-y-auto rounded-xl border border-white/12 bg-black/90 shadow-xl backdrop-blur-2xl animate-popover">
    {recentVenues === undefined ? (
      <div className="px-3 py-2 text-sm text-white/70 animate-pulse">A procurar...</div>
    ) : filteredLocationSuggestions.length === 0 ? (
      <div className="px-3 py-2 text-sm text-white/60">Sem locais recentes.</div>
    ) : (
      filteredLocationSuggestions.map((suggestion) => (
        <button
          key={`${suggestion.name}-${suggestion.city} ?? "?"`}
          type="button"
          onMouseDown={(e) => e.preventDefault()}
          onClick={() => handleSelectLocationSuggestion(suggestion)}
          className="flex w-full flex-col items-start gap-1 border-b border-white/5 px-3 py-2 text-left text-sm
hover:bg-white/8 last:border-0 transition"
        >
          <div className="flex w-full items-center justify-between gap-3">
            <span className="font-semibold text-white">{suggestion.name}</span>
            <span className="text-[12px] text-white/65">{suggestion.city || "Cidade por definir"}</span>
          </div>
          <span className="text-[11px] text-white/50">Usado em eventos deste organizador</span>
        </button>
      )));
    )
  )}
</div>
{fieldErrors.locationName && (
  <p className={errorTextClass}>
    <span aria-hidden>⚠</span>
    {fieldErrors.locationName}
  </p>
)
}
</div>

<div className="space-y-1">
  <label className={labelClass}>
    Cidade <span aria-hidden>*</span>
  </label>
  <select
    value={locationCity}
    onChange={(e) => {
      setLocationCity(e.target.value);
      setShowLocationSuggestions(true);
    }}
    ref={cityRef}
    aria-invalid={Boolean(fieldErrors.locationCity)}
    onFocus={() => setShowLocationSuggestions(true)}
    onBlur={() => {
      if (suggestionBlurTimeout.current) clearTimeout(suggestionBlurTimeout.current);
      suggestionBlurTimeout.current = setTimeout(() => setShowLocationSuggestions(false), 120);
    }}
    className={inputClass(Boolean(fieldErrors.locationCity))}>
    {PT_CITIES.map((city) => (
      <option key={city} value={city}>
        {city}
      </option>
    )))
  </select>
{fieldErrors.locationCity && (
  <p className={errorTextClass}>
    <span aria-hidden>⚠</span>
  </p>
)
}
</div>

```

```

        {fieldErrors.locationCity}
      </p>
    )}
</div>
</div>

<div className="space-y-1">
  <label className={labelClass}>Rua / morada (opcional)</label>
  <input
    type="text"
    value={address}
    onChange={(e) => setAddress(e.target.value)}
    className={inputClass(false)}
    placeholder="Rua, número ou complemento"
  />
</div>
</div>
);

const renderTicketsStep = () => (
  <div ref={ticketsRef} className="space-y-5 animate-fade-slide">
    <div className="flex flex-col gap-3 rounded-2xl border border-white/12 bg-[rgba(14,14,20,0.7)] p-4 shadow-[0_14px_36px_rgba(0,0,0,0.45)]">
      <div className="flex items-center justify-between">
        <div>
          <p className={labelClass}>Modelo</p>
          <p className="text-[12px] text-white/65">Escolhe se é pago ou gratuito. Copy adapta-se.</p>
        </div>
        <div className="inline-flex rounded-full border border-white/15 bg-black/40 p-1 text-[13px]">
          <button
            type="button"
            onClick={() => setIsFreeEvent(false)}
            className={` rounded-full px-3 py-1 font-semibold transition ${isFreeEvent ? "bg-white text-black shadow" : "text-white/70"}`}
          >
            Evento pago
          </button>
          <button
            type="button"
            onClick={() => setIsFreeEvent(true)}
            className={` rounded-full px-3 py-1 font-semibold transition ${isFreeEvent ? "bg-white text-black shadow" : "text-white/70"}`}
          >
            Evento grátis
          </button>
        </div>
      </div>
      <p className="text-[12px] text-white/55">
        Bilhetes pagos só ficam ativos com Stripe ligado. Eventos grátis focam-se em inscrições e vagas.
      </p>
      {fieldErrors.tickets && (
        <p className={errorTextClass}>
          <span aria-hidden>⚠</span>
          {fieldErrors.tickets}
        </p>
      )}
    </div>
    {isFreeEvent ? (
      <div className="space-y-3 rounded-2xl border border-white/12 bg-[rgba(12,12,20,0.65)] p-4">
        <div className="flex items-center justify-between">
          <p className={labelClass}>Inscrições gratuitas</p>
          <span className="rounded-full border border-emerald-300/40 bg-emerald-400/10 px-3 py-1 text-[12px] text-emerald-50">
            Sem taxas
          </span>
        </div>
        <div className="grid gap-3 md:grid-cols-2">
          <div className="space-y-1">
            <label className={labelClass}>Nome da inscrição</label>
            <input
              type="text"
              value={freeTicketName}
              onChange={(e) => setFreeTicketName(e.target.value)}
              className={inputClass(false)}
              placeholder="Inscrição geral, equipa..." />

```

```

        </div>
    <div className="space-y-1">
        <label className={labelClass}>Capacidade (opcional)</label>
        <input
            type="number"
            min={0}
            value={freeCapacity}
            onChange={(e) => setFreeCapacity(e.target.value)}
            className={inputClass(false)}
            placeholder="Ex.: 64"
        />
    </div>
</div>
<p className="text-[12px] text-white/60">
    Só precisas disto para registar vagas. Podes abrir inscrições avançadas (equipas, rankings) no passo Padel.
</p>
</div>
: (
<div className="space-y-4 rounded-2xl border border-white/12 bg-[rgba(12,12,20,0.65)] p-4">
    <div className="flex flex-wrap items-center justify-between gap-2">
        <h2 className={labelClass}>Bilhetes</h2>
        <button
            type="button"
            onClick={handleAddTicketType}
            className="inline-flex items-center rounded-full border border-white/20 bg-black/25 px-3 py-1 text-[13px] font-semibold hover:border-white/35 hover:bg-white/5 transition"
        >
            + Adicionar bilhete
        </button>
    </div>
</div>

<div className="grid gap-3">
    {ticketTypes.map((row, idx) => {
        const priceEuro = Number(row.price || "0");
        const preview = computeFeePreview(priceEuro, feeMode, platformFees, stripeFees);
        const combinedFeeCents = preview.feeCents + preview.stripeFeeCents;
        return (
            <div
                key={idx}
                className="space-y-3 rounded-xl border border-white/12 bg-white/[0.03] p-3 shadow-[0_12px_30px_rgba(0,0,0,0.35)] animate-step-pop"
            >
                <div className="flex flex-col gap-2 sm:flex-row sm:items-center sm:justify-between">
                    <div className="space-y-1 flex-1">
                        <label className={labelClass}>
                            Nome do bilhete <span aria-hidden>*</span>
                        </label>
                        <input
                            type="text"
                            value={row.name}
                            onChange={(e) => handleTicketChange(idx, "name", e.target.value)}
                            className={inputClass(false)}
                            placeholder="Early bird, Geral, VIP"
                        />
                    </div>
                    {ticketTypes.length > 1 && (
                        <button
                            type="button"
                            onClick={() => handleRemoveTicketType(idx)}
                            className="text-[11px] text-white/60 hover:text-white/90"
                        >
                            Remover
                        </button>
                    )}
                </div>
                <div className="grid grid-cols-1 gap-3 sm:grid-cols-3">
                    <div className="space-y-1">
                        <label className={labelClass}>
                            Preço (€) <span aria-hidden>*</span>
                        </label>
                        <input
                            type="number"
                            min={0}
                            step=".01"
                            value={row.price}
                            onChange={(e) => handleTicketChange(idx, "price", e.target.value)}
                            className={inputClass(false)}
                        />
                    </div>
                </div>
            </div>
        );
    })
</div>

```

```

        placeholder="Ex.: 12.50"
      />
    </div>
    <div className="space-y-1">
      <label className={labelClass}>Capacidade (opcional)</label>
      <input
        type="number"
        min={0}
        value={row.totalQuantity}
        onChange={(e) => handleTicketChange(idx, "totalQuantity", e.target.value)}
        className={inputClass(false)}
        placeholder="Ex.: 100"
      />
    </div>
    <div className="space-y-1">
      <p className="text-[12px] font-semibold text-white/75">Pré-visualização</p>
      <div className="text-[12px] rounded-lg border border-white/10 bg-black/25 px-3 py-2 text-white/85">
        <p>Cliente: {(preview.totalCliente / 100).toFixed(2)} €</p>
        <p>Recebes: {(preview.recebeOrganizador / 100).toFixed(2)} €</p>
        <p className="text-white/50">Taxa ORYA: {(combinedFeeCents / 100).toFixed(2)} €</p>
      </div>
    </div>
  </div>
);
});
</div>

<div className="space-y-2">
  <p className={labelClass}>Modo de taxas</p>
  <div className="inline-flex rounded-full border border-white/15 bg-black/40 p-1 text-[13px]">
    <button
      type="button"
      onClick={() => setFeeMode("ON_TOP")}
      className={`rounded-full px-3 py-1 font-semibold transition ${feeMode === "ON_TOP" ? "bg-white text-black shadow" : "text-white/70"}`}
    >
      Cliente paga taxa
    </button>
    <button
      type="button"
      onClick={() => setFeeMode("INCLUDED")}
      className={`rounded-full px-3 py-1 font-semibold transition ${feeMode === "INCLUDED" ? "bg-white text-black shadow" : "text-white/70"}`}
    >
      Preço inclui taxas
    </button>
  </div>
  <p className="text-[12px] text-white/55">
    Podes ajustar depois no resumo. Para eventos de plataforma, a taxa ORYA é zero.
  </p>
</div>
</div>
);
}

const renderReviewStep = () => {
  const previewTickets = buildTicketsPayload();
  const presetLabel = selectedPreset === "padel" ? "Padel / Torneio" : "Evento padrão";
  const presetDesc = selectedPreset === "padel" ? "Wizard Padel ativo" : "Fluxo base sem extras";
  const pendingIssues = collectStepErrors("all");
  const pendingLabel = pendingIssues.length === 0 ? "Campos ok" : `Falta corrigir ${pendingIssues.length}`;
  return (
    <div className="space-y-4 animate-fade-slide">
      <div className="rounded-2xl border border-white/12 bg-[rgba(12,12,20,0.72)] p-4 space-y-4 shadow-[0_14px_36px_rgba(0,0,0,0.45)]">
        <div className="flex items-start justify-between gap-3">
          <div className="space-y-1">
            <p className={labelClass}>Revisão final</p>
            <p className="text-white/70 text-sm">Todo pronto. Revê os detalhes antes de publicar.</p>
          </div>
          <div className="flex flex-col items-end gap-1 text-right">
            <span className="text-[11px] uppercase tracking-[0.18em] text-white/55">Passo 5/5</span>
            <span className="btn-chip bg-white/10 text-white/90">{pendingLabel}</span>
          </div>
        </div>
      </div>
    </div>
  );
}

```

```

    </div>
    <div className="grid grid-cols-1 gap-3 md:grid-cols-2">
      <div className="rounded-xl border border-white/12 bg-black/25 p-3 shadow-inner transition hover:border-white/25
      hover:bg-white/10">
        <div className="flex items-start justify-between gap-3">
          <div>
            <p className={labelClass}>Essenciais</p>
            <p className="font-semibold text-white">{title || "Sem título"}</p>
          </div>
          <button
            type="button"
            onClick={() => focusField("title")}
            className="btn-chip">
            >
            Editar
          </button>
        </div>
        <div className="mt-2 flex items-center gap-3">
          <div className="h-16 w-24 overflow-hidden rounded-lg border border-white/10 bg-gradient-to-br from-[#161623]
          via-[#0c0c18] to-[#241836] text-[11px] text-white/60">
            {coverUrl ? (
              // eslint-disable-next-line @next/next/no-img-element
              <img src={coverUrl} alt="Capa" className="h-full w-full object-cover" />
            ) : (
              <div className="flex h-full w-full items-center justify-center text-[11px] text-white/60">
                Sem imagem
              </div>
            )}
          </div>
          <p className="text-sm text-white/70 line-clamp-3">{description || "Sem descrição"}</p>
        </div>
      </div>
    <div className="rounded-xl border border-white/12 bg-black/25 p-3 shadow-inner transition hover:border-white/25
    hover:bg-white/10">
      <div className="flex items-start justify-between gap-3">
        <div>
          <p className={labelClass}>Datas</p>
          <p className="font-semibold text-white">
            {locationName || "Local a definir"} · {locationCity || "Cidade a definir"}
          </p>
        </div>
        <button
          type="button"
          onClick={() => focusField("startsAt")}
          className="btn-chip">
          >
          Editar
        </button>
      </div>
      <p className="text-sm text-white/70">
        {startsAt ? new Date(startsAt).toLocaleString() : "Início por definir"}{" "}
        {endsAt ? `~ ${new Date(endsAt).toLocaleString()}` : ""}
      </p>
      {address && <p className="text-[12px] text-white/60">{address}</p>}
    </div>
  <div className="rounded-xl border border-white/12 bg-black/25 p-3 shadow-inner transition hover:border-white/25
  hover:bg-white/10">
    <div className="flex items-start justify-between gap-3">
      <div>
        <p className={labelClass}>Bilhetes</p>
        <p className="font-semibold text-white">
          {isFreeEvent
            ? `Vagas: ${freeCapacity ? freeCapacity : "sem limite"}`
            : `${previewTickets.length} ${previewTickets.length === 1 ? "" : "s"} de bilhete`}
        </p>
      </div>
      <button
        type="button"
        onClick={() => focusField("tickets")}
        className="btn-chip">
        >
        Editar
      </button>
    </div>
    {!isFreeEvent && <ul className="mt-2 space-y-1 text-sm text-white/70">

```

```

        {previewTickets.map((t) => (
          <li key={`${t.name}-${t.price}`} className="flex items-center justify-between gap-2">
            <span>{t.name}</span>
            <span className="text-white/60">{t.price.toFixed(2)} €</span>
          </li>
        )));
      </ul>
    )}
    {isFreeEvent && <p className="text-sm text-white/70">Entrada gratuita com inscrições simples.</p>}
</div>

<div className="rounded-xl border border-white/12 bg-black/25 p-3 shadow-inner transition hover:border-white/25
hover:bg-white/10">
  <div className="flex items-start justify-between gap-3">
    <div>
      <p className={labelClass}>Modelo</p>
      <p className="font-semibold text-white">{isFreeEvent ? "Evento grátis" : "Evento pago"}</p>
    </div>
    <button
      type="button"
      onClick={() => focusField("preset")}
      className="btn-chip">
      Editar
    </button>
  </div>
  <p className="text-sm text-white/70">Formato: {presetLabel}</p>
  <p className="text-[12px] text-white/60">{presetDesc}</p>
</div>
</div>
</div>
</div>
);
};

return (
<form
  noValidate
  onSubmit={(e) => {
    e.preventDefault();
    goNext();
  }}
  className="max-w-5xl mx-auto px-4 py-8 space-y-6 md:px-6 lg:px-8 text-white"
>
  <div className="flex flex-col gap-2 sm:flex-row sm:items-center sm:justify-between">
    <div className="space-y-1">
      <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Novo evento</p>
      <h1 className="text-2xl font-semibold tracking-tight">Cria o teu evento</h1>
      <p className="text-sm text-white/70">Fluxo rápido com autosave, feedback premium e zero ruído.</p>
    </div>
    <div className="flex flex-wrap gap-2 text-[11px]">
      <Link
        href="/organizador"
        className="btn-ghost text-[12px] font-semibold">
        Voltar
      </Link>
      <button
        type="button"
        onClick={saveDraft}
        className="btn-ghost text-[12px] font-semibold">
        Guardar rascunho
      </button>
      {draftSavedAt && (
        <span className="rounded-full border border-white/10 px-1 text-white/70 bg-white/5">
          Guardado há pouco
        </span>
      )}
    </div>
  </div>

  <div className="relative rounded-3xl border border-white/8 bg-[rgba(9,10,16,0.78)] p-5 md:p-6 space-y-6 shadow-[0_24px_70px_rgba(0,0,0,0.6)] overflow-visible">
    <div className="pb-1">
      <StepperDots
        steps={wizardSteps}
        current={currentWizardStepId}>

```

```

maxUnlockedIndex={Math.max(maxStepReached, currentStep)}
onGoTo={(id) => {
  const idx = wizardSteps.findIndex((s) => s.id === id);
  const maxClickable = Math.max(maxStepReached, currentStep);
  if (idx >= 0 && idx <= maxClickable) setCurrentStep(idx);
}}
/>
</div>

{errorSummary.length > 0 && (
<div
  ref={errorSummaryRef}
  tabIndex={-1}
  className="rounded-xl border border-amber-400/40 bg-amber-500/10 p-3 text-sm text-amber-50 focus:outline-none
focus:ring-2 focus:ring-amber-200/70"
  aria-live="assertive"
>
  <div className="flex items-center gap-2 font-semibold">
    <span aria-hidden>⚠</span>
    <span>Revê estes campos antes de continuar</span>
  </div>
  <ul className="mt-2 space-y-1 text-[13px]">
    {errorSummary.map((err) => (
      <li key={`${err.field}-${err.message}`}>
        <button
          type="button"
          onClick={() => focusField(err.field)}
          className="inline-flex items-center gap-2 text-left font-semibold text-white underline decoration-pink-200
underline-offset-4 hover:text-pink-50"
        >
          <span aria-hidden>✖</span>
          <span>{err.message}</span>
        </button>
      </li>
    )))
  </ul>
</div>
)}
}

<div className="rounded-2xl border border-white/12 bg-[rgba(12,12,20,0.75)] p-4 md:p-5 min-h-[420px] md:min-h-[460px]">
<section
  key={activeStepKey}
  className={direction === "right" ? "wizard-step-in-right" : "wizard-step-in-left"}
>
  {activeStepKey === "preset" && renderPresetStep()}
  {activeStepKey === "details" && renderDetailsStep()}
  {activeStepKey === "schedule" && renderScheduleStep()}
  {activeStepKey === "tickets" && renderTicketsStep()}
  {activeStepKey === "review" && renderReviewStep()}
</section>
</div>

<div ref={ctaAlertRef} className="space-y-3">
{stripeAlert && (
  <FormAlert
    variant={hasPaidTicket ? "error" : "warning"}
    title="Stripe incompleto"
    message={stripeAlert}
    actionLabel="Abrir Finanças & Payouts"
    onAction={() => router.push("/organizador?tab=finance")}
  />
)
}
{validationAlert && <FormAlert variant="warning" message={validationAlert} />}
{errorMessage && <FormAlert variant="error" message={errorMessage} />}
{backendAlert && (
  <FormAlert
    variant="error"
    title="Algo correu mal ao guardar o evento"
    message={backendAlert}
  />
)
}
</div>

<FlowStickyFooter
  backLabel="Anterior"
  nextLabel={currentStep === stepOrder.length - 1 ? "Criar evento" : "Continuar"}
  helper={
    activeStepKey === "tickets" && isFreeEvent
  }
/>

```

```

        ? "Inscrições sem taxas; capacidade é opcional."
        : activeStepKey === "review"
        ? "Confirma blocos, edita no passo certo e cria com confiança."
        : "Navega sem perder contexto; feedback sempre visível."
    }
    disabledReason={nextDisabledReason}
    loading={isSubmitting}
    loadingLabel={currentStep === stepOrder.length - 1 ? "A criar..." : "A processar..."}
    showLoadingHint={showLoadingHint}
    disableBack={currentStep === 0}
    onBack={goPrev}
    onNext={goNext}
/>
</div>

{creationSuccess &&
<div className="fixed bottom-6 left-6 z-40 w-[320px] max-w-full rounded-2xl border border-emerald-400/50 bg-emerald-500/15 p-4 shadow-[0_18px_45px_rgba(0,0,0,0.55)] text-emerald-50">
    <div className="flex items-start justify-between gap-2">
        <div>
            <p className="text-sm font-semibold">Evento criado</p>
            <p className="text-[13px] text-emerald-50/85">Escolhe o próximo passo ou crie outro.</p>
        </div>
        <button
            type="button"
            onClick={() => setCreationSuccess(null)}
            className="text-[12px] text-emerald-50/80 hover:text-white"
            aria-label="Fechar alerta de criação"
        >
            ×
        </button>
    </div>
    <div className="mt-3 flex flex-wrap gap-2 text-[12px]">
        {creationSuccess.slug &&
            <Link
                href={`/eventos/${creationSuccess.slug}`}
                className="rounded-full border border-emerald-200/60 bg-emerald-500/15 px-3 py-1 font-semibold text-white hover:bg-emerald-500/25"
            >
                Ver página pública
            </Link>
        }
        {creationSuccess.eventId &&
            <Link
                href={`/organizador/eventos/${creationSuccess.eventId}`}
                className="rounded-full border border-emerald-200/60 bg-emerald-500/15 px-3 py-1 font-semibold text-white hover:bg-emerald-500/25"
            >
                Editar evento
            </Link>
        }
        <button
            type="button"
            onClick={resetForm}
            className="rounded-full border border-white/25 px-3 py-1 font-semibold text-white hover:bg-white/10"
        >
            Criar outro
        </button>
    </div>
</div>
)
}

{toasts.length > 0 &&
<div className="pointer-events-none fixed bottom-6 right-6 z-40 flex flex-col gap-2">
    {toasts.map((toast) =>
        <div
            key={toast.id}
            className={`pointer-events-auto min-w-[240px] rounded-lg border px-4 py-3 text-sm shadow-lg ${{
                toast.tone === "success"
                ? "border-emerald-400/50 bg-emerald-500/15 text-emerald-50"
                : "border-red-400/50 bg-red-500/15 text-red-50"
            }}`}
        >
            {toast.message}
        </div>
    )))
</div>
)
}

```

```
    </form>
);
}
```

app/organizador/(dashboard)/eventos/page.tsx

```
// app/organizador/eventos/page.tsx
// LEGACY - não usar na UI; manter apenas enquanto migramos tudo para as tabs do Dashboard.
/* eslint-disable @next/next/no-html-link-for-pages */

import { redirect } from "next/navigation";
import { prisma } from "@/lib/prisma";
import { createSupabaseServer } from "@/lib/supabaseServer";
import { TicketStatus } from "@prisma/client";

export default async function OrganizerEventsPage() {
  // 1) Garante que só entra quem está autenticado
  const supabase = await createSupabaseServer();
  const { data, error } = await supabase.auth.getUser();

  if (error || !data?.user) {
    redirect("/login?redirectTo=/organizador/eventos");
  }

  const userId = data.user.id;

  // 2) Encontrar o organizer associado a este utilizador
  const organizer = await prisma.organizer.findFirst({
    where: { userId },
  });

  if (!organizer) {
    // Ainda não é organizador → envia para a home do painel do organizador
    redirect("/organizador");
  }

  // 3) Buscar eventos deste organizer
  const events = await prisma.event.findMany({
    where: { organizerId: organizer.id },
    orderBy: {
      startsAt: "asc",
    },
    select: {
      id: true,
      slug: true,
      title: true,
      startsAt: true,
      endsAt: true,
      locationName: true,
      locationCity: true,
      status: true,
      organizerId: true,
    },
  });

  const ticketStats = await prisma.ticket.groupBy({
    by: ["eventId"],
    where: {
      status: { in: [TicketStatus.ACTIVE, TicketStatus.USED] },
      event: { organizerId: organizer.id },
    },
    _count: { _all: true },
    _sum: { pricePaid: true, totalPaidCents: true, platformFeeCents: true },
  });

  const statsMap = new Map<number, { tickets: number; revenueCents: number; totalPaidCents: number; platformFeeCents: number }>();
  ticketStats.forEach((stat) => {
    statsMap.set(stat.eventId, {
      tickets: stat._count._all,
      revenueCents: stat._sum.pricePaid ?? 0,
      totalPaidCents: stat._sum.totalPaidCents ?? 0,
      platformFeeCents: stat._sum.platformFeeCents ?? 0,
    });
  });
}
```

```

const now = new Date();
const totalEvents = events.length;
const upcomingEvents = events.filter((e) => e.startsAt > now).length;
const totalTickets = ticketStats.reduce((sum, s) => sum + s._count._all, 0);
const totalRevenueCents = ticketStats.reduce((sum, s) => sum + (s._sum.pricePaid ?? 0), 0);

return (
  <section className="mx-auto max-w-6xl px-4 py-8 md:px-6 md:py-10 space-y-6 text-white">
    {/* Métricas principais (simples) */}
    <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-4">
      <div className="rounded-2xl border border-white/14 bg-white/5 backdrop-blur-xl px-4 py-3.5">
        <p className="text-[11px] text-white/60">Eventos totais</p>
        <p className="mt-1 text-2xl font-semibold tracking-tight">
          {totalEvents}
        </p>
        <p className="mt-1 text-[11px] text-white/55">
          Todos os eventos que já criaste como organizador.
        </p>
      </div>

      <div className="rounded-2xl border border-white/14 bg-white/5 backdrop-blur-xl px-4 py-3.5">
        <p className="text-[11px] text-white/60">Próximos eventos</p>
        <p className="mt-1 text-2xl font-semibold tracking-tight">
          {upcomingEvents}
        </p>
        <p className="mt-1 text-[11px] text-white/55">
          Eventos com data ainda por acontecer.
        </p>
      </div>

      <div className="rounded-2xl border border-white/14 bg-white/5 backdrop-blur-xl px-4 py-3.5">
        <p className="text-[11px] text-white/60">Bilhetes vendidos</p>
        <p className="mt-1 text-2xl font-semibold tracking-tight">
          {totalTickets}
        </p>
        <p className="mt-1 text-[11px] text-white/55">
          Contam apenas bilhetes ativos/usados.
        </p>
      </div>

      <div className="rounded-2xl border border-white/14 bg-white/5 backdrop-blur-xl px-4 py-3.5">
        <p className="text-[11px] text-white/60">Receita bruta</p>
        <p className="mt-1 text-2xl font-semibold tracking-tight">
          {(totalRevenueCents / 100).toFixed(2)} €
        </p>
        <p className="mt-1 text-[11px] text-white/55">
          Soma do preço de bilhete recebido (antes de taxas Stripe).
        </p>
      </div>
    </div>
  </section>
  {/* Lista de eventos */}
  <section className="rounded-2xl border border-white/12 bg-black/40 backdrop-blur-xl p-5 space-y-4">
    <div className="flex items-center justify-between gap-2">
      <div>
        <h2 className="text-sm font-semibold text-white/90">
          Meus eventos
        </h2>
        <p className="text-[11px] text-white/65">
          Lista de eventos criados por ti como organizador.
        </p>
      </div>
    </div>
    <div flex-col col-md:flex-row md:items-center md:justify-between gap-3>
      <div>
        <p className="font-medium text-white/85">
          Ainda não tens eventos criados.
        </p>
        <p className="mt-1 text-white/60">
          Cria o teu primeiro evento para começares a vender bilhetes e
          testar o fluxo completo da ORYA.
        </p>
      </div>
      <a href="/organizador/eventos/novo">

```

```

    className="inline-flex px-3 py-1.5 rounded-xl bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-[11px] font-semibold text-black hover:scale-[1.03] active:scale-95 transition-transform shadow-[0_0_24px_rgba(107,255,255,0.6)]"
    >
      Criar evento
    </a>
</div>
)}

{events.length > 0 && (
  <div className="mt-3 space-y-3">
    {events.map((event) => {
      const stats = statsMap.get(event.id);
      const ticketsSold = stats?.tickets ?? 0;
      const revenueEuro = ((stats?.revenueCents ?? 0) / 100).toFixed(2);

      const isPast = event.startsAt < now;
      const statusLabel = isPast
        ? "Terminado"
        : event.status === "PUBLISHED"
        ? "Publicado"
        : event.status === "DRAFT"
        ? "Rascunho"
        : event.status;

      const statusClasses = isPast
        ? "bg-white/8 border-white/30 text-white/80"
        : "bg-emerald-500/10 border-emerald-400/60 text-emerald-100";

      const dateFormatted = event.startsAt.toLocaleString("pt-PT", {
        weekday: "short",
        day: "2-digit",
        month: "short",
        hour: "2-digit",
        minute: "2-digit",
      });
    });
  return (
    <article
      key={event.id}
      className="rounded-xl border border-white/14 bg-gradient-to-br from-white/5 via-black/80 to-black/95 px-4
py-3.5 flex flex-col gap-2.5"
    >
      <div className="flex items-start justify-between gap-3">
        <div>
          <h3 className="text-sm font-semibold text-white/95">
            {event.title}
          </h3>
          <p className="mt-0.5 text-[11px] text-white/60">
            {dateFormatted}
            {event.locationName ? ` • ${event.locationName}` : ""}
            {event.locationCity ? `, ${event.locationCity}` : ""}
          </p>
          <p className="mt-1 text-[11px] text-white/65">
            {ticketsSold} bilhetes · {revenueEuro} €
          </p>
          <p className="mt-1 text-[10px] text-white/55">
            Slug:{" "}
            <span className="font-mono text-[10px] text-white/75">
              {event.slug}
            </span>
          </p>
        </div>
      <div className="flex flex-col items-end gap-1">
        <span
          className={`${`px-2 py-1 rounded-full border text-[10px] ${statusClasses}`}`}
        >
          {statusLabel}
        </span>
      </div>
    </div>
  );
}

/* Ações */
<div className="mt-2 flex flex-wrap items-center justify-between gap-3 text-[10px]">
  <div className="flex items-center gap-2 text-white/60">
    <span className="inline-flex h-5 w-5 items-center justify-center rounded-full bg-white/8 border border-
white/15">
      <span>
    </span>
  </div>

```

```

        <span>
          Gere bilhetes, detalhes e estatísticas na página de
          gestão do evento.
        </span>
      </div>

      <div className="flex items-center gap-2">
        <a href={`/eventos/${event.slug}`}
            className="px-3 py-1.5 rounded-xl border border-white/25 bg-white/5 text-[10px] text-white/85
            hover:bg-white/10 transition">
          >
            Ver página pública
        </a>
        <a href={`/organizador/eventos/${event.id}`}
            className="px-3 py-1.5 rounded-xl bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-[10px] font-semibold text-black hover:scale-[1.02] active:scale-95 transition-transform shadow-[0_0_20px_rgba(107,255,255,0.6)]">
          >
            Detalhes & gestão
        </a>
      </div>
    </div>
  </article>
);
}
)}
)
</div>
</section>
</section>
);
}
}

```

app/organizador/(dashboard)/layout.tsx

```

export const runtime = "nodejs";

import { ReactNode, CSSProperties } from "react";
import Link from "next/link";
import { OrganizerSidebar } from "../OrganizerSidebar";
import { OrganizationSwitcher, type OrganizationSwitcherOption } from "../OrganizationSwitcher";
import { createSupabaseServer } from "@lib/supabaseServer";
import { getActiveOrganizerForUser } from "@lib/organizerContext";
import { prisma } from "@lib/prisma";
import { OrganizerLangSetter } from "../OrganizerLangSetter";
import { RoleBadge } from "../RoleBadge";
import { DASHBOARD_LABEL, DASHBOARD_SHELL_PADDING } from "../dashboardUi";

/**
 * Layout do dashboard do organizador (sidebar + topbar).
 * Não contém lógica de autenticação; isso é tratado no layout pai /organizador.
 * Busca o organizer ativo no server para alimentar o switcher e reduzir fetches client.
 */
export default async function OrganizerDashboardLayout({ children }: { children: ReactNode }) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  let currentId: number | null = null;
  let orgOptions: OrganizationSwitcherOption[] = [];
  let activeOrganizer: {
    id: number;
    displayName: string | null;
    publicName?: string | null;
    businessName: string | null;
    username: string | null;
    brandingAvatarUrl?: string | null;
    brandingPrimaryColor?: string | null;
    brandingSecondaryColor?: string | null;
    language?: string | null;
  } | null = null;
  let activeRole: string | null = null;
  if (user) {
    try {
      const { organizer, membership } = await getActiveOrganizerForUser(user.id);
      currentId = organizer.id;
      activeOrganizer = organizer;
      activeRole = membership.role;
      orgOptions = [
        ...Object.values(prisma.organicationSwitcherOption),
        { id: 0, name: "Novo organizador" },
      ];
    } catch (err) {
      console.error("Error getting active organizer", err);
    }
  }
  return (
    <div style={{ display: "flex", height: "100%", gap: DASHBOARD_SHELL_PADDING }}>
      <OrganizerSidebar
        activeOrganizer={activeOrganizer}
        orgOptions={orgOptions}
        currentId={currentId}
        activeRole={activeRole}
      />
      <div style={{ flex: 1 }}>
        {children}
      </div>
    </div>
  );
}

```

```

currentId = organizer?.id ?? null;
if (organizer && membership) {
  activeRole = membership.role;
  activeOrganizer = {
    id: organizer.id,
    displayName: organizer.displayName,
    publicName: (organizer as { publicName?: string | null }).publicName ?? null,
    businessName: organizer.businessName,
    username: (organizer as { username?: string | null }).username ?? null,
    brandingAvatarUrl: (organizer as { brandingAvatarUrl?: string | null }).brandingAvatarUrl ?? null,
    brandingPrimaryColor: (organizer as { brandingPrimaryColor?: string | null }).brandingPrimaryColor ?? null,
    brandingSecondaryColor: (organizer as { brandingSecondaryColor?: string | null }).brandingSecondaryColor ?? null,
    language: (organizer as { language?: string | null }).language ?? null,
  };
}
} catch {
  currentId = null;
}

try {
  const memberships = await prisma.organizerMember.findMany({
    where: { userId: user.id },
    include: { organizer: true },
    orderBy: [{ lastUsedAt: "desc" }, { createdAt: "asc" }],
  });

  orgOptions = memberships
    .filter((m) => m.organizer)
    .map((m) => ({
      organizerId: m.organizerId,
      role: m.role,
      organizer: {
        id: m.organizer!.id,
        username: m.organizer!.username,
        displayName: m.organizer!.displayName,
        publicName: (m.organizer as { publicName?: string | null }).publicName ?? null,
        businessName: m.organizer!.businessName,
        city: m.organizer!.city,
        entityType: m.organizer!.entityType,
        status: m.organizer!.status,
        brandingAvatarUrl: (m.organizer as { brandingAvatarUrl?: string | null }).brandingAvatarUrl ?? null,
      },
    }));
} catch (err: unknown) {
  const msg =
    typeof err === "object" && err && "message" in err ? String((err as { message?: unknown }).message) : "";
  if (!(msg.includes("does not exist") || msg.includes("organizer_members"))) {
    throw err;
  }
}

const organizerName =
  activeOrganizer?.publicName || activeOrganizer?.displayName || activeOrganizer?.businessName || "Organizador";
const organizerAvatarUrl = activeOrganizer?.brandingAvatarUrl ?? null;
const organizerUsername = activeOrganizer?.username ?? null;
const brandPrimary = activeOrganizer?.brandingPrimaryColor ?? undefined;
const brandSecondary = activeOrganizer?.brandingSecondaryColor ?? undefined;
const organizerLanguage = activeOrganizer?.language ?? "pt";

return (
  <div
    className="orya-body-bg h-screen text-white flex overflow-hidden"
    style={{
      "--brand-primary": brandPrimary,
      "--brand-secondary": brandSecondary,
    } as CSSProperties
  >
  <OrganizerLangSetter language={organizerLanguage} />
  {organizerUsername ? (
    <script
      dangerouslySetInnerHTML={{ __html: `try{sessionStorage.setItem("orya_last_organizer_username","${organizerUsername}");}catch(e){}` }}
    >
    />
  ) : null}

```

```

<OrganizerSidebar organizerName={organizerName} organizerAvatarUrl={organizerAvatarUrl} />

<div className="flex-1 flex flex-col min-h-0">
  {/* Top bar */}
  <header className={`${sticky_top-0 z-30 border-b border-white/10 bg-[#050915]/85 backdrop-blur-xl
${DASHBOARD_SHELL_PADDING} py-3 shrink-0`}>
    <div className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between">
      <div className="flex items-center gap-3">
        <div className="h-11 w-11 overflow-hidden rounded-2xl border border-white/15 bg-gradient-to-br from-[#0f172a] via-
[#111827] to-[#0b1224] shadow-[0_10px_40px_rgba(0,0,0,0.4)]">
          {organizerAvatarUrl ? (
            // eslint-disable-next-line @next/next/no-img-element
            <img src={organizerAvatarUrl} alt={organizerName} className="h-full w-full object-cover" />
          ) : (
            <div className="flex h-full w-full items-center justify-center text-xs font-black tracking-[0.22em] text-
[#6BFFFF]">
              OY
            </div>
          )}
        </div>
        <div className="space-y-1">
          <p className={DASHBOARD_LABEL}>Organização</p>
          <div className="flex flex-wrap items-center gap-2">
            <span className="text-lg font-semibold text-white">{organizerName}</span>
            {activeRole && <RoleBadge role={activeRole as any} />}
          </div>
          {organizerUsername && (
            <p className="text-[12px] text-white/55">{@organizerUsername}</p>
          )}
        </div>
      </div>
      <div className="flex flex-wrap items-center justify-end gap-2 text-[12px]">
        <Link
          href="/organizador/eventos/novo"
          className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow hover:scale-[1.01]"
        >
          Criar evento
        </Link>
        <details className="relative">
          <summary className="list-none cursor-pointer rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-
white/80 hover:bg-white/10">
            Mais opções ▾
          </summary>
          <div className="absolute right-0 mt-2 w-52 rounded-2xl border border-white/12 bg-black/85 p-2 text-[12px] text-
white/80 shadow-[0_20px_50px_rgba(0,0,0,0.7)]">
            <Link
              href={organizerUsername ? `/o/${organizerUsername}` : "/explorar"}
              className="block w-full rounded-xl px-2.5 py-1.5 text-left hover:bg-white/8"
            >
              Ver página pública
            </Link>
          </div>
        </details>
        <OrganizationSwitcher currentId={currentId} initialOrgs={orgOptions} />
      </div>
    </div>
  </header>
  <main className="flex-1 overflow-y-auto pb-0 pt-0 min-h-0">{children}</main>
</div>
</div>
);
}

```

app/organizador/(dashboard)/organizations/page.tsx

```

import { redirect } from "next/navigation";
import { prisma } from "@lib/prisma";
import { createSupabaseServer } from "@lib/supabaseServer";
import { getActiveOrganizerForUser } from "@lib/organizerContext";
import OrganizationsHubClient from "../../organizations/OrganizationsHubClient";
import { cookies } from "next/headers";

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

```

```

type OrgPayload = {
  organizerId: number;
  role: string;
  lastUsedAt: string | null;
  organizer: {
    id: number;
    username: string | null;
    displayName: string | null;
    businessName: string | null;
    city: string | null;
    entityType: string | null;
    status: string | null;
  };
};

export default async function OrganizationsHubPage() {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) {
    redirect("/login?next=/organizador/organizations");
  }

  let orgs: OrgPayload[] = [];

  try {
    const memberships = await prisma.organizerMember.findMany({
      where: { userId: user.id },
      include: { organizer: true },
      orderBy: [{ lastUsedAt: "desc" }, { createdAt: "asc" }],
    });

    orgs = memberships
      .filter((m) => m.organizer)
      .map((m) => {
        organizerId: m.organizerId,
        role: m.role,
        lastUsedAt: m.lastUsedAt ? m.lastUsedAt.toISOString() : null,
        organizer: {
          id: m.organizer!.id,
          username: m.organizer!.username,
          displayName: m.organizer!.displayName,
          businessName: m.organizer!.businessName,
          city: m.organizer!.city,
          entityType: m.organizer!.entityType,
          status: m.organizer!.status,
        },
      });
  } catch (err) {
    const msg =
      typeof err === "object" && err && "message" in err ? String((err as { message?: unknown }).message) : "";
    // Se a tabela ainda não existir em dev, deixa orgs = []
    if (!(msg.includes("does not exist") || msg.includes("organizer_members"))) {
      throw err;
    }
  }

  // Se não houver nenhuma organização, envia para o onboarding
  if (orgs.length === 0) {
    redirect("/organizador/become");
  }

  const cookieStore = await cookies();
  const cookieOrgId = cookieStore.get("orya_org")?.value;
  const forcedOrgId = cookieOrgId ? Number(cookieOrgId) : undefined;
  const { organizer: activeOrganizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: Number.isFinite(forcedOrgId) ? forcedOrgId : undefined,
  });
  const activeId = activeOrganizer?.id ?? (Number.isFinite(forcedOrgId) ? forcedOrgId! : null);

  return <OrganizationsHubClient initialOrgs={orgs} activeId={activeId} />;
}

```

app/organizador/(dashboard)/padel/PadelHubClient.tsx

```

"use client";

import Link from "next/link";
import { useEffect, useMemo, useState } from "react";
import useSWR from "swr";
import { ConfirmDestructiveActionDialog } from "@app/components/ConfirmDestructiveActionDialog";
import { trackEvent } from "@lib/analytics";
import { PORTUGAL_CITIES } from "@config/cities";

type PadelClub = {
  id: number;
  name: string;
  city: string | null;
  address: string | null;
  courtsCount: number;
  slug?: string | null;
  isActive: boolean;
  isDefault?: boolean;
  createdAt: string | Date;
};

type PadelClubCourt = {
  id: number;
  padelClubId: number;
  name: string;
  description: string | null;
  indoor: boolean;
  isActive: boolean;
  displayOrder: number;
};

type PadelClubStaff = {
  id: number;
  padelClubId: number;
  userId: string | null;
  email: string | null;
  fullName?: string | null;
  role: string;
  inheritToEvents: boolean;
};

type Player = {
  id: number;
  fullName: string;
  email: string | null;
  phone: string | null;
  level: string | null;
  isActive: boolean;
  createdAt: string | Date;
  tournamentsCount?: number;
};

type Props = {
  organizerId: number;
  organizationKind: string | null;
  initialClubs: PadelClub[];
  initialPlayers: Player[];
};

type OrganizerStaffMember = {
  userId: string;
  fullName: string | null;
  username: string | null;
  email: string | null;
  role: string | null;
};

type OrganizerStaffResponse = {
  ok: boolean;
  items: OrganizerStaffMember[];
};

const DEFAULT_FORM = {
  id: null as number | null,
  name: "",
  city: "",
  address: ""
};

```

```

courtsCount: "1",
isActive: true,
slug: "",
isDefault: false,
};

const DEFAULT_COURT_FORM = {
  id: null as number | null,
  name: "",
  description: "",
  indoor: false,
  isActive: true,
  displayOrder: 0,
};

const DEFAULT_STAFF_FORM = {
  id: null as number | null,
  email: "",
  staffMemberId: "",
  role: "STAFF",
  inheritToEvents: true,
};

const badge = (tone: "green" | "amber" | "slate" = "slate") =>
`rounded-full border px-2 py-[4px] text-[11px] ${
  tone === "green"
    ? "border-emerald-400/40 bg-emerald-500/15 text-emerald-100"
  : tone === "amber"
    ? "border-amber-300/40 bg-amber-400/10 text-amber-100"
  : "border-white/15 bg-white/10 text-white/70"
}`;

const fetcher = (url: string) => fetch(url).then((r) => r.json());

export default function PadelHubClient({ organizerId, organizationKind, initialClubs, initialPlayers }: Props) {
  const [activeTab, setActiveTab] = useState<"clubs" | "players" | "rankings">("clubs");
  const [clubs, setClubs] = useState<PadelClub[]>(initialClubs);
  const [players, setPlayers] = useState<Player[]>(initialPlayers);
  const [search, setSearch] = useState("");

  const [clubForm, setClubForm] = useState(DEFAULT_FORM);
  const [clubModalOpen, setClubModalOpen] = useState(false);
  const [savingClub, setSavingClub] = useState(false);
  const [clubError, setClubError] = useState<string | null>(null);
  const [clubMessage, setClubMessage] = useState<string | null>(null);

  const [drawerClubId, setDrawerClubId] = useState<number | null>(initialClubs[0]?.id ?? null);
  const [courts, setCourts] = useState<PadelClubCourt[]>([]);
  const [staff, setStaff] = useState<PadelClubStaff[]>([]);
  const [loadingDrawer, setLoadingDrawer] = useState(false);

  const [courtForm, setCourtForm] = useState(DEFAULT_COURT_FORM);
  const [courtMessage, setCourtMessage] = useState<string | null>(null);
  const [courtError, setCourtError] = useState<string | null>(null);
  const [savingCourt, setSavingCourt] = useState(false);
  const [courtDialog, setCourtDialog] = useState<{ court: PadelClubCourt; nextActive: boolean } | null>(null);

  const [staffForm, setStaffForm] = useState(DEFAULT_STAFF_FORM);
  const [staffMode, setStaffMode] = useState<"existing" | "external">("existing");
  const [staffSearch, setStaffSearch] = useState("");
  const [staffMessage, setStaffMessage] = useState<string | null>(null);
  const [staffError, setStaffError] = useState<string | null>(null);
  const [draggingCourtId, setDraggingCourtId] = useState<number | null>(null);
  const [clubDialog, setClubDialog] = useState<{ club: PadelClub; nextActive: boolean } | null>(null);

  const { data: organizerStaff } = useSWR<OrganizerStaffResponse>(
    organizerId ? `/api/organizador/organizations/members?organizerId=${organizerId}` : null,
    fetcher,
    { revalidateOnFocus: false },
  );
}

const hasActiveClub = useMemo(() => clubs.some((c) => c.isActive), [clubs]);
const sortedClubs = useMemo(() => {
  return [...clubs].sort((a, b) => {
    if (a.isActive !== b.isActive) return a.isActive ? -1 : 1;
    return new Date(b.createdAt).getTime() - new Date(a.createdAt).getTime();
  });
}, [clubs]);

```

```

const selectedClub = useMemo(() => clubs.find((c) => c.id === drawerClubId) || null, [clubs, drawerClubId]);

const filteredPlayers = useMemo(() => {
  const term = search.trim().toLowerCase();
  if (!term) return players;
  return players.filter((p) => p.fullName.toLowerCase().includes(term) || (p.email || "").toLowerCase().includes(term));
}, [players, search]);

const activeCourtsCount = useMemo(() => courts.filter((c) => c.isActive).length, [courts]);
const staffOptions = useMemo(() => {
  const list = organizerStaff?.items ?? [];
  const term = staffSearch.trim().toLowerCase();
  const filtered = term
    ? list.filter(
        (m) =>
          (m.fullName || "").toLowerCase().includes(term) ||
          (m.email || "").toLowerCase().includes(term) ||
          (m.username || "").toLowerCase().includes(term),
      )
    : list;
  return filtered;
}, [organizerStaff?.items, staffSearch]);
// Mantém a ordem recebida e renumeria sequencialmente
const renumberCourts = (list: PadelClubCourt[]) =>
  list.map((c, idx) => ({ ...c, displayOrder: idx + 1 }));

useEffect(() => {
  if (!drawerClubId) {
    setCourts([]);
    setStaff([]);
    return;
  }
  loadCourtsAndStaff(drawerClubId);
}, [drawerClubId]);

useEffect(() => {
  setClubs(initialClubs);
}, [initialClubs]);

useEffect(() => {
  setPlayers(initialPlayers);
}, [initialPlayers]);

useEffect(() => {
  if (drawerClubId) return;
  if (clubs.length === 0) return;
  const preferred = clubs.find((c) => c.isActive) ?? clubs[0];
  setDrawerClubId(preferred.id);
}, [clubs, drawerClubId]);

const persistCourtOrder = async (list: PadelClubCourt[]) => {
  if (!selectedClub) return;
  const payload = renumberCourts(list);
  const updates = payload.map((c) =>
    fetch(`/api/padel/clubs/${selectedClub.id}/courts`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ ...c, name: c.name, description: c.description || "", surface: null }),
    }).catch((err) => {
      console.error("[padel/clubs/reorder] failed", err);
      return null;
    })
  );
  await Promise.all(updates);
};

const reorderCourts = (targetId: number) => {
  if (!draggingCourtId || draggingCourtId === targetId) return null;
  const current = [...courts];
  const from = current.findIndex((ct) => ct.id === draggingCourtId);
  const to = current.findIndex((ct) => ct.id === targetId);
  if (from === -1 || to === -1) return null;
  const [moved] = current.splice(from, 1);
  current.splice(to, 0, moved);
  const renumbered = renumberCourts(current);
  setCourts(renumbered);
}

```

```

        return renumbered;
    };

    useEffect(() => {
        if (courtForm.id) return;
        const nextOrder = Math.max(1, activeCourtsCount + 1);
        if (courtForm.displayOrder !== nextOrder) {
            setCourtForm((prev) => ({ ...prev, displayOrder: nextOrder }));
        }
        // eslint-disable-next-line react-hooks/exhaustive-deps
    }, [activeCourtsCount, courtForm.id]);

    const openNewClubModal = () => {
        setClubForm(DEFAULT_FORM);
        setClubError(null);
        setClubMessage(null);
        setClubModalOpen(true);
    };

    const openEditClubModal = (club: PadelClub) => {
        setClubForm({
            id: club.id,
            name: club.name,
            city: club.city || "",
            address: club.address || "",
            courtsCount: club.courtsCount ? String(club.courtsCount) : "1",
            isActive: club.isActive,
            slug: club.slug || "",
            isDefault: Boolean(club.isDefault),
        });
        setClubError(null);
        setClubMessage(null);
        setClubModalOpen(true);
    };

    const loadCourtsAndStaff = async (clubId: number) => {
        setLoadingDrawer(true);
        setCourtMessage(null);
        setCourtError(null);
        setStaffMessage(null);
        setStaffError(null);
        try {
            const [courtsRes, staffRes] = await Promise.all([
                fetch(`/api/padel/clubs/${clubId}/courts`),
                fetch(`/api/padel/clubs/${clubId}/staff`),
            ]);
            const courtsJson = await courtsRes.json().catch(() => null);
            const staffJson = await staffRes.json().catch(() => null);
            if (courtsRes.ok && Array.isArray(courtsJson?.items)) setCourts(courtsJson.items as PadelClubCourt[]);
            else setCourtError(courtsJson?.error || "Erro ao carregar courts.");
            if (staffRes.ok && Array.isArray(staffJson?.items)) setStaff(staffJson.items as PadelClubStaff[]);
            else setStaffError(staffJson?.error || "Erro ao carregar equipa.");
        } catch (err) {
            console.error("[padel/clubs] load courts/staff", err);
            setCourtError("Erro ao carregar courts.");
            setStaffError("Erro ao carregar equipa.");
        } finally {
            setLoadingDrawer(false);
        }
    };
}

const handleSubmitClub = async () => {
    setClubError(null);
    setClubMessage(null);
    if (!clubForm.name.trim()) {
        setClubError("Nome do clube é obrigatório.");
        return;
    }
    const courtsNum = Number(clubForm.courtsCount);
    const courtsCount = Number.isFinite(courtsNum) ? Math.min(1000, Math.max(1, Math.floor(courtsNum))) : 1;
    setSavingClub(true);
    try {
        const res = await fetch("/api/padel/clubs", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                id: clubForm.id,
                organizerId,
            })
        });
        if (res.ok) {
            setClubModalOpen(true);
            setClubError(null);
            setClubMessage("Clube criado com sucesso!");
        } else {
            const error = await res.json();
            setClubError(error.message);
        }
    } catch (err) {
        setClubError("Ocorreu um erro ao tentar criar o clube.");
    }
};


```

```

        name: clubForm.name.trim(),
        city: clubForm.city.trim(),
        address: clubForm.address.trim(),
        courtsCount,
        isActive: clubForm.isActive,
        slug: clubForm.slug?.trim(),
        isDefault: clubForm.isDefault,
    }),
);
const json = await res.json().catch(() => null);
if (!res.ok || json?.ok === false) {
    setClubError(json?.error || "Erro ao guardar clube.");
} else {
    const club = json.club as PadelClub;
    setClubs((prev) => {
        const existing = prev.some((c) => c.id === club.id);
        if (existing) return prev.map((c) => (c.id === club.id ? club : c));
        return [club, ...prev];
    });
    setClubMessage(clubForm.id ? "Clube atualizado." : "Clube criado.");
    setClubModalOpen(false);
    setClubForm({ ...DEFAULT_FORM, courtsCount: String(courtsCount) });
    setDrawerClubId((prev) => prev ?? club.id);
    trackEvent(clubForm.id ? "padel_club_updated" : "padel_club_created", { clubId: club.id });
}
} catch (err) {
    console.error("[padel/clubs] save", err);
    setClubError("Erro inesperado ao guardar clube.");
} finally {
    setSavingClub(false);
}
};

const markDefaultClub = async (club: PadelClub) => {
setClubError(null);
setClubMessage(null);
try {
    const res = await fetch("/api/padel/clubs", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
            id: club.id,
            organizerId,
            name: club.name,
            city: club.city,
            address: club.address,
            courtsCount: club.courtsCount,
            isActive: club.isActive,
            slug: club.slug,
            isDefault: true,
        }),
    });
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {
        setClubError(json?.error || "Erro ao definir default.");
    } else {
        const saved = json.club as PadelClub;
        setClubs((prev) => prev.map((c) => ({ ...c, isDefault: c.id === saved.id })));
        setClubMessage("Clube definido como predefinido.");
        trackEvent("padel_club_marked_default", { clubId: saved.id });
    }
} catch (err) {
    console.error("[padel/clubs] default", err);
    setClubError("Erro inesperado ao definir default.");
}
};

const resetCourtForm = () => {
setCourtForm(DEFAULT_COURT_FORM);
setCourtMessage(null);
setCourtError(null);
};

const handleEditCourt = (court: PadelClubCourt) => {
setCourtForm({
    id: court.id,
    name: court.name,
    description: court.description || "",
}
);

```

```

    indoor: court.indoor,
    isActive: court.isActive,
    displayOrder: court.displayOrder,
  });
};

const handleSubmitCourt = async () => {
  if (!selectedClub) return;
  const fallbackName = courtForm.name.trim() || `Court ${courts.length + 1}`;
  const desiredOrder = Number.isFinite(courtForm.displayOrder) ? Math.max(1, Math.floor(courtForm.displayOrder)) : 1;
  const maxOrder = Math.max(1, activeCourtsCount + (courtForm.id ? 0 : courtForm.isActive ? 1 : 0));
  const normalizedOrder = Math.min(maxOrder, desiredOrder);
  setSavingCourt(true);
  setCourtError(null);
  setCourtMessage(null);
  try {
    const res = await fetch(`/api/padel/clubs/${selectedClub.id}/courts`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        ...courtForm,
        name: fallbackName,
        description: courtForm.description.trim(),
        surface: null,
        displayOrder: normalizedOrder,
      }),
    });
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {
      setCourtError(json?.error || "Erro ao guardar court.");
    } else {
      const court = json.court as PadelClubCourt;
      setCourts((prev) => {
        const exists = prev.some((c) => c.id === court.id);
        const updated = exists ? prev.map((c) => (c.id === court.id ? court : c)) : [...prev, court];
        return renumberCourts(updated);
      });
      trackEvent(courtForm.id ? "padel_court_updated" : "padel_court_created", {
        clubId: selectedClub.id,
        indoor: court.indoor,
      });
      setCourtMessage(courtForm.id ? "Court atualizado." : "Court criado.");
      resetCourtForm();
    }
  } catch (err) {
    console.error("[padel/clubs/courts] save", err);
    setCourtError("Erro inesperado ao guardar court.");
  } finally {
    setSavingCourt(false);
  }
};

const handleConfirmCourtToggle = async () => {
  if (!courtDialog || !selectedClub) return;
  await handleToggleCourtActive(courtDialog.court, courtDialog.nextActive);
  trackEvent(courtDialog.nextActive ? "padel_court_reactivated" : "padel_court_deactivated", {
    clubId: selectedClub.id,
    courtId: courtDialog.court.id,
  });
  setCourtDialog(null);
};

const handleToggleClubActive = async (club: PadelClub, next: boolean) => {
  setClubError(null);
  setClubMessage(null);
  try {
    const res = await fetch("/api/padel/clubs", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        id: club.id,
        organizerId,
        name: club.name,
        city: club.city,
        address: club.address,
        courtsCount: club.courtsCount,
        isActive: next,
        slug: club.slug,
      })
    });
  } catch (err) {
    console.error("[padel/clubs] update", err);
  }
};

```

```

        isDefault: club.isDefault,
    }),
});
const json = await res.json().catch(() => null);
if (!res.ok || json?.ok === false) {
    setClubError(json?.error || "Erro ao atualizar estado do clube.");
} else {
    const saved = json.club as PadelClub;
    setClubs((prev) => prev.map((c) => (c.id === saved.id ? saved : c)));
    setClubMessage(saved.isActive ? "Clube reativado." : "Clube arquivado.");
    trackEvent(saved.isActive ? "padel_club_reactivated" : "padel_club_archived", { clubId: saved.id });
}
} catch (err) {
    console.error("[padel/clubs] toggle active", err);
    setClubError("Erro inesperado ao atualizar clube.");
}
};

const handleToggleCourtActive = async (court: PadelClubCourt, next: boolean) => {
    if (!selectedClub) return;
    setSavingCourt(true);
    try {
        const res = await fetch(`api/padel/clubs/${selectedClub.id}/courts`, {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ ...court, isActive: next }),
        });
        const json = await res.json().catch(() => null);
        if (res.ok && json.court) {
            const updated = json.court as PadelClubCourt;
            setCourts((prev) => renumberCourts(prev.map((c) => (c.id === updated.id ? updated : c))));
        }
    } catch (err) {
        console.error("[padel/clubs/courts] toggle", err);
    } finally {
        setSavingCourt(false);
    }
};

const resetStaffForm = () => {
    setStaffForm(DEFAULT_STAFF_FORM);
    setStaffMode("existing");
    setStaffSearch("");
    setStaffError(null);
    setStaffMessage(null);
};

const handleEditStaff = (member: PadelClubStaff) => {
    setStaffForm({
        id: member.id,
        email: member.email || "",
        staffMemberId: member.userId || "",
        role: member.role,
        inheritToEvents: member.inheritToEvents,
    });
    setStaffMode(member.userId ? "existing" : "external");
};

const handleSubmitStaff = async () => {
    if (!selectedClub) return;
    const selectedMember = staffMode === "existing" ? staffOptions.find((m) => m.userId === staffForm.staffMemberId) : null;
    const emailToSend =
        staffMode === "existing" ? selectedMember?.email ?? "" : staffForm.email.trim();
    if (staffMode === "existing" && !selectedMember) {
        setStaffError("Escolhe um membro do staff global.");
        return;
    }
    if (staffMode === "external" && !emailToSend) {
        setStaffError("Indica o email do contacto externo.");
        return;
    }
    setStaffError(null);
    setStaffMessage(null);
    try {
        const res = await fetch(`api/padel/clubs/${selectedClub.id}/staff`, {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({

```

```

    id: staffForm.id,
    email: emailToSend,
    userId: staffMode === "existing" ? selectedMember?.userId : null,
    role: staffForm.role,
    inheritToEvents: staffForm.inheritToEvents,
  )),
);
const json = await res.json().catch(() => null);
if (!res.ok || json?.ok === false) {
  setStaffError(json?.error || "Erro ao guardar membro.");
} else {
  const member = json.staff as PadelClubStaff;
  setStaff((prev) => {
    const exists = prev.some((s) => s.id === member.id);
    if (exists) return prev.map((s) => (s.id === member.id ? member : s));
    return [member, ...prev];
  });
  setStaffMessage(staffForm.id ? "Membro atualizado." : "Membro adicionado.");
  resetStaffForm();
}
} catch (err) {
  console.error("[padel/clubs/staff] save", err);
  setStaffError("Erro inesperado ao guardar membro.");
}
};

const compactAddress = (club: PadelClub) => {
  const bits = [club.city, club.address].filter(Boolean);
  return bits.join(" · ") || "Morada por definir";
};

return (
  <div className="space-y-5 rounded-3xl border border-white/10 bg-black/35 px-4 py-5 shadow-[0_22px_70px_rgba(0,0,0,0.55)] md:px-6 md:py-6">
    <header className="flex flex-wrap items-start justify-between gap-3 rounded-2xl border border-white/10 bg-gradient-to-r from-[#0b1021] via-[#0d152f] to-[#0f1c3d] px-4 py-4">
      <div className="space-y-1">
        <p className="text-[11px] uppercase tracking-[0.2em] text-white/55">Padel</p>
        <h1 className="text-2xl font-semibold">Clubes, courts, staff e jogadores.</h1>
        <p className="text-sm text-white/65">Todo num hub único. Copy curta e ações claras.</p>
      </div>
      <div className="flex flex-wrap items-center gap-2">
        <button
          type="button"
          onClick={openNewClubModal}
          className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow hover:scale-[1.01]">
          Novo clube
        </button>
        <Link
          href="/organizador/eventos/novo?templateType=PADEL"
          className="rounded-full border border-white/15 bg-white/10 px-4 py-2 text-sm font-semibold text-white hover:border-white/30">
          Criar torneio
        </Link>
      </div>
    </header>
    <div className="grid grid-cols-1 gap-3 rounded-2xl border border-white/10 bg-black/40 p-4 sm:grid-cols-3">
      <div>
        <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Clubes</p>
        <p className="text-2xl font-semibold">{clubs.length}</p>
        <p className="text-[12px] text-white/60">{hasActiveClub ? "Ativos e prontos a usar." : "Ativa pelo menos um."}</p>
      </div>
      <div>
        <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Courts ativos</p>
        <p className="text-2xl font-semibold">
          {clubs.reduce((acc, c) => acc + (c.courtsCount || 0), 0) || "-"}
        </p>
        <p className="text-[12px] text-white/60">Usados como sugestão no wizard.</p>
      </div>
      <div>
        <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Jogadores</p>
        <p className="text-2xl font-semibold">{players.length}</p>
        <p className="text-[12px] text-white/60">Roster auto-alimentado pelas inscrições.</p>
      </div>
    </div>
  </div>
)

```

```

<div className="flex flex-wrap gap-2 border-b border-white/10 pb-3">
  [
    { key: "clubs", label: "Clubes" },
    { key: "players", label: "Jogadores" },
    { key: "rankings", label: "Rankings (em breve)" },
  ].map((tab) => (
    <button
      key={tab.key}
      className={`rounded-full border px-3 py-1 text-[12px] ${activeTab === tab.key ? "border-white/80 bg-white text-black" : "border-white/10 bg-white/5 text-white/75 hover:border-white/25"}`}
    >
      {tab.label}
    </button>
  )));
</div>

{activeTab === "clubs" && (
  <div className="space-y-4">
    {sortedClubs.length === 0 ? (
      <div className="rounded-2xl border border-dashed border-white/20 bg-black/35 p-6 text-white">
        <p className="text-lg font-semibold">Ainda sem clubes.</p>
        <p className="text-sm text-white/70">Adiciona o primeiro para preencher morada e courts no wizard.</p>
      </div>
    ) : (
      <div className="grid gap-3 md:grid-cols-2 xl:grid-cols-3">
        {sortedClubs.map((club) => {
          const activeCourts = club.courtsCount || 0;
          return (
            <div
              key={club.id}
              className={`rounded-2xl p-4 shadow-[0_16px_60px_rgba(0,0,0,0.45)] ${club.isActive ? "border border-emerald-400/40 bg-emerald-500/5" : "border border-red-500/40 bg-red-500/8"}`}
            >
              <div className="flex items-start justify-between gap-3">
                <div className="space-y-1">
                  <p className="text-base font-semibold text-white">{club.name}</p>
                  <p className="text-[12px] text-white/65">{compactAddress(club)}</p>
                  <p className="text-[12px] text-white/55">Courts ativos: {activeCourts}</p>
                </div>
                <span
                  className={
                    club.isActive ? badge("green") : "rounded-full border border-red-400/50 bg-red-500/15 px-3 py-1 text-[12px] text-red-100"
                  }
                >
                  {club.isActive ? "Ativo" : "Inativo"}
                </span>
              </div>
            </div>
            <button
              type="button"
              onClick={() => {
                setDrawerClubId(club.id);
                loadCourtsAndStaff(club.id);
              }}
              className="rounded-full border border-white/20 px-3 py-1.5 text-[12px] text-white hover:border-white/30"
            >
              Courts & equipa
            </button>
            <button
              type="button"
            >

```

```

        onClick={() => openEditClubModal(club)}
        className="rounded-full border border-white/20 px-3 py-1.5 text-[12px] text-white hover:border-white/30"
      >
  Editar
</button>
{!club.isDefault && (
  <button
    type="button"
    onClick={() => markDefaultClub(club)}
    className="rounded-full border border-white/20 px-3 py-1.5 text-[12px] text-white/80 hover:border-
white/30"
  >
    Tornar default
  </button>
)}
<button
  type="button"
  onClick={() => setClubDialog({ club, nextActive: !club.isActive })}
  className={`rounded-full border px-3 py-1.5 text-[12px] ${(
    club.isActive
    ? "border-red-400/50 bg-red-500/10 text-red-100 hover:border-red-300/70"
    : "border-emerald-400/50 bg-emerald-500/10 text-emerald-100 hover:border-emerald-300/70"
  )}`}
  >
  {club.isActive ? "Arquivar" : "Reativar"}
</button>
</div>
</div>
);
)}
</div>
)

{drawerClubId && selectedClub &&


<div className="flex flex-wrap items-center justify-between gap-2">
<div>
<p className="text-[12px] uppercase tracking-[0.18em] text-white/60">Courts & equipa</p>
<p className="text-sm text-white/70">Courts ativos e staff herdável vão para o wizard de torneio.</p>
</div>
<div className="flex items-center gap-2">
<span className={badge("slate")}>{selectedClub.name}</span>
<button
  type="button"
  onClick={() => setDrawerClubId(null)}
  className="rounded-full border border-white/15 px-3 py-1 text-white hover:border-white/30"
>
  Fechar
</button>
</div>
</div>
</div>

{loadingDrawer && (
<div className="space-y-3">
<div className="h-4 w-32 rounded bg-white/10 animate-pulse" />
<div className="grid gap-3 lg:grid-cols-2">
{[...Array(2)].map((_, idx) => (
<div key={idx} className="space-y-2 rounded-xl border border-white/10 bg-black/25 p-3 animate-pulse">
<div className="h-4 w-1/2 rounded bg-white/10" />
<div className="h-10 rounded bg-white/5" />
<div className="h-10 rounded bg-white/5" />
<div className="h-3 w-24 rounded bg-white/10" />
</div>
)))
</div>
</div>
)}
</div>
</div>
)

<div className="grid gap-4 lg:grid-cols-2">
<div className="space-y-3 rounded-xl border border-white/10 bg-black/30 p-3">
<div className="flex items-center justify-between">
<p className="text-sm font-semibold text-white">Courts do clube</p>
<span className={badge("slate")}>{courts.filter((c) => c.isActive).length} ativos</span>
</div>
<div className="grid gap-2 sm:grid-cols-2">
<input
  value={courtForm.name}
  onChange={(e) => setCourtForm((p) => ({ ...p, name: e.target.value }))}


```

```

        className="rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]""
            placeholder="Nome do court"
        />
<input
    value={courtForm.description}
    onChange={(e) => setCourtForm((p) => ({ ...p, description: e.target.value }))}
    className="rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]""
            placeholder="Descrição / patrocinador (opcional)"
        />
<div className="col-span-2 flex flex-wrap items-center gap-2 text-sm text-white/80">
    <span className="text-[12px] uppercase tracking-[0.2em] text-white/60">Tipo</span>
    <div className="inline-flex rounded-full border border-white/15 bg-black/40 p-1 text-[12px]">
        [
            { key: false, label: "Outdoor" },
            { key: true, label: "Indoor" },
        ].map((opt) => (
            <button
                key={String(opt.key)}
                type="button"
                onClick={() => setCourtForm((p) => ({ ...p, indoor: opt.key as boolean }))}
                className={`rounded-full px-3 py-1 transition ${
                    courtForm.indoor === opt.key
                        ? "bg-white text-black font-semibold shadow"
                        : "text-white/75 hover:bg-white/5"
                }`}
            >
                {opt.label}
            </button>
        )))
    </div>
    <div className="inline-flex rounded-full border border-white/15 bg-black/40 p-1 text-[12px]">
        [
            { key: true, label: "Ativo" },
            { key: false, label: "Inativo" },
        ].map((opt) => (
            <button
                key={String(opt.key)}
                type="button"
                onClick={() => setCourtForm((p) => ({ ...p, isActive: opt.key as boolean }))}
                className={`rounded-full px-3 py-1 transition ${
                    courtForm.isActive === opt.key
                        ? opt.key
                        ? "bg-emerald-400 text-black font-semibold"
                        : "bg-white text-black font-semibold"
                        : "text-white/75 hover:bg-white/5"
                }`}
            >
                {opt.label}
            </button>
        )))
    </div>
</div>
<div className="flex items-center justify-between text-[12px] text-white/60">
    <span>Arrasta os courts para ordenar (ordem automática).</span>
    <span className="rounded-full border border-white/15 px-2 py-[2px]">1º é o mais usado</span>
</div>
</div>
<div className="flex flex-wrap gap-2">
    <button
        type="button"
        onClick={handleSubmitCourt}
        disabled={savingCourt}
        className="rounded-full bg-white px-3 py-1.5 text-[12px] font-semibold text-black shadow disabled:opacity-
60"
    >
        {savingCourt ? "A guardar..." : courtForm.id ? "Atualizar court" : "Guardar court"}
    </button>
    {courtForm.id && (
        <button
            type="button"
            onClick={resetCourtForm}
            className="rounded-full border border-white/20 px-3 py-1.5 text-[12px] text-white hover:border-white/35"
        >
            Cancelar
        </button>
    )}

```

```

    {(courtError || courtMessage) && (
      <span className="text-[12px] text-white/70">{courtError || courtMessage}</span>
    )}
  </div>
<div className="space-y-2 rounded-lg border border-white/10 bg-white/5 p-2 text-[12px] text-white/80">
  {courts.length === 0 && <p className="text-white/60">Sem courts ainda.</p>}
  {courts.map((c, idx) => (
    <div
      key={c.id}
      draggable
      onDragStart={() => setDraggingCourtId(c.id)}
      onDragOver={(e) => e.preventDefault()}
      onDrop={(e) => {
        e.preventDefault();
        const updated = reorderCourts(c.id);
        if (updated) {
          persistCourtOrder(updated);
        }
        setDraggingCourtId(null);
      }}
      onDragEnd={() => setDraggingCourtId(null)}
      className={`flex items-center justify-between gap-3 rounded-md px-3 py-2 transition ${c.isActive
        ? "border border-emerald-400/35 bg-emerald-500/5"
        : "border border-red-500/40 bg-red-500/8"
      } ${draggingCourtId === c.id ? "opacity-60" : "opacity-100"}`}
    >
      <div className="flex items-center gap-3">
        <div
          className={`flex h-10 w-10 items-center justify-center rounded-full border text-lg font-bold ${c.isActive
            ? "border-emerald-400/40 bg-emerald-500/10 text-emerald-50"
            : "border-red-400/40 bg-red-500/10 text-red-100"
          }`}
        >
          {idx + 1}
        </div>
        <div>
          <p className="text-sm font-semibold text-white">{c.name}</p>
          <p className={`text-[11px] ${c.isActive ? "text-emerald-100/80" : "text-red-100/80"}`}>
            {c.indoor ? "Indoor" : "Outdoor"} · Ordem {c.displayOrder} · {c.isActive ? "Ativo" : "Inativo"}
          </p>
        </div>
      </div>
      <div className="flex items-center gap-2">
        <button
          type="button"
          onClick={() => handleEditCourt(c)}
          className="rounded-full border border-white/15 px-2 py-1 text-[11px] text-white hover:border-white/30"
        >
          Editar
        </button>
        <button
          type="button"
          onClick={() => setCourtDialog({ court: c, nextActive: !c.isActive })}
          className="rounded-full border px-2 py-1 text-[11px] ${c.isActive
            ? "border-white/15 text-white/80 hover:border-white/30"
            : "border-red-400/50 bg-red-500/10 text-red-100 hover:border-red-300/70"
          }"
        >
          {c.isActive ? "Desativar" : "Ativar"}
        </button>
      </div>
    </div>
  )));
</div>
</div>

<div className="space-y-3 rounded-xl border border-white/10 bg-black/30 p-3">
  <div className="flex items-center justify-between">
    <p className="text-sm font-semibold text-white">Staff do clube</p>
    <span className={badge("slate")}>{staff.filter((s) => s.inheritToEvents).length} herdam</span>
  </div>
  <div className="space-y-2">
    <div className="inline-flex rounded-full border border-white/15 bg-black/40 p-1 text-[12px]">
      {[{"key": "existing", "label": "Usar staff do organizador"}, {"key": "new", "label": "Criar novo staff"}]}
    </div>
  </div>
</div>

```

```

        { key: "external", label: "Adicionar contato externo" },
    ].map((opt) => (
        <button
            key={opt.key}
            type="button"
            onClick={() => setStaffMode(opt.key as typeof staffMode)}
            className={`rounded-full px-3 py-1 transition ${(
                staffMode === opt.key ? "bg-white text-black font-semibold shadow" : "text-white/75 hover:bg-white/5"
            )}`}
        >
            {opt.label}
        </button>
    )))
</div>

{staffMode === "existing" ? (
    <div className="grid gap-2 sm:grid-cols-2">
        <input
            value={staffSearch}
            onChange={(e) => setStaffSearch(e.target.value)}
            className="rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF]" placeholder="Pesquisar membro (nome, email, username)"
        />
        <select
            value={staffForm.staffMemberId}
            onChange={(e) => setStaffForm((p) => ({ ...p, staffMemberId: e.target.value }))}
            className="rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF]">
            >
                <option value="">Escolhe membro</option>
                {staffOptions.map((m) => (
                    <option key={m.userId} value={m.userId}>
                        {(m.fullName || m.username || m.email || "Membro").trim()} {m.email ? `· ${m.email}` : ""}</option>
                )))
            </select>
        </div>
    ) : (
        <div className="grid gap-2 sm:grid-cols-2">
            <input
                value={staffForm.email}
                onChange={(e) => setStaffForm((p) => ({ ...p, email: e.target.value }))}
                className="rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF]" placeholder="Email do contato"
            />
            <div className="rounded-lg border border-white/15 bg-black/25 px-3 py-2 text-[12px] text-white/70">
                Sem conta ORYA: guardamos só email + role.
            </div>
        </div>
    )
}

<div className="grid gap-2 sm:grid-cols-2">
    <select
        value={staffForm.role}
        onChange={(e) => setStaffForm((p) => ({ ...p, role: e.target.value }))}
        className="rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF]">
        >
            <option value="ADMIN_CLUBE">Admin clube</option>
            <option value="DIRETOR_PROVA">Diretor / Árbitro</option>
            <option value="STAFF">Staff de campo</option>
        </select>
    <div className="inline-flex rounded-full border border-white/15 bg-black/40 p-1 text-[12px]">
        [
            { key: true, label: "Herdar para torneios" },
            { key: false, label: "Só neste clube" },
        ].map((opt) => (
            <button
                key={String(opt.key)}
                type="button"
                onClick={() => setStaffForm((p) => ({ ...p, inheritToEvents: opt.key as boolean }))}
                className={`rounded-full px-3 py-1 transition ${(
                    staffForm.inheritToEvents === opt.key
                    ? "bg-white text-black font-semibold shadow"
                    : "text-white/75 hover:bg-white/5"
                )}`}
            >
                {opt.label}
            </button>
        )))
    </div>
</div>

```

```

        `}`}
      >
      {opt.label}
    </button>
  )})
</div>
</div>
</div>
<div className="flex flex-wrap gap-2">
  <button
    type="button"
    onClick={handleSubmitStaff}
    className="rounded-full bg-white px-3 py-1.5 text-[12px] font-semibold text-black shadow"
  >
    {staffForm.id ? "Atualizar" : "Adicionar"}
  </button>
  {staffForm.id && (
    <button
      type="button"
      onClick={resetStaffForm}
      className="rounded-full border border-white/20 px-3 py-1.5 text-[12px] text-white hover:border-white/35"
    >
      Cancelar
    </button>
  )}
  {((staffError || staffMessage) && (
    <span className="text-[12px] text-white/70">{staffError || staffMessage}</span>
  ))}
</div>
<div className="space-y-2 rounded-lg border border-white/10 bg-white/5 p-2 text-[12px] text-white/80">
  {staff.length === 0 && <p className="text-white/60">Sem staff ainda.</p>}
  {staff.map((s) => (
    <div key={s.id} className="flex items-center justify-between rounded-md border border-white/10 bg-black/40
px-2 py-1.5">
      <div>
        <div>
          <p className="text-sm text-white">{s.email || s.userId || "Sem contacto"}</p>
          <p className="text-[11px] text-white/55">
            {s.role} · {s.inheritToEvents ? "Herdar para torneios" : "Só no clube"} ·{" "}
            {s.userId ? "Staff global" : "Externo"}
          </p>
        </div>
        <button
          type="button"
          onClick={() => handleEditStaff(s)}
          className="rounded-full border border-white/15 px-2 py-1 text-[11px] text-white hover:border-white/30"
        >
          Editar
        </button>
      </div>
    ))}
  </div>
</div>
</div>
</div>
)
</div>
}

{activeTab === "players" && (
  <div className="space-y-4 rounded-2xl border border-white/10 bg-white/[0.04] p-4">
    <div className="flex flex-wrap items-center justify-between gap-3">
      <div>
        <p className="text-[12px] uppercase tracking-[0.2em] text-white/60">Jogadores</p>
        <p className="text-sm text-white/70">Roster automático. Sem criação manual nesta fase.</p>
      </div>
      <input
        value={search}
        onChange={(e) => setSearch(e.target.value)}
        placeholder="Procurar por nome ou email"
        className="w-60 rounded-full border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]" />
    </div>
    <div className="overflow-auto rounded-xl border border-white/10">
      <table className="min-w-full text-left text-sm text-white/80">
        <thead className="bg-white/5 text-[12px] uppercase tracking-[0.12em] text-white/60">
          <tr>
            <th className="px-3 py-2">Jogador</th>

```

```

        <th className="px-3 py-2">Email</th>
        <th className="px-3 py-2">Telefone</th>
        <th className="px-3 py-2">Torneios</th>
    </tr>
</thead>
<tbody>
    {filteredPlayers.length === 0 &&
    <tr>
        <td className="px-3 py-3 text-[13px] text-white/60" colSpan={4}>
            Sem jogadores ainda. Quando houver inscrições em Padel, a lista aparece aqui.
        </td>
    </tr>
    )}
    {filteredPlayers.map((p) => (
        <tr key={p.id} className="border-t border-white/10">
            <td className="px-3 py-2 font-semibold text-white">
                <div>{p.fullName}</div>
                <p className="text-[11px] text-white/60">{p.level || "Nível não definido"}</p>
            </td>
            <td className="px-3 py-2">{p.email || "-"}</td>
            <td className="px-3 py-2">{p.phone || "-"}</td>
            <td className="px-3 py-2">
                <span className={badge("slate")}>{p.tournamentsCount ?? 0} torneios</span>
            </td>
        </tr>
    )))
    </tbody>
</table>
</div>
</div>
)}

{activeTab === "rankings" &&
<div className="rounded-2xl border border-white/10 bg-white/[0.04] p-4 text-sm text-white/75 space-y-2">
    <p className="text-[12px] uppercase tracking-[0.2em] text-white/60">Rankings</p>
    <p>Rankings multi-torneio chegam numa próxima versão.</p>
</div>
)}

{organizationKind !== "CLUBE_PADEL" &&
<div className="rounded-2xl border border-white/10 bg-white/[0.03] p-4 text-[12px] text-white/65">
    Ferramentas de Padel disponíveis mesmo sem seres clube. Usa quando precisares.
</div>
)}

{clubModalOpen &&
<div className="fixed inset-0 z-40 flex items-center justify-center bg-black/70 px-4">
    <div className="w-full max-w-xl rounded-2xl border border-white/10 bg-[#0c142b] p-6 shadow-[0_30px_80px_rgba(0,0,0,0.55)]">
        <div className="flex items-start justify-between gap-2">
            <div>
                <p className="text-[12px] uppercase tracking-[0.2em] text-white/60">
                    {clubForm.id ? "Editar clube" : "Novo clube"}
                </p>
                <h3 className="text-xl font-semibold text-white">Só o essencial.</h3>
            </div>
            <button
                type="button"
                onClick={() => setClubModalOpen(false)}
                className="rounded-full border border-white/20 px-3 py-1 text-[12px] text-white hover:border-white/35"
            >
                Fechar
            </button>
        </div>
        <div className="mt-4 space-y-3">
            <input
                value={clubForm.name}
                onChange={(e) => setClubForm((p) => ({ ...p, name: e.target.value }))}
                placeholder="Nome do clube"
                className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF]"
            />
            <div className="grid gap-3 sm:grid-cols-2">
                <div className="relative">
                    <input
                        list="pt-cities"
                        value={clubForm.city}

```

```

        onChange={(e) => setClubForm((p) => ({ ...p, city: e.target.value }))}

        placeholder="Cidade"
        className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]""
      />
      <datalist id="pt-cities">
        {PORTUGAL_CITIES.map((city) => (
          <option key={city} value={city} />
        )))
      </datalist>
    </div>
    <input
      value={clubForm.address}
      onChange={(e) => setClubForm((p) => ({ ...p, address: e.target.value }))}
      placeholder="Morada"
      className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]""
    />
  </div>
  <div className="grid gap-3 sm:grid-cols-2">
    <input
      value={(clubForm as any).slug ?? ""}
      onChange={(e) => setClubForm((p: any) => ({ ...p, slug: e.target.value }))}
      placeholder="Slug / código curto (opcional)"
      className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]""
    />
    <input
      type="number"
      min={1}
      max={1000}
      value={clubForm.courtsCount}
      onChange={(e) => setClubForm((p) => ({ ...p, courtsCount: e.target.value }))}
      placeholder="Nº de courts"
      className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]""
    />
  </div>
  <label className="flex items-center gap-2 text-sm text-white/80">
    <input
      type="checkbox"
      checked={clubForm.isActive}
      onChange={(e) => setClubForm((p) => ({ ...p, isActive: e.target.checked }))}
      className="h-4 w-4"
    />
    Ativo (disponível no wizard)
  </label>
  <div className="flex flex-wrap items-center gap-2 text-[12px] text-white/70">
    {clubError && <span className="text-red-300">{clubError}</span>}
    {clubMessage && <span>{clubMessage}</span>}
  </div>
  <div className="flex flex-wrap gap-2">
    <button
      type="button"
      onClick={handleSubmitClub}
      disabled={savingClub}
      className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow disabled:opacity-60"
    >
      {savingClub ? "A guardar..." : clubForm.id ? "Guardar alterações" : "Criar clube"}
    </button>
    <button
      type="button"
      onClick={() => setClubModalOpen(false)}
      className="rounded-full border border-white/20 px-3 py-2 text-[12px] text-white hover:border-white/35"
    >
      Cancelar
    </button>
  </div>
</div>
</div>
)}
```

{clubDialog && (

```

<ConfirmDestructiveActionDialog
  open
  title={clubDialog.nextActive ? "Reativar clube?" : "Arquivar clube?"}
  description={
```

```

        clubDialog.nextActive
        ? "O clube volta a aparecer no wizard e sugestões."
        : "O clube ficará inativo e deixa de aparecer nas sugestões do wizard."
    }
    consequences={
        clubDialog.nextActive
        ? ["Courts ativos continuam disponíveis."]
        : ["Não aparecerá ao criar torneios.", "Podes reativar mais tarde."]
    }
    confirmLabel={clubDialog.nextActive ? "Reativar" : "Arquivar"}
    dangerLevel={clubDialog.nextActive ? "medium" : "high"}
    onClose={() => setClubDialog(null)}
    onConfirm={() => handleToggleClubActive(clubDialog.club, clubDialog.nextActive)}
/>
)
}

{courtDialog && (
<ConfirmDestructiveActionDialog
    open
    title={courtDialog.nextActive ? "Reativar court?" : "Desativar court?"}
    description={
        courtDialog.nextActive
        ? "O court volta a ser sugerido no wizard."
        : "O court fica inativo e deixa de ser sugerido."
    }
    consequences={
        courtDialog.nextActive
        ? ["Mantém a ordem e atributos."]
        : ["Sai das sugestões do wizard.", "Podes reativar mais tarde."]
    }
    confirmLabel={courtDialog.nextActive ? "Reativar" : "Desativar"}
    dangerLevel={courtDialog.nextActive ? "medium" : "high"}
    onClose={() => setCourtDialog(null)}
    onConfirm={handleConfirmCourtToggle}
/>
)
)
</div>
);
}

```

app/organizador/(dashboard)/padel/page.tsx

```

export const runtime = "nodejs";

import { redirect } from "next/navigation";

// LEGACY – hub de Padel vive em /organizador?tab=padel
export default async function OrganizerPadelPage() {
  redirect("/organizador?tab=padel");
}

```

app/organizador/(dashboard)/padel/torneios/novo/page.tsx

```

"use client";

import { useEffect, useMemo, useState } from "react";
import { useRouter } from "next/navigation";

type OrganizerDefaults = {
  organizationKind: string | null;
  padelDefaults?: {
    ruleSetId?: number | null;
  } | null;
  displayName?: string | null;
  city?: string | null;
};

type PadelCategory = { id: number; name: string; level?: string | null };

type PadelClub = {
  id: number;
  name: string;
  shortName?: string | null;
  city?: string | null;
};

```

```

address?: string | null;
courtsCount?: number | null;
favoriteCategoryIds?: number[];
isActive: boolean;
};

type PadelClubCourt = {
  id: number;
  padelClubId: number;
  name: string;
  description: string | null;
  surface: string | null;
  indoor: boolean;
  isActive: boolean;
  displayOrder: number;
};

type TicketRow = {
  name: string;
  categoryId: number | null;
  price: string;
  capacity: string;
  registrationType: "TEAM" | "INDIVIDUAL";
};

function formatDateSuggestion() {
  const now = new Date();
  const nextSaturday = new Date(now);
  nextSaturday.setDate(now.getDate() + ((6 - now.getDay()) + 7) % 7 || 7);
  nextSaturday.setHours(10, 0, 0, 0);
  const start = nextSaturday.toISOString().slice(0, 16);
  const endDate = new Date(nextSaturday);
  endDate.setHours(18, 0, 0, 0);
  const end = endDate.toISOString().slice(0, 16);
  return { start, end };
}

export default function PadelWizardSimple() {
  const router = useRouter();
  const [step, setStep] = useState<1 | 2 | 3 | 4>(1);
  const [loadingDefaults, setLoadingDefaults] = useState(true);
  const [defaults, setDefaults] = useState<OrganizerDefaults | null>(null);
  const [categories, setCategories] = useState<PadelCategory[]>([]);
  const [clubs, setClubs] = useState<PadelClub[]>([]);
  const [selectedClubId, setSelectedClubId] = useState<number | null>(null);
  const [partnerClubIds, setPartnerClubIds] = useState<number[]>([]);
  const [clubCourts, setClubCourts] = useState<PadelClubCourt[]>([]);
  const [loadingCourts, setLoadingCourts] = useState(false);

  const [title, setTitle] = useState("");
  const [startsAt, setStartsAt] = useState("");
  const [endsAt, setEndsAt] = useState("");
  const [city, setCity] = useState("");
  const [clubName, setClubName] = useState("");
  const [address, setAddress] = useState("");
  const [courtsCount, setCourtsCount] = useState("");
  const [tournamentState, setTournamentState] = useState<"OCULTO" | "INSCRICOES" | "PUBLICO" | "TERMINADO" | "CANCELADO">("OCULTO");
  const [coverUrl, setCoverUrl] = useState<string | null>(null);
  const [uploadingCover, setUploadingCover] = useState(false);
  const [visibility, setVisibility] = useState<"PUBLIC" | "PRIVATE">("PUBLIC");
  const [listed, setListed] = useState(true);
  const [gameDuration, setGameDuration] = useState("60");
  const [allowCancelGames, setAllowCancelGames] = useState(false);
  const [advancedOpen, setAdvancedOpen] = useState(false);
  const [maxEntriesTotal, setMaxEntriesTotal] = useState("");
  const [waitlistEnabled, setWaitlistEnabled] = useState(true);
  const [allowSecondCategory, setAllowSecondCategory] = useState(false);

  const [tickets, setTickets] = useState<TicketRow[]>([
    { name: "", categoryId: null, price: "", capacity: "", registrationType: "TEAM" },
  ]);

  const [error, setError] = useState<string | null>(null);
  const [submitting, setSubmitting] = useState(false);

  const tournamentStateOptions: { value: typeof tournamentState; label: string; hint: string }[] = useMemo(
    () => [

```

```

    { value: "OCULTO", label: "Oculto", hint: "Rascunho; não aparece ao público." },
    { value: "INSCRICOES", label: "Inscrições", hint: "Página pública com inscrições abertas." },
    { value: "PUBLICO", label: "Público", hint: "Agenda/jogos em destaque, inscrições fechadas." },
    { value: "TERMINADO", label: "Terminado", hint: "Resultados finais publicados." },
    { value: "CANCELADO", label: "Cancelado", hint: "Visível como cancelado." },
  ],
  [],
),
);

const selectedClub = useMemo(
  () => clubs.find((c) => c.id === selectedClubId) || null,
  [clubs, selectedClubId],
);

const hasActiveClub = useMemo(() => clubs.some((c) => c.isActive), [clubs]);

const applyClubDefaults = (club?: Partial<PadelClub> | null) => {
  if (!club) return;
  setCity(club.city || "");
  setClubName(club.shortName || club.name || "");
  setAddress(club.address || "");
  setCourtsCount(club.courtsCount ? String(club.courtsCount) : "");
};

// Load organizer defaults + categories + clubes
useEffect(() => {
  let cancelled = false;
  async function load() {
    try {
      const [orgRes, catRes, clubRes] = await Promise.all([
        fetch("/api/organizador/me"),
        fetch("/api/padel/categories/my"),
        fetch("/api/padel/clubs"),
      ]);
      const [orgJson, catJson, clubJson] = await Promise.all([
        orgRes.json().catch(() => null),
        catRes.json().catch(() => null),
        clubRes.json().catch(() => null),
      ]);
      if (!cancelled) {
        if (orgRes.ok && orgJson?.organizer) {
          const org = orgJson.organizer as OrganizerDefaults;
          setDefaults(org);
          setCity(org.city || "");
          setClubName(org.displayName || "");
        }
        if (catRes.ok && Array.isArray(catJson?.items)) {
          setCategories(catJson.items as PadelCategory[]);
        }
        if (clubRes.ok && Array.isArray(clubJson?.items)) {
          const clubList = (clubJson.items as PadelClub[])
            .filter((c) => c.isActive)
            .setClubs(clubList);
          const activeClub = clubList.find((c) => c.isActive) || clubList[0];
          if (activeClub) {
            setSelectedClubId(activeClub.id);
            applyClubDefaults(activeClub);
          }
        }
      }
    } catch (err) {
      console.warn("[PadelWizard] defaults load failed", err);
    } finally {
      if (!cancelled) setLoadingDefaults(false);
    }
  }
  load();
  return () => {
    cancelled = true;
  };
}, []);

useEffect(() => {
  if (selectedClub) {
    applyClubDefaults(selectedClub);
    loadCourts(selectedClub.id);
  }
}, [selectedClub]);

```

```

useEffect(() => {
  if (selectedClubId) {
    setPartnerClubIds((prev) => prev.filter((id) => id !== selectedClubId));
  }
}, [selectedClubId]);

const loadCourts = async (clubId: number) => {
  setLoadingCourts(true);
  try {
    const res = await fetch(`~/api/padel/clubs/${clubId}/courts`);
    const json = await res.json().catch(() => null);
    if (res.ok && Array.isArray(json?.items)) {
      const courts = (json.items as PadelClubCourt[]).filter((c) => c.isActive);
      setClubCourts(courts);
      if (!courtsCount && courts.length > 0) {
        const activeCount = courts.filter((c) => c.isActive).length || courts.length;
        setCourtsCount(String(activeCount));
      }
    } else {
      setClubCourts([]);
    }
  } catch (err) {
    console.warn("[PadelWizard] load courts failed", err);
    setClubCourts([]);
  } finally {
    setLoadingCourts(false);
  }
};

const handleUploadCover = async (file: File | null) => {
  if (!file) return;
  setUploadingCover(true);
  setError(null);
  try {
    const formData = new FormData();
    formData.append("file", file);
    const uploadRes = await fetch("/api/upload", { method: "POST", body: formData });
    const uploadJson = await uploadRes.json().catch(() => null);
    if (!uploadRes.ok || !uploadJson?.url) throw new Error(uploadJson?.error || "Falha no upload da imagem.");
    setCoverUrl(uploadJson.url as string);
  } catch (err) {
    console.error("[PadelWizard] upload cover", err);
    setError("Não foi possível carregar a imagem de capa.");
  } finally {
    setUploadingCover(false);
  }
};

// Quick fill
const handleQuickFill = () => {
  const { start, end } = formatDateSuggestion();
  const clubLabel = selectedClub?.shortName || selectedClub?.name || defaults?.displayName || "Padel";
  setTitle((prev) => prev || `Open ${clubLabel}`);
  setStartsAt(start);
  setEndsAt(end);
  const favIds = selectedClub?.favoriteCategoryId || [];
  const suggestedCat =
    (favIds[0] && categories.find((c) => c.id === favIds[0])) || categories[0];
  setTickets([
    {
      name: suggestedCat ? `Dupla ${suggestedCat.name}` : "Dupla Masculina 4/5",
      categoryId: suggestedCat?.id ?? null,
      price: "40",
      capacity: "16",
      registrationType: "TEAM",
    },
  ]);
  applyClubDefaults(selectedClub);
};

const addTicket = () => {
  setTickets((prev) => [...prev, { name: "", categoryId: null, price: "", capacity: "", registrationType: "TEAM" }]);
};

const updateTicket = (idx: number, field: keyof TicketRow, value: string | number | null) => {
  setTickets((prev) =>
    prev.map((t, i) => (i === idx ? { ...t, [field]: value as any } : t)),
  );
};

```

```

const removeTicket = (idx: number) => {
  setTickets((prev) => prev.filter((_, i) => i !== idx));
};

const canNext = useMemo(() => {
  if (step === 1) return Boolean(title.trim() && startsAt && (city || selectedClub?.city) && selectedClubId);
  if (step === 2) return tickets.some((t) => t.name.trim() && t.price);
  return true;
}, [step, title, startsAt, city, selectedClub?.city, tickets, selectedClubId]);

const grossEstimate = useMemo(() => {
  return tickets.reduce((acc, t) => {
    const price = Number(t.price.replace(",", ".") || 0);
    const cap = Number(t.capacity) || 0;
    return acc + price * cap;
  }, 0);
}, [tickets]);

const tournamentStateLabel =
  tournamentStateOptions.find((opt) => opt.value === tournamentState)?.label || "-";

const handleSubmit = async () => {
  setSubmitting(true);
  setError(null);
  try {
    if (!selectedClubId)
      setError("Adciona um clube de Padel e seleciona-o para continuares.");
    setStep(1);
    return;
  }
  const defaultRuleSetId = defaults?.padelDefaults?.ruleSetId ?? null;
  const payload = {
    title: title.trim(),
    startsAt,
    endsAt: endsAt || startsAt,
    locationCity: (city || selectedClub?.city || "").trim(),
    locationName: clubName.trim() || selectedClub?.shortName || selectedClub?.name || "",
    address: (address || selectedClub?.address || "").trim() || null,
    coverImageUrl: coverUrl,
    templateType: "PADEL",
    categories: ["DESPORTO"],
    feeMode: "ON_TOP",
    publicListingEnabled: listed,
    visibility,
    padelClubId: selectedClubId,
    partnerClubIds,
    courtsCount: courtsCount ? Number(courtsCount) : selectedClub?.courtsCount,
    tournamentState,
    advancedSettings: {
      maxEntriesTotal: maxEntriesTotal ? Number(maxEntriesTotal) : null,
      waitlistEnabled,
      allowSecondCategory,
      allowCancelGames,
      gameDurationMinutes: gameDuration ? Number(gameDuration) : null,
      categoriesMeta: tickets.map((t) => ({
        name: t.name.trim() || "Categoria",
        categoryId: t.categoryId ?? null,
        registrationType: t.registrationType ?? "TEAM",
        capacity: t.capacity ? Number(t.capacity) : null,
      })),
    },
    ticketTypes: tickets
      .filter((t) => t.name.trim())
      .map((t) => ({
        name: t.name.trim(),
        price: Number(t.price.replace(",", ".") || 0),
        totalQuantity: t.capacity ? Number(t.capacity) : null,
      })),
    padel: {
      padelV2Enabled: true,
      ruleSetId: defaultRuleSetId,
    },
  };
  const res = await fetch("/api/organizador/padel/tournaments/create", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(payload),
  });
};

```

```

    });
    const json = await res.json().catch(() => null);
    if (!res.ok || !json?.ok) {
      throw new Error(json?.error || "Erro ao criar torneio.");
    }
    if (json.event?.id) router.push('/organizador/eventos/${json.event.id}');
    else if (json.event?.slug) router.push('/eventos/${json.event.slug}');
    else router.push("/organizador/eventos");
  } catch (err) {
    console.error(err);
    setError(err instanceof Error ? err.message : "Erro ao criar torneio.");
  } finally {
    setSubmitting(false);
  }
};

const formDisabled = !hasActiveClub && !loadingDefaults;

return (
  <div className="mx-auto flex max-w-6xl flex-col gap-6 px-4 py-8 text-white md:flex-row md:items-start md:px-8">
    <div className="flex-1 space-y-4">
      <header className="space-y-1">
        <p className="text-[11px] uppercase tracking-[0.26em] text-white/55">Padel · Criar torneio</p>
        <div className="flex flex-wrap items-center gap-2">
          <h1 className="text-2xl font-semibold tracking-tight">Wizard rápido para clubes</h1>
          <button
            type="button"
            onClick={handleQuickFill}
            disabled={formDisabled}
            className="rounded-full border border-white/20 px-3 py-1 text-[12px] text-white/80 hover:bg-white/10
disabled:opacity-40"
          >
            Criar torneio rápido
          </button>
        </div>
        <p className="text-sm text-white/65">3 passos: Básico → Categorias → Rever & criar. FULL/SPLIT geridos pela
plataforma.</p>
      </header>
      {!hasActiveClub && !loadingDefaults && (
        <div className="rounded-2xl border border-amber-400/50 bg-amber-500/10 px-4 py-3 text-sm text-amber-100">
          <p className="font-semibold">Ainda não tens nenhum clube de Padel configurado.</p>
          <p className="text-amber-50/80">
            Vai a <a href="/organizador/padel" className="underline">Padel → Clubes</a> e adiciona pelo menos um clube para
continuares.
          </p>
        </div>
      )}
    <div className="flex items-center gap-2 text-[12px]">
      {[1, 2, 3, 4].map((s) => (
        <button
          key={s}
          type="button"
          onClick={() => setStep(s as 1 | 2 | 3 | 4)}
          className={` rounded-full px-3 py-1 border border-${step === s ? "bg-white text-black font-semibold" : "bg-black/40 text-white/70`}
`}
        >
          {s === 1
            ? "1. Básico"
            : s === 2
            ? "2. Categorias"
            : s === 3
            ? "3. Jogos & courts"
            : "4. Rever"
          }
        </button>
      ))}
    </div>
    {error && (
      <div className="rounded-md border border-red-500/40 bg-red-500/10 px-4 py-3 text-sm text-red-100">
        {error}
      </div>
    )}
  <div step === 1 && (
    <div className="space-y-3 rounded-2xl border border-white/10 bg-black/35 p-4 shadow-[0_20px_60px_rgba(0,0,0,0.55)]">

```

```


<label className="text-sm">Clube *</label>
  <select
    value={selectedClubId ?? ""}
    onChange={(e) => setSelectedClubId(e.target.value ? Number(e.target.value) : null)}
    disabled={formDisabled}
    className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50">
    >
      <option value="">Seleciona o clube</option>
      {clubs
        .filter((c) => c.isActive)
        .map((club) => (
          <option key={club.id} value={club.id}>
            {club.name} {club.city ? ` ${club.city}` : ""}</option>
        )))
      </select>
    <p className="text-[1px] text-white/60">
      Só clubes ativos são apresentados. Gerir em Padel → Clubes.
    </p>
  </div>
{clubs.filter((c) => c.isActive && c.id !== selectedClubId).length > 0 && (
  <div className="space-y-1">
    <label className="text-sm">Clubes parceiros (opcional)</label>
    <div className="flex flex-wrap gap-2">
      {clubs
        .filter((c) => c.isActive && c.id !== selectedClubId)
        .map((club) => {
          const checked = partnerClubIds.includes(club.id);
          return (
            <label
              key={club.id}
              className="flex cursor-pointer items-center gap-2 rounded-full border px-3 py-1 text-[12px] ${
                checked ? "border-white bg-white text-black" : "border-white/20 text-white/75"
              }">
              >
                <input
                  type="checkbox"
                  className="hidden"
                  checked={checked}
                  onChange={(e) => {
                    if (e.target.checked) {
                      setPartnerClubIds((prev) => [...prev, club.id]);
                    } else {
                      setPartnerClubIds((prev) => prev.filter((id) => id !== club.id));
                    }
                  }}
                  disabled={formDisabled}
                />
                {club.name} {club.city ? ` ${club.city}` : ""}
              </label>
            );
          ))}
        </div>
      <p className="text-[1px] text-white/55">
        Outside organizers podem usar vários clubes. Seleciona os parceiros para este torneio.
      </p>
    </div>
  )}

<div className="space-y-1">
  <label className="text-sm">Imagem de capa</label>
  <div className="flex flex-wrap items-center gap-3">
    <div className="h-24 w-36 overflow-hidden rounded-xl border border-white/15 bg-black/30 text-[11px] text-white/60 flex items-center justify-center">
      {coverUrl ? (
        // eslint-disable-next-line @next/next/no-img-element
        <img src={coverUrl} alt="Capa" className="h-full w-full object-cover" />
      ) : (
        "Sem imagem"
      )}
    </div>
    <label className="inline-flex cursor-pointer items-center gap-2 rounded-full border border-white/20 px-3 py-1 text-[12px] text-white/80 hover:bg-white/10">
      <span>{coverUrl ? "Substituir" : "Carregar imagem"}</span>
      <input
        type="file"
      >
    </label>
  </div>


```

```

        className="hidden"
        accept="image/*"
        onChange={(e) => handleUploadCover(e.target.files?.[0] ?? null)}
        disabled={uploadingCover || formDisabled}
      />
    </label>
    {coverUrl && (
      <button type="button" onClick={() => setCoverUrl(null)} className="text-[12px] text-white/70 underline">
        Remover
      </button>
    )}
    {uploadingCover && <span className="text-[11px] text-white/60">A carregar...</span>}
  </div>
</div>

<div className="space-y-1">
  <label className="text-sm">Título *</label>
  <input
    value={title}
    onChange={(e) => setTitle(e.target.value)}
    disabled={formDisabled}
    className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50"
    placeholder="Open do Clube"
  />
</div>
<div className="grid gap-3 md:grid-cols-2">
  <label className="space-y-1 text-sm">
    Início *
    <input
      type="datetime-local"
      value={startsAt}
      onChange={(e) => setStartsAt(e.target.value)}
      disabled={formDisabled}
      className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50"
    />
  </label>
  <label className="space-y-1 text-sm">
    Fim
    <input
      type="datetime-local"
      value={endsAt}
      onChange={(e) => setEndsAt(e.target.value)}
      disabled={formDisabled}
      className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50"
    />
  </label>
</div>
<div className="grid gap-3 md:grid-cols-2">
  <label className="space-y-1 text-sm">
    Nome curto do evento
    <input
      value={clubName}
      onChange={(e) => setClubName(e.target.value)}
      disabled={formDisabled}
      className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50"
      placeholder="Clube XPTO"
    />
  </label>
  <label className="space-y-1 text-sm">
    Cidade *
    <input
      value={city}
      onChange={(e) => setCity(e.target.value)}
      disabled={formDisabled}
      className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50"
      placeholder="Porto, Lisboa..."
    />
  </label>
</div>
<div className="grid gap-3 md:grid-cols-2">
  <label className="space-y-1 text-sm">
    Morada
    <input

```

```

        value={address}
        onChange={(e) => setAddress(e.target.value)}
        disabled={formDisabled}
        className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none
disabled:opacity-50"
        placeholder="Rua e número"
      />
    </label>
  </div>
<div className="space-y-2">
  <p className="text-sm font-medium text-white">Estado inicial do torneio</p>
  <div className="grid grid-cols-1 gap-2 sm:grid-cols-2">
    {tournamentStateOptions.map((opt) =>
      <button
        key={opt.value}
        type="button"
        onClick={() => setTournamentState(opt.value)}
        className={`${flex} flex-col items-start rounded-xl border px-3 py-2 text-left text-[12px] transition ${
          tournamentState === opt.value
            ? "border-white bg-white text-black shadow"
            : "border-white/15 bg-black/30 text-white/75 hover:border-white/40"
        }`}
        disabled={formDisabled}
      >
        <span className="text-sm font-semibold">{opt.label}</span>
        <span className="text-[11px] text-white/60">{opt.hint}</span>
      </button>
    )))
  </div>
</div>
<div className="grid gap-3 md:grid-cols-2">
  <label className="space-y-1 text-sm">
    Visibilidade
    <select
      value={visibility}
      onChange={(e) => setVisibility(e.target.value as "PUBLIC" | "PRIVATE")}
      disabled={formDisabled}
      className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none
disabled:opacity-50"
    >
      <option value="PUBLIC">Público</option>
      <option value="PRIVATE">Privado</option>
    </select>
  </label>
  <label className="space-y-1 text-sm">
    Listagem
    <select
      value={listed ? "yes" : "no"}
      onChange={(e) => setListed(e.target.value === "yes")}
      disabled={formDisabled}
      className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none
disabled:opacity-50"
    >
      <option value="yes">Listar na página da organização</option>
      <option value="no">Não listar</option>
    </select>
  </label>
</div>
</div>
)}

{step === 2 && (
  <div className="space-y-3 rounded-2xl border border-white/10 bg-black/35 p-4 shadow-[0_20px_60px_rgba(0,0,0,0.55)]">
    <div className="flex items-center justify-between">
      <div>
        <h2 className="text-sm font-semibold">Categorias & preço</h2>
        <p className="text-[11px] text-white/60">Preço por dupla (a plataforma divide por jogador automaticamente).</p>
      </div>
      <button
        type="button"
        onClick={addTicket}
        disabled={formDisabled}
        className="rounded-full border border-white/20 px-3 py-1 text-[12px] text-white/80 hover:bg-white/10
disabled:opacity-40"
      >
        + Adicionar categoria
      </button>
    </div>
  </div>
)
}

```

```

{tickets.map((t, idx) => (
  <div key={idx} className="space-y-2 rounded-xl border border-white/10 bg-black/30 p-3">
    <div className="flex items-center gap-2">
      <input
        value={t.name}
        onChange={(e) => updateTicket(idx, "name", e.target.value)}
        placeholder="Nome (ex.: Dupla Masc 4/5)"
        disabled={formDisabled}
        className="flex-1 rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none"
        disabled:opacity-50"
      />
      <button onClick={() => removeTicket(idx)} type="button" className="text-[11px] text-red-300">
        Remover
      </button>
    </div>
    <div className="grid gap-2 md:grid-cols-4">
      <select
        value={t.categoryId ?? ""}
        onChange={(e) => updateTicket(idx, "categoryId", e.target.value ? Number(e.target.value) : null)}
        disabled={formDisabled}
        className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50"
      >
        <option value="">Categoria Padel</option>
        {categories.map((c) => (
          <option key={c.id} value={c.id}>
            {c.name}
            {c.level ? ` · ${c.level}` : ""}
          </option>
        )))
      </select>
      <select
        value={t.registrationType}
        onChange={(e) => updateTicket(idx, "registrationType", e.target.value as "TEAM" | "INDIVIDUAL")}
        disabled={formDisabled}
        className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50"
      >
        <option value="TEAM">Equipa / Dupla</option>
        <option value="INDIVIDUAL">Individual</option>
      </select>
      <input
        type="number"
        min={0}
        value={t.price}
        onChange={(e) => updateTicket(idx, "price", e.target.value)}
        placeholder="Preço por dupla (€)"
        disabled={formDisabled}
        className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50"
      />
      <input
        type="number"
        min={0}
        value={t.capacity}
        onChange={(e) => updateTicket(idx, "capacity", e.target.value)}
        placeholder="Nº de duplas"
        disabled={formDisabled}
        className="rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none disabled:opacity-50"
      />
    </div>
  </div>
))
<p className="text-[11px] text-white/55">
  Vagas = nº máximo de duplas nesta categoria. FULL + SPLIT e convites são geridos pela plataforma.
</p>

<div className="rounded-xl border border-white/10 bg-black/25 p-3">
  <button
    type="button"
    onClick={() => setAdvancedOpen((v) => !v)}
    className="flex w-full items-center justify-between text-sm font-semibold"
  >
    Opções avançadas
    <span className="text-[11px] text-white/60">{advancedOpen ? "Esconder" : "Mostrar"}</span>
  </button>
  {advancedOpen && (
    <div className="mt-3 grid gap-3 md:grid-cols-3">
      <label className="space-y-1 text-sm">
        Limite total de inscrições
        <input

```

```

        type="number"
        min={0}
        value={maxEntriesTotal}
        onChange={(e) => setMaxEntriesTotal(e.target.value)}
        className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none"
        placeholder="ex.: 64"
      />
      <span className="text-[11px] text-white/55">Opcional. Bloqueia entradas extra.</span>
    </label>
    <label className="flex items-center gap-2 rounded-lg border border-white/15 bg-black/25 px-3 py-2 text-sm">
      <input
        type="checkbox"
        checked={waitlistEnabled}
        onChange={(e) => setWaitlistEnabled(e.target.checked)}
        className="accent-white"
      />
      Lista de espera quando lotado
    </label>
    <label className="flex items-center gap-2 rounded-lg border border-white/15 bg-black/25 px-3 py-2 text-sm">
      <input
        type="checkbox"
        checked={allowSecondCategory}
        onChange={(e) => setAllowSecondCategory(e.target.checked)}
        className="accent-white"
      />
      Permitir 2ª categoria
    </label>
  </div>
)
</div>
</div>
)}

{step === 3 && (
  <div className="space-y-3 rounded-2xl border border-white/10 bg-black/35 p-4 shadow-[0_20px_60px_rgba(0,0,0,0.55)]">
    <h2 className="text-sm font-semibold">Jogos & courts</h2>
    <div className="grid gap-3 md:grid-cols-2">
      <label className="space-y-1 text-sm">
        Nº de courts a usar
        <input
          type="number"
          min={1}
          max={1000}
          value={courtsCount}
          onChange={(e) => setCourtsCount(e.target.value)}
          disabled={formDisabled}
          className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none"
          disabled:opacity-50 [appearance:textfield]"
          placeholder={clubCourts.length ? String(clubCourts.filter((c) => c.isActive).length || clubCourts.length) : "4"}
        />
        <p className="text-[11px] text-white/60">Sugestão: {clubCourts.filter((c) => c.isActive).length || clubCourts.length} courts ativos neste clube.</p>
      </label>
      <label className="space-y-1 text-sm">
        Duração padrão de jogo (min)
        <input
          type="number"
          min={15}
          max={240}
          value={gameDuration}
          onChange={(e) => setGameDuration(e.target.value)}
          className="w-full rounded-lg border border-white/15 bg-black/30 px-3 py-2 text-sm outline-none"
          placeholder="60"
        />
      </label>
    </div>
    <label className="flex items-center gap-2 rounded-xl border border-white/15 bg-black/25 px-3 py-2 text-sm">
      <input
        type="checkbox"
        checked={allowCancelGames}
        onChange={(e) => setAllowCancelGames(e.target.checked)}
        className="accent-white"
      />
      Permitir cancelamento de jogos na grelha
    </label>
  </div>
  {selectedClub && (

```

```

<div className="space-y-2 rounded-xl border border-white/10 bg-black/30 p-3">
  <div className="flex items-center justify-between text-sm">
    <span className="font-semibold">Courts do clube</span>
    {loadingCourts && <span className="text-[11px] text-white/60">A carregar...</span>}
  </div>
  {clubCourts.length === 0 && !loadingCourts && (
    <p className="text-[12px] text-white/65">Ainda não adicionaste courts a este clube.</p>
  )}
  {clubCourts.length > 0 && (
    <div className="flex flex-wrap gap-2">
      {clubCourts.map((court) => (
        <span
          key={court.id}
          className={`inline-flex items-center gap-1 rounded-full border px-3 py-1 text-[12px] ${court.isActive ? "border-emerald-400/50 bg-emerald-400/10 text-emerald-50" : "border-white/15 bg-black/30 text-white/70"}`}
        >
          {court.name}
          {court.indoor && <span className="text-[10px] uppercase tracking-[0.14em]">Indoor</span>}
        </span>
      )))
    </div>
  )}
  <p className="text-[11px] text-white/55">
    Podes gerir courts em Padel → Clubes. O nº de courts acima é usado para gerar o calendário inicial.
  </p>
</div>
)
</div>
}

{step === 4 && (
  <div className="space-y-3 rounded-2xl border border-white/10 bg-black/35 p-4 shadow-[0_20px_60px_rgba(0,0,0,0.55)]">
    <h2 className="text-sm font-semibold uppercase tracking-[0.2em] text-white/70">Revisão</h2>
    <div className="space-y-2 text-sm text-white/80">
      <p><strong>Título:</strong> {title || "-"}</p>
      <p><strong>Quando:</strong> {startsAt || "-"} → {endsAt || "-"}</p>
      <p><strong>Onde:</strong> {clubName || selectedClub?.shortName || selectedClub?.name || "-"}, {city || selectedClub?.city || "-"}</p>
      <p><strong>Clubes parceiros:</strong> {partnerClubIds.length} selecionado(s) : "-"</p>
    </p>
    <p><strong>Listagem:</strong> {visibility === "PUBLIC" ? "Público" : "Privado"} · {listed ? "Listado" : "Não listado"}</p>
    <p><strong>Estado inicial:</strong> {tournamentStatusLabel}</p>
    <p><strong>Categorias:</strong> {tickets.length} linha(s)</p>
    <p><strong>Estimativa bruta:</strong> {grossEstimate.toFixed(2)} €</p>
    <p className="text-[11px] text-white/60">Courts: {courtsCount || selectedClub?.courtsCount || "-"} · Jogo: {gameDuration || "-"} min</p>
    <p className="text-[11px] text-white/60">
      Avançadas: limite total {maxEntriesTotal || "-"} · Waitlist {waitlistEnabled ? "ativa" : "off"} · 2ª categoria {allowSecondCategory ? "permitida" : "não"} · Cancelar jogos {allowCancelGames ? "sim" : "não"}.
    </p>
  </div>
)
</div>
)

<div className="flex flex-wrap gap-2">
  <button
    type="button"
    onClick={() => setStep((s) => (s > 1 ? ((s - 1) as 1 | 2 | 3 | 4) : s))}
    className="rounded-full border border-white/20 px-4 py-2 text-sm text-white/80 hover:bg-white/10"
  >
    Anterior
  </button>
  {step < 4 ? (
    <button
      type="button"
      disabled={!canNext || formDisabled}
      onClick={() => setStep((s) => (s < 4 ? ((s + 1) as 1 | 2 | 3 | 4) : s))}
      className="rounded-full border border-white/20 px-4 py-2 text-sm text-white hover:bg-white/10 disabled:opacity-40"
    >
      Seguinte
    </button>
  ) : (
    <button
      type="button"
      disabled={submitting || formDisabled}

```

```

        onClick={handleSubmit}
        className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-5 py-2 text-sm font-semibold
text-black shadow hover:scale-[1.01] disabled:opacity-60"
      >
    {submitting ? "A criar..." : "Criar torneio de Padel"}
  </button>
)
</div>
</div>

/* Preview */
<aside className="sticky top-6 w-full max-w-sm rounded-3xl border border-white/10 bg-white/[0.04] p-4 shadow-[0_24px_60px_rgba(0,0,0.55)] md:w-80">
  <p className="text-[11px] uppercase tracking-[0.24em] text-white/60">Pré-visualização</p>
  <div className="mt-3 space-y-3">
    <div className="h-36 w-full overflow-hidden rounded-2xl border border-white/10 bg-black/30 text-[11px] text-white/60
flex items-center justify-center">
      {coverUrl ? (
        // eslint-disable-next-line @next/next/no-img-element
        <img src={coverUrl} alt="Capa" className="h-full w-full object-cover" />
      ) : (
        "Sem imagem"
      )}
    </div>
    <div>
      <h3 className="text-lg font-semibold">{title || "Torneio sem título"}</h3>
      <p className="text-sm text-white/65">
        {(city || selectedClub?.city || "Cidade")} · {startsAt || "Data a definir"}
      </p>
    </div>
    <div className="space-y-1 rounded-2xl border border-white/10 bg-black/25 p-3 text-sm">
      <p className="text-[11px] uppercase tracking-[0.14em] text-white/60">Categorias & preço</p>
      {tickets.map((t, i) => (
        <div key={`${t.name}-${i}`} className="flex items-center justify-between text-white/80">
          <span>{t.name || "Sem nome"}</span>
          <span className="text-white">{t.price ? `${Number(t.price || 0).toFixed(2)} €` : "-"}</span>
        </div>
      )));
      {tickets.length === 0 && <p className="text-[12px] text-white/55">Adiciona pelo menos uma categoria.</p>}
    </div>
    <div className="rounded-2xl border border-white/10 bg-black/30 p-3 text-sm">
      <div className="flex items-center justify-between text-white/80">
        <span>Receita bruta estimada:</span>
        <span className="text-base font-semibold">{grossEstimate.toFixed(2)} €</span>
      </div>
      <p className="text-[11px] text-white/55">Preço x vagas. FULL/SPLIT e fees tratados pela plataforma.</p>
    </div>
  </div>
</aside>
</div>
);
}

```

app/organizador/(dashboard)/page.tsx

```

import { redirect } from "next/navigation";
import { OrganizerMemberRole } from "@prisma/client";
import { prisma } from "@lib/prisma";
import { createSupabaseServer } from "@lib/supabaseServer";
import { getActiveOrganizerForUser } from "@lib/organizerContext";
import DashboardClient from "../DashboardClient";
import { OrganizerTour } from "../OrganizerTour";
import { cookies } from "next/headers";

export const runtime = "nodejs";

/**
 * Router inteligente do /organizador.
 * Decide o destino com base no estado do utilizador e organizações.
 * Quando há organização ativa, renderiza o dashboard (overview como tab default no client).
 */
export default async function OrganizerRouterPage() {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

```

```

if (!user) {
  redirect("/login?next=/organizador");
}

// 1) Contar memberships; se tabela não existir em dev, assume 0
let membershipCount = 0;
try {
  membershipCount = await prisma.organizerMember.count({ where: { userId: user.id } });
} catch (err: unknown) {
  const msg =
    typeof err === "object" && err && "message" in err ? String((err as { message?: unknown }).message) : "";
  if (!(msg.includes("does not exist") || msg.includes("organizer_members"))) {
    throw err;
  }
}

// Backfill automático para contas antigas (organizer.userId)
if (membershipCount === 0) {
  try {
    const legacyOrganizers = await prisma.organizer.findMany({
      where: { userId: user.id, status: "ACTIVE" },
      select: { id: true },
    });
    if (legacyOrganizers.length > 0) {
      await prisma.organizerMember.createMany({
        data: legacyOrganizers.map((org) => ({
          organizerId: org.id,
          userId: user.id,
          role: OrganizerMemberRole.OWNER,
        })),
        skipDuplicates: true,
      });
      membershipCount = legacyOrganizers.length;
    }
  } catch (err: unknown) {
    const msg =
      typeof err === "object" && err && "message" in err ? String((err as { message?: unknown }).message) : "";
    if !(msg.includes("does not exist") || msg.includes("organizer_members"))) {
      throw err;
    }
  }
}

// Sem organizações → onboarding
if (membershipCount === 0) {
  redirect("/organizador/become");
}

// 2) Existe organização ativa?
let activeOrganizerId: number | null = null;
try {
  const cookieStore = await cookies();
  const cookieOrgId = cookieStore.get("orya_org")?.value;
  const forcedOrgId = cookieOrgId ? Number(cookieOrgId) : undefined;
  const { organizer } = await getActiveOrganizerForUser(user.id, {
    organizerId: Number.isFinite(forcedOrgId) ? forcedOrgId : undefined,
  });
  activeOrganizerId = organizer?.id ?? null;
} catch (err: unknown) {
  const msg =
    typeof err === "object" && err && "message" in err ? String((err as { message?: unknown }).message) : "";
  if !(msg.includes("does not exist") || msg.includes("organizer_members"))) {
    throw err;
  }
}

// Tem orgs mas nenhuma ativa → hub
if (!activeOrganizerId) {
  redirect("/organizador/organizations");
}

// Tem org ativa → dashboard
return (
  <>
  <DashboardClient />
  <OrganizerTour />
</>

```

```
 );
}
```

app/organizador/(dashboard)/settings/page.tsx

```
"use client";

import Link from "next/link";
import { useEffect, useMemo, useState } from "react";
import useSWR from "swr";
import { useRouter } from "next/navigation";
import { useUser } from "@/app/hooks/useUser";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";
import { isValidPhone, sanitizePhone } from "@lib/phone";
import { PORTUGAL_CITIES } from "@config/cities";
import { ConfirmDestructiveActionDialog } from "@/app/components/ConfirmDestructiveActionDialog";

type OrganizerMeResponse = {
  ok: boolean;
  organizer: {
    id: number;
    displayName: string | null;
    publicName?: string | null;
    username?: string | null;
    businessName: string | null;
    entityType: string | null;
    city: string | null;
    address?: string | null;
    showAddressPublicly?: boolean | null;
    payoutIban: string | null;
    language?: string | null;
    publicListingEnabled?: boolean | null;
    alertsEmail?: string | null;
    alertsSalesEnabled?: boolean | null;
    alertsPayoutEnabled?: boolean | null;
    brandingAvatarUrl?: string | null;
    brandingPrimaryColor?: string | null;
    brandingSecondaryColor?: string | null;
    organizationKind?: string | null;
    officialEmail?: string | null;
    officialEmailVerifiedAt?: string | null;
  } | null;
  profile: {
    fullName: string | null;
    city: string | null;
    contactPhone?: string | null;
  } | null;
  contactEmail?: string | null;
  membershipRole?: string | null;
};

const fetcher = (url: string) => fetch(url).then((r) => r.json());

export default function OrganizerSettingsPage() {
  const router = useRouter();
  const { user } = useUser();
  const { openModal } = useAuthModal();
  const { data, isLoading, mutate } = useSWR<OrganizerMeResponse>(
    user ? "/api/organizador/me" : null,
    fetcher,
    {
      revalidateOnFocus: false,
    },
  );

  const organizer = data?.organizer ?? null;
  const profile = data?.profile ?? null;
  const contactEmailFromAccount = data?.contactEmail ?? null;

  const [organizationKind, setOrganizationKind] = useState("PESSOA_SINGULAR");
  const [entityName, setEntityName] = useState("");
  const [publicName, setPublicName] = useState("");
  const [city, setCity] = useState("");
  const [address, setAddress] = useState("");
  const [showAddressPublicly, setShowAddressPublicly] = useState(false);
  const [contactEmail, setContactEmail] = useState("");
}
```

```

const [contactPhone, setContactPhone] = useState("");
const [phoneError, setPhoneError] = useState<string | null>(null);
const [orgMessage, setOrgMessage] = useState<string | null>(null);
const [savingOrg, setSavingOrg] = useState(false);
const [officialEmail, setOfficialEmail] = useState("");
const [officialEmailMessage, setOfficialEmailMessage] = useState<string | null>(null);
const [officialEmailSaving, setOfficialEmailSaving] = useState(false);

const [brandingAvatarUrl, setBrandingAvatarUrl] = useState("");
const [brandingPrimaryColor, setBrandingPrimaryColor] = useState("");
const [brandingSecondaryColor, setBrandingSecondaryColor] = useState("");
const [brandingUploading, setBrandingUploading] = useState(false);
const [brandingMessage, setBrandingMessage] = useState<string | null>(null);

const [username, setUsername] = useState("");
const [usernameMessage, setUsernameMessage] = useState<string | null>(null);
const [usernameError, setUsernameError] = useState<string | null>(null);
const [savingUsername, setSavingUsername] = useState(false);

const [dangerConfirm, setDangerConfirm] = useState("");
const [dangerFeedback, setDangerFeedback] = useState<string | null>(null);
const [dangerLoading, setDangerLoading] = useState(false);
const [dangerDialogOpen, setDangerDialogOpen] = useState(false);

useEffect(() => {
  if (!organizer) return;
  setOrganizationKind((organizer.organizationKind as string | null) ?? "PESSOA_SINGULAR");
  const name =
    organizer.displayName ||
    organizer.businessName ||
    profile?.fullName ||
    "";
  setEntityName(name);
  setPublicName(organizer.publicName || organizer.displayName || organizer.businessName || name);
  setCity(organizer.city ?? profile?.city ?? "");
  setAddress((organizer as { address?: string | null }).address ?? "");
  setShowAddressPublicly((organizer as { showAddressPublicly?: boolean | null }).showAddressPublicly ?? false);
  setContactEmail(contactEmailFromAccount ?? "");
  setOfficialEmail((organizer as { officialEmail?: string | null })?.officialEmail ?? contactEmailFromAccount ?? "");
  if (profile?.contactPhone) setContactPhone(profile.contactPhone);
  setBrandingAvatarUrl((organizer as { brandingAvatarUrl?: string | null }).brandingAvatarUrl ?? "");
  setBrandingPrimaryColor((organizer as { brandingPrimaryColor?: string | null }).brandingPrimaryColor ?? "");
  setBrandingSecondaryColor((organizer as { brandingSecondaryColor?: string | null }).brandingSecondaryColor ?? "");
  setUsername((organizer as { username?: string | null }).username ?? "");
}, [organizer, profile, contactEmailFromAccount]);

const hasOrganizer = useMemo(() => organizer && data?.ok, [organizer, data]);
const membershipRole = data?.membershipRole ?? null;
const isOwner = membershipRole === "OWNER";
const dangerReady = dangerConfirm.trim().toUpperCase() === "APAGAR";
const officialEmailVerifiedAt = organizer?.officialEmailVerifiedAt ? new Date(organizer.officialEmailVerifiedAt) : null;
const officialEmailStatusLabel = officialEmailVerifiedAt
  ? `Verificado ${officialEmailVerifiedAt.toLocaleDateString()}`
  : organizer?.officialEmail
  ? "A aguardar verificação"
  : "Por definir";
const officialEmailBadgeClass = officialEmailVerifiedAt
  ? "border-emerald-400/50 bg-emerald-500/10 text-emerald-50"
  : organizer?.officialEmail
  ? "border-amber-300/50 bg-amber-500/10 text-amber-50"
  : "border-white/20 bg-white/5 text-white/70";

async function handleSaveOrg() {
  if (!user) {
    openModal({ mode: "login", redirectTo: "/organizador/settings", showGoogle: true });
    return;
  }
  if (!entityName.trim()) {
    setOrgMessage("Preenche o nome da organização.");
    return;
  }
  if (!city.trim()) {
    setOrgMessage("Indica a cidade base.");
    return;
  }
  if (contactPhone && !isValidPhone(contactPhone)) {
    setPhoneError("Telefone inválido. Introduz um número válido (podes incluir indicativo, ex.: +351...).");
    return;
}

```

```

    }
    setSavingOrg(true);
    setOrgMessage(null);
    try {
      const res = await fetch("/api/organizador/me", {
        method: "PATCH",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          displayName: entityName,
          businessName: entityName,
          publicName,
          city,
          address,
          showAddressPublicly,
          contactPhone,
          fullName: profile?.fullName ?? entityName,
          organizationKind,
          entityType: organizationKind,
        }),
      });
      const json = await res.json().catch(() => null);
      if (!res.ok || json?.ok === false) {
        setOrgMessage(json?.error || "Não foi possível guardar as definições.");
      } else {
        setOrgMessage("Dados da organização guardados.");
        mutate();
      }
    } catch (err) {
      console.error("[organizador/settings] save", err);
      setOrgMessage("Erro inesperado ao guardar.");
    } finally {
      setSavingOrg(false);
    }
  }

  async function handleOfficialEmailUpdate() {
    if (!organizer?.id) {
      setOfficialEmailMessage("Seleciona uma organização primeiro.");
      return;
    }
    if (!isOwner) {
      setOfficialEmailMessage("Apenas o Owner pode alterar este email.");
      return;
    }
    if (!officialEmail.trim()) {
      setOfficialEmailMessage("Indica um email oficial válido.");
      return;
    }

    setOfficialEmailSaving(true);
    setOfficialEmailMessage(null);
    try {
      const res = await fetch("/api/organizador/organizations/settings/official-email", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ organizerId: organizer.id, email: officialEmail.trim() }),
      });
      const json = await res.json().catch(() => null);
      if (!res.ok || json?.ok === false) {
        setOfficialEmailMessage(json?.error || "Não foi possível iniciar a verificação.");
      } else {
        setOfficialEmailMessage("Pedido enviado. Verifica a caixa de email para confirmar.");
        mutate();
      }
    } catch (err) {
      console.error("[organizador/settings] official-email", err);
      setOfficialEmailMessage("Erro inesperado ao enviar pedido.");
    } finally {
      setOfficialEmailSaving(false);
    }
  }

  const handleLogoUpload = async (file: File | null) => {
    if (!file) return;
    setBrandingUploading(true);
    setBrandingMessage(null);
    try {
      const formData = new FormData();

```

```

formData.append("file", file);
const uploadRes = await fetch("/api/upload", { method: "POST", body: formData });
const uploadJson = await uploadRes.json().catch(() => null);
if (!uploadRes.ok || !uploadJson?.url) {
  setBrandingMessage(uploadJson?.error || "Falha no upload do logo.");
  return;
}
setBrandingAvatarUrl(uploadJson.url);
const saveRes = await fetch("/api/organizador/me", {
  method: "PATCH",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ brandingAvatarUrl: uploadJson.url }),
});
const saveJson = await saveRes.json().catch(() => null);
if (!saveRes.ok || saveJson?.ok === false) {
  setBrandingMessage(saveJson?.error || "Não foi possível guardar o logo.");
} else {
  setBrandingMessage("Logo atualizado.");
  mutate();
}
} catch (err) {
  console.error("[organizador/settings] upload logo", err);
  setBrandingMessage("Erro inesperado ao fazer upload.");
} finally {
  setBrandingUploading(false);
}
};

const handleSaveBranding = async () => {
  setBrandingMessage(null);
  try {
    const res = await fetch("/api/organizador/me", {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        brandingPrimaryColor,
        brandingSecondaryColor,
      }),
    });
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {
      setBrandingMessage(json?.error || "Não foi possível guardar as cores.");
    } else {
      setBrandingMessage("Branding atualizado.");
      mutate();
    }
  } catch (err) {
    console.error("[organizador/settings] branding", err);
    setBrandingMessage("Erro inesperado ao guardar branding.");
  }
};

const handleSaveUsername = async () => {
  if (!user) {
    openModal({ mode: "login", redirectTo: "/organizador/settings", showGoogle: true });
    return;
  }
  setUsernameMessage(null);
  const normalized = username.trim().toLowerCase();
  if (!/^[a-z0-9_-]{3,}$/.test(normalized)) {
    setUsernameError("Usa apenas letras minúsculas, números e - ou _. Mínimo 3 caracteres.");
    return;
  }
  setUsernameError(null);
  setSavingUsername(true);
  try {
    const res = await fetch("/api/organizador/username", {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ username: normalized }),
    });
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {
      setUsernameMessage(json?.error || "Não foi possível atualizar o username.");
    } else {
      setUsernameMessage("Username atualizado.");
      setUsername(normalized);
      mutate();
    }
  }
};

```

```

        }
    } catch (err) {
        console.error("[organizador/settings] username", err);
        setUsernameMessage("Erro inesperado ao atualizar username.");
    } finally {
        setSavingUsername(false);
    }
};

const handleDeleteOrganization = async () => {
    if (!organizer?.id) return;
    if (dangerConfirm.trim().toUpperCase() !== "APAGAR") {
        setDangerFeedback("Escreve APAGAR para confirmares.");
        return;
    }
    setDangerLoading(true);
    setDangerFeedback(null);
    try {
        const res = await fetch(`api/organizador/organizations/${organizer.id}`, { method: "DELETE" });
        const json = await res.json().catch(() => null);
        if (!res.ok || json?.ok === false) {
            setDangerFeedback(json?.error || "Não foi possível apagar a organização.");
        } else {
            setDangerFeedback("Organização apagada. Redirecionámos-te para gerir outras.");
            setDangerConfirm("");
            setDangerDialogOpen(false);
            router.push("/organizador/organizations");
        }
    } catch (err) {
        console.error("[organizador/settings] delete", err);
        setDangerFeedback("Erro inesperado ao apagar.");
    } finally {
        setDangerLoading(false);
    }
};

if (!user) {
    return (
        <div className="mx-auto max-w-5xl px-4 py-10 space-y-4 text-white md:px-6 lg:px-10">
            <h1 className="text-2xl font-semibold">Definições do organizador</h1>
            <p>Precisas de iniciar sessão para aceder a estas definições.</p>
            <button
                type="button"
                onClick={() => openModal({ mode: "login", redirectTo: "/organizador/settings", showGoogle: true })}
                className="rounded-full bg-white px-2 py-2 text-sm font-semibold text-black"
            >
                Entrar
            </button>
        </div>
    );
}

if (isLoading || !hasOrganizer) {
    return (
        <div className="mx-auto max-w-5xl px-4 py-10 text-white md:px-6 lg:px-10">
            {isLoading ? "A carregar definições..." : "Ativa a conta de organizador para gerir estas definições."}
        </div>
    );
}

return (
    <div className="mx-auto max-w-6xl px-4 py-8 space-y-6 text-white md:px-8 lg:px-10">
        <div className="space-y-1">
            <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Definições</p>
        <div className="flex flex-wrap items-center gap-3">
            <div>
                <h1 className="text-3xl font-semibold">Perfil do organizador</h1>
                <p className="text-sm text-white/65">Identidade, branding e contactos públicos.</p>
            </div>
            {organizer?.username && (
                <a
                    href={`org/${organizer.username}`}
                    target="_blank"
                    rel="noopener"
                    className="inline-flex items-center gap-2 rounded-full border border-white/20 bg-white/5 px-3 py-1.5 text-[12px] text-white hover:bg-white/10"
                >
                    Ver página pública
                </a>
            )
            }
        </div>
    </div>
)
}

```

```

        </a>
    )}
</div>
</div>

<section className="rounded-2xl border border-white/10 bg-white/5 p-4 space-y-3 shadow-[0_16px_50px_rgba(0,0,0,0.45)]">
<div className="flex items-center justify-between">
<div>
    <h2 className="text-lg font-semibold">Dados da organização</h2>
    <p className="text-[12px] text-white/65">Identidade pública, localização base e contactos.</p>
</div>
<button
    type="button"
    onClick={handleSaveOrg}
    disabled={savingOrg}
    className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-4 py-2 text-sm font-semibold
text-black shadow disabled:opacity-60"
    >
    {savingOrg ? "A guardar..." : "Guardar"}
</button>
</div>
<div className="grid gap-3 md:grid-cols-2">
<div className="space-y-1">
    <label className="text-[12px] text-white/70">Nome da organização / entidade*</label>
    <input
        value={entityName}
        onChange={(e) => setEntityName(e.target.value)}
        className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]">
        placeholder="Ex.: Clube XPTO Padel"
    />
</div>
<div className="space-y-1">
    <label className="text-[12px] text-white/70">Nome público (como os clientes te vêem)*</label>
    <input
        value={publicName}
        onChange={(e) => setPublicName(e.target.value)}
        className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]">
        placeholder="Ex.: XPTO Events"
    />
    <p className="text-[11px] text-white/55">Se deixares vazio, usamos o nome da organização.</p>
</div>
<div className="space-y-1">
    <label className="text-[12px] text-white/70">Cidade base*</label>
    <input
        list="pt-cities"
        value={city}
        onChange={(e) => setCity(e.target.value)}
        className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]">
        placeholder="Porto, Lisboa..."
    />
<datalist id="pt-cities">
    {PORTUGAL_CITIES.map((c) => (
        <option key={c} value={c} />
    )))
</datalist>
</div>
</div>
<div className="grid gap-3 md:grid-cols-2">
<div className="space-y-1">
    <label className="text-[12px] text-white/70">Morada (opcional)</label>
    <input
        value={address}
        onChange={(e) => setAddress(e.target.value)}
        className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]">
        placeholder="Rua e número"
    />
    <label className="mt-1 flex items-center gap-2 text-[12px] text-white/70">
        <input
            type="checkbox"
            checked={showAddressPublicly}
            onChange={(e) => setShowAddressPublicly(e.target.checked)}
            className="h-4 w-4"
        />
        Mostrar morada na página pública
    </label>
</div>
</div>

```

```

        </label>
    </div>
<div className="space-y-1">
    <label className="text-[12px] text-white/70">Email de contacto *</label>
    <input
        value={contactEmail}
        readOnly
        className="w-full rounded-xl border border-white/15 bg-black/25 px-3 py-2 text-sm text-white/80"
        placeholder="email@exemplo.pt"
    />
    <p className="text-[11px] text-white/50">Usamos o email da tua conta. Altera em /me/settings se precisares.</p>
</div>
</div>
<div className="grid gap-3 md:grid-cols-2">
    <div className="space-y-1">
        <label className="text-[12px] text-white/70">Telefone de contacto (opcional)</label>
        <input
            value={contactPhone}
            onChange={(e) => {
                const sanitized = sanitizePhone(e.target.value);
                setContactPhone(sanitized);
                if (sanitized && !isValidPhone(sanitized)) {
                    setPhoneError("Telefone inválido. Introduz um número válido (podes incluir indicativo, ex.: +351...).");
                } else {
                    setPhoneError(null);
                }
            }}
            inputMode="tel"
            pattern="\+?\d{6,15}"
            maxLength={18}
            className={`w-full rounded-xl border bg-black/40 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF] ${phoneError ? "border-red-400/60" : "border-white/15"}`}
            placeholder="+351912345678"
        />
        {phoneError && <p className="text-[11px] text-red-300">{phoneError}</p>}
    </div>
    </div>
    {orgMessage && <p className="text-[12px] text-white/70">{orgMessage}</p>}
</section>

<section className="rounded-2xl border border-white/10 bg-black/30 p-4 space-y-3 shadow-[0_16px_50px_rgba(0,0,0,0.45)]">
    <div className="flex flex-wrap items-center justify-between gap-3">
        <div>
            <h2 className="text-lg font-semibold">Email oficial da organização</h2>
            <p className="text-[12px] text-white/65">Apenas o Owner pode definir. Usamos para invoices, alertas críticos e transferências.</p>
        </div>
        <span className={`${inlineFlex itemsCenter roundedFull border px-3 py-1 text-[12px] ${officialEmailBadgeClass}`}>
            {officialEmailStatusLabel}
        </span>
    </div>
    <div className="grid gap-4 md:grid-cols-[1.1fr_0.9fr]">
        <div className="space-y-2">
            <label className="text-[12px] text-white/70">Email oficial (Owner)</label>
            <input
                value={officialEmail}
                onChange={(e) => setOfficialEmail(e.target.value)}
                disabled={!isOwner}
                className={`w-full rounded-xl border bg-black/40 px-3 py-2 text-sm outline-none ${isOwner ? "border-white/15 focus:border-[#6BFFFF]" : "border-white/15 text-white/60"}`}
                placeholder="equipa@organizacao.pt"
            />
            {!officialEmailVerifiedAt && organizer?.officialEmail && (
                <p className="text-[11px] text-amber-200">
                    Aguardamos confirmação. Reenvia se precisares de novo token.
                </p>
            )}
        </div>
        <div className="space-y-2 rounded-xl border border-white/10 bg-white/5 p-3 text-[12px] text-white/70">
            <p>Define o email institucional real da organização. Serve de contacto oficial para faturaçāo, alertas e trocas de Owner.</p>
            <div className="flex flex-wrap gap-2">
                <button
                    type="button"
                    onClick={handleOfficialEmailUpdate}
                    disabled={!isOwner || officialEmailSaving}
                >

```

```

        className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow disabled:opacity-60"
      >
        {officialEmailSaving ? "A enviar..." : officialEmailVerifiedAt ? "Revalidar email" : "Enviar verificação"}
      </button>
    </div>
    {officialEmailMessage && <p className="text-[11px] text-white">{officialEmailMessage}</p>}
    {(!organizer?.officialEmail || !officialEmailVerifiedAt) && (
      <p className="text-[11px] text-amber-200">Sem email oficial verificado - mostraremos aviso no dashboard.</p>
    )}
  </div>
</div>
</section>

<section className="rounded-2xl border border-white/10 bg-black/30 p-4 space-y-3 shadow-[0_16px_50px_rgba(0,0,0,0.45)]">
  <div className="flex items-center justify-between gap-2"className="text-lg font-semibold"className="text-[12px] text-white/65">Logo e cores aplicados na página pública da organização.</p>
    </div>
    <label className="inline-flex cursor-pointer items-center gap-2 rounded-full border border-white/20 bg-white/5 px-3 py-1.5 text-[12px] text-white hover:bg-white/10"type="file"
        accept="image/*"
        className="hidden"
        onChange={(e) => handleLogoUpload(e.target.files?[0] ?? null)}
        disabled={brandingUploading}
      />
      {brandingUploading ? "A enviar..." : "Upload logo"}
    </label>
  </div>
  <div className="grid gap-4 md:grid-cols-[1.1fr_0.9fr] md:items-start"className="flex items-center gap-3 rounded-xl border border-white/10 bg-white/5 p-3"className="h-16 w-16 overflow-hidden rounded-2xl border border-white/10 bg-white/10"src={brandingAvatarUrl} alt="Logo" className="h-full w-full object-cover" />
        ) : (
          <div className="flex h-full w-full items-center justify-center text-sm text-white/70">Logo</div>
        )
      </div>
      <div className="space-y-1 text-sm text-white/70"className="text-[11px] text-white/50">Fica visível no dashboard, listagens e página pública.</p>
      </div>
    </div>
    <div className="space-y-2 rounded-xl border border-white/10 bg-white/5 p-3"className="grid gap-2 md:grid-cols-2"className="space-y-1"className="text-[12px] text-white/70">Cor primária</label>
          <input
            type="color"
            value={brandingPrimaryColor || "#6bffff"
            onChange={(e) => setBrandingPrimaryColor(e.target.value)}
            className="h-10 w-full rounded border border-white/20 bg-black/30"
          />
        </div>
        <div className="space-y-1"className="text-[12px] text-white/70">Cor secundária</label>
          <input
            type="color"
            value={brandingSecondaryColor || "#0f172a"
            onChange={(e) => setBrandingSecondaryColor(e.target.value)}
            className="h-10 w-full rounded border border-white/20 bg-black/30"
          />
        </div>
      </div>
      <div className="flex items-center justify-between gap-2"className="text-[11px] text-white/55">Clica Guardar para aplicar as cores.</p>
        <button
          type="button"
          onClick={handleSaveBranding}
          className="rounded-full border border-white/20 px-3 py-1.5 text-[12px] font-semibold text-white hover:bg-white/10"
        >
          Guardar cores
        </button>
      </div>
    </div>
  </section>

```

```

        {brandingMessage && <p className="text-[12px] text-white/70">{brandingMessage}</p>}
    </div>
</div>
</section>

<section className="rounded-2xl border border-white/10 bg-white/5 p-4 space-y-3 shadow-[0_16px_50px_rgba(0,0,0,0.45)]">
    <div className="flex items-center justify-between gap-3">
        <div>
            <h2 className="text-lg font-semibold">Username ORYA:</h2>
            <p className="text-[12px] text-white/65">Handle público global do organizador.</p>
        </div>
        <div className="flex flex-wrap items-center gap-2">
            <input
                value={username}
                onChange={(e) => setUsername(e.target.value)}
                className="w-48 rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF]" placeholder="ex.: clube-xpto"
            />
            <button
                type="button"
                onClick={handleSaveUsername}
                disabled={savingUsername}
                className="rounded-full border border-white/20 bg-white/10 px-3 py-1.5 text-sm font-semibold text-white hover:bg-white/15 disabled:opacity-60"
            >
                {savingUsername ? "A guardar..." : "Guardar"}
            </button>
        </div>
    </div>
    <p className="text-[11px] text-white/60">Letras minúsculas, números e - ou _ . Mínimo 3 caracteres.</p>
    {(usernameMessage || usernameError) && (
        <p className="text-[12px] text-white/70">{usernameError || usernameMessage}</p>
    )}
</section>

<section className="rounded-2xl border border-white/10 bg-black/25 p-4 space-y-2 shadow-[0_16px_50px_rgba(0,0,0,0.45)]">
    <h2 className="text-lg font-semibold">Preferências (em breve)</h2>
    <p className="text-[12px] text-white/65">
        Aqui vais conseguir gerir idioma, notificações e visibilidade pública. Ainda não está disponível nesta versão.
    </p>
</section>

<section className="rounded-2xl border border-red-500/30 bg-red-500/5 p-4 space-y-3 shadow-[0_16px_50px_rgba(0,0,0,0.45)]">
    <div className="flex items-center justify-between">
        <div>
            <h2 className="text-lg font-semibold text-red-100">Zona de perigo</h2>
            <p className="text-[12px] text-red-100/80">
                Apagar a organização marca-a como suspensa e remove memberships. Apenas Owners podem fazê-lo.
            </p>
        </div>
    </div>
    <div className="grid gap-3 md:grid-cols-[1.2fr_0.8fr] md:items-end">
        <div className="space-y-1">
            <label className="text-[12px] text-white/80">Escreve APAGAR para confirmar</label>
            <input
                value={dangerConfirm}
                onChange={(e) => setDangerConfirm(e.target.value)}
                className="w-full rounded-lg border border-red-400/40 bg-black/40 px-3 py-2 text-sm outline-none focus:border-red-200" placeholder="APAGAR"
            />
            <p className="text-[11px] text-white/60">
                Ação irreversível (soft delete). Bloqueada se existirem bilhetes vendidos.
            </p>
        </div>
        <div className="flex flex-col gap-2 md:items-end">
            <button
                type="button"
                onClick={() => setDangerDialogOpen(true)}
                disabled={!isOwner || !dangerReady || dangerLoading}
                className="w-full rounded-full border border-red-400/60 bg-red-500/15 px-4 py-2 text-sm font-semibold text-red-100 shadow hover:bg-red-500/25 disabled:opacity-60 md:w-auto"
            >
                {dangerLoading ? "A apagar..." : "Apagar organização"}
            </button>
            {!isOwner &&

```

```

        <p className="text-[11px] text-white/60">Só Owners podem apagar esta organização.</p>
    )}
{dangerFeedback && (
    <p className="text-[12px] text-white/70">{dangerFeedback}</p>
)}
</div>
</div>
</section>

<ConfirmDestructiveActionDialog
open={dangerDialogOpen}
title="Apagar organização?"
description="Esta ação marca a organização como suspensa/arquivada. Não apaga vendas já feitas."
consequences={[
    "Perdes acesso ao dashboard desta organização.",
    "As equipas deixam de ter acesso.",
    "Eventos e dados ficam ocultos do público.",
]}
confirmLabel="Apagar organização"
cancelLabel="Cancelar"
dangerLevel="high"
onClose={() => setDangerDialogOpen(false)}
onConfirm={() => {
    if (dangerConfirm.trim().toUpperCase() !== "APAGAR") {
        setDangerFeedback("Escreve APAGAR para confirmares.");
        return;
    }
    handleDeleteOrganization();
}}
/>
</div>
);
}

```

app/organizador/(dashboard)/staff/page.tsx

```

"use client";

import { useMemo, useState } from "react";
import useSWR from "swr";
import { useRouter, useSearchParams } from "next/navigation";
import Link from "next/link";
import { useUser } from "@app/hooks/useUser";
import { useAuthModal } from "@app/components/autenticação/AuthModalContext";
import { ConfirmDestructiveActionDialog } from "@app/components/ConfirmDestructiveActionDialog";
import { trackEvent } from "@lib/analytics";
import { RoleBadge } from "../../RoleBadge";

type MemberRole = "OWNER" | "CO_OWNER" | "ADMIN" | "STAFF" | "VIEWER";

type Member = {
    userId: string;
    role: MemberRole;
    invitedByUserId: string | null;
    createdAt: string;
    fullName: string | null;
    username: string | null;
    email: string | null;
    avatarUrl: string | null;
};

type MembersResponse = {
    ok: boolean;
    items: Member[];
    viewerRole?: MemberRole | null;
    organizerId?: number | null;
    error?: string;
};

type AuditResponse =
| {
    ok: true;
    items: {
        id: string;
        action: string;
        actorUserId: string | null;
        fromUserId: string | null;
    };
}

```

```

        toUserId: string | null;
        metadata: unknown;
        createdAt: string;
    }[];
}
| { ok: false; error?: string };

type InviteStatus = "PENDING" | "EXPIRED" | "ACCEPTED" | "DECLINED" | "CANCELLED";
type Invite = {
    id: string;
    organizerId: number;
    role: MemberRole;
    targetIdentifier: string;
    targetUserId: string | null;
    status: InviteStatus;
    expiresAt: string | null;
    createdAt: string;
    invitedBy: { id: string; username: string | null; fullName: string | null; avatarUrl: string | null } | null;
    targetUser: { id: string; username: string | null; fullName: string | null; avatarUrl: string | null; email: string | null } | null;
    canRespond?: boolean;
};

type InvitesResponse = {
    ok: boolean;
    items: Invite[];
    viewerRole?: MemberRole | null;
    organizerId?: number | null;
    error?: string;
};

const fetcher = (url: string) => fetch(url).then((r) => r.json());

const roleLabels: Record<MemberRole, string> = {
    OWNER: "Owner",
    CO_OWNER: "Co-owner",
    ADMIN: "Admin",
    STAFF: "Staff",
    VIEWER: "Viewer",
};

const roleOrder: Record<MemberRole, number> = {
    OWNER: 0,
    CO_OWNER: 1,
    ADMIN: 2,
    STAFF: 3,
    VIEWER: 4,
};

const statusTone: Record<InviteStatus, string> = {
    PENDING: "border-amber-300/40 bg-amber-300/10 text-amber-100",
    EXPIRED: "border-white/15 bg-white/5 text-white/60",
    ACCEPTED: "border-emerald-400/40 bg-emerald-400/10 text-emerald-100",
    DECLINED: "border-red-400/40 bg-red-400/10 text-red-100",
    CANCELLED: "border-white/15 bg-white/5 text-white/60",
};

function canManageMember(actorRole: MemberRole | null, targetRole: MemberRole) {
    if (!actorRole) return false;
    if (actorRole === "OWNER") return true;
    if (actorRole === "CO_OWNER") return targetRole !== "OWNER" && targetRole !== "CO_OWNER";
    if (actorRole === "ADMIN") return targetRole === "STAFF" || targetRole === "VIEWER";
    return false;
}

function canAssignRole(actorRole: MemberRole | null, targetRole: MemberRole, desiredRole: MemberRole) {
    if (!actorRole) return false;
    if (actorRole === "OWNER") return true;
    if (actorRole === "CO_OWNER") {
        if (desiredRole === "OWNER" || desiredRole === "CO_OWNER") return false;
        return targetRole !== "OWNER" && targetRole !== "CO_OWNER";
    }
    if (actorRole === "ADMIN") {
        const allowed = desiredRole === "STAFF" || desiredRole === "VIEWER";
        return allowed && targetRole !== "OWNER" && targetRole !== "CO_OWNER" && targetRole !== "ADMIN";
    }
    return false;
}

```

```

function InviteBadge({ status }: { status: InviteStatus }) {
  return (
    <span className={`inline-flex items-center rounded-full border px-2 py-[2px] text-[11px] uppercase tracking-[0.16em] ${statusTone[status]}`}>
      {status === "PENDING" ? "Pendente" : status === "EXPIRED" ? "Expirado" : status === "ACCEPTED" ? "Aceite" : status === "DECLINED" ? "Recusado" : "Cancelado"}
    </span>
  );
}

function Avatar({ name, avatarUrl }: { name: string; avatarUrl: string | null }) {
  const initial = name?.trim()?.[0]?.toUpperCase() || "U";
  return (
    <div className="flex h-10 w-10 items-center justify-center overflow-hidden rounded-full border border-white/5 text-sm font-semibold">
      {avatarUrl ? <img src={avatarUrl} alt={name} className="h-full w-full object-cover" /> : <span>{initial}</span>}
    </div>
  );
}

type Toast = { id: number; message: string; type: "error" | "success" };

export default function OrganizerStaffPage() {
  const router = useRouter();
  const searchParams = useSearchParams();
  const { user, profile, isLoading: isUserLoading } = useUser();
  const { openModal } = useAuthModal();

  const [inviteModalOpen, setInviteModalOpen] = useState(false);
  const [inviteIdentifier, setInviteIdentifier] = useState("");
  const [inviteRole, setInviteRole] = useState<MemberRole>("STAFF");
  const [inviteLoading, setInviteLoading] = useState(false);

  const [transferModalOpen, setTransferModalOpen] = useState(false);
  const [transferTarget, setTransferTarget] = useState("");
  const [transferConfirm, setTransferConfirm] = useState("");
  const [transferLoading, setTransferLoading] = useState(false);

  const [leaveLoading, setLeaveLoading] = useState(false);
  const [leaveConfirmOpen, setLeaveConfirmOpen] = useState(false);
  const [removeTarget, setRemoveTarget] = useState<Member | null>(null);
  const [memberActionLoading, setMemberActionLoading] = useState<string | null>(null);
  const [inviteActionLoading, setInviteActionLoading] = useState<string | null>(null);
  const [roleConfirm, setRoleConfirm] = useState<{ userId: string; newRole: MemberRole; currentRole: MemberRole; label: string }>({
    userId: "",
    newRole: "STAFF",
    currentRole: "STAFF",
    label: "",
  });
  const [roleConfirmOpen, setRoleConfirmOpen] = useState(false);

  const [toasts, setToasts] = useState<Toast[]>([]);

  const { data: meData } = useSWR<{ ok: boolean; organizer?: { id: number; displayName?: string | null } | null; orgTransferEnabled?: boolean | null }>(
    user ? "/api/organizador/me" : null,
    fetcher,
    { revalidateOnFocus: false },
  );

  const eventIdParam = searchParams?.get("eventId");
  const eventId = eventIdParam ? Number(eventIdParam) : null;
  const organizerIdParam =
    searchParams?.get("organizerId") ?? (meData?.organizer?.id ? String(meData.organizer.id) : null);
  const organizerId = organizerIdParam ? Number(organizerIdParam) : null;
  const orgTransferEnabled = meData?.orgTransferEnabled ?? false;

  const membersKey = useMemo(() => {
    if (!user) return null;
    if (organizerId) return `/api/organizador/organizations/members?organizerId=${organizerId}`;
    if (eventId && !Number.isNaN(eventId)) return `/api/organizador/organizations/members?eventId=${eventId}`;
    return null;
  }, [user, organizerId, eventId]);

  const invitesKey = useMemo(() => {
    if (!user) return null;
  }, [user]);
}

```

```

if (organizerId) return `/api/organizador/organizations/members/invites?organizerId=${organizerId}`;
if (eventId && !Number.isNaN(eventId)) return `/api/organizador/organizations/members/invites?eventId=${eventId}`;
return null;
}, [user, organizerId, eventId]);

const { data: invitesData, isLoading: isInvitesLoading, mutate: mutateInvites } = useSWR<InvitesResponse>(
  invitesKey,
  fetcher,
  { revalidateOnFocus: false },
);

const { data: membersData, isLoading: isMembersLoading, mutate: mutateMembers } = useSWR<MembersResponse>(
  membersKey,
  fetcher,
  { revalidateOnFocus: false },
);

const members = membersData?.items ?? [];
const invites = useMemo(() => invitesData?.items ?? [], [invitesData?.items]);
const viewerRole: MemberRole | null = membersData?.viewerRole ?? invitesData?.viewerRole ?? null;
const resolvedOrganizerId = organizerId ?? membersData?.organizerId ?? invitesData?.organizerId ?? null;
const canInvite = viewerRole === "OWNER" || viewerRole === "CO_OWNER" || viewerRole === "ADMIN";
const ownerCount = useMemo(() => members.filter((m) => m.role === "OWNER").length, [members]);

const sortedMembers = useMemo(() => {
  return [...members].sort((a, b) => {
    return (roleOrder[a.role] ?? 99) - (roleOrder[b.role] ?? 99);
  });
}, [members]);

const isOrganizerProfile = profile?.roles?.includes("organizer") ?? false;
const hasMembership = !!viewerRole;

const auditKey = useMemo(() => {
  if (!user || !organizerId) return null;
  const canAudit = viewerRole === "OWNER" || viewerRole === "CO_OWNER" || viewerRole === "ADMIN";
  if (!canAudit) return null;
  return `/api/organizador/organizations/audit?organizerId=${organizerId}&limit=50`;
}, [user, organizerId, viewerRole]);

const { data: auditData, isLoading: auditLoading } = useSWR<AuditResponse>(auditKey, fetcher, {
  revalidateOnFocus: false,
});

const pushToast = (message: string, type: "error" | "success" = "error") => {
  const id = Date.now() + Math.random();
  setToasts((prev) => [...prev, { id, message, type }]);
  setTimeout(() => {
    setToasts((prev) => prev.filter((t) => t.id !== id));
  }, 4200);
};

const handleRequireLogin = () => {
  openModal({ mode: "login", redirectTo: "/organizador/staff", showGoogle: true });
};

const handleInviteSubmit = async () => {
  if (!inviteIdentifier.trim() || !resolvedOrganizerId) {
    pushToast("Indica o email ou username a convidar.");
    return;
  }
  if (!canInvite || !canAssignRole(viewerRole, inviteRole, inviteRole)) {
    pushToast("Não tens permissão para enviar este convite.");
    return;
  }
  setInviteLoading(true);
  try {
    const res = await fetch("/api/organizador/organizations/members/invites", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        organizerId: resolvedOrganizerId,
        identifier: inviteIdentifier.trim(),
        role: inviteRole,
      }),
    });
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {

```

```

        pushToast(json?.error || "Não foi possível enviar o convite.");
    } else {
        pushToast("Convite enviado.", "success");
        trackEvent("organizer_staff_invited", { organizerId: resolvedOrganizerId, role: inviteRole });
        setInviteIdentifier("");
        setInviteModalOpen(false);
        mutateInvites();
    }
} catch (err) {
    console.error("[staff] invite submit error", err);
    pushToast("Erro inesperado ao enviar convite.");
} finally {
    setInviteLoading(false);
}
};

const applyRoleChange = async (userId: string, newRole: MemberRole) => {
    if (!resolvedOrganizerId) return;
    setMemberActionLoading(userId);
    try {
        const res = await fetch("/api/organizador/organizations/members",
            {
                method: "PATCH",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ organizerId: resolvedOrganizerId, userId, role: newRole }),
            });
        const json = await res.json().catch(() => null);
        if (!res.ok || json?.ok === false) {
            pushToast(json?.error || "Não foi possível alterar o papel.");
        } else {
            pushToast("Role atualizado.", "success");
            trackEvent("organizer_staff_role_changed", { organizerId: resolvedOrganizerId, userId, newRole });
            mutateMembers();
        }
    } catch (err) {
        console.error("[staff] role change error", err);
        pushToast("Erro inesperado ao alterar role.");
    } finally {
        setMemberActionLoading(null);
        setRoleConfirmOpen(false);
    }
};

const handleRoleChange = (member: Member, newRole: MemberRole) => {
    if (!canAssignRole(viewerRole, member.role, newRole)) {
        pushToast("Não tens permissão para definir este papel.");
        return;
    }

    if (member.role === "OWNER" && newRole !== "OWNER") {
        setRoleConfirm({
            userId: member.userId,
            newRole,
            currentRole: member.role,
            label: member.fullName || member.username || member.email || "Owner",
        });
        setRoleConfirmOpen(true);
        return;
    }
    applyRoleChange(member.userId, newRole);
};

const handleRemove = async (member: Member) => {
    if (!resolvedOrganizerId) return;
    if (!canManageMember(viewerRole, member.role)) {
        pushToast("Sem permissão para remover este membro.");
        return;
    }
    setRemoveTarget(member);
};

const confirmRemove = async (member: Member) => {
    if (!resolvedOrganizerId) return;
    setMemberActionLoading(member.userId);
    try {
        const res = await fetch(
            `/api/organizador/organizations/members?organizerId=${resolvedOrganizerId}&userId=${member.userId}`,
            { method: "DELETE" },
        );
    }
}

```

```

const json = await res.json().catch(() => null);
if (!res.ok || json?.ok === false) {
  pushToast(json?.error || "Não foi possível remover o membro.");
} else {
  pushToast("Membro removido.", "success");
  mutateMembers();
  trackEvent("organizer_staff_removed", {
    organizerId: resolvedOrganizerId,
    userId: member.userId,
    role: member.role,
  });
}
} catch (err) {
  console.error("[staff] remove error", err);
  pushToast("Erro inesperado ao remover membro.");
} finally {
  setMemberActionLoading(null);
  setRemoveTarget(null);
}
};

const handleInviteAction = async (inviteId: string, action: "RESEND" | "CANCEL" | "ACCEPT" | "DECLINE") => {
  if (!resolvedOrganizerId) return;
  setInviteActionLoading(inviteId);
  try {
    const res = await fetch("/api/organizador/organizations/members/invites", {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ organizerId: resolvedOrganizerId, inviteId, action }),
    });
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {
      pushToast(json?.error || "Não foi possível atualizar o convite.");
    } else {
      pushToast(
        action === "RESEND"
          ? "Convite reenviado."
          : action === "CANCEL"
          ? "Convite cancelado."
          : action === "ACCEPT"
          ? "Convite aceite."
          : "Convite recusado.",
        "success",
      );
      mutateInvites();
      if (action === "ACCEPT") {
        mutateMembers();
      }
      trackEvent("organizer_staff_invite_action", { organizerId: resolvedOrganizerId, inviteId, action });
    }
  } catch (err) {
    console.error("[staff] invite action error", err);
    pushToast("Erro inesperado ao gerir convite.");
  } finally {
    setInviteActionLoading(null);
  }
};

const handleTransfer = async () => {
  if (!orgTransferEnabled) {
    pushToast("Transferências desativadas neste momento.");
    return;
  }
  if (!resolvedOrganizerId || !transferTarget.trim()) {
    pushToast("Indica o username/email de destino.");
    return;
  }
  if (transferTarget.trim() !== transferConfirm.trim()) {
    pushToast("Confirma o destino digitando o mesmo valor.");
    return;
  }
  setTransferLoading(true);
  try {
    const res = await fetch("/api/organizador/organizations/owner/transfer", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ organizerId: resolvedOrganizerId, targetUserId: transferTarget.trim() }),
    });
  }
};

```

```

    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {
      pushToast(json?.error || "Não foi possível transferir a organização.");
    } else {
      pushToast("Pedido criado. Enviámos o pedido de confirmação ao novo Owner.", "success");
      setTransferTarget("");
      setTransferConfirm("");
      setTransferModalOpen(false);
      router.refresh();
    }
  } catch (err) {
    console.error("[staff] transfer error", err);
    pushToast("Erro inesperado ao transferir organização.");
  } finally {
    setTransferLoading(false);
  }
};

const handleLeave = async () => {
  if (!resolvedOrganizerId) return;
  setLeaveLoading(true);
  try {
    const res = await fetch("/api/organizador/organizations/leave", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ organizerId: resolvedOrganizerId }),
    });
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {
      pushToast(json?.error || "Não foi possível sair desta organização.");
    } else {
      pushToast("Saíste da organização.", "success");
      router.push("/organizador/organizations");
    }
  } catch (err) {
    console.error("[staff] leave error", err);
    pushToast("Erro inesperado ao sair.");
  } finally {
    setLeaveLoading(false);
    setLeaveConfirmOpen(false);
  }
};

if (isUserLoading) {
  return (
    <div className="mx-auto max-w-5xl px-4 py-10 md:px-6 lg:px-8">
      <p className="text-sm text-white/70">A carregar a tua conta...</p>
    </div>
  );
}

if (!user) {
  return (
    <div className="mx-auto max-w-5xl px-4 py-10 space-y-4 md:px-6 lg:px-8">
      <h1 className="text-2xl font-semibold">Staff</h1>
      <p>Precisas de iniciar sessão para gerir o staff.</p>
      <button
        type="button"
        onClick={handleRequireLogin}
        className="inline-flex items-center rounded-md border border-white/10 bg-white/5 px-4 py-2 text-sm font-medium
        hover:bg-white/10"
      >
        Entrar
      </button>
    </div>
  );
}

if (!isOrganizerProfile && !hasMembership) {
  return (
    <div className="mx-auto max-w-5xl px-4 py-10 space-y-4 md:px-6 lg:px-8">
      <h1 className="text-2xl font-semibold">Staff</h1>
      <p className="text-sm text-white/70">Ativa primeiro o perfil de organizador ou aceita um convite para entrar.</p>
    </div>
  );
}

return (

```

```

<div className="mx-auto max-w-6xl px-4 py-10 space-y-6 md:px-6 lg:px-8">
  <div className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between">
    <div>
      <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Staff & segurança</p>
      <h1 className="text-3xl font-semibold">
        Controle quem tem acesso {meData?.organizer?.displayName ? ` · ${meData.organizer.displayName}` : ""}
      </h1>
      <p className="text-sm text-white/60">
        Define papéis, gera convites e, quando ativo, transfere a organização de forma segura. Pelo menos um Owner tem de existir sempre.
      </p>
      {viewerRole === "OWNER" && !orgTransferEnabled && (
        <p className="text-[11px] text-white/50">Transferência de Owner está desativada enquanto a flag global estiver off.
      </p>
      )}
      {auditData?.ok && auditData.items.length > 0 && (
        <div className="mt-2 rounded-xl border border-white/10 bg-white/5 p-3 text-[12px] text-white/70 space-y-1">
          <p className="text-[11px] uppercase tracking-[0.2em] text-white/50">Histórico rápido (sensível)</p>
          <div className="space-y-1 max-h-36 overflow-y-auto pr-1">
            {auditData.items.map((log) => (
              <div key={log.id} className="flex items-center justify-between text-[12px] border-b border-white/5 py-1 last:border-b-0">
                <span>{log.action}</span>
                <span className="text-white/50">{new Date(log.createdAt).toLocaleString("pt-PT")}</span>
              </div>
            )))
          </div>
        </div>
      )}
    </div>
  </div>
  <div className="flex flex-wrap gap-2 text-[12px]">
    <button
      type="button"
      onClick={() => setInviteModalOpen(true)}
      className="inline-flex items-center rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-5 py-2 text-sm font-semibold text-black shadow hover:opacity-95"
    >
      Convidar membro
    </button>
    <Link
      href={eventId ? `/organizador/scan?eventId=${eventId}` : "/organizador/scan"}
      className="inline-flex items-center rounded-full border border-white/20 bg-white/5 px-4 py-2 text-sm text-white hover:bg-white/10"
    >
      Abrir check-in (QR)
    </Link>
    {viewerRole === "OWNER" && orgTransferEnabled && (
      <button
        type="button"
        onClick={() => setTransferModalOpen(true)}
        className="rounded-full border border-white/20 bg-white/5 px-4 py-2 text-sm text-white hover:bg-white/10"
      >
        Transferir organização
      </button>
    )}
    {viewerRole && (
      <button
        type="button"
        onClick={() => setLeaveConfirmOpen(true)}
        disabled={leaveLoading}
        className="rounded-full border border-white/20 bg-white/0 px-4 py-2 text-sm text-white/80 hover:bg-white/10 disabled:opacity-60"
      >
        {leaveLoading ? "A sair..." : "Sair da organização"}
      </button>
    )}
  </div>
</div>

<div className="grid gap-4 lg:grid-cols-[1.1fr_0.9fr]">
  <section className="rounded-2xl border border-white/10 bg-black/30 p-4 space-y-3 shadow-[0_16px_50px_rgba(0,0,0,0.45)]">
    <div className="flex flex-wrap items-center justify-between gap-2">
      <div>
        <h2 className="text-sm font-semibold">Membros</h2>
        <p className="text-[12px] text-white/60">
          Papéis: Owner, Co-owner, Admin, Staff e Viewer.
        </p>
      </div>

```

```


{isMembersLoading ? "A carregar..." : `${sortedMembers.length} membros${sortedMembers.length === 1 ? "" : "s"}`}


```

{isMembersLoading &&
 <div className="space-y-2">
 {Array.from({ length: 3 }).map((_, idx) => (
 <div
 key={idx}
 className="flex animate-pulse items-center justify-between rounded-xl border border-white/5 bg-white/5 p-3"
 >
 <div className="flex items-center gap-3">
 <div className="h-10 w-10 rounded-full bg-white/10" />
 <div className="space-y-2">
 <div className="h-3 w-32 rounded bg-white/10" />
 <div className="h-3 w-24 rounded bg-white/5" />
 </div>
 </div>
 </div>
 <div className="h-8 w-28 rounded-full bg-white/5" />
 </div>
)))
 </div>
)
}

{!isMembersLoading && sortedMembers.length === 0 && (
 <div className="rounded-lg border border-white/10 bg-white/5 p-4 text-sm">
 <p>Ainda não tens membros nesta organização. Convida alguém com Owner/Co-owner/Admin para começares.</p>
 </div>
)
}

{sortedMembers.length > 0 && (
 <div className="space-y-2">
 {sortedMembers.map((m) => {
 const isOwnerRow = m.role === "OWNER";
 const isOnlyOwner = isOwnerRow && ownerCount <= 1;
 const canManageMemberRow = canManageMember(viewerRole, m.role);
 const roleDisabled = !canManageMemberRow || memberActionLoading === m.userId;
 const removeDisabled = memberActionLoading === m.userId || !canManageMemberRow || isOnlyOwner;
 const displayName = m.fullName || m.username || "Utilizador";
 return (
 <div
 key={m.userId}
 className="flex flex-col gap-2 rounded-md border border-white/10 bg-white/5 p-3 md:flex-row md:items-center
md:justify-between"
 >
 <div className="flex items-start gap-3">
 <Avatar name={displayName} avatarUrl={m.avatarUrl} />
 <div className="space-y-1">
 <div className="flex flex-wrap items-center gap-2">
 {displayName}
 <RoleBadge role={m.role} />

 {new Date(m.createdAt).toLocaleDateString("pt-PT")}

 </div>
 <div className="text-[12px] text-white/60 space-x-2">
 {m.username && @{m.username}}
 {m.email && · {m.email}}
 {isOnlyOwner && · Último Owner}
 </div>
 </div>
 </div>
 <div className="flex flex-wrap items-center gap-2">
 <select
 value={m.role}
 disabled={roleDisabled || memberActionLoading === m.userId}
 onChange={(e) => handleRoleChange(m, e.target.value as MemberRole)}
 className="rounded-md border border-white/15 bg-black/40 px-3 py-2 text-sm text-white outline-none
focus:border-white/40 disabled:opacity-60"
 >
 <option value="OWNER" disabled={!canAssignRole(viewerRole, m.role, "OWNER")}>
 Owner
 </option>
 <option value="CO_OWNER" disabled={!canAssignRole(viewerRole, m.role, "CO_OWNER")}>
 Co-owner
 </option>
 <option value="ADMIN" disabled={!canAssignRole(viewerRole, m.role, "ADMIN")}>

```

        Admin
        </option>
        <option value="STAFF" disabled={!canAssignRole(viewerRole, m.role, "STAFF")}>
            Staff
        </option>
        <option value="VIEWER" disabled={!canAssignRole(viewerRole, m.role, "VIEWER")}>
            Viewer
        </option>
    </select>
    <button
        type="button"
        onClick={() => setRemoveTarget(m)}
        disabled={removeDisabled}
        className="rounded-full border border-red-500/40 bg-red-500/10 px-3 py-1.5 text-[12px] text-red-200
        hover:bg-red-500/20 disabled:opacity-60"
    >
        Remover
    </button>
</div>
</div>
);
)}
</div>
)
}
</section>

<section className="rounded-2xl border border-white/10 bg-white/5 p-4 space-y-3 shadow-[0_16px_50px_rgba(0,0,0,0.45)]">
    <div className="flex flex-wrap items-center justify-between gap-2">
        <div>
            <h2 className="text-sm font-semibold">Convites</h2>
            <p className="text-[12px] text-white/60">Gerir convites pendentes, reenvios e respostas.</p>
        </div>
        <div className="text-[11px] text-white/60">
            {isInvitesLoading ? "A carregar..." : `${invites.length} convite${invites.length === 1 ? "" : "s"}`}
        </div>
    </div>
    {isInvitesLoading &&
        <div className="space-y-2">
            {Array.from({ length: 2 }).map(_, idx) =>
                <div key={idx} className="flex animate-pulse justify-between rounded-xl border border-white/5 bg-black/30 p-3">
                    <div className="space-y-2">
                        <div className="h-3 w-40 rounded bg-white/10" />
                        <div className="h-3 w-24 rounded bg-white/5" />
                    </div>
                    <div className="h-8 w-24 rounded-full bg-white/10" />
                </div>
            ))}
        </div>
    )
}

{!isInvitesLoading && invites.length === 0 && (
    <div className="rounded-lg border border-dashed border-white/15 bg-white/5 p-4 text-sm text-white/70">
        Sem convites pendentes. Convida por email ou username para novos acessos.
    </div>
)
}

{invites.length > 0 && (
    <div className="space-y-2">
        {invites.map((inv) => {
            const isPending = inv.status === "PENDING";
            const isExpired = inv.status === "EXPIRED";
            const canRespond = inv.canRespond && isPending;
            const targetLabel =
                inv.targetUser?.fullName ||
                inv.targetUser?.username ||
                inv.targetIdentifier ||
                "Convite";
            return (
                <div
                    key={inv.id}
                    className="flex flex-col gap-2 rounded-xl border border-white/10 bg-black/30 p-3"
                >
                    <div className="flex flex-wrap items-center justify-between gap-2">
                        <div className="space-y-1">
                            <div className="flex items-center gap-2">
                                <span className="font-semibold text-white">{targetLabel}</span>
                                <RoleBadge role={inv.role} />
                            </div>
                        </div>
                    </div>
                </div>
            );
        })
    </div>
)
}

```

```

        <InviteBadge status={inv.status} />
    </div>
    <div className="text-[12px] text-white/60 space-x-2">
        <span>{inv.targetIdentifier}</span>
        {inv.targetUser?.email && <span className="text-white/50"> {inv.targetUser.email}</span>}
        {inv.expiresAt &&
            <span className="text-white/50">
                · {isExpired ? "Expirou" : "Expira"} {new Date(inv.expiresAt).toLocaleDateString("pt-PT")}
            </span>
        )}
    </div>
    {inv.invitedBy &&
        <p className="text-[11px] text-white/45">
            Enviado por {inv.invitedBy.fullName || inv.invitedBy.username || "owner"}
        </p>
    )}
</div>
<div className="flex flex-wrap items-center gap-2">
    {canRespond &&
        <>
            <button
                type="button"
                disabled={inviteActionLoading === inv.id}
                onClick={() => handleInviteAction(inv.id, "DECLINE")}
                className="rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-[12px] text-white/80
hover:bg-white/10 disabled:opacity-60"
            >
                Recusar
            </button>
            <button
                type="button"
                disabled={inviteActionLoading === inv.id}
                onClick={() => handleInviteAction(inv.id, "ACCEPT")}
                className="rounded-full bg-white px-4 py-1.5 text-[12px] font-semibold text-black shadow
disabled:opacity-60"
            >
                Aceitar
            </button>
        </>
    )}
    {canInvite &&
        <>
            <button
                type="button"
                disabled={inviteActionLoading === inv.id}
                onClick={() => handleInviteAction(inv.id, "RESEND")}
                className="rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-[12px] text-white/85
hover:bg-white/10 disabled:opacity-60"
            >
                Re-enviar
            </button>
            <button
                type="button"
                disabled={inviteActionLoading === inv.id}
                onClick={() => handleInviteAction(inv.id, "CANCEL")}
                className="rounded-full border border-red-400/40 bg-red-500/10 px-3 py-1.5 text-[12px] text-red-100 hover:bg-red-500/20 disabled:opacity-60"
            >
                Cancelar
            </button>
        </>
    )}
    {!canRespond && !canInvite &&
        <span className="text-[11px] text-white/50">A aguardar resposta.</span>
    )
}
</div>
</div>
</div>
);
)}
</div>
)
}
</section>
</div>

<ConfirmDestructiveActionDialog
    open={removeTarget !== null}
    title="Remover membro do staff?">

```

```

        description={`Isto remove ${removeTarget?.fullName || removeTarget?.username || "este membro"} desta organização.`}
        consequences={"Perde o acesso ao dashboard e check-ins desta organização."}
        confirmLabel="Remover"
        cancelLabel="Cancelar"
        dangerLevel="high"
        onClose={() => setRemoveTarget(null)}
        onConfirm={() => {
          if (removeTarget) confirmRemove(removeTarget);
        }}
      />
<ConfirmDestructiveActionDialog
  open={leaveConfirmOpen}
  title="Sair desta organização?"
  description="Perdes acesso ao dashboard e às equipas desta organização."
  confirmLabel="Sair"
  cancellabel="Cancelar"
  dangerLevel="medium"
  onClose={() => setLeaveConfirmOpen(false)}
  onConfirm={handleLeave}
/>

/* Role confirm modal */
{roleConfirmOpen && (
  <div className="fixed inset-0 z-50 flex items-center justify-center bg-black/70 px-4 backdrop-blur">
    <div className="w-full max-w-lg space-y-4 rounded-2xl border border-white/10 bg-[#0c1424] p-5 shadow-2xl">
      <div className="space-y-1">
        <p className="text-[11px] uppercase tracking-[0.22em] text-white/50">Confirmar</p>
        <h3 className="text-xl font-semibold text-white">Despromover Owner?</h3>
        <p className="text-sm text-white/70">
          Vais descer o papel de <span className="font-semibold text-white">{roleConfirm.label}</span> de Owner para
        {roleLabels[roleConfirm.newRole]}. Garante que fica pelo menos um Owner ativo.
        </p>
      </div>
      <div className="flex justify-end gap-2">
        <button
          type="button"
          onClick={() => setRoleConfirmOpen(false)}
          className="rounded-full border border-white/20 bg-white/5 px-4 py-2 text-sm text-white hover:bg-white/10"
        >
          Cancelar
        </button>
        <button
          type="button"
          onClick={() => applyRoleChange(roleConfirm.userId, roleConfirm.newRole)}
          className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow"
        >
          Confirmar
        </button>
      </div>
    </div>
  </div>
)
}

/* Invite modal */
{inviteModalOpen && (
  <div className="fixed inset-0 z-50 flex items-center justify-center bg-black/70 px-4 backdrop-blur">
    <div className="w-full max-w-lg space-y-4 rounded-2xl border border-white/10 bg-[#0c1424] p-5 shadow-2xl">
      <div className="space-y-1">
        <p className="text-[11px] uppercase tracking-[0.22em] text-white/50">Convite</p>
        <h3 className="text-xl font-semibold text-white">Convidar membro</h3>
        <p className="text-sm text-white/70">Aceita email, username ou ID ORYA. O convite expira em 14 dias.</p>
      </div>
      <div className="space-y-3">
        <div className="space-y-1">
          <label className="text-[12px] text-white/70">Email / username</label>
          <input
            type="text"
            value={inviteIdentifier}
            onChange={(e) => setInviteIdentifier(e.target.value)}
            className="w-full rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF]"
          placeholder="email@dominio.com ou @username"
          />
        </div>
        <div className="space-y-1">
          <label className="text-[12px] text-white/70">Role proposto</label>
          <select
            value={inviteRole}

```

```

        onChange={(e) => setInviteRole(e.target.value as MemberRole)}
        className="w-full rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]"}
      >
        <option value="OWNER" disabled={!canAssignRole(viewerRole, inviteRole, "OWNER")}>
          Owner
        </option>
        <option value="CO_OWNER" disabled={!canAssignRole(viewerRole, inviteRole, "CO_OWNER")}>
          Co-owner
        </option>
        <option value="ADMIN" disabled={!canAssignRole(viewerRole, inviteRole, "ADMIN")}>
          Admin
        </option>
        <option value="STAFF" disabled={!canAssignRole(viewerRole, inviteRole, "STAFF")}>
          Staff
        </option>
        <option value="VIEWER" disabled={!canAssignRole(viewerRole, inviteRole, "VIEWER")}>
          Viewer
        </option>
      </select>
    </div>
  </div>
<div className="flex justify-end gap-2">
  <button
    type="button"
    onClick={() => setInviteModalOpen(false)}
    className="rounded-full border border-white/20 bg-white/5 px-4 py-2 text-sm text-white hover:bg-white/10"
  >
    Fechar
  </button>
  <button
    type="button"
    onClick={handleInviteSubmit}
    disabled={inviteLoading}
    className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow disabled:opacity-60"
  >
    {inviteLoading ? "A enviar.." : "Enviar convite"}
  </button>
</div>
</div>
</div>
}

/* Transfer modal */
ifTransferModalOpen && orgTransferEnabled &&
<div className="fixed inset-0 z-50 flex items-center justify-center bg-black/70 px-4 backdrop-blur">
  <div className="w-full max-w-lg space-y-4 rounded-2xl border border-white/10 bg-[#0c1424] p-5 shadow-2xl">
    <div className="space-y-1">
      <p className="text-[11px] uppercase tracking-[0.22em] text-white/50">Transferir organização</p>
      <h3 className="text-xl font-semibold text-white">Passar a propriedade</h3>
      <p className="text-sm text-white/70">
        A organização será atribuída ao destino como Owner. O teu papel passa para Admin automaticamente.
      </p>
    </div>
    <div className="space-y-3">
      <div className="space-y-1">
        <label className="text-[12px] text-white/70">Username / email do novo Owner</label>
        <input
          type="text"
          value={transferTarget}
          onChange={(e) => setTransferTarget(e.target.value)}
          className="w-full rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]"}
          placeholder="@destino ou email@dominio.com"
        />
      </div>
      <div className="space-y-1">
        <label className="text-[12px] text-white/70">Confirma o destino</label>
        <input
          type="text"
          value={transferConfirm}
          onChange={(e) => setTransferConfirm(e.target.value)}
          className="w-full rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]"}
          placeholder="Escreve novamente para confirmar"
        />
      </div>
    </div>
  </div>
</div>

```

```


<div className="flex justify-end gap-2">
        <button
            type="button"
            onClick={() => setTransferModalOpen(false)}
            className="rounded-full border border-white/20 bg-white/5 px-4 py-2 text-sm text-white hover:bg-white/10"
        >
            Cancelar
        </button>
        <button
            type="button"
            onClick={handleTransfer}
            disabled={transferLoading}
            className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow disabled:opacity-60"
        >
            {transferLoading ? "A transferir..." : "Transferir"}
        </button>
    </div>
</div>
</div>
}

/* Toasts */
<div className="pointer-events-none fixed top-4 right-4 z-50 space-y-2">
    {toasts.map((toast) =>
        <div
            key={toast.id}
            className={`pointer-events-auto rounded-xl border px-4 py-3 text-sm shadow ${toast.type === "success" ? "border-emerald-400/40 bg-emerald-500/15 text-emerald-50" : "border-red-400/40 bg-red-500/15 text-red-50"}`}
        >
            {toast.message}
        </div>
    )));
</div>
</div>
);
}


```

app/organizador/(dashboard)/tournaments/[id]/finance/page.tsx

```

// app/organizador/tournaments/[id]/finance/page.tsx
import { notFound, redirect } from "next/navigation";
import { createSupabaseServer } from "@/lib/supabaseServer";

type PageProps = { params: Promise<{ id: string }> };

async function fetchFinance(tournamentId: number, cookieHeader: string | null) {
    const res = await fetch(`#${process.env.NEXT_PUBLIC_SITE_URL || ""}/api/organizador/tournaments/${tournamentId}/finance`, {
        headers: cookieHeader ? { cookie: cookieHeader } : {},
        cache: "no-store",
    });
    if (!res.ok) return null;
    return res.json();
}

export default async function TournamentFinancePage({ params }: PageProps) {
    const resolved = await params;
    const tournamentId = Number(resolved.id);
    if (!Number.isFinite(tournamentId)) notFound();

    const supabase = await createSupabaseServer();
    const { data, error } = await supabase.auth.getUser();
    if (error || !data?.user) redirect("/login");

    const cookie = (await fetch(`#${process.env.NEXT_PUBLIC_SITE_URL || ""}/api/auth/me`, { headers: { cookie: "" } }))?.headers.get("set-cookie");
    const finance = await fetchFinance(tournamentId, cookie);
    if (!finance?.ok) {
        if (finance?.error === "FORBIDDEN") redirect("/organizador");
        notFound();
    }

    const summary = finance.summary || {};
    const recent = finance.recent || [];
}

```

```

const kycIncomplete = summary.payoutMode !== "PLATFORM" && (summary.holdReason || summary.holdCents > 0);

return (
  <div className="space-y-4 rounded-2xl border border-white/10 bg-black/40 p-4">
    <div className="flex items-center justify-between">
      <div>
        <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Faturação do torneio</p>
        <h1 className="text-xl font-semibold text-white">Receita & Payouts</h1>
        <p className="text-white/70 text-sm">
          Disponível em: {summary.releaseAt ? new Date(summary.releaseAt).toLocaleDateString("pt-PT") : "N/D"}
        </p>
      </div>
      <div className="rounded-xl border border-white/15 bg-white/5 px-3 py-2 text-sm text-white/80">
        <p>Total bruto: {(summary.totalCents ?? 0) / 100} €</p>
        <p>Net: {(summary.netCents ?? 0) / 100} €</p>
        <p>Taxa plataforma: {(summary.platformFeeCents ?? 0) / 100} €</p>
      </div>
    </div>
  </div>

{kycIncomplete && (
  <div className="rounded-xl border border-amber-400/50 bg-amber-500/10 p-3 text-sm text-amber-100">
    <p className="font-semibold">Stripe/KYC por concluir</p>
    <p className="text-amber-100/80 text-[12px]">Liga a Stripe e conclui KYC para desbloquear payouts.</p>
  </div>
)}

<div className="grid gap-3 md:grid-cols-2">
  <div className="rounded-xl border border-white/10 bg-white/5 p-3 space-y-1 text-sm text-white/80">
    <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Resumo</p>
    <p>Vendas: {(summary.countSales ?? 0)} transações</p>
    <p>Em reserva: {(summary.holdCents ?? 0) / 100} € {summary.holdReason ? `(${summary.holdReason})` : ""}</p>
    <p>Payout mode: {summary.payoutMode ?? "N/D"}</p>
  </div>
  <div className="rounded-xl border border-white/10 bg-white/5 p-3 space-y-1 text-sm text-white/80">
    <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Movimentos recentes</p>
    {recent.length === 0 && <p className="text-white/60 text-sm">Sem vendas.</p>}
    {recent.map((r: any) => (
      <div key={r.id} className="rounded border border-white/10 bg-black/30 px-2 py-1 flex items-center justify-between">
        <div>
          <p className="text-white text-sm">{r.purchaseId || r.paymentIntentId || `#${r.id}'`}</p>
          <p className="text-[11px] text-white/60">{new Date(r.createdAt).toLocaleString("pt-PT")}</p>
        </div>
        <p className="text-white font-semibold text-sm">{(r.totalCents ?? 0) / 100} €</p>
      </div>
    )));
  </div>
</div>
);
}

```

app/organizador/(dashboard)/tournaments/[id]/live/page.tsx

```

"use client";

import { useEffect, useMemo, useState } from "react";
import { useRouter } from "next/navigation";
import useSWR from "swr";
import { summarizeMatchStatus, computeStandingsForGroup } from "@domain/tournaments/structure";
import { type TieBreakRule } from "@domain/tournaments/standings";
import { computeLiveWarnings } from "@domain/tournaments/liveWarnings";

type PageProps = { params: { id: string } };

const fetcher = (url: string) => fetch(url).then((r) => r.json());

function Filters({ stages, setFilters }: { stages: any[]; setFilters: (f: any) => void }) {
  const [status, setStatus] = useState<string>("");
  const [stageId, setStageId] = useState<string>("");
  const [court, setCourt] = useState<string>("");
  const [todayOnly, setTodayOnly] = useState(false);
  const [search, setSearch] = useState("");

  return (
    <div className="flex flex-wrap items-center gap-2 text-sm">
      <select

```

```

        value={status}
        onChange={(e) => {
          setStatus(e.target.value);
          setFilters((prev: any) => ({ ...prev, status: e.target.value || null }));
        }}
      className="rounded-full border border-white/15 bg-white/5 px-3 py-1 text-white/80"
    >
    <option value="">Todos os estados</option>
    <option value="PENDING">Pendente</option>
    <option value="IN_PROGRESS">Em jogo</option>
    <option value="DONE">Terminado</option>
  </select>
  <select
    value={stageId}
    onChange={(e) => {
      setStageId(e.target.value);
      setFilters((prev: any) => ({ ...prev, stageId: e.target.value ? Number(e.target.value) : null }));
    }}
  className="rounded-full border border-white/15 bg-white/5 px-3 py-1 text-white/80"
  >
    <option value="">Todas as fases</option>
    {stages.map((s) => (
      <option key={s.id} value={s.id}>
        {s.name || s.stageType}
      </option>
    )))
  </select>
  <input
    value={court}
    onChange={(e) => {
      setCourt(e.target.value);
      setFilters((prev: any) => ({ ...prev, court: e.target.value || null }));
    }}
    placeholder="Court #"
    className="w-24 rounded-full border border-white/15 bg-white/5 px-3 py-1 text-white/80"
  />
  <label className="flex items-center gap-1 text-white/75">
    <input
      type="checkbox"
      checked={todayOnly}
      onChange={(e) => {
        setTodayOnly(e.target.checked);
        setFilters((prev: any) => ({ ...prev, todayOnly: e.target.checked }));
      }}
    />
    Só hoje
  </label>
  <input
    value={search}
    onChange={(e) => {
      setSearch(e.target.value);
      setFilters((prev: any) => ({ ...prev, search: e.target.value }));
    }}
    placeholder="Pesquisar dupla #"
    className="rounded-full border border-white/15 bg-white/5 px-3 py-1 text-white/80"
  />
</div>
);
}

export default function OrganizerTournamentLivePage({ params }: PageProps) {
  const router = useRouter();
  const tournamentId = Number(params.id);
  const [authError, setAuthError] = useState<string | null>(null);

  useEffect(() => {
    if (authError === "login") router.replace("/login");
    if (authError === "organizador") router.replace("/organizador");
  }, [authError, router]);
}

if (!Number.isFinite(tournamentId)) {
  return (
    <div className="p-4 text-white/70">
      <p>ID de torneio inválido.</p>
      <button
        onClick={() => router.back()}
        className="mt-3 rounded-full border border-white/20 px-3 py-1 text-sm text-white hover:border-white/40"
      >

```

```

        Voltar
      </button>
    </div>
  );
}

const { data, error } = useSWR(`/api/organizador/tournaments/${tournamentId}/live`, fetcher);

useEffect(() => {
  if (!data?.error) return;
  if (data.error === "UNAUTHENTICATED") setAuthError("login");
  if (data.error === "FORBIDDEN") setAuthError("organizador");
}, [data?.error]);

if (error) {
  return <div className="p-4 text-white/70">Erro a carregar dados do torneio.</div>;
}
if (!data?.tournament) return <div className="p-4 text-white/70">A carregar...</div>;

const tournament = data.tournament;
const tieBreakRules: TieBreakRule[] = Array.isArray(tournament.tieBreakRules)
  ? (tournament.tieBreakRules as TieBreakRule[])
  : (["WINS", "SET_DIFF", "GAME_DIFF", "HEAD_TO_HEAD", "RANDOM"] as TieBreakRule[]);

const stages = useMemo(
() =>
  tournament.stages.map((s: any) => ({
    ...s,
    groups: s.groups.map((g: any) => ({
      ...g,
      standings: computeStandingsForGroup(g.matches, tieBreakRules, tournament.generationSeed || undefined),
      matches: g.matches.map((m: any) => ({ ...m,StatusLabel: summarizeMatchStatus(m.status) })),
    })),
    matches: s.matches.map((m: any) => ({ ...m,StatusLabel: summarizeMatchStatus(m.status) })),
  })),
  [tournament, tieBreakRules],
);
}

const flatMatches = stages.flatMap((s: any) => [...s.matches, ...s.groups.flatMap((g: any) => g.matches)]);
const warnings = computeLiveWarnings({
  matches: flatMatches,
  pairings: data.pairings ?? [],
  startThresholdMinutes: 60,
});

const [filters, setFilters] = useState<any>({
  status: null,
  stageId: null,
  court: null,
  todayOnly: false,
  search: "",
});

const now = new Date();
const filteredMatches = flatMatches.filter((m: any) => {
  if (filters.status && m.status !== filters.status) return false;
  if (filters.stageId && m.stageId !== filters.stageId) return false;
  if (filters.court && `${m.courtId ?? ""}` !== filters.court) return false;
  if (filters.todayOnly && m.startAt) {
    const d = new Date(m.startAt);
    if (
      d.getFullYear() !== now.getFullYear() ||
      d.getMonth() !== now.getMonth() ||
      d.getDate() !== now.getDate()
    )
      return false;
  }
  if (filters.search) {
    const term = filters.search.trim();
    if (!term) return true;
    return `${m.pairing1Id ?? ""}`.includes(term) || `${m.pairing2Id ?? ""}`.includes(term);
  }
  return true;
});

const summary = {
  pending: flatMatches.filter((m: any) => m.status === "PENDING").length,
  inProgress: flatMatches.filter((m: any) => m.status === "IN_PROGRESS").length,
}

```

```

done: flatMatches.filter((m: any) => m.status === "DONE").length,
};

return (
  <div className="space-y-4 rounded-2xl border border-white/10 bg-black/40 p-4">
    <div className="flex items-center justify-between gap-2">
      <div>
        <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Live Torneio</p>
        <h1 className="text-xl font-semibold text-white">{tournament?.event?.title ?? "Torneio"}</h1>
        <p className="text-white/70 text-sm">Formato: {tournament.format}</p>
      </div>
      <div className="rounded-xl border border-white/15 bg-white/5 px-3 py-2 text-sm text-white/80">
        <p>Jogos: {flatMatches.length}</p>
        <p>Pendentes {summary.pending} · Em jogo {summary.inProgress} · Terminados {summary.done}</p>
      </div>
    </div>
  </div>

{warnings.length > 0 && (
  <div className="rounded-xl border border-amber-400/40 bg-amber-400/10 p-3 text-sm text-amber-100">
    <p className="font-semibold">Avisos</p>
    <ul className="list-disc pl-4 space-y-1">
      {warnings.map((w: any, idx: number) => (
        <li key={`${w.type}-${w.matchId ?? w.pairingId}-${idx}`}>
          {w.type === "REQUIRES_ACTION" && <>>Dupla #{w.pairingId} exige ação</>}
          {w.type === "MISSING_COURT" && <>>Jogo #{w.matchId}: sem court</>}
          {w.type === "MISSING_START" && <>>Jogo #{w.matchId}: sem horário definido</>}
          {w.type === "INVALID_SCORE" && <>>Jogo #{w.matchId}: score inválido</>}
        </li>
      )))
    </ul>
  </div>
)}

<Filters stages={stages} setFilters={setFilters} />

<div className="grid gap-4 md:grid-cols-2">
  {stages.map((stage: any) =>
    <div key={stage.id} className="rounded-xl border border-white/10 bg-white/5 p-3 space-y-3">
      <div className="flex items-center justify-between">
        <div>
          <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">
            {stage.name || stage.stageType}
          </p>
          <p className="text-white/75 text-sm">{stage.matches.length} jogos</p>
        </div>
        <div className="rounded-full border border-white/15 bg-white/10 px-3 py-1 text-[11px] text-white/70">
          {stage.stageType}
        </div>
      </div>
    </div>

    {stage.groups.length > 0 && (
      <div className="space-y-2">
        {stage.groups.map((group: any) => (
          <div key={group.id} className="rounded-lg border border-white/10 bg-black/40 p-2">
            <p className="text-[12px] text-white/70 mb-1">{group.name}</p>
            <div className="space-y-1">
              {group.standings.map((row: any, idx: number) => (
                <div key={row.pairingId ?? idx} className="flex items-center justify-between text-[12px] text-white/80">
                  <span>
                    #{idx + 1} · Dupla {row.pairingId ?? "-"}
                  </span>
                  <span>{row.points} pts</span>
                </div>
              )))
            </div>
            <div className="mt-2 space-y-1">
              {group.matches.map((match: any) => (
                <div key={match.id} className="rounded border border-white/10 bg-white/5 px-2 py-1 text-[12px] text-white/75">
                  <div className="flex items-center justify-between">
                    <span>Jogo #{match.id}</span>
                    <span>{matchStatusLabel}</span>
                  </div>
                  <div className="text-white/60">
                    {match.pairing1Id ?? "-"} vs {match.pairing2Id ?? "-"}
                  </div>
                </div>
              )))
            </div>
          </div>
        ))
      </div>
    )
  )
}

```

```

        </div>
    )}
</div>
</div>
)}
</div>
)}

{stage.matches.length > 0 && (
<div className="space-y-1">
{stage.matches.map((match: any) => (
<div
key={match.id}
className="rounded border border-white/10 bg-white/5 px-3 py-2 text-[12px] text-white/75"
>
<div className="flex items-center justify-between">
<span>Jogo #{match.id}</span>
<span>{match.statusLabel}</span>
</div>
<div className="text-white/60">
{match.pairing1Id ?? "-"} vs {match.pairing2Id ?? "-"}
</div>
</div>
))}
</div>
)}
</div>
);
}

```

app/organizador/BackButton.tsx

```

"use client";

import { useMemo } from "react";
import { usePathname, useRouter, useSearchParams } from "next/navigation";

type Props = {
  className?: string;
  hideOnRoot?: boolean;
};

export function BackButton({ className = "", hideOnRoot = true }: Props) {
  const router = useRouter();
  const pathname = usePathname();
  const searchParams = useSearchParams();

  const target = useMemo(() => {
    if (!pathname?.startsWith("/organizador")) return null;
    const segments = pathname.split("/").filter(Boolean); // e.g., ["organizador", "promo", "evento", "123"]

    // Se estamos em /organizador com tabs/sections → volta ao overview ou à tab base
    if (segments.length <= 1) {
      const tab = searchParams?.get("tab");
      if (tab && tab !== "overview") {
        return `/organizador?tab=${tab}`;
      }
      return "/organizador";
    }

    // Rotas profundas → remove último segmento
    const trimmed = "/" + segments.slice(0, -1).join("/");
    return trimmed || "/organizador";
  }, [pathname, searchParams]);

  if (!target) return null;
  if (hideOnRoot && target === "/organizador" && (!searchParams?.get("tab") || searchParams.get("tab") === "overview")) {
    return null;
  }

  return (
    <button
      type="button"

```

```

    onClick={() => router.push(target)}
    className={`inline-flex items-center gap-2 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-sm text-white
    hover:bg-white/10 transition ${className}`}
  >
  ~ Voltar
  </button>
);
}

```

app/organizador/become/page.tsx

```

export const runtime = "nodejs";

import { redirect } from "next/navigation";
import { createSupabaseServer } from "@/lib/supabaseServer";
import BecomeOrganizerForm from "@/components/organizer/BecomeOrganizerForm";
import BackLink from "@/components/BackLink";

// app/organizador/become/page.tsx

export default async function BecomeOrganizerPage() {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) {
    redirect("/login?next=/organizador/become");
  }

  return (
    <div className="orya-body-bg min-h-screen px-4 py-12 text-white">
      <div className="mx-auto max-w-[1160px] space-y-10">
        <div className="flex items-center justify-start">
          <BackLink hrefFallback="/explorar" label="Voltar" />
        </div>

        <header className="space-y-2.5 text-center md:space-y-3">
          <div className="mx-auto inline-flex items-center rounded-full border border-white/10 bg-white/5 px-3 py-1 text-[11px]
uppercase tracking-[0.16em] text-white/70">
            Organizador • Onboarding
          </div>
          <h1 className="text-3xl font-semibold md:text-[32px]">Cria a tua organização na ORYA</h1>
          <p className="mx-auto max-w-3xl text-[15px] text-white/75 md:text-base">
            Em menos de 1 minuto ficas pronto para vender bilhetes, sem burocracias.
          </p>
        </header>

        <BecomeOrganizerForm />

        <footer className="pt-4 text-center text-[12px] text-white/60">
          Ao continuar, confirmas que representas esta entidade e aceitas os{" "}
          <a href="#" className="underline underline-offset-2 hover:text-white">
            Termos do Organizador da ORYA
          </a>
          .
        </footer>
      </div>
    );
}

```

app/organizador/DashboardClient.tsx

```

"use client";

import { useCallback, useEffect, useMemo, useState } from "react";
import Link from "next/link";
import { use pathname, useRouter, useSearchParams } from "next/navigation";
import useSWR from "swr";
import { ConfirmDestructiveActionDialog } from "@app/components/ConfirmDestructiveActionDialog";
import { trackEvent } from "@lib/analytics";
import { useUser } from "@app/hooks/useUser";
import { useAuthModal } from "@app/components/autenticação/AuthModalContext";

```

```

import PromoCodesPage from "./promo/PromoCodesClient";
import OrganizerSettingsPage from "./(dashboard)/settings/page";
import OrganizerStaffPage from "./(dashboard)/staff/page";
import { BackButton } from "./BackButton";
import PadelHubClient from "./(dashboard)/padel/PadelHubClient";
import { SalesLineChart } from "@app/components/charts/SalesLineChart";
import { SalesAreaChart } from "@app/components/charts/SalesAreaChart";
import InvoicesClient from "./pagamentos/invoices/invoices-client";

const fetcher = (url: string) => fetch(url).then((res) => res.json());

type OverviewResponse = {
  ok: boolean;
  totalTickets: number;
  totalRevenueCents: number;
  grossCents?: number;
  discountCents?: number;
  platformFeeCents?: number;
  netRevenueCents?: number;
  eventsWithSalesCount: number;
  activeEventsCount: number;
};

type EventItem = {
  id: number;
  slug: string;
  title: string;
  startsAt: string;
  endsAt?: string | null;
  templateType?: string | null;
  locationName: string | null;
  locationCity: string | null;
  status: string;
  isFree: boolean;
  ticketsSold?: number;
  revenueCents?: number;
  capacity?: number | null;
  categories?: string[];
  padelClubId?: number | null;
  padelPartnerClubIds?: number[];
  padelClubName?: string | null;
  padelPartnerClubNames?: Array<string | null>;
};

type EventsResponse = { ok: boolean; items: EventItem[] };

type PayoutSummaryResponse =
  | {
      ok: true;
      ticketsSold: number;
      revenueCents: number;
      grossCents: number;
      platformFeesCents: number;
      eventsWithSales: number;
      estimatedPayoutCents: number;
    }
  | { ok: false; error?: string };

type PromoCodeRow = {
  id: number;
  code: string;
  type: "PERCENTAGE" | "FIXED";
  value: number;
  maxUses: number | null;
  perUserLimit: number | null;
  validFrom: string | null;
  validUntil: string | null;
  active: boolean;
  eventId: number | null;
  redemptionsCount?: number;
  autoApply?: boolean;
  minQuantity?: number | null;
  minTotalCents?: number | null;
};

type PromoListResponse = {
  ok: boolean;
  promoCodes: PromoCodeRow[];
};

```

```

events: { id: number; title: string; slug: string }[];
error?: string;
};

type BuyersResponse =
| {
  ok: true;
  eventId: number;
  items: {
    id: string;
    ticketType: string;
    priceCents: number;
    totalPaidCents: number;
    status: string;
    purchasedAt: string;
    buyerName: string;
    buyerEmail: string;
    buyerCity: string | null;
    paymentIntentId: string | null;
  }[];
}
| { ok: false; error?: string };

type FinanceOverviewResponse =
| {
  ok: true;
  totals: { grossCents: number; netCents: number; feesCents: number; tickets: number; eventsWithSales: number };
  rolling: {
    last7: { grossCents: number; netCents: number; feesCents: number; tickets: number };
    last30: { grossCents: number; netCents: number; feesCents: number; tickets: number };
  };
  upcomingPayoutCents: number;
  events: {
    id: number;
    title: string;
    slug: string;
    startsAt: string | null;
    status: string | null;
    grossCents: number;
    netCents: number;
    feesCents: number;
    ticketsSold: number;
  }[];
  error?: string;
}
| { ok: false; error?: string };

type MarketingOverviewResponse = {
  ok: boolean;
  totalTickets: number;
  ticketsWithPromo: number;
  guestTickets: number;
  totalRevenueCents: number;
  marketingRevenueCents: number;
  topPromo: { id: number; code: string; redemptionsCount: number; revenueCents: number } | null;
  events?: {
    id: number;
    title: string;
    slug: string;
    startsAt: string | null;
    templateType: string | null;
    locationName: string | null;
    locationCity: string | null;
    capacity: number | null;
    ticketsSold: number;
    revenueCents: number;
  }[];
};

type AudienceSummaryResponse =
| {
  ok: true;
  segments: {
    frequent: number;
    newLast60d: number;
    highSpenders: number;
    groups: number;
    dormant90d: number;
    local: number;
  };
}

```

```

| { ok: false; error?: string };
type OrganizerStatus = {
  paymentsStatus?: "NO_STRIPE" | "PENDING" | "READY";
  paymentsMode?: "CONNECT" | "PLATFORM";
  profileStatus?: "MISSING_CONTACT" | "OK";
  contactEmail?: string | null;
};
type OrganizerLite = {
  id?: number;
  status?: string | null;
  entityType?: string | null;
  displayName?: string | null;
  city?: string | null;
  payoutIban?: string | null;
  officialEmail?: string | null;
  officialEmailVerifiedAt?: string | null;
};

type TabKey =
  | "overview"
  | "events"
  | "sales"
  | "finance"
  | "invoices"
  | "marketing"
  | "padel"
  | "restaurants"
  | "volunteer"
  | "night"
  | "staff"
  | "settings";

const ALL_TABS: TabKey[] = ["overview", "events", "sales", "marketing", "staff", "finance", "invoices", "padel", "settings"];
type SalesRange = "7d" | "30d" | "90d" | "365d" | "all";

export default function OrganizadorPage() {
  const { user, profile, isLoading: userLoading, mutate: mutateUser } = useUser();
  const { openModal } = useAuthModal();
  const [ctaLoading, setCtaLoading] = useState(false);
  const [ctaError, setCtaError] = useState<string | null>(null);
  const [ctaSuccess, setCtaSuccess] = useState<string | null>(null);
  const [stripeCtaLoading, setStripeCtaLoading] = useState(false);
  const [stripeCtaError, setStripeCtaError] = useState<string | null>(null);
  const [billingSaving, setBillingSaving] = useState(false);
  const [billingMessage, setBillingMessage] = useState<string | null>(null);
  const [refundPolicy, setRefundPolicy] = useState<string>("");
  const [vatRate, setVatRate] = useState<string>("");
  const [entityType, setEntityType] = useState<string>("");
  const [businessName, setBusinessName] = useState<string>("");
  const [city, setCity] = useState<string>("");
  const [payoutIban, setPayoutIban] = useState<string>("");
  const [eventStatusFilter, setEventStatusFilter] = useState<"all" | "active" | "draft" | "finished" | "ongoing">("all");
  const [eventTypeFilter, setEventTypeFilter] = useState<string>("all");
  const [eventCategoryFilter, setEventCategoryFilter] = useState<string>("all");
  const [eventDateFilter, setEventDateFilter] = useState<"any" | "today" | "week" | "month" | "weekend">("any");
  const [eventPartnerClubFilter, setEventPartnerClubFilter] = useState<string>("all");
  const [salesEventId, setSalesEventId] = useState<number | null>(null);
  const [searchTerm, setSearchTerm] = useState("");
  const [timeScope, setTimeScope] = useState<"all" | "upcoming" | "ongoing" | "past">("all");
  const [viewMode, setViewMode] = useState<"cards" | "table">("cards");
  const [eventActionLoading, setEventActionLoading] = useState<number | null>(null);
  const [eventDialog, setEventDialog] = useState<{ mode: "archive" | "delete"; ev: EventItem } | null>(null);
  const router = useRouter();
  const pathname = usePathname();
  const searchParams = useSearchParams();
  const [marketingSection, setMarketingSection] = useState<"overview" | "promos" | "promoters" | "content">("overview");
  const [salesRange, setSalesRange] = useState<SalesRange>("30d");
  const salesRangeLabelShort = (range: SalesRange) => {
    switch (range) {
      case "7d":
        return "7d";
      case "30d":
        return "30d";
      case "90d":
        return "3m";
      case "365d":
        return "1a";
      default:
    }
  };
}

```

```

        return "sempre";
    }
};

const salesRangeLabelLong = (range: SalesRange) => {
    switch (range) {
        case "7d":
            return "Últimos 7 dias";
        case "30d":
            return "Últimos 30 dias";
        case "90d":
            return "Últimos 3 meses";
        case "365d":
            return "Último ano";
        default:
            return "Todo o histórico";
    }
};

const tabParam = searchParams?.get("tab") || undefined;
const activeTab: TabKey = ALL_TABS.includes((tabParam as TabKey) || "overview") ? ((tabParam as TabKey) || "overview") : "overview";

useEffect(() => {
    if (typeof window === "undefined") return;
    const stored = window.localStorage.getItem("organizadorFinanceLocal");
    if (stored) {
        try {
            const parsed = JSON.parse(stored) as { refundPolicy?: string; vatRate?: string };
            if (parsed.refundPolicy) setRefundPolicy(parsed.refundPolicy);
            if (parsed.vatRate) setVatRate(parsed.vatRate);
        } catch {
            // ignore invalid
        }
    }
}, []);

// Redirecionar view=categories legacy para a nova página de categorias
useEffect(() => {
    if (!searchParams) return;
    const viewParam = searchParams.get("view");
    const tabParam = searchParams.get("tab");
    if (tabParam === "events" && viewParam === "categories") {
        router.replace("/organizador/categorias", { scroll: false });
    }
}, [router, searchParams]);

useEffect(() => {
    const statusParam = searchParams?.get("status");
    const typeParam = searchParams?.get("type");
    const catParam = searchParams?.get("cat");
    const dateParam = searchParams?.get("date");
    const clubParam = searchParams?.get("club");
    const searchParam = searchParams?.get("search");
    const scopeParam = searchParams?.get("scope");
    const viewParam = searchParams?.get("view");
    const eventIdParam = searchParams?.get("eventId");
    const marketingSectionParam = searchParams?.get("section");

    if (statusParam) setEventStatusFilter(statusParam as typeof eventStatusFilter);
    if (typeParam) setEventTypeFilter(typeParam);
    if (catParam) setEventCategoryFilter(catParam);
    if (dateParam) setEventDateFilter(dateParam as typeof eventDateFilter);
    if (clubParam) setEventPartnerClubFilter(clubParam);
    if (searchParam) setSearchTerm(searchParam);
    if (scopeParam) setTimeScope(scopeParam as typeof timeScope);
    if (viewParam === "table" || viewParam === "cards") setViewMode(viewParam);
    if (eventIdParam) setSalesEventId(Number(eventIdParam));
    if (marketingSectionParam) {
        const allowed = ["overview", "promos", "promoters", "content"] as const;
        if (allowed.includes(marketingSectionParam as (typeof allowed)[number])) {
            setMarketingSection(marketingSectionParam as typeof marketingSection);
        }
    }
}, [searchParams]);

const orgParam = searchParams?.get("org");
const orgMeUrl = useMemo(() => {
    if (!user) return null;

```

```

    return orgParam ? `/api/organizador/me?org=${orgParam}` : "/api/organizador/me";
}, [user, orgParam];

const { data: organizerData, isLoading: organizerLoading, mutate: mutateOrganizer } = useSWR<
  OrganizerStatus & {
  profile?: { fullName?: string | null; city?: string | null } | null;
  organizer?: OrganizerLite | null;
  ok?: boolean;
  orgTransferEnabled?: boolean | null;
}
>(orgMeUrl, fetcher);

const organizer = organizerData?.organizer ?? null;
const loading = userLoading || organizerLoading;
const paymentsStatus = organizerData?.paymentsStatus ?? "NO_STRIPE";
const paymentsMode = organizerData?.paymentsMode ?? "CONNECT";
const profileStatus = organizerData?.profileStatus ?? "MISSING_CONTACT";
const officialEmail = (organizer as { officialEmail?: string | null })?.officialEmail ?? null;
const officialEmailVerifiedAtRaw = (organizer as { officialEmailVerifiedAt?: string | null })?.officialEmailVerifiedAt ?? null;
const officialEmailVerifiedAt = officialEmailVerifiedAtRaw ? new Date(officialEmailVerifiedAtRaw) : null;
const showOfficialEmailWarning = Boolean(organizer) && !officialEmailVerifiedAt;
const onboardingParam = searchParams?.get("onboarding");
const [stripeRequirements, setStripeRequirements] = useState<string>([]);
const [stripeSuccessMessage, setStripeSuccessMessage] = useState<string | null>(null);

useEffect(() => {
  const refreshStripe = async () => {
    try {
      const res = await fetch("/api/organizador/payouts/status");
      const data = await res.json().catch(() => null);
      if (res.ok && data?.status) {
        setStripeRequirements(Array.isArray(data.requirements_due) ? data.requirements_due : []);
        if (data.status === "CONNECTED" && onboardingParam === "done") {
          setStripeSuccessMessage("Conta Stripe ligada. Já podes vender bilhetes pagos.");
          setTimeout(() => setStripeSuccessMessage(null), 3200);
        }
      }
    }
    mutateOrganizer();
  } catch (err) {
    console.error("[stripe][refresh-status] err", err);
  }
};
if (activeTab === "finance") {
  refreshStripe();
}
}, [onboardingParam, activeTab, mutateOrganizer]);

// Prefill onboarding fields quando já existirem dados
useEffect(() => {
  if (!businessName && profile?.fullName) setBusinessName(profile.fullName);
  if (!city && profile?.city) setCity(profile.city);
  if (organizer) {
    if (!entityType && organizer.entityType) setEntityType(organizer.entityType);
    if (!businessName && organizer.displayName) setBusinessName(organizer.displayName);
    if (!city && organizer.city) setCity(organizer.city);
    if (!payoutIban && organizer.payoutIban) setPayoutIban(organizer.payoutIban);
  }
}, [organizer, profile, businessName, city, entityType, payoutIban]);

const { data: overview } = useSWR<OverviewResponse>(
  organizer?.status === "ACTIVE" ? "/api/organizador/estatisticas/overview?range=30d" : null,
  fetcher,
  { revalidateOnFocus: false }
);

type TimeSeriesResponse = { ok: boolean; points: TimeSeriesPoint[]; range: { from: string | null; to: string | null } };
const { data: timeSeries } = useSWR<TimeSeriesResponse>(
  organizer?.status === "ACTIVE" ? "/api/organizador/estatisticas/time-series?range=30d" : null,
  fetcher,
  { revalidateOnFocus: false }
);

const {
  data: events,
  error: eventsError,
  isLoading: eventsLoading,
  mutate: mutateEvents,

```

```

} = useSWR<EventsResponse>(
  organizer?.status === "ACTIVE" ? "/api/organizador/events/list" : null,
  fetcher,
  { revalidateOnFocus: false }
);

useEffect(() => {
  if (!salesEventId && events?.items?.length) {
    setSalesEventId(events.items[0].id);
  }
}, [events, salesEventId]);

const { data: payoutSummary } = useSWR<PayoutSummaryResponse>(
  organizer?.status === "ACTIVE" ? "/api/organizador/payouts/summary" : null,
  fetcher,
  { revalidateOnFocus: false }
);
const { data: financeOverview } = useSWR<FinanceOverviewResponse>(
  organizer?.status === "ACTIVE" && activeTab === "finance" ? "/api/organizador/finance/overview" : null,
  fetcher,
  { revalidateOnFocus: false }
);

const oneYearAgoIso = useMemo(() => {
  const d = new Date();
  d.setHours(0, 0, 0, 0);
  d.setDate(d.getDate() - 365);
  return d.toISOString();
}, []);

const salesSeriesKey = useMemo(() => {
  if (!salesEventId) return null;
  if (salesRange === "7d" || salesRange === "30d" || salesRange === "90d") {
    return `/api/organizador/estatisticas/time-series?range=${salesRange}&eventId=${salesEventId}`;
  }
  if (salesRange === "365d") {
    return `/api/organizador/estatisticas/time-series?eventId=${salesEventId}&from=${oneYearAgoIso}`;
  }
  return `/api/organizador/estatisticas/time-series?eventId=${salesEventId}`;
}, [salesEventId, salesRange, oneYearAgoIso]);

const { data: salesSeries } = useSWR<TimeSeriesResponse>(
  salesSeriesKey,
  fetcher,
  { revalidateOnFocus: false }
);

const { data: buyers } = useSWR<BuyersResponse>(
  salesEventId ? `/api/organizador/estatisticas/buyers?eventId=${salesEventId}` : null,
  fetcher,
  { revalidateOnFocus: false }
);

const archiveEvent = useCallback(
  async (target: EventItem, mode: "archive" | "delete") => {
    setEventActionLoading(target.id);
    setCtaError(null);
    try {
      const res = await fetch("/api/organizador/events/update", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ eventId: target.id, archive: true })
      });
      const json = await res.json().catch(() => null);
      if (!res.ok || json?.ok === false) {
        setCtaError(json?.error || "Não foi possível concluir esta ação.");
      } else {
        mutateEvents();
        setCtaSuccess(mode === "delete" ? "Rascunho apagado." : "Evento arquivado.");
        trackEvent(mode === "delete" ? "event_draft_deleted" : "event_archived", {
          eventId: target.id,
          status: target.status,
        });
        setTimeout(() => setCtaSuccess(null), 3000);
      }
    } catch (err) {
      console.error("[events][archive]", err);
      setCtaError("Erro inesperado ao processar a ação.");
    }
  }
);

```

```

    } finally {
      setEventActionLoading(null);
      setEventDialog(null);
    }
  },
  [mutateEvents],
);
const { data: marketingOverview } = useSWR<MarketingOverviewResponse>(
  organizer?.status === "ACTIVE" && activeTab === "marketing" ? "/api/organizador/marketing/overview" : null,
  fetcher,
  { revalidateOnFocus: false }
);

const [marketingFilters, setMarketingFilters] = useState({ eventId: "all", status: "all" as "all" | "active" | "inactive" });
const { data: promoData, mutate: mutatePromos } = useSWR<PromoListResponse>(
  organizer?.status === "ACTIVE" ? "/api/organizador/promo" : null,
  fetcher,
  { revalidateOnFocus: false }
);
const { data: audienceSummary } = useSWR<AudienceSummaryResponse>(
  organizer?.status === "ACTIVE" && activeTab === "marketing" ? "/api/organizador/marketing/audience/summary" : null,
  fetcher,
  { revalidateOnFocus: false }
);
const { data: padelClubs } = useSWR<{ ok: boolean; items: any[] }>(
  organizer?.status === "ACTIVE" && activeTab === "padel" ? "/api/padel/clubs" : null,
  fetcher,
  { revalidateOnFocus: false }
);
const { data: padelPlayers } = useSWR<{ ok: boolean; items: any[] }>(
  organizer?.status === "ACTIVE" && activeTab === "padel" ? "/api/padel/players" : null,
  fetcher,
  { revalidateOnFocus: false }
);

const currentQuery = searchParams?.toString() || "";

useEffect(() => {
  const params = new URLSearchParams(currentQuery);
  const setParam = (key: string, value: string, defaultVal: string) => {
    if (!value || value === defaultVal) params.delete(key);
    else params.set(key, value);
  };
  setParam("status", eventStatusFilter, "all");
  setParam("type", eventTypeFilter, "all");
  setParam("cat", eventCategoryFilter, "all");
  setParam("date", eventDateFilter, "any");
  setParam("club", eventPartnerClubFilter, "all");
  setParam("search", searchTerm, "");
  setParam("scope", timeScope, "all");
  setParam("view", viewMode, "cards");
  const qs = params.toString();
  if (qs !== currentQuery) {
    router.replace(qs ? `${pathname}?${qs}` : pathname, { scroll: false });
  }
}, [
  eventCategoryFilter,
  eventDateFilter,
  eventStatusFilter,
  eventTypeFilter,
  pathname,
  router,
  searchTerm,
  timeScope,
  viewMode,
  currentQuery,
]);

async function handleBecomeOrganizer() {
  if (!user) {
    openModal({ mode: "login", redirectTo: "/organizador", showGoogle: true });
    return;
  }
  setCtaSuccess(null);
  if (!entityType.trim() || !businessName.trim() || !city.trim()) {
    setCtaError("Preenche tipo de entidade, nome e cidade.");
    return;
}

```

```

if (payoutIban && payoutIban.length < 10) {
  setCtaError("IBAN inválido. Deixa vazio ou insere um IBAN válido.");
  return;
}
setCtaLoading(true);
setCtaError(null);
try {
  const res = await fetch("/api/organizador/become", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      entityType,
      businessName,
      city,
      payoutIban,
    })),
  const data = await res.json().catch(() => null);
  if (!res.ok || data?.ok === false) {
    setCtaError(data?.error || "Não foi possível ativar a conta de organizador.");
    setCtaLoading(false);
    return;
  }
  await mutateOrganizer();
  await mutateUser();
  setCtaSuccess("Conta de organizador ativa. Podes começar a criar eventos.");
} catch (err) {
  console.error("Erro inesperado ao tornar organizador", err);
  setCtaError("Erro inesperado ao ativar conta de organizador.");
} finally {
  setCtaLoading(false);
}
}

async function handleStripeConnect() {
  import("@/lib/analytics").then(({ trackEvent }) =>
  trackEvent("connect_stripe_clicked", { status: paymentsStatus }),
);
setStripeCtaError(null);
setStripeCtaLoading(true);
try {
  const res = await fetch("/api/organizador/payouts/connect", { method: "POST" });
  const json = await res.json().catch(() => null);
  if (!res.ok || !json?.ok || !json.url) {
    setStripeCtaError(json?.error || "Não foi possível gerar o link de onboarding.");
    setStripeCtaLoading(false);
    return;
  }
  window.location.href = json.url;
} catch (err) {
  console.error(err);
  setStripeCtaError("Erro inesperado ao gerar link de onboarding.");
  setStripeCtaLoading(false);
}
}

async function handleSaveBilling() {
  setBillingMessage(null);
  setBillingSaving(true);
  try {
    const res = await fetch("/api/organizador/me", {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        businessName,
        entityType,
        city,
        payoutIban,
      })),
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {
      setBillingMessage(json?.error || "Não foi possível guardar os dados de faturaçāo.");
    } else {
      setBillingMessage("Dados de faturaçāo guardados.");
      await mutateOrganizer();
    }
  }
}

```

```

    } catch (err) {
      console.error("[finance] guardar faturação", err);
      setBillingMessage("Erro inesperado ao guardar os dados.");
    } finally {
      setBillingSaving(false);
    }
  }

const handleSavePolicy = useCallback(() => {
  if (typeof window === "undefined") return;
  const payload = { refundPolicy, vatRate };
  window.localStorage.setItem("organizadorFinanceLocal", JSON.stringify(payload));
  setBillingMessage("Política e IVA guardados localmente.");
}, [refundPolicy, vatRate]);

const statsCards = useMemo(() => {
  const grossEuros = (overview?.grossCents ?? overview?.totalRevenueCents ?? 0) / 100;
  const netEuros = (overview?.netRevenueCents ?? overview?.totalRevenueCents ?? 0) / 100;
  const discountEuros = (overview?.discountCents ?? 0) / 100;
  const feeEuros = (overview?.platformFeeCents ?? 0) / 100;
  return [
    {
      label: "Bilhetes 30d",
      value: overview ? overview.totalTickets : "-",
      hint: "Bilhetes vendidos nos últimos 30 dias",
    },
    {
      label: "Receita líquida 30d",
      value: overview ? `${netEuros.toFixed(2)}€` : "-",
      hint: overview
        ? `Bruto ${grossEuros.toFixed(2)}€ - Descontos ${discountEuros.toFixed(2)}€ - Taxas ${feeEuros.toFixed(2)}€`
        : "-",
    },
    {
      label: "Eventos com vendas",
      value: overview ? overview.eventsWithSalesCount : "-",
      hint: "Eventos com pelo menos 1 venda",
    },
    {
      label: "Eventos publicados",
      value: overview ? overview.activeEventsCount : "-",
      hint: "Eventos PUBLISHED ligados a ti",
    },
  ];
}, [overview]);

const marketingCards = [
  "Sem mensalidades. Pagas só quando vendes – defines se absorves ou passas a taxa ao cliente.",
  "Pagamentos seguros via Stripe; dinheiro direto na tua conta, com proteção anti-fraude.",
  "Bilhetes digitais com QR e validação em tempo real no check-in.",
  "Staff e acessos rápidos: dá permissões de check-in sem partilhar a conta.",
];

const containerClasses = "mx-auto max-w-7xl px-4 pb-12 pt-6 md:pt-8 md:px-6 lg:px-8";
const eventsList = useMemo(() => events?.items ?? [], [events]);
const eventsListLoading = organizer?.status === "ACTIVE" && activeTab === "events" && !events;
const overviewLoading = organizer?.status === "ACTIVE" && !overview;
const partnerClubOptions = useMemo(() => {
  const map = new Map<number, string>();
  eventsList.forEach((ev) => {
    if (ev.templateType !== "SPORT" && ev.templateType !== "PADEL") return;
    if (Number.isFinite(ev.padelClubId as number)) {
      map.set(ev.padelClubId as number, ev.padelClubName || `Clube ${ev.padelClubId}`);
    }
  });
  (ev.padelPartnerClubIds || []).forEach((id, idx) => {
    if (!Number.isFinite(id)) return;
    const label = ev.padelPartnerClubNames?.[idx] || `Clube ${id}`;
    map.set(id as number, label);
  });
  return Array.from(map.entries()).map(([id, name]) => ({ id, name }));
}, [eventsList]);
const persistFilters = useCallback(
  (params: URLSearchParams) => {
    const paramString = params.toString();
    if (paramString !== currentQuery) {
      router.replace(paramString ? `${pathname}?${paramString}` : pathname, { scroll: false });
    }
  }
);

```

```

const payload = {
  status: eventStatusFilter,
  type: eventTypeFilter,
  cat: eventCategoryFilter,
  date: eventDateFilter,
  club: eventPartnerClubFilter,
  search: searchTerm,
  scope: timeScope,
  view: viewMode,
  section: marketingSection,
};

if (typeof window !== "undefined") {
  localStorage.setItem("organizadorFilters", JSON.stringify(payload));
}

},
[
  eventCategoryFilter,
  eventDateFilter,
  eventPartnerClubFilter,
  eventStatusFilter,
  eventTypeFilter,
  pathname,
  router,
  searchTerm,
  timeScope,
  viewMode,
  marketingSection,
  currentQuery,
]
);

useEffect(() => {
  const params = new URLSearchParams(currentQuery);
  const setParam = (key: string, value: string, defaultVal: string) => {
    if (!value || value === defaultVal) params.delete(key);
    else params.set(key, value);
  };

  setParam("status", eventStatusFilter, "all");
  setParam("type", eventTypeFilter, "all");
  setParam("cat", eventCategoryFilter, "all");
  setParam("date", eventDateFilter, "any");
  setParam("club", eventPartnerClubFilter, "all");
  setParam("search", searchTerm, "");
  setParam("scope", timeScope, "all");
  setParam("view", viewMode, "cards");
  setParam("section", marketingSection, "overview");
  if (salesEventId) params.set("eventId", String(salesEventId));
  else params.delete("eventId");
  persistFilters(params);
}, [
  eventCategoryFilter,
  eventDateFilter,
  eventPartnerClubFilter,
  eventStatusFilter,
  eventTypeFilter,
  marketingSection,
  persistFilters,
  salesEventId,
  searchTerm,
  timeScope,
  viewMode,
  currentQuery,
]);

useEffect(() => {
  if (typeof window === "undefined") return;
  if (searchParams?.toString()) return;
  const saved = localStorage.getItem("organizadorFilters");
  if (!saved) return;
  try {
    const parsed = JSON.parse(saved) as {
      status?: string;
      type?: string;
      cat?: string;
      date?: string;
      club?: string;
      search?: string;
      scope?: string;
      view?: string;
    };
    eventCategoryFilter(parsed.cat);
    eventDateFilter(parsed.date);
    eventPartnerClubFilter(parsed.club);
    eventStatusFilter(parsed.status);
    eventTypeFilter(parsed.type);
    marketingSection(parsed.section);
    persistFilters(parsed);
  }
});

```

```

    section?: string;
};

if (parsed.status) setEventStatusFilter(parsed.status as typeof eventStatusFilter);
if (parsed.type) setEventTypeFilter(parsed.type);
if (parsed.cat) setEventCategoryFilter(parsed.cat);
if (parsed.date) setEventDateFilter(parsed.date as typeof eventDateFilter);
if (parsed.club) setEventPartnerClubFilter(parsed.club);
if (parsed.search) setSearchTerm(parsed.search);
if (parsed.scope) setTimeScope(parsed.scope as typeof timeScope);
if (parsed.view === "cards" || parsed.view === "table") setViewMode(parsed.view);
if (
  parsed.section &&
  ["overview", "campaigns", "audience", "promoters", "content", "automation"].includes(parsed.section)
) {
  setMarketingSection(parsed.section as typeof marketingSection);
}
} catch {
  // ignore parse errors
}
}, [searchParams]);
const upcomingCount = useMemo(
() =>
  eventsList.filter((ev) => {
    const start = ev.startsAt ? new Date(ev.startsAt) : null;
    return ev.status === "PUBLISHED" && start && start.getTime() > Date.now();
  }).length,
[eventsList]
);
const ongoingCount = useMemo(
() =>
  eventsList.filter((ev) => {
    const start = ev.startsAt ? new Date(ev.startsAt) : null;
    const end = ev.endsAt ? new Date(ev.endsAt) : null;
    const now = Date.now();
    return ev.status === "PUBLISHED" && start && end && start.getTime() <= now && now <= end.getTime();
  }).length,
[eventsList]
);
const finishedCount = useMemo(
() =>
  eventsList.filter((ev) => {
    const end = ev.endsAt ? new Date(ev.endsAt) : null;
    return ev.status === "PUBLISHED" && end && end.getTime() < Date.now();
  }).length,
[eventsList]
);
const filteredEvents = useMemo(() => {
  const now = new Date();
  const search = searchTerm.trim().toLowerCase();
  return eventsList.filter((ev) => {
    const startsAt = ev.startsAt ? new Date(ev.startsAt) : null;
    const endsAt = ev.endsAt ? new Date(ev.endsAt) : null;
    const isFinished = endsAt ? endsAt.getTime() < now.getTime() : false;
    const isFuture = startsAt ? startsAt.getTime() >= now.getTime() : false;
    const isOngoing = startsAt && endsAt ? startsAt.getTime() <= now.getTime() && now.getTime() <= endsAt.getTime() : false;

    if (eventStatusFilter === "draft" && ev.status !== "DRAFT") return false;
    if (eventStatusFilter === "active" && !(ev.status === "PUBLISHED" && isFuture)) return false;
    if (eventStatusFilter === "finished" && !isFinished) return false;
    if (eventStatusFilter === "ongoing" && !isOngoing) return false;

    if (eventTypeFilter !== "all" && ev.templateType !== eventTypeFilter) return false;
    if (eventCategoryFilter !== "all") {
      const cats = ev.categories ?? [];
      if (!cats.includes(eventCategoryFilter)) return false;
    }
    if (eventPartnerClubFilter !== "all") {
      const clubId = Number(eventPartnerClubFilter);
      if (!Number.isFinite(clubId)) {
        const partners = ev.padelPartnerClubIds ?? [];
        const mainClub = ev.padelClubId ?? null;
        if (mainClub !== clubId && !partners.includes(clubId)) return false;
      }
    }
    if (search) {
      if (!ev.title.toLowerCase().includes(search)) return false;
    }
  })
});

```

```

    if (eventDateFilter !== "any" && startsAt) {
      const diffDays = (startsAt.getTime() - now.getTime()) / (1000 * 60 * 60 * 24);
      if (eventDateFilter === "today" && Math.floor(diffDays) !== 0) return false;
      if (eventDateFilter === "week" && diffDays > 7) return false;
      if (eventDateFilter === "month" && diffDays > 31) return false;
      if (eventDateFilter === "weekend") {
        const day = startsAt.getDay();
        if (day !== 6 && day !== 0) return false;
      }
    }

    if (timeScope === "upcoming" && !isFuture) return false;
    if (timeScope === "ongoing" && !isOngoing) return false;
    if (timeScope === "past" && !isFinished) return false;

    return true;
  });
}, [eventCategoryFilter, eventDateFilter, eventStatusFilter, eventTypeFilter, eventsList, searchTerm, timeScope]);
const activeFilterCount = useMemo(() =>
  [
    eventStatusFilter !== "all",
    eventTypeFilter !== "all",
    eventCategoryFilter !== "all",
    eventDateFilter !== "any",
    eventPartnerClubFilter !== "all",
    timeScope !== "all",
    searchTerm.trim() !== "",
    .filter(Boolean).length,
  ],
  [eventCategoryFilter, eventDateFilter, eventPartnerClubFilter, eventStatusFilter, eventTypeFilter, searchTerm, timeScope]
);

const selectedSalesEvent = salesEventId ? eventsList.find((ev) => ev.id === salesEventId) ?? null : null;
const financeData = financeOverview && financeOverview.ok ? financeOverview : null;
const financeSummary = payoutSummary && "ok" in payoutSummary && payoutSummary.ok ? payoutSummary : null;
const stripeState = useMemo(() => {
  const hasReqs = stripeRequirements.length > 0;
  if (paymentsStatus === "READY") {
    return { badge: "Ativo", tone: "success", title: "Conta Stripe ligada ✅", desc: "Já podes vender bilhetes pagos e receber os teus payouts normalmente.", cta: "Abrir painel Stripe" };
  }
  if (paymentsStatus === "PENDING") {
    return {
      badge: hasReqs ? "Requer atenção" : "Onboarding incompleto",
      tone: hasReqs ? "error" : "warning",
      title: hasReqs ? "Falta concluir dados no Stripe" : "Conta Stripe em configuração",
      desc: hasReqs
        ? "A tua conta Stripe precisa de dados antes de ativar pagamentos."
        : "Conclui o onboarding no Stripe para começas a receber os pagamentos dos teus bilhetes.",
      cta: hasReqs ? "Rever ligação Stripe" : "Continuar configuração no Stripe",
    };
  }
  return { badge: "Por ligar", tone: "neutral", title: "Ainda não ligaste a tua conta Stripe", desc: "Podes criar eventos gratuitos, mas para vender bilhetes pagos precisas de ligar uma conta Stripe.", cta: "Ligar conta Stripe" };
}, [paymentsStatus, stripeRequirements]);

const marketingPromos = useMemo(() => promoData?.promoCodes ?? [], [promoData]);
const marketingEvents = useMemo(() => promoData?.events ?? [], [promoData]);
const filteredPromos = useMemo(() => {
  return marketingPromos.filter((p) => {
    if (marketingFilters.eventId !== "all" && `${p.eventId ?? "global"}` !== marketingFilters.eventId) return false;
    if (marketingFilters.status === "active" && !p.active) return false;
    if (marketingFilters.status === "inactive" && p.active) return false;
    return true;
  });
}, [marketingFilters.eventId, marketingFilters.status, marketingPromos]);
const marketingKpis = useMemo(() => {
  const activePromos = marketingPromos.filter((p) => p.active).length;
  const fallbackTop = [...marketingPromos].sort(
    (a, b) => (b.redemptionsCount ?? 0) - (a.redemptionsCount ?? 0)
  )[0];
  return {
    totalTickets: marketingOverview?.totalTickets ?? overview?.totalTickets ?? 0,
    ticketsWithPromo: marketingOverview?.ticketsWithPromo ?? marketingPromos.reduce((sum, p) => sum + (p.redemptionsCount ?? 0), 0),
    guestTickets: marketingOverview?.guestTickets ?? 0,
    marketingRevenueCents: marketingOverview?.marketingRevenueCents ?? 0,
    activePromos,
  };
});

```

```

    topPromo: marketingOverview?.topPromo ?? (fallbackTop
    ? {
        id: fallbackTop.id,
        code: fallbackTop.code,
        redemptionsCount: fallbackTop.redemptionsCount ?? 0,
        revenueCents: 0,
    }
    : null),
};

}, [marketingOverview, marketingPromos, overview]);
const buyersItems = buyers && buyers.ok != false ? buyers.items : [];
const salesLoading = !!salesEventId && !salesSeries;
const buyersLoading = !!salesEventId && !buyers;
const salesKpis = useMemo(() => {
    const tickets = salesSeries?.points?.reduce((sum, p) => sum + p.tickets, 0) ?? 0;
    const revenueCents = salesSeries?.points?.reduce((sum, p) => sum + p.revenueCents, 0) ?? 0;
    const eventsWithSales = tickets > 0 ? 1 : 0;
    const avgOccupancy = () => {
        const capacity = selectedSalesEvent?.capacity ?? null;
        if (!capacity) return null;
        const sold = selectedSalesEvent?.ticketsSold ?? 0;
        return Math.min(100, Math.round((sold / capacity) * 100));
    }();
    return { tickets, revenueCents, eventsWithSales, avgOccupancy };
}, [salesSeries?.points, selectedSalesEvent]);

const topEvents = useMemo(() => {
    return [...eventsList]
        .filter(ev => (ev.revenueCents ?? 0) > 0 || (ev.ticketsSold ?? 0) > 0)
        .sort((a, b) => (b.revenueCents ?? 0) - (a.revenueCents ?? 0) || (b.ticketsSold ?? 0) - (a.ticketsSold ?? 0))
        .slice(0, 5);
}, [eventsList]);

const formatEuros = (val: number) => `${(val / 100).toFixed(2)} €`;

const overviewSeriesBreakdown = useMemo(() => {
    if (!timeSeries?.points?.length) return null;
    const gross = timeSeries.points.reduce((acc, p) => acc + (p.grossCents ?? 0), 0);
    const discount = timeSeries.points.reduce((acc, p) => acc + (p.discountCents ?? 0), 0);
    const fees = timeSeries.points.reduce((acc, p) => acc + (p.platformFeeCents ?? 0), 0);
    const net = timeSeries.points.reduce((acc, p) => acc + (p.netCents ?? p.revenueCents ?? 0), 0);
    return { gross, discount, fees, net };
}, [timeSeries?.points]);

const salesSeriesBreakdown = useMemo(() => {
    if (!salesSeries?.points?.length) return null;
    const gross = salesSeries.points.reduce((acc, p) => acc + (p.grossCents ?? 0), 0);
    const discount = salesSeries.points.reduce((acc, p) => acc + (p.discountCents ?? 0), 0);
    const fees = salesSeries.points.reduce((acc, p) => acc + (p.platformFeeCents ?? 0), 0);
    const net = salesSeries.points.reduce((acc, p) => acc + (p.netCents ?? p.revenueCents ?? 0), 0);
    return { gross, discount, fees, net };
}, [salesSeries?.points]);

const salesChartPoints = useMemo(() => {
    if (!salesSeries?.points?.length) return [];
    return salesSeries.points.map((p) => ({
        date: p.date,
        gross: (p.grossCents ?? p.revenueCents ?? 0) / 100,
        net: (p.netCents ?? p.revenueCents ?? 0) / 100,
        tickets: p.tickets ?? 0,
    }));
}, [salesSeries?.points]);

const exportFinanceCsv = useCallback(() => {
    if (!financeData || !financeData.events.length) return;
    const header = ["ID", "Evento", "Bilhetes", "Bruto (€)", "Taxas (€)", "Líquido (€)", "Estado", "Data"];
    const rows = financeData.events.map((ev) => [
        ev.id,
        ev.title,
        ev.ticketsSold,
        (ev.grossCents / 100).toFixed(2),
        (ev.feesCents / 100).toFixed(2),
        (ev.netCents / 100).toFixed(2),
        ev.status ?? "",
        ev.startsAt ? new Date(ev.startsAt).toLocaleDateString("pt-PT") : "",
    ]);
    const csv = [header.join(";"), ...rows.map((r) => r.join(";"))].join("\n");
    const blob = new Blob([csv], { type: "text/csv" });
    const url = URL.createObjectURL(blob);

```

```

const a = document.createElement("a");
a.href = url;
a.download = "vendas-por-evento.csv";
a.click();
URL.revokeObjectURL(url);
}, [financeData]);

const handleExportSalesCsv = useCallback(() => {
  if (!salesSeries?.points?.length || !selectedSalesEvent) return;
  const header = ["Data", "Bilhetes", "Bruto (€)", "Desconto (€)", "Taxas (€)", "Líquido (€)"];
  const rows = salesSeries.points.map((p) => {
    const date = new Date(p.date).toLocaleDateString("pt-PT");
    const gross = (p.grossCents ?? p.revenueCents ?? 0) / 100;
    const discount = (p.discountCents ?? 0) / 100;
    const fees = (p.platformFeeCents ?? 0) / 100;
    const net = (p.netCents ?? p.revenueCents ?? 0) / 100;
    return [
      date,
      p.tickets,
      gross.toFixed(2),
      (-discount).toFixed(2),
      (-fees).toFixed(2),
      net.toFixed(2),
    ];
  });
  const csv = [header.join(";"), ...rows.map((r) => r.join(";"))].join("\n");
  const blob = new Blob([csv], { type: "text/csv" });
  const url = URL.createObjectURL(blob);
  const a = document.createElement("a");
  a.href = url;
  const rangeLabel = salesRangeLabelShort(salesRange);
  a.download = `vendas-${selectedSalesEvent.title}-${rangeLabel}.csv`;
  a.click();
  URL.revokeObjectURL(url);
}, [salesRange, salesSeries?.points, selectedSalesEvent]);
const fillTheRoomEvents = useMemo(() => {
  const sourceEvents =
    marketingOverview?.events && marketingOverview.events.length > 0 ? marketingOverview.events : eventsList;
  const now = new Date();
  return sourceEvents
    .filter((ev) => {
      const start = ev.startsAt ? new Date(ev.startsAt) : null;
      return start && start.getTime() >= now.getTime();
    })
    .sort((a, b) => (a.startsAt && b.startsAt ? new Date(a.startsAt).getTime() - new Date(b.startsAt).getTime() : 0))
    .slice(0, 6)
    .map((ev) => {
      const start = ev.startsAt ? new Date(ev.startsAt) : null;
      const diffDays = start ? Math.ceil((start.getTime() - now.getTime()) / (1000 * 60 * 60 * 24)) : null;
      const capacity = ev.capacity ?? null;
      const sold = ev.ticketsSold ?? 0;
      const occupancy = capacity ? Math.min(1, sold / capacity) : null;
      let tag: { label: string; tone: string; suggestion: string } = {
        label: "Atenção",
        tone: "border-amber-400/40 bg-amber-400/10 text-amber-100",
        suggestion: "Criar código -10% 48h",
      };
      if (occupancy !== null) {
        if (occupancy >= 0.8) {
          tag = {
            label: "Confortável",
            tone: "border-emerald-400/40 bg-emerald-500/10 text-emerald-100",
            suggestion: "Preparar lista de espera",
          };
        } else if (occupancy < 0.4 && (diffDays ?? 0) <= 7) {
          tag = {
            label: "Crítico",
            tone: "border-red-400/50 bg-red-500/10 text-red-100",
            suggestion: "Last-minute boost",
          };
        } else if ((diffDays ?? 0) <= 5) {
          tag = {
            label: "Sem lotação",
            tone: "border-white/20 bg-white/5 text-white/70",
            suggestion: "Definir capacidade e criar código",
          };
        }
      }
    });
}

```

```

        return { ...ev, diffDays, capacity, occupancy, tag };
    });
}, [eventsList, marketingOverview?.events]);

if (loading) {
    return (
        <div className={`${containerClasses} space-y-6`}>
            <div className="h-8 w-48 rounded-full bg-white/10 animate-pulse" />
            <div className="h-24 rounded-3xl bg-white/5 border border-white/10 animate-pulse" />
        </div>
    );
}

if (organizer?.status !== "ACTIVE") {
    return (
        <div className={`${containerClasses} space-y-8`}>
            <div className="grid gap-6 md:grid-cols-[1.2fr_0.8fr] items-center">
                <div className="space-y-4">
                    <p className="text-[11px] font-semibold uppercase tracking-[0.3em] text-white/60">
                        Organizar com a ORYA
                    </p>
                <div className="space-y-2">
                    <h1 className="text-3xl md:text-4xl font-bold leading-tight">
                        Abre o teu espaço ou evento ao público em minutos.
                    </h1>
                    <p className="text-sm text-white/70 max-w-2xl">
                        Self-serve, sem convites: cria eventos, vende bilhetes e recebe pagamentos. Sem mensalidades nem contratos
                longos.
                    </p>
                </div>
            </div>
            <div className="grid gap-3 md:grid-cols-2 w-full max-w-2xl">
                <div className="space-y-2">
                    <label className="text-xs text-white/70">Tipo de entidade</label>
                    <select
                        value={entityType}
                        onChange={({e}) => setEntityType(e.target.value)}
                        className="w-full rounded-xl bg-black/40 border border-white/15 px-3 py-2 text-sm outline-none focus:border-#6BFFFF focus:ring-1 focus:ring-[#6BFFFF]">
                        <option value="">Seleciona</option>
                        <option value="Clube">Clube</option>
                        <option value="Bar">Bar</option>
                        <option value="Associação">Associação</option>
                        <option value="Freelance">Freelance</option>
                        <option value="Outro">Outro</option>
                    </select>
                </div>
                <div className="space-y-2">
                    <label className="text-xs text-white/70">Nome do espaço/negócio</label>
                    <input
                        type="text"
                        value={businessName}
                        onChange={({e}) => setBusinessName(e.target.value)}
                        placeholder="Ex.: Clube XPTO"
                        className="w-full rounded-xl bg-black/40 border border-white/15 px-3 py-2 text-sm outline-none focus:border-#6BFFFF focus:ring-1 focus:ring-[#6BFFFF]">
                </div>
                <div className="space-y-2">
                    <label className="text-xs text-white/70">Cidade</label>
                    <input
                        type="text"
                        value={city}
                        onChange={({e}) => setCity(e.target.value)}
                        placeholder="Ex.: Lisboa"
                        className="w-full rounded-xl bg-black/40 border border-white/15 px-3 py-2 text-sm outline-none focus:border-#6BFFFF focus:ring-1 focus:ring-[#6BFFFF]">
                </div>
                <div className="space-y-2">
                    <label className="text-xs text-white/70">IBAN para payouts (opcional)</label>
                    <input
                        type="text"
                        value={payoutIban}
                        onChange={({e}) => setPayoutIban(e.target.value)}
                        placeholder="PT50 ...."
                        className="w-full rounded-xl bg-black/40 border border-white/15 px-3 py-2 text-sm outline-none focus:border-#6BFFFF focus:ring-1 focus:ring-[#6BFFFF]">
                </div>
            </div>
        </div>
    );
}

```

```

[#6BFFFF] focus:ring-1 focus:ring-[#6BFFFF]"
    />
  </div>
</div>
<div className="flex flex-wrap gap-2">
  <button
    type="button"
    onClick={handleBecomeOrganizer}
    disabled={ctaLoading}
    className="px-5 py-2.5 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] font-semibold
text-black shadow-lg disabled:opacity-60"
    >
    {ctaLoading ? "A ativar conta..." : "Começar a organizar"}
  </button>
  {!user && (
    <button
      type="button"
      onClick={() =>
        openModal({ mode: "login", redirectTo: "/organizador", showGoogle: true })
      }
      className="px-5 py-2.5 rounded-full border border-white/20 text-white/80 hover:bg-white/10 transition"
    >
      Criar conta / Entrar
    </button>
  )}
</div>
{ctaError && (
  <div className="rounded-xl border border-red-500/40 bg-red-500/10 px-4 py-3 text-sm text-red-100">
    {ctaError}
  </div>
)}
{ctaSuccess && (
  <div className="rounded-xl border border-emerald-500/40 bg-emerald-500/10 px-4 py-3 text-sm text-emerald-100">
    {ctaSuccess}
  </div>
)}
</div>

<div className="rounded-3xl border border-white/10 bg-white/[0.06] backdrop-blur-xl p-5 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.55)]">
  <h3 className="text-sm font-semibold text-white">Porqué a ORYA?</h3>
  <div className="space-y-2 text-sm text-white/80">
    {marketingCards.map((item) =>
      <p
        key={item}
        className="rounded-2xl border border-white/10 bg-black/40 p-3"
      >
        {item}
      </p>
    )));
  </div>
</div>
</div>
</div>
);
}

const isPlatformStripe = paymentsMode === "PLATFORM";
const stripeReady = isPlatformStripe || paymentsStatus === "READY";
const stripeIncomplete = !isPlatformStripe && paymentsStatus === "PENDING";
const quickTasks = [
  {
    label: "Liga Stripe para vender",
    done: stripeReady,
    href: "/organizador?tab=finance",
  },
  {
    label: "Cria o teu primeiro evento",
    done: (events?.items?.length ?? 0) > 0,
    href: "/organizador/eventos/novo",
  },
  {
    label: "Convida staff para check-in",
    done: false,
    href: "/organizador?tab=staff",
  },
];

```

```

const hasIban = Boolean(organizer.payoutIban);
const nextEvent = events?.items?.[0] ?? null;

return (
  <div className={`${containerClasses} space-y-6 text-white`}>
    {showOfficialEmailWarning && (
      <div className="rounded-2xl border border-amber-400/40 bg-amber-500/10 px-4 py-3 text-sm text-amber-50">
        <div className="flex flex-wrap items-center justify-between gap-2">
          <div className="space-y-1">
            <p className="font-semibold">
              {officialEmail
                ? "Email oficial pendente de verificação."
                : "Define o email oficial da organização para faturação e alertas críticos."}
            </p>
            <p className="text-[12px] text-amber-100/80">
              Usamos este email para invoices, alertas de vendas/payouts e transferências de Owner.
            </p>
          </div>
          <Link href="/organizador/settings"
            className="rounded-full bg-white px-3 py-1.5 text-[12px] font-semibold text-black shadow hover:scale-[1.01]"
          >
            Atualizar email oficial
          </Link>
        </div>
      </div>
    )}
  {activeTab !== "overview" && <BackButton className="mb-2" />}
  {activeTab === "overview" && (
    <>
      /* Header + alerta onboarding */
      <div
        className="rounded-3xl border border-white/10 bg-black/40 backdrop-blur-xl p-4 md:p-5 shadow-[0_18px_60px_rgba(0,0,0,0.65)]"
        data-tour="overview"
      >
        <div className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between">
          <div>
            <p className="text-[11px] uppercase tracking-[0.3em] text-white/50">Resumo</p>
            <h1 className="text-3xl font-bold leading-tight">
              Olá, {organizer.displayName || profile?.fullName || "organizador"} 🌟
            </h1>
            <p className="text-sm text-white/70">Aqui está o resumo dos teus eventos e vendas.</p>
          </div>
        </div>
      </div>
      {profileStatus === "MISSING_CONTACT" && (
        <div className="mt-4 rounded-2xl border border-amber-400/40 bg-amber-400/10 p-3 text-sm text-amber-100 space-y-2">
          <div className="flex items-center justify-between gap-2">
            <p className="font-semibold">Completa os dados básicos do organizador (nome, tipo, cidade, email).</p>
            <Link href="/organizador/settings"
              className="rounded-full bg-white/10 px-3 py-1 text-[11px] text-white hover:bg-white/20"
            >
              Preencher dados
            </Link>
          </div>
        </div>
      )}
      {!stripeReady && paymentsMode === "CONNECT" && (
        <div className="mt-4 rounded-2xl border border-amber-400/40 bg-amber-400/10 p-3 text-sm text-amber-100 space-y-2">
          <div className="flex items-center justify-between gap-2">
            <p className="font-semibold">
              Podes publicar eventos gratuitos. Para bilhetes pagos, liga a tua conta Stripe.
            </p>
            <Link href="/organizador?tab=finance"
              className="rounded-full bg-white/10 px-3 py-1 text-[11px] text-white hover:bg-white/20"
            >
              Ligar Stripe
            </Link>
          </div>
        </div>
      )}
      {isPlatformStripe && (
        <div className="mt-4 rounded-2xl border border-emerald-400/40 bg-emerald-500/10 p-3 text-sm text-emerald-50 space-y-1">
          <p className="font-semibold">Conta interna ORYA</p>
        </div>
      )}
    </>
  )}

```

```

<p className="text-white/80 text-xs">
  Este organizador usa a conta Stripe principal da ORYA. Não é necessário onboarding em Connect.
</p>
</div>
)}
<div className="mt-4 grid gap-3 md:grid-cols-2">
  <Link
    href="/organizador/scan"
    className="rounded-2xl border border-white/15 bg-white/5 p-4 shadow-[0_18px_50px_rgba(0,0,0,0.6)] transition
    hover:border-white/30 hover:bg-white/10"
  >
    <p className="text-[11px] uppercase tracking-[0.24em] text-white/50">Check-in rápido</p>
    <p className="text-lg font-semibold">Abrir scanner</p>
    <p className="text-sm text-white/70">Valida bilhetes com feedback imediato. Otimizado para telemóvel.</p>
  </Link>
  <Link
    href="/organizador/staff"
    className="rounded-2xl border border-white/15 bg-white/5 p-4 shadow-[0_18px_50px_rgba(0,0,0,0.6)] transition
    hover:border-white/30 hover:bg-white/10"
  >
    <p className="text-[11px] uppercase tracking-[0.24em] text-white/50">Equipa & acessos</p>
    <p className="text-lg font-semibold">Gerir staff</p>
    <p className="text-sm text-white/70">Convida staff e controla quem pode fazer check-in.</p>
  </Link>
</div>
</div>

<div className="grid gap-4 xl:grid-cols-4 md:grid-cols-2">
  {overviewLoading
    ? [...Array(4)].map(_, idx) =>
      <div
        key={idx}
        className="rounded-3xl border border-white/10 bg-white/5 p-4 shadow-[0_18px_50px_rgba(0,0,0,0.6)] animate-
        pulse space-y-2"
      >
        <div className="h-3 w-24 rounded bg-white/15" />
        <div className="h-6 w-20 rounded bg-white/20" />
        <div className="h-3 w-32 rounded bg-white/10" />
      </div>
    )
    : statsCards.map((card, idx) => (
      <div
        key={card.label}
        className="rounded-3xl border border-white/10 bg-white/5 p-4 shadow-[0_18px_50px_rgba(0,0,0,0.6)]"
      >
        <p className="text-white/60 text-xs">{card.label}</p>
        <p className="text-2xl font-bold text-white mt-1">{card.value}</p>
        <p className="text-[11px] text-white/45">{card.hint}</p>
        {idx === 0 && nextEvent && (
          <Link
            href={`/eventos/${nextEvent.slug}`}
            className="mt-2 inline-flex text-[11px] text-[#6BFFFF] hover:underline"
          >
            Ver evento →
          </Link>
        )}
      </div>
    )))
  </div>

<div className="grid gap-4 lg:grid-cols-[1.2fr_0.8fr]">
  <div className="rounded-3xl border border-white/10 bg-black/40 p-4 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.65)]">
    <div className="flex items-center justify-between">
      <h3 className="text-lg font-semibold">Vendas ao longo do tempo</h3>
      <div className="rounded-full border border-white/15 bg-white/5 px-3 py-1 text-[11px] text-white/70">
        Receita • 30 dias
      </div>
    </div>
    <div className="h-48 rounded-2xl border border-white/10 bg-gradient-to-br from-white/5 to-white/0 shadow-inner
    overflow-hidden px-2 py-3">
      {timeSeries?.points?.length ? (
        <SalesLineChart
          className="h-full"
          data={timeSeries.points.map((p) => ({ date: p.date, value: p.revenueCents / 100 }))}
          unit="eur"
          periodLabel="Últimos 30 dias"
          metricLabel="Receita"
          rangeDays={30}
      ) : null}
    </div>
  </div>
</div>

```

```

        />
    ) : (
      <span className="text-white/40 text-xs">Sem dados suficientes.</span>
    )}
</div>
{overviewSeriesBreakdown && (
  <div className="flex flex-wrap gap-3 text-[11px] text-white/70">
    <span>Bruto: {formatEuros(overviewSeriesBreakdown.gross)}</span>
    <span>Desconto: -{formatEuros(overviewSeriesBreakdown.discount)}</span>
    <span>Taxas: -{formatEuros(overviewSeriesBreakdown.fees)}</span>
    <span>Líquido: {formatEuros(overviewSeriesBreakdown.net)}</span>
  </div>
)
)}
</div>

<div className="rounded-3xl border border-white/10 bg-black/30 p-4 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.65)]">
  <div className="flex items-center justify-between">
    <h3 className="text-lg font-semibold">Próximos passos</h3>
    <span className="text-[11px] text-white/60">{quickTasks.filter((t) => t.done).length}/{quickTasks.length}</span>
  </div>
  <div className="space-y-2 text-[12px]">
    {quickTasks.map((task) => (
      <Link
        key={task.label}
        href={task.href}
        className={`flex items-center justify-between rounded-2xl border px-3 py-2 transition ${task.done ? "border-emerald-400/25 bg-emerald-400/10 text-emerald-100" : "border-white/12 bg-white/5 text-white/80 hover:bg-white/10"}`}
      >
        <span>{task.label}</span>
        <span className={`text-[11px] rounded-full px-2 py-0.5 ${task.done ? "bg-emerald-400/20" : "bg-white/10"}`}>
          {task.done ? "Feito" : "Ir"}
        </span>
      </Link>
    )))
  </div>
  <div className="rounded-2xl border border-white/10 bg-white/5 p-3 text-[11px] text-white/70">
    Organiza-te: cria eventos, ativa promo codes e convida staff. Tudo começa aqui.
  </div>
</div>
</div>

<div className="rounded-3xl border border-white/10 bg-black/40 backdrop-blur-xl p-4 md:p-5 shadow-[0_18px_60px_rgba(0,0,0,0.65)] space-y-3">
  <div className="flex items-center justify-between gap-3">
    <div>
      <h3 className="text-lg font-semibold text-white">Os teus eventos</h3>
      <p className="text-[11px] text-white/60">Próximos e passados ligados à tua conta de organizador.</p>
    </div>
  </div>
  <div className="space-y-2">
    {!events?.items && (
      <div className="grid gap-2 md:grid-cols-2">
        {[1, 2].map((i) => (
          <div key={i} className="h-24 rounded-2xl border border-white/10 bg-white/5 animate-pulse" />
        )))
      </div>
    )}
    {events?.items?.length === 0 && (
      <div className="rounded-2xl border border-dashed border-white/15 bg-white/5 px-4 py-3 text-white/70">
        Ainda não tens eventos. Cria o primeiro e começa a vender.
      </div>
    )}
    {events?.items && events.items.length > 0 && (
      <div className="grid gap-3 md:grid-cols-2">
        {events.items.slice(0, 6).map((ev) => {
          const date = ev.startsAt ? new Date(ev.startsAt) : null;
          const ticketsSold = ev.ticketsSold ?? 0;
          const revenue = ((ev.revenueCents ?? 0) / 100).toFixed(2);
          const dateLabel = date
            ? date.toLocaleString("pt-PT", {
              day: "2-digit",
              month: "short",
              hour: "2-digit",
            })
            : "Ainda não tens eventos. Cria o primeiro e começa a vender."
        ))}
      </div>
    )}
  </div>
</div>

```

```

        minute: "2-digit",
    })
    : "Data a confirmar";
return (
    <div
        key={ev.id}
        className="rounded-2xl border border-white/12 bg-white/5 p-3 flex flex-col gap-2"
    >
        <div className="flex items-center justify-between gap-2">
            <div className="flex flex-col">
                <p className="text-sm font-semibold text-white line-clamp-2">{ev.title}</p>
                <p className="text-[11px] text-white/60">{dateLabel}</p>
                <p className="text-[11px] text-white/60">
                    {ev.locationName || ev.locationCity || "Local a anunciar"}
                </p>
                <p className="text-[11px] text-white/60">
                    {ticketsSold} bilhetes · {revenue} €
                </p>
            </div>
            <span className="rounded-full border border-white/20 px-2 py-0.5 text-[10px] text-white/80">
                {ev.status}
            </span>
        </div>
        <div className="flex flex-wrap gap-2 text-[11px]">
            <Link
                href={`/organizador/eventos/${ev.id}/edit`}
                className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
            >
                Editar
            </Link>
            <Link
                href={`/eventos/${ev.slug}`}
                className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
            >
                Página pública
            </Link>
            <Link
                href={`/organizador?tab=sales&eventId=${ev.id}`}
                className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
            >
                Vendas
            </Link>
        </div>
    </div>
);
});
</div>
</div>
</div>
)
}

{activeTab === "events" && (
<section className="space-y-4">
    {/* Header + ação */}
    <div className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between">
        <div className="space-y-1">
            <h2 className="text-2xl font-semibold">Gestão de eventos</h2>
            <p className="text-sm text-white/65">Vê os teus eventos, filtra o que precisas e cria novos em segundos.</p>
        </div>
        <div className="flex flex-col items-start gap-2 sm:flex-row sm:items-center">
            <input
                type="search"
                placeholder="Procurar por evento..."
                value={searchTerm}
                onChange={(e) => {
                    const val = e.target.value;
                    const normalized = val.toLowerCase();
                    setSearchTerm(val);
                    if (normalized.includes("padel") || normalized.includes("pádel")) setEventTypeFilter("SPORT");
                    if (normalized.includes("jantar") || normalized.includes("restaurante")) setEventTypeFilter("COMIDA");
                    if (normalized.includes("solid")) setEventTypeFilter("VOLUNTEERING");
                    if (normalized.includes("festa")) setEventTypeFilter("PARTY");
                }}
            >
            <span className="w-full rounded-full border border-white/15 bg-black/40 px-3 py-2 text-sm text-white outline-none focus:border-[#6BFFFF] sm:min-w-[260px]">
            </span>
        </div>
    </div>
)
}

```

```

    </div>
</div>

/* Filtros em faixa única */

```

```

        <option value="CONCERTO">Concertos</option>
        <option value="DRINKS">Drinks</option>
    </select>
    <select
        value={eventDateFilter}
        onChange={(e) => setEventDateFilter(e.target.value as typeof eventDateFilter)}
        className={`w-full rounded-xl border px-3 py-2 text-[12px] text-white outline-none focus:border-[#6BFFFF] ${eventDateFilter !== "any" ? "border-[#6BFFFF]/60 bg-[#0b1224]" : "border-white/15 bg-black/40"}`}
    >
        <option value="any">Datas: Qualquer</option>
        <option value="today">Hoje</option>
        <option value="weekend">Este fim-de-semana</option>
        <option value="week">Esta semana</option>
        <option value="month">Este mês</option>
    </select>
    <select
        value={eventPartnerClubFilter}
        onChange={(e) => setEventPartnerClubFilter(e.target.value)}
        className={`w-full rounded-xl border px-3 py-2 text-[12px] text-white outline-none focus:border-[#6BFFFF] ${eventPartnerClubFilter !== "all" ? "border-[#6BFFFF]/60 bg-[#0b1224]" : "border-white/15 bg-black/40"}`}
    >
        <option value="all">Clube parceiro: Todos</option>
        {partnerClubOptions.map((opt) => (
            <option key={opt.id} value={opt.id}>
                {opt.name}
            </option>
        )))
    </select>
</div>
</div>

<div className="flex flex-wrap items-center gap-2 text-[12px] text-white/70">
    <span className="rounded-full border border-white/15 px-2 py-0.5">Vista:</span>
    <button
        type="button"
        onClick={() => setViewMode("cards")}
        className={`rounded-full px-3 py-1 transition ${viewMode === "cards" ? "bg-white text-black font-semibold shadow" : "border border-white/20 hover:border-white/30"}`}
    >
        Cartões
    </button>
    <button
        type="button"
        onClick={() => setViewMode("table")}
        className={`rounded-full px-3 py-1 transition ${viewMode === "table" ? "bg-white text-black font-semibold shadow" : "border border-white/20 hover:border-white/30"}`}
    >
        Tabela
    </button>
<div className="h-4 w-px bg-white/20" />
    <span className="rounded-full border border-white/15 px-2 py-0.5">Mostrar:</span>
{[
    { key: "all", label: "Todos" },
    { key: "upcoming", label: "Próximos" },
    { key: "ongoing", label: "A decorrer" },
    { key: "past", label: "Passados" },
] as const).map((opt) => (
    <button
        key={opt.key}
        type="button"
        onClick={() => setTimeScope(opt.key)}
        className={`rounded-full px-3 py-1 transition ${timeScope === opt.key ? "bg-[#6BFFFF]/20 border-[#6BFFFF]/40 text-white" : "border border-white/20 text-white/70 hover:border-white/30"}`}
    >
        {opt.label}
    </button>
))
</div>
{[
```

```

eventStatusFilter !== "all",
eventTypeFilter !== "all",
eventCategoryFilter !== "all",
eventDateFilter !== "any",
timeScope !== "all",
searchTerm.trim() !== "",
].some(Boolean) && (
<div className="flex flex-wrap items-center gap-2 rounded-2xl border border-white/10 bg-white/5 px-3 py-2 text-[12px] text-white/80 shadow-[0_10px_30px_rgba(0,0,0,0.35)]">
  <span className="text-white/70 font-semibold">Filtros ativos</span>
  {eventStatusFilter !== "all" && (
    <button
      type="button"
      onClick={() => setEventStatusFilter("all")}
      className="inline-flex items-center gap-1 rounded-full border border-white/25 bg-white/10 px-2 py-0.5
      hover:border-white/40"
    >
      Estado: {eventStatusFilter} ×
    </button>
  )}
  {eventTypeFilter !== "all" && (
    <button
      type="button"
      onClick={() => setEventTypeFilter("all")}
      className="inline-flex items-center gap-1 rounded-full border border-white/25 bg-white/10 px-2 py-0.5
      hover:border-white/40"
    >
      Tipo: {eventTypeFilter} ×
    </button>
  )}
  {eventCategoryFilter !== "all" && (
    <button
      type="button"
      onClick={() => setEventCategoryFilter("all")}
      className="inline-flex items-center gap-1 rounded-full border border-white/25 bg-white/10 px-2 py-0.5
      hover:border-white/40"
    >
      Categória: {eventCategoryFilter} ×
    </button>
  )}
  {eventPartnerClubFilter !== "all" && (
    <button
      type="button"
      onClick={() => setEventPartnerClubFilter("all")}
      className="inline-flex items-center gap-1 rounded-full border border-white/25 bg-white/10 px-2 py-0.5
      hover:border-white/40"
    >
      Clube: {partnerClubOptions.find((o) => `${o.id}` === eventPartnerClubFilter)?.name ?? eventPartnerClubFilter} ×
    </button>
  )}
  {eventDateFilter !== "any" && (
    <button
      type="button"
      onClick={() => setEventDateFilter("any")}
      className="inline-flex items-center gap-1 rounded-full border border-white/25 bg-white/10 px-2 py-0.5
      hover:border-white/40"
    >
      Datas: {eventDateFilter} ×
    </button>
  )}
  {timeScope !== "all" && (
    <button
      type="button"
      onClick={() => setTimeScope("all")}
      className="inline-flex items-center gap-1 rounded-full border border-white/25 bg-white/10 px-2 py-0.5
      hover:border-white/40"
    >
      Mostrar: {timeScope} ×
    </button>
  )}
  {searchTerm.trim() && (
    <button
      type="button"
      onClick={() => setSearchTerm("")}
      className="inline-flex items-center gap-1 rounded-full border border-white/25 bg-white/10 px-2 py-0.5
      hover:border-white/40"
    >

```

```

        Pesquisa: "{searchTerm}" x
      </button>
    )}
</div>
)}

<div className="space-y-3">
  <div className="flex flex-wrap items-center justify-between gap-2">
    <div className="flex items-center gap-2 text-sm text-white/80">
      <h3 className="text-lg font-semibold">Eventos</h3>
      <span className="text-[11px] rounded-full bg-white/10 px-2 py-0.5">{filteredEvents.length}</span>
    </div>
    <div className="flex items-center gap-2 text-[12px] text-white/70">
      <span className="rounded-full border border-white/15 px-2 py-0.5">Próximos: {upcomingCount}</span>
      <span className="rounded-full border border-white/15 px-2 py-0.5">A decorrer: {ongoingCount}</span>
      <span className="rounded-full border border-white/15 px-2 py-0.5">Concluidos: {finishedCount}</span>
    </div>
  </div>

```

{eventsListLoading && (

```

    <div className="grid gap-2 md:grid-cols-2">
      {[1, 2, 3].map((i) => (
        <div key={i} className="h-28 rounded-2xl border border-white/10 bg-white/5 animate-pulse" />
      )))
    </div>
  )}

{eventsError && (
  <div className="rounded-2xl border border-red-500/40 bg-red-500/10 px-4 py-3 text-sm text-red-100 flex items-center justify-between gap-3">
    <div>
      <p className="font-semibold">Não foi possível carregar os eventos.</p>
      <p className="text-[12px] text-red-100/80">Verifica a ligação e tenta novamente.</p>
    </div>
    <button
      type="button"
      onClick={() => mutateEvents()}
      className="rounded-full border border-red-200/50 px-3 py-1 text-[12px] font-semibold hover:bg-red-500/20">
      Tentar novamente
    </button>
  </div>
)}

```

{!eventsListLoading && events?.items?.length === 0 && (

```

  <div className="rounded-2xl border border-dashed border-white/15 bg-white/5 px-4 py-6 text-center text-sm text-white/70 space-y-2">
    <p className="text-base font-semibold text-white">Ainda não tens eventos criados.</p>
    <p>Começa por criar o teu primeiro evento e acompanha tudo a partir daqui.</p>
  </div>
)

```

{!eventsListLoading && events?.items && events.items.length > 0 && filteredEvents.length === 0 && (

```

  <div className="rounded-2xl border border-white/15 bg-white/5 px-4 py-6 text-center text-sm text-white/70 space-y-2">
    <p className="text-base font-semibold text-white">Nenhum evento corresponde a estes filtros.</p>
    <p className="text-white/65">Alarga as datas, limpa filtros ou procura por outro nome.</p>
    <div className="flex flex-wrap justify-center gap-2 text-[12px]">
      <button
        type="button"
        onClick={() => {
          setEventStatusFilter("all");
          setEventTypeFilter("all");
          setEventCategoryFilter("all");
          setEventDateFilter("any");
          setTimeScope("all");
          setSearchTerm("");
        }}
        className="rounded-full border border-white/20 px-3 py-1.5 text-white/80 hover:bg-white/10">
        Limpar filtros
      </button>
      <Link
        href="/organizador/eventos/novo"
        className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-3 py-1.5 font-semibold text-black shadow">
        Criar novo evento
      </Link>
    </div>
  </div>
)
}

```

```

        </Link>
      </div>
    </div>
  )}

{filteredEvents.length > 0 && (
  <div className="space-y-2">
    {viewMode === "table" ? (
      <div className="overflow-auto rounded-2xl border border-white/10 bg-white/5">
        <table className="min-w-full text-sm text-white/80">
          <thead className="text-left text-[11px] uppercase tracking-wide text-white/60">
            <tr>
              <th className="px-4 py-3">Evento</th>
              <th className="px-4 py-3">Data</th>
              <th className="px-4 py-3">Estado</th>
              <th className="px-4 py-3">Tipo</th>
              <th className="px-4 py-3">Bilhetes</th>
              <th className="px-4 py-3">Receita</th>
              <th className="px-4 py-3 text-right">Ações</th>
            </tr>
          </thead>
          <tbody className="divide-y divide-white/5">
            {filteredEvents.map((ev) => {
              const date = ev.startsAt ? new Date(ev.startsAt) : null;
              const endsAt = ev.endsAt ? new Date(ev.endsAt) : null;
              const now = new Date();
              const isOngoing = date && endsAt ? date.getTime() <= now.getTime() && now.getTime() <=
endsAt.getTime() : false;
              const isFuture = date ? date.getTime() > now.getTime() : false;
              const isFinished = endsAt ? endsAt.getTime() < now.getTime() : false;
              const dateLabel = date
                ? date.toLocaleString("pt-PT", {
                  day: "2-digit",
                  month: "short",
                  hour: "2-digit",
                  minute: "2-digit",
                })
                : "Data a confirmar";
              const ticketsSold = ev.ticketsSold ?? 0;
              const capacity = ev.capacity ?? null;
              const revenue = ((ev.revenueCents ?? 0) / 100).toFixed(2);
              const typeLabel =
                ev.templateType === "SPORT"
                  ? "Padel"
                  : ev.templateType === "COMIDA"
                  ? "Restaurantes & Jantares"
                  : ev.templateType === "VOLUNTEERING"
                  ? "Solidário / Voluntariado"
                  : ev.templateType === "PARTY"
                  ? "Festas & Noite"
                  : ev.templateType || "Outro";
              const statusBadge =
                ev.status === "CANCELLED"
                  ? { label: "Cancelado", classes: "text-red-200" }
                  : ev.status === "DRAFT"
                  ? { label: "Draft", classes: "text-white/70" }
                  : isOngoing
                  ? { label: "A decorrer", classes: "text-emerald-200" }
                  : isFuture
                  ? { label: "Publicado", classes: "text-sky-200" }
                  : isFinished
                  ? { label: "Concluído", classes: "text-purple-200" }
                  : { label: ev.status, classes: "text-white/70" };

              const goToTab = (tab: string) => {
                const params = new URLSearchParams(searchParams?.toString() || "");
                params.set("tab", tab);
                params.set("eventId", String(ev.id));
                router.replace(`${pathname}?${params.toString()}`, { scroll: false });
              };
            );
            return (
              <tr key={ev.id} className="hover:bg-white/5 transition">
                <td className="px-4 py-3 font-semibold text-white">
                  <button
                    type="button"
                    className="text-left hover:underline"
                    onClick={() => goToTab("sales")}
                  >

```

```

        >
          {ev.title}
        </button>
      </td>
      <td className="px-4 py-3 text-[12px]">{dateLabel}</td>
      <td className={`px-4 py-3 text-[12px] ${statusBadge.classes}`}>{statusBadge.label}</td>
      <td className="px-4 py-3 text-[12px]">
        {ticketsSold} / {capacity ?? "-"}
      </td>
      <td className="px-4 py-3 text-[12px]">{revenue} €</td>
      <td className="px-4 py-3 text-right text-[11px]">
        <div className="flex flex-wrap items-center justify-end gap-2">
          <button
            type="button"
            onClick={() => goToTab("sales")}
            className="rounded-full border border-white/20 px-2 py-1 hover:bg-white/10"
          >
            Vendas
          </button>
          <a href={`/organizador/eventos/${ev.id}/edit`}
            className="rounded-full border border-white/20 px-2 py-1 hover:bg-white/10"
          >
            Editar
          </a>
        </div>
      </td>
    </tr>
  );
);
}
</tbody>
</table>
</div>
) : (
  filteredEvents.map((ev) => {
    const date = ev.startsAt ? new Date(ev.startsAt) : null;
    const endsAt = ev.endsAt ? new Date(ev.endsAt) : null;
    const now = new Date();
    const isOngoing = date && endsAt ? date.getTime() <= now.getTime() && now.getTime() <= endsAt.getTime() :
false;
    const isFuture = date ? date.getTime() > now.getTime() : false;
    const isFinished = endsAt ? endsAt.getTime() < now.getTime() : false;
    const dateLabel = date
      ? date.toLocaleString("pt-PT", {
        day: "2-digit",
        month: "short",
        hour: "2-digit",
        minute: "2-digit",
      })
      : "Data a confirmar";
    const ticketsSold = ev.ticketsSold ?? 0;
    const capacity = ev.capacity ?? null;
    const revenue = ((ev.revenueCents ?? 0) / 100).toFixed(2);
    const typeLabel =
      ev.templateType === "SPORT"
        ? "Padel"
        : ev.templateType === "COMIDA"
        ? "Restaurantes & Jantares"
        : ev.templateType === "VOLUNTEERING"
        ? "Solidário / Voluntariado"
        : ev.templateType === "PARTY"
        ? "Festas & Noite"
        : ev.templateType || "Outro";
    const typeTone =
      ev.templateType === "SPORT"
        ? "border-sky-400/40 bg-sky-400/10 text-sky-100"
        : ev.templateType === "COMIDA"
        ? "border-amber-300/40 bg-amber-300/10 text-amber-100"
        : ev.templateType === "VOLUNTEERING"
        ? "border-emerald-300/40 bg-emerald-300/10 text-emerald-100"
        : ev.templateType === "PARTY"
        ? "border-fuchsia-300/40 bg-fuchsia-300/10 text-fuchsia-100"
        : "border-white/20 bg-white/5 text-white/80";

    const statusBadge =
      ev.status === "CANCELLED"
        ? { label: "Cancelado", classes: "border-red-400/50 bg-red-500/10 text-red-100" }

```

```

        : ev.status === "DRAFT"
        ? { label: "Draft", classes: "border-white/20 bg-white/5 text-white/70" }
        : isOngoing
        ? { label: "A decorrer", classes: "border-emerald-400/50 bg-emerald-500/10 text-emerald-100" }
        : isFuture
        ? { label: "Publicado", classes: "border-sky-400/50 bg-sky-500/10 text-sky-100" }
        : isFinished
        ? { label: "Concluido", classes: "border-purple-400/50 bg-purple-500/10 text-purple-100" }
        : { label: ev.status, classes: "border-white/20 bg-white/5 text-white/70" };

    const handlePrimaryOpen = () => {
        router.push(`/organizador?tab=sales&eventId=${ev.id}`);
    };

    const goToTab = (tab: string) => {
        const params = new URLSearchParams(searchParams?.toString() || "");
        params.set("tab", tab);
        params.set("eventId", String(ev.id));
        router.replace(`/${pathname}?${params.toString()}`, { scroll: false });
    };

    return (
        <div
            key={ev.id}
            role="button"
            tabIndex={0}
            onClick={(e) => {
                if ((e.target as HTMLElement).closest("a,button")) return;
                handlePrimaryOpen();
            }}
            onKeyDown={(e) => {
                if (e.key === "Enter" || e.key === " ") {
                    e.preventDefault();
                    handlePrimaryOpen();
                }
            }}
            className="group rounded-2xl border border-white/10 bg-white/5 p-4 hover:border-[#6BFFFF]/60
        hover:shadow-[0_12px_40px_rgba(0,0,0,0.5)] transition cursor-pointer"
        >
            <div className="flex flex-col gap-3 md:flex-row md:items-start md:justify-between">
                <div className="space-y-2">
                    <div className="flex flex-wrap items-center gap-2">
                        <p className="text-sm font-semibold text-white line-clamp-2">{ev.title}</p>
                        <span className={`rounded-full border px-2 py-0.5 text-[11px] ${statusBadge.classes}`}>
                            {statusBadge.label}</span>
                        <span className={`rounded-full border px-2 py-0.5 text-[11px] ${typeTone}`}>{typeLabel}</span>
                        {ev.categories?.[0] &&
                            <span className="rounded-full border border-white/20 bg-white/5 px-2 py-0.5 text-[11px] text-white/75">
                                {ev.categories[0]}
                            </span>
                        }
                    </div>
                    <div className="flex flex-wrap gap-2 text-[11px] text-white/70">
                        <span>📅 {dateLabel}</span>
                        <span>📍 {ev.locationName || ev.locationCity || "Local a anunciar"}</span>
                        <span>🎟️ {ticketsSold} / {capacity ?? "-"} bilhetes</span>
                        <span>€ {revenue} €</span>
                    </div>
                </div>
                <div className="flex flex-wrap items-center gap-2 text-[11px]">
                    <button
                        type="button"
                        onClick={(e) => {
                            e.stopPropagation();
                            handlePrimaryOpen();
                        }}
                        className="rounded-full border border-white/25 bg-white/10 px-3 py-1 font-semibold text-white
        hover:border-[#6BFFFF]/60"
                    >
                        Dashboard
                    </button>
                    <button
                        type="button"
                        onClick={(e) => {

```

```

        e.stopPropagation();
        goToTab("sales");
    }
    className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
>
    Ver vendas
</button>
<button
    type="button"
    onClick={(e) => {
        e.stopPropagation();
        goToTab("finance");
    }}
    className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
>
    Pagamentos
</button>
<Link
    href={`/organizador/eventos/${ev.id}/edit`}
    onClick={(e) => e.stopPropagation()}
    className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
>
    Editar
</Link>
<Link
    href={`/eventos/${ev.slug}`}
    onClick={(e) => e.stopPropagation()}
    className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
>
    Página pública
</Link>
<button
    type="button"
    disabled={eventActionLoading === ev.id}
    onClick={(e) => {
        e.stopPropagation();
        setEventDialog({ mode: ev.status === "DRAFT" ? "delete" : "archive", ev });
    }}
    className="rounded-full border border-red-200/30 px-2.5 py-1 text-red-100/90 hover:bg-red-500/10
disabled:opacity-60"
>
    {ev.status === "DRAFT" ? "Apagar rascunho" : "Arquivar"}
</button>
</div>
</div>
);
)
)
);
</div>
</div>
)
}
</div>
</section>
}

{activeTab === "sales" && (
<section className="space-y-4">
<div className="flex flex-col gap-3 md:flex-row md:items-center md:justify-between">
<div>
<p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Bilhetes & Vendas</p>
<h2 className="text-2xl font-semibold">Vendas por evento</h2>
<p className="text-sm text-white/65">Escolhe um evento e vê evolução + compradores.</p>
</div>
<div className="flex flex-wrap items-center gap-2">
<span className="text-[11px] text-white/60">Período</span>
<div className="inline-flex rounded-full border border-white/15 bg-black/40 p-[3px] text-[11px]">
{(["7d", "30d", "90d", "365d", "all"] as SalesRange[]).map((range) => (
<button
    key={range}
    type="button"
    onClick={() => setSalesRange(range)}
    className={` rounded-full px-3 py-1 transition ${(
        salesRange === range
            ? "bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold shadow-[0_0_12px_rgba(107,255,255,0.6)]"
            : "text-white/70 hover:bg-white/5"
    )}`}
>

```

```

        {range === "7d"
        ? "7 dias"
        : range === "30d"
        ? "30 dias"
        : range === "90d"
        ? "3 meses"
        : range === "365d"
        ? "1 ano"
        : "Sempre"}
    </button>
  )))
</div>
</div>
</div>

<div className="flex flex-wrap items-center gap-3">
  <div className="w-full max-w-md">
    <label className="text-xs uppercase tracking-[0.18em] text-white/60 block mb-1">Seleciona o evento</label>
    <div className="flex rounded-2xl border border-white/15 bg-black/40 px-3 py-2">
      <input
        type="text"
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        placeholder="Procurar por título ou cidade"
        className="flex-1 bg-transparent text-sm text-white outline-none placeholder:text-white/40"
      />
      <select
        value={salesEventId ?? ""}
        onChange={(e) => setSalesEventId(e.target.value ? Number(e.target.value) : null)}
        className="ml-2 w-48 rounded-xl border border-white/15 bg-black/60 px-2 py-2 text-sm text-white outline-none
focus:border-[#6BFFFF]"
      >
        <option value="">Escolhe</option>
        {eventsList.map((ev) =>
          <option key={ev.id} value={ev.id}>
            {ev.title}
          </option>
        )))
      </select>
    </div>
  </div>
  {!eventsList.length && <span className="text-[12px] text-white/60">Sem eventos para analisar.</span>}
</div>

<div className="grid gap-3 md:grid-cols-2 lg:grid-cols-4">
  {!salesEventId && (
    <div className="col-span-full rounded-2xl border border-dashed border-white/20 bg-black/30 p-4 text-white/70 text-
sm">
      Seleciona um evento para ver as métricas de vendas.
    </div>
  )}
  {salesLoading && (
    <>
      {[...Array(4)].map(_, idx) =>
        <div key={idx} className="rounded-2xl border border-white/10 bg-white/5 p-3 animate-pulse space-y-2">
          <div className="h-3 w-24 rounded bg-white/15" />
          <div className="h-7 w-20 rounded bg-white/20" />
          <div className="h-3 w-28 rounded bg-white/10" />
        </div>
      )})
    </>
  )}
  {!salesLoading && salesSeries && salesSeries.points?.length === 0 && (
    <div className="col-span-full rounded-2xl border border-dashed border-white/20 bg-black/30 p-4 text-white/70 text-
sm">
      Sem dados de vendas neste período. Escolhe outro evento ou intervalo.
    </div>
  )}
  {!salesLoading && salesSeries && salesSeries.points?.length !== 0 && (
    <>
      <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] text-white/60">Receita no período</p>
        <p className="text-2xl font-bold text-white mt-1">{(salesKpis.revenueCents / 100).toFixed(2)} €</p>
        <p className="text-[11px] text-white/50">{salesRangeLabelLong(salesRange)}</p>
      </div>
      <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] text-white/60">Bilhetes vendidos</p>
        <p className="text-2xl font-bold text-white mt-1">{(salesKpis.tickets)}</p>
      </div>
    </>
  )}

```

```

        <p className="text-[11px] text-white/50">No período selecionado</p>
    </div>
    <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] text-white/60">Eventos com vendas</p>
        <p className="text-2xl font-bold text-white mt-1">{salesKpis.eventsWithSales}</p>
        <p className="text-[11px] text-white/50">Eventos com pelo menos 1 venda</p>
    </div>
    <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] text-white/60">Ocupação média</p>
        <p className="text-2xl font-bold text-white mt-1">
            {salesKpis.avgOccupancy !== null ? `${salesKpis.avgOccupancy}%` : "-"}
        </p>
        <p className="text-[11px] text-white/50">Calculado nos eventos com capacidade</p>
    </div>
</>
)}
</div>

<div className="rounded-3xl border border-white/10 bg-black/40 p-4 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.65)]">
    <div className="flex items-center justify-between">
        <h3 className="text-lg font-semibold">Evolução</h3>
        {selectedSalesEvent && (
            <div className="flex items-center gap-2">
                <span className="text-[11px] text-white/60">{selectedSalesEvent.title}</span>
                <button
                    type="button"
                    disabled={!salesSeries?.points?.length}
                    onClick={handleExportSalesCsv}
                    className="rounded-full border border-white/20 px-3 py-1 text-[11px] text-white/80 hover:bg-white/10
disabled:opacity-50"
                >
                    Exportar vendas
                </button>
            </div>
        )}
    </div>
    <div className="rounded-2xl border border-white/10 bg-gradient-to-br from-white/5 to-white/0 shadow-inner overflow-hidden px-2 py-3 min-h-[260px]">
        {salesLoading ? (
            <div className="flex w-full items-center gap-3 px-4">
                <div className="h-28 flex-1 rounded-xl bg-white/10 animate-pulse" />
                <div className="hidden h-28 w-20 rounded-xl bg-white/10 animate-pulse md:block" />
            </div>
        ) : !salesEventId ? (
            <span className="text-white/40 text-xs">Escolhe um evento para ver a evolução.</span>
        ) : salesSeries?.points?.length ? (
            <SalesAreaChart
                data={salesChartPoints}
                periodLabel={salesRangeLabelLong(salesRange)}
            />
        ) : (
            <span className="text-white/40 text-xs">Sem dados de vendas para este evento.</span>
        )
    </div>
    {salesSeriesBreakdown && (
        <div className="flex flex-wrap gap-3 text-[11px] text-white/70">
            <span>Bruto: {formatEuros(salesSeriesBreakdown.gross)}</span>
            <span>Desconto: -{formatEuros(salesSeriesBreakdown.discount)}</span>
            <span>Taxas: -{formatEuros(salesSeriesBreakdown.fees)}</span>
            <span>Líquido: {formatEuros(salesSeriesBreakdown.net)}</span>
        </div>
    )}
</div>

<div className="rounded-3xl border border-white/10 bg-black/40 p-4 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.65)]">
    <div className="flex items-center justify-between">
        <div>
            <h3 className="text-lg font-semibold">Eventos com mais vendas</h3>
            <p className="text-[11px] text-white/60">Top por receita total. Usa como atalho para ver o detalhe.</p>
        </div>
    </div>
    {topEvents.length === 0 && <p className="text-sm text-white/60">Ainda sem eventos com vendas para ordenar.</p>}
    {topEvents.length > 0 && (
        <div className="overflow-auto">
            <table className="min-w-full text-sm">
                <thead className="text-left text-[11px] text-white/60">
                    <tr>

```

```

<th className="py-2 pr-3">Evento</th>
<th className="py-2 pr-3">Bilhetes</th>
<th className="py-2 pr-3">Receita</th>
<th className="py-2 pr-3 text-right">Ações</th>
</tr>
</thead>
<tbody className="divide-y divide-white/5">
{topEvents.map((ev) => {
  const statusBadge =
    ev.status === "CANCELLED"
      ? { label: "Cancelado", classes: "border-red-400/50 bg-red-500/10 text-red-100" }
      : ev.status === "DRAFT"
        ? { label: "Draft", classes: "border-white/20 bg-white/5 text-white/70" }
        : { label: "Publicado", classes: "border-sky-400/50 bg-sky-500/10 text-sky-100" };
  return (
    <tr key={ev.id}>
      <td className="py-2 pr-3 text-white">{ev.title}</td>
      <td className="py-2 pr-3 text-white/80">{ev.ticketsSold ?? 0}</td>
      <td className="py-2 pr-3 text-white">{(ev.revenueCents ?? 0) / 100}.toFixed(2) €</td>
      <td className="py-2 pr-3 text-[11px]">
        <span className={`rounded-full border px-2 py-0.5 ${statusBadge.classes}`}>{statusBadge.label}</span>
      </td>
      <td className="py-2 pr-3 text-right text-[11px]">
        <div className="flex items-center justify-end gap-2">
          <button
            type="button"
            onClick={() => setSalesEventId(ev.id)}
            className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10">
            Ver vendas
          </button>
          <Link
            href={`/organizador?tab=sales&eventId=${ev.id}`}
            className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10">
            Dashboard de vendas
          </Link>
        </div>
      </td>
    </tr>
  );
})
);
</tbody>
</table>
</div>
)
</div>

<div className="rounded-3xl border border-white/10 bg-black/40 p-4 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.65)]">
<div className="flex items-center justify-between">
<div>
  <h3 className="text-lg font-semibold">Compradores</h3>
  <p className="text-[11px] text-white/60">Lista rápida por bilhete. Exporta para CSV para detalhe.</p>
</div>
<button
  type="button"
  disabled={!buyers || buyers.ok === false || buyersItems.length === 0}
  onClick={() => {
    if (!buyers || buyers.ok === false) return;
    const rows = buyersItems;
    const header = ["ID", "Nome", "Email", "Cidade", "Tipo", "Preço (€)", "Estado", "Comprado em"];
    const body = rows
      .map((r) =>
        [
          r.id,
          r.buyerName,
          r.buyerEmail,
          r.buyerCity ?? "",
          r.ticketType,
          (r.totalPaidCents / 100).toFixed(2),
          r.status,
          new Date(r.purchasedAt).toLocaleString("pt-PT"),
        ].join(";"))
      )
      .join("\n");
    const blob = new Blob([header.join(""), body.join("\n")], { type: "text/csv" });
  }}>

```

```

        const url = URL.createObjectURL(blob);
        const a = document.createElement("a");
        a.href = url;
        a.download = "compradores.csv";
        a.click();
        URL.revokeObjectURL(url);
    }
    className="text-[11px] rounded-full border border-white/20 px-3 py-1 text-white/80 hover:bg-white/10
disabled:opacity-50"
>
    Exportar CSV
</button>
</div>

{buyersLoading && (
<div className="space-y-2">
    {...Array(4)].map(_, idx) => (
<div
    key={idx}
    className="flex items-center justify-between rounded-xl border border-white/10 bg-black/25 p-3 animate-pulse"
>
    <div className="space-y-2">
        <div className="h-3 w-32 rounded bg-white/10" />
        <div className="h-3 w-20 rounded bg-white/5" />
    </div>
    <div className="h-3 w-16 rounded bg-white/10" />
    </div>
    ))}
</div>
)}
{!buyersLoading && !salesEventId && (
<p className="text-sm text-white/60">Escolhe um evento para ver compradores.</p>
)
}
{!buyersLoading && salesEventId && buyers && buyers.ok === false && (
<p className="text-sm text-red-400">Não foi possível carregar os compradores.</p>
)
}
{!buyersLoading && salesEventId && buyers && buyers.ok !== false && buyersItems.length === 0 && (
<p className="text-sm text-white/60">Sem compras registadas para este evento.</p>
)
}
{!buyersLoading && salesEventId && buyers && buyers.ok !== false && buyersItems.length > 0 && (
<div className="overflow-auto">
    <table className="min-w-full text-sm">
        <thead className="text-left text-[11px] text-white/60">
            <tr>
                <th className="py-2 pr-3">Comprador</th>
                <th className="py-2 pr-3">Email</th>
                <th className="py-2 pr-3">Bilhete</th>
                <th className="py-2 pr-3">Estado</th>
                <th className="py-2 pr-3 text-right">Pago</th>
                <th className="py-2 pr-3">Data</th>
            </tr>
        </thead>
        <tbody className="divide-y divide-white/5">
            {buyersItems.map((row) => (
<tr key={row.id}>
                <td className="py-2 pr-3 text-white">{row.buyerName}</td>
                <td className="py-2 pr-3 text-white/70">{row.buyerEmail}</td>
                <td className="py-2 pr-3 text-white/80">{row.ticketType}</td>
                <td className="py-2 pr-3 text-[11px]">
                    <span className="rounded-full border border-white/15 px-2 py-0.5 text-white/70">{row.status}</span>
                </td>
                <td className="py-2 pr-3 text-right text-white">
                    {(row.totalPaidCents / 100).toFixed(2)} €
                </td>
                <td className="py-2 pr-3 text-white/70">
                    {new Date(row.purchasedAt).toLocaleString("pt-PT")}
                </td>
            </tr>
        )))
        </tbody>
    </table>
</div>
)
}
</div>
</section>
)
}

```

```

{activeTab === "finance" && (
  <section className="space-y-4">
    <div className="flex flex-col gap-2">
      <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Finanças</p>
      <h2 className="text-2xl font-semibold">Receita, bilhetes e Stripe</h2>
      <p className="text-sm text-white/65">Visão simples do dinheiro e estado da conta Stripe.</p>
    </div>

  {paymentsMode === "CONNECT" && paymentsStatus !== "READY" && (
    <div
      className={`${`rounded-2xl px-4 py-3 text-sm ${stripeIncomplete ? "border border-amber-400/40 bg-amber-400/10 text-amber-50" : "border border-amber-400/30 bg-amber-400/10 text-amber-50"}`}
    >
      <div className="flex flex-wrap items-center justify-between gap-2">
        <div className="space-y-1">
          <p className="font-semibold">
            {stripeIncomplete ? "Onboarding incompleto no Stripe." : "Liga o Stripe para começar a receber."}
          </p>
          <p className="text-[12px] text-amber-100/80">
            {paymentsStatus === "NO_STRIPE"
              ? "Sem ligação Stripe não há payouts. O resto da gestão continua disponível."
              : stripeRequirements.length > 0
              ? `Faltam ${stripeRequirements.length} passos no Stripe Connect. Abre o painel para concluir.`
              : "Conclui o onboarding no Stripe para ativar payouts."}
          </p>
        </div>
        <button
          type="button"
          onClick={handleStripeConnect}
          disabled={stripeCtaLoading}
          className="rounded-full bg-white px-3 py-1.5 text-[12px] font-semibold text-black shadow hover:scale-[1.01] disabled:opacity-60"
        >
          {stripeCtaLoading ? "A ligar..." : stripeIncomplete ? "Continuar configuração" : "Ligar conta Stripe"}
        </button>
      </div>
    </div>
  )}

  {paymentsMode === "PLATFORM" && (
    <div className="rounded-2xl border border-emerald-400/40 bg-emerald-500/10 px-4 py-3 text-sm text-emerald-50">
      <div className="flex flex-wrap items-center justify-between gap-2">
        <div className="space-y-1">
          <p className="font-semibold">Conta interna ORYA</p>
          <p className="text-[12px] text-emerald-50/80">
            Pagamentos processados na conta principal da ORYA. Não precisas de ligar Stripe Connect.
          </p>
        </div>
      </div>
    </div>
  )}
  {stripeSuccessMessage && (
    <div className="rounded-2xl border border-emerald-400/40 bg-emerald-500/10 px-4 py-3 text-sm text-emerald-50">
      {stripeSuccessMessage}
    </div>
  )}

  <div className="grid gap-3 md:grid-cols-2 lg:grid-cols-4">
    {[

      {
        label: "Receita líquida total",
        value:
          financeData?.totals.netCents !== undefined
            ? `${(financeData.totals.netCents / 100).toFixed(2)} €`
            : financeSummary
            ? `${(financeSummary.estimatedPayoutCents / 100).toFixed(2)} €`
            : "-",
        hint: "Valor que fica para ti (bruto - taxas.)",
      },
      {
        label: "Receita últimos 30d",
        value:
          financeData?.rolling.last30.netCents !== undefined
            ? `${(financeData.rolling.last30.netCents / 100).toFixed(2)} €`
            : "-",
        hint: "Líquido nos últimos 30 dias.",
      }
    ]}
  </div>
)

```

```

},
{
  label: "Taxas",
  value:
    financeData?.totals.feesCents !== undefined
    ? `${(financeData.totals.feesCents / 100).toFixed(2)} €`
    : financeSummary
      ? `${(financeSummary.platformFeesCents / 100).toFixed(2)} €`
      : "-",
  hint: "Custos de processamento + eventuais fees.",
},
{
  label: "Eventos com vendas",
  value: financeData?.totals.eventsWithSales ?? financeSummary?.eventsWithSales ?? "-",
  hint: "Eventos pagos com pelo menos 1 bilhete.",
},
].map((card) => (
  <div key={card.label} className="rounded-2xl border border-white/10 bg-white/5 p-3">
    <p className="text-[11px] text-white/60">{card.label}</p>
    <p className="text-2xl font-bold text-white mt-1">{card.value}</p>
    <p className="text-[11px] text-white/50">{card.hint}</p>
  </div>
))
</div>

<div className="grid gap-4 md:grid-cols-2">
  <div className="rounded-3xl border border-white/10 bg-white/5 backdrop-blur-xl p-4 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.65)]">
    <div className="flex flex-wrap items-center justify-between gap-2">
      <div className="flex items-center gap-2">
        <h3 className="text-lg font-semibold">Stripe</h3>
        <span
          className={`rounded-full px-2 py-0.5 text-[11px] ${stripeState.tone === "success"
            ? "border-emerald-400/50 bg-emerald-500/10 text-emerald-100"
            : stripeState.tone === "warning"
              ? "border-amber-400/50 bg-amber-500/10 text-amber-100"
              : stripeState.tone === "error"
                ? "border-red-400/50 bg-red-500/10 text-red-100"
                : "border-white/20 bg-white/5 text-white/70"
            }`}
        >
          {stripeState.badge}
        </span>
      </div>
      <div className="flex flex-wrap items-center gap-2">
        {paymentsStatus === "READY" ? (
          <a
            href="https://dashboard.stripe.com/"
            target="_blank"
            rel="noreferrer"
            className="text-[11px] rounded-full border border-white/20 px-3 py-1 text-white/80 hover:bg-white/10"
          >
            {stripeState.cta}
          </a>
        ) : (
          <button
            type="button"
            onClick={handleStripeConnect}
            disabled={stripeCtaLoading}
            className="text-[11px] rounded-full border border-white/20 px-3 py-1 text-white/80 hover:bg-white/10
disabled:opacity-60"
          >
            {stripeState.cta}
          </button>
        )}
      </div>
    </div>
    <div className="rounded-2xl border border-white/10 bg-black/40 p-3 text-sm space-y-1">
      <p className="text-white/60">Conta: {organizer.stripeAccountId ? `...${organizer.stripeAccountId.slice(-6)}` : "Por ligar"}</p>
      <p className="text-white/60">Cobranças: {organizer.stripeChargesEnabled ? "Ativo" : "Inativo"}</p>
      <p className="text-white/60">Payouts: {organizer.stripePayoutsEnabled ? "Ativo" : "Inativo"}</p>
    </div>
    <div className="text-[11px] text-white/70 space-y-2">
      <p>{stripeState.desc}</p>
      {stripeRequirements.length > 0 && (
        <p className="text-white/65">

```

```

        {stripeRequirements.length} itens pendentes no Stripe. Conclui-os no painel Connect para ativar payouts.
    </p>
  )}
</div>
{stripeCtaError && <div className="text-xs text-red-300">{stripeCtaError}</div>
</div>

<div className="rounded-3xl border border-white/10 bg-black/30 backdrop-blur-xl p-4 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.65)]">
  <div className="flex items-center justify-between">
    <h3 className="text-lg font-semibold">Payouts</h3>
    <span className="text-[11px] text-white/65">Informativo</span>
  </div>
  <div className="grid gap-2 sm:grid-cols-2 text-sm">
    <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
      <p className="text-white/60 text-xs">Próximo payout (estimado)</p>
      <p className="text-xl font-semibold text-white">
        {financeData ? (financeData.upcomingPayoutCents / 100).toFixed(2) : financeSummary ?
(financeSummary.estimatedPayoutCents / 100).toFixed(2) : "--"} €
      </p>
      <p className="text-[11px] text-white/55">Baseado em vendas recentes. Funcionalidade de payouts automáticos em breve.</p>
    </div>
    <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
      <p className="text-white/60 text-xs">Receita bruta (total)</p>
      <p className="text-xl font-semibold text-white">
        {financeData ? (financeData.totals.grossCents / 100).toFixed(2) : financeSummary ?
(financeSummary.revenueCents / 100).toFixed(2) : "--"} €
      </p>
      <p className="text-[11px] text-white/55">Inclui todos os eventos.</p>
    </div>
    <div className="grid gap-2 sm:grid-cols-2 text-sm">
      <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
        <p className="text-white/60 text-xs">Taxas acumuladas</p>
        <p className="text-xl font-semibold text-white">
          {financeData ? (financeData.totals.feesCents / 100).toFixed(2) : financeSummary ?
(financeSummary.platformFeesCents / 100).toFixed(2) : "--"} €
        </p>
        <p className="text-[11px] text-white/55">Inclui processamento Stripe e fees aplicadas.</p>
      </div>
      <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
        <p className="text-white/60 text-xs">Eventos com vendas</p>
        <p className="text-xl font-semibold text-white">
          {financeData ? financeData.totals.eventsWithSales : financeSummary.eventsWithSales : "--"} %
        </p>
      </div>
    </div>
    <p className="text-[11px] text-white/60">
      Payouts automáticos e gestão avançada de taxas chegam em breve. Estes valores são informativos.
    </p>
  </div>
</div>
</div>
</div>

<div className="rounded-3xl border border-white/10 bg-black/35 backdrop-blur-xl p-4 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.65)]">
  <div className="flex items-center justify-between">
    <div>
      <h3 className="text-lg font-semibold text-white">Por evento</h3>
      <p className="text-[12px] text-white/65">Bruto, taxas e líquido por evento.</p>
    </div>
    <div className="flex items-center gap-2">
      <button
        type="button"
        onClick={exportFinanceCsv}
        disabled={!financeData || financeData.events.length === 0}
        className="rounded-full border border-white/20 px-3 py-1 text-[12px] text-white/80 hover:bg-white/10
disabled:opacity-50">
        >
        Exportar CSV
      </button>
    </div>
  </div>
  {!financeData && <p className="text-sm text-white/60">A carregar finanças...</p>}
  {financeData && financeData.events.length === 0 && (
    <div className="rounded-2xl border border-dashed border-white/15 bg-white/5 px-4 py-4 text-sm text-white/70">
      Sem vendas ainda. Assim que venderes bilhetes, verás aqui os totais por evento.
    </div>
  )
}

```

```

        </div>
    )}
{stripeSuccessMessage && (
<div className="rounded-2xl border border-emerald-400/40 bg-emerald-500/10 px-4 py-3 text-sm text-emerald-50">
    {stripeSuccessMessage}
</div>
)}

{financeData && financeData.events.length > 0 && (
<div className="overflow-auto">
    <table className="min-w-full text-sm text-white/80">
        <thead className="text-left text-[11px] uppercase tracking-wide text-white/60">
            <tr>
                <th className="px-4 py-3">Evento</th>
                <th className="px-4 py-3">Bilhetes</th>
                <th className="px-4 py-3">Bruto</th>
                <th className="px-4 py-3">Taxas</th>
                <th className="px-4 py-3">Líquido</th>
                <th className="px-4 py-3">Estado</th>
            </tr>
        </thead>
        <tbody className="divide-y divide-white/5">
            {financeData.events.map((ev) =>
                <tr key={ev.id} className="hover:bg-white/5 transition">
                    <td className="px-4 py-3">
                        <div className="flex flex-col">
                            <span className="font-semibold text-white">{ev.title}</span>
                            <span className="text-[11px] text-white/60">
                                {ev.startsAt ? new Date(ev.startsAt).toLocaleDateString("pt-PT") : "Data a definir"}
                            </span>
                        </div>
                    </td>
                    <td className="px-4 py-3 text-[12px]">{ev.ticketsSold}</td>
                    <td className="px-4 py-3 text-[12px]">{(ev.grossCents / 100).toFixed(2)} €</td>
                    <td className="px-4 py-3 text-[12px]">{(ev.feesCents / 100).toFixed(2)} €</td>
                    <td className="px-4 py-3 text-[12px]">{(ev.netCents / 100).toFixed(2)} €</td>
                    <td className="px-4 py-3 text-[11px]">
                        <span className="rounded-full border border-white/20 px-2 py-0.5 text-white/70">{ev.status ?? "-"}</span>
                    </td>
                </tr>
            )))
        </tbody>
    </table>
)
)}
</div>
</section>
)}

{activeTab === "invoices" && (
<section className="space-y-4">
    <InvoicesClient />
</section>
)}

{activeTab === "marketing" && (
<section className="space-y-4">
    <div className="flex flex-col gap-2 md:flex-row md:items-center md:justify-between">
        <div>
            <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Marketing & Crescimento</p>
            <h2 className="text-2xl font-semibold">Marketing · {marketingSection === "overview" ? "Visão geral" : "Painel"}</h2>
        </div>
        <p className="text-sm text-white/65">Receita atribuída a códigos e ações rápidas.</p>
    </div>
    <div className="flex flex-wrap gap-2 text-[11px]">
        <Link href="/organizador?tab=marketing&section=promos" className="rounded-full border border-white/20 px-4 py-2 text-sm text-white/80 hover:bg-white/10">
            Ver todos os códigos
        </Link>
    </div>
</div>
)}

<div className="flex flex-wrap gap-2 rounded-2xl border border-white/10 bg-black/30 px-2 py-2 text-sm">
    {[{"key": "overview", "label": "Visão geral"}, {"key": "payouts", "label": "Payouts"}, {"key": "payments", "label": "Pagamentos"}, {"key": "customers", "label": "Clientes"}, {"key": "products", "label": "Produtos"}, {"key": "orders", "label": "Pedidos"}, {"key": "refunds", "label": "Reembolsos"}, {"key": "reports", "label": "Relatórios"}]}

```

```

        { key: "promos", label: "Códigos promocionais" },
        { key: "promoters", label: "Promotores & Parcerias" },
        { key: "content", label: "Conteúdo & Kits" },
    ].map((opt) => (
        <button
            key={opt.key}
            type="button"
            onClick={() => setMarketingSection(opt.key as typeof marketingSection)}
            className={`rounded-xl px-3 py-2 transition ${marketingSection === opt.key ? "bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] text-black font-semibold shadow-[0_0_16px_rgba(107,255,255,0.35)]" : "text-white/75 hover:bg-white/5"}`}
            marketingSection === opt.key
            style={{ border: `1px solid ${opt.key === marketingSection ? "#1646F5" : "#6BFFFF"}` }}
        >
            {opt.label}
        </button>
    )))
</div>

{marketingSection === "overview" && (
    <div className="space-y-4">
        <div className="grid gap-3 md:grid-cols-2 lg:grid-cols-4">
            {marketingOverview
                ? (
                    <>
                        <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
                            <p className="text-[11px] text-white/60">Receita atribuída a marketing</p>
                            <p className="text-2xl font-bold text-white mt-1">${marketingKpis.marketingRevenueCents ? `${(marketingKpis.marketingRevenueCents / 100).toFixed(2)} €` : "-"}</p>
                            <p className="text-[11px] text-white/50">Receita estimada através de códigos.</p>
                        </div>
                        <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
                            <p className="text-[11px] text-white/60">Bilhetes via marketing</p>
                            <p className="text-2xl font-bold text-white mt-1">${marketingKpis.ticketsWithPromo}</p>
                            <p className="text-[11px] text-white/50">Contagem de utilizações de códigos.</p>
                        </div>
                        <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
                            <p className="text-[11px] text-white/60">Top código</p>
                            <p className="text-2xl font-bold text-white mt-1">${marketingKpis.topPromo ? marketingKpis.topPromo.code : "-"}</p>
                            <p className="text-[11px] text-white/50">${marketingKpis.topPromo ? `${marketingKpis.topPromo.redemptionsCount ?? 0} utilizações` : "Sem dados ainda."}</p>
                        </div>
                        <div className="rounded-2xl border border-white/10 bg-white/5 p-3">
                            <p className="text-[11px] text-white/60">Promo codes ativos</p>
                            <p className="text-2xl font-bold text-white mt-1">${marketingKpis.activePromos}</p>
                            <p className="text-[11px] text-white/50">Disponíveis para vender agora.</p>
                        </div>
                    </>
                )
                : [...Array(4)].map((_, idx) =>
                    <div key={idx} className="rounded-2xl border border-white/10 bg-white/5 p-3 space-y-2 animate-pulse">
                        <div className="h-3 w-24 rounded bg-white/15" />
                        <div className="h-6 w-20 rounded bg-white/20" />
                        <div className="h-3 w-32 rounded bg-white/10" />
                    </div>
                )))
        </div>

        <div className="rounded-3xl border border-white/10 bg-black/35 p-4 space-y-3">
            <div className="flex items-center justify-between">
                <div>
                    <h3 className="text-lg font-semibold text-white">Fill the Room</h3>
                    <p className="text-[12px] text-white/65">Próximos eventos com ocupação, urgência e sugestões.</p>
                </div>
                <Link href="/organizador?tab=marketing&section=promos" className="rounded-full border border-white/20 px-3 py-1 text-white/80 hover:bg-white/10">
                    Ver todas as ações
                </Link>
            </div>
        </div>
    )
)

```

```

{fillTheRoomEvents.length === 0 && (
  <div className="rounded-2xl border border-dashed border-white/15 bg-white/5 px-4 py-4 text-sm text-white/70">
    Sem eventos futuros para otimizar. Cria um evento ou define datas para ver sugestões.
  </div>
)}

{fillTheRoomEvents.length > 0 && (
  <div className="space-y-2">
    {fillTheRoomEvents.map((ev) => (
      <div
        key={ev.id}
        className="flex flex-col gap-2 rounded-2xl border border-white/10 bg-white/5 p-3 md:flex-row md:items-center md:justify-between"
      >
        <div className="space-y-1">
          <div className="flex flex-wrap items-center gap-2">
            <p className="text-sm font-semibold">{ev.title}</p>
            <span className={` rounded-full border px-2 py-0.5 text-[11px] ${ev.tag.tone}`}>{ev.tag.label}</span>
            <span className="rounded-full border border-white/20 bg-white/5 px-2 py-0.5 text-[11px] text-white/75">
              {ev.templateType || "Evento"}
            </span>
            {typeof ev.diffDays === "number" &&
              <span className="rounded-full border border-white/15 bg-white/5 px-2 py-0.5 text-[11px] text-white/70">
                Faltam {ev.diffDays} dia{ev.diffDays === 1 ? "" : "s"}
              </span>
            }
          </div>
          <div className="flex flex-wrap gap-2 text-[11px] text-white/70">
            <span>{ev.startsAt ? new Date(ev.startsAt).toLocaleString("pt-PT", { day: "2-digit", month: "short", hour: "2-digit", minute: "2-digit" }) : "Data a definir"}</span>
            <span></span>
            <span>{ev.locationCity || ev.locationName || "Local a anunciar"}</span>
            <span></span>
            <span>
              Lotação: {ev.ticketsSold ?? 0} / {ev.capacity ?? "-"}{" "}
              {ev.occupancy !== null ? `(${Math.round((ev.occupancy ?? 0) * 100)})%` : ""}
            </span>
          </div>
        </div>
        <div className="flex flex-col gap-2 text-[12px] md:text-right">
          <div className="flex items-center gap-2 text-[11px] text-white/70">
            <div className="h-2 w-28 rounded-full bg-white/10">
              <div
                className="h-2 rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5]"
                style={{ width: `${Math.min(100, Math.round((ev.occupancy ?? 0) * 100))}%` }}
              >
              </div>
            <span>{ev.occupancy !== null ? `(${Math.round((ev.occupancy ?? 0) * 100)})%` : "-"}</span>
          </div>
          <div className="flex flex-wrap justify-end gap-2 text-[11px]">
            <Link
              href="/organizador?tab=marketing&section=promos"
              className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
            >
              {ev.tag.suggestion}
            </Link>
            <Link
              href={`/organizador/eventos/${ev.id}/edit`}
              className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
            >
              Ajustar evento
            </Link>
            <Link
              href={`/eventos/${ev.slug}`}
              className="rounded-full border border-white/20 px-2.5 py-1 text-white/80 hover:bg-white/10"
            >
              Partilhar
            </Link>
          </div>
        </div>
      </div>
    )))
  </div>
)
</div>

```

```

<div className="rounded-3xl border border-white/10 bg-black/35 p-4">
  <div className="flex items-center justify-between gap-2">
    <div>
      <h4 className="text-lg font-semibold text-white">Funil de marketing (v1)</h4>
      <p className="text-[12px] text-white/65">Bilhetes totais vs. com promo vs. convidados.</p>
    </div>
    <span className="rounded-full border border-white/15 bg-white/5 px-2 py-1 text-[11px] text-white/70">Baseado
em códigos</span>
  </div>
  <div className="mt-3 grid gap-3 md:grid-cols-3">
    {[<br/>
      { label: "Bilhetes totais", value: marketingKpis.totalTickets ?? "-" },
      { label: "Bilhetes com promo", value: marketingKpis.ticketsWithPromo ?? 0 },
      { label: "Guest / convidados", value: marketingKpis.guestTickets ?? 0 },
    ].map((item) => (
      <div key={item.label} className="rounded-2xl border border-white/10 bg-white/5 p-3">
        <p className="text-[11px] text-white/60">{item.label}</p>
        <p className="text-xl font-bold text-white mt-1">{item.value}</p>
      </div>
    )));
  </div>
</div>
</div>
)
}

{marketingSection === "promos" && (
  <PromoCodesPage />
)
}

{marketingSection === "promoters" && (
  <div className="space-y-3">
    <div className="flex items-center justify-between">
      <div>
        <h3 className="text-xl font-semibold text-white">Promotores & Parcerias</h3>
        <p className="text-[12px] text-white/65">Quem te ajuda a vender (pessoas, grupos, parceiros).</p>
      </div>
      <button
        type="button"
        className="rounded-full border border-white/20 px-4 py-2 text-sm font-semibold text-white/70 cursor-not-
allowed"
        disabled
      >
        Em breve
      </button>
    </div>
    <div className="rounded-3xl border border-white/10 bg-black/35 p-4 text-sm text-white/70 space-y-3">
      <p className="text-white/80 font-semibold">Em breve</p>
      <p className="text-[12px] text-white/65">Dashboard de vendas por promotor e links com comissão estimada.</p>
    </div>
  </div>
)
}

{marketingSection === "content" && (
  <div className="space-y-3">
    <div className="flex items-center justify-between">
      <div>
        <h3 className="text-xl font-semibold text-white">Conteúdo & Kits</h3>
        <p className="text-[12px] text-white/65">Copiar e partilhar: textos rápidos por evento.</p>
      </div>
      <span className="rounded-full border border-white/15 bg-white/5 px-3 py-1 text-[11px] text-white/70">Em
breve</span>
    </div>
    <div className="rounded-3xl border border-white/10 bg-black/35 p-4 text-sm text-white/70">
      Em breve: kits rápidos para Instagram, WhatsApp e email por evento, com botões de copiar.
    </div>
  </div>
)
}

{activeTab === "padel" && (
  <section className="space-y-4">
    <div className="flex flex-wrap items-center justify-between gap-3">
      <div>
        <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Categorias</p>
        <h2 className="text-2xl font-semibold">Padel</h2>
        <p className="text-sm text-white/65">Clubes, courts, staff e jogadores num só sitio.</p>
      </div>
    </div>
)
}

```

```

        </div>
        <div className="flex flex-wrap gap-2">
          <Link
            href="/organizador/eventos/novo?templateType=PADEL"
            className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFF] to-[#1646F5] px-4 py-2 text-sm font-semibold text-black shadow-[0_0_16px_rgba(107,255,255,0.35)] hover:scale-[1.02] transition"
          >
            Criar torneio
          </Link>
          <Link
            href="/organizador?tab=events&type=PADEL"
            className="rounded-full border border-white/20 px-4 py-2 text-sm text-white/80 hover:bg-white/10"
          >
            Ver torneios
          </Link>
        </div>
      </div>

{!organizer?.id && <p className="text-sm text-white/70">Sem organização ativa.</p>}
{organizer?.id && (
  <PadelHubClient
    organizerId={organizer.id}
    organizationKind={organizer as { organizationKind?: string | null }}.organizationKind ?? "PESSOA_SINGULAR"
    initialClubs={padelClubs?.items ?? []}
    initialPlayers={padelPlayers?.items ?? []}
  />
)
</section>
)}

{activeTab === "restaurants" && (
  <section className="space-y-3">
    <div className="rounded-2xl border border-white/10 bg-white/5 p-4">
      <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Categorias</p>
      <h2 className="text-xl font-semibold text-white">Restaurantes & Jantares</h2>
      <p className="text-sm text-white/65">Em breve – reservas e menus fixos.</p>
    </div>
  </section>
)
}

{activeTab === "volunteer" && (
  <section className="space-y-3">
    <div className="rounded-2xl border border-white/10 bg-white/5 p-4">
      <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Categorias</p>
      <h2 className="text-xl font-semibold text-white">Solidário / Voluntariado</h2>
      <p className="text-sm text-white/65">Em breve – inscrições de voluntários e donativos.</p>
    </div>
  </section>
)
}

{activeTab === "night" && (
  <section className="space-y-3">
    <div className="rounded-2xl border border-white/10 bg-white/5 p-4">
      <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Categorias</p>
      <h2 className="text-xl font-semibold text-white">Festas & Noite</h2>
      <p className="text-sm text-white/65">Em breve – guest lists, packs e consumo mínimo.</p>
    </div>
  </section>
)
}

{activeTab === "staff" && (
  <section className="space-y-3">
    <OrganizerStaffPage />
  </section>
)
}

{activeTab === "settings" && (
  <section className="space-y-3">
    <OrganizerSettingsPage />
  </section>
)
}

{eventDialog && (
  <ConfirmDestructiveActionDialog
    open
    title={eventDialog.mode === "delete" ? "Apagar rascunho?" : "Arquivar evento?"}
    description={
      eventDialog.mode === "delete"
    }
)
}

```

```

        ? "Esta ação remove o rascunho e bilhetes associados."
        : "O evento deixa de estar visível para o público. Vendas e relatórios mantêm-se."
    }
    consequences={
        eventDialog.mode === "delete"
        ? ["Podes criar outro evento quando quiseres."]
        : ["Sai de /explorar e das listas do dashboard.", "Mantém histórico para relatórios/finanças."]
    }
    confirmLabel={eventDialog.mode === "delete" ? "Apagar rascunho" : "Arquivar evento"}
    dangerLevel="high"
    onConfirm={() => archiveEvent(eventDialog.ev, eventDialog.mode)}
    onClose={() => setEventDialog(null)}
    />
)
);
</div>
);
}
};

type TimeSeriesPoint = {
    date: string;
    tickets: number;
    revenueCents: number; // líquido (net)
    netCents?: number; // alias
    grossCents?: number;
    discountCents?: number;
    platformFeeCents?: number;
};


```

app/organizador/DashboardTabs.tsx

```

"use client";

import Link from "next/link";
import { usePathname, useRouter, useSearchParams } from "next/navigation";

export type TabKey =
    | "overview"
    | "events"
    | "sales"
    | "finance"
    | "invoices"
    | "marketing"
    | "padel"
    | "staff"
    | "settings";

type Tab = { id: TabKey; label: string };

const TABS: Tab[] = [
    { id: "overview", label: "Resumo" },
    { id: "events", label: "Eventos" },
    { id: "marketing", label: "Marketing" },
    { id: "staff", label: "Equipa" },
    { id: "finance", label: "Finanças" },
    { id: "invoices", label: "Faturação" },
    { id: "padel", label: "Padel" },
    { id: "settings", label: "Definições" },
];

function tabHref(id: TabKey) {
    return `/organizador?tab=${id}`;
}

export function DashboardTabs() {
    const router = useRouter();
    const pathname = usePathname();
    const searchParams = useSearchParams();
    const tabParam = searchParams?.get("tab");
    const activeTab = TABS.some((t) => t.id === tabParam) ? (tabParam as TabKey) : "overview";

    const handleClick = (tab: TabKey) => {
        const params = new URLSearchParams(searchParams ?? undefined);
        params.set("tab", tab);
        router.replace(`${pathname}?${params.toString()}`, { scroll: false });
    };
}


```

```

    return (
      <div className="flex w-full flex-wrap items-center gap-2 rounded-2xl border border-white/10 bg-white/5 p-2 text-sm text-white/80 shadow-[0_16px_60px_rgba(0,0,0,0.35)]">
        {TABS.map((tab) => {
          const active = activeTab === tab.id || (!tabParam && tab.id === "overview");
          return (
            <button
              key={tab.id}
              type="button"
              onClick={() => handleClick(tab.id)}
              className={`rounded-2xl px-3.5 py-2 transition ${active ? "bg-white/15 text-white shadow-[0_0_18px_rgba(107,255,255,0.25)] border border-white/20" : "border border-transparent hover:border-white/15 hover:bg-white/5"}`}
            >
              {tab.label}
            </button>
          );
        ))}
      </div>
    );
  }
}

```

app/organizador/dashboardUi.ts

```

export const DASHBOARD_SHELL_PADDING = "px-4 md:px-6 lg:px-8";
export const DASHBOARD_CARD = "rounded-2xl border border-white/10 bg-white/5 shadow-[0_16px_60px_rgba(0,0,0,0.35)]";
export const DASHBOARD_MUTED = "text-white/65";
export const DASHBOARD_HEADING = "text-lg font-semibold text-white";
export const DASHBOARD_SUBHEADING = "text-sm text-white/70";
export const DASHBOARD_SKELETON = "animate-pulse rounded-2xl border border-white/5 bg-white/5";

export const DASHBOARD_TITLE = "text-2xl font-semibold text-white";
export const DASHBOARD_LABEL = "text-[11px] uppercase tracking-[0.22em] text-white/55";

```

app/organizador/estatisticas/page.tsx

```

"use client";

import { useEffect } from "react";
import { useRouter, useSearchParams } from "next/navigation";

// LEGACY – estatísticas vivem em Bilhetes & Vendas e Finanças
export default function OrganizerStatsLegacy() {
  const router = useRouter();
  const searchParams = useSearchParams();

  useEffect(() => {
    const params = new URLSearchParams(searchParams?.toString() || "");
    params.set("tab", "sales");
    router.replace(`/organizador?${params.toString()}`);
  }, [router, searchParams]);

  return (
    <div className="max-w-3xl mx-auto px-4 py-10 text-white">
      <p className="text-sm text-white/70">
        Estatísticas foram integradas em Bilhetes & Vendas e Finanças. A redirecionar...
      </p>
    </div>
  );
}

```

app/organizador/faturacao/page.tsx

```

// app/organizador/faturacao/page.tsx
"use client";

import useSWR from "swr";
import { useState } from "react";
import { useRouter } from "next/navigation";

```

```

const fetcher = (url: string) => fetch(url).then((r) => r.json());

export default function OrganizerFinanceDashboard() {
  const { data, error } = useSWR("/api/organizador/faturacao", fetcher, { revalidateOnFocus: false });
  const router = useRouter();
  const [showTests, setShowTests] = useState(false);

  if (error) return <div className="p-4 text-white/70">Erro ao carregar faturação.</div>;
  if (!data) return <div className="p-4 text-white/70">A carregar...</div>;
  if (!data.ok) {
    if (data.error === "UNAUTHENTICATED") router.push("/login");
    if (data.error === "FORBIDDEN") router.push("/organizador");
    return <div className="p-4 text-white/70">Sem acesso.</div>;
  }

  const summary = data.summary || {};
  const events = (data.events || []).filter((e: any) => showTests || !e.isTest);

  return (
    <div className="space-y-4 rounded-2xl border border-white/10 bg-black/40 p-4">
      <div className="flex items-center justify-between">
        <div>
          <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Faturação</p>
          <h1 className="text-xl font-semibold text-white">Receita & Pagamentos</h1>
          <label className="flex items-center gap-1 text-white/70 text-sm mt-1">
            <input type="checkbox" checked={showTests} onChange={(e) => setShowTests(e.target.checked)} />
            Mostrar eventos de teste
          </label>
        </div>
        <div className="rounded-xl border border-white/15 bg-white/5 px-3 py-2 text-sm text-white/80">
          <p>Total bruto: {(summary.totalCents ?? 0) / 100} €</p>
          <p>Net: {(summary.netCents ?? 0) / 100} €</p>
          <p>Taxa plataforma: {(summary.platformFeeCents ?? 0) / 100} €</p>
          <p>Em reserva: {(summary.holdCents ?? 0) / 100} €</p>
        </div>
      </div>
      <div className="rounded-xl border border-white/10 bg-white/5 p-3 space-y-2 text-sm text-white/80">
        <div className="flex items-center justify-between">
          <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Eventos</p>
          <span className="text-white/60 text-[12px]">Filtre por "test" no toggle acima</span>
        </div>
        {events.length === 0 && <p className="text-white/60">Sem eventos.</p>}
        <div className="grid gap-2 md:grid-cols-2">
          {events.map((e: any) => {
            const kycIncomplete = e.connectStatus !== "READY";
            return (
              <div key={e.eventId} className="rounded-lg border border-white/10 bg-black/30 p-3 space-y-1">
                <div className="flex items-center justify-between">
                  <p className="text-white font-semibold">{e.title}</p>
                  {e.isTest && <span className="text-[11px] text-white/50">TEST</span>}
                </div>
                <p className="text-white/70 text-[12px]">Total: {(e.totalCents ?? 0) / 100} € · Vendas: {e.countSales}</p>
                <p className="text-white/60 text-[12px]">Release: {e.releaseAt ? new Date(e.releaseAt).toLocaleDateString("pt-PT") : "N/D"} · Hold: {(e.holdCents ?? 0) / 100} €</p>
                <p>Connect: {e.connectStatus}</p>
                {kycIncomplete && (
                  <div className="rounded border border-amber-400/40 bg-amber-500/10 px-2 py-1 text-[11px] text-amber-100">
                    Liga a Stripe/KYC para desbloquear pagamentos.
                  </div>
                )}
              </div>
            );
          })}
        </div>
      </div>
    </div>
  );
}

```

app/organizador/layout.tsx

```
export const runtime = "nodejs";
```

```

import { ReactNode } from "react";
import { redirect } from "next/navigation";
import { createSupabaseServer } from "@/lib/supabaseServer";

/**
 * Layout minimal para /organizador: apenas garante que o utilizador está autenticado.
 * O shell do dashboard (sidebar/topbar) vive em app/organizador/(dashboard)/layout.tsx.
 */
export default async function OrganizerLayout({ children }: { children: ReactNode }) {
  const supabase = await createSupabaseServer();
  const {
    data: { user },
  } = await supabase.auth.getUser();

  if (!user) {
    redirect("/login?next=/organizador");
  }

  return <>{children}</>;
}

```

app/organizador/OrganizationActions.tsx

```

"use client";

import Link from "next/link";

export function OrganizationActions({ organizerId }: { organizerId: number }) {
  return (
    <div className="space-y-3 rounded-2xl border border-white/10 bg-black/30 p-4 text-sm">
      <div className="space-y-1">
        <h3 className="text-base font-semibold text-white">Gestão da organização</h3>
        <p className="text-[12px] text-white/70">
          Transferências, convites e saída movidos para a página de Staff. Apagar organização está em Definições.
        </p>
      </div>
      <div className="flex flex-wrap gap-2">
        <Link
          href={`/organizador/staff${organizerId ? `?organizerId=${organizerId}` : ""}`}
          className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow hover:opacity-90"
        >
          Abrir Staff
        </Link>
        <Link
          href="/organizador/settings"
          className="rounded-full border border-white/20 bg-white/5 px-4 py-2 text-sm text-white hover:bg-white/10"
        >
          Ir a Definições
        </Link>
      </div>
    </div>
  );
}

```

app/organizador/organizations/OrganizationsHubClient.tsx

```

"use client";

import { useState } from "react";
import { useRouter } from "next/navigation";
import { sanitizeUsername, validateUsername, USERNAME_RULES_HINT } from "@/lib/username";

type OrgItem = {
  organizerId: number;
  role: string;
  lastUsedAt: string | null;
  organizer: {
    id: number;
    username: string | null;
    displayName: string | null;
    businessName: string | null;
    city: string | null;
    entityType: string | null;
    status: string | null;
  }
};

```

```

        brandingAvatarUrl?: string | null;
    };
};

type Props = {
    initialOrgs: OrgItem[];
    activeId: number | null;
};

export default function OrganizationsHubClient({ initialOrgs, activeId }: Props) {
    const router = useRouter();

    const [orgs, setOrgs] = useState<OrgItem[]>(initialOrgs);
    const [currentActive, setCurrentActive] = useState<number | null>(activeId);
    const [businessName, setBusinessName] = useState("");
    const [entityType, setEntityType] = useState("");
    const [city, setCity] = useState("");
    const [orgUsername, setOrgUsername] = useState("");
    const [usernameHint, setUsernameHint] = useState<string | null>(null);
    const [checkingUsername, setCheckingUsername] = useState(false);
    const [usernameStatus, setUsernameStatus] = useState<"idle" | "checking" | "available" | "taken" | "error">("idle");
    const [saving, setSaving] = useState(false);
    const [error, setError] = useState<string | null>(null);
    const [actionMessage, setActionMessage] = useState<string | null>(null);
    const [loadingSwitch, setLoadingSwitch] = useState(false);
    const [showForm, setShowForm] = useState(false);

    const checkUsernameAvailability = async (value: string) => {
        const cleaned = sanitizeUsername(value);
        if (!cleaned) {
            setUsernameHint(USERNAME_RULES_HINT);
            setUsernameStatus("idle");
            return false;
        }
        const validation = validateUsername(cleaned);
        if (!validation.valid) {
            setUsernameHint(validation.error);
            setUsernameStatus("error");
            return false;
        }
        setUsernameHint(null);
        setUsernameStatus("checking");
        setCheckingUsername(true);
        try {
            const res = await fetch(`/api/username/check?username=${encodeURIComponent(cleaned)}`);
            if (!res.ok) {
                setUsernameHint("Não foi possível verificar o @ agora.");
                setUsernameStatus("error");
                return false;
            }
            const data = (await res.json()).catch(() => null) as { available?: boolean } | null;
            const available = Boolean(data?.available);
            if (!available) setUsernameHint("Este @ já está a ser usado – escolhe outro.");
            setUsernameStatus(available ? "available" : "taken");
            return available;
        } catch (err) {
            console.error("[org hub] check username error", err);
            setUsernameHint("Erro ao verificar o @.");
            setUsernameStatus("error");
            return false;
        } finally {
            setCheckingUsername(false);
        }
    };

    const handleSwitch = async (organizerId: number, redirectToDashboard = false) => {
        if (loadingSwitch) return;
        setLoadingSwitch(true);
        setActionMessage(null);
        try {
            const res = await fetch("/api/organizador/organizations/switch", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ organizerId }),
            });
            const json = await res.json().catch(() => null);
            if (!res.ok || json?.ok === false) {
                setActionMessage(json?.error || "Não foi possível mudar de organização.");
            }
        } catch (err) {
            console.error("[org hub] switch error", err);
        } finally {
            setLoadingSwitch(false);
        }
    };
}

```

```

        return;
    }
    setCurrentActive(organizerId);
    if (redirectToDashboard) {
        // força cookie no browser e navegação direta com org na query
        try {
            document.cookie = `orya_org=${organizerId}; path=/; SameSite=Lax`;
        } catch (err) {
            console.warn("[org switch] não foi possível escrever cookie no browser", err);
        }
        // usa router para evitar cache, depois fallback para reload completo
        router.replace(`/organizador?tab=overview&org=${organizerId}`);
        setTimeout(() => {
            if (window?.location.href.includes(`org=${organizerId}`) === false) {
                window.location.href = `/organizador?tab=overview&org=${organizerId}`;
            }
        }, 50);
    } else {
        setActionMessage("Organização ativa atualizada.");
    }
} catch (err) {
    console.error("[org hub] switch error", err);
    setActionMessage("Erro inesperado ao mudar de organização.");
} finally {
    setLoadingSwitch(false);
}
};

const handleCreate = async () => {
    if (!businessName.trim() || !entityType.trim() || !city.trim()) {
        setError("Preenche nome, tipo de entidade e cidade.");
        return;
    }
    const usernameValid = validateUsername(orgUsername);
    if (!usernameValid.valid) {
        setError(usernameValid.error);
        return;
    }
    const available = await checkUsernameAvailability(orgUsername);
    if (!available) {
        setError("Este @ já está a ser usado – escolhe outro.");
        return;
    }
    setSaving(true);
    setError(null);
    setActionMessage(null);
    try {
        const res = await fetch("/api/organizador/organizations", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                businessName: businessName.trim(),
                entityType: entityType.trim(),
                city: city.trim(),
                displayName: businessName.trim(),
                username: usernameValid.normalized,
            }),
        });
        const json = await res.json().catch(() => null);
        if (!res.ok || json?.ok === false) {
            setError(json?.error || "Não foi possível criar a organização.");
        } else {
            const newId = json?.organizer?.id as number | undefined;
            setBusinessName("");
            setCity("");
            setEntityType("");
            setOrgUsername("");
            setUsernameStatus("idle");
            setUsernameHint(null);
            if (newId) {
                // Atualiza lista localmente para evitar refetch
                setOrgs((prev) => [
                    ...prev,
                    {
                        organizerId: newId,
                        role: "OWNER",
                        lastUsedAt: null,
                        organizer: {
                            name: businessName,
                            type: entityType,
                            city: city,
                            status: "idle",
                            hint: null,
                            id: newId,
                        },
                    },
                ]);
            }
        }
    } catch (err) {
        console.error("Error creating organization", err);
    }
};

```

```

        id: newId,
        username: usernameValid.normalized,
        displayName: json.organizer.displayName ?? json.organizer.businessName ?? "Organização",
        businessName: json.organizer.businessName ?? null,
        city: json.organizer.city ?? null,
        entityType: json.organizer.entityType ?? null,
        status: "ACTIVE",
    },
},
);
await handleSwitch(newId, true);
} else {
    router.push("/organizador?tab=overview");
}
}
} catch (err) {
    console.error("[org hub] create error", err);
    setError("Erro inesperado ao criar organização.");
} finally {
    setSaving(false);
}
};

const renderOrgCard = (item: OrgItem) => {
    const isActive = currentActive === item.organizerId;
    const normalizedRole = item.role.toUpperCase();
    const isOwnerOrAdmin = ["OWNER", "CO_OWNER", "ADMIN"].includes(normalizedRole);
    const cityLine = item.organizer.city || "Cidade não definida";
    const typeLine = item.organizer.entityType || "Tipo não definido";
    const handle = item.organizer.username ? `@${item.organizer.username}` : "Sem username";
    const roleLabel = normalizedRole;
    const statusLabel = (item.organizer.status || "-").toUpperCase();

    const badgeClass = (kind: "status" | "role", value: string) => {
        if (kind === "status" && value === "ACTIVE") {
            return "border-emerald-400/50 bg-emerald-400/15 text-emerald-50";
        }
        if (kind === "status" && value === "PENDING") {
            return "border-amber-400/50 bg-amber-400/15 text-amber-50";
        }
        if (kind === "status" && value === "SUSPENDED") {
            return "border-red-400/50 bg-red-400/15 text-red-50";
        }
        if (kind === "role" && value === "OWNER") {
            return "border-cyan-300/60 bg-cyan-300/15 text-cyan-50";
        }
        if (kind === "role" && value === "ADMIN") {
            return "border-sky-300/60 bg-sky-300/15 text-sky-50";
        }
        return "border-white/20 bg-white/10 text-white/70";
    };
    return (
        <div
            key={item.organizerId}
            className={`${rounded-2xl border p-4 shadow-[0_16px_50px_rgba(0,0,0,0.45)] transition hover:-translate-y-[3px]
hover:border-[#6BFFFF]/50 hover:bg-white/8 ${` ${isActive ? "border-[#6BFFFF]/60 bg-[#0b152d]/50" : "border-white/10 bg-white/5"}`}`}
        >
            <div className="flex items-start justify-between gap-2">
                <div className="flex items-start gap-3">
                    <div className="h-10 w-10 rounded-full border border-white/15 bg-white/10 overflow-hidden">
                        {item.organizer.brandingAvatarUrl ? (
                            // eslint-disable-next-line @next/next/no-img-element
                            <img
                                src={item.organizer.brandingAvatarUrl}
                                alt={item.organizer.displayName || "Organização"}
                                className="h-full w-full object-cover"
                            />
                        ) : (
                            <div className="flex h-full w-full items-center justify-center text-sm font-semibold">
                                {((item.organizer.displayName || item.organizer.businessName || "0")[0]}

                            </div>
                        )}
                    </div>
                    <div className="space-y-1">
                        <h3 className="text-lg font-semibold">

```

```

        {item.organizer.displayName || item.organizer.businessName || "Organização"}
    </h3>
    <p className="text-[12px] text-white/60">
        {handle} · {typeLine}
    </p>
    </div>
</div>
<div className="flex flex-col items-end gap-1 text-[11px]">
    <span
        className={`${rounded-full border px-3 py-[5px] uppercase tracking-[0.2em] text-[10px]} ${badgeClass("status", statusLabel)}}`>
        {statusLabel}
    </span>
    <span
        className={`${rounded-full border px-3 py-[5px] uppercase tracking-[0.2em] text-[10px]} ${badgeClass("role", roleLabel)}}`>
        {roleLabel}
    </span>
</div>
</div>

<div className="mt-4 flex flex-wrap items-center gap-2 text-[12px] text-white/80">
    {isActive} ? (
        <button
            type="button"
            disabled
            className="rounded-full border border-emerald-400/50 bg-emerald-400/15 px-5 py-2 text-sm font-semibold text-emerald-50"
        >
            Já estás neste dashboard
        </button>
    ) : (
        <button
            type="button"
            onClick={() => handleSwitch(item.organizerId, true)}
            className="rounded-full border border-white/25 bg-white/10 px-5 py-2 text-sm font-semibold text-white hover:bg-white/15 transition"
        >
            Entrar no dashboard de {item.organizer.displayName || "organização"}
        </button>
    )
    {isOwnerOrAdmin && (
        <button
            type="button"
            onClick={() => router.push(`/organizador/staff?organizerId=${item.organizerId}`)}
            className="rounded-full border border-white/20 bg-white/5 px-5 py-2 text-sm text-white hover:bg-white/10 transition"
        >
            Gerir equipa
        </button>
    )}
</div>
</div>
);
};

const loading = false; // server já enviou dados; não repetir fetch
const emptyState = orgs.length === 0;
const hasError = false; // como não há fetch client, não há erro aqui

return (
    <div className="orya-body-bg min-h-screen text-white px-4 py-10 md:px-8 lg:px-12">
        <div className="mx-auto max-w-6xl space-y-8">
            {hasError && (
                <div className="rounded-2xl border border-red-400/40 bg-red-900/30 p-4 text-sm text-red-100">
                    Não foi possível carregar as organizações neste momento.
                </div>
            )}
            {loading && (
                <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-3">
                    {[...Array(4)].map((_, i) => (
                        <div
                            key={i}
                            className="h-40 rounded-2xl border border-white/10 bg-white/5 animate-pulse"
                        />
                    ))}
                </div>
            )}
        </div>
    </div>
);

```

```

        ))}
    </div>
)}

{!loading && !hasError && !emptyState && (
<section className="space-y-3">
    <div className="space-y-1">
        <h2 className="text-lg font-semibold">As tuas organizações</h2>
        <p className="text-[12px] text-white/65">Escolhe em que organização estás a trabalhar.</p>
    </div>
    <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-3">
        {orgs.map(renderOrgCard)}
        <button
            type="button"
            onClick={() => router.push("/organizador/become")}
            className="flex flex-col justify-between rounded-2xl border border-dashed border-white/20 bg-white/5 p-4 shadow-[0_16px_50px_rgba(0,0,0,0.35)] hover:-translate-y-[3px] hover:border-white/30 transition text-left"
        >
            <div className="space-y-2">
                <div className="flex items-center gap-2">
                    <div className="flex h-12 w-12 items-center justify-center rounded-full border border-white/20 bg-white/10 text-xl font-bold">
                        +
                    </div>
                </div>
                <h3 className="text-lg font-semibold">Adicionar nova organização</h3>
            </div>
            </button>
        </div>
    </div>
</section>
)}

{emptyState && !hasError && (
<div className="rounded-3xl border border-white/10 bg-white/5 p-6 shadow-[0_20px_60px_rgba(0,0,0,0.5)] space-y-4">
    <div className="space-y-2">
        <h2 className="text-2xl font-semibold">Ainda não tens nenhuma organização</h2>
        <p className="text-sm text-white/65">
            Cria a primeira para vender bilhetes, gerir equipa e pagamentos.
        </p>
        <ul className="list-disc space-y-1 pl-5 text-sm text-white/70">
            <li>Cria o teu clube, bar, espaço ou marca.</li>
            <li>Vende bilhetes e recebe os pagamentos diretamente na tua conta.</li>
            <li>Adiciona staff com acesso a check-in e relatórios.</li>
        </ul>
    </div>
    <button
        type="button"
        onClick={() => router.push("/organizador/become")}
        className="rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-5 py-2 text-sm font-semibold text-black shadow hover:brightness-110"
    >
        Criar primeira organização
    </button>
</div>
)}

{showForm && (
<div
    className={`${emptyState ? "" : "fixed inset-0 z-50 flex items-center justify-center bg-black/70 px-4 backdrop-blur"}`}
>
    <section className="w-full max-w-4xl rounded-2xl border border-white/10 bg-white/5 p-5 space-y-3 shadow-[0_18px_60px_rgba(0,0,0,0.45)]">
        <div className="flex items-center justify-between">
            <div>
                <h3 className="text-lg font-semibold">Criar nova organização</h3>
                <p className="text-[12px] text-white/65">Define nome, tipo de entidade e cidade base para começares.</p>
            </div>
            <div className="flex items-center gap-3">
                {actionMessage && <p className="text-[12px] text-emerald-200">{actionMessage}</p>}
                {!emptyState &&
                    <button
                        type="button"
                        onClick={() => setShowForm(false)}
                        className="rounded-full border border-white/20 bg-white/5 px-3 py-1 text-[12px] text-white hover:bg-white/10"
                    >
                        Fechar
                    </button>
                }
            </div>
        </div>
    </section>
</div>
)
}

```

```

        </button>
    )}
</div>
</div>
<div className="grid gap-3 md:grid-cols-3">
    <div className="space-y-1 md:col-span-1">
        <label className="text-[12px] text-white/70">Nome da organização</label>
        <input
            value={businessName}
            onChange={(e) => setBusinessName(e.target.value)}
            className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]" placeholder="Ex.: ORYA TEAM, Casa Guedes"
        />
    </div>
    <div className="space-y-1">
        <label className="text-[12px] text-white/70">Cidade base</label>
        <input
            value={city}
            onChange={(e) => setCity(e.target.value)}
            className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]" placeholder="Lisboa, Porto..."'
        />
    </div>
    <div className="space-y-1">
        <label className="text-[12px] text-white/70">Tipo de entidade</label>
        <select
            value={entityType}
            onChange={(e) => setEntityType(e.target.value)}
            className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-
[#6BFFFF]">
            <option value="">Seleciona</option>
            <option value="PESSOA_SINGULAR">Pessoa singular</option>
            <option value="EMPRESA">Empresa</option>
            <option value="ASSOCIACAO">Associação</option>
        </select>
    </div>
    <div className="space-y-1">
        <label className="text-[12px] text-white/70">Username ORYA</label>
        <div className="relative">
            <span className="pointer-events-none absolute left-3 top-1/2 -translate-y-1/2 text-white/50 text-
sm">@</span>
            <input
                value={orgUsername}
                onChange={(e) => {
                    const cleaned = sanitizeUsername(e.target.value);
                    setOrgUsername(cleaned);
                    const validation = validateUsername(cleaned);
                    setUsernameHint(validation.valid ? null : validation.error);
                    setUsernameStatus("idle");
                }}
                onBlur={(e) => checkUsernameAvailability(e.target.value)}
                className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 pl-7 text-sm outline-none
focus:border-[#6BFFFF]" maxLength={30}
                placeholder="casaguedes"
            />
        </div>
        <p className="text-[11px] text-white/55">@ é único na ORYA (3-30 chars, letras/números/_ ou .)</p>
        {usernameHint && <p className="text-[11px] text-amber-300">{usernameHint}</p>}
        {checkingUsername && <p className="text-[11px] text-white/60">A verificar disponibilidade...</p>}
        {usernameStatus === "taken" && !checkingUsername && (
            <p className="text-[11px] text-red-300">Este @ já está a ser usado.</p>
        )}
        {usernameStatus === "available" && !checkingUsername && (
            <p className="text-[11px] text-emerald-300">Disponível ✓</p>
        )}
    </div>
</div>
{error && <p className="text-sm text-red-300">{error}</p>}
<button
    type="button"
    onClick={handleCreate}
    disabled={
        saving ||
        !businessName.trim() ||

```

```

        !entityType.trim() ||
        !city.trim() ||
        !validateUsername(sanitizeUsername(orgUsername)).valid
    }
    className="self-start rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-5 py-2 text-sm
font-semibold text-black shadow hover:brightness-110 disabled:opacity-60"
    >
    {saving ? "A criar..." : "Criar organização"}
  </button>
</section>
</div>
)
</div>
</div>
);
}
}

```

app/organizador/OrganizationSwitcher.tsx

```

"use client";

import { useEffect, useMemo, useState } from "react";
import Link from "next/link";

export type OrganizationSwitcherOption = {
  organizerId: number;
  role: string;
  organizer: {
    id: number;
    username: string | null;
    publicName?: string | null;
    displayName: string | null;
    businessName: string | null;
    city: string | null;
    entityType: string | null;
    status: string | null;
    brandingAvatarUrl?: string | null;
  };
};

type Props = {
  currentId: number | null;
  initialOrgs?: OrganizationSwitcherOption[];
};

export function OrganizationSwitcher({ currentId, initialOrgs = [] }: Props) {
  const [options, setOptions] = useState<OrganizationSwitcherOption[]>(initialOrgs);
  const [activeId, setActiveId] = useState<number | null>(currentId);

  useEffect(() => {
    setActiveId(currentId);
    setOptions(initialOrgs);
  }, [currentId, initialOrgs]);

  const current = useMemo(
    () => options.find((i) => i.organizerId === activeId) ?? options[0] ?? null,
    [activeId, options],
  );

  if (!current) {
    return (
      <Link
        href="/organizador/organizations"
        className="flex items-center gap-2 rounded-full border border-white/10 bg-white/5 px-3 py-2 text-[12px] text-white/80
        hover:border-white/20"
        >
        Escolher organização
        </Link>
    );
  }

  return (
    <div className="relative">
      <details className="group">
        <summary className="flex cursor-pointer select-none items-center gap-2 rounded-full border border-white/15 bg-white/5
        px-2 py-1 text-[12px] text-white/85 transition hover:border-white/30">

```

```

{current.organizer.brandingAvatarUrl ? (
    // eslint-disable-next-line @next/next/no-img-element
    <img
        src={current.organizer.brandingAvatarUrl}
        alt={current.organizer.publicName || current.organizer.displayName || "Organização"}
        className="h-8 w-8 rounded-full border border-white/10 object-cover"
    />
) : (
    <span className="inline-flex h-8 w-8 items-center justify-center rounded-full border border-white/10 bg-white/5
text-[11px] font-semibold">
        {(current.organizer.publicName || current.organizer.displayName || current.organizer.businessName || "0")[0]}
    </span>
)
<span className="text-white/70 group-open:rotate-180 transition-transform pr-1">▼</span>
</summary>
<div className="absolute right-0 z-40 mt-2 w-56 rounded-2xl border border-white/10 bg-black/85 p-2 shadow-
[0_20px_40px_rgba(0,0,0,0.55)] backdrop-blur-xl">
    <div className="space-y-1">
        <Link
            href="/organizador/organizations"
            className="flex w-full items-center justify-between rounded-xl px-3 py-2 text-sm text-white hover:bg-white/10">
            <span>Gerir organizações</span>
            <span className="text-[10px] text-white/60"></span>
        </Link>
        {current.organizer.username && (
            <Link
                href={`/org/${current.organizer.username}`}
                target="_blank"
                rel="noopener"
                className="flex w-full items-center justify-between rounded-xl px-3 py-2 text-sm text-white hover:bg-white/10">
                <span>Página pública</span>
                <span className="text-[10px] text-white/60"></span>
            </Link>
        )}
        </div>
    </div>
</div>
</details>
</div>
);
}

```

app/organizador/OrganizerLangSetter.tsx

```

"use client";

import { useEffect } from "react";

export function OrganizerLangSetter({ language }: { language?: string | null }) {
    useEffect(() => {
        if (!language || typeof document === "undefined") return;
        const lang = language.toLowerCase() === "en" ? "en" : "pt";
        const previous = document.documentElement.lang;
        document.documentElement.lang = lang;
        return () => {
            if (previous) document.documentElement.lang = previous;
        };
    }, [language]);
    return null;
}

```

app/organizador/OrganizerSidebar.tsx

```

"use client";

import Link from "next/link";
import { usePathname, useSearchParams } from "next/navigation";
import { useState } from "react";
import { DASHBOARD_LABEL } from "./dashboardUi";

const baseTabHref = (tab: string) => `/organizador?tab=${tab}`;

```

```

type Props = {
  organizerName?: string | null;
  organizerAvatarUrl?: string | null;
};

export function OrganizerSidebar({ organizerName, organizerAvatarUrl }: Props) {
  const pathname = usePathname();
  const searchParams = useSearchParams();
  const tabParamRaw = searchParams?.get("tab") || "overview";
  const allowedTabs = ["overview", "events", "sales", "finance", "invoices", "marketing", "padel", "staff", "settings"] as const;
  const tabParam = allowedTabs.includes(tabParamRaw as any) ? tabParamRaw : "overview";
  const [catsOpen, setCatsOpen] = useState(true);

  const linkClass = (active: boolean) =>
    `flex items-center justify-between rounded-xl px-3 py-2 transition ${
      active ? "bg-white/10 text-white font-semibold border border-white/20" : "hover:bg-white/10"
    }`;

  const currentKey = (() => {
    if (pathname?.startsWith("/organizador/pagamentos/invoices")) return "invoices";
    if (pathname?.startsWith("/organizador/eventos/novo") || pathname?.endsWith("/edit")) return "create";
    if (pathname?.startsWith("/organizador/eventos/")) return "events";
    if (pathname?.startsWith("/organizador/staff")) return "staff";
    if (pathname?.startsWith("/organizador/settings")) return "settings";
    if (pathname === "/organizador") return tabParam;
    return tabParam;
  })();

  return (
    <aside className="hidden lg:flex w-60 shrink-0 flex-col gap-2 border-r border-white/10 bg-black/40 backdrop-blur-xl px-4 py-6 text-[13px] text-white/80 shadow-[0_18px_60px_rgba(0,0,0,0.55)] sticky top-0 h-screen overflow-y-auto">
      <div className="flex items-center gap-2 px-2">
        <div className="h-9 w-9 rounded-2xl bg-gradient-to-br from-[#0f172a] via-[#111827] to-[#0b1224] flex items-center justify-center text-xs font-black tracking-[0.2em] text-[#6BFFFF] shadow-[0_0_14px_rgba(107,255,255,0.3)] overflow-hidden border border-white/10">
          {organizerAvatarUrl ? (
            // eslint-disable-next-line @next/next/no-img-element
            <img src={organizerAvatarUrl} alt={organizerName || "Organizador"} className="h-full w-full object-cover" />
          ) : (
            "0Y"
          )}
        </div>
        <div>
          <p className={DASHBOARD_LABEL}>Dashboard</p>
          <p className="text-sm font-semibold text-white">{organizerName || "Organizador"}</p>
        </div>
      </div>
      <nav className="mt-4 space-y-1">
        <p className="px-2 text-[10px] uppercase tracking-[0.25em] text-white/40">Operações</p>
        <Link href={baseTabHref("overview")} className={linkClass(currentKey === "overview")}>
          <span>Resumo</span>
        </Link>
        <Link href={baseTabHref("events")} className={linkClass(currentKey === "events")}>
          <span>Eventos</span>
        </Link>
        <Link href="/organizador/eventos/novo" className={linkClass(currentKey === "create")}>
          <span>Criar evento</span>
        </Link>
        <Link href={baseTabHref("sales")} className={linkClass(currentKey === "sales")}>
          <span>Vendas</span>
        </Link>
        <Link href={baseTabHref("finance")} className={linkClass(currentKey === "finance")}>
          <span>Finanças</span>
        </Link>
        <Link href={baseTabHref("invoices")} className={linkClass(currentKey === "invoices")}>
          <span>Faturação</span>
        </Link>
        <Link href={baseTabHref("marketing")} className={linkClass(currentKey === "marketing")}>
          <span>Marketing</span>
        </Link>
      </nav>
      <div className="border-t border-white/10 pt-2" />
      <button>

```

```

        type="button"
        onClick={() => setCatsOpen((v) => !v)}
        className="flex w-full items-center justify-between rounded-xl px-3 py-2 text-left transition hover:bg-white/10"
      >
      <span className="text-[10px] uppercase tracking-[0.25em] text-white/60">Categorias</span>
      <span className="text-white/60">(catsOpen ? "▲" : "▼")</span>
    </button>
  {catsOpen && (
    <div className="space-y-1">
      <Link href={baseTabHref("padel")} className={linkClass(currentKey === "padel")}>
        <span>Padel</span>
      </Link>
      {[{
        { key: "restauracao", label: "Restauração" },
        { key: "solidario", label: "Solidário" },
        { key: "festas", label: "Festas" },
        { key: "outro", label: "Outro tipo" },
      ]}.map((item) => (
        <div key={item.key} className="flex items-center justify-between rounded-xl px-3 py-2 text-white/60">
          <span>{item.label}</span>
          <span className="rounded-full border border-amber-300/30 bg-amber-400/15 px-2 py-[1px] uppercase tracking-[0.12em] text-amber-100">
            Em breve
          </span>
        </div>
      )))
    </div>
  )}
<p className="px-2 pt-2 uppercase tracking-[0.25em] text-white/40">Estrutura</p>
<Link href={baseTabHref("staff")} className={linkClass(currentKey === "staff")}>
  <span>Staff</span>
</Link>
<Link href={baseTabHref("settings")} className={linkClass(currentKey === "settings")}>
  <span>Definições</span>
</Link>
</nav>
</aside>
);
}

```

app/organizador/OrganizerTour.tsx

```

"use client";

import { useEffect, useMemo, useState } from "react";
import { trackEvent } from "@/lib/analytics";

type Step = {
  title: string;
  body: string;
  anchor?: string;
};

const steps: Step[] = [
  {
    title: "Bem-vindo ao painel de organizador",
    body: "Aqui geres eventos, vendas, finanças e marketing num só lugar.",
  },
  {
    title: "Criar evento",
    body: "Começa sempre aqui. Usa templates de Padel ou eventos gerais e publica em minutos.",
    anchor: "[data-tour='criar-evento']",
  },
  {
    title: "Finanças & Stripe",
    body: "Liga o Stripe, acompanha receita e payouts. Se precisares de atenção, mostramos-te logo aqui.",
    anchor: "[data-tour='finance']",
  },
  {
    title: "Marketing & códigos",
    body: "Códigos promocionais e boosts para encher eventos mais rápido.",
    anchor: "[data-tour='marketing']",
  },
  {
    title: "Voltar à experiência de utilizador",
  }
];

```

```

    body: "Podes ver sempre como o público vê os teus eventos e inscrições.",
    anchor: "[data-tour='user-experience']",
  },
];

const TOUR_KEY = "orya_org_tour_seen_v1";
const TOUR_EVENT = "orya:startTour";

export function OrganizerTour() {
  const [open, setOpen] = useState(false);
  const [index, setIndex] = useState(0);
  const [mounted, setMounted] = useState(false);
  const [viewport, setViewport] = useState({ width: number; height: number }>({ width: 0, height: 0 }));
  const [anchorRect, setAnchorRect] = useState<DOMRect | null>(null);

  const shouldShow = mounted && open;

  useEffect(() => {
    setMounted(true);
    const seen = typeof window !== "undefined" ? localStorage.getItem(TOUR_KEY) : "1";
    if (!seen) setOpen(true);
    const handler = () => {
      localStorage.removeItem(TOUR_KEY);
      setIndex(0);
      setOpen(true);
    };
    window.addEventListener(TOUR_EVENT, handler);
    return () => window.removeEventListener(TOUR_EVENT, handler);
  }, []);

  useEffect(() => {
    const updateViewport = () => {
      if (typeof window === "undefined") return;
      setViewport({ width: window.innerWidth, height: window.innerHeight });
    };
    updateViewport();
    window.addEventListener("resize", updateViewport);
    return () => window.removeEventListener("resize", updateViewport);
  }, []);

  const step = useMemo(() => steps[index], [index]);
  const isMobile = viewport.width < 768 && viewport.width > 0;

  useEffect(() => {
    if (!shouldShow) return;
    if (!step.anchor) {
      setAnchorRect(null);
      return;
    }
    const el = document.querySelector(step.anchor);
    if (!el) {
      setAnchorRect(null);
      return;
    }
    const rect = el.getBoundingClientRect();
    setAnchorRect(rect);
    const observer = new ResizeObserver(() => {
      const nextRect = el.getBoundingClientRect();
      setAnchorRect(nextRect);
    });
    observer.observe(el);
    return () => observer.disconnect();
  }, [shouldShow, step.anchor]);

  const goNext = () => {
    trackEvent("organizer_tour_next", { step: index });
    if (index < steps.length - 1) {
      setIndex((v) => v + 1);
    } else {
      finish();
    }
};

const finish = () => {
  trackEvent("organizer_tour_finish");
  localStorage.setItem(TOUR_KEY, "1");
  setOpen(false);
};

```

```

useEffect(() => {
  if (shouldShow && anchorRect && step.anchor) {
    const el = document.querySelector(step.anchor);
    el?.scrollIntoView({ behavior: "smooth", block: "center" });
  }
}, [anchorRect, shouldShow, step.anchor]);

useEffect(() => {
  const onKey = (e: KeyboardEvent) => {
    if (e.key === "Escape") {
      finish();
    }
  };
  if (shouldShow) {
    document.body.style.overflow = "hidden";
    window.addEventListener("keydown", onKey);
  }
  return () => {
    document.body.style.overflow = "";
    window.removeEventListener("keydown", onKey);
  };
}, [shouldShow]);

if (!shouldShow) return null;

const cardWidth = isMobile ? viewport.width - 32 : Math.min(480, viewport.width - 32);
const margin = 16;
const estimatedHeight = isMobile ? 260 : 240;
let cardLeft = (viewport.width - cardWidth) / 2;
let cardTop = isMobile ? viewport.height - estimatedHeight - 24 : 96;
let arrowPos: { x: number; y: number; side: "top" | "bottom" | "left" | "right" } | null = null;

if (!isMobile && anchorRect) {
  const spaceBelow = viewport.height - anchorRect.bottom - margin;
  const spaceAbove = anchorRect.top - margin;
  const centerX = anchorRect.left + anchorRect.width / 2;
  cardLeft = Math.max(margin, Math.min(viewport.width - cardWidth - margin, centerX - cardWidth / 2));
  if (spaceBelow >= estimatedHeight) {
    cardTop = anchorRect.bottom + margin;
    arrowPos = { x: Math.min(cardWidth - 32, Math.max(32, centerX - cardLeft)), y: -12, side: "top" };
  } else if (spaceAbove >= estimatedHeight) {
    cardTop = Math.max(margin, anchorRect.top - estimatedHeight - margin);
    arrowPos = { x: Math.min(cardWidth - 32, Math.max(32, centerX - cardLeft)), y: estimatedHeight - 4, side: "bottom" };
  } else {
    cardTop = Math.max(margin, Math.min(viewport.height - estimatedHeight - margin, anchorRect.bottom + margin));
    arrowPos = { x: Math.min(cardWidth - 32, Math.max(32, centerX - cardLeft)), y: -12, side: "top" };
  }
}

return (
  <div className="fixed inset-0 z-[99] pointer-events-none">
    <div className="absolute inset-0 bg-[radial-gradient(circle_at_30%_20%,rgba(107,255,255,0.08),rgba(0,0,0,0),rgba(0,0,0,0.6)] backdrop-blur-sm" />
    {!isMobile && anchorRect && (
      <div
        className="absolute rounded-2xl pointer-events-none"
        style={{
          left: anchorRect.left - 6,
          top: anchorRect.top - 6,
          width: anchorRect.width + 12,
          height: anchorRect.height + 12,
          boxShadow: "0 0 0 1px rgba(107,255,255,0.35), 0 0 24px rgba(107,255,255,0.18)",
          background: "radial-gradient(circle at center, rgba(107,255,255,0.06), rgba(7,11,19,0))",
        }}
      />
    )}
    <div
      className="absolute rounded-2xl border border-white/10 bg-black/80 backdrop-blur-xl p-5 shadow-[0_30px_120px_rgba(0,0,0,0.7)] pointer-events-auto"
      style={{
        width: cardWidth,
        left: cardLeft,
        top: cardTop,
        maxHeight: isMobile ? "70vh" : "60vh",
        overflow: "auto",
      }}
    >

```

```

{!isMobile && arrowPos && (
  <div
    className={`absolute h-3 w-3 rotate-45 border border-white/15 bg-black/80`}
    style={{
      left: arrowPos.side === "top" || arrowPos.side === "bottom" ? arrowPos.x - 6 : arrowPos.side === "left" ? -6 : cardWidth - 10,
      top: arrowPos.side === "top" ? arrowPos.y : arrowPos.side === "bottom" ? undefined : cardTop + estimatedHeight / 2,
      bottom: arrowPos.side === "bottom" ? -6 : undefined,
    }}
  />
)}
<div className="flex items-start justify-between gap-2">
  <div className="space-y-1">
    <p className="text-[11px] uppercase tracking-[0.18em] text-white/50">Tour</p>
    <h3 className="text-lg font-semibold text-white">{step.title}</h3>
    <p className="text-sm text-white/70">{step.body}</p>
  </div>
  <button
    onClick={finish}
    className="text-white/60 hover:text-white rounded-full p-1 transition"
    aria-label="Fechar tour"
  >
    <span>
      Passo {index + 1} / {steps.length}
    </span>
  </button>
</div>
<div className="mt-4 flex items-center justify-between text-[12px] text-white/60">
  <span>
    Passo {index + 1} / {steps.length}
  </span>
  <div className="flex gap-2">
    <button
      onClick={finish}
      className="rounded-full border border-white/20 px-3 py-1 text-white/75 hover:bg-white/10"
    >
      Saltar
    </button>
    <button
      onClick={goNext}
      className="rounded-full bg-white text-black px-4 py-1.5 font-semibold hover:scale-[1.01] active:scale-95 transition"
    >
      {index === steps.length - 1 ? "Terminar" : "Seguinte"}
    </button>
  </div>
</div>
</div>
);
}

```

app/organizador/OrganizerTourTrigger.tsx

```

"use client";

export function OrganizerTourTrigger() {
  return (
    <button
      type="button"
      onClick={() => window.dispatchEvent(new Event("orya:startTour"))}
      className="rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-white/80 hover:bg-white/10"
    >
      Ver tour
    </button>
  );
}

```

app/organizador/pagamentos/invoices/invoices-client.tsx

```

"use client";

import useSWR from "swr";
import { useSearchParams, useRouter } from "next/navigation";
import { formatMoney } from "@lib/money";

```

```

import { useUser } from "@/app/hooks/useUser";

const fetcher = (url: string) => fetch(url).then((r) => r.json());

function toQuery(params: Record<string, string | number | null | undefined>) {
  const url = new URLSearchParams();
  Object.entries(params).forEach(([k, v]) => {
    if (v !== null && v !== undefined && String(v).trim() !== "") {
      url.set(k, String(v));
    }
  });
  const qs = url.toString();
  return qs ? `?${qs}` : "";
}

export default function InvoicesClient() {
  const searchParams = useSearchParams();
  const router = useRouter();
  const { organizerId: orgIdFromProfile } = useUser()?.organizer ?? {};
  const organizerIdParam = searchParams?.get("organizerId") ?? (orgIdFromProfile ? String(orgIdFromProfile) : null);
  const from = searchParams?.get("from") ?? "";
  const to = searchParams?.get("to") ?? "";
  const organizerId = organizerIdParam ? Number(organizerIdParam) : null;
  const qs = toQuery({ organizerId, from, to });
  const { data, isLoading } = useSWR(() => (organizerId ? `/api/organizador/pagamentos/invoices${qs}` : null), fetcher, {
    revalidateOnFocus: false,
  });

  const summary = data?.summary ?? { grossCents: 0, discountCents: 0, platformFeeCents: 0, netCents: 0, tickets: 0 };

  const downloadCsv = () => {
    const rows = [
      ["Data", "Evento", "Payout Mode", "Subtotal", "Desconto", "Taxas", "Total", "Líquido", "Bilhetes"],
      ... (data?.items || []).map((sale: any) => [
        sale.createdAt,
        sale.event?.title ?? "",
        sale.event?.payoutMode ?? "",
        (sale.subtotalCents / 100).toFixed(2),
        (sale.discountCents / 100).toFixed(2),
        (sale.platformFeeCents / 100).toFixed(2),
        (sale.totalCents / 100).toFixed(2),
        (sale.netCents / 100).toFixed(2),
        sale.lines?.reduce((s: number, l: any) => s + l.quantity, 0) ?? 0,
      ]),
    ];
    const csvContent = rows.map((r) => r.join(",")).join("\n");
    const blob = new Blob([csvContent], { type: "text/csv;charset=utf-8;" });
    const url = URL.createObjectURL(blob);
    const a = document.createElement("a");
    a.href = url;
    a.download = "invoices.csv";
    a.click();
    URL.revokeObjectURL(url);
  };
}

return (
  <div className="mx-auto max-w-6xl px-4 py-6 space-y-5 text-white">
    <div className="flex flex-col gap-2 md:flex-row md:items-center md:justify-between">
      <div>
        <p className="text-[11px] uppercase tracking-[0.3em] text-white/60">Faturação</p>
        <h1 className="text-3xl font-semibold">Resumo de vendas</h1>
        <p className="text-sm text-white/65">Receita bruta, taxas e líquido por evento. Exporta CSV quando precisares.</p>
      </div>
      <div className="flex flex-wrap gap-2 text-[12px]">
        <input
          type="date"
          value={from}
          onChange={(e) =>
            router.push(`/organizador${toQuery({ tab: "invoices", organizerId, from: e.target.value, to })}`)
          }
          className="rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF]"
        />
        <input
          type="date"
          value={to}
          onChange={(e) =>
            router.push(`/organizador${toQuery({ tab: "invoices", organizerId, from, to: e.target.value })}`)
          }
        />
      </div>
    </div>
  </div>
)

```

```

        }
        className="rounded-lg border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none focus:border-[#6BFFFF]"
    />
    <button
      type="button"
      onClick={downloadCsv}
      className="rounded-full border border-white/20 bg-white/5 px-4 py-2 text-sm text-white hover:bg-white/10"
    >
      Exportar CSV
    </button>
  </div>
</div>

{isLoading ? (
  <div className="p-4 text-white/70">A carregar faturaçāo.</div>
) : !data || data.ok === false ? (
  <div className="p-4 text-white/80">Nāo foi possivel carregar faturaçāo.</div>
) : data.items.length === 0 ? (
  <div className="rounded-2xl border border-white/10 bg-white/5 p-5 text-white/70">
    Ainda nāo existem vendas neste intervalo. Ajusta as datas ou volta mais tarde.
  </div>
) : (
  <>
    <div className="grid gap-3 md:grid-cols-4">
      <SummaryCard label="Receita bruta" value={formatMoney(summary.grossCents / 100)} />
      <SummaryCard label="Descontos" value={formatMoney(summary.discountCents / 100)} muted />
      <SummaryCard label="Taxa ORYA" value={formatMoney(summary.platformFeeCents / 100)} muted />
      <SummaryCard label="Líquido" value={formatMoney(summary.netCents / 100)} highlight />
    </div>
  </>
  <div className="rounded-2xl border border-white/10 bg-white/5 p-4 shadow-[0_18px_50px_rgba(0,0,0,0.55)] overflow-x-
auto">
    <table className="min-w-full text-sm text-white/80">
      <thead className="text-left text-[11px] uppercase tracking-[0.16em] text-white/60">
        <tr>
          <th className="py-2 pr-3">Evento</th>
          <th className="py-2 pr-3">Data</th>
          <th className="py-2 pr-3">Subtotal</th>
          <th className="py-2 pr-3">Taxes</th>
          <th className="py-2 pr-3">Liquido</th>
        </tr>
      </thead>
      <tbody className="divide-y divide-white/10">
        {data.items.map((sale: any) =>
          <tr key={sale.id}>
            <td className="py-2 pr-3">
              <div className="font-semibold">{sale.event?.title ?? "Evento"}</div>
              <div className="text-[11px] text-white/60">{sale.event?.payoutMode ?? ""}</div>
            </td>
            <td className="py-2 pr-3 text-[12px] text-white/70">{new Date(sale.createdAt).toLocaleString("pt-PT")}</td>
            <td className="py-2 pr-3">{formatMoney(sale.subtotalCents / 100)}</td>
            <td className="py-2 pr-3">{formatMoney(sale.platformFeeCents / 100)}</td>
            <td className="py-2 pr-3 font-semibold">{formatMoney(sale.netCents / 100)}</td>
          </tr>
        )));
      </tbody>
    </table>
  </div>
  </>
)
</div>
);
}

function SummaryCard({ label, value, muted, highlight }: { label: string; value: string; muted?: boolean; highlight?: boolean }) {
  return (
    <div
      className={`rounded-2xl border border-white/10 p-4 shadow-sm ${highlight ? "bg-emerald-500/10 border-emerald-400/30 text-emerald-50" : muted ? "bg-white/5 text-white/60" : "bg-
white/10 text-white"}`}
    >
      <p className="text-[12px] uppercase tracking-[0.2em]">{label}</p>
      <p className="text-xl font-semibold">{value}</p>
    </div>
  );
}

```

```
 );
}
```

app/organizador/pagamentos/invoices/page.tsx

```
import { redirect } from "next/navigation";

export const metadata = {
  title: "Faturação | ORYA",
};

export default function InvoicesPage() {
  redirect("/organizador?tab=invoices");
}
```

app/organizador/pagamentos/page.tsx

```
"use client";

import { useEffect } from "react";
import { useRouter, useSearchParams } from "next/navigation";

// LEGACY – conteúdo vive em Finanças & Pagamentos (tab=finance)
export default function PaymentsLegacyRedirect() {
  const router = useRouter();
  const searchParams = useSearchParams();

  useEffect(() => {
    const params = new URLSearchParams(searchParams?.toString() || "");
    params.set("tab", "finance");
    router.replace(`/organizador?${params.toString()}`);
  }, [router, searchParams]);

  return (
    <div className="mx-auto max-w-3xl px-4 py-10 text-white">
      <p className="text-sm text-white/70">
        Pagamentos foi integrado na tab Finanças & Pagamentos. A redirecionar...
      </p>
    </div>
  );
}
```

app/organizador/promo/page.tsx

```
"use client";

import { useEffect } from "react";
import { useRouter, useSearchParams } from "next/navigation";

// LEGACY – promo codes vivem agora dentro da tab Marketing (section=promos)
export default function PromoCodesLegacyRedirect() {
  const router = useRouter();
  const searchParams = useSearchParams();

  useEffect(() => {
    const params = new URLSearchParams(searchParams?.toString() || "");
    params.set("tab", "marketing");
    params.set("section", "promos");
    router.replace(`/organizador?${params.toString()}`);
  }, [router, searchParams]);

  return (
    <div className="mx-auto max-w-3xl px-4 py-10 text-white">
      <p className="text-sm text-white/70">
        A área de códigos promocionais vive agora em Marketing. A redirecionar...
      </p>
    </div>
  );
}
```

app/organizador/promo/PromoCodesClient.tsx

```
"use client";

import { useEffect, useMemo, useState } from "react";
import useSWR from "swr";
import { useAuthModal } from "@app/components/autenticação/AuthModalContext";
import { useUser } from "@app/hooks/useUser";
import { trackEvent } from "@lib/analytics";
import { ConfirmDestructiveActionDialog } from "@app/components/ConfirmDestructiveActionDialog";

type PromoCodeDto = {
  id: number;
  name?: string | null;
  description?: string | null;
  code: string;
  type: "PERCENTAGE" | "FIXED";
  value: number;
  status?: "ACTIVE" | "INACTIVE" | "EXPIRED";
  maxUses: number | null;
  perUserLimit: number | null;
  validFrom: string | null;
  validUntil: string | null;
  active: boolean;
  eventId: number | null;
  minCartValueCents?: number | null;
  createdat: string;
  updatedAt: string;
  redemptionsCount?: number;
  autoApply: boolean;
  minQuantity?: number | null;
  minTotalCents?: number | null;
};

type ListResponse = {
  ok: boolean;
  promoCodes: PromoCodeDto[];
  events: { id: number; title: string; slug: string }[];
  promoStats?: {
    promoCodeId: number;
    tickets: number;
    grossCents: number;
    discountCents: number;
    platformFeeCents: number;
    netCents: number;
    usesTotal?: number;
    usersUnique?: number;
  }[];
  error?: string;
};

type PromoDetailResponse =
| {
  ok: true;
  promo: PromoCodeDto & {
    name?: string | null;
    description?: string | null;
    minCartValueCents?: number | null;
  };
  stats: {
    usesTotal: number;
    usersUnique: number;
    tickets: number;
    grossCents: number;
    discountCents: number;
    netCents: number;
    newUsers: number;
    returningUsers: number;
  };
  topEvents: { id: number; title: string; slug: string | null; uses: number }[];
  history: {
    id: number;
    usedAt: string;
    discountCents: number;
    items: number;
    userLabel: string;
  };
}
```

```

    event: { id: number; title: string; slug: string | null } | null;
}[];
}
| { ok: false; error?: string };

const fetcher = (url: string) => fetch(url).then((r) => r.json());

const SUGGESTED_CODES = [
  { code: "ORYA10", type: "PERCENTAGE" as const, value: "10", hint: "Friends & Family + 10%" },
  { code: "EARLY15", type: "PERCENTAGE" as const, value: "15", hint: "Early bird limitado" },
  { code: "TEAM5", type: "FIXED" as const, value: "5", hint: "5 € por pessoa para grupos" },
];
}

const EMPTY_FORM = {
  name: "",
  description: "",
  code: "",
  type: "PERCENTAGE" as "PERCENTAGE" | "FIXED",
  value: "10",
  maxUses: "",
  perUserLimit: "",
  validFrom: "",
  validUntil: "",
  eventId: "global",
  active: true,
  autoApply: false,
  minQuantity: "",
  minTotal: "",
  minCart: "",
};

function PromoStatusBadge({ status, active }: { status?: "ACTIVE" | "INACTIVE" | "EXPIRED"; active: boolean }) {
  const finalStatus = status || (active ? "ACTIVE" : "INACTIVE");
  const className =
    finalStatus === "EXPIRED"
      ? "border border-amber-400/40 bg-amber-500/15 text-amber-100"
      : finalStatus === "ACTIVE"
        ? "border border-emerald-400/40 bg-emerald-500/15 text-emerald-100"
        : "border border-white/15 bg-white/10 text-white/70";
  return (
    <span className={`inline-flex rounded-full px-3 py-1 text-[11px] ${className}`}>
      {finalStatus === "EXPIRED" ? "Expirada" : finalStatus === "ACTIVE" ? "Ativo" : "Inativo"}
    </span>
  );
}

export default function PromoCodesClient() {
  const { user } = useUser();
  const { openModal } = useAuthModal();
  const { data, mutate } = useSWR<ListResponse>(user ? "/api/organizador/promo" : null, fetcher);

  const [filters, setFilters] = useState({
    eventId: "all",
    status: "all" as "all" | "active" | "inactive" | "auto",
    q: "",
  });
  const [form, setForm] = useState(EMPTY_FORM);
  const [saving, setSaving] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [success, setSuccess] = useState<string | null>(null);
  const [editingId, setEditingId] = useState<number | null>(null);
  const [deleteId, setDeleteId] = useState<number | null>(null);
  const [detailId, setDetailId] = useState<number | null>(null);
  const [detail, setDetail] = useState<PromoDetailResponse | null>(null);
  const [detailLoading, setDetailLoading] = useState(false);
  const [detailError, setDetailError] = useState<string | null>(null);

  const loading = !data;
  useEffect(() => {
    if (!user && !data) {
      openModal({ mode: "login", redirectTo: "/organizador/promo", showGoogle: true });
    }
  }, [user, data, openModal]);

  const events = useMemo(() => data?.events ?? [], [data]);
  const promos = useMemo(() => data?.promoCodes ?? [], [data]);
  const promoStats = useMemo(() => data?.promoStats ?? [], [data]);
  const promoStatsMap = useMemo(() => new Map(promoStats.map((s) => [s.promoCodeId, s])), [promoStats]);
}

```

```

const filteredPromos = useMemo(() => {
  return promos.filter((p) => {
    const autoApply = !!p.autoApply;
    if (filters.eventId !== "all" && `${p.eventId} ?? "global"` !== filters.eventId) return false;
    if (filters.status === "active" && !p.active) return false;
    if (filters.status === "inactive" && p.active) return false;
    if (filters.status === "auto" && !autoApply) return false;
    if (filters.q.trim()) {
      const q = filters.q.toLowerCase();
      const haystack = `${p.code} ${p.name ?? ""} ${p.description ?? ""}`.toLowerCase();
      if (!haystack.includes(q)) return false;
    }
    return true;
  });
}, [promos, filters]);

const handleSubmit = async () => {
  setSaving(true);
  setError(null);
  setSuccess(null);
  try {
    const numericValue = Number(form.value);
    if (form.type === "PERCENTAGE") {
      if (!Number.isFinite(numericValue) || numericValue <= 0 || numericValue > 100) {
        setError("Percentagem inválida. Usa 1% a 100.");
        setSaving(false);
        return;
      }
    } else if (!Number.isFinite(numericValue) || numericValue < 0) {
      setError("Valor inválido. Usa um número igual ou superior a 0.");
      setSaving(false);
      return;
    }
    const payload = {
      name: form.name.trim() || undefined,
      description: form.description.trim() || undefined,
      code: form.code.trim(),
      type: form.type,
      value:
        form.type === "PERCENTAGE"
          ? Math.round(numericValue * 100) // guardamos em bps para manter compat
            : Math.round(numericValue * 100), // euros -> cents
      maxUses: form.maxUses ? Number(form.maxUses) : null,
      perUserLimit: form.perUserLimit ? Number(form.perUserLimit) : null,
      validFrom: form.validFrom || null,
      validUntil: form.validUntil || null,
      eventId: form.eventId === "global" ? null : Number(form.eventId),
      active: form.active,
      autoApply: form.autoApply,
      minQuantity: form.minQuantity ? Number(form.minQuantity) : null,
      minTotalCents: form.minTotal ? Math.round(Number(form.minTotal) * 100) : null,
      minCartValueCents: form.minCart ? Math.round(Number(form.minCart) * 100) : null,
    };
    const res = await fetch("/api/organizador/promo", {
      method: editingId ? "PATCH" : "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(editingId ? { id: editingId, ...payload } : payload),
    });
    const json = await res.json().catch(() => null);
    if (!res.ok || json?.ok === false) {
      setError(json?.error || "Não foi possível guardar o código.");
    } else {
      trackEvent("promo_code_created", {
        scope: payload.eventId ? "event" : "global",
        type: payload.type,
      });
      setSuccess(editingId ? "Código atualizado." : "Código criado com sucesso.");
      setForm({ ...EMPTY_FORM, code: "" });
      setEditingId(null);
      mutate();
    }
  } catch (err) {
    console.error(err);
    setError("Erro inesperado ao guardar código.");
  } finally {
    setSaving(false);
  }
}

```

```

};

const handleEdit = (promo: PromoCodeDto) => {
  setEditingId(promo.id);
  setForm({
    name: promo.name ?? "",
    description: promo.description ?? "",
    code: promo.code,
    type: promo.type,
    value: promo.type === "PERCENTAGE" ? String(promo.value / 100) : String(promo.value / 100),
    maxUses: promo.maxUses != null ? String(promo.maxUses) : "",
    perUserLimit: promo.perUserLimit != null ? String(promo.perUserLimit) : "",
    validFrom: promo.validFrom ?? "",
    validUntil: promo.validUntil ?? "",
    eventId: promo.eventId == null ? "global" : String(promo.eventId),
    active: promo.active,
    autoApply: !promo.autoApply,
    minQuantity: promo.minQuantity != null ? String(promo.minQuantity) : "",
    minTotal: promo.minTotalCents != null ? String(promo.minTotalCents / 100) : "",
    minCart: promo.minCartValueCents != null ? String(promo.minCartValueCents / 100) : "",
  });
  setSuccess(null);
  setError(null);
};

const handleDelete = async (id: number) => {
  try {
    const res = await fetch("/api/organizador/promo", {
      method: "DELETE",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ id }),
    });
    if (!res.ok) {
      const j = await res.json().catch(() => null);
      setError(j?.error || "Erro ao apagar o código.");
    } else {
      trackEvent("promo_code_deleted", { promoId: id });
      if (editingId === id) setEditingId(null);
      mutate();
    }
  } catch (err) {
    console.error(err);
    setError("Erro inesperado ao apagar código.");
  } finally {
    setDeleteId(null);
  }
};

const handleToggle = async (id: number, active: boolean) => {
  try {
    await fetch("/api/organizador/promo", {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ id, active }),
    });
    trackEvent(active ? "promo_code_activated" : "promo_code_deactivated", { promoId: id });
    mutate();
  } catch (err) {
    console.error(err);
  }
};

const handleOpenDetail = async (promoid: number) => {
  setDetailId(promoid);
  setDetail(null);
  setDetailError(null);
  setDetailLoading(true);
  try {
    const res = await fetch(`^/api/organizador/promo/${promoid}`);
    const json: PromoDetailResponse = await res.json();
    if (!res.ok || json.ok === false) {
      setDetailError(json && "error" in json ? json.error || "Erro ao carregar detalhe." : "Erro ao carregar detalhe.");
    } else {
      setDetail(json);
    }
  } catch (err) {
    console.error(err);
  }
};

```

```

        setDetailError("Erro ao carregar detalhe.");
    } finally {
        setDetailLoading(false);
    }
};

const emptyState = filteredPromos.length === 0 && !loading;

const stats = useMemo(() => {
    const total = promos.length;
    const active = promos.filter((p) => p.active).length;
    const redemptions = promos.reduce((acc, p) => acc + (p.redemptionsCount ?? 0), 0);
    const totalDiscount = promoStats.reduce((acc, s) => acc + (s.discountCents ?? 0), 0);
    const totalGross = promoStats.reduce((acc, s) => acc + (s.grossCents ?? 0), 0);
    const usersUnique = promoStats.reduce((acc, s) => acc + (s.usersUnique ?? 0), 0);
    return { total, active, redemptions, totalDiscount, totalGross, usersUnique };
}, [promos, promoStats]);

const formatEuro = (cents: number) => `${(cents / 100).toFixed(2)} €`;
const formatDate = (iso: string) => new Date(iso).toLocaleString("pt-PT");
const statsFor = (promoId: number) =>
    promoStatsMap.get(promoId) ?? { tickets: 0, grossCents: 0, discountCents: 0, netCents: 0, usesTotal: 0, usersUnique: 0 };

const previewPrice = 20; // € exemplo
const preview = useMemo(() => {
    const valueNum = Number(form.value) || 0;
    const baseCents = Math.max(0, Math.round(previewPrice * 100));
    const discountCents =
        form.type === "PERCENTAGE"
            ? Math.min(baseCents, Math.round((baseCents * Math.min(100, Math.max(0, valueNum))) / 100))
            : Math.min(baseCents, Math.round(valueNum * 100));
    const totalCents = Math.max(0, baseCents - discountCents);
    return {
        base: baseCents / 100,
        discount: discountCents / 100,
        total: totalCents / 100,
    };
}, [form.type, form.value]);

const applySuggestion = (code: string, type: "PERCENTAGE" | "FIXED", value: string) => {
    setForm((prev) => ({ ...prev, code, type, value }));
};

return (
<section className="mx-auto max-w-6xl px-4 py-10 space-y-6 text-white md:px-6 lg:px-8">
    <header className="space-y-1">
        <h1 className="text-2xl font-semibold">Códigos promocionais</h1>
        <p className="text-sm text-white/70">
            Cria códigos de desconto por evento ou globais. Podes ativar/desativar ou definir auto-aplicação a qualquer momento.
        </p>
    </header>

    <div className="grid grid-cols-1 gap-3 rounded-xl border border-white/10 bg-white/5 p-3 sm:grid-cols-4">
        {loading}
        ? [...Array(3)].map((_, idx) =>
            <div key={idx} className="animate-pulse space-y-2">
                <div className="h-3 w-24 rounded bg-white/15" />
                <div className="h-6 w-20 rounded bg-white/20" />
            </div>
        )))
        : (
            <>
                <div>
                    <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Total</p>
                    <p className="text-2xl font-semibold">{stats.total}</p>
                </div>
                <div>
                    <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Ativos</p>
                    <p className="text-2xl font-semibold">{stats.active}</p>
                </div>
                <div>
                    <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Utilizações</p>
                    <p className="text-2xl font-semibold">{stats.redemptions}</p>
                    <p className="text-[11px] text-white/50">Conta bilhetes (sale_lines)</p>
                </div>
                <div>
                    <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Desconto total</p>
                    <p className="text-2xl font-semibold">{formatEuro(stats.totalDiscount)}</p>
                </div>
            </>
        )
    </div>

```



```

        <div className="h-3 w-24 rounded bg-white/15" />
        <div className="h-3 w-32 rounded bg-white/10" />
    </div>
    <div className="h-7 w-20 rounded-full bg-white/10" />
</div>
    )}}
</div>
) : emptyState ? (
<div className="flex flex-col gap-3 p-5 text-white">
<div>
    <p className="text-lg font-semibold">Ainda não tens códigos promocionais.</p>
    <p className="text-sm text-white/70">Cria o teu primeiro código para recompensar equipas, amigos ou early birds.
</p>
</div>
<div className="flex flex-wrap gap-2">
    <button
        type="button"
        onClick={() => {
            const el = document?.getElementById("promo-form");
            el?.scrollIntoView({ behavior: "smooth", block: "start" });
        }}
        className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow hover:scale-[1.01]"
    >
        Criar primeiro código
    </button>
<div className="flex flex-wrap gap-2">
    {SUGGESTED_CODES.map((item) => (
        <button
            key={item.code}
            type="button"
            onClick={() => applySuggestion(item.code, item.type, item.value)}
            className="rounded-full border border-white/15 bg-black/40 px-3 py-1.5 text-[12px] text-white/80
hover:border-white/30"
        >
            {item.code} · {item.hint}
        </button>
    )))
    </div>
</div>
</div>
) : (
<table className="min-w-full text-left text-sm text-white/80">
    <thead className="bg-white/5 text-[12px] uppercase tracking-[0.12em] text-white/60">
        <tr>
<th className="px-3 py-2">Código</th>
<th className="px-3 py-2">Nome</th>
<th className="px-3 py-2">Evento</th>
<th className="px-3 py-2">Valor</th>
        <th className="px-3 py-2">Usos</th>
        <th className="px-3 py-2">Utilizadores</th>
        <th className="px-3 py-2">Bilhetes</th>
        <th className="px-3 py-2">Bruto</th>
        <th className="px-3 py-2">Desconto</th>
        <th className="px-3 py-2">Líquido</th>
        <th className="px-3 py-2">Estado</th>
        <th className="px-3 py-2 text-right">Ações</th>
    </tr>
</thead>
<tbody>
    {filteredPromos.map((promo) => {
        const s = statsFor(promo.id);
        return (
            <tr key={promo.id} className="border-t border-white/10">
                <td className="px-3 py-2 font-semibold text-white">
                    <div className="flex flex-col">
                        <span>{promo.code}</span>
                        {promo.autoApply && (
                            <span className="mt-1 inline-flex w-fit rounded-full border border-emerald-300/40 bg-emerald-500/10
px-2 py-0.5 text-[10px] uppercase tracking-wide text-emerald-100">
                                Auto
                            </span>
                        )}
                    </div>
                </td>
                <td className="px-3 py-2 text-white/80">{promo.name ?? "-"}</td>
                <td className="px-3 py-2">
                    {promo.eventId == null
                    ? "Global"

```

```

        : events.find((ev) => ev.id === promo.eventId)?.title || "Evento removido"
    </td>
    <td className="px-3 py-2">
        {promo.type === "PERCENTAGE"
            ? `${(promo.value / 100).toFixed(2).replace(/\.\.\.00$/, "")}%` 
            : `${(promo.value / 100).toFixed(2).replace(/\.\.\.00$/, "")} €`}
    </td>
    <td className="px-3 py-2">{s.usesTotal ?? promo.redemptionsCount ?? 0}</td>
    <td className="px-3 py-2">{s.usersUnique ?? 0}</td>
    <td className="px-3 py-2">
        {s.tickets}
    </td>
    <td className="px-3 py-2">
        {formatEuro(s.grossCents ?? 0)}
    </td>
    <td className="px-3 py-2 text-emerald-200">
        -{formatEuro(s.discountCents ?? 0)}
    </td>
    <td className="px-3 py-2">
        {formatEuro(s.netCents ?? 0)}
    </td>
    <td className="px-3 py-2">
        <PromoStatusBadge status={promo.status} active={promo.active} />
    </td>
    <td className="px-3 py-2 text-right">
        <div className="flex flex-wrap items-center justify-end gap-2">
            <button
                type="button"
                onClick={() => handleOpenDetail(promo.id)}
                className="rounded-full border border-white/20 px-2 py-1 text-[11px] hover:bg-white/10"
            >
                Ver detalhe
            </button>
            <button
                type="button"
                onClick={() => handleEdit(promo)}
                className="rounded-full border border-white/20 px-2 py-1 text-[11px] hover:bg-white/10"
            >
                Editar
            </button>
            <button
                type="button"
                onClick={() => handleToggle(promo.id, !promo.active)}
                className="rounded-full border border-white/20 px-2 py-1 text-[11px] hover:bg-white/10"
            >
                {promo.active ? "Desativar" : "Ativar"}
            </button>
            <button
                type="button"
                onClick={() => setDeleteId(promo.id)}
                className="text-[11px] text-red-300 hover:text-red-200"
            >
                Apagar
            </button>
        </div>
    </td>
    </tr>
);
});
</tbody>
</table>
)
</div>

<ConfirmDestructiveActionDialog
open={deleteId !== null}
title="Apagar código promocional?"
description="Esta ação desativa o código e impede novas utilizações. Mantemos o histórico existente."
consequences={[
    "O código deixa de ser aplicável no checkout.",
    "Mantemos histórico de utilizações anteriores.",
]}
confirmLabel="Apagar código"
cancelLabel="Cancelar"
dangerLevel="high"
onClose={() => setDeleteId(null)}
onConfirm={() => {
    if (deleteId !== null) handleDelete(deleteId);
}}
```

```

        }}
    />

<div id="promo-form" className="rounded-xl border border-white/10 bg-white/5 p-4 space-y-3">
    <h2 className="text-lg font-semibold">Criar / editar código</h2>
    <div className="grid grid-cols-1 gap-3 sm:grid-cols-2">
        <label className="space-y-1 text-sm">
            Nome interno
            <input
                value={form.name}
                onChange={(e) => setForm((p) => ({ ...p, name: e.target.value }))}
                className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none focus:border-white"
                placeholder="Ex.: Early bird maio"
            />
        </label>
        <label className="space-y-1 text-sm">
            Descrição (opcional)
            <input
                value={form.description}
                onChange={(e) => setForm((p) => ({ ...p, description: e.target.value }))}
                className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none focus:border-white"
                placeholder="Notas internas ou condições"
            />
        </label>
        <label className="space-y-1 text-sm">
            Código
            <input
                value={form.code}
                onChange={(e) => setForm((p) => ({ ...p, code: e.target.value.toUpperCase() }))}
                className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none focus:border-white"
                placeholder="ORYA10"
            />
        </label>
        <label className="space-y-1 text-sm">
            Tipo
            <select
                value={form.type}
                onChange={(e) => setForm((p) => ({ ...p, type: e.target.value as "PERCENTAGE" | "FIXED" }))}
                className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none"
            >
                <option value="PERCENTAGE">% desconto</option>
                <option value="FIXED">€ desconto</option>
            </select>
        </label>
        <label className="space-y-1 text-sm">
            Valor
            <div className="relative">
                <input
                    type="number"
                    min={0}
                    step=".01"
                    value={form.value}
                    onChange={(e) => setForm((p) => ({ ...p, value: e.target.value }))}
                    className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 pr-12 text-sm outline-none"
                    placeholder={form.type === "PERCENTAGE" ? "Ex.: 10 = 10%" : "Ex.: 5 = 5 €"}
                />
                <span className="absolute inset-y-0 right-3 flex items-center text-[12px] text-white/60">
                    {form.type === "PERCENTAGE" ? "%" : "€"}
                </span>
            </div>
            <p className="text-[11px] text-white/60">
                {form.type === "PERCENTAGE" ? "Entre 1% e 100%" : "Valor fixo em euros."}
            </p>
        </label>
        <label className="space-y-1 text-sm">
            Nº máximo de utilizações
            <input
                type="number"
                min={0}
                value={form.maxUses}
                onChange={(e) => setForm((p) => ({ ...p, maxUses: e.target.value }))}
                className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none"
                placeholder="Ex.: 200 (opcional)"
            />
        </label>
        <label className="space-y-1 text-sm">
            Limite por pessoa
            <input

```

```

        type="number"
        min={0}
        value={form.perUserLimit}
        onChange={(e) => setForm((p) => ({ ...p, perUserLimit: e.target.value }))}
        className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none"
        placeholder="Ex.: 2 (opcional)"
      />
    </label>
<label className="space-y-1 text-sm">
  Evento
  <select
    value={form.eventId}
    onChange={(e) => setForm((p) => ({ ...p, eventId: e.target.value }))}
    className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none"
  >
    <option value="global">Global</option>
    {events.map((ev) => (
      <option key={ev.id} value={ev.id}>
        {ev.title}
      </option>
    )))
  </select>
</label>
<label className="space-y-1 text-sm">
  Válido de
  <input
    type="datetime-local"
    value={form.validFrom}
    onChange={(e) => setForm((p) => ({ ...p, validFrom: e.target.value }))}
    className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none"
  />
</label>
<label className="space-y-1 text-sm">
  Válido até
  <input
    type="datetime-local"
    value={form.validUntil}
    onChange={(e) => setForm((p) => ({ ...p, validUntil: e.target.value }))}
    className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none"
  />
</label>
<label className="space-y-1 text-sm">
  Min. quantidade por compra
  <input
    type="number"
    min={0}
    value={form.minQuantity}
    onChange={(e) => setForm((p) => ({ ...p, minQuantity: e.target.value }))}
    className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none"
    placeholder="Ex.: 2"
  />
</label>
<label className="space-y-1 text-sm">
  Mín. valor total (€)
  <input
    type="number"
    min={0}
    value={form.minTotal}
    onChange={(e) => setForm((p) => ({ ...p, minTotal: e.target.value }))}
    className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none"
    placeholder="Ex.: 20"
  />
</label>
<label className="space-y-1 text-sm">
  Mín. valor carrinho (€)
  <input
    type="number"
    min={0}
    value={form.minCart}
    onChange={(e) => setForm((p) => ({ ...p, minCart: e.target.value }))}
    className="w-full rounded-lg border border-white/15 bg-black/20 px-3 py-2 text-sm outline-none"
    placeholder="Ex.: 30"
  />
<p className="text-[11px] text-white/60">Opcional; preferível ao min total legacy.</p>
</label>
</div>

<div className="flex flex-wrap gap-3 text-[12px] text-white/75">

```

```

<label className="flex items-center gap-2">
  <input
    type="checkbox"
    checked={form.active}
    onChange={(e) => setForm((p) => ({ ...p, active: e.target.checked }))}
    className="h-4 w-4"
  />
  Ativo
</label>
<label className="flex items-center gap-2">
  <input
    type="checkbox"
    checked={form.autoApply}
    onChange={(e) => setForm((p) => ({ ...p, autoApply: e.target.checked }))}
    className="h-4 w-4"
  />
  Auto-aplicar em checkout (se válido)
</label>
</div>

{(!error || success) &&
<div className="text-[12px]">
  {error && <p className="text-red-300">{error}</p>}
  {success && <p className="text-emerald-200">{success}</p>}
</div>
)}

<div className="flex flex-wrap gap-2">
  <button
    type="button"
    onClick={handleSubmit}
    disabled={saving}
    className="rounded-full bg-white px-4 py-2 text-sm font-semibold text-black shadow hover:scale-[1.01]"
    disabled:opacity="60"
  >
    {saving ? "A guardar..." : editingId ? "Guardar alterações" : "Criar código"}
  </button>
  <button
    type="button"
    onClick={() => {
      setForm(EMPTY_FORM);
      setEditingId(null);
      setError(null);
      setSuccess(null);
    }}
    className="rounded-full border border-white/20 px-3 py-2 text-[12px] text-white hover:bg-white/10"
  >
    Limpar
  </button>
</div>

<div className="rounded-xl border border-white/10 bg-black/30 p-3 text-[12px] text-white/80">
  <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Preview rápido</p>
  <p className="text-sm">
    Exemplo com bilhete de {preview.base.toFixed(2)} € → Cliente paga{" "}
    <span className="font-semibold text-white">{preview.total.toFixed(2)} €</span>{" "}
    (desconto {preview.discount.toFixed(2)} €)
  </p>
</div>

{detailId !== null &&
<div className="fixed inset-0 z-40 flex items-end justify-center bg-black/60 px-4 py-6 backdrop-blur-sm sm:items-center">
  <div className="w-full max-w-3xl rounded-2xl border border-white/15 bg-neutral-900 p-5 shadow-2xl">
    <div className="flex items-start justify-between gap-4">
      <div>
        <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Promo</p>
        <h3 className="text-xl font-semibold text-white">
          {detail?.ok ? detail.promo.code : "Detalhe da promoção"}
        </h3>
        {detail?.ok && detail.promo.name && (
          <p className="text-sm text-white/70">{detail.promo.name}</p>
        )}
      </div>
      <button
        type="button"
        onClick={() => {
          setDetailId(null);
        }}
      >
        Fechar
      </button>
    </div>
  </div>
</div>

```

```

        setDetail(null);
        setDetailError(null);
    )}
    className="rounded-full border border-white/20 px-3 py-1 text-[12px] text-white hover:bg-white/10"
>
    Fechar
</button>
</div>

{detailLoading && <p className="mt-4 text-sm text-white/70">A carregar detalhe...</p>}
{detailError && (
    <p className="mt-4 text-sm text-red-300">
        {detailError || "Não foi possível carregar o detalhe."}
    </p>
)}
{detail?.ok && (
    <div className="mt-4 space-y-5">
        <div className="grid grid-cols-2 gap-3 md:grid-cols-3">
            <div className="rounded-lg border border-white/10 bg-white/5 p-3">
                <p className="text-[11px] text-white/60">Utilizações</p>
                <p className="text-lg font-semibold text-white">{detail.stats.usesTotal}</p>
            </div>
            <div className="rounded-lg border border-white/10 bg-white/5 p-3">
                <p className="text-[11px] text-white/60">Utilizadores únicos</p>
                <p className="text-lg font-semibold text-white">{detail.stats.usersUnique}</p>
            </div>
            <div className="rounded-lg border border-white/10 bg-white/5 p-3">
                <p className="text-[11px] text-white/60">Bilhetes</p>
                <p className="text-lg font-semibold text-white">{detail.stats.tickets}</p>
            </div>
            <div className="rounded-lg border border-white/10 bg-white/5 p-3">
                <p className="text-[11px] text-white/60">Bruto</p>
                <p className="text-lg font-semibold text-white">{formatEuro(detail.stats.grossCents)}</p>
            </div>
            <div className="rounded-lg border border-white/10 bg-white/5 p-3">
                <p className="text-[11px] text-white/60">Desconto</p>
                <p className="text-lg font-semibold text-emerald-200">-{formatEuro(detail.stats.discountCents)}</p>
            </div>
            <div className="rounded-lg border border-white/10 bg-white/5 p-3">
                <p className="text-[11px] text-white/60">Líquido</p>
                <p className="text-lg font-semibold text-white">{formatEuro(detail.stats.netCents)}</p>
            </div>
            <div className="rounded-lg border border-white/10 bg-white/5 p-3">
                <p className="text-[11px] text-white/60">Novos</p>
                <p className="text-lg font-semibold text-white">{detail.stats.newUsers}</p>
            </div>
            <div className="rounded-lg border border-white/10 bg-white/5 p-3">
                <p className="text-[11px] text-white/60">Recorrentes</p>
                <p className="text-lg font-semibold text-white">{detail.stats.returningUsers}</p>
            </div>
        </div>
        <div className="grid grid-cols-1 gap-4 md:grid-cols-2">
            <div className="rounded-lg border border-white/10 bg-black/30 p-3">
                <div className="flex items-center justify-between">
                    <p className="text-sm font-semibold text-white">Top eventos</p>
                    {detail.topEvents.length === 0 && (
                        <span className="text-[12px] text-white/50">Sem utilizações</span>
                    )}
                </div>
                <div className="mt-2 space-y-2">
                    {detail.topEvents.map((ev) => (
                        <div key={ev.id} className="flex items-center justify-between rounded-lg border border-white/10 bg-white/5 px-3 py-2">
                            <div>
                                <p className="text-sm text-white">{ev.title}</p>
                                {ev.slug && <p className="text-[11px] text-white/50">{ev.slug}</p>}
                            </div>
                            <span className="rounded-full border border-white/15 px-2 py-1 text-[11px] text-white/80">
                                {ev.uses} uso{ev.uses === 1 ? "" : "s"}
                            </span>
                        </div>
                    )))
                </div>
            </div>
        </div>
        <div className="rounded-lg border border-white/10 bg-black/30 p-3">

```

```
<div className="flex items-center justify-between">
    <p className="text-sm font-semibold text-white">Histórico recente</p>
    <span className="text-[11px] text-white/60">Últimos {detail.history.length} registos</span>
</div>
<div className="mt-2 space-y-2 max-h-64 overflow-auto pr-1">
    {detail.history.map((h) => (
        <div key={h.id} className="rounded-lg border border-white/10 bg-white/5 p-2 text-[12px] text-white/80">
            <div className="flex items-center justify-between">
                <span className="font-semibold text-white">{formatEuro(h.discountCents)}</span>
                <span className="text-white/60">{formatDateTime(h.usedAt)}</span>
            </div>
            <p className="text-white/70">Utilizador: {h.userLabel}</p>
            <p className="text-white/60">
                Itens: {h.items} · Evento: {h.event?.title ?? "-"}
            </p>
        </div>
    )));
    {detail.history.length === 0 && (
        <p className="text-[12px] text-white/60">Sem histórico registo.</p>
    )}
</div>
</div>
</div>
</div>
);
</div>
</section>
);
```

app/organizador/RoleBadge.tsx

```
use client";  
  
import { OrganizerMemberRole } from "@prisma/client";  
  
type Props = {  
  role: OrganizerMemberRole | "OWNER" | "CO_OWNER" | "ADMIN" | "STAFF" | "VIEWER";  
  subtle?: boolean;  
};  
  
const ROLE_STYLES: Record<Required<Props>["role"], string> = {  
  OWNER: "border-amber-300/60 bg-amber-400/15 text-amber-50",  
  CO_OWNER: "border-emerald-300/50 bg-emerald-400/10 text-emerald-50",  
  ADMIN: "border-sky-300/50 bg-sky-400/10 text-sky-50",  
  STAFF: "border-white/20 bg-white/10 text-white/80",  
  VIEWER: "border-white/15 bg-white/5 text-white/60",  
};  
  
const ROLE_LABEL: Record<Required<Props>["role"], string> = {  
  OWNER: "Owner",  
  CO_OWNER: "Co-owner",  
  ADMIN: "Admin",  
  STAFF: "Staff",  
  VIEWER: "Viewer",  
};  
  
export function RoleBadge({ role, subtle }: Props) {  
  const tone = ROLE_STYLES[role] ?? ROLE_STYLES.VIEWER;  
  const padding = subtle ? "px-2 py-[2px]" : "px-3 py-[6px]";  
  return (  
    <span  
      className={`inline-flex items-center rounded-full border ${padding} text-[11px] uppercase tracking-[0.16em] ${tone}`}  
    >  
      {ROLE_LABEL[role] ?? role}  
    </span>  
  );  
}
```

app/organizador/scan/page.tsx

```

import { Suspense } from "react";
import ScanClient from "./scan-client";

export const metadata = {
  title: "Scan | ORYA",
};

export default function ScanPage() {
  return (
    <Suspense fallback={<div className="p-6 text-white/70">A preparar scanner...</div>}>
      <ScanClient />
    </Suspense>
  );
}

```

app/organizador/scan/scan-client.tsx

```

"use client";

import { useEffect, useMemo, useState } from "react";
import useSWR from "swr";
import { useSearchParams, useRouter } from "next/navigation";
import Link from "next/link";
import { useUser } from "@/app/hooks/useUser";

const fetcher = (url: string) => fetch(url).then((r) => r.json());

type MeResponse = {
  ok: boolean;
  organizer?: { id: number; displayName?: string | null } | null;
  membershipRole?: string | null;
};

type ScanResultStatus = "OK" | "ALREADY_USED" | "CANCELLED" | "REFUNDED" | "INVALID" | "WRONG_EVENT";

type ScanResponse = {
  status: ScanResultStatus;
  message: string;
  ticket: {
    id: string;
    holderName: string | null;
    ticketTypeName: string | null;
    checkins: number;
    maxCheckins: number;
  } | null;
  checkedInAt: string | null;
  firstCheckinAt: string | null;
};

const statusTone: Record<ScanResultStatus, { bg: string; text: string }> = {
  OK: { bg: "bg-emerald-500", text: "text-emerald-50" },
  ALREADY_USED: { bg: "bg-amber-400", text: "text-amber-950" },
  CANCELLED: { bg: "bg-rose-500", text: "text-rose-50" },
  REFUNDED: { bg: "bg-rose-500", text: "text-rose-50" },
  INVALID: { bg: "bg-rose-600", text: "text-rose-50" },
  WRONG_EVENT: { bg: "bg-rose-600", text: "text-rose-50" },
};

function StatusBanner({ result }: { result: ScanResponse | null }) {
  if (!result) return null;
  const tone = statusTone[result.status];
  return (
    <div className={`rounded-2xl px-4 py-3 ${tone.bg} ${tone.text} shadow-lg`}>
      <div className="flex flex-wrap items-center justify-between gap-2">
        <div>
          <p className="text-sm font-semibold">{result.message}</p>
          {result.ticket && (
            <p className="text-xs opacity-80">
              {result.ticket.holderName || "Portador"} · {result.ticket.ticketTypeName || "Bilhete"}
            </p>
          )}
        </div>
        <div className="text-[11px] uppercase tracking-[0.18em]">{result.status}</div>
      </div>
      {result.checkedInAt && (

```

```

        <p className="mt-1 text-[12px] opacity-80">Check-in: {new Date(result.checkedInAt).toLocaleTimeString()}</p>
    )
    {result.firstCheckinAt && result.status === "ALREADY_USED" &&
     <p className="text-[12px] opacity-80">Primeiro uso: {new Date(result.firstCheckinAt).toLocaleString()}</p>
    )
  </div>
);
}

export default function ScanClient() {
  const { user, isLoading: userLoading } = useUser();
  const searchParams = useSearchParams();
  const router = useRouter();
  const eventIdParam = searchParams?.get("eventId");
  const eventId = eventIdParam ? Number(eventIdParam) : null;
  const { data: me } = useSWR<MeResponse>(user ? "/api/organizador/me" : null, fetcher, { revalidateOnFocus: false });

  const [ticketCode, setTicketCode] = useState("");
  const [deviceId, setDeviceId] = useState<string | null>(null);
  const [loading, setLoading] = useState(false);
  const [result, setResult] = useState<ScanResponse | null>(null);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    const existing = window.localStorage.getItem("orya_scan_device");
    if (existing) setDeviceId(existing);
    else {
      const random = crypto.randomUUID();
      setDeviceId(random);
      window.localStorage.setItem("orya_scan_device", random);
    }
  }, []);

  const resolvedEventId = useMemo(() => {
    if (!eventId || Number.isNaN(eventId)) return null;
    return eventId;
  }, [eventId]);

  const canScan = Boolean(me?.membershipRole === "OWNER" || me?.membershipRole === "CO_OWNER" || me?.membershipRole === "ADMIN" || me?.membershipRole === "STAFF");

  const handleScan = async () => {
    if (!resolvedEventId) {
      setError("Seleciona um evento válido.");
      return;
    }
    if (!ticketCode.trim()) {
      setError("Introduz um código/QR.");
      return;
    }
    setLoading(true);
    setError(null);
    try {
      const res = await fetch("/api/tickets/scan", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ eventId: resolvedEventId, ticketCode: ticketCode.trim(), deviceId })
      });
      const json = await res.json();
      if (!res.ok) {
        setError(json?.error || "Sem permissão para fazer check-in.");
        setResult(null);
        return;
      }
      setResult(json as ScanResponse);
      setTicketCode("");
    } catch (err) {
      console.error("[scan]", err);
      setError("Erro inesperado ao validar bilhete.");
    } finally {
      setLoading(false);
    }
  };

  const handleKey = (e: React.KeyboardEvent<HTMLInputElement>) => {
    if (e.key === "Enter") {
      handleScan();
    }
  }
}

```

```

};

if (userLoading) {
  return <div className="p-6 text-white/70">A carregar...</div>;
}

if (!user) {
  return (
    <div className="p-6 text-white/80 space-y-3">
      <p className="text-lg font-semibold">Precisas de iniciar sessão para fazer check-in.</p>
      <Link href="/login" className="rounded-full bg-white px-3 py-2 text-sm font-semibold text-black shadow">
        Entrar
      </Link>
    </div>
  );
}

if (!canScan) {
  return (
    <div className="p-6 space-y-3 text-white/80">
      <p className="text-lg font-semibold">Sem permissão para scan.</p>
      <p className="text-sm text-white/60">Pede ao Owner/Co-owner/Admin para te dar acesso ou adicionar-te como staff deste evento.</p>
    </div>
  );
}

return (
  <div className="min-h-screen bg-gradient-to-b from-[#0B1220] via-[#0A0F1A] to-black text-white px-4 py-6 space-y-5">
    <div className="flex items-center justify-between">
      <div>
        <p className="text-[11px] uppercase tracking-[0.3em] text-white/50">Check-in / QR</p>
        <h1 className="text-3xl font-bold">Scanner rápido</h1>
        <p className="text-sm text-white/70">Optimizado para telemóvel. Usa a câmara nativa ou introduz o código.</p>
      </div>
      <Link href="/organizador" className="rounded-full border border-white/20 bg-white/5 px-3 py-1 text-[12px] text-white hover:bg-white/10">
        Dashboard
      </Link>
    </div>

    <div className="rounded-3xl border border-white/10 bg-white/5 p-4 space-y-4 shadow-[0_18px_50px_rgba(0,0,0,0.6)]">
      <div className="flex flex-col gap-3">
        <label className="text-[12px] text-white/70">Código do bilhete / QR</label>
        <input
          value={ticketCode}
          onChange={(e) => setTicketCode(e.target.value)}
          onKeyDown={handleKey}
          className="w-full rounded-2xl border border-white/15 bg-black/30 px-3 py-3 text-lg outline-none focus:border-[#6BFFFF]"
          placeholder="Lê o QR ou cola o código"
          autoFocus
        />
        <button
          type="button"
          onClick={handleScan}
          disabled={loading}
          className="rounded-2xl bg-white px-4 py-3 text-sm font-semibold text-black shadow disabled:opacity-60"
        >
          {loading ? "A validar..." : "Validar entrada"}
        </button>
        {error && <p className="text-[12px] text-rose-200">{error}</p>}
      </div>
      <StatusBanner result={result} />
    </div>

    <div className="rounded-3xl border border-white/10 bg-white/5 p-4 space-y-3 text-sm text-white/70 shadow-[0_18px_50px_rgba(0,0,0,0.6)]">
      <p className="text-[12px] uppercase tracking-[0.2em] text-white/50">Modo mobile</p>
      <p>Para usar a câmara nativa, abre o leitor de QR do telemóvel e escolhe "Abrir app". Este ecrã fica pronto para colar o código.</p>
      <p className="text-[12px] text-white/60">Em breve: leitura direta da câmara.</p>
    </div>
  </div>
);
}

```

app/organizador/settings/verify/page.tsx

```
"use client";

import { useEffect, useState } from "react";
import { useRouter, useSearchParams } from "next/navigation";

type State = "idle" | "loading" | "ok" | "error";

export default function VerifyOfficialEmailPage() {
  const search = useSearchParams();
  const router = useRouter();
  const token = search?.get("token");
  const [state, setState] = useState<State>("idle");
  const [message, setMessage] = useState<string | null>(null);

  useEffect(() => {
    if (!token) {
      setState("error");
      setMessage("Token em falta. Usa o link mais recente do email.");
      return;
    }
    const confirm = async () => {
      try {
        setState("loading");
        const res = await fetch("/api/organizador/organizations/settings/official-email/confirm", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ token }),
        });
        const json = await res.json().catch(() => null);
        if (!res.ok || json?.ok === false) {
          setState("error");
          setMessage(json?.error || "Não foi possível confirmar o email.");
          return;
        }
        setState("ok");
        setMessage("Email oficial confirmado.");
        setTimeout(() => router.push("/organizador?tab=settings"), 1200);
      } catch (err) {
        setState("error");
        setMessage("Erro inesperado ao confirmar o email.");
      }
    };
    void confirm();
  }, [token, router]);

  return (
    <main className="flex min-h-screen items-center justify-center bg-gradient-to-br from-[#050712] via-[#090f2b] to-[#0b132d] px-4 text-white">
      <div className="w-full max-w-md space-y-4 rounded-3xl border border-white/10 bg-black/50 p-6 text-center shadow-[0_22px_70px_rgba(0,0,0,0.55)]">
        <h1 className="text-2xl font-semibold">Verificar email oficial</h1>
        {state === "loading" && <p className="text-white/70">A confirmar token...</p>}
        {state === "ok" && <p className="text-emerald-300">Email confirmado. A redirecionar...</p>}
        {state === "error" && <p className="text-amber-300">{message || "Token inválido ou expirado."}</p>}
        <div className="flex justify-center">
          <button
            onClick={() => router.push("/organizador?tab=settings")}
            className="rounded-full border border-white/20 bg-white/10 px-2 py-2 text-sm font-semibold text-white hover:border-white/35"
          >
            Voltar ao dashboard
          </button>
        </div>
      </div>
    </main>
  );
}
}
```

app/padel/duplas/page.tsx

```
"use client";
```

```

import Link from "next/link";
import { useEffect, useState } from "react";

type Slot = {
  id: number;
  slotStatus: string;
  paymentStatus: string;
  slotRole: string;
  ticket?: { id: string; status: string | null } | null;
};

type Pairing = {
  id: number;
  paymentMode: string;
  pairingStatus: string;
  inviteToken: string | null;
  lockedUntil: string | null;
  slots: Slot[];
  event: { id: number; title: string; slug: string; templateType: string | null };
  category?: { label: string } | null;
};

export default function MinhasDuplasPage() {
  const [pairings, setPairings] = useState<Pairing[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [actionLoading, setActionLoading] = useState<string | null>(null);

  const refresh = async () => {
    setLoading(true);
    setError(null);
    try {
      const res = await fetch("/api/padel/pairings/my");
      const json = await res.json();
      if (!res.ok || !json?.ok) throw new Error(json?.error || "Erro ao carregar duplas");
      setPairings(json.pairings ?? []);
    } catch (err) {
      setError(err instanceof Error ? err.message : "Erro ao carregar duplas");
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    refresh();
  }, []);

  const doAction = async (pairingId: number, action: "cancel" | "assume") => {
    setActionLoading(`#${action}-${pairingId}`);
    try {
      const res = await fetch(`/api/padel/pairings/${pairingId}/${action === "cancel" ? "cancel" : "assume"}`, {
        method: "POST",
      });
      const json = await res.json();
      if (!res.ok || !json?.ok) throw new Error(json?.error || "Ação falhou");
      await refresh();
    } catch (err) {
      alert(err instanceof Error ? err.message : "Erro na ação");
    } finally {
      setActionLoading(null);
    }
  };

  const formatDateTime = (iso: string | null) => {
    if (!iso) return "";
    const d = new Date(iso);
    return d.toLocaleString("pt-PT", { dateStyle: "short", timeStyle: "short" });
  };

  if (loading) return <div className="p-6 text-white">A carregar...</div>;
  if (error) return <div className="p-6 text-red-200">{error}</div>;

  return (
    <div className="min-h-screen orya-body-bg text-white px-4 py-10">
      <div className="mx-auto max-w-4xl space-y-6">
        <div>
          <h1 className="text-2xl font-semibold">As minhas duplas (Padel)</h1>
          <p className="text-white/70 text-sm">Acompanha o estado das tuas inscrições e pagamentos.</p>

```

```

</div>

{pairings.length === 0 && <p className="text-white/70 text-sm">Ainda não tens duplas.</p>

<div className="space-y-4">
  {pairings.map((p) => {
    const pendingSlot = p.slots.find((s) => s.slotStatus === "PENDING");
    const canPayPending = pendingSlot && pendingSlot.paymentStatus === "UNPAID" && p.paymentMode === "SPLIT";
    const inviteLink = p.inviteToken ? `${typeof window !== "undefined" ? window.location.origin : ""}/eventos/${p.eventSlug ?? ""}?inviteToken=${p.inviteToken}` : null;
    return (
      <div key={p.id} className="rounded-2xl border border-white/10 bg-white/5 p-4 shadow">
        <div className="flex items-center justify-between gap-2 flex-wrap">
          <div>
            <p className="text-lg font-semibold">{p.event.title}</p>
            <p className="text-xs text-white/60">
              {p.category?.label ? `${p.category.label} · ` : ""}Modo {p.paymentMode} · Estado {p.pairingStatus}
            </p>
          </div>
          {p.lockedUntil && (
            <span className="rounded-full border border-amber-300/40 bg-amber-300/10 px-3 py-1 text-[12px] text-amber-100">
              Limite: {formatDateTime(p.lockedUntil)}
            </span>
          )}
        </div>
      </div>
    )
  )}
</div>

<div className="mt-3 flex flex-wrap gap-2 text-[12px] text-white/75">
  {p.slots.map((s) => (
    <span key={s.id} className="rounded-full border border-white/15 px-2 py-1">
      {s.slotRole} · {s.slotStatus} · {s.paymentStatus}
    </span>
  )))
</div>

<div className="mt-4 flex gap-3 flex-wrap">
  {canPayPending && (
    <Link href={`/eventos/${p.eventSlug ?? ""}?pairingId=${p.id}`}
      className="rounded-full bg-white text-black px-4 py-2 text-sm font-semibold"
    >
      Pagar o meu lugar
    </Link>
  )}
  {p.pairingStatus !== "CANCELLED" && (
    <button type="button" onClick={() => doAction(p.id, "cancel")}
      disabled={actionLoading === `cancel-${p.id}`}
      className="rounded-full border border-white/20 px-4 py-2 text-sm text-white hover:bg-white/10 disabled:opacity-60"
    >
      {actionLoading === `cancel-${p.id}` ? "A cancelar..." : "Cancelar dupla"}
    </button>
  )}
  {canPayPending && (
    <button type="button" onClick={() => doAction(p.id, "assume")}
      disabled={actionLoading === `assume-${p.id}`}
      className="rounded-full border border-white/20 px-4 py-2 text-sm text-white hover:bg-white/10 disabled:opacity-60"
    >
      {actionLoading === `assume-${p.id}` ? "A assumir..." : "Assumir resto"}
    </button>
  )}
  {inviteLink && (
    <button type="button" onClick={() => {
      navigator.clipboard
        .writeText(inviteLink)
        .then(() => alert("Link copiado"))
        .catch(() => alert("Não foi possível copiar o link"));
    }}
      className="rounded-full border border-white/20 px-4 py-2 text-sm"
    >
      Copiar link de convite
    </button>
  )}
</div>

```

```

        </button>
    )}
</div>
</div>
);
});
</div>
</div>
</div>
);
}

```

app/page.tsx

```

import Link from "next/link";
import Image from "next/image";
import { prisma } from "@/lib/prisma";
import { mapEventToCardDTO, type EventCardDTO } from "@/lib/events";
import { defaultBlurDataURL, optimizeImageUrl } from "@/lib/image";

export const runtime = "nodejs";
export const dynamic = "force-dynamic";

function buildEventLink(event: EventCardDTO) {
  return event.type === "EXPERIENCE" ? `/experiencias/${event.slug}` : `/eventos/${event.slug}`;
}

function formatDateLabel(event: EventCardDTO) {
  if (!event.startsAt) return "Data a anunciar";
  const start = new Date(event.startsAt);
  const end = event.endsAt ? new Date(event.endsAt) : null;

  const day = start.toLocaleDateString("pt-PT", {
    weekday: "long",
    day: "2-digit",
    month: "2-digit",
  });

  const startTime = start.toLocaleTimeString("pt-PT", {
    hour: "2-digit",
    minute: "2-digit",
  });

  const endTime =
    end &&
    end.toLocaleTimeString("pt-PT", {
      hour: "2-digit",
      minute: "2-digit",
    });

  if (endTime) {
    return `${day} · ${startTime} - ${endTime}`;
  }
  return `${day} · ${startTime}`;
}

function formatPriceLabel(event: EventCardDTO) {
  if (event.isFree) return "Entrada gratuita";
  if (event.priceFrom == null) return "Preço a anunciar";
  const formatted = event.priceFrom.toLocaleString("pt-PT", {
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  });
  return `Desde ${formatted} €`;
}

export default async function HomePage() {
  let eventsRaw: Awaited<ReturnType<typeof prisma.event.findMany>> = [];

  try {
    eventsRaw = await prisma.event.findMany({
      where: { status: "PUBLISHED", isTest: false },
      orderBy: { startsAt: "asc" },
      include: {
        ticketTypes: {
          orderBy: { sortOrder: "asc" },
        }
      }
    });
  } catch (error) {
    console.error(error);
  }
}


```

```

        },
    },
    take: 12,
});
} catch (err) {
  console.error("[home] falha ao ligar à BD para listar eventos", err);
}

const events: EventCardDTO[] = eventsRaw
  .map(mapEventToCardDTO)
  .filter((e): e is EventCardDTO => e !== null);

const spotlight = events[0] ?? null;
const spotlightCover = spotlight
  ? optimizeImageUrl(spotlight.coverImageUrl, 1400, 70, "webp")
  : null;

return (
<main className="min-h-screen text-white pb-28 md:pb-8">
  <section className="mx-auto max-w-4xl px-4 py-6 space-y-6">
    <div className="flex items-center justify-between">
      <h1 className="text-lg font-semibold tracking-tight">ORYA</h1>
      <Link
        href="/explorar"
        className="text-[12px] text-[#6BFFFF] underline underline-offset-4"
      >
        Explorar
      </Link>
    </div>
    <div className="rounded-3xl border border-white/12 bg-gradient-to-br from-[#0B1229] via-[#0A0E1A] to-[#05060f] shadow-[0_24px_70px_rgba(15,23,42,0.85)]">
      <div className="flex items-center justify-between px-4 py-3">
        <h2 className="text-sm font-semibold">Em alta agora</h2>
        <Link href="/explorar" className="text-[11px] text-[#6BFFFF]">
          Ver tudo
        </Link>
      </div>
      <div className="mx-3 mb-3 overflow-hidden rounded-2xl border border-white/10 bg-black/30">
        {spotlight ?
          <>
            <div className="relative h-44 w-full overflow-hidden">
              {spotlightCover ? (
                <Image
                  src={spotlightCover}
                  alt={spotlight.title}
                  fill
                  sizes="(max-width: 768px) 100vw, (max-width: 1200px) 80vw, 1200px"
                  priority
                  className="object-cover transition-transform duration-500"
                  placeholder="blur"
                  blurDataURL={defaultBlurDataURL}
                />
              ) : (
                <div className="absolute inset-0 bg-[radial-gradient(circle_at_20%_20%,rgba(255,0,200,0.3),transparent_25%),radial-gradient(circle_at_80%_20%,rgba(107,255,255,0.3),transparent_30%),linear-gradient(135deg,#0b1224_0%,#050915_60%)]" />
              )}
            <div className="absolute inset-0 bg-gradient-to-t from-black/85 via-black/55 to-transparent" />
          </div>
        : (
          <div className="absolute bottom-3 left-4 right-4 flex items-end justify-between gap-3">
            <div>
              <p className="text-xs font-semibold text-zinc-50">
                {spotlight.title}
              </p>
              <p className="mt-0.5 text-[11px] text-zinc-200">
                {formatDateLabel(spotlight)}
              </p>
            </div>
            <p className="rounded-full bg-black/75 px-3 py-1 text-[11px] font-medium text-zinc-50">
              {formatPriceLabel(spotlight)}
            </p>
          </div>
        </div>
      </div>
      <div className="space-y-3 px-4 pb-4 pt-3">
        <p className="text-xs text-zinc-300">

```

```

        Evento em destaque. Abre para veres todos os detalhes.
    </p>
    <Link
        href={buildEventLink(spotlight)}
        className="mt-1 inline-flex w-full items-center justify-center rounded-2xl bg-gradient-to-r from-[#FF00C8] via-[#6BFFF] to-[#1646F5] px-4 py-2 text-xs font-semibold text-black shadow-lg shadow-[#6bffff80] hover:brightness-110"
    >
        Abrir página do evento
    </Link>
</div>
</>
) : (
<div className="px-4 py-6 text-sm text-zinc-200">
    Ainda não tens eventos criados.
    <br />
    <Link
        href="/organizador/eventos/novo"
        className="mt-2 inline-flex text-[13px] font-medium text-[#6BFFF] underline underline-offset-4"
    >
        Cria o teu primeiro evento →
    </Link>
</div>
)}
</div>
</div>

<div className="space-y-3 rounded-3xl border border-white/10 bg-black/25 p-4 shadow-[0_16px_50px_rgba(0,0,0,0.6)]">
<div className="flex items-center justify-between">
    <h2 className="text-sm font-semibold">Os teus eventos</h2>
    <Link href="/explorar" className="text-[11px] text-[#6BFFF]">
        Ver mais
    </Link>
</div>
<div className="rounded-2xl border border-white/10 bg-white/5 p-4 text-sm text-zinc-300">
    Ainda não tens eventos. Explora e junta-te a um evento para aparecer aqui.
</div>
</div>

<div className="space-y-3 rounded-3xl border border-white/10 bg-black/20 p-4 shadow-[0_16px_50px_rgba(0,0,0,0.6)]">
<div className="flex items-center justify-between">
    <h2 className="text-sm font-semibold">Sugestões personalizadas</h2>
</div>
<div className="rounded-2xl border border-white/10 bg-white/5 p-4 text-sm text-zinc-300">
    Ainda estamos a conhecer-te. À medida que usares a ORYA, as sugestões vão aparecer aqui.
</div>
</div>

<div className="space-y-3 rounded-3xl border border-white/10 bg-black/20 p-4 shadow-[0_16px_50px_rgba(0,0,0,0.6)]">
<div className="flex items-center justify-between">
    <h2 className="text-sm font-semibold">Oportunidades perto de ti agora</h2>
</div>
<div className="rounded-2xl border border-white/10 bg-white/5 p-4 text-sm text-zinc-300">
    Sem oportunidades perto de ti neste momento. Vais ser o primeiro a saber quando surgir algo porreiro.
</div>
</div>

<div className="space-y-2 rounded-3xl border border-white/10 bg-black/25 p-4 shadow-[0_16px_50px_rgba(0,0,0,0.6)]">
    <h2 className="text-sm font-semibold">Amigos vão a...
    <p className="text-sm text-zinc-300">
        Quando os teus amigos começarem a ir a eventos, vais ver aqui onde eles vão.
    </p>
</div>
</section>
</main>
);
}

```

app/signup/page.tsx

```

"use client";
import { Suspense, useEffect } from "react";
import { useSearchParams } from "next/navigation";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";

function SignupContent() {

```

```

const { openModal } = useAuthModal();
const searchParams = useSearchParams();
const redirectTo = searchParams.get("redirectTo") ?? "/";

useEffect(() => {
  openModal({ mode: "signup", redirectTo });
}, [openModal, redirectTo]);

return (
  <main className="min-h-screen flex items-center justify-center text-white">
    <p>
      Se o modal não abriu, clica aqui
      <button onClick={() => openModal({ mode: "signup", redirectTo })}>
        onClick={() => openModal({ mode: "signup", redirectTo })}
        className="underline"
      </button>
    </p>
  </main>
);
}

export default function SignupRedirectPage() {
  return (
    <Suspense fallback={null}>
      <SignupContent />
    </Suspense>
  );
}

```

app/staff/eventos/page.tsx

```

"use client";

import { useEffect } from "react";
import { useRouter } from "next/navigation";
import useSWR from "swr";
import Link from "next/link";
import { useUser } from "@/app/hooks/useUser";

const fetcher = async (url: string) => {
  const res = await fetch(url);
  if (!res.ok) {
    throw new Error("Falha ao carregar eventos de staff.");
  }
  return res.json();
};

type StaffEvent = {
  id: number;
  slug: string;
  title: string;
  startsAt: string;
  locationName: string | null;
  locationCity: string | null;
  organizerName: string | null;
};

type Invitation = {
  id: number;
  scope: "GLOBAL" | "EVENT";
  eventId: number | null;
  createdAt: string;
  event: {
    id: number;
    title: string;
    startsAt: string;
    locationName: string | null;
    locationCity: string | null;
  } | null;
  organizer: {
    id: number;
  };
};

```

```

    displayName: string | null;
} | null;
};

export default function StaffEventsPage() {
  const router = useRouter();
  const { user, isLoading: isUserLoading } = useUser();

  const {
    data,
    error,
    isLoading: isEventsLoading,
  } = useSWR<{ ok: boolean; events: StaffEvent[] }>(
    user ? "/api/staff/events" : null,
    fetcher
  );

  const {
    data: invitationsData,
    mutate: mutateInvites,
    isLoading: isInvitesLoading,
  } = useSWR<{ ok: boolean; invitations: Invitation[] }>(
    user ? "/api/staff/invitations" : null,
    fetcher
  );

  useEffect(() => {
    if (!isUserLoading && !user) {
      router.replace("/staff/login");
    }
  }, [user, isUserLoading, router]);

  if (isUserLoading) {
    return (
      <div className="max-w-4xl mx-auto px-4 py-8">
        <p>A carregar a tua conta...</p>
      </div>
    );
  }

  if (!user) {
    // Enquanto o redirect não acontece, mostramos um fallback simples
    return (
      <div className="max-w-4xl mx-auto px-4 py-8">
        <p>Redirecionar para a área de login de staff...</p>
      </div>
    );
  }
}

const events = data?.events ?? [];
const invitations = invitationsData?.invitations ?? [];

const acceptInvite = async (id: number) => {
  await fetch("/api/staff/invitations/accept", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ assignmentId: id }),
  });
  mutateInvites();
};

const rejectInvite = async (id: number) => {
  await fetch("/api/staff/invitations/reject", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ assignmentId: id }),
  });
  mutateInvites();
};

const getBadgeLabel = (startsAt: string) => {
  if (!startsAt) return "";
  const now = new Date();
  const start = new Date(startsAt);

  const sameDay =
    start.getFullYear() === now.getFullYear() &&
    start.getMonth() === now.getMonth() &&

```

```

start.getDate() === now.getDate();

if (sameDay) return "Hoje";
if (start < now) return "Já aconteceu";
return "Próximo";
};

const formatDateTime = (startsAt: string) => {
  if (!startsAt) return "Data por definir";
  const d = new Date(startsAt);
  return d.toLocaleString("pt-PT", {
    day: "2-digit",
    month: "short",
    year: "numeric",
    hour: "2-digit",
    minute: "2-digit",
  });
};

return (
  <div className="max-w-5xl mx-auto px-4 py-8 space-y-6">
    <div className="space-y-2">
      <h1 className="text-2xl font-semibold">Eventos para check-in</h1>
      <p className="text-sm text-white/70">
        Aqui vês os eventos onde tens permissões de staff para fazer check-in
        de participantes.
      </p>
    </div>

    <section className="space-y-2">
      <h2 className="text-sm font-semibold">Convites pendentes</h2>
      {isInvitesLoading && <p className="text-sm text-white/70">A carregar convites...</p>}
      {!isInvitesLoading && invitations.length === 0 && (
        <p className="text-sm text-white/50">Nenhum convite pendente.</p>
      )}
      {invitations.length > 0 && (
        <div className="space-y-2">
          {invitations.map((inv) => {
            const event = inv.event;
            const dateLabel = event?.startsAt
              ? new Date(event.startsAt).toLocaleString("pt-PT", {
                  day: "2-digit",
                  month: "short",
                  hour: "2-digit",
                  minute: "2-digit",
                })
              : "Data a anunciar";
            return (
              <div
                key={inv.id}
                className="flex items-center justify-between gap-3 rounded-lg border border-white/10 bg-white/5 px-4 py-3
text-sm"
            >
              <div className="min-w-0">
                <p className="font-medium">
                  {event?.title ?? "Evento"}
                </p>
                <p className="text-xs text-white/60">
                  {dateLabel} · {event?.locationName ?? event?.locationCity ?? "Local a definir"}
                </p>
                <p className="text-[11px] text-white/50">
                  Organizador: {inv.organizer?.displayName ?? "ORYA"}
                </p>
              </div>
              <div className="flex gap-2">
                <button
                  type="button"
                  onClick={() => acceptInvite(inv.id)}
                  className="rounded-md border border-emerald-400/60 px-3 py-1.5 text-xs font-medium text-emerald-100
hover:bg-emerald-500/10"
                >
                  Aceitar
                </button>
                <button
                  type="button"
                  onClick={() => rejectInvite(inv.id)}
                  className="rounded-md border border-red-400/60 px-3 py-1.5 text-xs font-medium text-red-100 hover:bg-red-
500/10"
                >
                  Recusar
                </button>
              </div>
            </div>
          )
        )
      )}
    </section>
  </div>
)

```

```

        >
        Recusar
      </button>
    </div>
  </div>
);
})
</div>
)}
</section>

{isEventsLoading && (
  <p>A carregar eventos onde tens acesso como staff...</p>
)}

{error && (
  <p className="text-sm text-red-400">
    Ocorreu um erro ao carregar os eventos de staff.
  </p>
)}
{!isEventsLoading && !error && events.length === 0 && (
  <div className="rounded-lg border border-white/10 bg-white/5 p-6 text-sm">
    <p>
      Neste momento não tens nenhum evento atribuído como staff. Pede ao
      organizador para te adicionar.
    </p>
  </div>
)}
{events.length > 0 && (
  <div className="space-y-3">
    {events.map((event) => {
      const badge = getBadgeLabel(event.startsAt);
      const dateLabel = formatDateTime(event.startsAt);
      const locationLabel = event.locationCity
        ? `${event.locationCity}${event.locationName} ? - ${event.locationName}` : ""
        : event.locationName ?? "Local a definir";

      return (
        <div
          key={event.id}
          className="flex items-center justify-between gap-4 rounded-lg border border-white/10 bg-white/5 px-4 py-3"
        >
          <div className="min-w-0">
            <div className="flex items-center gap-2">
              <p className="truncate text-sm font-medium">
                {event.title || "Evento sem título"}
              </p>
              {badge &&
                <span className="inline-flex items-center rounded-full bg-white/10 px-2 py-0.5 text-[10px] font-medium uppercase tracking-wide">
                  {badge}
                </span>
              }
            </div>
            <p className="mt-1 text-xs text-white/60">{dateLabel}</p>
            <p className="mt-1 text-xs text-white/60">{locationLabel}</p>
            {event.organizerName &&
              <p className="mt-1 text-[11px] text-white/50">
                Organizador: {event.organizerName}
              </p>
            }
          </div>
          <button
            type="button"
            onClick={() => router.push(`/staff/scan?eventId=${event.id}`)}
            className="shrink-0 rounded-md border border-white/15 px-3 py-1.5 text-xs font-medium hover:bg-white/10"
          >
            Abrir scanner
          </button>
        </div>
      );
    })
  </div>
)}
</div>
)
}

```

```
</div>
);
}
```

app/staff/login/page.tsx

```
"use client";

import { Suspense, useEffect, useState } from "react";
import { useRouter, useSearchParams } from "next/navigation";
import { useUser } from "@/app/hooks/useUser";
import { useAuthModal } from "@/app/components/autenticação/AuthModalContext";
import { supabaseBrowser } from "@/lib/supabaseBrowser";

function StaffLoginContent() {
  const router = useRouter();
  const search = useSearchParams();
  const { user, isLoading } = useUser();
  const { openModal } = useAuthModal();

  const [identifier, setIdentifier] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState<string | null>(null);
  const [submitting, setSubmitting] = useState(false);

  const redirectTo = search.get("redirectTo") || "/staff/eventos";

  useEffect(() => {
    if (!isLoading && user) {
      router.replace(redirectTo);
    }
  }, [user, isLoading, router, redirectTo]);

  const handleLogin = async () => {
    setError(null);
    setSubmitting(true);
    try {
      if (!identifier || !password) {
        setError("Preenche email/username e password.");
        return;
      }
      let emailToUse = identifier;
      if (!identifier.includes("@")) {
        const res = await fetch("/api/auth/resolve-identifier", {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ identifier }),
        });
        const data = await res.json().catch(() => null);
        if (!res.ok || !data?.ok || !data?.email) {
          setError("Credenciais inválidas. Confirma username/email e password.");
          return;
        }
        emailToUse = data.email;
      }
    } catch (err) {
      console.error("staff/login error", err);
      setError("Não foi possível iniciar sessão.");
    } finally {
      setSubmitting(false);
    }
  };

  if (isLoading) {
```

```

    return (
      <div className="min-h-screen bg-black text-white flex items-center justify-center px-6">
        <p>A carregar a tua sessão...</p>
      </div>
    );
  }

  return (
    <div className="min-h-screen bg-black text-white flex items-center justify-center px-6">
      <div className="w-full max-w-sm bg-white/5 backdrop-blur-xl p-8 rounded-2xl border border-white/10 shadow-2xl space-y-4">
        <div className="space-y-2 text-center">
          <h1 className="text-2xl font-semibold">Modo Staff ORYA</h1>
          <p className="text-sm text-white/70">
            Entra com a tua conta ORYA para aceder à área de staff e fazer o
            check-in dos participantes nos eventos onde tens permissão.
          </p>
        </div>

        <div className="space-y-2 text-sm">
          <input
            type="text"
            value={identifier}
            onChange={(e) => setIdentifier(e.target.value)}
            className="w-full rounded-md border border-white/10 bg-white/5 px-3 py-2 text-sm outline-none focus:border-white/30"
            placeholder="email@exemplo.com ou @username"
          />
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="w-full rounded-md border border-white/10 bg-white/5 px-3 py-2 text-sm outline-none focus:border-white/30"
            placeholder="Palavra-passe"
          />
          {error && <p className="text-xs text-red-300">{error}</p>}
        </div>

        <button
          type="button"
          disabled={submitting}
          onClick={handleLogin}
          className="w-full bg-gradient-to-r from-[#FF00C8] via-[#BFFF00] to-[#1646F5] text-black py-3 rounded-lg font-semibold
          hover:opacity-90 transition disabled:opacity-50"
        >
          {submitting ? "A entrar..." : "Entrar com a minha conta ORYA"}
        </button>

        <button
          type="button"
          onClick={() => openModal({ mode: "login", redirectTo, showGoogle: true })}
          className="w-full text-[12px] text-white/70 underline underline-offset-4"
        >
          Preferes usar o modal standard?
        </button>
      </div>
    );
  }

  export default function StaffLoginPage() {
    return (
      <Suspense fallback={null}>
        <StaffLoginContent />
      </Suspense>
    );
  }
}

```

app/staff/scan/page.tsx

```

"use client";

import { Suspense, useEffect, useRef, useState } from "react";
import { useSearchParams } from "next/navigation";

type ValidateQrResponse = {
  ok: boolean;
  reason?: string;
}

```

```

message?: string;
ticketId?: string;
eventId?: number;
status?: string;
};

function StaffScanContent() {
  const videoRef = useRef<HTMLVideoElement | null>(null);
  const [scanning, setScanning] = useState(false);
  const [result, setResult] = useState<ValidateOrResponse | null>(null);
  const [error, setError] = useState<string | null>(null);
  const [checkingSession, setCheckingSession] = useState(true);

  const searchParams = useSearchParams();
  const eventIdParam = searchParams.get("eventId");
  const eventId = eventIdParam ? Number(eventIdParam) : null;

  useEffect(() => {
    let cancelled = false;
    async function check() {
      try {
        const res = await fetch("/api/staff/events", { cache: "no-store" });
        if (res.status === 401) {
          window.location.href = `/staff/login?redirectTo=/staff/scan${eventId ? `?eventId=${eventId}` : ""}`;
          return;
        }
        const json = await res.json().catch(() => null);
        if (!json?.ok) {
          setError("Não tens sessão de staff válida.");
          return;
        }
        const events: { id: number }[] = Array.isArray(json?.events) ? json.events : [];
        const hasAccess =
          events.some((ev) => ev.id === eventId) || events.length > 0;
        if (!hasAccess) {
          setError("Não tens permissões para este evento como staff.");
          return;
        }
      } catch (e) {
        console.error("Erro ao validar sessão de staff:", e);
        setError("Erro ao validar sessão de staff.");
      } finally {
        if (!cancelled) setCheckingSession(false);
      }
    }
    check();
    return () => {
      cancelled = true;
    };
  }, [eventId]);

  // Start camera
  async function startCamera() {
    setError(null);
    try {
      const stream = await navigator.mediaDevices.getUserMedia({
        video: { facingMode: "environment" },
      });
      if (videoRef.current) {
        videoRef.current.srcObject = stream;
        await videoRef.current.play();
      }
      setScanning(true);
    } catch (e) {
      console.error("Camera error:", e);
      setError("Não foi possível aceder à câmara.");
    }
  }

  // Fake scan (MVP) - real scanner virá depois
  async function simulateScan() {
    setError(null);
    setResult(null);
    const fake = prompt("Cola aqui o token ORYA2 para validar:");
    if (!fake) return;
    if (!eventId) {
      setError("Falta o eventId na URL. Abre o scanner a partir da página de eventos de staff.");
      return;
    }
  }
}

```

```

}

try {
  const res = await fetch("/api/staff/validate-qr", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ token: fake, eventId }),
  });

  const json = await res.json();
  setResult(json);

  if (!res.ok || json?.ok === false) {
    setError("Bilhete inválido, expirado ou já utilizado.");
  }
} catch (e) {
  console.error("Erro a validar QR:", e);
  setError("Erro ao comunicar com o servidor. Tenta novamente.");
}
}

if (checkingSession) {
  return (
    <main className="min-h-screen bg-black text-white flex items-center justify-center">
      <p className="text-sm text-white/70">A validar sessão de staff...</p>
    </main>
  );
}

return (
  <main className="min-h-screen bg-black text-white p-6">
    <h1 className="text-2xl font-bold mb-6">Scanner ORYA – Staff</h1>

    <p className="text-sm text-white/70 mb-4">
      {eventId
        ? `A validar bilhetes para o evento #${eventId}.`
        : "Nenhum evento seleccionado. Volta à página de eventos de staff e entra pelo botão 'Abrir scanner'."}
    </p>

    {eventId ? (
      <>
        {/* Camera Box */}
        <div className="w-full max-w-md mx-auto rounded-xl border border-white/20 bg-white/5 p-4">
          <video
            ref={videoRef}
            className="w-full rounded-lg"
            autoPlay
            muted
            playsInline
          />

          {!scanning && (
            <button
              onClick={startCamera}
              className="mt-4 w-full py-2 rounded-lg bg-gradient-to-r from-fuchsia-500 to-cyan-400 text-black font-semibold"
            >
              Activar Câmara
            </button>
          )}
        </div>

        {error && (
          <p className="text-red-400 mt-4 text-center text-sm">{error}</p>
        )}
      </>
    )/* Fake Scan Button */
    <div className="text-center mt-6">
      <button
        onClick={simulateScan}
        className="px-4 py-2 rounded-lg bg-white text-black font-semibold"
      >
        Simular Scan (MVP)
      </button>
    </div>

    /* Result Box */
    {result && (

```

```

        <div className="mt-6 p-4 rounded-xl border border-white/20 bg-white/5 max-w-md mx-auto">
          <pre className="text-xs whitespace-pre-wrap">
            {JSON.stringify(result, null, 2)}
          </pre>
        </div>
      )}
    </>
  ) : (
<p className="mt-6 text-sm text-red-300">
  Nenhum evento selecionado. Volta à página de eventos de staff e usa o
  botão Abrir scanner para escolher o evento certo.
</p>
)
</main>
);
}

export default function StaffScanPage() {
  return (
    <Suspense fallback={null}>
      <StaffScanContent />
    </Suspense>
  );
}

```

app/torneios/[slug]/live/page.tsx

```

// app/torneios/[slug]/live/page.tsx
"use client";

import useSWR from "swr";
import { useMemo } from "react";
import { useSearchParams } from "next/navigation";

const fetcher = (url: string) => fetch(url).then((r) => r.json());

export default function TournamentLivePage({ params }: { params: { slug: string } }) {
  const slug = params.slug;
  const searchParams = useSearchParams();
  const url = useMemo(() => {
    const base = `/api/tournaments/${slug}/live`;
    return base;
  }, [slug, searchParams]);

  const { data, error } = useSWR(url, fetcher, { refreshInterval: 1000 });
  if (error) return <div className="p-4 text-white/70">Erro a carregar live.</div>;
  if (!data?.ok) return <div className="p-4 text-white/70">A carregar...</div>

  const tour = data.tournament;
  const stages = tour.stages || [];
  const nextMatch = tour.nextMatch || null;
  const lastMatch = tour.lastMatch || null;
  const pairingIdFromQuery = searchParams?.get("pairingId");

  return (
    <div className="space-y-4 rounded-2xl border border-white/10 bg-black/40 p-4">
      <div className="flex items-center justify-between">
        <div>
          <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Live</p>
          <h1 className="text-xl font-semibold text-white">{data.event?.title ?? "Torneio"}</h1>
          <p className="text-white/70 text-sm">Formato: {tour.format}</p>
        </div>
        <div className="rounded-xl border border-white/15 bg-white/5 px-3 py-2 text-sm text-white/80">
          <p>Próximo jogo: {nextMatch ? `#${nextMatch.id}` : "-"}</p>
          <p>Último: {lastMatch ? `#${lastMatch.id}` : "-"}</p>
        </div>
      </div>
      <div className="grid gap-4 md:grid-cols-2">
        {stages.map((s: any) => (
          <div key={s.id} className="rounded-xl border border-white/10 bg-white/5 p-3 space-y-2">
            <div className="flex items-center justify-between">
              <p className="text-white font-semibold">{s.name || s.stageType}</p>
              <span className="text-[11px] text-white/60">{s.stageType}</span>
            </div>
          </div>
        ))
        {s.groups.map((g: any) => (

```

```

<div key={g.id} className="rounded-lg border border-white/10 bg-black/30 p-2 space-y-1">
  <p className="text-white text-sm">{g.name}</p>
  {g.standings?.length ? (
    <div className="space-y-1 text-[12px] text-white/80">
      {g.standings.map((st: any, idx: number) => (
        <div
          key={st.pairingId}
          className={` flex items-center justify-between rounded border px-2 py-1 ${(
            pairingIdFromQuery && `${st.pairingId}` === pairingIdFromQuery
            ? "border-emerald-400/60 bg-emerald-500/10"
            : "border-white/10 bg-white/5"
          )}`}
        >
          <span className="text-white">{idx + 1}º · Dupla #${st.pairingId}</span>
          <span className="text-white/60">V {st.wins} · Sets {st.setDiff}</span>
        </div>
      )))
    </div>
  ) : (
    <p className="text-[12px] text-white/60">Sem standings.</p>
  )}
<div className="text-[12px] text-white/70 space-y-1">
  {g.matches.map((m: any) => (
    <div
      key={m.id}
      className={` rounded border px-2 py-1 flex items-center justify-between ${(
        pairingIdFromQuery &&
        (`${m.pairing1Id}` === pairingIdFromQuery || `${m.pairing2Id}` === pairingIdFromQuery)
        ? "border-emerald-400/60 bg-emerald-500/10"
        : "border-white/10 bg-white/5"
      )}`}
    >
      <span className="text-white">#${m.id} · {m.pairing1Id ?? "?"} vs {m.pairing2Id ?? "?"}</span>
      <span className="text-white/60">{m.statusLabel}</span>
    </div>
  )))
</div>
</div>
))}

{s.matches.length > 0 && (
  <div className="space-y-1 text-[12px] text-white/80">
    <p className="text-[11px] uppercase tracking-[0.18em] text-white/60">Playoffs</p>
    {s.matches.map((m: any) => (
      <div
        key={m.id}
        className={` rounded border px-2 py-1 flex items-center justify-between ${(
          pairingIdFromQuery &&
          (`${m.pairing1Id}` === pairingIdFromQuery || `${m.pairing2Id}` === pairingIdFromQuery)
          ? "border-emerald-400/60 bg-emerald-500/10"
          : "border-white/10 bg-white/5"
        )}`}
      >
        <span className="text-white">#${m.id} · {m.pairing1Id ?? "?"} vs {m.pairing2Id ?? "?"}</span>
        <span className="text-white/60">{m.statusLabel}</span>
      </div>
    )))
  </div>
  </div>
));
</div>
);
}

```

app/types/event.ts

```

// types/event.ts

export type EventVisibility = 'public' | 'unlisted';

export type TicketType = {
  id: string;
  name: string;           // ex: "Wave 1", "Geral", "VIP"
  price: number;          // 0 = grátis
}

```

```

    currency: 'EUR';
    quantity?: number;      // stock opcional
    waveOrder?: number;     // 1, 2, 3...
};

export type Event = {
  id: string;
  slug: string;

  title: string;
  description: string;

  coverImage: string;
  galleryImages?: string[];

  locationName: string; // "Fly Padel"
  address: string;      // "Rua X, Porto"
  latitude?: number;
  longitude?: number;

  // Datas em ISO + timezone para no futuro tratar bem os fusos
  startDate: string;    // "2025-12-20T09:00:00.000Z"
  endDate: string;       // "2025-12-20T19:00:00.000Z"
  timezone: string;      // ex: "Europe/Lisbon"

  visibility: EventVisibility; // public ou unlisted (só link)

  isFree: boolean;
  tickets: TicketType[];

  capacity?: number;
  remainingSpots?: number;

  minAge?: number;        // ex: 18
  tags: string[];          // ["padel", "desporto", "torneio"]
}

organizerName: string;
organizerAvatarUrl?: string;

interestedCount: number; // likes / interessados
goingCount: number;      // pessoas que marcaram "vou"
};

```

components/BackLink.tsx

```

"use client";

import { useRouter } from "next/navigation";

type Props = {
  hrefFallback?: string;
  label?: string;
  className?: string;
};

export default function BackLink({ hrefFallback = "/", label = "Voltar", className = "" }: Props) {
  const router = useRouter();

  const handleClick = (e: React.MouseEvent<HTMLButtonElement>) => {
    e.preventDefault();
    if (typeof window !== "undefined" && window.history.length > 1) {
      window.history.back();
    } else {
      router.push(hrefFallback);
    }
  };

  return (
    <button
      type="button"
      onClick={handleClick}
      className={` inline-flex items-center gap-2 rounded-full border border-white/15 bg-white/5 px-3 py-1.5 text-sm text-white/80 hover:bg-white/10 ${className}`}
    >
      <span className="text-lg leading-none"></span>
      <span>{label}</span>
    
```

```
    </button>
);
}
```

components/organizador/eventos/wizard/StepperDots.tsx

```
"use client";

import React from "react";

type StepId = "formato" | "essenciais" | "datas_local" | "bilhetes" | "revisao";

export type WizardStep = { id: StepId; title: string };

function CheckIcon(props: React.SVGProps<SVGSVGELEMENT>) {
  return (
    <svg viewBox="0 0 24 24" fill="none" {...props}>
      <path
        d="M20 6L9 17L5 5"
        stroke="currentColor"
        strokeWidth="2.2"
        strokeLinecap="round"
        strokeLinejoin="round"
      />
    </svg>
  );
}

export function StepperDots({
  steps,
  current,
  maxUnlockedIndex,
  onGoTo,
}: {
  steps: WizardStep[];
  current: StepId;
  maxUnlockedIndex: number;
  onGoTo?: (id: StepId) => void;
}) {
  const currentIndex = Math.max(
    0,
    steps.findIndex((s) => s.id === current),
  );
  const progress =
    steps.length > 1 ? Math.min(100, Math.max(0, (currentIndex / (steps.length - 1)) * 100)) : 0;
  const dotSize = 36; // px
  const lineTop = dotSize / 2; // center of the dot

  return (
    <div className="w-full">
      <div className="relative py-3">
        <div
          className="absolute left-0 right-0 h-px bg-white/10"
          style={{ top: lineTop }}
          aria-hidden
        />
        <div
          className="absolute left-0 h-px bg-gradient-to-r from-[var(--orya-blue)] via-[var(--orya-cyan)] to-[var(--orya-pink)]"
          style={{ width: `${progress}%`, top: lineTop }}
          aria-hidden
        />
      </div>
      <ol className="relative flex items-start justify-between gap-4 px-1">
        {steps.map((s, i) => {
          const clickable = i <= maxUnlockedIndex;
          const status = i < currentIndex ? "done" : i === currentIndex ? "current" : "todo";
          const isLockedFuture = i > maxUnlockedIndex;
          const isUnlockedFuture = i > currentIndex && !isLockedFuture;
          return (
            <li
              key={s.id}
              className="group min-w-0 flex flex-1 flex-col items-center text-center"
            >
              <button
                type="button"
                onClick={() => clickable && onGoTo?.(s.id)}
              >
```

```

        disabled={!clickable}
        className={[
          "relative grid place-items-center rounded-full outline-none",
          "transition-all duration-250 ease-[cubic-bezier(0.2,0.8,0.2,1)] transform",
          clickable ? "cursor-pointer focus-visible:ring-2 focus-visible:ring-[var(--orya-cyan)]/60 focus-
visible:ring-offset-2 focus-visible:ring-offset-transparent" : "cursor-not-allowed",
          status === "done" &&
          "bg-emerald-400/10 border border-emerald-300/25 backdrop-blur-md shadow-
[0_0_0_1px_rgba(16,185,129,.12),0_0_18px_rgba(16,185,129,.14)]",
          status === "current" &&
          "bg-white/8 border border-white/25 backdrop-blur-md shadow-
[0_0_0_1px_rgba(255,255,255,.10),0_0_26px_rgba(255,255,255,.16)] scale-100",
          status === "todo" &&
          (isLockedFuture
            ? "bg-white/4 border border-white/10 opacity-35"
            : "bg-white/5 border border-white/14 opacity-70 hover:opacity-90"),
          clickable && "hover:border-white/20 hover:shadow-[0_0_18px_rgba(255,255,255,0.1)]",
          status !== "current" && "scale-[0.98]",
        ].join(" "))
      aria-current={status === "current" ? "step" : undefined}
      title={clickable ? "Editar este passo" : undefined}
      style={{ width: dotSize, height: dotSize }}
    >
  <span
    className={[
      "text-xs font-semibold",
      status === "done" ? "text-emerald-200" : status === "current" ? "text-white/90" : "text-white/80",
    ].join(" ")}

  >
  {i + 1}
  </span>
</button>

<span
  className={[
    "mt-2 text-[11px] font-semibold tracking-[0.18em] uppercase leading-tight transition-colors",
    status === "done"
      ? "text-emerald-200/80"
      : status === "current"
        ? "text-white"
        : isLockedFuture
          ? "text-white/65 group-hover:text-white/85"
          : "text-white/40",
  ].join(" ")}

  >
  {s.title}
  </span>
</li>
);
});
</ol>
</div>
</div>
);
}

```

components/organizer/BecomeOrganizerForm.tsx

```

"use client";

import { useEffect, useMemo, useRef, useState } from "react";
import Link from "next/link";
import { useRouter } from "next/navigation";
import { Controller, useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { sanitizeUsername, validateUsername } from "@lib/username";
import {
  BecomeOrganizerFormValues,
  becomeOrganizerSchema,
} from "@lib/validation/organization";

type UsernameStatus = "idle" | "checking" | "available" | "taken" | "error";

const USERNAME_HELPER = "O teu @ é único na ORYA e vai aparecer no teu perfil, eventos e links.";

const gradientByEntity: Record<string, string> = {

```

```

PROMOTOR_ORGANIZADOR: "from-[#7C3AED]/20 via-[#0B1A38]/70 to-[#0EA5E9]/20",
EMPRESA_MARCA: "from-[#0EA5E9]/18 via-[#0B122B]/75 to-[#10B981]/15",
OUTRO: "from-[#FF6BCA]/15 via-[#0B132D]/75 to-[#6BFFFF]/18",
};

const badgeColors = [
  "bg-[#6BFFFF]",
  "bg-[#FF6BCA]",
  "bg-[#7C3AED]",
  "bg-[#10B981]",
  "bg-[#F59E0B]",
  "bg-[#38BDF8]",
];

const suggestionSuffixes = ["events", "official", "pt", "live", "club", "hq"];

const InfoTooltip = ({ text }: { text: string }) => (
  <span className="group relative inline-flex items-center">
    <span className="ml-1 inline-flex h-4 w-4 items-center justify-center rounded-full border border-white/25 bg-white/10 text-[10px] leading-none text-white/80">
      i
    </span>
    <span className="pointer-events-none absolute left-1/2 top-full mt-2 w-52 -translate-x-1/2 rounded-md border border-white/10 bg-black/80 px-3 py-2 text-[11px] leading-snug text-white/80 opacity-0 shadow-lg backdrop-blur transition-opacity duration-150 group-hover:opacity-100">
      {text}
    </span>
  </span>
);

function hashToIndex(value: string, length: number) {
  let hash = 0;
  for (let i = 0; i < value.length; i += 1) {
    hash = (hash << 5) - hash + value.charCodeAt(i);
    hash |= 0;
  }
  return Math.abs(hash) % length;
}

function initialsFromName(name: string) {
  if (!name.trim()) return "OR";
  const parts = name.trim().split(/\s+/).slice(0, 2);
  const initials = parts.map((p) => p[0]?.toUpperCase()).join("");
  return initials || "OR";
}

function buildUsernameSuggestions(base: string) {
  if (!base) return [];
  const cleaned = sanitizeUsername(base);
  const suggestions = suggestionSuffixes
    .map((suffix) => sanitizeUsername(` ${cleaned}${suffix.length ? `-${suffix}` : ""}`))
    .filter(Boolean);

  const unique: string[] = [];
  suggestions.forEach((s) => {
    if (s && !unique.includes(s) && s.length <= 30) unique.push(s);
  });
  return unique.slice(0, 3);
}

export default function BecomeOrganizerForm() {
  const formId = "become-organizer-form";
  const router = useRouter();
  const [usernameHelper, setUsernameHelper] = useState(USERNAME_HELPER);
  const [usernameStatus, setUsernameStatus] = useState<UsernameStatus>("idle");
  const [saving, setSaving] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [success, setSuccess] = useState<string | null>(null);
  const usernameCheckTimeout = useRef<NodeJS.Timeout | null>(null);
  const lastChecked = useRef<string>("");
  const [isLoaded, setIsLoaded] = useState(false);

  const form = useForm<BecomeOrganizerFormValues>({
    resolver: zodResolver(becomeOrganizerSchema),
    mode: "onChange",
    defaultValues: {
      entityType: "",
      businessName: "",
```

```

    city: "",
    website: "",
    iban: "",
    taxId: "",
    username: "",
  },
});

const watchEntityType = form.watch("entityType");
const watchBusinessName = form.watch("businessName");
const watchCity = form.watch("city");
const watchUsername = form.watch("username");

useEffect(() => {
  const t = setTimeout(() => setIsLoaded(true), 200);
  return () => clearTimeout(t);
}, []);

const avatarSeed = watchUsername || watchBusinessName || "orya";
const avatarColor = badgeColors[hashToIndex(avatarSeed, badgeColors.length)];
const avatarInitials = initialsFromName(watchBusinessName || "Organizaçāo");
const usernameClean = sanitizeUsername(watchUsername);

const isFormValid =
  !saving && form.formState.isValid && validateUsername(usernameClean).valid;

const gradientOverlay = useMemo(() => {
  if (watchEntityType && gradientByEntity[watchEntityType]) return gradientByEntity[watchEntityType];
  return "from-white/6 via-transparent to-[#6BFFFF]/8";
}, [watchEntityType]);

const checkUsername = async (value: string) => {
  const cleaned = sanitizeUsername(value);
  if (!cleaned) {
    setUsernameStatus("idle");
    setUsernameHelper(USERNAME_HELPER);
    return false;
  }
  const validation = validateUsername(cleaned);
  if (!validation.valid) {
    setUsernameStatus("error");
    setUsernameHelper(validation.error);
    return false;
  }
  if (lastChecked.current === cleaned && usernameStatus === "available") {
    return true;
  }
  setUsernameHelper("A verificar disponibilidade...");
  setUsernameStatus("checking");
  try {
    const res = await fetch(`^/api/username/check?username=${encodeURIComponent(cleaned)}`);
    if (!res.ok) {
      setUsernameStatus("error");
      setUsernameHelper("Não foi possível verificar o @ agora.");
      return false;
    }
    const data = (await res.json()).catch(() => null) as { available?: boolean } | null;
    const available = Boolean(data?.available);
    lastChecked.current = cleaned;
    if (available) {
      setUsernameStatus("available");
      setUsernameHelper("Este @ está disponível.");
    } else {
      setUsernameStatus("taken");
      setUsernameHelper("Este @ já está a ser usado – escolhe outro.");
    }
    return available;
  } catch (err) {
    console.error("[organizador/become] erro check username", err);
    setUsernameStatus("error");
    setUsernameHelper("Erro ao verificar o @.");
    return false;
  }
};

useEffect(() => {
  if (usernameCheckTimeout.current) clearTimeout(usernameCheckTimeout.current);
  usernameCheckTimeout.current = setTimeout(() => {

```

```

    checkUsername(watchUsername);
}, 450);
return () => {
  if (usernameCheckTimeout.current) clearTimeout(usernameCheckTimeout.current);
};
// eslint-disable-next-line react-hooks/exhaustive-deps
}, [watchUsername]);

const handleSubmit = form.handleSubmit(async (values) => {
  setError(null);
  setSuccess(null);

  const cleanedUsername = sanitizeUsername(values.username);
  const usernameValidation = validateUsername(cleanedUsername);
  if (!usernameValidation.valid) {
    setUsernameStatus("error");
    setUsernameHelper(usernameValidation.error);
    return;
  }

  const usernameAvailable = await checkUsername(cleanedUsername);
  if (!usernameAvailable) {
    setError("Este @ já está a ser usado – escolhe outro.");
    return;
  }

  const normalizedWebsite = (() => {
    if (!values.website) return null;
    if (values.website.startsWith("@")) return values.website;
    return `/https?:\/\/i.test(${values.website})` ? values.website : `https://${values.website}`;
  })();
  setSaving(true);
  try {
    const res = await fetch("/api/organizador/organizations", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        entityType: values.entityType.trim(),
        businessName: values.businessName.trim(),
        city: values.city.trim(),
        website: normalizedWebsite,
        payoutIban: values.iban ? values.iban.replace(/\s+/g, "") : null,
        nif: values.taxId || null,
        displayName: values.businessName.trim(),
        username: cleanedUsername,
      }),
    });
    const data = await res.json().catch(() => null);
    if (!res.ok || data?.ok === false) {
      setError(data?.error || "Não foi possível criar a organização.");
      setSaving(false);
      return;
    }

    if (data?.organizer?.id) {
      await fetch("/api/organizador/organizations/switch", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ organizerId: data.organizer.id }),
      });
    }

    setSuccess("Organização criada com sucesso. Bem-vindo ao painel da ORYA.");
    setTimeout(() => {
      router.replace("/organizador");
    }, 320);
  } catch (err) {
    console.error("[organizador/become] erro:", err);
    setError("Erro inesperado ao criar organização.");
    setSaving(false);
  }
});

const usernameMessageClass = () => {
  if (usernameStatus === "available") return "text-emerald-300";
  if (usernameStatus === "taken" || usernameStatus === "error") return "text-red-300";
  if (usernameStatus === "checking") return "text-white/65";
}

```

```

        return "text-white/55";
    })();

const usernameBorderClass =
  usernameStatus === "available"
    ? "border-emerald-300/60 focus:border-emerald-300/80 shadow-[0_0_1px_rgba(16,185,129,0.35)]"
    : usernameStatus === "taken" || form.formState.errors.username
    ? "border-red-400/70 focus:border-red-300 shadow-[0_0_1px_rgba(248,113,113,0.4)]"
    : "border-white/15 focus:border-[#6BFFFF]";

const usernameSuggestions =
  usernameStatus === "taken" ? buildUsernameSuggestions(usernameClean || watchBusinessName) : [];

if (!isLoading) {
  return (
    <div className="relative mx-auto max-w-[1160px] overflow-hidden rounded-3xl border border-white/8 bg-white/[0.04] p-8 md:p-9 lg:p-10 shadow-[0_25px_80px_rgba(0,0,0,0.45)]">
      <div className="pointer-events-none absolute inset-0 bg-gradient-to-br from-white/6 via-transparent to-[#6BFFFF]/8" />
      <div className="relative grid gap-8 md:grid-cols-2">
        <div className="space-y-4 animate-pulse">
          <div className="h-4 w-32 rounded bg-white/10" />
          <div className="h-6 w-3/4 rounded bg-white/10" />
          <div className="h-4 w-32 rounded bg-white/10" />
          <div className="h-20 rounded-2xl bg-white/5" />
          <div className="h-20 rounded-2xl bg-white/5" />
          <div className="h-20 rounded-2xl bg-white/5" />
        </div>
        <div className="h-16 rounded-2xl bg-white/5" />
        <div className="h-10 w-1/2 rounded-full bg-white/5" />
      </div>
      <div className="space-y-4 animate-pulse">
        <div className="h-4 w-40 rounded bg-white/10" />
        <div className="space-y-3">
          {[...Array(6)].map((_, i) => (
            <div key={i} className="h-10 rounded-xl bg-white/5" />
          )))
        </div>
        <div className="h-24 rounded-2xl bg-white/5" />
        <div className="h-12 rounded-full bg-white/10" />
      </div>
    </div>
    <p className="mt-4 text-center text-[12px] text-white/55">A preparar o teu espaço na ORYAN!</p>
  </div>
);
}

return (
  <div className="relative mx-auto max-w-[1160px] overflow-hidden rounded-3xl border border-white/8 bg-white/[0.04] p-8 md:p-9 lg:p-10 shadow-[0_25px_80px_rgba(0,0,0,0.45)]">
    <div className={`${pointerEventsNone} absolute inset-0 bg-gradient-to-br ${gradientOverlay}`} />
    <div className="pointer-events-none absolute inset-0 bg-[radial-gradient(circle_at_20%_20%,rgba(255,255,255,0.06),transparent_35%),radial-gradient(circle_at_80%_0%,rgba(107,255,255,0.04),transparent_40%)]" />

    <div className="relative grid items-start gap-y-10 md:grid-cols-2 md:gap-x-28 lg:gap-x-32">
      <div className="pointer-events-none absolute inset-y-8 left-1/2 hidden w-14 -translate-x-1/2 rounded-full bg-gradient-to-b from-transparent via-white/10 to-transparent md:block" />
      {/* Coluna ESQUERDA - formulário completo */}
      <div className="order-1 md:order-1 md:pr-10 lg:pr-12">
        <form
          id={formId}
          onSubmit={handleSubmit}
          className="space-y-8 rounded-2xl border border-white/10 bg-black/30 p-6 md:p-7 lg:p-8 backdrop-blur">
          <div>
            {error && (
              <div className="rounded-2xl border border-red-400/40 bg-red-900/30 px-4 py-3 text-sm text-red-100">
                {error}
              </div>
            )}
            {success && (
              <div className="rounded-2xl border border-emerald-400/40 bg-emerald-900/30 px-4 py-3 text-sm text-emerald-100">
                {success}
              </div>
            )}
          </div>
        <div className="flex items-start justify-between gap-2">
          <div className="space-y-1">
            <p className="text-[11px] uppercase tracking-[0.18em] text-white/65">Dados da organização</p>

```

```

        <h3 className="text-lg font-semibold">Informação base</h3>
    </div>
    <p className="text-[11px] text-white/55 leading-relaxed text-right">
        Campos marcados com * são obrigatórios.
    </p>
</div>

<div className="space-y-5">
    <div className="space-y-2">
        <div className="flex items-center gap-1 text-[12px] text-white/70">
            <span>Tipo de entidade *</span>
            <InfoTooltip text="Escolhe se és promotor de eventos, empresa/marca ou outro tipo de organização que cria experiências." />
        </div>
        <Controller
            name="entityType"
            control={form.control}
            render={({ field }) => (
                <select
                    {...field}
                    className={`w-full rounded-xl border bg-black/40 px-3 py-2 text-sm outline-none transition focus:border-[#6BFFFF] ${form.formState.errors.entityType ? "border-red-400/60" : "border-white/15"}`}
                >
                    <option value="">Seleciona</option>
                    <option value="PROMOTOR_ORGANIZADOR">Promotor / Organizador</option>
                    <option value="EMPRESA_MARCA">Empresa ou marca</option>
                    <option value="OUTRO">Outro tipo de organização</option>
                </select>
            )}
        />
        {form.formState.errors.entityType && (
            <p className="text-[12px] text-red-300">{form.formState.errors.entityType.message}</p>
        )}
    </div>
<div className="space-y-2">
    <label className="text-[12px] text-white/70">Nome da organização *</label>
    <Controller
        name="businessName"
        control={form.control}
        render={({ field }) => (
            <input
                {...field}
                className={`w-full rounded-xl border bg-black/40 px-3 py-2 text-sm outline-none transition focus:border-[#6BFFFF] ${form.formState.errors.businessName ? "border-red-400/60" : "border-white/15"}`}
                placeholder="Nome da organização"
            />
        )}
    />
    {form.formState.errors.businessName && (
        <p className="text-[12px] text-red-300">{form.formState.errors.businessName.message}</p>
    )}
</div>
<div className="space-y-2">
    <div className="flex items-center gap-1 text-[12px] text-white/75">
        <span>Username ORYA *</span>
        <InfoTooltip text="Este será o @ da tua marca na ORYA. Vai aparecer no teu perfil, nos eventos e nos links públicos." />
    </div>
    <Controller
        name="username"
        control={form.control}
        render={({ field }) => (
            <div className="relative">
                <span className="pointer-events-none absolute left-1/2 top-1/2 -translate-y-1/2 text-sm text-white/50">
                    @
                </span>
                <input
                    {...field}
                    value={field.value || ""}
                    onChange={(e) => {
                        const cleaned = sanitizeUsername(e.target.value);
                        field.onChange(cleaned);
                    }}
                />
            </div>
        )}
    />
</div>

```

```

        const validation = validateUsername(cleaned);
        setUsernameHelper(validation.valid ? USERNAME_HELPER : validation.error);
        setUsernameStatus("idle");
    )}
onBlur={(e) => checkUsername(e.target.value)}
className={`w-full rounded-xl border bg-black/40 px-3 py-2 pl-7 text-sm outline-none transition
${usernameBorderClass}`}
maxLength={30}
placeholder="0 teu username"
/>
</div>
)}
/>
<p className={`text-[11px] leading-relaxed ${usernameMessageClass}`}>{usernameHelper}</p>
{usernameSuggestions.length > 0 && (
<div className="text-[11px] text-white/65">
    Sugestões:" "
    {usernameSuggestions.map((sug, idx) =>
        <button
            key={sug}
            type="button"
            onClick={() => form.setValue("username", sanitizeUsername(sug), { shouldValidate: true })}
            className="rounded-full border border-white/10 bg-white/5 px-2 py-1 text-[11px] text-white/80 transition
hover:border-white/25 hover:bg-white/10"
        >
            @{sug}
            {idx < usernameSuggestions.length - 1 ? " " : ""}
        </button>
    )))
</div>
)}
{form.formState.errors.username && (
<p className="text-[12px] text-red-300">{form.formState.errors.username.message}</p>
)}
</div>

<div className="space-y-2">
    <label className="text-[12px] text-white/70">Cidade base *</label>
    <Controller
        name="city"
        control={form.control}
        render={({ field }) => (
            <input
                {...field}
                className={`w-full rounded-xl border bg-black/40 px-3 py-2 text-sm outline-none transition focus:border-
[#6BFFFF] ${(
                    form.formState.errors.city ? "border-red-400/60" : "border-white/15"
                )}`}
                placeholder="Cidade base"
            />
        ))}
    />
    {form.formState.errors.city && (
        <p className="text-[12px] text-red-300">{form.formState.errors.city.message}</p>
    )}
</div>

<div className="space-y-2">
    <label className="text-[12px] text-white/70">Website ou Instagram (opcional)</label>
    <Controller
        name="website"
        control={form.control}
        render={({ field }) => (
            <input
                {...field}
                className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none transition
focus:border-[#6BFFFF]"
                placeholder="ex: orya.pt ou @orya.app"
                autoCapitalize="none"
                onChange={(e) => field.onChange(e.target.value)}
            />
        ))}
    />
    {form.formState.errors.website && (
        <p className="text-[12px] text-red-300">{form.formState.errors.website.message}</p>
    )}
</div>
</div>

```

```


<div className="space-y-2">
    <div className="flex items-center gap-1">
      <p className="text-[11px] uppercase tracking-[0.2em] text-white/60">Payouts (opcional)</p>
      <InfoTooltip text="Usamos este IBAN para enviar os pagamentos dos teus eventos. Podes adicionar ou alterar mais tarde nas Definições." />
    </div>
    <h3 className="text-lg font-medium">Prepara os pagamentos</h3>
    <p className="text-[12px] text-white/65">
      Liga os teus dados de pagamento para começar a receber o dinheiro dos teus eventos. Se preferires, podes completar esta parte mais tarde nas Definições.
    </p>
  </div>

  <div className="space-y-2">
    <label className="text-[12px] text-white/70">IBAN para pagamentos (opcional)</label>
    <Controller
      name="iban"
      control={form.control}
      render={({ field }) => (
        <input
          {...field}
          value={field.value || ""}
          onChange={(e) => {
            const raw = e.target.value.replace(/\s+/g, "").toUpperCase();
            const withSpaces = raw.replace(/(\.{4})/g, "$1 ").trim();
            field.onChange(withSpaces);
          }}
          className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none transition
focus:border-[#6BFFF]" uppercase
          placeholder="PT50 0000 0000 0000 0000 0000 0000 0"
        />
      )}
    />
    {form.formState.errors.iban && (
      <p className="text-[12px] text-red-300">{form.formState.errors.iban.message}</p>
    )}
  </div>

  <div className="space-y-2">
    <label className="text-[12px] text-white/70">NIF para faturação (opcional)</label>
    <Controller
      name="taxId"
      control={form.control}
      render={({ field }) => (
        <input
          {...field}
          value={field.value || ""}
          inputMode="numeric"
          maxLength={9}
          onChange={(e) => {
            const digits = e.target.value.replace(/\D/g, "").slice(0, 9);
            field.onChange(digits);
          }}
          className="w-full rounded-xl border border-white/15 bg-black/40 px-3 py-2 text-sm outline-none transition
focus:border-[#6BFFF]""
          placeholder="123456789"
        />
      )}
    />
    {form.formState.errors.taxId && (
      <p className="text-[12px] text-red-300">{form.formState.errors.taxId.message}</p>
    )}
  </div>
</form>
</div>

/* Coluna DIREITA - benefícios, preview e CTA */
<div className="order-2 space-y-8 md:order-2 md:pl-10 lg:pl-12 md:pt-1">
  <div className="space-y-6 text-sm text-white/80">
    <div className="space-y-1.5">
      <p className="text-[12px] uppercase tracking-[0.22em] text-white/60">0 que ganhas</p>
      <h3 className="text-xl font-semibold">0 que ganhas ao criar a tua organização</h3>
    </div>
  </div>
</div>


```

```

{{{
  {
    title: "Vendas & pagamentos",
    desc: "Vende bilhetes online, vê as vendas em tempo real e liga facilmente os teus payouts.",
    icon: "💻",
  },
  {
    title: "Equipa & acessos",
    desc: "Convida staff para gerir eventos, check-in e finanças com diferentes níveis de acesso.",
    icon: "👤",
  },
  {
    title: "Controlo & transparéncia",
    desc: "Acompanha receitas, reembolsos e estatísticas de cada evento num só painel.",
    icon: "📊",
  },
},
].map((item) => (
  <div
    key={item.title}
    className="group flex gap-3 rounded-2xl border border-white/10 bg-white/5 px-5 py-4 transition-all duration-200 hover:-translate-y-[1px] hover:border-white/20 hover:shadow-[0_14px_45px_rgba(0,0,0,0.45)]"
  >
    <div className="mt-0.5 text-lg transition-transform duration-150 group-hover:scale-105">
      {item.icon}
    </div>
    <div className="space-y-1">
      <p className="font-semibold text-white">{item.title}</p>
      <p className="leading-relaxed text-white/70">{item.desc}</p>
    </div>
  </div>
))
</div>

<div className="space-y-2 max-w-[65ch]">
  <div className="rounded-2xl border border-white/10 bg-white/5 p-4 text-sm text-white/75 shadow-[0_12px_40px_rgba(0,0,0,0.35)]">
    Funciona para qualquer organização que cria eventos ou experiências – clubes, associações, marcas, equipas de desporto, projectos independentes e muito mais.
  </div>
  <p className="text-[12px] text-white/45">
    Podes alterar todos os dados da organização mais tarde nas Definições.
  </p>
</div>
</div>

<div className="space-y-3 rounded-2xl border border-white/8 bg-white/5 p-5 text-sm text-white/85 shadow-[0_14px_45px_rgba(0,0,0,0.35)]">
  <div className="flex items-center gap-3">
    <div
      className={`flex h-11 w-11 items-center justify-center rounded-full ${avatarColor} font-semibold text-black`}
    >
      {avatarInitials}
    </div>
    <div className="space-y-0.5">
      <p className="text-base font-semibold text-white">
        {watchBusinessName || "Nome da tua organização"}
      </p>
      <p className="text-[12px] text-white/70">{@usernameClean || "teuusername"}</p>
      <p className="text-[12px] text-white/60">
        {watchCity || "Cidade"} .{" "}
        {watchEntityType
          ? watchEntityType === "PROMOTOR_ORGANIZADOR"
          ? "Promotor / Organizador"
          : watchEntityType === "EMPRESA_MARCA"
          ? "Empresa ou marca"
          : "Outro tipo de organização"
          : "Tipo de entidade"}
      </p>
      <p className="text-[12px] text-white/55">orya.pt/{usernameClean || "teuusername"}</p>
    </div>
  </div>
</div>

<div className="space-y-3 rounded-2xl border border-white/10 bg-white/5/20 p-4 text-sm text-white/75 shadow-[0_10px_35px_rgba(0,0,0,0.35)] md:flex md:items-center md:justify-between md:gap-4 md:space-y-0">
  <p className="text-[12px] text-white/70">
    A seguir: criar o teu primeiro evento · convidar a tua equipa · ativar pagamentos
  </p>

```

```


<Link
    href="/organizador/organizations"
    className="text-sm text-white/75 underline-offset-4 transition hover:text-white hover:underline"
  >
  Já tens uma organização? Ver lista
</Link>
<button
  form={formId}
  type="submit"
  disabled={!isValid}
  className={`${w-full rounded-full bg-gradient-to-r from-[#FF00C8] via-[#6BFFFF] to-[#1646F5] px-5 py-2.5 text-sm font-semibold text-black shadow transition focus:outline-none focus:ring-2 focus:ring-[#6BFFFF]/50 md:w-auto ${
    isValid
      ? "hover:brightness-110 shadow-[0_0_30px_rgba(107,255,255,0.22)] animate-[pulse_2.8s_ease-in-out_infinite]"
      : "opacity-60"
    }`}
  >
  {saving ? "A criar organização..." : "Começar a organizar"}
</button>
</div>
</div>
</div>
</div>
</div>
);
}


```

docs/cleanup_report.md

```

# Cleanup Report (ronda 1)

## DELETE ✅ (0 hits comprovado)

- app/api/internal/notifications/process/route.ts
  - comando: `rg -n --hidden --glob '!.git' "internal/notifications/process|notifications/process"`
  - resultado: 0 hits
  - ação: `git rm app/api/internal/notifications/process/route.ts`

- app/api/internal/notifications/tournament-eve/route.ts
  - comando: `rg -n --hidden --glob '!.git' "tournament-eve|internal/notifications/tournament-eve"`
  - resultado: 0 hits
  - ação: `git rm app/api/internal/notifications/tournament-eve/route.ts`

- app/api/social/following/route.ts
  - comando: `rg -n --hidden --glob '!.git' "/api/social/following\b|social/following\b"`
  - resultado: 0 hits
  - ação: `git rm app/api/social/following/route.ts`

## KEEP ⚠️ (tem referências)

- app/api/checkout/route.ts (já removido em commit anterior)
  - comando: `rg -n --hidden --glob '!.git' "/api/checkout\b|api/checkout\b"`
  - hits: `app/components/checkout/Step3Sucesso.tsx` (status polling), `app/api/checkout/resale/route.ts` logs.
  - decisão: manter referências atuais (rota /api/checkout/status e revenda ativas); nada removido aqui.

- app/api/notifications/route.ts
  - comando: `rg -n --hidden --glob '!.git' "/api/notifications\b|api/notifications\b"`
  - hits: `/app/me/edit`, `/app/me/settings`, `/app/components/notifications/NotificationBell.tsx`.
  - decisão: KEEP (rota em uso).

- app/api/padel/matches/generate/route.ts
  - comando: `rg -n --hidden --glob '!.git' "padel/matches/generate|maches/generate"`
  - hit: `app/organizador/(dashboard)/eventos/[id]/PadelTournamentSection.tsx`.
  - decisão: KEEP (rota em uso).

- app/api/social/follow-status/route.ts
  - comando: `rg -n --hidden --glob '!.git' "/api/social/follow-status\b|social/follow-status\b"`
  - hit: `app/[username]/FollowClient.tsx`.
  - decisão: KEEP.

## REFATORAR ❌ (SSOT / duplicações)

- paymentScenario
  - hits: webhook Stripe e /api/payments/intent + padel checkout local.
  - ação mínima desta ronda: criado wrapper SSOT `lib/payments/paymentScenario.ts` a reexportar o helper atual (no-op). Refactor

```

completo do webhook fica para ronda seguinte.

- Webhook Stripe
 - comando: `rg -n --hidden --glob '!.git' 'ORYA PATCH|patch|legacy' app/api/stripe/webhook/route.ts`
 - hit: linha 229 (`ORYA PATCH v1`).
 - decisão: REFATORAR numa ronda futura (separar handlers por cenário, sem tocar agora).

Gates

- lint: `npm run lint` ****FALHOU**** – 84 errors / 114 warnings (muitos `any`/hooks); não alterado nesta ronda.
- typecheck: `npm run typecheck` ****FALHOU**** – erro em `app/organizador/dashboard/tournaments/[id]/live/page.tsx` (string mal fechada).
- build: `npm run build` ****FALHOU**** – mesmo parse error (`"use client\";`) + falta de dependência `seedrandom` em `domain/tournaments/generation.ts` e `standings.ts`.

docs/dev-branches.md

Branches e ambientes

- `main` → produção (porto seguro, auth estável). Deploy de produção em Vercel.
- `developer` → dev contínuo com novas áreas (admin, checkout, organizador/Stripe, home) mas lógica de auth da `main`. Deploy de preview/staging em Vercel.

Fluxo de trabalho

- Desenvolvimento normal: `git checkout developer` → commits → `git push origin developer` (gera preview ligado à DB DEV).
- Release: `git checkout main` → `git merge developer` (ou PR) → `git push origin main` (deploy produção).
- Hotfix prod: `git checkout main`, corrigir, `git push origin main`; depois `git checkout developer` → `git merge main` → `git push origin developer`.
- Se a `developer` ficar estragada: `git checkout developer` → `git reset --hard main` → `git push -f origin developer` (só se necessário).

Supabase

Recomendado: projeto separado para DEV (ex.: `orya-dev`) com schema clonado da produção (`orya_supabase_schema.sql` ou migrations).

Ambiente PROD (main) usa o projeto atual.

Ambiente DEV (developer) usa o projeto novo.

Variáveis a definir em Vercel:

- PROD (main): `SUPABASE_URL`, `SUPABASE_ANON_KEY`, `SUPABASE_SERVICE_ROLE`, `NEXT_PUBLIC_SUPABASE_URL`, `NEXT_PUBLIC_SUPABASE_ANON_KEY`.
- DEV/previews (developer): mesmos nomes mas apontados ao projeto DEV.

Stripe / Resend / webhooks

- Guardar chaves separadas para DEV (Stripe test mode) e PROD.
- Configurar webhooks de Stripe/Resend para o domínio de preview (developer) com as chaves de DEV, e para o domínio prod (main) com chaves de PROD.

Deploys em Vercel

- Production branch: `main` → domínio prod (ex.: `https://orya.pt`).
- Preview branches (incl. `developer`): URLs `https://<project>-git-<branch>-*.vercel.app` → usar envs DEV.

docs/legacy_map.md

Legacy / cleanup map (estado atual)

- Removidos/sem referências ativas:
 - Roles antigos ('CHECKIN_ONLY', enums desatualizados) – não há refs em código.
 - `organizers.user_id` para auth – não usado; fallback apenas comentado como legacy (não usar).
 - `event_sales_agg` – não usado em código; reporting usa `sale_summaries`/`sale_lines`.
- Mantidos como LEGACY (read-only) para compatibilidade:
 - `promo_redemptions` – fonte de verdade é `sale_summaries`/`sale_lines` com snapshots; previsto remover se não houver dependências escondidas.
 - Boas práticas:
 - Qualquer rota/ficheiro que dependa de algo legacy deve trazer comentário `// TODO legacy – remover quando ...` indicando a condição.

docs/organizer_dashboard.md

Organizer dashboard (mapa rápido)

- Secções esperadas: Informações do evento, Bilhetes & preços, Participantes/Inscrições, Vendas & relatórios, Staff & permissões.
- Cartões/listagens de eventos devem mostrar: nome, data, estado (draft/published/running/ended) e ações principais.
- Feedback: todos os POST/PUT/DELETE devem devolver toast de sucesso/erro com mensagem específica (escopo mínimo desta fase; redesign completo fica fora de scope).
- Erros de API: normalizar para `{ code, message }` e expor mensagens claras (evitar “Algo correu mal” genérico).
- Navegação: usar tabs/section keys consistentes entre servidor e cliente.

docs/padel_v2_plan.md

Padel v2 (duplas, FULL/SPLIT) – implementação faseada

Escopo e flags

- Aplicar apenas se `padel_v2_enabled` no `PadelTournamentConfig`; legacy intocado.
- Slug do torneio e checkout atual permanecem.

Schema novo (Prisma)

- Organizer: `refundFeePayer` (CUSTOMER/ORGANIZER).
- PadelTournamentConfig: `padelV2Enabled`, `splitDeadlineHours`, `autoCancelUnpaid`, `allowCaptainAssume`, `defaultPaymentMode`, `refundFeePayer?`.
- Novos modelos: `PadelPairing`, `PadelPairingSlot` com enums `PadelPaymentMode`, `PadelPairingStatus`, `PadelPairingSlotStatus`, `PadelPairingPaymentStatus`, `PadelPairingSlotRole`.
- Ticket: campos opcionais `pairingId`, `pairingSlotId`, `padelSplitShareCents`, `padelPairingVersion`.

Endpoints previstos (stubs criados)

- POST `/api/padel/pairings` – criar pairing/checkout (guardado por flag).
- GET/POST `/api/padel/pairings/[token]/claim` – preview/claim do convite.
- POST `/api/padel/pairings/[id]/assume` – capitão assume restante (stub).
- POST `/api/padel/pairings/[id]/cancel` – cancel/refund simples (stub).
- PATCH `/api/padel/pairings/[id]/public` – toggles para dupla aberta (stub).

Taxas e invoice (para checkout)

- Se `fee_mode=ADDED`: mostrar subtotal bilhetes + linha “Taxas de serviço (ORYA+Stripe)” + total.
- Se `fee_mode=INCLUDED`: ocultar linha de taxas no cliente; guardar breakdown interno.
- Reembolsos: por defeito taxas não devolvidas ao cliente; `refund_fee_payer` define se organizer absorve custos.

Reembolsos (MVP)

- SPLIT expirado e incompleto: refund ao capitão do bilhete dele (taxas conforme `refund_fee_payer`), pairing CANCELLED, capacidade libertada.
- FULL: cancelamento aplica refund dos bilhetes, pairing CANCELLED; transferências continuam para reatribuir parceiro.
- Sempre atualizar tickets->REFUNDED, pairing/slots status e logs.

UX base

- Pré-checkout Padel: escolha FULL vs SPLIT.
- Checkout: invoice com linhas por categoriaxqty, subtotal, taxas (se ADDED), total.
- Pós-checkout FULL: link para convidar parceiro (slot PENDING/PAID).
- Pós-checkout SPLIT: mostra prazo `locked_until` para parceiro pagar e CTA convite.
- “As minhas duplas”: estados COMPLETA/INCOMPLETA/CANCELADA, prazo, CTAs Convidar, Assumir resto, Cancelar.

Próximos passos imediatos

- Gerar migrations (Prisma) para o novo schema.
- Implementar lógica real nos endpoints e integrar com checkout/webhooks.
- UI: toggles FULL/SPLIT no fluxo Padel e invoice discriminado no resumo.

docs/performance_rules.md

Regras rápidas de performance

- Todas as listas usam paginação ou cursor. Default take/limit: 50 (até 200 em listas de membros/convites onde é aceitável).
- Nunca devolver tudo sem limite por defeito (eventos, bilhetes, participantes, sale_lines, invites/members).
- Evitar `include` gigante; preferir `select` do necessário.
- Índices críticos confirmados: `events.organizer_id`, `tickets.event_id`, `sale_lines.event_id`, `sale_lines.sale_summary_id`, `organizer_members.organizer_id`, `organizer_members.user_id`.
- Preferir SWR/SSR com revalidação em páginas grandes; evitar múltiplos fetches duplicados.

docs/qa_checklist.md

QA checklist (estática)

- Auth + roles:
 - Organizer core usa auth + helpers (`organizerPermissions`) - ex.: `app/api/organizador/organizations/members/route.ts`, `.../invites/route.ts`, `.../events/list|create|update`, `.../payouts/*`, `.../finance/overview`.
 - Admin endpoints validam role admin (`app/api/admin/utilizadores/list`, `app/api/admin/eventos/list`, `app/api/admin/payments/list`).
- Validação:
 - Organizador/events create/update com validação manual de campos obrigatórios e fees; outros usam checks equivalentes (sem zod) - indicado para futura melhoria.
 - Payloads críticos (convites, transfer, leave) têm checks de types/roles; bodies parseados com guardas.
- Invariante:
 - Regra “nunca ficar sem OWNER” centralizada nos endpoints de members (PATCH/DELETE/transfer) e transação de promoção a OWNER despromove restantes.
- Fonte de verdade dinheiro/notifs:
 - Sales/reporting: `sale_summaries`/`sale_lines` (snapshots de promo); event_sales_agg legado.
 - Notificações via `createNotification`; enums finalizados; endpoints list/mark usam `isRead`.
- RLS:
 - organizers via organizer_members; tickets/sale_summaries/sale_lines/notifications/guest_ticket_links com policies ativas; bypass service_role aplicado.
- Perfis apagados:
 - Delete handler libera username e marca is_deleted/deleted_at.
 - Resolvers ignoram isDeleted; listas follows/following/members/invites sanitizam perfis apagados/privados (`lib/profileVisibility`).
- Observações:
 - Sem QA manual; esta checklist é estática. Melhorias futuras: zod/safeParse em todos os POST/PUT e alinhamento de mensagens de erro.

docs/release-checklist.md

Release checklist ORYA (organizador + público)

- Explorar
 - Carrega a página `/explorar`
 - Aplica filtro de preço (mover ambos os handles)
 - Aplica filtro de datas (Hoje/Próximos dias)
 - Aplica filtro de localização (cidade da whitelist)
 - Verifica se o empty state surge quando não há resultados e se “Limpar filtros” devolve resultados
- Criação de evento simples
 - Cria um evento básico (gratuito) e confirma que aparece em `/explorar`
 - Cria evento pago e valida que bloqueia publicação se o organizer não tiver Stripe ligado
- Stripe Connect / Finanças
 - Abre tab Finanças
 - Clica “Ligar conta Stripe” (ou “Rever ligação” se incompleto)
 - Conclui onboarding → estado “Ativo” na tab
 - Se houver requirements pendentes, verifica o callout e CTA “Rever ligação”
- Checkout / promo codes
 - Cria código em Marketing > Códigos
 - Vai a um evento pago, aplica código no checkout e verifica desconto
 - Para evento gratuito, confirma que o checkout salta Stripe e cria bilhete/reserva
- Navegação organizador
 - Sidebar ativa uma tab de cada vez
 - Dropdown Categorias abre/fecha e mantém active state correto
 - BackButton volta ao expected path/tab
- Padel (se aplicável)
 - Criar/editar clube, courts, staff
 - Wizard padel respeita nº máximo de courts ativos e só mostra clubes/courts ativos
- Delete/Arquivo
 - Dashboard eventos: DRAFT apaga com confirmação; restantes arquivam e somem de /explorar
 - Clubes/courts: confirmação antes de arquivar/desativar/reactivar; courts inativos não aparecem em sugestões
- Notificações
 - Definições `/me/settings`: toggles de email/reminders/friend requests/vendas/Stripe/anúncios do sistema guardam prefs
 - Campanha mostra tabs (Todas/Vendas/Convites/Sistema/Social) e badge de não lidas
- Smoke responsável
 - DevTools mobile 320-414px: filtros/popovers viram modal, sem scroll horizontal

Executa testes automáticos antes de deploy:

- Unit (node test runner): `node --test tests/filters.test.mjs`
- e2e (se configurado/Playwright/Cypress): `npm run test:e2e` (quando disponível)***

docs/rls_policies.sql

```
-- RLS baseline for organizer-owned data (Supabase / Postgres).
-- Apply with psql against your Supabase DB (schema app_v3).
```

```

-- Organizer members: a user only sees/edits memberships of orgs where they are a member.
ALTER TABLE app_v3.organizer_members ENABLE ROW LEVEL SECURITY;
DROP POLICY IF EXISTS organizer_members_select ON app_v3.organizer_members;
CREATE POLICY organizer_members_select ON app_v3.organizer_members
    FOR SELECT USING (EXISTS (SELECT 1 FROM app_v3.organizer_members m2 WHERE m2.organizer_id = organizer_id AND m2.user_id =
auth.uid()));
DROP POLICY IF EXISTS organizer_members_modify ON app_v3.organizer_members;
CREATE POLICY organizer_members_modify ON app_v3.organizer_members
    FOR ALL USING (EXISTS (SELECT 1 FROM app_v3.organizer_members m2 WHERE m2.organizer_id = organizer_id AND m2.user_id =
auth.uid()));

-- Organizer member invites: visible if you are a manager of that org OR you are the invite target.
ALTER TABLE app_v3.organizer_member_invites ENABLE ROW LEVEL SECURITY;
DROP POLICY IF EXISTS organizer_member_invites_select ON app_v3.organizer_member_invites;
CREATE POLICY organizer_member_invites_select ON app_v3.organizer_member_invites
    FOR SELECT USING (
        EXISTS (SELECT 1 FROM app_v3.organizer_members m2 WHERE m2.organizer_id = organizer_id AND m2.user_id = auth.uid()
        OR target_user_id = auth.uid())
    );
DROP POLICY IF EXISTS organizer_member_invites_modify ON app_v3.organizer_member_invites;
CREATE POLICY organizer_member_invites_modify ON app_v3.organizer_member_invites
    FOR ALL USING (EXISTS (SELECT 1 FROM app_v3.organizer_members m2 WHERE m2.organizer_id = organizer_id AND m2.user_id =
auth.uid()));

-- Notifications: owner only.
ALTER TABLE app_v3.notifications ENABLE ROW LEVEL SECURITY;
DROP POLICY IF EXISTS notifications_owner_only ON app_v3.notifications;
CREATE POLICY notifications_owner_only ON app_v3.notifications
    FOR ALL USING (user_id = auth.uid());
DROP POLICY IF EXISTS notifications_service_role ON app_v3.notifications;
CREATE POLICY notifications_service_role ON app_v3.notifications
    FOR ALL USING (current_setting('request.jwt.claim.role', true) = 'service_role');

-- Guest ticket links: owner only (used for migrate guest -> user).
ALTER TABLE app_v3.guest_ticket_links ENABLE ROW LEVEL SECURITY;
DROP POLICY IF EXISTS guest_ticket_links_owner_only ON app_v3.guest_ticket_links;
CREATE POLICY guest_ticket_links_owner_only ON app_v3.guest_ticket_links
    FOR ALL USING (guest_email IS NOT NULL AND auth.jwt() ->> 'email' = guest_email);
DROP POLICY IF EXISTS guest_ticket_links_service_role ON app_v3.guest_ticket_links;
CREATE POLICY guest_ticket_links_service_role ON app_v3.guest_ticket_links
    FOR ALL USING (current_setting('request.jwt.claim.role', true) = 'service_role');

-- Payout settings: only members of the organizer.
-- NOTE: organizers.id is int; auth.uid() is uuid. Cast uid to uuid to match organizer_members.user_id.
ALTER TABLE app_v3.organizers ENABLE ROW LEVEL SECURITY;
DROP POLICY IF EXISTS organizers_select_member ON app_v3.organizers;
CREATE POLICY organizers_select_member ON app_v3.organizers
    FOR SELECT USING (
        EXISTS (
            SELECT 1 FROM app_v3.organizer_members m
            WHERE m.organizer_id = app_v3.organizers.id
            AND m.user_id = auth.uid()::uuid
        )
    );
DROP POLICY IF EXISTS organizers_update_member ON app_v3.organizers;
CREATE POLICY organizers_update_member ON app_v3.organizers
    FOR UPDATE USING (
        EXISTS (
            SELECT 1 FROM app_v3.organizer_members m
            WHERE m.organizer_id = app_v3.organizers.id
            AND m.user_id = auth.uid()::uuid
        )
    );

-- Tickets (optional hardening): user can see own tickets.
ALTER TABLE app_v3.tickets ENABLE ROW LEVEL SECURITY;
DROP POLICY IF EXISTS tickets_owner_or_guest ON app_v3.tickets;
CREATE POLICY tickets_owner_or_guest ON app_v3.tickets
    FOR SELECT USING (user_id = auth.uid());
DROP POLICY IF EXISTS tickets_service_role ON app_v3.tickets;
CREATE POLICY tickets_service_role ON app_v3.tickets
    FOR SELECT USING (current_setting('request.jwt.claim.role', true) = 'service_role');

-- Sale summaries/lines (optional hardening): only by organizer members or buyer.
ALTER TABLE app_v3.sale_summaries ENABLE ROW LEVEL SECURITY;
DROP POLICY IF EXISTS sale_summaries_view ON app_v3.sale_summaries;
CREATE POLICY sale_summaries_view ON app_v3.sale_summaries
    FOR SELECT USING (

```

```

user_id = auth.uid()
OR EXISTS (
  SELECT 1
  FROM app_v3.events e
  JOIN app_v3.organizer_members m ON m.organizer_id = e.organizer_id
  WHERE e.id = event_id
  AND m.user_id = auth.uid()
)
);
DROP POLICY IF EXISTS sale_summaries_service_role ON app_v3.sale_summaries;
CREATE POLICY sale_summaries_service_role ON app_v3.sale_summaries
FOR SELECT USING (current_setting('request.jwt.claim.role', true) = 'service_role');

ALTER TABLE app_v3.sale_lines ENABLE ROW LEVEL SECURITY;
DROP POLICY IF EXISTS sale_lines_view ON app_v3.sale_lines;
CREATE POLICY sale_lines_view ON app_v3.sale_lines
FOR SELECT USING (
EXISTS (
  SELECT 1 FROM app_v3.sale_summaries s
  WHERE s.id = sale_summary_id
  AND (
    s.user_id = auth.uid()
    OR EXISTS (
      SELECT 1
      FROM app_v3.events e
      JOIN app_v3.organizer_members m ON m.organizer_id = e.organizer_id
      WHERE e.id = event_id
      AND m.user_id = auth.uid()
    )
  )
)
);
DROP POLICY IF EXISTS sale_lines_service_role ON app_v3.sale_lines;
CREATE POLICY sale_lines_service_role ON app_v3.sale_lines
FOR SELECT USING (current_setting('request.jwt.claim.role', true) = 'service_role');

-- NOTE: adjust or drop optional policies if they conflict with analytics/reporting needs.

```

domain/finance/disputes.ts

```

import { prisma } from "@/lib/prisma";
import { PaymentEventSource, SaleSummaryStatus } from "@prisma/client";

export async function markSaleDisputed(params: { saleSummaryId: number; paymentIntentId?: string | null; purchaseId?: string | null; reason?: string | null }) {
  const { saleSummaryId, paymentIntentId, purchaseId, reason } = params;
  return prisma.$transaction(async (tx) => {
    const sale = await tx.saleSummary.update({
      where: { id: saleSummaryId },
      data: { status: SaleSummaryStatus.DISPUTED },
    });

    await tx.paymentEvent.create({
      data: {
        stripePaymentIntentId: paymentIntentId ?? sale.paymentIntentId,
        status: "DISPUTED",
        purchaseId: purchaseId ?? sale.purchaseId ?? undefined,
        source: PaymentEventSource.WEBHOOK,
        errorMessage: reason ?? "Dispute received",
      },
    });
    return sale;
  });
}

```

domain/finance/payoutPolicy.ts

```

export function computeReleaseAt(eventEndsAt: Date | null) {
  if (!eventEndsAt) return null;
  const release = new Date(eventEndsAt.getTime() + 72 * 60 * 60 * 1000);
  return release;
}

export function computeHold(amountCents: number, hasDisputes: boolean) {

```

```
    if (hasDisputes) return { holdCents: amountCents, reason: "DISPUTES" };
    return { holdCents: 0, reason: null };
}
```

domain/finance/stripeConnectStatus.ts

```
export type ConnectStatus = "READY" | "INCOMPLETE" | "MISSING";

export function resolveConnectStatus(stripeAccountId?: string | null, chargesEnabled?: boolean | null, payoutsEnabled?: boolean | null): ConnectStatus {
  if (!stripeAccountId) return "MISSING";
  if (!chargesEnabled || !payoutsEnabled) return "INCOMPLETE";
  return "READY";
}
```

domain/notifications/matchChangeDedupe.ts

```
import crypto from "crypto";

export function computeDedupeKey(matchId: number, startAt: Date | null, courtId: number | null) {
  const payload = `${matchId}|${startAt ? startAt.toISOString() : "null"}|${courtId ?? "null"}`;
  return crypto.createHash("sha256").update(payload).digest("hex");
}
```

domain/notifications/outbox.ts

```
import { prisma } from "@lib/prisma";
import type { Prisma } from "@prisma/client";

export type OutboxStatus = "PENDING" | "SENT" | "FAILED";

type EnqueueParams = {
  dedupeKey: string;
  userId?: string | null;
  notificationType: string;
  templateVersion?: string | null;
  payload?: Prisma.JsonValue;
  force?: boolean;
};

/**
 * Enqueue a notification with dedupe guarantees.
 * If the dedupeKey already exists, it returns the existing record unless force=true,
 * in which case it resets status to PENDING and updates payload/templateVersion.
 */
export async function enqueueNotification(params: EnqueueParams) {
  const {
    dedupeKey,
    userId,
    notificationType,
    templateVersion,
    payload = {},
    force = false,
  } = params;

  const existing = await prisma.notificationOutbox.findUnique({
    where: { dedupeKey },
  });
  if (existing && !force) {
    return existing;
  }

  return prisma.notificationOutbox.upsert({
    where: { dedupeKey },
    create: {
      dedupeKey,
      userId: userId ?? null,
      notificationType,
      templateVersion: templateVersion ?? null,
      payload,
      status: "PENDING",
    },
  });
}
```

```

update: {
  userId: userId ?? null,
  notificationType,
  templateVersion: templateVersion ?? null,
  payload,
  status: "PENDING",
  retries: 0,
  lastError: null,
  sentAt: null,
},
};

}

export async function markOutboxSent(id: string) {
  return prisma.notificationOutbox.update({
    where: { id },
    data: { status: "SENT", sentAt: new Date(), lastError: null },
  });
}

export async function markOutboxFailed(id: string, error: string) {
  return prisma.notificationOutbox.update({
    where: { id },
    data: { status: "FAILED", lastError: error, retries: { increment: 1 } },
  });
}

```

domain/notifications/producer.ts

```

import { enqueueNotification } from "@/domain/notifications/outbox";
import type { NotificationTemplate } from "@/domain/notifications/types";

type CommonArgs = {
  userId: string;
  templateVersion?: string;
  payload?: Record<string, unknown>;
};

function buildDedupe(prefix: string, parts: Array<string | number | null | undefined>) {
  return [prefix, ...parts.map((p) => (p === null || p === undefined ? "null" : String(p)))].join(":");
}

async function queue(
  type: NotificationTemplate,
  dedupeKey: string,
  args: CommonArgs & { force?: boolean },
) {
  return enqueueNotification({
    dedupeKey,
    notificationType: type,
    templateVersion: args.templateVersion ?? "v1",
    userId: args.userId,
    payload: args.payload ?? {},
    force: args.force ?? false,
  });
}

export async function notifyPairingInvite(params: {
  pairingId: number;
  tournamentId?: number;
  targetUserId: string;
  inviterUserId?: string;
  token?: string;
}) {
  const dedupeKey = buildDedupe("PAIRING_INVITE", [params.pairingId, params.targetUserId]);
  return queue("PAIRING_INVITE", dedupeKey, {
    userId: params.targetUserId,
    payload: {
      pairingId: params.pairingId,
      tournamentId: params.tournamentId,
      inviterUserId: params.inviterUserId,
      token: params.token,
    },
  });
}

```

```

export async function notifyPairingReminder(params: { pairingId: number; targetUserId: string }) {
  const dedupeKey = buildDedupe("PAIRING_RemINDER", [params.pairingId, params.targetUserId]);
  return queue("PAIRING_RemINDER", dedupeKey, {
    userId: params.targetUserId,
    payload: { pairingId: params.pairingId },
    templateVersion: "v1",
  });
}

export async function notifyPartnerPaid(params: {
  pairingId: number;
  captainUserId: string;
  partnerUserId?: string;
}) {
  const dedupeKey = buildDedupe("PARTNER_PAID", [params.pairingId, params.captainUserId]);
  return queue("PARTNER_PAID", dedupeKey, {
    userId: params.captainUserId,
    payload: { pairingId: params.pairingId, partnerUserId: params.partnerUserId },
  });
}

export async function notifyDeadlineExpired(params: { pairingId: number; userId: string }) {
  const dedupeKey = buildDedupe("DEADLINE_EXPIRED", [params.pairingId, params.userId]);
  return queue("DEADLINE_EXPIRED", dedupeKey, {
    userId: params.userId,
    payload: { pairingId: params.pairingId },
  });
}

export async function notifyOffsessionActionRequired(params: { pairingId: number; userId: string }) {
  const dedupeKey = buildDedupe("OFFSESSION_ACTION_REQUIRED", [params.pairingId, params.userId]);
  return queue("OFFSESSION_ACTION_REQUIRED", dedupeKey, {
    userId: params.userId,
    payload: { pairingId: params.pairingId },
  });
}

export async function notifyNewFollower(params: { targetUserId: string; followerUserId: string }) {
  const dedupeKey = buildDedupe("NEW_FOLLOWER", [params.targetUserId, params.followerUserId]);
  return queue("NEW_FOLLOWER", dedupeKey, {
    userId: params.targetUserId,
    payload: { followerUserId: params.followerUserId },
  });
}

export async function notifyPairingRequestReceived(params: { targetUserId: string; pairingId: number }) {
  const dedupeKey = buildDedupe("PAIRING_REQUEST_RECEIVED", [params.pairingId, params.targetUserId]);
  return queue("PAIRING_REQUEST_RECEIVED", dedupeKey, {
    userId: params.targetUserId,
    payload: { pairingId: params.pairingId },
  });
}

export async function notifyPairingRequestAccepted(params: { targetUserId: string; pairingId: number }) {
  const dedupeKey = buildDedupe("PAIRING_REQUEST_ACCEPTED", [params.pairingId, params.targetUserId]);
  return queue("PAIRING_REQUEST_ACCEPTED", dedupeKey, {
    userId: params.targetUserId,
    payload: { pairingId: params.pairingId },
  });
}

export async function notifyTicketWaitingClaim(params: { userId: string; ticketId: string }) {
  const dedupeKey = buildDedupe("TICKET_WAITING CLAIM", [params.ticketId, params.userId]);
  return queue("TICKET_WAITING CLAIM", dedupeKey, {
    userId: params.userId,
    payload: { ticketId: params.ticketId },
  });
}

export async function notifyBracketPublished(params: { userId: string; tournamentId: number }) {
  const dedupeKey = buildDedupe("BRACKET_PUBLISHED", [params.tournamentId, params.userId]);
  return queue("BRACKET_PUBLISHED", dedupeKey, {
    userId: params.userId,
    payload: { tournamentId: params.tournamentId },
  });
}

export async function notifyTournamentEve(params: { userId: string; tournamentId: number }) {

```

```

const dedupeKey = buildDedupe("TOURNAMENT_EVE_REMINDER", [params.tournamentId, params.userId]);
return queue("TOURNAMENT_EVE_REMINDER", dedupeKey, {
  userId: params.userId,
  payload: { tournamentId: params.tournamentId },
});
}

export async function notifyMatchResult(params: { userId: string; matchId: number; tournamentId?: number }) {
  const dedupeKey = buildDedupe("MATCH_RESULT", [params.matchId, params.userId]);
  return queue("MATCH_RESULT", dedupeKey, {
    userId: params.userId,
    payload: { matchId: params.matchId, tournamentId: params.tournamentId },
  });
}

export async function notifyNextOpponent(params: { userId: string; matchId: number; tournamentId?: number }) {
  const dedupeKey = buildDedupe("NEXT OPPONENT", [params.matchId, params.userId]);
  return queue("NEXT OPPONENT", dedupeKey, {
    userId: params.userId,
    payload: { matchId: params.matchId, tournamentId: params.tournamentId },
  });
}

export async function notifyMatchChanged(params: {
  userId: string;
  matchId: number;
  startAt?: Date | null;
  courtId?: number | null;
}) {
  const dedupeKey = buildDedupe("MATCH_CHANGED", [
    params.matchId,
    params.startAt ? params.startAt.toISOString() : null,
    params.courtId ?? null,
  ]);
  return queue("MATCH_CHANGED", dedupeKey, {
    userId: params.userId,
    payload: { matchId: params.matchId, startAt: params.startAt ?? null, courtId: params.courtId ?? null },
  });
}

export async function notifyEliminated(params: { userId: string; tournamentId: number }) {
  const dedupeKey = buildDedupe("ELIMINATED", [params.tournamentId, params.userId]);
  return queue("ELIMINATED", dedupeKey, {
    userId: params.userId,
    payload: { tournamentId: params.tournamentId },
  });
}

export async function notifyChampion(params: { userId: string; tournamentId: number }) {
  const dedupeKey = buildDedupe("CHAMPION", [params.tournamentId, params.userId]);
  return queue("CHAMPION", dedupeKey, {
    userId: params.userId,
    payload: { tournamentId: params.tournamentId },
  });
}

export async function notifyBroadcast(params: {
  tournamentId: number;
  userId: string;
  broadcastId: string;
  audienceKey: string;
}) {
  const dedupeKey = buildDedupe("BROADCAST", [params.tournamentId, params.audienceKey, params.userId]);
  return queue("BROADCAST", dedupeKey, {
    userId: params.userId,
    payload: { tournamentId: params.tournamentId, broadcastId: params.broadcastId },
  });
}

```

domain/notifications/splitPayments.ts

```

import {
  notifyDeadlineExpired,
  notifyOffsessionActionRequired,
  notifyPairingInvite,
  notifyPairingReminder,

```

```

    notifyPartnerPaid,
} from "@domain/notifications/producer";

export async function queuePairingInvite(params: {
  pairingId: number;
  tournamentId?: number;
  targetUserId: string;
  inviterUserId?: string;
  token?: string;
}) {
  return notifyPairingInvite(params);
}

export async function queuePairingReminder(pairingId: number, targetUserId: string) {
  return notifyPairingReminder({ pairingId, targetUserId });
}

export async function queuePartnerPaid(pairingId: number, captainUserId: string, partnerUserId?: string) {
  return notifyPartnerPaid({ pairingId, captainUserId, partnerUserId });
}

export async function queueDeadlineExpired(pairingId: number, userIds: string[]) {
  await Promise.all(userIds.map((userId) => notifyDeadlineExpired({ pairingId, userId })));
}

export async function queueOffsessionActionRequired(pairingId: number, userIds: string[]) {
  await Promise.all(userIds.map((userId) => notifyOffsessionActionRequired({ pairingId, userId })));
}

```

domain/notifications/tournament.ts

```

import {
  notifyBracketPublished,
  notifyChampion,
  notifyEliminated,
  notifyMatchChanged,
  notifyMatchResult,
  notifyNextOpponent,
  notifyTournamentEve,
  notifyBroadcast,
} from "@domain/notifications/producer";
import { computeDedupKey as dedupeMatchChange } from "@domain/notifications/matchChangeDedup";

export async function queueBracketPublished(userIds: string[], tournamentId: number) {
  await Promise.all(userIds.map((userId) => notifyBracketPublished({ userId, tournamentId })));
}

export async function queueTournamentEve(userIds: string[], tournamentId: number) {
  await Promise.all(userIds.map((userId) => notifyTournamentEve({ userId, tournamentId })));
}

export async function queueMatchResult(userIds: string[], matchId: number, tournamentId?: number) {
  await Promise.all(userIds.map((userId) => notifyMatchResult({ userId, matchId, tournamentId })));
}

export async function queueNextOpponent(userIds: string[], matchId: number, tournamentId?: number) {
  await Promise.all(userIds.map((userId) => notifyNextOpponent({ userId, matchId, tournamentId })));
}

export async function queueMatchChanged(params: {
  userIds: string[];
  matchId: number;
  startAt?: Date | null;
  courtId?: number | null;
}) {
  const { userIds, matchId, startAt = null, courtId = null } = params;
  // Use the same dedupe hash as scheduling dedupe so we never send twice for identical change.
  const dedupeKey = dedupeMatchChange(matchId, startAt, courtId);
  await Promise.all(
    userIds.map((userId) =>
      notifyMatchChanged({
        userId,
        matchId,
        startAt,
        courtId,
        // force so the shared dedupeKey applies across recipients
      })
    )
  );
}

```

```

        },
    ),
);
return dedupeKey;
}

export async function queueEliminated(userIds: string[], tournamentId: number) {
    await Promise.all(userIds.map((userId) => notifyEliminated({ userId, tournamentId })));
}

export async function queueChampion(userIds: string[], tournamentId: number) {
    await Promise.all(userIds.map((userId) => notifyChampion({ userId, tournamentId })));
}

export async function queueBroadcast(params: {
    audienceUserIds: string[];
    tournamentId: number;
    broadcastId: string;
    audienceKey: string;
}) {
    const { audienceUserIds, tournamentId, broadcastId, audienceKey } = params;
    await Promise.all(
        audienceUserIds.map((userId) =>
            notifyBroadcast({ userId, tournamentId, broadcastId, audienceKey }),
        ),
    );
}
}

```

domain/notifications/types.ts

```

export type NotificationTemplate =
    | "PAIRING_INVITE"
    | "PAIRING_REMINDER"
    | "PARTNER_PAID"
    | "DEADLINE_EXPIRED"
    | "OFFSESSION_ACTION_REQUIRED"
    | "NEW_FOLLOWER"
    | "PAIRING_REQUEST_RECEIVED"
    | "PAIRING_REQUEST_ACCEPTED"
    | "TICKET_WAITING_CLAIM"
    | "BRACKET_PUBLISHED"
    | "TOURNAMENT_EVE_RemINDER"
    | "MATCH_RESULT"
    | "NEXT OPPONENT"
    | "MATCH_CHANGED"
    | "ELIMINATED"
    | "CHAMPION"
    | "BROADCAST";

export type NotificationPayload = Record<string, unknown>;

export type DedupeKey = string;

```

domain/padel/pairingPolicy.ts

```

import { PadelPairingLifecycleStatus } from "@prisma/client";

export function canSwapPartner(lifecycleStatus: PadelPairingLifecycleStatus, now: Date, partnerSwapAllowedUntilAt?: Date | null) {
    if (!partnerSwapAllowedUntilAt) return false;
    if (lifecycleStatus?.toString().startsWith("CONFIRMED")) return false;
    return now.getTime() <= partnerSwapAllowedUntilAt.getTime();
}

export function canMarkWalkover(status: PadelPairingLifecycleStatus) {
    return status === "CONFIRMED_BOTH_PAID" || status === "CONFIRMED_CAPTAIN_FULL";
}

```

domain/padelDeadlines.ts

```

const MIN_DEADLINE_HOURS = 48;
const MAX_DEADLINE_HOURS = 168;

```

```

const MIN_LINK_MINUTES = 15;
const MAX_LINK_MINUTES = 30;
const DEFAULT_LINK_MINUTES = 30;
const GRACE_HOURS = 24;

export function clampDeadlineHours(raw?: number | null): number {
  const base = typeof raw === "number" && !Number.isNaN(raw) ? raw : MIN_DEADLINE_HOURS;
  return Math.min(Math.max(base, MIN_DEADLINE_HOURS), MAX_DEADLINE_HOURS);
}

export function computeDeadlineAt(now: Date, splitDeadlineHours?: number | null): Date {
  const hours = clampDeadlineHours(splitDeadlineHours);
  return new Date(now.getTime() + hours * 60 * 60 * 1000);
}

export function clampLinkMinutes(raw?: number | null): number {
  const base = typeof raw === "number" && !Number.isNaN(raw) ? raw : DEFAULT_LINK_MINUTES;
  return Math.min(Math.max(base, MIN_LINK_MINUTES), MAX_LINK_MINUTES);
}

export function computePartnerLinkExpiresAt(now: Date, minutes?: number | null): Date {
  const mins = clampLinkMinutes(minutes);
  return new Date(now.getTime() + mins * 60 * 1000);
}

export function computeGraceUntil(now: Date): Date {
  return new Date(now.getTime() + GRACE_HOURS * 60 * 60 * 1000);
}

```

domain/padelEligibility.ts

```

import { Gender, PadelEligibilityType } from "@prisma/client";

export type EligibilityResult =
  | { ok: true }
  | { ok: false; code: "GENDER_REQUIRED_FOR_TOURNAMENT" | "INELIGIBLE_FOR_TOURNAMENT" };

export function validateEligibility(
  eligibilityType: PadelEligibilityType,
  player1Gender: Gender | null | undefined,
  player2Gender?: Gender | null,
): EligibilityResult {
  if (eligibilityType === "OPEN") return { ok: true };

  const p1 = player1Gender ?? null;
  const p2 = player2Gender ?? null;

  if (!p1) return { ok: false, code: "GENDER_REQUIRED_FOR_TOURNAMENT" };
  if (eligibilityType === "MALE_ONLY") {
    return p1 === "MALE" && (!p2 || p2 === "MALE")
      ? { ok: true }
      : { ok: false, code: "INELIGIBLE_FOR_TOURNAMENT" };
  }
  if (eligibilityType === "FEMALE_ONLY") {
    return p1 === "FEMALE" && (!p2 || p2 === "FEMALE")
      ? { ok: true }
      : { ok: false, code: "INELIGIBLE_FOR_TOURNAMENT" };
  }

  // MIXED
  if (!p2) return { ok: true }; // pode criar/entrar à procura; valida no fecho
  const comboOk =
    (p1 === "MALE" && p2 === "FEMALE") || (p1 === "FEMALE" && p2 === "MALE");
  return comboOk ? { ok: true } : { ok: false, code: "INELIGIBLE_FOR_TOURNAMENT" };
}

```

domain/padelPairingHold.ts

```

import { Prisma, PadelPairingHoldStatus } from "@prisma/client";

type TxClient = Prisma.TransactionClient;

const DEFAULT_HOLD_MINUTES = 30;

```

```

export async function upsertActiveHold(
  tx: TxClient,
  params: { pairingId: number; eventId: number; ttlMinutes?: number },
) {
  const { pairingId, eventId, ttlMinutes } = params;
  const minutes = ttlMinutes && ttlMinutes > 0 ? ttlMinutes : DEFAULT_HOLD_MINUTES;
  const expiresAt = new Date(Date.now() + minutes * 60 * 1000);

  await tx.padelPairingHold.upsert({
    where: { pairingId_status: { pairingId, status: PadelPairingHoldStatus.ACTIVE } },
    update: { expiresAt },
    create: {
      pairingId,
      eventId,
      holds: 2,
      status: PadelPairingHoldStatus.ACTIVE,
      expiresAt,
    },
  });
}

return { expiresAt };
}

export async function cancelActiveHold(tx: TxClient, pairingId: number) {
  await tx.padelPairingHold.updateMany({
    where: { pairingId, status: PadelPairingHoldStatus.ACTIVE },
    data: { status: PadelPairingHoldStatus.CANCELLED },
  });
}

export async function expireHolds(tx: TxClient, now: Date) {
  await tx.padelPairingHold.updateMany({
    where: { status: PadelPairingHoldStatus.ACTIVE, expiresAt: { lt: now } },
    data: { status: PadelPairingHoldStatus.EXPIRED },
  });
}

```

domain/padelPairingStateMachine.ts

```

import {
  PadelPairingGuaranteeStatus,
  PadelPairingLifecycleStatus,
} from "@prisma/client";

export type PairingAction =
  | "CAPTAIN_PAID"
  | "PARTNER_ASSIGNED"
  | "PARTNER_PAID"
  | "CAPTAIN_FULL_CONFIRMED"
  | "CANCEL";

export type GuaranteeAction =
  | "ARM"
  | "SCHEDULE_CHARGE"
  | "CHARGE_SUCCEEDED"
  | "CHARGE_FAILED"
  | "REQUIRES_ACTION"
  | "EXPIRE"
  | "CANCEL_GUARANTEE";

const terminalStatuses: PadelPairingLifecycleStatus[] = [
  "CONFIRMED_BOTH_PAID",
  "CONFIRMED_CAPTAIN_FULL",
  "CANCELLED_INCOMPLETE",
];

const transitionMap: Record<
  PadelPairingLifecycleStatus,
  Partial<Record<PairingAction, PadelPairingLifecycleStatus>>
> = {
  PENDING_ONE_PAID: {
    CAPTAIN_PAID: "PENDING_PARTNER_PAYMENT",
    CAPTAIN_FULL_CONFIRMED: "CONFIRMED_CAPTAIN_FULL",
    CANCEL: "CANCELLED_INCOMPLETE",
  },
  PENDING_PARTNER_PAYMENT: {
    PARTNER_PAID: "PENDING_CAPTAIN_PAYMENT",
    EXPIRE: "EXPIRED",
  },
};

```

```

    PARTNER_ASSIGNED: "PENDING_PARTNER_PAYMENT",
    PARTNER_PAID: "CONFIRMED_BOTH_PAID",
    CAPTAIN_FULL_CONFIRMED: "CONFIRMED_CAPTAIN_FULL",
    CANCEL: "CANCELLED_INCOMPLETE",
  },
  CONFIRMED_BOTH_PAID: {},
  CONFIRMED_CAPTAIN_FULL: {},
  CANCELLED_INCOMPLETE: {},
};

export function isTerminal(status: PadelPairingLifecycleStatus) {
  return terminalStatuses.includes(status);
}

export function canTransition(
  from: PadelPairingLifecycleStatus,
  to: PadelPairingLifecycleStatus,
): boolean {
  if (from === to) return true;
  if (isTerminal(from)) return false;
  const actions = transitionMap[from] || {};
  return Object.values(actions).includes(to);
}

export function transition(
  current: PadelPairingLifecycleStatus,
  action: PairingAction,
): PadelPairingLifecycleStatus {
  if (isTerminal(current) && action !== "CANCEL") {
    return current;
  }
  const next = transitionMap[current]?.[action];
  return next ?? current;
}

// Guarantee (Modelo A) state machine
const guaranteeMap: Record<
  PadelPairingGuaranteeStatus,
  Partial<Record<GuaranteeAction, PadelPairingGuaranteeStatus>>
> = {
  NONE: {
    ARM: "ARMED",
  },
  ARMED: {
    SCHEDULE_CHARGE: "SCHEDULED",
    CANCEL_GUARANTEE: "NONE",
  },
  SCHEDULED: {
    CHARGE_SUCCEEDED: "SUCCEEDED",
    CHARGE_FAILED: "FAILED",
    REQUIRES_ACTION: "REQUIRES_ACTION",
    EXPIRE: "EXPIRED",
  },
  REQUIRES_ACTION: {
    CHARGE_SUCCEEDED: "SUCCEEDED",
    CHARGE_FAILED: "FAILED",
    EXPIRE: "EXPIRED",
  },
  FAILED: {
    REQUIRES_ACTION: "REQUIRES_ACTION",
  },
  SUCCEEDED: {},
  EXPIRED: {},
};

export function transitionGuarantee(
  current: PadelPairingGuaranteeStatus,
  action: GuaranteeAction,
): PadelPairingGuaranteeStatus {
  const next = guaranteeMap[current]?.[action];
  return next ?? current;
}

export function computeGraceUntil(
  nextGuaranteeStatus: PadelPairingGuaranteeStatus,
  now: Date = new Date(),
): Date | null {
  if (nextGuaranteeStatus === "REQUIRES_ACTION") {

```

```

        return new Date(now.getTime() + 24 * 60 * 60 * 1000);
    }
    return null;
}

```

domain/tournaments/ensureEntriesForConfirmedPairing.ts

```

import { prisma } from "@/lib/prisma";
import { TournamentEntryStatus, TournamentEntryRole } from "@prisma/client";

export async function ensureEntriesForConfirmedPairing(pairingId: number) {
    const pairing = await prisma.padelPairing.findUnique({
        where: { id: pairingId },
        select: {
            id: true,
            eventId: true,
            player1UserId: true,
            player2UserId: true,
        },
    });
    if (!pairing) return;

    const entriesData: Array<{ userId: string; role: TournamentEntryRole }> = [];
    if (pairing.player1UserId) entriesData.push({ userId: pairing.player1UserId, role: "CAPTAIN" });
    if (pairing.player2UserId) entriesData.push({ userId: pairing.player2UserId, role: "PARTNER" });

    if (!entriesData.length) return;

    const entryIdsByUser: Record<string, number> = {};

    for (const entry of entriesData) {
        const upserted = await prisma.tournamentEntry.upsert({
            where: {
                eventId_userId: { eventId: pairing.eventId, userId: entry.userId },
            },
            update: {
                status: TournamentEntryStatus.CONFIRMED,
                role: entry.role,
                pairingId: pairing.id,
                ownerUserId: entry.userId,
                ownerIdentityId: null,
            },
            create: {
                eventId: pairing.eventId,
                userId: entry.userId,
                pairingId: pairing.id,
                role: entry.role,
                status: TournamentEntryStatus.CONFIRMED,
                ownerUserId: entry.userId,
                ownerIdentityId: null,
            },
        });
        entryIdsByUser[entry.userId] = upserted.id;
    }

    // Linkar tickets existentes (se já criados) ao tournament_entry
    const ticketUpdates = Object.entries(entryIdsByUser).map(([userId, entryId]) =>
        prisma.ticket.updateMany({
            where: { pairingId: pairing.id, userId, tournamentEntryId: null },
            data: { tournamentEntryId: entryId },
        }),
    );
    if (ticketUpdates.length) {
        await Promise.all(ticketUpdates);
    }
}

```

domain/tournaments/generation.ts

```

import seedrandom from "seedrandom";
import { prisma } from "@/lib/prisma";
import { Prisma, TournamentFormat, TournamentMatchStatus, TournamentStageType } from "@prisma/client";

type PairingId = number;

```

```

export type RoundRobinMatch = { a: PairingId; b: PairingId };
export type RoundRobinSchedule = RoundRobinMatch[][];

export function generateRoundRobin(pairings: PairingId[], seed?: string): RoundRobinSchedule {
  const rng = seedrandom(seed || `${Date.now()}`);
  const players = [...pairings];
  if (players.length % 2 !== 0) players.push(-1); // bye = -1
  const n = players.length;
  const rounds: RoundRobinSchedule = [];

  const arr = [...players];
  for (let round = 0; round < n - 1; round += 1) {
    const matches: RoundRobinMatch[] = [];
    for (let i = 0; i < n / 2; i += 1) {
      const home = arr[i];
      const away = arr[n - 1 - i];
      if (home === -1 && away === -1) {
        // shuffle home/away to vary
        const swap = rng() > 0.5;
        matches.push({ a: swap ? away : home, b: swap ? home : away });
      }
    }
    rounds.push(matches);
    // rotate array except first element
    const fixed = arr[0];
    const rest = arr.slice(1);
    rest.unshift(rest.pop() as number);
    arr.splice(0, arr.length, fixed, ...rest);
  }
  return rounds;
}

export type EliminationMatch = { a?: PairingId; b?: PairingId };
export type EliminationBracket = EliminationMatch[][];

export function generateSingleElimination(pairings: PairingId[], seed?: string): EliminationBracket {
  const rng = seedrandom(seed || `${Date.now()}`);
  const shuffled = [...pairings].sort(() => (rng() > 0.5 ? 1 : -1));
  // next power of two
  const size = 1 << Math.ceil(Math.log2(shuffled.length || 1));
  while (shuffled.length < size) shuffled.push(undefined as unknown as PairingId);

  const rounds: EliminationBracket = [];
  let current = shuffled;
  while (current.length > 1) {
    const matches: EliminationMatch[] = [];
    for (let i = 0; i < current.length; i += 2) {
      const a = current[i];
      const b = current[i + 1];
      matches.push({ a, b });
    }
    rounds.push(matches);
    current = matches.map((_m, idx) => idx as unknown as PairingId); // placeholders for next round seeds
  }
  return rounds;
}

export type ABBBracket = {
  main: EliminationBracket;
  consolation: EliminationBracket;
};

export function generateDrawAB(pairings: PairingId[], seed?: string): ABBBracket {
  // main bracket normal; consolation fed by losers (handled by engine later)
  const main = generateSingleElimination(pairings, seed);
  const consolation: EliminationBracket = [];
  return { main, consolation };
}

const CONFIRMED_PAIRING_STATUSES = ["CONFIRMED_BOTH_PAID", "CONFIRMED_CAPTAIN_FULL"] as const;

export async function getConfirmedPairings(eventId: number) {
  const pairings = await prisma.padelPairing.findMany({
    where: {
      eventId,
      lifecycleStatus: { in: CONFIRMED_PAIRING_STATUSES as unknown as Prisma.PadelPairingWhereInput["lifecycleStatus"] },
    },
  });
}

```

```

        select: { id: true },
        orderBy: { id: "asc" },
      });
      return pairings.map((p) => p.id);
    }

  type PersistOptions = {
    tournamentId: number;
    format: TournamentFormat;
    pairings: number[];
    seed?: string | null;
    inscriptionDeadlineAt?: Date | null;
    forceGenerate?: boolean;
    userId?: string;
  };
}

export async function generateAndPersistTournamentStructure(opts: PersistOptions) {
  const { tournamentId, format, pairings, seed, inscriptionDeadlineAt, forceGenerate, userId } = opts;
  const rngSeed = seed || `${Date.now()}`;
  const confirmed = pairings.filter((id) => typeof id === "number");

  return prisma.$transaction(async (tx) => {
    // Deadline: se ainda não passou e não foi forçado, bloqueia
    if (inscriptionDeadlineAt && new Date() < new Date(inscriptionDeadlineAt) && !forceGenerate) {
      throw new Error("INSCRIPTION_NOT_CLOSED");
    }

    const started = await tx.tournamentMatch.count({
      where: { stage: { tournamentId }, status: { in: ["IN_PROGRESS", "DONE", "SCHEDULED"] as TournamentMatchStatus[] } },
    });
    if (started > 0) throw new Error("TOURNAMENT_ALREADY_STARTED");

    await tx.tournamentMatch.deleteMany({ where: { stage: { tournamentId } } });
    await tx.tournamentGroup.deleteMany({ where: { stage: { tournamentId } } });
    await tx.tournamentStage.deleteMany({ where: { tournamentId } });

    if (confirmed.length === 0) return { stagesCreated: 0, matchesCreated: 0, seed: rngSeed };

    let stagesCreated = 0;
    let matchesCreated = 0;

    const createRoundRobin = async (stageName: string, groupName: string, order: number) => {
      const stage = await tx.tournamentStage.create({
        data: { tournamentId, name: stageName, stageType: TournamentStageType.GROUPS, order },
      });
      stagesCreated += 1;
      const group = await tx.tournamentGroup.create({
        data: { stageId: stage.id, name: groupName, order: 1 },
      });
      const rr = generateRoundRobin(confirmed, rngSeed);
      for (let r = 0; r < rr.length; r += 1) {
        for (const m of rr[r]) {
          await tx.tournamentMatch.create({
            data: {
              stageId: stage.id,
              groupId: group.id,
              pairing1Id: m.a,
              pairing2Id: m.b,
              round: r + 1,
              status: TournamentMatchStatus.PENDING,
            },
          });
          matchesCreated += 1;
        }
      }
    };

    const createBracket = async (stageName: string, bracket: EliminationBracket, order: number) => {
      const stage = await tx.tournamentStage.create({
        data: { tournamentId, name: stageName, stageType: TournamentStageType.PLAYOFF, order },
      });
      stagesCreated += 1;
      for (let r = 0; r < bracket.length; r += 1) {
        for (const m of bracket[r]) {
          await tx.tournamentMatch.create({
            data: {
              stageId: stage.id,
              pairing1Id: m.a,
            },
          });
        }
      }
    };
  });
}

```

```

        pairing2Id: m.b,
        round: r + 1,
        status: TournamentMatchStatus.PENDING,
    },
});
matchesCreated += 1;
}
}
return stage.id;
};

const createClassificationFromBracket = async (sourceBracket: EliminationBracket, order: number) => {
    const stage = await tx.tournamentStage.create({
        data: { tournamentId, name: "Classificação", stageType: TournamentStageType.CONSOLATION, order },
    });
    stagesCreated += 1;
    // Cria jogos de classificação por round (exceto final), placeholders a preencher após resultados
    for (let r = 0; r < sourceBracket.length; r += 1) {
        const matchesInRound = sourceBracket[r];
        if (matchesInRound.length < 2) continue;
        const classificationCount = Math.floor(matchesInRound.length / 2);
        for (let i = 0; i < classificationCount; i += 1) {
            await tx.tournamentMatch.create({
                data: {
                    stageId: stage.id,
                    round: r + 1,
                    roundLabel: `Classificação R${r + 1}`,
                    status: TournamentMatchStatus.PENDING,
                },
            });
            matchesCreated += 1;
        }
    }
};

// Consolation placeholder: cria stage e matches entre perdedores de primeira ronda
const createConsolationFromBracket = async (sourceStageId: number, order: number) => {
    const stage = await tx.tournamentStage.create({
        data: { tournamentId, name: "Consolação", stageType: TournamentStageType.CONSOLATION, order },
    });
    stagesCreated += 1;
    // Busca jogos da primeira ronda do stage fonte
    const firstRound = await tx.tournamentMatch.findMany({
        where: { stageId: sourceStageId, round: 1 },
        orderBy: { id: "asc" },
    });
    // Pares de derrotados (placeholder: pairing1/2 losers → um jogo)
    let roundNum = 1;
    for (let i = 0; i < firstRound.length; i += 2) {
        const m1 = firstRound[i];
        const m2 = firstRound[i + 1];
        if (!m1 || !m2) continue;
        await tx.tournamentMatch.create({
            data: {
                stageId: stage.id,
                pairing1Id: m1.pairing1Id ?? m1.pairing2Id ?? null,
                pairing2Id: m2.pairing1Id ?? m2.pairing2Id ?? null,
                round: roundNum,
                status: TournamentMatchStatus.PENDING,
            },
        });
        matchesCreated += 1;
        roundNum += 1;
    }
};

if (format === "GROUPS_PLUS_PLAYOFF" || format === "CHAMPIONSHIP_ROUND_ROBIN" || format === "NONSTOP_ROUND_ROBIN") {
    await createRoundRobin("Fase de Grupos", "Grupo Único", 1);
    if (format === "GROUPS_PLUS_PLAYOFF" && confirmed.length > 2) {
        const bracket = generateSingleElimination(confirmed, rngSeed);
        const playoffStageId = await createBracket("Playoff", bracket, 2);
        // Consolação automática dos derrotados da ronda 1
        await createConsolationFromBracket(playoffStageId, 3);
    }
} else if (format === "DRAW_A_B") {
    const bracket = generateSingleElimination(confirmed, rngSeed);
    const mainStageId = await createBracket("Quadro Principal", bracket, 1);
    // Consolação (Quadro B) a partir dos derrotados da ronda 1
}

```

```

    await createConsolationFromBracket(mainStageId, 2);
} else if (format === "GROUPS_PLUS_FINALS_ALL_PLACES") {
    await createRoundRobin("Fase de Grupos", "Grupo Único", 1);
    const finalsBracket = generateSingleElimination(confirmed, rngSeed);
    await createBracket("Finais por posições", finalsBracket, 2);
    await createClassificationFromBracket(finalsBracket, 3);
} else if (format === "MANUAL") {
    await tx.tournamentStage.create({
        data: { tournamentId, name: "Manual", stageType: TournamentStageType.PLAYOFF, order: 1 },
    });
    stagesCreated += 1;
}

await tx.tournament.update({
    where: { id: tournamentId },
    data: { generationSeed: rngSeed, updatedAt: new Date(), generatedAt: new Date(), generatedByUserId: userId || null },
});

// Audit log da geração
if (userId) {
    await tx.tournamentAuditLog.create({
        data: {
            tournamentId,
            userId,
            action: "GENERATE_BRACKET",
            payloadBefore: Prisma.DbNull,
            payloadAfter: { format, seed: rngSeed, pairings: confirmed },
        },
    });
}

return { stagesCreated, matchesCreated, seed: rngSeed };
});
}

```

domain/tournaments/liveWarnings.ts

```

import { Prisma } from "@prisma/client";
import { validateScore } from "@domain/tournaments/matchRules";

type Warning =
| { type: "REQUIRES_ACTION"; pairingId: number }
| { type: "INVALID_SCORE"; matchId: number }
| { type: "MISSING_WINNER"; matchId: number }
| { type: "MISSING_COURT"; matchId: number }
| { type: "MISSING_START"; matchId: number };

export function computeLiveWarnings(opts: {
    matches: Array<{ id: number; courtId: number | null; startAt: Date | null; score: any; status: string; winnerPairingId?: number | null }>;
    pairings: Array<{ id: number; guaranteeStatus?: string | null }>;
    startThresholdMinutes?: number;
}): Warning[] {
    const { matches, pairings, startThresholdMinutes = 60 } = opts;
    const warnings: Warning[] = [];

    // guarantee REQUIRES_ACTION
    pairings.forEach((p) => {
        if ((p.guaranteeStatus || "").toUpperCase() === "REQUIRES_ACTION") {
            warnings.push({ type: "REQUIRES_ACTION", pairingId: p.id });
        }
    });

    // matches: missing court/start + score inválido
    const threshold = new Date(Date.now()) + startThresholdMinutes * 60 * 1000;
    matches.forEach((m) => {
        if (!m.courtId) warnings.push({ type: "MISSING_COURT", matchId: m.id });
        if (!m.startAt || m.startAt < threshold) warnings.push({ type: "MISSING_START", matchId: m.id });
        if ((m.status || "").toUpperCase() === "DONE") {
            const sets = Array.isArray(m.score?.sets) ? m.score.sets : [];
            const res = validateScore({ sets } as any);
            if (!res.ok) warnings.push({ type: "INVALID_SCORE", matchId: m.id });
            if (!m.winnerPairingId) warnings.push({ type: "MISSING_WINNER", matchId: m.id });
        }
    });
}

```

```
    return warnings;
}
```

domain/tournaments/matchRules.ts

```
import { TournamentMatchStatus } from "@prisma/client";

export type SetScore = { a: number; b: number };
export type ScorePayload = { sets: SetScore[] };

export type ValidationResult =
  | { ok: true; winner: "A" | "B"; normalized: ScorePayload }
  | { ok: false; code: string; message: string };

const WIN_BY = 2;
const MAX_SETS = 3; // B03
const MIN_GAMES_TO_WIN = 6;
const TIEBREAK_GAMES = 7;

function isValidSet(set: SetScore) {
  const a = Number(set.a);
  const b = Number(set.b);
  if (!Number.isFinite(a) || !Number.isFinite(b) || a < 0 || b < 0) return false;
  const max = Math.max(a, b);
  const diff = Math.abs(a - b);
  // tiebreak 7-6 é aceite
  if (max === TIEBREAK_GAMES && diff === 1) return true;
  if (max >= MIN_GAMES_TO_WIN) return diff >= WIN_BY;
  return false;
}

export function validateScore(score: ScorePayload): ValidationResult {
  if (!score || !Array.isArray(score.sets) || score.sets.length === 0) {
    return { ok: false, code: "INVALID_SCORE", message: "Score vazio ou inválido." };
  }
  if (score.sets.length > MAX_SETS) {
    return { ok: false, code: "TOO_MANY_SETS", message: "Máximo de 3 sets (B03)." };
  }

  let winsA = 0;
  let winsB = 0;
  for (const s of score.sets) {
    if (!isValidSet(s)) return { ok: false, code: "INVALID_SET", message: "Set inválido." };
    if (s.a > s.b) winsA += 1;
    else winsB += 1;
  }

  if (winsA === winsB) return { ok: false, code: "NO_WINNER", message: "Empate não permitido no B03." };
  if (winsA > MAX_SETS || winsB > MAX_SETS) {
    return { ok: false, code: "TOO_MANY_WINS", message: "Vitórias a mais para B03." };
  }
  if (winsA > winsB && winsA > MAX_SETS - 1) {
    return { ok: true, winner: "A", normalized: score };
  }
  if (winsB > winsA && winsB > MAX_SETS - 1) {
    return { ok: true, winner: "B", normalized: score };
  }
  return { ok: false, code: "NO_WINNER", message: "Score não determina vencedor." };
}

export function canEditMatch(status: TournamentMatchStatus, force?: boolean) {
  if (force) return true;
  // não permite editar DONE se não for force
  return status !== "DONE";
}
```

domain/tournaments/matchUpdate.ts

```
import { prisma } from "@/lib/prisma";
import { Prisma, TournamentMatchStatus } from "@prisma/client";
import { validateScore, ScorePayload, canEditMatch } from "@domain/tournaments/matchRules";

type UpdateResultInput = {
  matchId: number;
}
```

```

score?: ScorePayload;
status?: TournamentMatchStatus;
explicitWinnerPairingId?: number | null;
expectedUpdatedAt?: Date | string | null;
userId?: string | null;
force?: boolean;
};

export async function updateMatchResult({
  matchId,
  score,
  status,
  explicitWinnerPairingId,
  expectedUpdatedAt,
  userId,
  force = false,
}: UpdateResultInput) {
  return prisma.$transaction(async (tx) => {
    const current = await tx.tournamentMatch.findUnique({
      where: { id: matchId },
      include: { stage: { select: { tournamentId: true, tournament: { select: { eventId: true } } } } },
    });
    if (!current) throw new Error("MATCH_NOT_FOUND");

    if (!canEditMatch(current.status, force)) {
      throw new Error("MATCH_LOCKED");
    }

    if (!expectedUpdatedAt) {
      throw new Error("MISSING_VERSION");
    }
    const expected = new Date(expectedUpdatedAt);
    if (current.updatedAt.getTime() !== expected.getTime()) {
      throw new Error("MATCH_CONFLICT");
    }

    let winnerSide: "A" | "B" | null = null;
    let normalizedScore: ScorePayload | undefined = undefined;
    if (score) {
      const validation = validateScore(score);
      if (!validation.ok && !force) {
        const err = validation;
        const error = err.code === "NO_WINNER" ? "INVALID_SCORE" : err.code;
        throw new Error(error);
      }
      if (validation.ok) {
        winnerSide = validation.winner;
        normalizedScore = validation.normalized;
      }
    }

    const winnerPairingId =
      explicitWinnerPairingId ||
      (winnerSide === "A" ? current.pairing1Id ?? null : winnerSide === "B" ? current.pairing2Id ?? null : null);
    const before = {
      status: current.status,
      score: current.score,
      pairing1Id: current.pairing1Id,
      pairing2Id: current.pairing2Id,
    };

    const的新状态: TournamentMatchStatus = status ?? "DONE";
    const updated = await tx.tournamentMatch.update({
      where: { id: matchId },
      data: {
        status: 新状态,
        score: normalizedScore ?? score ?? current.score,
        winnerPairingId: winnerPairingId ?? undefined,
      },
    });

    // Propagar vencedor para o próximo jogo se houver
    if (winnerPairingId && updated.nextMatchId && updated.nextSlot) {
      await tx.tournamentMatch.update({
        where: { id: updated.nextMatchId },
        data: updated.nextSlot === 1 ? { pairing1Id: winnerPairingId } : { pairing2Id: winnerPairingId },
      });
    }
  });
}

```

```

// Audit log
await tx.tournamentAuditLog.create({
  data: {
    tournamentId: current.stage.tournamentId,
    userId: userId ?? null,
    action: "EDIT_MATCH",
    payloadBefore: before,
    payloadAfter: {
      status: newStatus,
      score: score ?? current.score,
      propagated: Boolean(winner && updated.nextMatchId),
    },
  },
});

return updated;
});
}

```

domain/tournaments/schedulePolicy.ts

```

import { TournamentMatchStatus } from "@prisma/client";

export function canReschedule(status: TournamentMatchStatus, startAt: Date | null, newStart: Date | null) {
  if (status === "IN_PROGRESS" || status === "DONE") return false;
  if (!newStart) return true;
  const now = Date.now();
  if (newStart.getTime() < now) return false;
  return true;
}

export function canNotify(status: TournamentMatchStatus) {
  return status !== "DONE" && status !== "IN_PROGRESS";
}

```

domain/tournaments/standings.ts

```

import seedrandom from "seedrandom";
import { TournamentMatchStatus } from "@prisma/client";

export type TieBreakRule = "WINS" | "SET_DIFF" | "GAME_DIFF" | "HEAD_TO_HEAD" | "RANDOM";

export type MatchResult = {
  pairing1Id: number;
  pairing2Id: number;
  status: TournamentMatchStatus;
  score?: { sets?: Array<{ a: number; b: number }>; games?: Array<{ a: number; b: number }> };
};

export type Standing = {
  pairingId: number;
  wins: number;
  losses: number;
  setDiff: number;
  gameDiff: number;
  headToHead: Record<number, number>; // pairingId -> wins against
};

export function computeGroupStandings(
  pairings: number[],
  matches: MatchResult[],
  rules: TieBreakRule[],
  seed?: string,
): Standing[] {
  const rng = seedrandom(seed || `${Date.now()}`);
  const map = new Map<number, Standing>();
  pairings.forEach((p) =>
    map.set(p, { pairingId: p, wins: 0, losses: 0, setDiff: 0, gameDiff: 0, headToHead: {} }),
  );

  const finished = matches.filter((m) => m.status === "DONE");
  for (const m of finished) {
    const s1 = map.get(m.pairing1Id);
    const s2 = map.get(m.pairing2Id);
  }
}

```

```

const s2 = map.get(m.pairing2Id);
if (!s1 || !s2) continue;
const sets = m.score?.sets ?? [];
let aSets = 0;
let bSets = 0;
let aGames = 0;
let bGames = 0;
for (const set of sets) {
  aSets += set.a;
  bSets += set.b;
  aGames += set.a;
  bGames += set.b;
}
if (aSets > bSets) {
  s1.wins += 1;
  s2.losses += 1;
  s1.headToHead[m.pairing2Id] = (s1.headToHead[m.pairing2Id] ?? 0) + 1;
} else if (bSets > aSets) {
  s2.wins += 1;
  s1.losses += 1;
  s2.headToHead[m.pairing1Id] = (s2.headToHead[m.pairing1Id] ?? 0) + 1;
}
s1.setDiff += aSets - bSets;
s2.setDiff += bSets - aSets;
s1.gameDiff += aGames - bGames;
s2.gameDiff += bGames - aGames;
}

const standings = Array.from(map.values());

const comparator = (a: Standing, b: Standing) => {
  for (const rule of rules) {
    if (rule === "WINS") {
      if (a.wins !== b.wins) return b.wins - a.wins;
    } else if (rule === "SET_DIFF") {
      if (a.setDiff !== b.setDiff) return b.setDiff - a.setDiff;
    } else if (rule === "GAME_DIFF") {
      if (a.gameDiff !== b.gameDiff) return b.gameDiff - a.gameDiff;
    } else if (rule === "HEAD_TO_HEAD") {
      const aHH = a.headToHead[b.pairingId] ?? 0;
      const bHH = b.headToHead[a.pairingId] ?? 0;
      if (aHH !== bHH) return bHH - aHH;
    } else if (rule === "RANDOM") {
      const r = rng() - 0.5;
      if (r !== 0) return r > 0 ? 1 : -1;
    }
  }
  return 0;
};

return standings.sort(comparator);
}

```

domain/tournaments/structure.ts

```

import { prisma } from "@/lib/prisma";
import { TournamentMatchStatus } from "@prisma/client";
import { computeGroupStandings, TieBreakRule } from "@domain/tournaments/standings";

export async function getTournamentStructure(tournamentId: number) {
  return prisma.tournament.findUnique({
    where: { id: tournamentId },
    include: {
      stages: {
        orderBy: { order: "asc" },
        include: {
          groups: { orderBy: { order: "asc" }, include: { matches: true } },
          matches: true,
        },
      },
      event: { select: { id: true, title: true, slug: true, startsAt: true } },
    },
  });
}

export function summarizeMatchStatus(status: TournamentMatchStatus) {

```

```

if (status === "IN_PROGRESS") return "Em jogo";
if (status === "DONE") return "Terminado";
if (status === "SCHEDULED") return "Agendado";
if (status === "CANCELLED") return "Cancelado";
return "Pendente";
}

export function computeStandingsForGroup(
  matches: { pairing1Id: number | null; pairing2Id: number | null; status: TournamentMatchStatus; score: unknown }[],
  rules: TieBreakRule[],
  seed?: string,
) {
  const pairings = Array.from(
    new Set(
      matches
        .flatMap((m) => [m.pairing1Id, m.pairing2Id])
        .filter((v): v is number => typeof v === "number"),
    ),
  );
  if (pairings.length === 0) return [];
  return computeGroupStandings(
    pairings,
    matches
      .filter((m) => m.pairing1Id && m.pairing2Id)
      .map((m) => ({
        pairing1Id: m.pairing1Id as number,
        pairing2Id: m.pairing2Id as number,
        status: m.status,
        score: m.score as unknown,
      })),
    rules,
    seed,
  );
}

```

lib/accountEvents.ts

```

import { prisma } from "@/lib/prisma";

export async function logAccountEvent(params: {
  userId: string;
  type: "account_delete_requested" | "account_delete_cancelled" | "account_delete_completed" | "account_restored";
  metadata?: Record<string, unknown>;
}) {
  // Modelo legacy removido; fallback para log simples.
  console.info("[accountEvents] event", {
    userId: params.userId,
    type: params.type,
    metadata: params.metadata ?? {},
  });
}

```

lib/analytics.ts

```

export type AnalyticsPayload = Record<string, any>;

export function trackEvent(name: string, payload?: AnalyticsPayload) {
  if (!name) return;
  // Por agora, apenas console.log; preparado para PostHog/Amplitude/GA.
  // Mantém formato consistente para fácil troca futura.

  console.log("[trackEvent]", name, payload ?? {});
}

```

lib/checkoutSchemas.ts

```

import crypto from "crypto";
import { z } from "zod";
import { paymentScenarioSchema, type PaymentScenario } from "./paymentScenario";

const checkoutItemOutputSchema = z.object({
  ticketTypeId: z.number().int().positive(),

```

```

quantity: z.number().int().positive(),
// FE não é pricing engine: unitPriceCents/currency são ignorados pelo BE e ficam opcionais.
unitPriceCents: z.number().int().nonnegative().optional().default(0),
currency: z.string().trim().min(1).optional().default("EUR"),
});

export const checkoutItemSchema = z.preprocess((raw) => {
if (!raw || typeof raw === "object") return raw;
const v = raw as Record<string, unknown>;

// Compat: alguns flows antigos enviam `ticketId` em vez de `ticketTypeId`.
const ticketTypeId =
  typeof v.ticketTypeId === "number"
    ? v.ticketTypeId
    : typeof v.ticketId === "number"
      ? v.ticketId
      : v.ticketTypeId;

// Compat: unitPriceCents/currency podem faltar (pricing é do BE).
const unitPriceCents =
  typeof v.unitPriceCents === "number" && Number.isFinite(v.unitPriceCents)
    ? v.unitPriceCents
    : undefined;

const currencyRaw = typeof v.currency === "string" ? v.currency : undefined;

return {
  ...v,
  ticketTypeId,
  unitPriceCents,
  currency: currencyRaw,
};
}, checkoutItemOutputSchema);

export type NormalizedCheckoutItem = z.infer<typeof checkoutItemSchema>;

const legacyOwnerShape = z.object({
  userId: z.string().uuid().nullish(),
  guestEmail: z.string().email().trim().toLowerCase().nullish(),
  guestName: z.string().trim().min(1).nullish(),
  guestPhone: z.string().trim().nullish(),
});

const newOwnerShape = z.object({
  ownerUserId: z.string().uuid().nullish(),
  ownerIdentityId: z.string().uuid().nullish(),
  emailNormalized: z.string().email().trim().toLowerCase().nullish(),
});

export const checkoutOwnerSchema = newOwnerShape
  .merge(legacyOwnerShape.partial())
  .refine(
    (value) =>
      Boolean(
        value.ownerUserId ||
        value.ownerIdentityId ||
        value.userId ||
        value.guestEmail,
      ),
    {
      message:
        "Owner inválido: requer ownerUserId/ownerIdentityId ou userId/guestEmail.",
    },
  );

export type CheckoutOwner = z.infer<typeof checkoutOwnerSchema>;

export const purchaseIdSchema = z.union([
  z.string().uuid(),
  z
    .string()
    .trim()
    .regex(/^pur_[a-f0-9]{32}$/i, "Invalid purchaseId"),
]);

const checkoutMetadataOutputSchema = z.object({
  paymentScenario: paymentScenarioSchema,
  purchaseId: purchaseIdSchema,
});

```

```

items: z.array(checkoutItemSchema).min(1),
eventId: z.number().int().positive().optional(),
eventSlug: z.string().trim().min(1).optional(),
pairingId: z.number().int().positive().optional(),
owner: checkoutOwnerSchema.optional(),
});

export const checkoutMetadataSchema = z.preprocess((raw) => {
  if (!raw || typeof raw !== "object") return raw;
  const v = raw as Record<string, unknown>;
  // Compat: FE pode enviar `slug` em vez de `eventSlug`.
  const slug = typeof v.slug === "string" ? v.slug : undefined;
  const eventSlug = typeof v.eventSlug === "string" ? v.eventSlug : undefined;

  return {
    ...v,
    eventSlug: eventSlug ?? slug,
  };
}, checkoutMetadataOutputSchema);

export type CheckoutMetadata = z.infer<typeof checkoutMetadataSchema>

export function createPurchaseId() {
  // Default to the canonical purchaseId format used across the checkout core,
  // 16 random bytes => 32 hex chars.
  return `pur_${crypto.randomBytes(16).toString("hex")}`;
}

export function parseCheckoutItems(raw: unknown): NormalizedCheckoutItem[] {
  const parsed = z.array(checkoutItemSchema).safeParse(raw);
  if (parsed.success) return parsed.data;
  return [];
}

export function normalizeItemsForMetadata(items: NormalizedCheckoutItem[]): NormalizedCheckoutItem[] {
  return items.map((item) => ({
    ticketTypeId: Number((item as any).ticketTypeId),
    quantity: Number(item.quantity),
    unitPriceCents: Number(item.unitPriceCents ?? 0),
    currency: String(item.currency || "EUR").toUpperCase(),
  }));
}

export function normalizePaymentsScenarioSafe(raw: unknown): PaymentScenario {
  const parsed = paymentScenarioSchema.safeParse(
    typeof raw === "string" ? raw.toUpperCase() : raw,
  );
  if (parsed.success) return parsed.data;
  return "SINGLE";
}

```

lib/constants/ptCities.ts

```

export const PT_CITIES = [
  "Porto",
  "Lisboa",
  "Braga",
  "Coimbra",
  "Aveiro",
  "Faro",
  "Setúbal",
  "Leiria",
  "Viseu",
  "Guimarães",
  "Matosinhos",
  "Vila Nova de Gaia",
  "Maia",
  "Póvoa de Varzim",
  "Funchal",
  "Évora",
  "Cascais",
  "Sintra",
  "Amadora",
  "Almada",
] as const;

```

```
export type PTCity = (typeof PT_CITIES)[number];
```

lib/emailSender.ts

```
"use server";

import { sendEmail, assertResendReady } from "@/lib/resendClient";
import {
  renderPurchaseConfirmationEmail,
  renderTournamentScheduleEmail,
  renderOwnerTransferEmail,
  renderOfficialEmailVerificationEmail,
} from "@/lib/emailTemplates";

type PurchaseEmailInput = {
  to: string;
  eventTitle: string;
  eventSlug: string;
  startsAt?: string | null;
  endsAt?: string | null;
  locationName?: string | null;
  ticketsCount: number;
  ticketUrl: string;
};

export async function sendPurchaseConfirmationEmail(input: PurchaseEmailInput) {
  assertResendReady();
  const { subject, html, text } = renderPurchaseConfirmationEmail({
    eventTitle: input.eventTitle,
    eventSlug: input.eventSlug,
    startsAt: input.startsAt,
    endsAt: input.endsAt,
    locationName: input.locationName,
    ticketsCount: input.ticketsCount,
    ticketUrl: input.ticketUrl,
  });

  return sendEmail({
    to: input.to,
    subject,
    html,
    text,
  });
}

type TournamentEmailInput = {
  to: string;
  eventTitle: string;
  scheduleHtml?: string;
  scheduleText?: string;
  ticketUrl?: string;
};

export async function sendTournamentScheduleEmail(input: TournamentEmailInput) {
  assertResendReady();
  const { subject, html, text } = renderTournamentScheduleEmail({
    eventTitle: input.eventTitle,
    scheduleHtml: input.scheduleHtml,
    scheduleText: input.scheduleText,
    ticketUrl: input.ticketUrl,
  });

  return sendEmail({
    to: input.to,
    subject,
    html,
    text,
  });
}

function getAppBaseUrl() {
  return (
    process.env.NEXT_PUBLIC_BASE_URL ??
    process.env.NEXT_PUBLIC_APP_URL ??
    "https://orya.pt"
  );
}
```

```

    );
}

type OwnerTransferEmailInput = {
  to: string;
  organizerName: string;
  actorName: string;
  token: string;
  expiresAt?: Date | null;
};

export async function sendOwnerTransferEmail(input: OwnerTransferEmailInput) {
  assertResendReady();
  const baseUrl = getAppBaseUrl();
  const confirmUrl = `${baseUrl}/organizador/owner/confirm?token=${encodeURIComponent(input.token)}`;
  const { subject, html, text } = renderOwnerTransferEmail({
    organizerName: input.organizerName,
    actorName: input.actorName,
    confirmUrl,
    expiresAt: input.expiresAt,
  });

  return sendEmail({
    to: input.to,
    subject,
    html,
    text,
  });
}

type OfficialEmailVerificationInput = {
  to: string;
  organizerName: string;
  token: string;
  pendingEmail: string;
  expiresAt?: Date | null;
};

export async function sendOfficialEmailVerificationEmail(input: OfficialEmailVerificationInput) {
  assertResendReady();
  const baseUrl = getAppBaseUrl();
  const confirmUrl = `${baseUrl}/organizador/settings/verify?token=${encodeURIComponent(input.token)}`;
  const { subject, html, text } = renderOfficialEmailVerificationEmail({
    organizerName: input.organizerName,
    confirmUrl,
    expiresAt: input.expiresAt,
    pendingEmail: input.pendingEmail,
  });

  return sendEmail({
    to: input.to,
    subject,
    html,
    text,
  });
}

```

lib/emailTemplates.ts

```

type PurchaseEmailPayload = {
  eventTitle: string;
  eventSlug: string;
  startsAt?: string | null;
  endsAt?: string | null;
  locationName?: string | null;
  ticketsCount: number;
  ticketUrl: string;
};

type OwnerTransferEmailPayload = {
  organizerName: string;
  actorName: string;
  confirmUrl: string;
  expiresAt?: Date | null;
};

```

```

type OfficialEmailVerificationPayload = {
  organizerName: string;
  confirmUrl: string;
  expiresAt?: Date | null;
  pendingEmail: string;
};

export function renderPurchaseConfirmationEmail(payload: PurchaseEmailPayload) {
  const formatter = new Intl.DateTimeFormat("pt-PT", {
    weekday: "long",
    day: "2-digit",
    month: "long",
    hour: "2-digit",
    minute: "2-digit",
  });
  const startStr = payload.startsAt
    ? formatter.format(new Date(payload.startsAt))
    : null;
  const endStr = payload.endsAt ? formatter.format(new Date(payload.endsAt)) : null;

  const dateLine = startStr
    ? endStr
      ? `${startStr} - ${endStr}`
      : startStr
    : "Data a anunciar";

  const whereLine = payload.locationName?.trim() || "Local a anunciar";

  return {
    subject: `💡 Bilhetes confirmados - ${payload.eventTitle}`,
    html: `
      <div style="font-family: Arial, sans-serif; color: #0f172a;">
        <h2 style="color:#111827;">Obrigado pela tua compra!</h2>
        <p>Os teus bilhetes para <strong>${payload.eventTitle}</strong> estão confirmados.</p>
        <ul>
          <li><strong>Data & Hora:</strong> ${dateLine}</li>
          <li><strong>Local:</strong> ${whereLine}</li>
          <li><strong>Quantidade:</strong> ${payload.ticketsCount} bilhete(s)</li>
        </ul>
        <p>Podes ver os bilhetes e o QR diretamente aqui:</p>
        <p><a href="${payload.ticketUrl}" style="color:#2563eb;font-weight:bold;">Ver bilhetes</a></p>
        <p style="margin-top:24px; font-size:12px; color:#6b7280;">Se não foste tu a fazer esta compra, contacta a equipa ORYA.
      </p>
      </div>
    `,
    text: `Obrigado pela tua compra!
Evento: ${payload.eventTitle}
Data & Hora: ${dateLine}
Local: ${whereLine}
Bilhetes: ${payload.ticketsCount}
Ver bilhetes: ${payload.ticketUrl}
`,
  };
}

type TournamentEmailPayload = {
  eventTitle: string;
  scheduleHtml?: string;
  scheduleText?: string;
  ticketUrl?: string;
};

export function renderTournamentScheduleEmail(payload: TournamentEmailPayload) {
  const html = `
    <div style="font-family: Arial, sans-serif; color: #0f172a;">
      <h2 style="color:#111827;">Horário do torneio - ${payload.eventTitle}</h2>
      <p>Segue o plano de jogos/horários:</p>
      <div style="margin:12px 0; padding:12px; background:#f8fafc; border-radius:8px;">
        ${payload.scheduleHtml ?? "<p>Horários brevemente.</p>"}
      </div>
      ${
        payload.ticketUrl
        ? `<p>Podes ver os teus bilhetes aqui: <a href="${payload.ticketUrl}" style="color:#2563eb;font-weight:bold;">Bilhetes</a></p>`
        : ""
      }
    </div>
  `;
}

```

```

`;

const text = `Horário do torneio - ${payload.eventTitle}

${payload.scheduleText ?? "Horários brevemente."}

${payload.ticketUrl ? `Bilhetes: ${payload.ticketUrl}` : ""}`;

return {
  subject: `🕒 Horário do torneio - ${payload.eventTitle}`,
  html,
  text,
};
}

export function renderOwnerTransferEmail(payload: OwnerTransferEmailPayload) {
  const expiresLine = payload.expiresAt ? `O pedido expira em ${payload.expiresAt.toLocaleString("pt-PT")}.` : "";
  const html = `
    <div style="font-family: Arial, sans-serif; color: #0f172a;">
      <h2 style="color:#111827;">Pedido para te tornares OWNER</h2>
      <p><strong>${payload.actorName}</strong> pediu para te passar o papel de OWNER da organização
      <strong>${payload.organizerName}</strong>.</p>
      <p>Confirmares significa que ficas como OWNER único e os outros Owners passam a Co-owner.</p>
      <p style="margin:16px 0;"><a href="${payload.confirmUrl}" style="background:#111827;color:#ffffff;padding:12px
      18px;border-radius:10px;text-decoration:none;font-weight:bold;">Confirmar transferência</a></p>
      <p style="color:#6b7280; font-size:12px;">${expiresLine} || "O pedido expira em breve."</p>
    </div>
  `;

  const text = `Pedido para te tornares OWNER
${payload.actorName} quer passar a organização "${payload.organizerName}" para ti.
Confirma aqui: ${payload.confirmUrl}
${expiresLine}`;

  return {
    subject: `🚀 Pedido de OWNER - ${payload.organizerName}`,
    html,
    text,
  };
}

export function renderOfficialEmailVerificationEmail(payload: OfficialEmailVerificationPayload) {
  const expiresLine = payload.expiresAt ? `O pedido expira em ${payload.expiresAt.toLocaleString("pt-PT")}.` : "";
  const html = `
    <div style="font-family: Arial, sans-serif; color: #0f172a;">
      <h2 style="color:#111827;">Verifica o email oficial</h2>
      <p>Queres definir <strong>${payload.pendingEmail}</strong> como email oficial da organização
      <strong>${payload.organizerName}</strong>.</p>
      <p>Usamos este email para faturação, alertas e pedidos sensíveis.</p>
      <p style="margin:16px 0;"><a href="${payload.confirmUrl}" style="background:#111827;color:#ffffff;padding:12px
      18px;border-radius:10px;text-decoration:none;font-weight:bold;">Confirmar email</a></p>
      <p style="color:#6b7280; font-size:12px;">${expiresLine} || "O pedido expira em breve."</p>
    </div>
  `;

  const text = `Verifica o email oficial - ${payload.organizerName}

Email: ${payload.pendingEmail}
Confirmar: ${payload.confirmUrl}
${expiresLine}`;

  return {
    subject: `🕒 Verifica o email oficial - ${payload.organizerName}`,
    html,
    text,
  };
}

```

lib/env.ts

```

import "server-only";
// Central helper for server-side environment variables (server-only).
// ⚠ Não importar este módulo em componentes com "use client".
const required = [
  "SUPABASE_URL",
  "SUPABASE_ANON_KEY",

```

```

"SUPABASE_SERVICE_ROLE",
"DATABASE_URL",
"STRIPE_SECRET_KEY",
"STRIPE_WEBHOOK_SECRET",
"QR_SECRET_KEY",
"RESEND_API_KEY",
] as const;

type EnvKey = (typeof required)[number];

function getEnv(key: EnvKey): string {
  const value = process.env[key];
  if (!value) {
    throw new Error(`Missing env var: ${key}`);
  }
  return value;
}

export const env = {
  supabaseUrl: getEnv("SUPABASE_URL"),
  supabaseAnonKey: getEnv("SUPABASE_ANON_KEY"),
  serviceRoleKey: getEnv("SUPABASE_SERVICE_ROLE"),
  dbUrl: getEnv("DATABASE_URL"),
  stripeSecretKey: getEnv("STRIPE_SECRET_KEY"),
  stripeWebhookSecret: getEnv("STRIPE_WEBHOOK_SECRET"),
  qrSecretKey: getEnv("QR_SECRET_KEY"),
  resendApiKey: getEnv("RESEND_API_KEY"),
  resendFrom: [
    process.env.RESEND_FROM ??
    process.env.RESEND_FROM_EMAIL ??
    "no-reply@orya.pt",
  ],
};

```

lib/events.ts

```

// lib/events.ts
import type { Event, TicketType } from "@prisma/client";

type EventLike = {
  startsAt: Date | string;
  endsAt?: Date | string | null;
};

function toDate(value: Date | string | null | undefined): Date | null {
  if (!value) return null;
  if (value instanceof Date) return value;

  const d = new Date(value);
  if (Number.isNaN(d.getTime())) return null;

  return d;
}

export function isPast(event: EventLike | Event): boolean {
  const end = "endsAt" in event && event.endsAt ? toDate(event.endsAt) : toDate(event.startsAt);
  if (!end) return false;

  return end.getTime() < Date.now();
}

export function isToday(event: EventLike | Event): boolean {
  const start = toDate(event.startsAt);
  if (!start) return false;

  const now = new Date();
  return (
    start.getFullYear() === now.getFullYear() &&
    start.getMonth() === now.getMonth() &&
    start.getDate() === now.getDate()
  );
}

export function formatEventDateTime(
  event: EventLike | Event,
  locale: string = "pt-PT"
): string {

```

```

const start = toDate(event.startsAt);
const end =
  "endsAt" in event && event.endsAt ? toDate(event.endsAt) : null;

if (!start) return "";

const dateFormatter = new Intl.DateTimeFormat(locale, {
  weekday: "short",
  day: "2-digit",
  month: "short",
});

const timeFormatter = new Intl.DateTimeFormat(locale, {
  hour: "2-digit",
  minute: "2-digit",
});

const datePart = dateFormatter.format(start);
const startTime = timeFormatter.format(start);

if (end && start.toDateString() === end.toDateString()) {
  const endTime = timeFormatter.format(end);
  return `${datePart}, ${startTime}-${endTime}`;
}

return `${datePart}, ${startTime}`;
}

export type EventCardDTO = {
  id: number;
  slug: string;
  title: string;
  description: string;
  type: Event["type"];
  startsAt: Date | null;
  endsAt: Date | null;
  locationName: string | null;
  locationCity: string | null;
  isFree: boolean;
  priceFrom: number | null;
  coverImageUrl: string | null;
};

/**
 * Mapeia um Event (com ticketTypes) para o formato usado nos cards da home.
 */
export function mapEventToCardDTO(
  event:
    | (Partial<Event> & { ticketTypes?: (Partial<TicketType> | null)[] | null })
    | null
): EventCardDTO | null {
  if (!event) return null;

  if (
    typeof event.id !== "number" ||
    typeof event.slug !== "string" ||
    typeof event.title !== "string" ||
    typeof event.description !== "string" ||
    typeof event.type !== "string"
  ) {
    return null;
  }

  let priceFrom: number | null = null;

  if (event.ticketTypes && event.ticketTypes.length > 0) {
    const prices = event.ticketTypes
      .filter((tt): tt is { price: number } => Boolean(tt) && typeof tt?.price === "number")
      .map((tt) => tt.price);

    if (prices.length > 0) {
      priceFrom = Math.min(...prices);
    }
  }

  return {
    id: event.id,
    slug: event.slug,
  };
}

```

```

    title: event.title,
    description: event.description,
    startsAt: event.startsAt ?? null,
    endsAt: event.endsAt ?? null,
    locationName: event.locationName ?? null,
    locationCity: event.locationCity ?? null,
    isFree: Boolean(event.isFree),
    priceFrom: priceFrom !== null ? priceFrom / 100 : null,
    type: event.type,
    coverImageUrl: event.coverImageUrl ?? null,
  };
}

```

lib/filters.js

```

// JS shim para testes Node (sem loader TS)
export function clampWithGap(minValue, maxValue, step, gap, bounds) {
  const quantize = (v) => Math.round(v / step) * step;
  const snappedMin = Math.max(bounds.min, Math.min(minValue, maxValue - gap));
  const snappedMax = Math.min(bounds.max, Math.max(maxValue, snappedMin + gap));
  return { min: quantize(snappedMin), max: quantize(snappedMax) };
}

```

lib/filters.ts

```

export function clampWithGap(
  minValue: number,
  maxValue: number,
  step: number,
  gap: number,
  bounds: { min: number; max: number }
) {
  const quantize = (v: number) => Math.round(v / step) * step;
  const snappedMin = Math.max(bounds.min, Math.min(minValue, maxValue - gap));
  const snappedMax = Math.min(bounds.max, Math.max(maxValue, snappedMin + gap));
  return { min: quantize(snappedMin), max: quantize(snappedMax) };
}

```

lib/flags.ts

```

export function isFlagEnabled(name: string): boolean {
  // Flags estão todos ativos por padrão; ficheiro mantém API para evitar refactors grandes.
  return true;
}

export const featureFlags = {
  NEW_NAVBAR: () => true,
  NEW_EXPLORE_FILTERS: () => true,
  NEW_STRIPE_CONNECT_FLOW: () => true,
  NOTIFICATIONS_V1: () => true,
};

```

lib/globalUsernames.ts

```

import { Prisma, PrismaClient } from "@prisma/client";
import { prisma } from "@/lib/prisma";
import { validateUsername } from "@/lib/username";

type Tx = Prisma.TransactionClient | PrismaClient;
export type UsernameOwnerType = "user" | "organizer";

export class UsernameTakenError extends Error {
  code = "USERNAME_TAKEN";
  constructor(username: string) {
    super(`Username ${username} já está a ser usado`);
  }
}

export function normalizeAndValidateUsername(raw: string) {

```

```

const result = validateUsername(raw);
if (!result.valid) {
  return { ok: false as const, error: result.error };
}
return { ok: true as const, username: result.normalized };
}

export async function checkUsernameAvailability(username: string, tx: Tx = prisma) {
  const normalizedResult = normalizeAndValidateUsername(username);
  if (!normalizedResult.ok) return normalizedResult;

  try {
    const existing = await tx.globalUsername.findUnique({
      where: { username: normalizedResult.username },
      select: { ownerType: true, ownerId: true },
    });
    return { ok: true as const, available: !existing, username: normalizedResult.username };
  } catch (err) {
    const code = (err as { code?: string })?.code;
    const msg = err instanceof Error ? err.message : "";
    const missingTable = code === "P2021" || code === "P2022" || msg.toLowerCase().includes("does not exist");
    if (missingTable) {
      console.warn("[globalUsernames] table/column missing while checking availability");
      return { ok: true as const, available: true, username: normalizedResult.username };
    }
    throw err;
  }
}

export async function setUsernameForOwner(options: {
  username: string;
  ownerType: UsernameOwnerType;
  ownerId: string | number;
  tx?: Tx;
}) {
  const { username, ownerType, ownerId } = options;
  const providedTx = options.tx;

  const validated = normalizeAndValidateUsername(rawUsername);
  if (!validated.ok) {
    return { ok: false as const, error: validated.error };
  }

  const username = validated.username;
  const ownerIdStr = String(ownerId);

  const run = async (trx: Tx) => {
    const existing = await trx.globalUsername.findUnique({
      where: { username },
      select: { ownerType: true, ownerId: true },
    });

    if (existing && (existing.ownerType !== ownerType || existing.ownerId !== ownerIdStr)) {
      throw new UsernameTakenError(username);
    }

    await trx.globalUsername.deleteMany({
      where: {
        ownerType,
        ownerId: ownerIdStr,
        username: { not: username },
      },
    });

    await trx.globalUsername.upsert({
      where: { username },
      update: { ownerType, ownerId: ownerIdStr, updatedAt: new Date() },
      create: {
        username,
        ownerType,
        ownerId: ownerIdStr,
      },
    });
  };

  return { ok: true as const, username };
};

if (providedTx) {

```

```

try {
  return await run(providedTx);
} catch (err) {
  const code = (err as { code?: string })?.code;
  const msg = err instanceof Error ? err.message : "";
  const missingTable = code === "P2021" || code === "P2022" || msg.toLowerCase().includes("relation") ||
msg.toLowerCase().includes("does not exist");
  if (missingTable) {
    console.warn("[globalUsernames] table/column missing, skipping username reservation");
    return { ok: false as const, error: "USERNAME_TABLE_MISSING" as const };
  }
  throw err;
}

try {
  return await prisma.$transaction(run);
} catch (err) {
  const code = (err as { code?: string })?.code;
  const msg = err instanceof Error ? err.message : "";
  const missingTable = code === "P2021" || code === "P2022" || msg.toLowerCase().includes("relation") ||
msg.toLowerCase().includes("does not exist");
  if (missingTable) {
    console.warn("[globalUsernames] table/column missing, skipping username reservation");
    return { ok: false as const, error: "USERNAME_TABLE_MISSING" as const };
  }
  throw err;
}

/**
 * Remove usernames associados a um owner específico (user ou organizer).
 * Util em deletes/cleanup de conta/org.
 */
export async function clearUsernameForOwner(options: {
  ownerType: UsernameOwnerType;
  ownerId: string | number;
  tx?: Tx;
}) {
  const { ownerType, ownerId } = options;
  const client = options.tx ?? prisma;
  await client.globalUsername.deleteMany({
    where: { ownerType, ownerId: String(ownerId) },
  });
  return { ok: true as const };
}

```

lib/image.ts

```

export function optimizeImageUrl(
  url: string | null | undefined,
  width = 1200,
  quality = 75,
  format: "webp" | "avif" | "auto" = "webp",
) {
  if (!url || typeof url !== "string") return url ?? "";
  try {
    const parsed = new URL(url);
    if (parsed.hostname.includes("supabase.co")) {
      parsed.searchParams.set("width", String(width));
      parsed.searchParams.set("quality", String(quality));
      parsed.searchParams.set("format", format);
      return parsed.toString();
    }
    return url;
  } catch (err) {
    return url;
  }
}

export const defaultBlurDataURL =
  "data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='16' height='9' viewBox='0 0 16
9'%3E%3Cdefs%3E%3ClinearGradient id='g' x1='0' x2='1' y1='0' y2='1'%3E%3Cstop stop-color='%23050B16'
offset='0'%3E%3Cstop%3E%3Cstop stop-color='%23111F3B' offset='1'%3E%3Cstop%3E%3C/linearGradient%3E%3Crect
width='16' height='9' fill='url(%23g)'%3E%3C/svg%3E";

```

lib/money.ts

```
// lib/money.ts
const EUR_NUMBER_FORMATTER = new Intl.NumberFormat("pt-PT", {
  minimumFractionDigits: 2,
  maximumFractionDigits: 2,
});

/**
 * Formata um valor em EUR para apresentação (ex.: 24,95 €).
 */
export function formatEuro(amount: number | null | undefined): string {
  if (amount === null || amount === undefined || Number.isNaN(amount)) return "";
}

// Intl usa espaços não separáveis em PT; trocamos para espaço normal para evitar chars estranhos.
const formatted = EUR_NUMBER_FORMATTER.format(amount).replace(/\u00A0/g, " ");
return `${formatted} €`;
}

/**
 * Alias histórico usado em vários componentes. Mantém compatibilidade.
 */
export const formatMoney = formatEuro;

/**
 * Converte um valor em céntimos para euros mantendo float com 2 casas.
 */
export function centsToEuro(cents: number | null | undefined): number | null {
  if (cents === null || cents === undefined || Number.isNaN(cents)) return null;
  return cents / 100;
}
```

lib/notifications.ts

```
import { prisma } from "./prisma";
import type { NotificationPriority, NotificationType } from "@prisma/client";

export type CreateNotificationInput = {
  userId: string;
  type: NotificationType;
  title?: string | null;
  body?: string | null;
  payload?: Record<string, unknown> | null;
  ctaUrl?: string | null;
  ctaLabel?: string | null;
  priority?: NotificationPriority;
  senderVisibility?: "PUBLIC" | "PRIVATE";
  fromUserId?: string | null;
  organizerId?: number | null;
  eventId?: number | null;
  ticketId?: string | null;
  inviteId?: string | null;
};

export async function shouldNotify(userId: string, type: NotificationType) {
  const prefs = await getNotificationPrefs(userId);
  switch (type) {
    case "EVENT_SALE":
      return prefs.allowSalesAlerts;
    case "FRIEND_REQUEST":
    case "FRIEND_ACCEPT":
      return prefs.allowFriendRequests;
    case "FOLLOWED_YOU":
      return prefs.allowFriendRequests;
    case "SYSTEM_ANNOUNCE":
    case "STRIPE_STATUS":
      return prefs.allowSystemAnnouncements;
    case "EVENT_Reminder":
    case "NEW_EVENT_FROM_FOLLOWED_ORGANIZER":
      return prefs.allowEventReminders;
    default:
      return true;
  }
}
```

```

function sanitizeActor(
  actor: any,
  options: { isPrivate?: boolean; viewerId?: string | null },
) {
  if (!actor || typeof actor !== "object") return actor;
  const isSelf = options.viewerId && actor.id === options.viewerId;
  if (!options.isPrivate || isSelf) return actor;
  return {
    id: actor.id ?? null,
    username: actor.username ?? null,
    avatarUrl: actor.avatarUrl ?? null,
    fullName: null,
    email: null,
  };
}

function sanitizePayload(payload: any, opts: { senderVisibility?: "PUBLIC" | "PRIVATE"; viewerId?: string | null }) {
  if (!payload || typeof payload !== "object") return payload;
  const clone: Record<string, unknown> = { ...payload };
  if (clone.actor) {
    clone.actor = sanitizeActor(clone.actor, { isPrivate: opts.senderVisibility === "PRIVATE", viewerId: opts.viewerId });
  }
  return clone;
}

export async function createNotification(input: CreateNotificationInput) {
  const {
    userId,
    type,
    title,
    body,
    payload,
    ctaUrl = null,
    ctaLabel = null,
    priority = "NORMAL",
    senderVisibility = "PUBLIC",
    fromUserId = null,
    organizerId = null,
    eventId = null,
    ticketId = null,
    inviteId = null,
  } = input;

  const data = {
    userId,
    type,
    title: title ?? null,
    body: body ?? null,
    payload: payload ? sanitizePayload(payload, { senderVisibility, viewerId: userId }) : undefined,
    ctaUrl: ctaUrl || undefined,
    ctaLabel: ctaLabel || undefined,
    priority,
    fromUserId: fromUserId || undefined,
    organizerId: organizerId ?? undefined,
    eventId: eventId ?? undefined,
    ticketId: ticketId ?? undefined,
    inviteId: inviteId ?? undefined,
  };

  return prisma.notification.create({ data });
}

export async function getNotificationPrefs(userId: string) {
  const existing = await prisma.notificationPreference.findUnique({ where: { userId } });
  if (existing) return existing;

  const profile = await prisma.profile.findUnique({
    where: { id: userId },
    select: {
      allowEmailNotifications: true,
      allowEventReminders: true,
      allowFriendRequests: true,
    },
  });

  const defaults = {
    userId,
  };
}

```

```

    allowEmailNotifications: profile?.allowEmailNotifications ?? true,
    allowEventReminders: profile?.allowEventReminders ?? true,
    allowFriendRequests: profile?.allowFriendRequests ?? true,
    allowSalesAlerts: true,
    allowSystemAnnouncements: true,
  };

  return prisma.notificationPreference.upsert({
    where: { userId },
    update: defaults,
    create: defaults,
  });
}

```

lib/organizationAudit.ts

```

import { Prisma, PrismaClient } from "@prisma/client";
import { prisma } from "@/lib/prisma";

type TxLike = Prisma.TransactionClient | PrismaClient;

export type OrganizationAuditInput = {
  organizerId: number;
  actorUserId?: string | null;
  action: string;
  fromUserId?: string | null;
  toUserId?: string | null;
  metadata?: Record<string, unknown> | Prisma.JsonValue;
  ip?: string | null;
  userAgent?: string | null;
};

/**
 * Regista ações sensíveis para audit trail da organização.
 * Usa TransactionClient quando já estivermos em transação.
 */
export async function recordOrganizationAudit(
  client: TxLike,
  input: OrganizationAuditInput,
) {
  // Alguns schemas podem não ter a tabela de audit; nesse caso, faz no-op.
  const auditModel = (client as any).organizationAuditLog;
  if (!auditModel?.create) return null;
  return auditModel.create({
    data: {
      organizerId: input.organizerId,
      actorUserId: input.actorUserId ?? null,
      action: input.action,
      fromUserId: input.fromUserId ?? null,
      toUserId: input.toUserId ?? null,
      metadata: (input.metadata ?? {}) as Prisma.InputJsonValue,
      ip: input.ip ?? null,
      userAgent: input.userAgent ?? null,
    },
  });
}

export async function recordOrganizationAuditSafe(input: OrganizationAuditInput) {
  return recordOrganizationAudit(prisma, input);
}

```

lib/organizerAccess.ts

```

import { OrganizerMemberRole, StaffRole, StaffScope, StaffStatus } from "@prisma/client";
import { prisma } from "@/lib/prisma";

export async function getOrganizerRole(userId: string, organizerId: number) {
  if (!userId || !organizerId) return null;
  const membership = await prisma.organizerMember.findUnique({
    where: { organizerId_userId: { organizerId, userId } },
    select: { role: true },
  });
  return membership?.role ?? null;
}

```

```

export async function canManageMembersDb(userId: string, organizerId: number) {
  const role = await getOrganizerRole(userId, organizerId);
  return role === OrganizerMemberRole.OWNER || role === OrganizerMemberRole.CO_OWNER || role === OrganizerMemberRole.ADMIN;
}

export async function canManageEventsDb(userId: string, organizerId: number) {
  const role = await getOrganizerRole(userId, organizerId);
  return role === OrganizerMemberRole.OWNER || role === OrganizerMemberRole.CO_OWNER || role === OrganizerMemberRole.ADMIN;
}

export async function canScanTickets(userId: string, eventId: number) {
  const event = await prisma.event.findUnique({
    where: { id: eventId },
    select: { organizerId: true },
  });
  if (!event || !event.organizerId) {
    return { allowed: false, reason: "EVENT_NOT_FOUND", membershipRole: null as OrganizerMemberRole | null };
  }

  const membership = await prisma.organizerMember.findUnique({
    where: { organizerId_userId: { organizerId: event.organizerId, userId } },
    select: { role: true },
  });

  const managerRoles: OrganizerMemberRole[] = [
    OrganizerMemberRole.OWNER,
    OrganizerMemberRole.CO_OWNER,
    OrganizerMemberRole.ADMIN,
  ];
  if (membership && managerRoles.includes(membership.role)) {
    return { allowed: true, membershipRole: membership.role, staffAssignmentId: null as number | null };
  }

  const staffAssignment = await prisma.staffAssignment.findFirst({
    where: {
      userId,
      status: StaffStatus.ACCEPTED,
      revokedAt: null,
      role: { in: [StaffRole.OWNER, StaffRole.ADMIN, StaffRole.CHECKIN] },
      OR: [
        { scope: StaffScope.EVENT, eventId },
        { scope: StaffScope.GLOBAL, organizerId: event.organizerId },
      ],
      select: { id: true, role: true },
    });
    if (staffAssignment) {
      return { allowed: true, membershipRole: membership?.role ?? null, staffAssignmentId: staffAssignment.id };
    }
  }

  return { allowed: false, membershipRole: membership?.role ?? null, staffAssignmentId: null as number | null, reason: "NO_PERMISSION" };
}

```

lib/organizerContext.ts

```

import { OrganizerMemberRole } from "@prisma/client";
import { prisma } from "@lib/prisma";

type Options = {
  organizerId?: number | null;
  roles?: OrganizerMemberRole[];
  // Se quisermos forçar leitura de cookie, basta passar organizerId externamente
};

export async function getActiveOrganizerForUser(userId: string, opts: Options = {}) {
  const { roles } = opts;
  const organizerId = opts.organizerId;

  const client = prisma as unknown as {
    organizerMember?: { findFirst: typeof prisma.organizerMember.findFirst; findMany: typeof prisma.organizerMember.findMany };
    organizer?: { findFirst: typeof prisma.organizer.findFirst };
  };

```

```

if (!client || typeof client.organizerMember?.findFirst !== "function") {
  console.error("[organizerContext] prisma client sem modelo organizerMember");
  return { organizer: null, membership: null };
}

let membershipsFallbackAllowed = false;
let memberships: Array< Awaited<ReturnType<typeof prisma.organizerMember.findMany>>[number]
> | null = null;

// 1) Se organizerId foi especificado, tenta buscar diretamente essa membership primeiro
if (organizerId) {
  try {
    const direct = await client.organizerMember!.findFirst({
      where: {
        userId,
        organizerId,
        ...(roles ? { role: { in: roles } } : {}),
        organizer: { status: "ACTIVE" },
      },
      include: { organizer: true },
    });
    if (direct?.organizer) {
      return { organizer: direct.organizer, membership: direct };
    }
  // fallback: se não houver membership mas o org existe, devolve só o org
  if (!direct) {
    const orgOnly = await client.organizer?.findFirst({
      where: { id: organizerId, status: "ACTIVE" },
    });
    if (orgOnly) return { organizer: orgOnly as any, membership: null };
  }
  } catch (err) {
    // se falhar, continua para os fallbacks
    const code = typeof err === "object" && err && "code" in err ? (err as { code?: string }).code : undefined;
    const msg = typeof err === "object" && err && "message" in err ? String((err as { message?: unknown }).message) : "";
    if (!(code === "P2021" || msg.includes("does not exist"))) {
      throw err;
    }
  }
}
}

try {
  memberships = await client.organizerMember!.findMany({
    where: {
      userId,
      ...(roles ? { role: { in: roles } } : {}),
      organizer: { status: "ACTIVE" },
    },
    include: { organizer: true },
    orderBy: [{ lastUsedAt: "desc" }, { createdAt: "asc" }],
  });
} catch (err: unknown) {
  const message = typeof err === "object" && err && "message" in err ? String((err as { message?: unknown }).message) : "";
  const code = typeof err === "object" && err && "code" in err ? (err as { code?: string }).code : undefined;
  // Se a coluna lastUsedAt não existir ou a tabela não existir, faz fallback sem ela
  if (code === "P2021" || message.includes("does not exist") || message.includes("Unknown argument `lastUsedAt`")) {
    membershipsFallbackAllowed = true;
    try {
      memberships = await client.organizerMember!.findMany({
        where: {
          userId,
          ...(roles ? { role: { in: roles } } : {}),
          organizer: { status: "ACTIVE" },
        },
        include: { organizer: true },
        orderBy: [{ createdAt: "asc" }],
      });
    } catch (fallbackErr) {
      const fallbackCode =
        typeof fallbackErr === "object" && fallbackErr && "code" in fallbackErr
        ? (fallbackErr as { code?: string }).code
        : undefined;
      const fallbackMsg =
        typeof fallbackErr === "object" && fallbackErr && "message" in fallbackErr
        ? String((fallbackErr as { message?: unknown }).message)
        : "";
      if (fallbackCode === "P2021" || fallbackMsg.includes("does not exist")) {

```

```

        console.warn("[organizerContext] organizer_members não existe no schema atual");
    } else {
        throw fallbackErr;
    }
}
} else {
    throw err;
}
}

if (memberships && memberships.length > 0) {
    const selected =
        (organizerId ? memberships.find((m) => m.organizerId === organizerId) : null) ??
        memberships[0];
    if (selected?.organizer) {
        return { organizer: selected.organizer, membership: selected };
    }
}

// 3) Legacy fallback (organizers.user_id) – apenas se a tabela de memberships estiver em falta. Não usar para autorização.
if (memberships === null && membershipsFallbackAllowed && typeof client.organizer?.findFirst === "function") {
    const organizer = await client.organizer.findFirst({
        where: { userId, status: "ACTIVE" },
    });
    return { organizer, membership: null };
}

return { organizer: null, membership: null };
}

```

lib/organizerPermissions.ts

```

import { OrganizerMemberRole } from "@prisma/client";

const ROLE_WEIGHT: Record<OrganizerMemberRole, number> = {
    [OrganizerMemberRole.VIEWER]: 0,
    [OrganizerMemberRole.STAFF]: 1,
    [OrganizerMemberRole.ADMIN]: 2,
    [OrganizerMemberRole.CO_OWNER]: 3,
    [OrganizerMemberRole.OWNER]: 4,
};

const ADMIN_MANAGEABLE = new Set<OrganizerMemberRole>([
    OrganizerMemberRole.STAFF,
    OrganizerMemberRole.VIEWER,
]);

const CO_OWNER_MANAGEABLE = new Set<OrganizerMemberRole>([
    OrganizerMemberRole.ADMIN,
    OrganizerMemberRole.STAFF,
    OrganizerMemberRole.VIEWER,
]);

export function isOrgOwner(role: OrganizerMemberRole | null | undefined) {
    return role === OrganizerMemberRole.OWNER;
}

export function isOrgCoOwnerOrAbove(role: OrganizerMemberRole | null | undefined) {
    return role ? ROLE_WEIGHT[role] >= ROLE_WEIGHT[OrganizerMemberRole.CO_OWNER] : false;
}

export function isOrgAdminOrAbove(role: OrganizerMemberRole | null | undefined) {
    return role ? ROLE_WEIGHT[role] >= ROLE_WEIGHT[OrganizerMemberRole.ADMIN] : false;
}

export function canManageEvents(role: OrganizerMemberRole | null | undefined) {
    return isOrgAdminOrAbove(role);
}

export function canManageBilling(role: OrganizerMemberRole | null | undefined) {
    return role === OrganizerMemberRole.OWNER;
}

export function canManageMembers(
    actorRole: OrganizerMemberRole | null | undefined,
    targetCurrentRole: OrganizerMemberRole | null | undefined,
)

```

```

desiredRole: OrganizerMemberRole | null | undefined,
) {
  if (!actorRole) return false;
  if (actorRole === OrganizerMemberRole.OWNER) return true;

  if (actorRole === OrganizerMemberRole.CO_OWNER) {
    if (
      targetCurrentRole === OrganizerMemberRole.OWNER ||
      targetCurrentRole === OrganizerMemberRole.CO_OWNER ||
      desiredRole === OrganizerMemberRole.OWNER ||
      desiredRole === OrganizerMemberRole.CO_OWNER
    ) {
      return false;
    }
    const target = targetCurrentRole ?? desiredRole;
    return target ? CO_OWNER_MANAGEABLE.has(target) : true;
  }

  if (actorRole === OrganizerMemberRole.ADMIN) {
    const target = targetCurrentRole ?? desiredRole;
    if (target && !ADMIN_MANAGEABLE.has(target)) return false;
    return desiredRole ? ADMIN_MANAGEABLE.has(desiredRole) : true;
  }

  return false;
}

```

lib/organizerRoles.ts

```

import { OrganizerMemberRole, Prisma, PrismaClient } from "@prisma/client";
import { prisma } from "@/lib/prisma";

type TxLike = Prisma.TransactionClient | PrismaClient;

/**
 * Garante unicidade de OWNER numa organização:
 * - promove/cria o utilizador como OWNER
 * - despromove todos os outros OWNER para CO_OWNER
 */
export async function setSoleOwner(
  client: TxLike,
  organizerId: number,
  userId: string,
  invitedByUserId?: string | null,
) {
  await client.organizerMember.upsert({
    where: { organizerId_userId: { organizerId, userId } },
    update: { role: OrganizerMemberRole.OWNER },
    create: {
      organizerId,
      userId,
      role: OrganizerMemberRole.OWNER,
      invitedByUserId: invitedByUserId ?? undefined,
    },
  });

  await client.organizerMember.updateMany({
    where: { organizerId, role: OrganizerMemberRole.OWNER, userId: { not: userId } },
    data: { role: OrganizerMemberRole.CO_OWNER },
  });
}

/**
 * Marca o perfil como "organizer" no array de roles (idempotente).
 */
export async function ensureUserIsOrganizer(client: TxLike, userId: string) {
  const targetProfile = await client.profile.findUnique({
    where: { id: userId },
    select: { roles: true },
  });
  if (!targetProfile) return;

  const roles = Array.isArray(targetProfile.roles) ? targetProfile.roles : [];
  if (!roles.includes("organizer")) {
    await client.profile.update({
      where: { id: userId },

```

```

        data: { roles: [...roles, "organizer"] },
    });
}
}

/***
 * Wrapper que permite usar helper fora de transação quando necessário.
 */
export async function setSoleOwnerSafe(
    organizerId: number,
    userId: string,
    invitedByUserId?: string | null,
) {
    return setSoleOwner(prisma, organizerId, userId, invitedByUserId);
}

```

lib/ownership/claimIdentity.ts

```

import { prisma } from "@/lib/prisma";
import { normalizeEmail } from "@/lib/utils/email";

// Claim automático: quando email é verificado, transfere ownership para userId.
export async function claimIdentity(email: string, userId: string, opts?: { requireVerified?: boolean }) {
    const emailNormalized = normalizeEmail(email);
    if (!emailNormalized) return;
    const client: any = prisma as any;
    if (!client.emailIdentity || typeof client.emailIdentity.findUnique !== "function") return;

    const identity = await client.emailIdentity.findUnique({
        where: { emailNormalized },
        select: { id: true, userId: true, emailVerifiedAt: true },
    });
    if (!identity) return;

    // já está claimed
    if (identity.userId === userId) return;

    if (opts?.requireVerified && !identity.emailVerifiedAt) {
        // Sem email verificado, não fazemos claim
        return;
    }

    await prisma.$transaction(async (tx) => {
        await (tx as any).emailIdentity.update({
            where: { emailNormalized },
            data: { userId, emailVerifiedAt: identity.emailVerifiedAt ?? new Date() },
        });

        // tickets
        await tx.ticket.updateMany({
            where: { ownerIdentityId: identity.id },
            data: { ownerUserId: userId, ownerIdentityId: null },
        });
        // sale_summaries
        await tx.saleSummary.updateMany({
            where: { ownerIdentityId: identity.id },
            data: { ownerUserId: userId, ownerIdentityId: null },
        });
        // tournament_entries
        await tx.tournamentEntry.updateMany({
            where: { ownerIdentityId: identity.id },
            data: { ownerUserId: userId, ownerIdentityId: null },
        });
    });
}

```

lib/ownership/resolveOwner.ts

```

import { prisma } from "@/lib/prisma";
import { normalizeEmail } from "@/lib/utils/email";

export type OwnerInput = {
    sessionId?: string | null;
    guestEmail?: string | null;
}

```

```

};

export async function resolveOwner(input: OwnerInput) {
  const sessionId = input.sessionUserId?.trim() || null;
  const guestEmail = normalizeEmail(input.guestEmail);
  const client: any = prisma as any;
  const hasEmailIdentity =
    client.EmailIdentity &&
    typeof client.EmailIdentity.findUnique === "function" &&
    typeof client.EmailIdentity.create === "function";

  if (sessionId) {
    return { ownerId: sessionId, ownerIdentityId: null, emailNormalized: guestEmail };
  }

  if (!guestEmail) {
    return { ownerId: null, ownerIdentityId: null, emailNormalized: null };
  }

  // Procura/gera EmailIdentity
  if (!hasEmailIdentity) {
    return { ownerId: null, ownerIdentityId: null, emailNormalized: guestEmail };
  }

  let identity = await client.EmailIdentity.findUnique({
    where: { emailNormalized: guestEmail },
    select: { id: true, userId: true, emailVerifiedAt: true },
  });

  if (!identity) {
    identity = await client.EmailIdentity.create({
      data: { emailNormalized: guestEmail },
      select: { id: true, userId: true, emailVerifiedAt: true },
    });
  }

  // Claim automático se já tiver user e verificado
  if (identity.userId && identity.emailVerifiedAt) {
    return { ownerId: identity.userId, ownerIdentityId: null, emailNormalized: guestEmail };
  }

  return { ownerId: null, ownerIdentityId: identity.id, emailNormalized: guestEmail };
}

```

lib/padel/eventSnapshot.ts

```

import { prisma } from "@/lib/prisma";

type CourtLite = { name: string; clubName: string | null; indoor?: boolean | null };
type PartnerClubLite = { id: number; name: string | null; city: string | null };

export type PadelEventSnapshot = {
  eventId: number;
  title: string;
  status: string;
  startsAt: string | null;
  endsAt: string | null;
  clubName: string | null;
  clubCity: string | null;
  partnerClubs: PartnerClubLite[];
  courts: CourtLite[];
  timeline: Array<{ key: string; label: string; state: "done" | "active" | "pending"; cancelled?: boolean; date: string | null }>;
};

function buildTimeline(params: { status: string; startsAt: Date | null; endsAt: Date | null }) {
  const { status, startsAt, endsAt } = params;
  const now = new Date();
  const started = startsAt ? startsAt.getTime() <= now.getTime() : false;
  const finished = status === "FINISHED" || (endsAt ? endsAt.getTime() < now.getTime() : false);
  const cancelled = status === "CANCELLED";

  return [
    {
      key: "draft",
      label: "Pré-lançamento",
    },
  ];
}

```

```

    state: status === "DRAFT" ? "active" : "done",
    date: startsAt ? startsAt.toISOString() : null,
},
{
  key: "signup",
  label: "Inscrições",
  state: status === "PUBLISHED" && !started ? "active" : status === "DRAFT" ? "pending" : "done",
  date: startsAt ? startsAt.toISOString() : null,
},
{
  key: "games",
  label: "Jogos",
  state: cancelled ? "pending" : started ? (finished ? "done" : "active") : "pending",
  date: startsAt ? startsAt.toISOString() : null,
},
{
  key: "finish",
  label: cancelled ? "Cancelado" : "Terminado",
  state: finished || cancelled ? "done" : "pending",
  cancelled,
  date: endsAt ? endsAt.toISOString() : null,
},
];
}
}

export async function buildPadelEventSnapshot(eventId: number): Promise<PadelEventSnapshot | null> {
  if (!Number.isFinite(eventId)) return null;

  const event = await prisma.event.findUnique({
    where: { id: eventId, isDeleted: false },
    select: {
      id: true,
      title: true,
      status: true,
      startsAt: true,
      endsAt: true,
      templateType: true,
      locationCity: true,
      locationName: true,
      padelTournamentConfig: {
        select: {
          numberOfCourts: true,
          partnerClubIds: true,
          advancedSettings: true,
          club: { select: { id: true, name: true, city: true, address: true } },
        },
      },
    },
  });
}

if (!event || event.templateType !== "PADEL") return null;

const config = event.padelTournamentConfig;
const advanced = (config?.advancedSettings || {}) as {
  courtsFromClubs?: Array<{ name?: string | null; clubName?: string | null; indoor?: boolean | null }>;
};

const partnerIds = config?.partnerClubIds ?? [];
const partnerClubs: PartnerClubLite[] =
  partnerIds.length > 0
    ? await prisma.padelClub.findMany({
        where: { id: { in: partnerIds } },
        select: { id: true, name: true, city: true },
      })
    : [];

const courtsFromClubs = Array.isArray(advanced?.courtsFromClubs)
  ? (advanced.courtsFromClubs || []).map((c, idx) => ({
      name: c.name || `Court ${idx + 1}`,
      clubName: c.clubName || config?.club?.name || null,
      indoor: c.indoor ?? null,
    }))
  : [];

const courts: CourtLite[] =
  courtsFromClubs.length > 0
    ? courtsFromClubs
    : Array.from({ length: Math.max(1, config?.numberOfCourts || 1) }).map(_,
      idx) => ({

```

```

        name: `Court ${idx + 1}`,
        clubName: config?.club?.name || event.locationName || null,
        indoor: null,
      )));

return {
  eventId: event.id,
  title: event.title,
  status: event.status,
  startsAt: event.startsAt?.toISOString() ?? null,
  endsAt: event.endsAt?.toISOString() ?? null,
  clubName: config?.club?.name || event.locationName || null,
  clubCity: config?.club?.city || event.locationCity || null,
  partnerClubs,
  courts,
  timeline: buildTimeline({ status: event.status, startsAt: event.startsAt, endsAt: event.endsAt ?? event.startsAt }),
};
}

```

lib/padel/validation.ts

```

export type PadelTieBreakRule =
| "HEAD_TO_HEAD"
| "SET_DIFFERENCE"
| "GAME_DIFFERENCE"
| "POINTS"
| "COIN_TOSS";

export type PadelPointsTable = Record<string, number>;

export type PadelScore = {
  sets?: Array<{ teamA: number; teamB: number }>;
  notes?: string;
};

export function isValidTieBreakRules(value: unknown): value is PadelTieBreakRule[] {
  if (!Array.isArray(value)) return false;
  return value.every((item) =>
    [
      "HEAD_TO_HEAD",
      "SET_DIFFERENCE",
      "GAME_DIFFERENCE",
      "POINTS",
      "COIN_TOSS",
    ].includes(String(item)),
  );
}

export function isValidPointsTable(value: unknown): value is PadelPointsTable {
  if (!value || typeof value !== "object") return false;
  return Object.values(value).every((v) => typeof v === "number" && Number.isFinite(v));
}

export function isValidScore(value: unknown): value is PadelScore {
  if (!value || typeof value !== "object") return false;
  const obj = value as { sets?: unknown; notes?: unknown };
  if (obj.sets) {
    if (!Array.isArray(obj.sets)) return false;
    const okSets = obj.sets.every(
      (s) =>
        s &&
        typeof s === "object" &&
        Number.isFinite((s as { teamA?: unknown }).teamA) &&
        Number.isFinite((s as { teamB?: unknown }).teamB),
    );
    if (!okSets) return false;
  }
  if (obj.notes && typeof obj.notes !== "string") return false;
  return true;
}

```

lib/payments/paymentScenario.ts

```
// SSOT wrapper: reexporta o helper atual de paymentScenario
export * from "@lib/paymentScenario";
```

lib/paymentScenario.ts

```
import { z } from "zod";

export const paymentScenarioSchema = z.enum([
  "SINGLE",
  "GROUP_SPLIT",
  "GROUP_FULL",
  "RESALE",
  "SUBSCRIPTION",
  "FREE_CHECKOUT",
]);

export type PaymentScenario = z.infer<typeof paymentScenarioSchema>

export function normalizePaymentScenario(raw: string | null | undefined): PaymentScenario {
  const value = (raw || "").toUpperCase();
  if (value === "GROUP_SPLIT") return "GROUP_SPLIT";
  if (value === "GROUP_FULL") return "GROUP_FULL";
  if (value === "RESALE") return "RESALE";
  if (value === "SUBSCRIPTION") return "SUBSCRIPTION";
  if (value === "FREE_CHECKOUT") return "FREE_CHECKOUT";
  return "SINGLE";
}
```

lib/phone.ts

```
// Utils partilhados para normalizar/validar telefones de forma consistente

// Remove caracteres inválidos, permitindo apenas dígitos e um único "+" no início
export function sanitizePhone(input: string): string {
  let cleaned = input.replace(/\^d+/g, "");
  if (cleaned.includes("+")) {
    const firstPlus = cleaned.indexOf("+");
    cleaned = "+" + cleaned.slice(firstPlus + 1).replace(/\^+/g, "");
  }
  return cleaned;
}

// Validação: dígitos com opcional "+" no início, 6 a 15 dígitos totais
export function isValidPhone(input: string): boolean {
  const value = sanitizePhone(input);
  if (!value) return false;
  return /\^+?\d{6,15}\$/.test(value);
}

// Normalização simples: devolve o valor sanitizado (já usado para guardar)
export function normalizePhone(input: string): string {
  return sanitizePhone(input);
}
```

lib/platformSettings.ts

```
import { prisma } from "@lib/prisma";

export type PlatformFeeConfig = {
  feeBps: number;
  feeFixedCents: number;
};

type FeeKeys =
  | "platform_fee_bps"
  | "platform_fee_fixed_cents"
  | "stripe_fee_bps_eu"
  | "stripe_fee_fixed_cents_eu";

type PlatformSettingKey = FeeKeys | "org_transfer_enabled";
```

```

const envPlatformFeeBps = process.env.PLATFORM_FEE_BPS ?? process.env.NEXT_PUBLIC_PLATFORM_FEE_BPS;
const envPlatformFeePercent = process.env.PLATFORM_FEE_PERCENT ?? process.env.NEXT_PUBLIC_PLATFORM_FEE_PERCENT;
const envPlatformFeeFixedCents =
  process.env.PLATFORM_FEE_FIXED_CENTS ?? process.env.NEXT_PUBLIC_PLATFORM_FEE_FIXED_CENTS;
const envPlatformFeeFixedEur =
  process.env.PLATFORM_FEE_FIXED_EUR ?? process.env.NEXT_PUBLIC_PLATFORM_FEE_FIXED_EUR;

const DEFAULT_PLATFORM_FEE_BPS = Number.isFinite(Number(envPlatformFeeBps))
  ? Number(envPlatformFeeBps)
  : Math.round(Number(envPlatformFeePercent ?? 0.08) * 10_000) || 800; // 8%
const DEFAULT_PLATFORM_FEE_FIXED_CENTS = Number.isFinite(Number(envPlatformFeeFixedCents))
  ? Number(envPlatformFeeFixedCents)
  : Math.round(Number(envPlatformFeeFixedEur ?? 0.3) * 100) || 30; // €0.30

const DEFAULT_STRIPE_FEE_BPS_EU = Number.isFinite(Number(process.env.STRIPE_FEE_BPS_EU))
  ? Number(process.env.STRIPE_FEE_BPS_EU)
  : Math.round(Number(process.env.STRIPE_FEE_PERCENT_EU ?? 0.014) * 10_000) || 140; // 1.4%
const DEFAULT_STRIPE_FEE_FIXED_CENTS_EU = Number.isFinite(Number(process.env.STRIPE_FEE_FIXED_CENTS_EU))
  ? Number(process.env.STRIPE_FEE_FIXED_CENTS_EU)
  : Math.round(Number(process.env.STRIPE_FEE_FIXED_EUR_EU ?? 0.25) * 100) || 25; // €0.25

function parseNumber(raw: unknown, fallback: number) {
  const n = Number(raw);
  return Number.isFinite(n) ? n : fallback;
}

function parseBoolean(raw: unknown, fallback: boolean) {
  if (typeof raw === "boolean") return raw;
  if (typeof raw === "string") {
    const normalized = raw.trim().toLowerCase();
    if ("1", "true", "yes", "on"].includes(normalized)) return true;
    if ("0", "false", "no", "off"].includes(normalized)) return false;
  }
  return fallback;
}

async function getSettingsMap(keys: PlatformSettingKey[]): Promise<Record<string, string>> {
  const rows = await prisma.platformSetting.findMany({
    where: {
      key: {
        in: keys,
      },
    },
  });

  return rows.reduce<Record<string, string>>((acc, row) => {
    acc[row.key] = row.value;
    return acc;
  }, {});
}

async function upsertSettings(values: { key: PlatformSettingKey; value: string }[]) {
  const tasks = values.map(({ key, value }) =>
    prisma.platformSetting.upsert({
      where: { key },
      create: { key, value },
      update: { value },
    }),
  );
  await Promise.all(tasks);
}

/**
 * Lê platform_settings (DB). Se não houver valores guardados, aplica defaults/env.
 */
export async function getPlatformFees(): Promise<PlatformFeeConfig> {
  const map = await getSettingsMap(["platform_fee_bps", "platform_fee_fixed_cents"]);

  return {
    feeBps: parseNumber(map["platform_fee_bps"], DEFAULT_PLATFORM_FEE_BPS),
    feeFixedCents: parseNumber(map["platform_fee_fixed_cents"], DEFAULT_PLATFORM_FEE_FIXED_CENTS),
  };
}

export async function setPlatformFees(config: Partial<PlatformFeeConfig>) {
  const updates: { key: FeeKeys; value: string }[] = [];

  if (config.feeBps !== undefined) {

```

```

    updates.push({ key: "platform_fee_bps", value: String(Math.max(0, Math.round(config.feeBps))) });
}
if (config.feeFixedCents !== undefined) {
  updates.push({
    key: "platform_fee_fixed_cents",
    value: String(Math.max(0, Math.round(config.feeFixedCents))),
  });
}

if (updates.length > 0) {
  await upsertSettings(updates);
}

return getPlatformFees();
}

export async function getStripeBaseFees() {
  const map = await getSettingsMap(["stripe_fee_bps_eu", "stripe_fee_fixed_cents_eu"]);

  return {
    feeBps: parseNumber(map["stripe_fee_bps_eu"], DEFAULT_STRIPE_FEE_BPS_EU),
    feeFixedCents: parseNumber(map["stripe_fee_fixed_cents_eu"], DEFAULT_STRIPE_FEE_FIXED_CENTS_EU),
    region: "UE",
  };
}

export async function setStripeBaseFees(config: Partial<PlatformFeeConfig>) {
  const updates: { key: FeeKeys; value: string }[] = [];

  if (config.feeBps !== undefined) {
    updates.push({ key: "stripe_fee_bps_eu", value: String(Math.max(0, Math.round(config.feeBps))) });
  }
  if (config.feeFixedCents !== undefined) {
    updates.push({
      key: "stripe_fee_fixed_cents_eu",
      value: String(Math.max(0, Math.round(config.feeFixedCents))),
    });
  }

  if (updates.length > 0) {
    await upsertSettings(updates);
  }

  return getStripeBaseFees();
}

export async function getPlatformAndStripeFees() {
  const [orya, stripe] = await Promise.all([getPlatformFees(), getStripeBaseFees()]);
  return { orya, stripe };
}

export async function getOrgTransferEnabled(): Promise<boolean> {
  const map = await getSettingsMap(["org_transfer_enabled"]);
  return parseBoolean(map["org_transfer_enabled"], false);
}

```

lib/pricing.ts

```

import { FeeMode } from "@prisma/client";

export type CheckoutLine = {
  ticketTypeId: number;
  quantity: number;
  unitPriceCents: number;
  currency: string;
};

export type FeeContext = {
  eventFeeModeOverride?: FeeMode | null;
  eventFeeMode?: FeeMode | null;
  organizerFeeMode?: FeeMode | null;
  platformDefaultFeeMode?: FeeMode | null;
  eventPlatformFeeBpsOverride?: number | null;
  eventPlatformFeeFixedCentsOverride?: number | null;
  organizerPlatformFeeBps?: number | null;
  organizerPlatformFeeFixedCents?: number | null;
}

```

```

platformDefaultFeeBps: number;
platformDefaultFeeFixedCents: number;
isPlatformOrg?: boolean;
};

export type PricingResult = {
  subtotalCents: number;
  discountCents: number;
  platformFeeCents: number;
  totalCents: number;
  feeMode: FeeMode;
  feeBpsApplied: number;
  feeFixedApplied: number;
};

function resolveFeeMode(ctx: FeeContext): FeeMode {
  return (
    ctx.eventFeeModeOverride ||
    ctx.eventFeeMode ||
    ctx.organizerFeeMode ||
    ctx.platformDefaultFeeMode ||
    FeeMode.ADDED
  );
}

function resolvePlatformFees(ctx: FeeContext) {
  if (ctx.isPlatformOrg) {
    return { feeBps: 0, feeFixedCents: 0 };
  }
  const feeBps =
    ctx.eventPlatformFeeBpsOverride ??
    ctx.organizerPlatformFeeBps ??
    ctx.platformDefaultFeeBps;
  const feeFixedCents =
    ctx.eventPlatformFeeFixedCentsOverride ??
    ctx.organizerPlatformFeeFixedCents ??
    ctx.platformDefaultFeeFixedCents;

  return {
    feeBps: Math.max(0, Math.round(feeBps ?? 0)),
    feeFixedCents: Math.max(0, Math.round(feeFixedCents ?? 0)),
  };
}

/**
 * Função central de cálculo de checkout/fees.
 * - Prioridade: override do evento -> configs do organizer -> defaults da plataforma.
 * - feeMode: ADDED (ON_TOP) ou INCLUDED.
 */
export function computePricing(
  subtotalCents: number,
  discountCents: number,
  ctx: FeeContext,
): PricingResult {
  const feeMode = resolveFeeMode(ctx);
  const { feeBps, feeFixedCents } = resolvePlatformFees(ctx);

  const netSubtotal = Math.max(0, subtotalCents - Math.max(0, discountCents));
  const platformFeeCents =
    netSubtotal === 0
      ? 0
      : Math.max(
        0,
        Math.round((netSubtotal * feeBps) / 10_000) + feeFixedCents,
      );
  const totalCents =
    feeMode === FeeMode.ADDED ? netSubtotal + platformFeeCents : netSubtotal;

  return {
    subtotalCents,
    discountCents: Math.max(0, discountCents),
    platformFeeCents,
    totalCents,
    feeMode,
    feeBpsApplied: feeBps,
    feeFixedApplied: feeFixedCents,
  };
}

```

```
};  
}
```

lib/prisma.ts

```
// lib/prisma.ts  
import { Prisma, PrismaClient } from "@prisma/client";  
import { PrismaPg } from "@prisma/adapter-pg";  
import { Pool } from "pg";  
import { env } from "@/lib/env";  
  
// Pool para ligar ao Postgres do Supabase  
const pool = new Pool({  
  connectionString: env.dbUrl, // usa a chave que já tens no env.ts  
  ssl:  
    process.env.NODE_ENV === "production"  
      ? undefined // em produção usas SSL normal (já tens sslmode=require na connection string)  
      : { rejectUnauthorized: false }, // em dev ignoras o certificado (já estavas a fazer)  
});  
  
const adapter = new PrismaPg(pool);  
  
// Toggle de logs verbose (queries) via env: PRISMA_LOG_QUERIES=true  
const enableQueryLog = process.env.PRISMA_LOG_QUERIES === "true";  
const logLevels: (Prisma.LogLevel | Prisma.LogDefinition)[] =  
  process.env.NODE_ENV === "development"  
    ? enableQueryLog  
    ? ["query", "error", "warn"]  
    : ["error", "warn"]  
  : ["error"];  
  
// Evitar múltiplas instâncias em dev (hot reload)  
const globalForPrisma = globalThis as unknown as {  
  prisma?: PrismaClient;  
};  
  
export const prisma =  
  globalForPrisma.prisma ??  
  new PrismaClient({  
    adapter,  
    log: logLevels,  
  });  
  
if (process.env.NODE_ENV !== "production") {  
  globalForPrisma.prisma = prisma;  
}
```

lib/profileVisibility.ts

```
export type BasicProfile = {  
  id: string;  
  username: string | null;  
  fullName: string | null;  
  avatarUrl: string | null;  
  visibility?: string | null;  
  isDeleted?: boolean | null;  
};  
  
/**  
 * Normaliza visibilidade de perfis, ocultando dados de contas apagadas ou privadas.  
 */  
export function sanitizeProfileVisibility(profile: BasicProfile | null | undefined, viewerId?: string | null) {  
  if (!profile) return null;  
  if (profile.isDeleted) {  
    return {  
      id: profile.id,  
      username: null,  
      fullName: "Conta apagada",  
      avatarUrl: null,  
      visibility: "PRIVATE",  
      isDeleted: true,  
    };  
  }  
}
```

```

const isSelf = viewerId && profile.id === viewerId;
const isPrivate = profile.visibility === "PRIVATE";

return {
  id: profile.id,
  username: profile.username,
  fullName: isPrivate && !isSelf ? null : profile.fullName,
  avatarUrl: isPrivate && !isSelf ? null : profile.avatarUrl,
  visibility: profile.visibility ?? null,
  isDeleted: !profile.isDeleted,
};
}

```

lib/promoMath.js

```

// JS shim para testes Node (sem loader TS)
export function computePromoDiscountCents({ promo, totalQuantity, amountInCents }) {
  if (amountInCents <= 0) return 0;
  if (promo.minQuantity && totalQuantity < promo.minQuantity) return 0;
  if (promo.minTotalCents && amountInCents < promo.minTotalCents) return 0;

  let discount = 0;
  if (promo.type === "PERCENTAGE") {
    discount = Math.floor((amountInCents * promo.value) / 10_000);
  } else {
    discount = Math.max(0, promo.value);
  }

  return Math.min(discount, amountInCents);
}

```

lib/promoMath.ts

```

export type PromoShape = {
  type: "PERCENTAGE" | "FIXED";
  value: number; // percentage in basis points (1000 = 10%)
  minQuantity?: number | null;
  minTotalCents?: number | null;
};

export function computePromoDiscountCents(params: {
  promo: PromoShape;
  totalQuantity: number;
  amountInCents: number;
}) {
  const { promo, totalQuantity, amountInCents } = params;
  if (amountInCents <= 0) return 0;
  if (promo.minQuantity && totalQuantity < promo.minQuantity) return 0;
  if (promo.minTotalCents && amountInCents < promo.minTotalCents) return 0;

  let discount = 0;
  if (promo.type === "PERCENTAGE") {
    discount = Math.floor((amountInCents * promo.value) / 10_000);
  } else {
    discount = Math.max(0, promo.value);
  }

  return Math.min(discount, amountInCents);
}

```

lib/qr.ts

```

// lib/qr.ts
// ORYA QR Generator v1 - preparado para ORYA2 payload

import QRCode from "qrcode";
import crypto from "crypto";
import { env } from "@lib/env";

export type QRPayload = {
  v: string;           // versão do QR
}

```

```

    t: string;           // token único (qrToken)
    ts?: number;        // timestamp opcional
};

export type QROptions = {
  theme?: "light" | "dark";
};

export async function generateQR(
  payload: QRPayload,
  options: QROptions = {}
): Promise<string> {
  const json = JSON.stringify(payload);

  const theme =
    options.theme === "dark"
      ? {
          dark: "#FFFFFF",
          light: "#000000",
        }
      : {
          dark: "#000000",
          light: "#FFFFFF",
        };

  return await QRCode.toDataURL(json, {
    width: 512,
    margin: 2,
    color: theme,
  });
}

// -----
// ORYA2 - payload & assinatura
// -----


export type ORYA2Payload = {
  v: 2; // versão do payload
  typ: "ticket";
  alg: "HS256";
  tok: string; // qrToken
  tid: string; // ticketId
  eid: number; // eventId
  uid: string | null; // userId
  ts: number; // emitido (epoch seconds)
  exp: number; // expiração (epoch seconds)
  seed?: string; // seed opcional para QR dinâmico
  rot?: number; // janela de rotação (ex: Date.now()/15000)
};

const ORYA2_PREFIX = "ORYA2:" as const;

function hmacSignPayloadB64(payloadB64: string): string {
  const hmac = crypto.createHmac("sha256", env.qrSecretKey);
  hmac.update(payloadB64);
  return hmac.digest("base64url");
}

export type SignTicketInput = {
  qrToken: string;
  ticketId: string;
  eventId: number;
  userId: string | null;
  issuedAtSec: number;
  expSec: number;
  seed?: string;
  rot?: number;
};

/**
 * Gera uma string ORYA2:<payload>.<signature>
 * a partir dos dados do bilhete.
 */
export function signTicketToORYA2(input: SignTicketInput): string {
  const payload: ORYA2Payload = {
    v,
    typ: "ticket",
    alg: "HS256",
  };
}

```

```

tok: input.qrToken,
tid: input.ticketId,
eid: input.eventId,
uid: input.userId,
ts: input.issuedAtSec,
exp: input.expSec,
...(input.seed ? { seed: input.seed } : {}),
...(typeof input.rot === "number" ? { rot: input.rot } : {}),
};

const payloadJson = JSON.stringify(payload);
const payloadB64 = Buffer.from(payloadJson).toString("base64url");
const sigB64 = hmacSignPayloadB64(payloadB64);

return `${ORYA2_PREFIX}${payloadB64}.${sigB64}`;
}

export type VerifyORYA2Result =
| { ok: true; payload: ORYA2Payload }
| { ok: false; reason: string };

/**
 * Faz parse + validação criptográfica de um token ORYA2.
 */
export function parseAndVerifyORYA2(token: string): VerifyORYA2Result {
if (!token || typeof token !== "string") {
  return { ok: false, reason: "MISSING_TOKEN" };
}

if (!token.startsWith(ORYA2_PREFIX)) {
  return { ok: false, reason: "INVALID_PREFIX" };
}

const stripped = token.slice(ORYA2_PREFIX.length);
const parts = stripped.split(".");
if (parts.length !== 2) {
  return { ok: false, reason: "INVALID_FORMAT_PARTS" };
}

const [payloadB64, sigB64] = parts;

let payloadJson: unknown;
try {
  const jsonString = Buffer.from(payloadB64, "base64url").toString("utf8");
  payloadJson = JSON.parse(jsonString);
} catch {
  return { ok: false, reason: "INVALID_PAYLOAD_B64_OR_JSON" };
}

const p = payloadJson as Partial<ORYA2Payload>;
const requiredKeys: (keyof ORYA2Payload)[] = [
  "v",
  "typ",
  "alg",
  "tok",
  "tid",
  "eid",
  "uid",
  "ts",
  "exp",
];
for (const key of requiredKeys) {
  if (p[key] === undefined || p[key] === null) {
    return { ok: false, reason: `MISSING_FIELD_${String(key)}` };
  }
}

if (p.v !== 2 || p.typ !== "ticket" || p.alg !== "HS256") {
  return { ok: false, reason: "INVALID_VERSION_OR_TYPE" };
}

const expectedSig = hmacSignPayloadB64(payloadB64);
if (expectedSig !== sigB64) {
  return { ok: false, reason: "INVALID_SIGNATURE" };
}

const nowSec = Math.floor(Date.now() / 1000);

```

```

if (typeof p.exp === "number" && p.exp < nowSec) {
  return { ok: false, reason: "TICKET_EXPIRED" };
}

return { ok: true, payload: p as ORYA2Payload };
}

export type BuildQrTokenParams = {
  ticketId: string;
  eventId: number;
  userId: string | null;
  qrToken: string;
  lifetimeSeconds?: number;
  seed?: string;
  useRotationWindow?: boolean;
};

export function buildQrToken(input: BuildQrTokenParams): string {
  const nowSec = Math.floor(Date.now() / 1000);
  const lifetime =
    typeof input.lifetimeSeconds === "number" && input.lifetimeSeconds > 0
      ? input.lifetimeSeconds
      : 60 * 60 * 8; // 8h por defeito

  const expSec = nowSec + lifetime;
  const rot = input.useRotationWindow
    ? Math.floor(Date.now() / 15000)
    : undefined;

  return signTicketToORYA2({
    qrToken: input.qrToken,
    ticketId: input.ticketId,
    eventId: input.eventId,
    userId: input.userId,
    issuedAtSec: nowSec,
    expSec,
    seed: input.seed,
    rot,
  });
}

export function parseQrToken(token: string): VerifyORYA2Result {
  return parseAndVerifyORYA2(token);
}

export function isQrTokenExpired(token: string, nowSec?: number): boolean {
  const res = parseAndVerifyORYA2(token);
  if (!res.ok) {
    // Se o token for inválido ou expirar, consideramos expirado
    return true;
  }

  const now = nowSec ?? Math.floor(Date.now() / 1000);
  return typeof res.payload.exp === "number" && res.payload.exp < now;
}

```

lib/resend.ts

```

import "server-only";
import { Resend } from "resend";
import { env } from "@lib/env";

export const resend = new Resend(env.resendApiKey);

```

lib/resendClient.ts

```

import "server-only";

type SendEmailParams = {
  to: string | string[];
  subject: string;
  html?: string;
  text?: string;
}

```

```

    replyTo?: string;
    from?: string;
};

function ensureResendConfigured() {
  const apiKey = process.env.RESEND_API_KEY;
  const from = process.env.RESEND_FROM_EMAIL;

  if (!apiKey || !from) {
    throw new Error(
      "Resend não está configurado (faltam RESEND_API_KEY ou RESEND_FROM_EMAIL).",
    );
  }

  return { apiKey, from };
}

export async function sendEmail(params: SendEmailParams) {
  const { apiKey, from } = ensureResendConfigured();

  const payload = {
    from: params.from ?? from,
    to: Array.isArray(params.to) ? params.to : [params.to],
    subject: params.subject,
    html: params.html,
    text: params.text,
    reply_to: params.replyTo,
  };

  const res = await fetch("https://api.resend.com/emails", {
    method: "POST",
    headers: {
      Authorization: `Bearer ${apiKey}`,
      "Content-Type": "application/json",
    },
    body: JSON.stringify(payload),
  });

  if (!res.ok) {
    const txt = await res.text().catch(() => "");
    throw new Error(
      `Falha ao enviar email via Resend (status ${res.status}): ${txt || res.statusText}`,
    );
  }

  return res.json().catch(() => {});
}

export function assertResendReady() {
  ensureResendConfigured();
}

```

lib/security.ts

```

// lib/security.ts
//
// Helpers simples de segurança/autenticação para ser usados nas rotas/API.
//

import type { SupabaseClient, User } from "@supabase/supabase-js";

/**
 * Garante que existe um utilizador autenticado.
 * - Se não houver sessão, lança um erro "UNAUTHENTICATED".
 * - Se houver, devolve o user do Supabase.
 */
export async function ensureAuthenticated(
  supabase: SupabaseClient
): Promise<User> {
  const {
    data: { user },
    error,
  } = await supabase.auth.getUser();

  if (error || !user) {
    throw new Error("UNAUTHENTICATED");
  }
}

```

```

}

    return user;
}

/**
 * Tipo mínimo de evento necessário para validações de ownership.
 * Não depende de Prisma, apenas da forma dos campos usados nas checks.
 */
export type BasicEventForSecurity = {
    id: string;
    ownerUserId: string;
    type: string;
};

/**
 * Garante que o utilizador é o dono de uma experiência (event.type = "EXPERIENCE").
 * - Lança "EVENT_NOT_FOUND" se o evento vier nulo/undefined.
 * - Lança "NOT_EXPERIENCE" se não for uma experiência.
 * - Lança "NOT_EVENT_OWNER" se o ownerUserId não coincidir com o userId.
 * Se tudo estiver OK, não devolve nada (return void).
 */
export function assertExperienceOwner(
    userId: string,
    event: BasicEventForSecurity | null | undefined
): void {
    if (!event) {
        throw new Error("EVENT_NOT_FOUND");
    }

    if (event.type !== "EXPERIENCE") {
        throw new Error("NOT_EXPERIENCE");
    }

    if (event.ownerUserId !== userId) {
        throw new Error("NOT_EVENT_OWNER");
    }
}

export function isOrganizer(
    profile:
        | { roles?: string[] | null }
        | null
        | undefined
): boolean {
    if (!profile || !profile.roles) return false;
    return profile.roles.includes("organizer");
}

export function assertOrganizer(
    user: User | null | undefined,
    profile:
        | { id: string; roles?: string[] | null }
        | null
        | undefined,
    organizer?: { userId: string } | null
): void {
    if (!user) {
        throw new Error("UNAUTHENTICATED");
    }

    if (!isOrganizer(profile)) {
        throw new Error("NOT_ORGANIZER");
    }

    if (organizer && (!profile || organizer.userId !== profile.id)) {
        throw new Error("ORGANIZER_MISMATCH");
    }
}

export type BasicStaffAssignment = {
    organizerId: number | null;
    eventId: number | null;
    scope: "GLOBAL" | "EVENT";
    revokedAt?: Date | null;
};

export type BasicEventForStaff = {

```

```

    id: number;
    organizerId: number | null;
};

export function isStaffAssignmentForEvent(
  assignments: BasicStaffAssignment[] | null | undefined,
  event: BasicEventForStaff | null | undefined
): boolean {
  if (!assignments || !event) return false;

  const eventId = event.id;
  const organizerId = event.organizerId;

  return assignments.some((assignment) => {
    if (assignment.revokedAt) return false;

    if (assignment.scope === "EVENT" && assignment.eventId === eventId) {
      return true;
    }

    if (
      assignment.scope === "GLOBAL" &&
      organizerId != null &&
      assignment.organizerId === organizerId
    ) {
      return true;
    }

    return false;
  });
}

export function assertStaffForEvent(
  user: User | null | undefined,
  assignments: BasicStaffAssignment[] | null | undefined,
  event: BasicEventForStaff | null | undefined
): void {
  if (!user) {
    throw new Error("UNAUTHENTICATED");
  }

  if (!event) {
    throw new Error("EVENT_NOT_FOUND");
  }

  const hasAccess = isStaffAssignmentForEvent(assignments, event);

  if (!hasAccess) {
    throw new Error("NOT_STAFF_FOR_EVENT");
  }
}

```

lib/stripeClient.ts

```

// ⚠ Nunca importar este cliente em componentes com "use client" (apenas backend / API routes).
import { env } from "@/lib/env";
import Stripe from "stripe";

export const stripe = new Stripe(env.stripeSecretKey, {
  maxNetworkRetries: 2,
  timeout: 20000,
});

```

lib/supabase/server.ts

```

export { createSupabaseServer } from "../supabaseServer";

```

lib/supabaseAdmin.ts

```

import "server-only";
import { createClient } from "@supabase/supabase-js";
import { env } from "@/lib/env";

```

```
export const supabaseAdmin = createClient(env.supabaseUrl, env.serviceRoleKey);
```

lib/supabaseBrowser.ts

```
"use client";

import { createBrowserClient } from "@supabase/ssr";

/**
 * Cliente Supabase para uso no browser.
 * Usa as variáveis NEXT_PUBLIC_*, que são expostas ao front-end.
 * ⚠ Não importar '@/lib/env' aqui, porque isso é só para código server-side.
 */

function cleanupAuthStorage() {
  if (typeof window === "undefined") return;
  const storages = [window.localStorage, window.sessionStorage];
  for (const store of storages) {
    try {
      for (let i = store.length - 1; i >= 0; i--) {
        const key = store.key(i);
        if (!key || !key.startsWith("sb-")) continue;
        // não removemos mais base64, para não apagar sessões válidas
      }
    } catch {
      // ignore
    }
  }
}

// Mantemos cookies sb- intactas; não apagamos base64 para não destruir sessões válidas
}

function getBrowserSupabaseClient() {
  const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL;
  const supabaseAnonKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY;

  if (!supabaseUrl || !supabaseAnonKey) {
    throw new Error(
      "Missing NEXT_PUBLIC_SUPABASE_URL or NEXT_PUBLIC_SUPABASE_ANON_KEY"
    );
  }

  cleanupAuthStorage();
  return createBrowserClient(supabaseUrl, supabaseAnonKey);
}

// Export compatível com o que já usavas antes

export const supabaseBrowser = getBrowserSupabaseClient();

export function createSupabaseBrowserClient() {
  return getBrowserSupabaseClient();
}

export function createSupabaseClient() {
  return getBrowserSupabaseClient();
}
```

lib/supabaseClient.ts

```
import { createClient } from '@supabase/supabase-js';

const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL!;
const supabaseAnonKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!;

export const supabase = createClient(supabaseUrl, supabaseAnonKey);
```

lib/supabaseServer.ts

```
import "server-only";
import { createServerClient } from "@supabase/ssr";
```

```

import { cookies } from "next/headers";
import { env } from "@lib/env";

function decodeBase64Cookie(raw: string) {
  const BASE64_PREFIX = "base64-";
  if (!raw.startsWith(BASE64_PREFIX)) return raw;

  const base = raw.slice(BASE64_PREFIX.length);
  const encodings: BufferEncoding[] = ["base64url", "base64"];

  for (const enc of encodings) {
    try {
      return Buffer.from(base, enc).toString("utf-8");
    } catch {
      /* try next */
    }
  }

  // Se não conseguirmos decodificar, tratamos como cookie ausente para evitar JSON.parse de strings inválidas
  return undefined;
}

/**
 * Server-side Supabase client (SSR + Route Handlers)
 * - Safe cookie reading
 * - Safe cookie writing
 * - Prevents JSON parse errors
 * - No profile fetching here
 */
export async function createSupabaseServer() {
  const cookieStore = (await cookies());

  const supabase = createServerClient(
    env.supabaseUrl,
    env.supabaseAnonKey,
  {
    cookies: {
      get(name: string) {
        try {
          // Só devolvemos cookies do Supabase (sb-*) e ignoramos o resto
          if (!name.startsWith("sb-")) return undefined;
          const raw = cookieStore.get(name)?.value;
          if (!raw) return undefined;

          // Se for um chunk (sb-*.0, sb-*.1, ...), deixamos intacto para o combinador do Supabase tratar
          const isChunk = /\.\d+$/ .test(name);
          return isChunk ? raw : decodeBase64Cookie(raw);
        } catch {
          return undefined;
        }
      },
      set(name: string, value: string, options: Record<string, unknown>) {
        try {
          cookieStore.set({ name, value, ...options });
        } catch {
          /* ignore errors for RSC */
        }
      },
      remove(name: string, options: Record<string, unknown>) {
        try {
          cookieStore.set({ name, value: "", ...options, maxAge: 0 });
        } catch {
          /* ignore */
        }
      },
    },
  );
}

return supabase;
}

export async function getCurrentUser() {
  const supabase = await createSupabaseServer();

  try {
    const { data, error } = await supabase.auth.getUser();
  
```

```

    if (error || !data?.user) {
      return { user: null, error };
    }

    return { user: data.user, error: null };
  } catch (err) {
    return { user: null, error: err };
  }
}

```

lib/tickets.ts

```

export function clampWaveQuantity(qty: number, remaining?: number | null) {
  const safeQty = Number.isFinite(qty) && qty > 0 ? Math.floor(qty) : 0;
  const cap =
    remaining === null || remaining === undefined
      ? Number.MAX_SAFE_INTEGER
      : Math.max(0, remaining);

  return Math.min(safeQty, cap);
}

```

lib/username.ts

```

const USERNAME_REGEX = /^[a-z0-9](?:[a-z0-9._]*[a-z0-9])?$/;

export type UsernameValidation =
  | { valid: true; normalized: string }
  | { valid: false; error: string };

/**
 * Remove acentos, espaços e caracteres inválidos, deixando apenas letras, números, _ e . (lowercase).
 * Limita a 30 chars e evita que termine/comence em '..'.
 */
export function sanitizeUsername(input: string): string {
  const base = (input ?? "") .normalize("NFKD") .replace(/[\u0300-\u036f]/g, ""); // remove diacríticos
  const cleaned = base.replace(/[^A-Za-z0-9._]/g, "");
  const trimmed = cleaned.replace(/^\.+/, "").replace(/\.\+$/, "");
  const collapsedDots = trimmed.replace(/\.\{2\}/g, ".");
  return collapsedDots.toLowerCase().slice(0, 30);
}

export function validateUsername(raw: string): UsernameValidation {
  const normalized = sanitizeUsername(raw);
  if (!normalized || normalized.length < 3 || normalized.length > 30) {
    return {
      valid: false,
      error: "Escolhe um username entre 3 e 30 caracteres (letras, números, _ ou .).",
    };
  }
  if (!USERNAME_REGEX.test(normalized)) {
    return {
      valid: false,
      error: "O username só pode ter letras, números, _ e . (sem espaços ou acentos).",
    };
  }
  if (normalized.includes(".")) {
    return {
      valid: false,
      error: "O username não pode ter '...' seguido.",
    };
  }
  return { valid: true, normalized };
}

export const USERNAME_RULES_HINT =
  "3-30 caracteres, letras ou números, opcionalmente _ ou ., sem espaços ou acentos.";

```

lib/userResolver.ts

```

import { prisma } from "@/lib/prisma";

export type ResolvedUser = {
  userId: string;
  profile: {
    id: string;
    username: string | null;
    fullName: string | null;
    avatarUrl: string | null;
    email: string | null;
  };
};

/**
 * Resolve um identificador fornecido (email, username ou UUID) para um utilizador.
 * Retorna informação básica do profile para evitar round-trips nos handlers.
 */
export async function resolveUserIdentifier(identifier: string): Promise<ResolvedUser | null> {
  const value = identifier.trim();
  if (!value) return null;

  const select = {
    id: true,
    username: true,
    fullName: true,
    avatarUrl: true,
  };

  // Se for UUID válido, tenta match direto
  if (/^[\d-a-fA-F]{36}$/.test(value)) {
    const getById = await prisma.profile.findUnique({
      where: { id: value },
      select,
    });
    if ( getById) {
      return {
        userId: getById.id,
        profile: {
          id: getById.id,
          username: getById.username,
          fullName: getById.fullName,
          avatarUrl: getById.avatarUrl,
          email: null,
        },
      };
    }
  }

  // Username (case insensitive). Para email, procura em auth.users e mapeia para profile.
  const lowered = value.toLowerCase();

  // Tenta por username no profile (ciphertext)
  let match = await prisma.profile.findFirst({
    where: {
      OR: [{ username: value }, { username: lowered }],
      isDeleted: false,
    },
    select,
  });

  let resolvedEmail: string | null = null;
  // Se for email, procurar em auth.users e ligar ao profile
  if (!match && value.includes("@")) {
    const userByEmail = await prisma.users.findFirst({
      where: { email: lowered },
      select: { id: true, email: true },
    });

    if (userByEmail) {
      match = await prisma.profile.findUnique({
        where: { id: userByEmail.id },
        select,
      });
      resolvedEmail = userByEmail.email ?? null;
    }
  }
}

```

```

if (!match) return null;

return {
  userId: match.id,
  profile: {
    id: match.id,
    username: match.username,
    fullName: match.fullName,
    avatarUrl: match.avatarUrl,
    email: resolvedEmail ?? (value.includes("@") ? value : null),
  },
};
}

```

lib/utils/email.ts

```

export function normalizeEmail(email?: string | null) {
  if (!email) return null;
  const trimmed = email.trim().toLowerCase();
  return trimmed || null;
}

```

lib/validation/organization.ts

```

import { z } from "zod";

/**
 * Valida NIF português (9 dígitos, dígito de controlo módulo 11).
 */
export function isValidPortugueseNIF(value: string): boolean {
  const numeric = value.replace(/\D/g, "");
  if (numeric.length !== 9) return false;

  const firstDigit = Number(numeric[0]);
  if (![1, 2, 3, 5, 6, 8, 9].includes(firstDigit)) return false;

  const digits = numeric.split("").map((d) => Number(d));
  const sum = digits.slice(0, 8).reduce((acc, digit, idx) => acc + digit * (9 - idx), 0);
  const modulo11 = sum % 11;
  const checkDigit = modulo11 < 2 ? 0 : 11 - modulo11;

  return checkDigit === digits[8];
}

/**
 * Valida IBAN (internacional) pelo algoritmo oficial.
 * Aceita qualquer país; não restringe a PT.
 */
export function isValidIBAN(value: string): boolean {
  const cleaned = value.replace(/\s+/g, "").toUpperCase();
  if (!/^([A-Z]{2})\d{2}[A-Z0-9]{1,30}$/.test(cleaned)) return false;
  if (cleaned.length < 15 || cleaned.length > 34) return false;

  // Move os 4 primeiros caracteres para o fim
  const rearranged = `${cleaned.slice(4)}${cleaned.slice(0, 4)}`;

  // Substitui letras por números (A=10 ... Z=35)
  const numericRepresentation = rearranged
    .split("")
    .map((char) => ([/[^A-Z]/.test(char) ? (char.charCodeAt(0) - 55).toString() : char]))
    .join("");

  // Calcula mod 97 com BigInt para evitar overflow
  let remainder = 0n;
  for (let i = 0; i < numericRepresentation.length; i += 1) {
    const digit = BigInt(numericRepresentation[i] ?? "0");
    remainder = (remainder * 10n + digit) % 97n;
  }

  return remainder === 1n;
}

/**
 * Valida website ou handle de Instagram.
*/

```

```

/*
export function isValidWebsiteOrInstagram(value: string): boolean {
  const trimmed = value.trim();
  if (!trimmed) return true;

  if (trimmed.startsWith("@")) {
    return /^[a-zA-Z0-9_.]{2,30}$/.test(trimmed);
  }

  const normalized = /^https?:\/\/i.test(trimmed) ? trimmed : `https://${trimmed}`;
  try {
    // Valida hostname e URL globalmente

    new URL(normalized);
    return true;
  } catch {
    return false;
  }
}

const optionalTrimmedString = z
  .union([z.string(), z.undefined(), z.null()])
  .transform((v) => (v ?? "").trim());

export const becomeOrganizerSchema = z.object({
  entityType: z
    .string()
    .trim()
    .min(1, "Escolhe o tipo de entidade."),
  businessName: z
    .string()
    .trim()
    .min(1, "Indica o nome da tua organização."),
  city: z
    .string()
    .trim()
    .min(1, "Escolhe a cidade base."),
  username: z
    .string()
    .trim()
    .min(1, "O username é obrigatório.")
    .max(30, "Máximo 30 caracteres."),
  website: optionalTrimmedString.refine(
    (value) => value === "" || isValidWebsiteOrInstagram(value),
    {
      message:
        "Website ou Instagram inválido. Usa um URL válido (ex: orya.pt) ou um @handle de Instagram.",
    },
  ),
  iban: optionalTrimmedString.refine(
    (value) => value === "" || isValidIBAN(value),
    {
      message: "IBAN inválido. Verifica os dados do teu banco."
    },
  ),
  taxId: optionalTrimmedString.refine(
    (value) => value === "" || isValidPortugueseNIF(value),
    {
      message: "NIF inválido. Verifica se tem 9 dígitos e está correto."
    },
  );
};

export type BecomeOrganizerSchema = typeof becomeOrganizerSchema;
export type BecomeOrganizerFormValues = z.infer<typeof becomeOrganizerSchema>;

```

middleware.ts

```

// Middleware para manter a sessão do Supabase fresca em cada request.
// Não faz redirect nem proteção de rotas – apenas refresh de sessão.
import { NextResponse } from "next/server";
import type { NextRequest } from "next/server";
import { createServerClient } from "@supabase/ssr";

export async function middleware(req: NextRequest) {
  const supabaseUrl = process.env.NEXT_PUBLIC_SUPABASE_URL;
  const supabaseAnonKey = process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY;

  if (!supabaseUrl || !supabaseAnonKey) {
    return NextResponse.next();
  }

  const client = createServerClient({ url: supabaseUrl, key: supabaseAnonKey });

  const { data, error } = await client.auth.refreshSession();
  if (error) {
    return NextResponse.error(error.message);
  }

  req.headers.set("Authorization", `Bearer ${data.access_token}`);
}

```

```

}

const res = NextResponse.next({
  request: {
    headers: req.headers,
  },
});

const supabase = createServerClient(supabaseUrl, supabaseAnonKey, {
  cookies: {
    get(name: string) {
      const raw = req.cookies.get(name)?.value;
      return raw ?? undefined;
    },
    set(name: string, value: string, options: Record<string, unknown>) {
      res.cookies.set({ name, value, ...options });
    },
    remove(name: string, options: Record<string, unknown>) {
      res.cookies.set({ name, value: "", ...options, maxAge: 0 });
    },
  },
});

await supabase.auth.getSession();

return res;
}

```

package.json

```
{
  "name": "orya",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "NODE_TLS_REJECT_UNAUTHORIZED=0 next dev",
    "build": "next build",
    "start": "next start",
    "backup": "node backup.js",
    "lint": "eslint",
    "typecheck": "tsc --noEmit",
    "postinstall": "prisma generate",
    "prisma:generate": "prisma generate"
  },
  "dependencies": {
    "@hookform/resolvers": "^5.2.2",
    "@prisma/adapter-pg": "^7.0.0",
    "@prisma/client": "^7.0.0",
    "@stripe/react-stripe-js": "^5.4.0",
    "@stripe/stripe-js": "^8.5.2",
    "@supabase/auth-helpers-nextjs": "^0.10.0",
    "@supabase/ssr": "^0.7.0",
    "@supabase/supabase-js": "^2.81.1",
    "@types/seederandom": "^3.0.8",
    "@vercel/analytics": "^1.6.1",
    "framer-motion": "12.23.24",
    "libphonenumber-js": "1.12.31",
    "next": "16.0.7",
    "pg": "8.16.3",
    "qrcode": "^1.5.4",
    "react": "19.2.0",
    "react-datepicker": "^8.9.0",
    "react-dom": "19.2.0",
    "react-hook-form": "^7.68.0",
    "recharts": "^3.6.0",
    "resend": "^4.8.0",
    "seedrandom": "3.0.5",
    "stripe": "^19.3.1",
    "swr": "^2.3.6",
    "zod": "^4.1.13"
  },
  "devDependencies": {
    "@tailwindcss/postcss": "^4",
    "@types/node": "^20",
    "@types/pg": "^8.16.0",
    "@types/qrcode": "^1.5.6",
  }
}
```

```

    "@types/react": "^19",
    "@types/react-dom": "^19",
    "baseline-browser-mapping": "^2.9.0",
    "eslint": "^8",
    "eslint-config-next": "16.0.2",
    "prisma": "^7.0.1",
    "tailwindcss": "^4",
    "ts-node": "^10.9.2",
    "typescript": "5",
    "vercel": "^48.11.0"
  }
}

```

prisma/migrations/20250219_add_stripe_fee_cents/migration.sql

```

-- Add Stripe fee storage to sale summaries and payment events (explicit, not estimated)
ALTER TABLE "app_v3"."sale_summaries" ADD COLUMN IF NOT EXISTS "stripe_fee_cents" INTEGER NOT NULL DEFAULT 0;
ALTER TABLE "app_v3"."payment_events" ADD COLUMN IF NOT EXISTS "stripe_fee_cents" INTEGER;

```

prisma/migrations/20250401_consolidated_plan_v2/migration.sql

```

-- CreateSchema
CREATE SCHEMA IF NOT EXISTS "app_v3";

-- CreateExtension
CREATE EXTENSION IF NOT EXISTS "citext";

-- CreateEnum
CREATE TYPE "app_v3"."NotificationType" AS ENUM ('ORGANIZER_INVITE', 'ORGANIZER_TRANSFER', 'STAFF_INVITE', 'STAFF_ROLE_CHANGE',
'EVENT_SALE', 'EVENT_PAYOUT_STATUS', 'STRIPE_STATUS', 'FRIEND_REQUEST', 'FRIEND_ACCEPT', 'EVENT_RemINDER', 'CHECKIN_READY',
'TICKET_SHARED', 'MARKETING_PROMO_ALERT', 'SYSTEM_ANNOUNCE');

-- CreateEnum
CREATE TYPE "app_v3"."NotificationPriority" AS ENUM ('LOW', 'NORMAL', 'HIGH');

-- CreateEnum
CREATE TYPE "app_v3"."OrganizerMemberRole" AS ENUM ('OWNER', 'ADMIN', 'STAFF', 'CHECKIN_ONLY');

-- CreateEnum
CREATE TYPE "app_v3"."PadelPreferredSide" AS ENUM ('ESQUERDA', 'DIREITA', 'QUALQUER');

-- CreateEnum
CREATE TYPE "app_v3"."OrganizerStatus" AS ENUM ('PENDING', 'ACTIVE', 'SUSPENDED');

-- CreateEnum
CREATE TYPE "app_v3"."EventType" AS ENUM ('EXPERIENCE', 'ORGANIZER_EVENT');

-- CreateEnum
CREATE TYPE "app_v3"."Visibility" AS ENUM ('PUBLIC', 'PRIVATE');

-- CreateEnum
CREATE TYPE "app_v3"."EventCategoryType" AS ENUM ('FESTA', 'DESPORTO', 'CONCERTO', 'PALESTRA', 'ARTE', 'COMIDA', 'DRINKS');

-- CreateEnum
CREATE TYPE "app_v3"."EventTemplateType" AS ENUM ('PARTY', 'SPORT', 'PADEL', 'VOLUNTEERING', 'TALK', 'OTHER');

-- CreateEnum
CREATE TYPE "app_v3"."OrganizationKind" AS ENUM ('CLUBE_PADEL', 'RESTAURANTE', 'EMPRESA_EVENTOS', 'ASSOCIACAO',
'PESSOA_SINGULAR');

-- CreateEnum
CREATE TYPE "app_v3"."EventStatus" AS ENUM ('DRAFT', 'PUBLISHED', 'CANCELLED', 'FINISHED');

-- CreateEnum
CREATE TYPE "app_v3"."ResaleMode" AS ENUM ('ALWAYS', 'AFTER SOLD OUT', 'DISABLED');

-- CreateEnum
CREATE TYPE "app_v3"."PadelFormat" AS ENUM ('TODOS CONTRA TODOS', 'QUADRO ELIMINATORIO', 'GRUPOS ELIMINATORIAS',
'CAMPEONATO_LIGA', 'QUADRO_AB', 'NON_STOP');

-- CreateEnum
CREATE TYPE "app_v3"."RefundFeePayer" AS ENUM ('ORGANIZER', 'CUSTOMER');

-- CreateEnum

```

```

CREATE TYPE "app_v3"."PadelPaymentMode" AS ENUM ('FULL', 'SPLIT');

-- CreateEnum
CREATE TYPE "app_v3"."PadelPairingStatus" AS ENUM ('INCOMPLETE', 'COMPLETE', 'CANCELLED');

-- CreateEnum
CREATE TYPE "app_v3"."PadelPairingSlotStatus" AS ENUM ('PENDING', 'FILLED', 'CANCELLED');

-- CreateEnum
CREATE TYPE "app_v3"."PadelPairingPaymentStatus" AS ENUM ('UNPAID', 'PAID');

-- CreateEnum
CREATE TYPE "app_v3"."PadelPairingSlotRole" AS ENUM ('CAPTAIN', 'PARTNER');

-- CreateEnum
CREATE TYPE "app_v3"."PadelMatchStatus" AS ENUM ('PENDING', 'IN_PROGRESS', 'DONE', 'CANCELLED');

-- CreateEnum
CREATE TYPE "app_v3"."FeeMode" AS ENUM ('INCLUDED', 'ADDED', 'ON_TOP');

-- CreateEnum
CREATE TYPE "app_v3"."PayoutMode" AS ENUM ('ORGANIZER', 'PLATFORM');

-- CreateEnum
CREATE TYPE "app_v3"."PromoType" AS ENUM ('PERCENTAGE', 'FIXED');

-- CreateEnum
CREATE TYPE "app_v3"."TicketTypeStatus" AS ENUM ('ON_SALE', 'UPCOMING', 'CLOSED', 'SOLD_OUT');

-- CreateEnum
CREATE TYPE "app_v3"."TicketStatus" AS ENUM ('ACTIVE', 'USED', 'REFUNDED', 'TRANSFERRED', 'RESALE_LISTED');

-- CreateEnum
CREATE TYPE "app_v3"."ReservationStatus" AS ENUM ('ACTIVE', 'COMPLETED', 'EXPIRED', 'CANCELED');

-- CreateEnum
CREATE TYPE "app_v3"."StaffScope" AS ENUM ('GLOBAL', 'EVENT');

-- CreateEnum
CREATE TYPE "app_v3"."StaffRole" AS ENUM ('OWNER', 'ADMIN', 'STAFF', 'CHECKIN');

-- CreateEnum
CREATE TYPE "app_v3"."StaffStatus" AS ENUM ('PENDING', 'ACCEPTED', 'REVOKED');

-- CreateEnum
CREATE TYPE "app_v3"."TransferStatus" AS ENUM ('PENDING', 'ACCEPTED', 'CANCELLED');

-- CreateEnum
CREATE TYPE "app_v3"."ResaleStatus" AS ENUM ('LISTED', 'SOLD', 'CANCELLED');

-- CreateTable
CREATE TABLE "app_v3"."profiles" (
    "id" UUID NOT NULL,
    "username" CITEXT,
    "full_name" TEXT,
    "avatar_url" TEXT,
    "bio" TEXT,
    "city" TEXT,
    "favourite_categories" TEXT[] DEFAULT ARRAY[]::TEXT[],
    "onboarding_done" BOOLEAN NOT NULL DEFAULT false,
    "roles" TEXT[] DEFAULT ARRAY['user']::TEXT[],
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "allow_email_notifications" BOOLEAN NOT NULL DEFAULT true,
    "allow_event_reminders" BOOLEAN NOT NULL DEFAULT true,
    "allow_friend_requests" BOOLEAN NOT NULL DEFAULT true,
    "deleted_at" TIMESTAMP(6),
    "is_deleted" BOOLEAN NOT NULL DEFAULT false,
    "visibility" "app_v3"."Visibility" NOT NULL DEFAULT 'PUBLIC',
    "is_verified" BOOLEAN NOT NULL DEFAULT false,
    "contact_phone" TEXT,
    CONSTRAINT "profiles_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."organizers" (
    "id" SERIAL NOT NULL,

```

```

"display_name" TEXT NOT NULL,
"username" CITEXT,
"stripe_account_id" TEXT,
"status" "app_v3"."OrganizerStatus" NOT NULL DEFAULT 'PENDING',
"created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"user_id" UUID,
"fee_mode" "app_v3"."FeeMode" NOT NULL DEFAULT 'ADDED',
"platform_fee_bps" INTEGER NOT NULL DEFAULT 200,
"platform_fee_fixed_cents" INTEGER NOT NULL DEFAULT 0,
"stripe_charges_enabled" BOOLEAN NOT NULL DEFAULT false,
"stripe_payouts_enabled" BOOLEAN NOT NULL DEFAULT false,
"entity_type" TEXT,
"business_name" TEXT,
"public_name" TEXT,
"city" TEXT,
"address" TEXT,
"show_address_publicly" BOOLEAN NOT NULL DEFAULT false,
"payout_iban" TEXT,
"language" TEXT DEFAULT 'pt',
"public_listing_enabled" BOOLEAN NOT NULL DEFAULT true,
"alerts_email" TEXT,
"alerts_sales_enabled" BOOLEAN NOT NULL DEFAULT true,
"alerts_payout_enabled" BOOLEAN NOT NULL DEFAULT false,
"refund_fee_payer" "app_v3"."RefundFeePayer" NOT NULL DEFAULT 'CUSTOMER',
"branding_avatar_url" TEXT,
"branding_primary_color" TEXT,
"branding_secondary_color" TEXT,
"organization_kind" "app_v3"."OrganizationKind" NOT NULL DEFAULT 'PESSOA_SINGULAR',
"padel_default_short_name" TEXT,
"padel_default_city" TEXT,
"padel_default_address" TEXT,
"padel_default_courts" INTEGER NOT NULL DEFAULT 0,
"padel_default_hours" TEXT,
"padel_default_rule_set_id" INTEGER,
"padel_favorite_categories" INTEGER[] DEFAULT ARRAY[]::INTEGER[],
CONSTRAINT "organizers_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."global_usernames" (
    "username" CITEXT NOT NULL,
    "owner_type" TEXT NOT NULL,
    "owner_id" TEXT NOT NULL,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT "global_usernames_pkey" PRIMARY KEY ("username")
);

-- CreateTable
CREATE TABLE "app_v3"."notifications" (
    "id" UUID NOT NULL,
    "userId" UUID NOT NULL,
    "type" "app_v3"."NotificationType" NOT NULL,
    "title" TEXT NOT NULL,
    "body" TEXT NOT NULL,
    "payload" JSONB,
    "cta_url" TEXT,
    "cta_label" TEXT,
    "priority" "app_v3"."NotificationPriority" NOT NULL DEFAULT 'NORMAL',
    "read_at" TIMESTAMP(3),
    "seen_at" TIMESTAMP(3),
    "expires_at" TIMESTAMP(3),
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT "notifications_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."notification_preferences" (
    "userId" UUID NOT NULL,
    "allow_email_notifications" BOOLEAN NOT NULL DEFAULT true,
    "allow_event_reminders" BOOLEAN NOT NULL DEFAULT true,
    "allow_friend_requests" BOOLEAN NOT NULL DEFAULT true,
    "allow_sales_alerts" BOOLEAN NOT NULL DEFAULT true,
    "allow_system_announcements" BOOLEAN NOT NULL DEFAULT true,

```

```

"created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT "notification_preferences_pkey" PRIMARY KEY ("userId")
);

-- CreateTable
CREATE TABLE "app_v3"."organizer_members" (
    "id" UUID NOT NULL,
    "organizer_id" INTEGER NOT NULL,
    "user_id" UUID NOT NULL,
    "role" "app_v3"."OrganizerMemberRole" NOT NULL,
    "invited_by_user_id" UUID,
    "last_used_at" TIMESTAMP(3),
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT "organizer_members_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."organizer_member_invites" (
    "id" UUID NOT NULL,
    "organizer_id" INTEGER NOT NULL,
    "invited_by_user_id" UUID NOT NULL,
    "target_identifier" CITEXT NOT NULL,
    "target_user_id" UUID,
    "role" "app_v3"."OrganizerMemberRole" NOT NULL,
    "token" UUID NOT NULL,
    "expires_at" TIMESTAMP(3) NOT NULL,
    "accepted_at" TIMESTAMP(3),
    "declined_at" TIMESTAMP(3),
    "cancelled_at" TIMESTAMP(3),
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT "organizer_member_invites_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3".events" (
    "id" SERIAL NOT NULL,
    "slug" TEXT NOT NULL,
    "title" TEXT NOT NULL,
    "description" TEXT NOT NULL,
    "type" "app_v3"."EventType" NOT NULL DEFAULT 'EXPERIENCE',
    "template_type" "app_v3"."EventTemplateType",
    "organizer_id" INTEGER,
    "starts_at" TIMESTAMP(3) NOT NULL,
    "ends_at" TIMESTAMP(3) NOT NULL,
    "location_name" TEXT NOT NULL,
    "location_city" TEXT,
    "address" TEXT,
    "lat" DOUBLE PRECISION,
    "lng" DOUBLE PRECISION,
    "is_free" BOOLEAN NOT NULL DEFAULT false,
    "status" "app_v3"."EventStatus" NOT NULL DEFAULT 'DRAFT',
    "timezone" TEXT NOT NULL DEFAULT 'Europe/Lisbon',
    "cover_image_url" TEXT,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL,
    "owner_user_id" UUID NOT NULL,
    "deleted_at" TIMESTAMP(6),
    "is_deleted" BOOLEAN NOT NULL DEFAULT false,
    "resale_mode" "app_v3"."ResaleMode" NOT NULL DEFAULT 'ALWAYS',
    "fee_mode_override" "app_v3"."FeeMode",
    "platform_fee_bps_override" INTEGER,
    "platform_fee_fixed_cents_override" INTEGER,
    "fee_mode" "app_v3"."FeeMode" NOT NULL DEFAULT 'INCLUDED',
    "is_test" BOOLEAN NOT NULL DEFAULT false,
    "payout_mode" "app_v3"."PayoutMode" NOT NULL DEFAULT 'ORGANIZER',
CONSTRAINT "events_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3".event_categories" (
    "id" SERIAL NOT NULL,

```

```

"event_id" INTEGER NOT NULL,
"category" "app_v3"."EventCategoryType" NOT NULL,
"created_at" TIMESTAMPTZ(6) NOT NULL DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT "event_categories_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."ticket_types" (
"id" SERIAL NOT NULL,
"eventId" INTEGER NOT NULL,
"name" TEXT NOT NULL,
"description" TEXT,
"price" INTEGER NOT NULL,
"currency" TEXT NOT NULL DEFAULT 'EUR',
"total_quantity" INTEGER,
"sold_quantity" INTEGER NOT NULL DEFAULT 0,
"status" "app_v3"."TicketTypeStatus" NOT NULL DEFAULT 'ON_SALE',
"starts_at" TIMESTAMP(3),
"ends_at" TIMESTAMP(3),
"sort_order" INTEGER NOT NULL DEFAULT 0,
"created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMP(3) NOT NULL,
CONSTRAINT "ticket_types_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."tickets" (
"id" TEXT NOT NULL,
"eventId" INTEGER NOT NULL,
"ticket_type_id" INTEGER NOT NULL,
"purchased_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"status" "app_v3"."TicketStatus" NOT NULL DEFAULT 'ACTIVE',
"qr_secret" TEXT NOT NULL,
"rotating_seed" UUID,
"price_paid" INTEGER NOT NULL,
"currency" TEXT NOT NULL DEFAULT 'EUR',
"stripe_payment_intent_id" TEXT,
"used_at" TIMESTAMP(3),
"user_id" UUID,
"platform_fee_cents" INTEGER NOT NULL DEFAULT 0,
"total_paid_cents" INTEGER NOT NULL DEFAULT 0,
"pairing_id" INTEGER,
"padel_split_share_cents" INTEGER,
"padel_pairing_version" INTEGER,
CONSTRAINT "tickets_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."guest_ticket_links" (
"ticket_id" TEXT NOT NULL,
"guest_email" TEXT NOT NULL,
"guest_name" TEXT NOT NULL,
"guest_phone" TEXT,
"migrated_to_user_id" UUID,
"migrated_at" TIMESTAMPTZ(6),
"created_at" TIMESTAMPTZ(6) DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMPTZ(6) DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT "guest_ticket_links_pkey" PRIMARY KEY ("ticket_id")
);

-- CreateTable
CREATE TABLE "app_v3"."ticket_reservations" (
"id" TEXT NOT NULL,
"eventId" INTEGER NOT NULL,
"ticket_type_id" INTEGER NOT NULL,
"quantity" INTEGER NOT NULL,
"status" "app_v3"."ReservationStatus" NOT NULL DEFAULT 'ACTIVE',
"expires_at" TIMESTAMP(3) NOT NULL,
"created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMP(3) NOT NULL,
"user_id" UUID,
CONSTRAINT "ticket_reservations_pkey" PRIMARY KEY ("id")
);

```

```

-- CreateTable
CREATE TABLE "app_v3"."experience_participants" (
    "id" TEXT NOT NULL,
    "eventId" INTEGER NOT NULL,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "userId" UUID NOT NULL,
    "volunteer_minutes" INTEGER,

    CONSTRAINT "experience_participants_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."event_interests" (
    "id" TEXT NOT NULL,
    "eventId" INTEGER NOT NULL,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "userId" UUID NOT NULL,

    CONSTRAINT "event_interests_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."staff_assignments" (
    "id" SERIAL NOT NULL,
    "organizer_id" INTEGER NOT NULL,
    "scope" "app_v3"."StaffScope" NOT NULL,
    "event_id" INTEGER,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "revoked_at" TIMESTAMP(3),
    "user_id" UUID NOT NULL,
    "accepted_at" TIMESTAMP(6),
    "status" "app_v3"."StaffStatus" NOT NULL DEFAULT 'PENDING',
    "role" "app_v3"."StaffRole" NOT NULL DEFAULT 'STAFF',

    CONSTRAINT "staff_assignments_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."ticket_transfers" (
    "id" TEXT NOT NULL,
    "ticket_id" TEXT NOT NULL,
    "status" "app_v3"."TransferStatus" NOT NULL DEFAULT 'PENDING',
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "completed_at" TIMESTAMP(3),
    "from_user_id" UUID NOT NULL,
    "to_user_id" UUID NOT NULL,

    CONSTRAINT "ticket_transfers_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."ticket_resales" (
    "id" TEXT NOT NULL,
    "ticket_id" TEXT NOT NULL,
    "price" INTEGER NOT NULL,
    "currency" TEXT NOT NULL DEFAULT 'EUR',
    "status" "app_v3"."ResaleStatus" NOT NULL DEFAULT 'LISTED',
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "completed_at" TIMESTAMP(3),
    "seller_user_id" UUID NOT NULL,

    CONSTRAINT "ticket_resales_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."event_views" (
    "id" TEXT NOT NULL,
    "eventId" INTEGER NOT NULL,
    "session_id" TEXT,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "userId" UUID,

    CONSTRAINT "event_views_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."event_sales_agg" (

```

```

"id" SERIAL NOT NULL,
"eventId" INTEGER NOT NULL,
"date" TIMESTAMP(3) NOT NULL,
"tickets_sold" INTEGER NOT NULL DEFAULT 0,
"revenue" INTEGER NOT NULL DEFAULT 0,

CONSTRAINT "event_sales_agg_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."payment_events" (
    "id" SERIAL NOT NULL,
    "stripe_payment_intent_id" TEXT NOT NULL,
    "status" TEXT NOT NULL DEFAULT 'PROCESSING',
    "event_id" INTEGER,
    "user_id" UUID,
    "amount_cents" INTEGER,
    "platform_fee_cents" INTEGER,
    "error_message" TEXT,
    "created_at" TIMESTAMPTZ(6) DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMPTZ(6) DEFAULT CURRENT_TIMESTAMP,

CONSTRAINT "payment_events_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."platform_settings" (
    "id" SERIAL NOT NULL,
    "key" TEXT NOT NULL,
    "value" TEXT NOT NULL,
    "created_at" TIMESTAMPTZ(6) DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMPTZ(6) DEFAULT CURRENT_TIMESTAMP,

CONSTRAINT "platform_settings_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."promo_codes" (
    "id" SERIAL NOT NULL,
    "code" TEXT NOT NULL,
    "type" "app_v3"."PromoType" NOT NULL,
    "value" INTEGER NOT NULL,
    "max_uses" INTEGER,
    "per_user_limit" INTEGER,
    "valid_from" TIMESTAMPTZ(6),
    "valid_until" TIMESTAMPTZ(6),
    "active" BOOLEAN NOT NULL DEFAULT true,
    "event_id" INTEGER,
    "created_at" TIMESTAMPTZ(6) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMPTZ(6) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "auto_apply" BOOLEAN DEFAULT false,
    "min_quantity" INTEGER,
    "min_total_cents" INTEGER,

CONSTRAINT "promo_codes_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."promo_redemptions" (
    "id" SERIAL NOT NULL,
    "promo_code_id" INTEGER NOT NULL,
    "user_id" UUID,
    "guest_email" TEXT,
    "used_at" TIMESTAMPTZ(6) NOT NULL DEFAULT CURRENT_TIMESTAMP,

CONSTRAINT "promo_redemptions_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_player_profiles" (
    "id" SERIAL NOT NULL,
    "organizer_id" INTEGER NOT NULL,
    "user_id" UUID,
    "display_name" TEXT,
    "full_name" TEXT NOT NULL,
    "email" CITEXT,
    "phone" TEXT,
    "gender" TEXT,

```

```

"level" TEXT,
"preferred_side" "app_v3"."PadelPreferredSide",
"club_name" TEXT,
"birth_date" TIMESTAMP(3),
"is_active" BOOLEAN NOT NULL DEFAULT true,
"notes" TEXT,
"created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

CONSTRAINT "padel_player_profiles_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_clubs" (
"id" SERIAL NOT NULL,
"organizer_id" INTEGER NOT NULL,
"name" TEXT NOT NULL,
"short_name" TEXT,
"city" TEXT,
"address" TEXT,
"courts_count" INTEGER NOT NULL DEFAULT 1,
"hours" TEXT,
"favorite_category_ids" INTEGER[] DEFAULT ARRAY[]::INTEGER[],
"slug" TEXT,
"is_active" BOOLEAN NOT NULL DEFAULT true,
"is_default" BOOLEAN NOT NULL DEFAULT false,
"created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

CONSTRAINT "padel_clubs_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_club_courts" (
"id" SERIAL NOT NULL,
"padel_club_id" INTEGER NOT NULL,
"name" TEXT NOT NULL,
"description" TEXT,
"surface" TEXT,
"indoor" BOOLEAN NOT NULL DEFAULT false,
"is_active" BOOLEAN NOT NULL DEFAULT true,
"display_order" INTEGER NOT NULL DEFAULT 0,
"created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

CONSTRAINT "padel_club_courts_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_club_staff" (
"id" SERIAL NOT NULL,
"padel_club_id" INTEGER NOT NULL,
"user_id" UUID,
"email" CITEXT,
"role" TEXT NOT NULL,
"inherit_to_events" BOOLEAN NOT NULL DEFAULT true,
"created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

CONSTRAINT "padel_club_staff_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_categories" (
"id" SERIAL NOT NULL,
"organizer_id" INTEGER NOT NULL,
"label" TEXT NOT NULL,
"gender_restriction" TEXT,
"min_level" TEXT,
"max_level" TEXT,
"is_default" BOOLEAN NOT NULL DEFAULT false,
"is_active" BOOLEAN NOT NULL DEFAULT true,
"season" TEXT,
"year" INTEGER,
"created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

CONSTRAINT "padel_categories_pkey" PRIMARY KEY ("id")
);

```

```

);

-- CreateTable
CREATE TABLE "app_v3"."padel_rule_sets" (
    "id" SERIAL NOT NULL,
    "organizer_id" INTEGER NOT NULL,
    "name" TEXT NOT NULL,
    "tie_break_rules" JSONB NOT NULL DEFAULT '{}',
    "points_table" JSONB NOT NULL DEFAULT '{}',
    "enabled_formats" TEXT[] DEFAULT ARRAY['TODOS CONTRA TODOS', 'QUADRO ELIMINATORIO']::TEXT[],
    "season",
    "year" INTEGER,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT "padel_rule_sets_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_tournament_configs" (
    "id" SERIAL NOT NULL,
    "event_id" INTEGER NOT NULL,
    "organizer_id" INTEGER NOT NULL,
    "padel_club_id" INTEGER,
    "partner_club_ids" INTEGER[] DEFAULT ARRAY[]::INTEGER[],
    "format" "app_v3"."PadelFormat" NOT NULL,
    "number_of_courts" INTEGER NOT NULL DEFAULT 1,
    "club_hours" TEXT,
    "rule_set_id" INTEGER,
    "default_category_id" INTEGER,
    "enabled_formats" TEXT[] DEFAULT ARRAY[]::TEXT[],
    "padel_v2_enabled" BOOLEAN NOT NULL DEFAULT false,
    "advanced_settings" JSONB,
    "split_deadline_hours" INTEGER,
    "auto_cancel_unpaid" BOOLEAN NOT NULL DEFAULT true,
    "allow_captain_assume" BOOLEAN NOT NULL DEFAULT true,
    "default_payment_mode" "app_v3"."PadelPaymentMode",
    "refund_fee_payer" "app_v3"."RefundFeePayer",
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT "padel_tournament_configs_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_pairings" (
    "id" SERIAL NOT NULL,
    "event_id" INTEGER NOT NULL,
    "organizer_id" INTEGER NOT NULL,
    "category_id" INTEGER,
    "paymentMode" "app_v3"."PadelPaymentMode" NOT NULL,
    "pairing_status" "app_v3"."PadelPairingStatus" NOT NULL DEFAULT 'INCOMPLETE',
    "created_by_user_id" UUID,
    "created_by_ticket_id" TEXT,
    "invite_token" TEXT,
    "invite_expires_at" TIMESTAMP(3),
    "locked_until" TIMESTAMP(3),
    "is_public_open" BOOLEAN NOT NULL DEFAULT false,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT "padel_pairings_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_pairing_slots" (
    "id" SERIAL NOT NULL,
    "pairing_id" INTEGER NOT NULL,
    "ticket_id" TEXT,
    "profile_id" UUID,
    "player_profile_id" INTEGER,
    "slotRole" "app_v3"."PadelPairingSlotRole" NOT NULL,
    "slot_status" "app_v3"."PadelPairingSlotStatus" NOT NULL DEFAULT 'PENDING',
    "payment_status" "app_v3"."PadelPairingPaymentStatus" NOT NULL DEFAULT 'UNPAID',
    "invited_contact" TEXT,
    "is_public_open" BOOLEAN NOT NULL DEFAULT false,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

```

```

    CONSTRAINT "padel_pairing_slots_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_teams" (
    "id" SERIAL NOT NULL,
    "event_id" INTEGER NOT NULL,
    "category_id" INTEGER,
    "player1_id" INTEGER,
    "player2_id" INTEGER,
    "is_from_matchmaking" BOOLEAN NOT NULL DEFAULT false,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT "padel_teams_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_matches" (
    "id" SERIAL NOT NULL,
    "event_id" INTEGER NOT NULL,
    "category_id" INTEGER,
    "court_number" INTEGER,
    "court_name" TEXT,
    "start_time" TIMESTAMP(3),
    "round_label" TEXT,
    "team_a_id" INTEGER,
    "team_b_id" INTEGER,
    "score" JSONB NOT NULL DEFAULT '{}',
    "score_sets" JSONB,
    "pairing_a_id" INTEGER,
    "pairing_b_id" INTEGER,
    "winner_pairing_id" INTEGER,
    "group_label" TEXT,
    "round_type" TEXT,
    "status" "app_v3"."PadelMatchStatus" NOT NULL DEFAULT 'PENDING',
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT "padel_matches_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."padel_ranking_entries" (
    "id" SERIAL NOT NULL,
    "organizer_id" INTEGER NOT NULL,
    "player_id" INTEGER NOT NULL,
    "event_id" INTEGER NOT NULL,
    "points" INTEGER NOT NULL DEFAULT 0,
    "position" INTEGER,
    "level" TEXT,
    "season" TEXT,
    "year" INTEGER,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT "padel_ranking_entries_pkey" PRIMARY KEY ("id")
);

-- CreateTable
CREATE TABLE "app_v3"."follows" (
    "id" SERIAL NOT NULL,
    "follower_id" UUID NOT NULL,
    "following_id" UUID NOT NULL,
    "created_at" TIMESTAMPTZ(6) NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT "follows_pkey" PRIMARY KEY ("id")
);

-- CreateIndex
CREATE UNIQUE INDEX "profiles_username_key" ON "app_v3"."profiles"("username");

-- CreateIndex
CREATE UNIQUE INDEX "organizers_username_key" ON "app_v3"."organizers"("username");

-- CreateIndex
CREATE INDEX "organizers_user_id_idx" ON "app_v3"."organizers"("user_id");

```

```

-- CreateIndex
CREATE UNIQUE INDEX "global_usernames_owner_type_owner_id_key" ON "app_v3"."global_usernames"("owner_type", "owner_id");

-- CreateIndex
CREATE INDEX "notifications_userId_created_at_idx" ON "app_v3"."notifications"("userId", "created_at");

-- CreateIndex
CREATE INDEX "notifications_userId_read_at_idx" ON "app_v3"."notifications"("userId", "read_at");

-- CreateIndex
CREATE INDEX "organizer_members_user_id_idx" ON "app_v3"."organizer_members"("user_id");

-- CreateIndex
CREATE INDEX "organizer_members_organizer_id_role_idx" ON "app_v3"."organizer_members"("organizer_id", "role");

-- CreateIndex
CREATE UNIQUE INDEX "organizer_members_organizer_id_user_id_key" ON "app_v3"."organizer_members"("organizer_id", "user_id");

-- CreateIndex
CREATE UNIQUE INDEX "organizer_member_invites_token_key" ON "app_v3"."organizer_member_invites"("token");

-- CreateIndex
CREATE INDEX "organizer_member_invites_organizer_id_idx" ON "app_v3"."organizer_member_invites"("organizer_id");

-- CreateIndex
CREATE INDEX "organizer_member_invites_target_user_id_idx" ON "app_v3"."organizer_member_invites"("target_user_id");

-- CreateIndex
CREATE INDEX "organizer_member_invites_target_identifier_idx" ON "app_v3"."organizer_member_invites"("target_identifier");

-- CreateIndex
CREATE UNIQUE INDEX "events_slug_key" ON "app_v3"."events"("slug");

-- CreateIndex
CREATE INDEX "events_type_status_idx" ON "app_v3"."events"("type", "status");

-- CreateIndex
CREATE INDEX "events_owner_user_id_idx" ON "app_v3"."events"("owner_user_id");

-- CreateIndex
CREATE INDEX "events_owner_user_id_starts_at_idx" ON "app_v3"."events"("owner_user_id", "starts_at");

-- CreateIndex
CREATE INDEX "events_organizer_id_idx" ON "app_v3"."events"("organizer_id");

-- CreateIndex
CREATE INDEX "event_categories_category_idx" ON "app_v3"."event_categories"("category");

-- CreateIndex
CREATE UNIQUE INDEX "event_categories_unique" ON "app_v3"."event_categories"("event_id", "category");

-- CreateIndex
CREATE INDEX "ticket_types_eventId_idx" ON "app_v3"."ticket_types"("eventId");

-- CreateIndex
CREATE UNIQUE INDEX "tickets_qr_secret_key" ON "app_v3"."tickets"("qr_secret");

-- CreateIndex
CREATE INDEX "tickets_eventId_idx" ON "app_v3"."tickets"("eventId");

-- CreateIndex
CREATE INDEX "tickets_user_id_idx" ON "app_v3"."tickets"("user_id");

-- CreateIndex
CREATE INDEX "tickets_ticket_type_id_idx" ON "app_v3"."tickets"("ticket_type_id");

-- CreateIndex
CREATE INDEX "tickets_stripe_payment_intent_id_idx" ON "app_v3"."tickets"("stripe_payment_intent_id");

-- CreateIndex
CREATE INDEX "tickets_pairing_id_idx" ON "app_v3"."tickets"("pairing_id");

-- CreateIndex
CREATE INDEX "guest_ticket_links_migrated_idx" ON "app_v3"."guest_ticket_links"("migrated_to_user_id");

-- CreateIndex
CREATE INDEX "ticket_reservations_eventId_idx" ON "app_v3"."ticket_reservations"("eventId");

```

```

-- CreateIndex
CREATE INDEX "ticket_reservations_ticket_type_id_idx" ON "app_v3"."ticket_reservations"("ticket_type_id");

-- CreateIndex
CREATE INDEX "ticket_reservations_user_id_idx" ON "app_v3"."ticket_reservations"("user_id");

-- CreateIndex
CREATE INDEX "ticket_reservations_status_expires_at_idx" ON "app_v3"."ticket_reservations"("status", "expires_at");

-- CreateIndex
CREATE INDEX "experience_participants_userId_idx" ON "app_v3"."experience_participants"("userId");

-- CreateIndex
CREATE UNIQUE INDEX "experience_participants_eventId_userId_key" ON "app_v3"."experience_participants"("eventId", "userId");

-- CreateIndex
CREATE INDEX "event_interests_userId_idx" ON "app_v3"."event_interests"("userId");

-- CreateIndex
CREATE UNIQUE INDEX "event_interests_eventId_userId_key" ON "app_v3"."event_interests"("eventId", "userId");

-- CreateIndex
CREATE INDEX "staff_assignments_organizer_id_idx" ON "app_v3"."staff_assignments"("organizer_id");

-- CreateIndex
CREATE INDEX "staff_assignments_user_id_idx" ON "app_v3"."staff_assignments"("user_id");

-- CreateIndex
CREATE INDEX "staff_assignments_event_id_idx" ON "app_v3"."staff_assignments"("event_id");

-- CreateIndex
CREATE INDEX "ticket_transfers_ticket_id_idx" ON "app_v3"."ticket_transfers"("ticket_id");

-- CreateIndex
CREATE INDEX "ticket_transfers_from_user_id_idx" ON "app_v3"."ticket_transfers"("from_user_id");

-- CreateIndex
CREATE INDEX "ticket_transfers_to_user_id_idx" ON "app_v3"."ticket_transfers"("to_user_id");

-- CreateIndex
CREATE INDEX "ticket_resales_ticket_id_idx" ON "app_v3"."ticket_resales"("ticket_id");

-- CreateIndex
CREATE INDEX "ticket_resales_seller_user_id_idx" ON "app_v3"."ticket_resales"("seller_user_id");

-- CreateIndex
CREATE INDEX "event_views_eventId_idx" ON "app_v3"."event_views"("eventId");

-- CreateIndex
CREATE INDEX "event_views_userId_idx" ON "app_v3"."event_views"("userId");

-- CreateIndex
CREATE UNIQUE INDEX "event_sales_agg_eventId_key" ON "app_v3"."event_sales_agg"("eventId");

-- CreateIndex
CREATE UNIQUE INDEX "payment_events_stripe_payment_intent_id_key" ON "app_v3"."payment_events"("stripe_payment_intent_id");

-- CreateIndex
CREATE INDEX "payment_events_stripe_payment_intent_id_idx" ON "app_v3"."payment_events"("stripe_payment_intent_id");

-- CreateIndex
CREATE INDEX "payment_events_event_id_idx" ON "app_v3"."payment_events"("event_id");

-- CreateIndex
CREATE UNIQUE INDEX "platform_settings_key_idx" ON "app_v3"."platform_settings"("key");

-- CreateIndex
CREATE INDEX "promo_codes_event_id_idx" ON "app_v3"."promo_codes"("event_id");

-- CreateIndex
CREATE INDEX "promo_codes_event_active_idx" ON "app_v3"."promo_codes"("event_id", "active");

-- CreateIndex
CREATE INDEX "promo_codes_valid_idx" ON "app_v3"."promo_codes"("valid_from", "valid_until");

-- CreateIndex
CREATE INDEX "promo_redemptions_code_idx" ON "app_v3"."promo_redemptions"("promo_code_id");

```

```

-- CreateIndex
CREATE INDEX "promo_redemptions_promo_idx" ON "app_v3"."promo_redemptions"("promo_code_id");

-- CreateIndex
CREATE INDEX "promo_redemptions_user_idx" ON "app_v3"."promo_redemptions"("user_id");

-- CreateIndex
CREATE UNIQUE INDEX "padel_player_profiles_uniq_email" ON "app_v3"."padel_player_profiles"("organizer_id", "email");

-- CreateIndex
CREATE UNIQUE INDEX "padel_clubs_slug_key" ON "app_v3"."padel_clubs"("slug");

-- CreateIndex
CREATE INDEX "padel_clubs_organizer_id_idx" ON "app_v3"."padel_clubs"("organizer_id");

-- CreateIndex
CREATE INDEX "padel_club_courts_padel_club_id_idx" ON "app_v3"."padel_club_courts"("padel_club_id");

-- CreateIndex
CREATE INDEX "padel_club_staff_padel_club_id_idx" ON "app_v3"."padel_club_staff"("padel_club_id");

-- CreateIndex
CREATE INDEX "padel_tournament_configs_padel_club_id_idx" ON "app_v3"."padel_tournament_configs"("padel_club_id");

-- CreateIndex
CREATE UNIQUE INDEX "padel_tournament_configs_event_id_key" ON "app_v3"."padel_tournament_configs"("event_id");

-- CreateIndex
CREATE UNIQUE INDEX "padel_pairings_invite_token_key" ON "app_v3"."padel_pairings"("invite_token");

-- CreateIndex
CREATE INDEX "padel_pairings_event_id_idx" ON "app_v3"."padel_pairings"("event_id");

-- CreateIndex
CREATE INDEX "padel_pairings_organizer_id_idx" ON "app_v3"."padel_pairings"("organizer_id");

-- CreateIndex
CREATE INDEX "padel_pairings_category_id_idx" ON "app_v3"."padel_pairings"("category_id");

-- CreateIndex
CREATE INDEX "padel_pairing_slots_pairing_id_idx" ON "app_v3"."padel_pairing_slots"("pairing_id");

-- CreateIndex
CREATE INDEX "padel_pairing_slots_profile_id_idx" ON "app_v3"."padel_pairing_slots"("profile_id");

-- CreateIndex
CREATE INDEX "padel_pairing_slots_player_profile_id_idx" ON "app_v3"."padel_pairing_slots"("player_profile_id");

-- CreateIndex
CREATE UNIQUE INDEX "padel_pairing_slots_ticket_id_key" ON "app_v3"."padel_pairing_slots"("ticket_id");

-- CreateIndex
CREATE INDEX "idx_follows_follower" ON "app_v3"."follows"("follower_id");

-- CreateIndex
CREATE INDEX "idx_follows_following" ON "app_v3"."follows"("following_id");

-- CreateIndex
CREATE UNIQUE INDEX "follows_unique" ON "app_v3"."follows"("follower_id", "following_id");

-- AddForeignKey
ALTER TABLE "app_v3"."organizers" ADD CONSTRAINT "organizers_user_id_fkey" FOREIGN KEY ("user_id") REFERENCES "app_v3"."profiles"("id") ON DELETE SET NULL ON UPDATE NO ACTION;

-- AddForeignKey
ALTER TABLE "app_v3"."organizers" ADD CONSTRAINT "organizers_padel_default_rule_set_id_fkey" FOREIGN KEY ("padel_default_rule_set_id") REFERENCES "app_v3"."padel_rule_sets"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."notifications" ADD CONSTRAINT "notifications_userId_fkey" FOREIGN KEY ("userId") REFERENCES "app_v3"."profiles"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."notification_preferences" ADD CONSTRAINT "notification_preferences_userId_fkey" FOREIGN KEY ("userId") REFERENCES "app_v3"."profiles"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."organizer_members" ADD CONSTRAINT "organizer_members_organizer_id_fkey" FOREIGN KEY ("organizer_id")

```

```

REFERENCES "app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."organizer_members" ADD CONSTRAINT "organizer_members_user_id_fkey" FOREIGN KEY ("user_id") REFERENCES "app_v3"."profiles"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."organizer_member_invites" ADD CONSTRAINT "organizer_member_invites_organizer_id_fkey" FOREIGN KEY ("organizer_id") REFERENCES "app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."organizer_member_invites" ADD CONSTRAINT "organizer_member_invites_invited_by_user_id_fkey" FOREIGN KEY ("invited_by_user_id") REFERENCES "app_v3"."profiles"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."organizer_member_invites" ADD CONSTRAINT "organizer_member_invites_target_user_id_fkey" FOREIGN KEY ("target_user_id") REFERENCES "app_v3"."profiles"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."events" ADD CONSTRAINT "events_organizer_id_fkey" FOREIGN KEY ("organizer_id") REFERENCES "app_v3"."organizers"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."events" ADD CONSTRAINT "events_owner_user_id_fkey" FOREIGN KEY ("owner_user_id") REFERENCES "app_v3"."profiles"("id") ON DELETE RESTRICT ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."event_categories" ADD CONSTRAINT "event_categories_event_fk" FOREIGN KEY ("event_id") REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON UPDATE NO ACTION;

-- AddForeignKey
ALTER TABLE "app_v3"."ticket_types" ADD CONSTRAINT "ticket_types_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."tickets" ADD CONSTRAINT "tickets_pairing_id_fkey" FOREIGN KEY ("pairing_id") REFERENCES "app_v3"."padel_pairings"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."tickets" ADD CONSTRAINT "tickets_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."tickets" ADD CONSTRAINT "tickets_ticket_type_id_fkey" FOREIGN KEY ("ticket_type_id") REFERENCES "app_v3"."ticket_types"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."guest_ticket_links" ADD CONSTRAINT "guest_ticket_links_ticket_id_fkey" FOREIGN KEY ("ticket_id") REFERENCES "app_v3"."tickets"("id") ON DELETE CASCADE ON UPDATE NO ACTION;

-- AddForeignKey
ALTER TABLE "app_v3"."ticket_reservations" ADD CONSTRAINT "ticket_reservations_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."ticket_reservations" ADD CONSTRAINT "ticket_reservations_ticket_type_id_fkey" FOREIGN KEY ("ticket_type_id") REFERENCES "app_v3"."ticket_types"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."experience_participants" ADD CONSTRAINT "experience_participants_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."event_interests" ADD CONSTRAINT "event_interests_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."staff_assignments" ADD CONSTRAINT "staff_assignments_event_id_fkey" FOREIGN KEY ("event_id") REFERENCES "app_v3"."events"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."staff_assignments" ADD CONSTRAINT "staff_assignments_organizer_id_fkey" FOREIGN KEY ("organizer_id") REFERENCES "app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."ticket_transfers" ADD CONSTRAINT "ticket_transfers_ticket_id_fkey" FOREIGN KEY ("ticket_id") REFERENCES "app_v3"."tickets"("id") ON DELETE CASCADE ON UPDATE CASCADE;

```

```

-- AddForeignKey
ALTER TABLE "app_v3"."ticket_resales" ADD CONSTRAINT "ticket_resales_ticket_id_fkey" FOREIGN KEY ("ticket_id") REFERENCES
"app_v3"."tickets"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."event_views" ADD CONSTRAINT "event_views_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES
"app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."event_sales_agg" ADD CONSTRAINT "event_sales_agg_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES
"app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."promo_codes" ADD CONSTRAINT "promo_codes_event_fk" FOREIGN KEY ("event_id") REFERENCES "app_v3"."events"
("id") ON DELETE SET NULL ON UPDATE NO ACTION;

-- AddForeignKey
ALTER TABLE "app_v3"."promo_redemptions" ADD CONSTRAINT "promo_redemptions_promo_code_id_fkey" FOREIGN KEY ("promo_code_id")
REFERENCES "app_v3"."promo_codes"("id") ON DELETE CASCADE ON UPDATE NO ACTION;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_player_profiles" ADD CONSTRAINT "padel_player_profiles_organizer_id_fkey" FOREIGN KEY
("organizer_id") REFERENCES "app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_clubs" ADD CONSTRAINT "padel_clubs_organizer_id_fkey" FOREIGN KEY ("organizer_id") REFERENCES
"app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_club_courts" ADD CONSTRAINT "padel_club_courts_padel_club_id_fkey" FOREIGN KEY ("padel_club_id")
REFERENCES "app_v3"."padel_clubs"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_club_staff" ADD CONSTRAINT "padel_club_staff_padel_club_id_fkey" FOREIGN KEY ("padel_club_id")
REFERENCES "app_v3"."padel_clubs"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_categories" ADD CONSTRAINT "padel_categories_organizer_id_fkey" FOREIGN KEY ("organizer_id")
REFERENCES "app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_rule_sets" ADD CONSTRAINT "padel_rule_sets_organizer_id_fkey" FOREIGN KEY ("organizer_id")
REFERENCES "app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_tournament_configs" ADD CONSTRAINT "padel_tournament_configs_event_id_fkey" FOREIGN KEY ("event_id")
REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_tournament_configs" ADD CONSTRAINT "padel_tournament_configs_organizer_id_fkey" FOREIGN KEY
("organizer_id") REFERENCES "app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_tournament_configs" ADD CONSTRAINT "padel_tournament_configs_padel_club_id_fkey" FOREIGN KEY
("padel_club_id") REFERENCES "app_v3"."padel_clubs"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_tournament_configs" ADD CONSTRAINT "padel_tournament_configs_rule_set_id_fkey" FOREIGN KEY
("rule_set_id") REFERENCES "app_v3"."padel_rule_sets"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_tournament_configs" ADD CONSTRAINT "padel_tournament_configs_default_category_id_fkey" FOREIGN KEY
("default_category_id") REFERENCES "app_v3"."padel_categories"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_pairings" ADD CONSTRAINT "padel_pairings_event_id_fkey" FOREIGN KEY ("event_id") REFERENCES
"app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_pairings" ADD CONSTRAINT "padel_pairings_organizer_id_fkey" FOREIGN KEY ("organizer_id") REFERENCES
"app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_pairings" ADD CONSTRAINT "padel_pairings_category_id_fkey" FOREIGN KEY ("category_id") REFERENCES
"app_v3"."padel_categories"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_pairings" ADD CONSTRAINT "padel_pairings_created_by_ticket_id_fkey" FOREIGN KEY

```

```

("created_by_ticket_id") REFERENCES "app_v3"."tickets"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_pairing_slots" ADD CONSTRAINT "padel_pairing_slots_pairing_id_fkey" FOREIGN KEY ("pairing_id")
REFERENCES "app_v3"."padel_pairings"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_pairing_slots" ADD CONSTRAINT "padel_pairing_slots_ticket_id_fkey" FOREIGN KEY ("ticket_id")
REFERENCES "app_v3"."tickets"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_pairing_slots" ADD CONSTRAINT "padel_pairing_slots_profile_id_fkey" FOREIGN KEY ("profile_id")
REFERENCES "app_v3"."profiles"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_pairing_slots" ADD CONSTRAINT "padel_pairing_slots_player_profile_id_fkey" FOREIGN KEY ("player_profile_id")
REFERENCES "app_v3"."padel_player_profiles"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_teams" ADD CONSTRAINT "padel_teams_event_id_fkey" FOREIGN KEY ("event_id") REFERENCES
"app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_teams" ADD CONSTRAINT "padel_teams_category_id_fkey" FOREIGN KEY ("category_id") REFERENCES
"app_v3"."padel_categories"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_teams" ADD CONSTRAINT "padel_teams_player1_id_fkey" FOREIGN KEY ("player1_id") REFERENCES
"app_v3"."padel_player_profiles"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_teams" ADD CONSTRAINT "padel_teams_player2_id_fkey" FOREIGN KEY ("player2_id") REFERENCES
"app_v3"."padel_player_profiles"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_matches" ADD CONSTRAINT "padel_matches_event_id_fkey" FOREIGN KEY ("event_id") REFERENCES
"app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_matches" ADD CONSTRAINT "padel_matches_category_id_fkey" FOREIGN KEY ("category_id") REFERENCES
"app_v3"."padel_categories"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_matches" ADD CONSTRAINT "padel_matches_team_a_id_fkey" FOREIGN KEY ("team_a_id") REFERENCES
"app_v3"."padel_teams"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_matches" ADD CONSTRAINT "padel_matches_team_b_id_fkey" FOREIGN KEY ("team_b_id") REFERENCES
"app_v3"."padel_teams"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_matches" ADD CONSTRAINT "padel_matches_pairing_a_id_fkey" FOREIGN KEY ("pairing_a_id") REFERENCES
"app_v3"."padel_pairings"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_matches" ADD CONSTRAINT "padel_matches_pairing_b_id_fkey" FOREIGN KEY ("pairing_b_id") REFERENCES
"app_v3"."padel_pairings"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_matches" ADD CONSTRAINT "padel_matches_winner_pairing_id_fkey" FOREIGN KEY ("winner_pairing_id")
REFERENCES "app_v3"."padel_pairings"("id") ON DELETE SET NULL ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_ranking_entries" ADD CONSTRAINT "padel_ranking_entries_organizer_id_fkey" FOREIGN KEY ("organizer_id")
REFERENCES "app_v3"."organizers"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_ranking_entries" ADD CONSTRAINT "padel_ranking_entries_player_id_fkey" FOREIGN KEY ("player_id")
REFERENCES "app_v3"."padel_player_profiles"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."padel_ranking_entries" ADD CONSTRAINT "padel_ranking_entries_event_id_fkey" FOREIGN KEY ("event_id")
REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON UPDATE CASCADE;

-- AddForeignKey
ALTER TABLE "app_v3"."follows" ADD CONSTRAINT "follows_follower_id_fkey" FOREIGN KEY ("follower_id") REFERENCES
"app_v3"."profiles"("id") ON DELETE CASCADE ON UPDATE NO ACTION;

```

```
-- AddForeignKey
ALTER TABLE "app_v3"."follows" ADD CONSTRAINT "follows_following_id_fkey" FOREIGN KEY ("following_id") REFERENCES
"app_v3"."profiles"("id") ON DELETE CASCADE ON UPDATE NO ACTION;
```

prisma/migrations/20250415000100_add_sale_summary_lines/migration.sql

```
-- Sale summaries (fonte de verdade por PaymentIntent) e linhas por bilhete

CREATE TABLE IF NOT EXISTS "app_v3"."sale_summaries" (
    "id" SERIAL PRIMARY KEY,
    "payment_intent_id" TEXT NOT NULL,
    "event_id" INTEGER NOT NULL,
    "user_id" UUID,
    "promo_code_id" INTEGER,
    "subtotal_cents" INTEGER NOT NULL,
    "discount_cents" INTEGER NOT NULL,
    "platform_fee_cents" INTEGER NOT NULL,
    "total_cents" INTEGER NOT NULL,
    "net_cents" INTEGER NOT NULL,
    "fee_mode" "app_v3"."FeeMode",
    "currency" TEXT NOT NULL DEFAULT 'EUR',
    "created_at" TIMESTAMPTZ(6) NOT NULL DEFAULT NOW(),
    "updated_at" TIMESTAMPTZ(6) NOT NULL DEFAULT NOW(),
    CONSTRAINT "sale_summaries_payment_intent_id_key" UNIQUE ("payment_intent_id"),
    CONSTRAINT "sale_summaries_event_id_fkey" FOREIGN KEY ("event_id") REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON
UPDATE NO ACTION,
    CONSTRAINT "sale_summaries_user_id_fkey" FOREIGN KEY ("user_id") REFERENCES "app_v3"."profiles"("id") ON DELETE SET NULL ON
UPDATE NO ACTION,
    CONSTRAINT "sale_summaries_promo_code_id_fkey" FOREIGN KEY ("promo_code_id") REFERENCES "app_v3"."promo_codes"("id") ON
DELETE SET NULL ON UPDATE NO ACTION
);

CREATE INDEX IF NOT EXISTS "sale_summaries_event_idx" ON "app_v3"."sale_summaries" ("event_id");
CREATE INDEX IF NOT EXISTS "sale_summaries_user_idx" ON "app_v3"."sale_summaries" ("user_id");
CREATE INDEX IF NOT EXISTS "sale_summaries_promo_idx" ON "app_v3"."sale_summaries" ("promo_code_id");

CREATE TABLE IF NOT EXISTS "app_v3"."sale_lines" (
    "id" SERIAL PRIMARY KEY,
    "sale_summary_id" INTEGER NOT NULL,
    "event_id" INTEGER NOT NULL,
    "ticket_type_id" INTEGER NOT NULL,
    "promo_code_id" INTEGER,
    "quantity" INTEGER NOT NULL,
    "unit_price_cents" INTEGER NOT NULL,
    "discount_per_unit_cents" INTEGER NOT NULL DEFAULT 0,
    "gross_cents" INTEGER NOT NULL,
    "net_cents" INTEGER NOT NULL,
    "platform_fee_cents" INTEGER NOT NULL DEFAULT 0,
    "created_at" TIMESTAMPTZ(6) NOT NULL DEFAULT NOW(),
    CONSTRAINT "sale_lines_summary_fkey" FOREIGN KEY ("sale_summary_id") REFERENCES "app_v3"."sale_summaries"("id") ON DELETE
CASCADE ON UPDATE NO ACTION,
    CONSTRAINT "sale_lines_event_fkey" FOREIGN KEY ("event_id") REFERENCES "app_v3"."events"("id") ON DELETE CASCADE ON UPDATE
NO ACTION,
    CONSTRAINT "sale_lines_ticket_type_fkey" FOREIGN KEY ("ticket_type_id") REFERENCES "app_v3"."ticket_types"("id") ON DELETE
CASCADE ON UPDATE NO ACTION,
    CONSTRAINT "sale_lines_promo_code_fkey" FOREIGN KEY ("promo_code_id") REFERENCES "app_v3"."promo_codes"("id") ON DELETE SET
NULL ON UPDATE NO ACTION
);

CREATE INDEX IF NOT EXISTS "sale_lines_summary_idx" ON "app_v3"."sale_lines" ("sale_summary_id");
CREATE INDEX IF NOT EXISTS "sale_lines_event_idx" ON "app_v3"."sale_lines" ("event_id");
CREATE INDEX IF NOT EXISTS "sale_lines_ticket_type_idx" ON "app_v3"."sale_lines" ("ticket_type_id");
CREATE INDEX IF NOT EXISTS "sale_lines_promo_code_idx" ON "app_v3"."sale_lines" ("promo_code_id");
```

prisma/migrations/20250428090000_org_roles_notifications/migration.sql

```
-- Novos tipos e colunas para organizações e notificações + atualização de roles

-- OrgType e colunas em organizers
DO $$
BEGIN
    CREATE TYPE app_v3."OrgType" AS ENUM ('PLATFORM', 'EXTERNAL');
EXCEPTION
```

```

WHEN duplicate_object THEN NULL;
END $$;

ALTER TABLE app_v3.organizers
ADD COLUMN IF NOT EXISTS org_type app_v3."OrgType" NOT NULL DEFAULT 'EXTERNAL';

ALTER TABLE app_v3.organizers
ADD COLUMN IF NOT EXISTS official_email text;

-- Atualização do enum OrganizerMemberRole (mapear CHECKIN_ONLY -> STAFF)
DO $$
BEGIN
    CREATE TYPE app_v3."OrganizerMemberRole_new" AS ENUM ('OWNER', 'CO_OWNER', 'ADMIN', 'STAFF', 'VIEWER');
EXCEPTION
    WHEN duplicate_object THEN NULL;
END $$;

ALTER TABLE app_v3.organizer_members
ALTER COLUMN role TYPE app_v3."OrganizerMemberRole_new"
USING (
CASE role::text
    WHEN 'CHECKIN_ONLY' THEN 'STAFF'
    ELSE role::text
END::app_v3."OrganizerMemberRole_new"
);

ALTER TABLE app_v3.organizer_member_invites
ALTER COLUMN role TYPE app_v3."OrganizerMemberRole_new"
USING (
CASE role::text
    WHEN 'CHECKIN_ONLY' THEN 'STAFF'
    ELSE role::text
END::app_v3."OrganizerMemberRole_new"
);

ALTER TYPE app_v3."OrganizerMemberRole" RENAME TO "OrganizerMemberRole_old";
ALTER TYPE app_v3."OrganizerMemberRole_new" RENAME TO "OrganizerMemberRole";

DROP TYPE IF EXISTS app_v3."OrganizerMemberRole_old";

-- Garantir que NotificationType existe mesmo em bases antigas
DO $$
BEGIN
    CREATE TYPE app_v3."NotificationType" AS ENUM (
        'ORGANIZER_INVITE',
        'ORGANIZER_TRANSFER',
        'STAFF_INVITE',
        'STAFF_ROLE_CHANGE',
        'EVENT_SALE',
        'EVENT_PAYOUT_STATUS',
        'STRIPE_STATUS',
        'FRIEND_REQUEST',
        'FRIEND_ACCEPT',
        'EVENT_RemINDER',
        'CHECKIN_READY',
        'TICKET_SHARED',
        'MARKETING_PROMO_ALERT',
        'SYSTEM_ANNOUNCE',
        'FOLLOWED_YOU',
        'TICKET_TRANSFER_RECEIVED',
        'TICKET_TRANSFER_ACCEPTED',
        'TICKET_TRANSFER_DECLINED',
        'CLUB_INVITE',
        'NEW_EVENT_FROM_FOLLOWED_ORGANIZER'
    );
EXCEPTION
    WHEN duplicate_object THEN NULL;
END $$;

-- Novos valores no enum NotificationType para flows sociais e bilhetes
ALTER TYPE app_v3."NotificationType" ADD VALUE IF NOT EXISTS 'FOLLOWED_YOU';
ALTER TYPE app_v3."NotificationType" ADD VALUE IF NOT EXISTS 'TICKET_TRANSFER_RECEIVED';
ALTER TYPE app_v3."NotificationType" ADD VALUE IF NOT EXISTS 'TICKET_TRANSFER_ACCEPTED';
ALTER TYPE app_v3."NotificationType" ADD VALUE IF NOT EXISTS 'TICKET_TRANSFER_DECLINED';
ALTER TYPE app_v3."NotificationType" ADD VALUE IF NOT EXISTS 'CLUB_INVITE';
ALTER TYPE app_v3."NotificationType" ADD VALUE IF NOT EXISTS 'NEW_EVENT_FROM_FOLLOWED_ORGANIZER';

-- Tabela notifications: alinhar com novo modelo (FKs e metadata)

```

```

DO $$

BEGIN
    ALTER TABLE app_v3.notifications RENAME COLUMN "userId" TO user_id;
EXCEPTION
    WHEN undefined_column THEN NULL;
END $$;

-- Permitir título/corpo opcionais e payload default {}
ALTER TABLE app_v3.notifications ALTER COLUMN title DROP NOT NULL;
ALTER TABLE app_v3.notifications ALTER COLUMN body DROP NOT NULL;
ALTER TABLE app_v3.notifications ALTER COLUMN payload SET DEFAULT '{}':jsonb;
UPDATE app_v3.notifications SET payload = '{}'::jsonb WHERE payload IS NULL;

-- Novas colunas de relação
ALTER TABLE app_v3.notifications
    ADD COLUMN IF NOT EXISTS from_user_id uuid,
    ADD COLUMN IF NOT EXISTS organizer_id integer,
    ADD COLUMN IF NOT EXISTS event_id integer,
    ADD COLUMN IF NOT EXISTS ticket_id text,
    ADD COLUMN IF NOT EXISTS invite_id uuid,
    ADD COLUMN IF NOT EXISTS is_read boolean NOT NULL DEFAULT false;

-- Recriar FK do user (ajustar nome) e adicionar novas FKs
ALTER TABLE app_v3.notifications DROP CONSTRAINT IF EXISTS "notifications_userId_fkey";

ALTER TABLE app_v3.notifications
    ADD CONSTRAINT notifications_user_id_fkey FOREIGN KEY (user_id) REFERENCES app_v3.profiles(id) ON DELETE CASCADE ON UPDATE CASCADE,
    ADD CONSTRAINT notifications_from_user_id_fkey FOREIGN KEY (from_user_id) REFERENCES app_v3.profiles(id) ON DELETE SET NULL ON UPDATE NO ACTION,
    ADD CONSTRAINT notifications_organizer_id_fkey FOREIGN KEY (organizer_id) REFERENCES app_v3.organizers(id) ON DELETE SET NULL ON UPDATE NO ACTION,
    ADD CONSTRAINT notifications_event_id_fkey FOREIGN KEY (event_id) REFERENCES app_v3.events(id) ON DELETE SET NULL ON UPDATE NO ACTION,
    ADD CONSTRAINT notifications_ticket_id_fkey FOREIGN KEY (ticket_id) REFERENCES app_v3.tickets(id) ON DELETE SET NULL ON UPDATE NO ACTION,
    ADD CONSTRAINT notifications_invite_id_fkey FOREIGN KEY (invite_id) REFERENCES app_v3.organizer_member_invites(id) ON DELETE SET NULL ON UPDATE NO ACTION;

-- Índices atualizados
DROP INDEX IF EXISTS "notifications_userId_created_at_idx";
DROP INDEX IF EXISTS "notifications_userId_read_at_idx";

CREATE INDEX IF NOT EXISTS notifications_user_id_created_at_idx ON app_v3.notifications(user_id, created_at);
CREATE INDEX IF NOT EXISTS notifications_user_id_read_at_idx ON app_v3.notifications(user_id, read_at);
CREATE INDEX IF NOT EXISTS notifications_from_user_id_idx ON app_v3.notifications(from_user_id);
CREATE INDEX IF NOT EXISTS notifications_organizer_id_idx ON app_v3.notifications(organizer_id);
CREATE INDEX IF NOT EXISTS notifications_event_id_idx ON app_v3.notifications(event_id);
CREATE INDEX IF NOT EXISTS notifications_ticket_id_idx ON app_v3.notifications(ticket_id);
CREATE INDEX IF NOT EXISTS notifications_invite_id_idx ON app_v3.notifications(invite_id);

-- Alinhar flag de leitura com read_at existente
UPDATE app_v3.notifications SET is_read = TRUE WHERE read_at IS NOT NULL;

-- Constraint de unicidade em follows
DO $$
BEGIN
    ALTER TABLE app_v3.follows
        ADD CONSTRAINT follows_unique UNIQUE (follower_id, following_id);
EXCEPTION
    WHEN duplicate_object THEN NULL;
END $$;

-- Marcar a organização principal da ORYA como plataforma (preencher ID antes de correr em produção)
UPDATE app_v3.organizers
SET org_type = 'PLATFORM'
WHERE id = 4;

```

prisma/migrations/20250507090000_payment_events_mode_is_test/migration.sql

```

-- Add PaymentMode enum and is_test flag to payment_events
DO $$
BEGIN
    CREATE TYPE app_v3."PaymentMode" AS ENUM ('LIVE', 'TEST');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

```

```

ALTER TABLE app_v3.payment_events
ADD COLUMN IF NOT EXISTS mode app_v3."PaymentMode" NOT NULL DEFAULT 'LIVE';

ALTER TABLE app_v3.payment_events
ADD COLUMN IF NOT EXISTS is_test boolean NOT NULL DEFAULT false;

```

prisma/migrations/20250507093000_promo_snapshots/migration.sql

```

DO $$
BEGIN
    ALTER TABLE app_v3.sale_summaries
        ADD COLUMN IF NOT EXISTS promo_code_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_label_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_type_snapshot app_v3."PromoType" NULL,
        ADD COLUMN IF NOT EXISTS promo_value_snapshot integer NULL;
EXCEPTION WHEN undefined_object THEN
    -- Se o enum ainda não existir, criamos como text (fallback)
    ALTER TABLE app_v3.sale_summaries
        ADD COLUMN IF NOT EXISTS promo_code_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_label_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_type_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_value_snapshot integer NULL;
END $$;

DO $$
BEGIN
    ALTER TABLE app_v3.sale_lines
        ADD COLUMN IF NOT EXISTS promo_code_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_label_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_type_snapshot app_v3."PromoType" NULL,
        ADD COLUMN IF NOT EXISTS promo_value_snapshot integer NULL;
EXCEPTION WHEN undefined_object THEN
    ALTER TABLE app_v3.sale_lines
        ADD COLUMN IF NOT EXISTS promo_code_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_label_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_type_snapshot text NULL,
        ADD COLUMN IF NOT EXISTS promo_value_snapshot integer NULL;
END $$;

```

prisma/migrations/20250507100000_padel_soft_delete/migration.sql

```

ALTER TABLE app_v3.padel_clubs
    ADD COLUMN IF NOT EXISTS deleted_at timestamptz NULL;

ALTER TABLE app_v3.padel_club_staff
    ADD COLUMN IF NOT EXISTS is_active boolean NOT NULL DEFAULT true,
    ADD COLUMN IF NOT EXISTS deleted_at timestamptz NULL;

```

prisma/migrations/20250508120000_phase1_core_guardrails/migration.sql

```

-- Fase 1: bases para transferências seguras, email oficial e check-ins

-- official_email_verified_at nos organizers
ALTER TABLE app_v3.organizers
    ADD COLUMN IF NOT EXISTS official_email_verified_at timestamptz;

-- Feature flag global para transferências de owner
INSERT INTO app_v3.platform_settings (key, value, created_at, updated_at)
VALUES ('org_transfer_enabled', 'false', now(), now())
ON CONFLICT (key) DO UPDATE SET value = EXCLUDED.value, updated_at = now();

-- Estados de transferência de owner
DO $$
BEGIN
    CREATE TYPE app_v3."OrganizerOwnerTransferStatus" AS ENUM ('PENDING', 'CONFIRMED', 'CANCELLED', 'EXPIRED');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

-- Estados de verificação de email oficial
DO $$
BEGIN

```

```

CREATE TYPE app_v3."OrganizerEmailRequestStatus" AS ENUM ('PENDING', 'CONFIRMED', 'CANCELLED', 'EXPIRED');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

CREATE TABLE IF NOT EXISTS app_v3.organization_owner_transfers (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    organizer_id integer NOT NULL,
    from_user_id uuid NOT NULL,
    to_user_id uuid NOT NULL,
    status app_v3."OrganizerOwnerTransferStatus" NOT NULL DEFAULT 'PENDING',
    token text NOT NULL UNIQUE,
    expires_at timestamptz NOT NULL,
    created_at timestamptz DEFAULT now(),
    confirmed_at timestamptz,
    cancelled_at timestamptz,
    CONSTRAINT organization_owner_transfers_organizer_id_fkey FOREIGN KEY (organizer_id) REFERENCES app_v3.organizers(id) ON
DELETE CASCADE
);
CREATE INDEX IF NOT EXISTS organization_owner_transfers_org_idx ON app_v3.organization_owner_transfers (organizer_id);
CREATE INDEX IF NOT EXISTS organization_owner_transfers_from_idx ON app_v3.organization_owner_transfers (from_user_id);
CREATE INDEX IF NOT EXISTS organization_owner_transfers_to_idx ON app_v3.organization_owner_transfers (to_user_id);

CREATE TABLE IF NOT EXISTS app_v3.organizer_official_email_requests (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    organizer_id integer NOT NULL,
    requested_by_user_id uuid NOT NULL,
    new_email citext NOT NULL,
    token text NOT NULL UNIQUE,
    status app_v3."OrganizerEmailRequestStatus" NOT NULL DEFAULT 'PENDING',
    expires_at timestamptz NOT NULL,
    created_at timestamptz DEFAULT now(),
    confirmed_at timestamptz,
    cancelled_at timestamptz,
    CONSTRAINT organizer_official_email_requests_org_fkey FOREIGN KEY (organizer_id) REFERENCES app_v3.organizers(id) ON DELETE
CASCADE
);
CREATE INDEX IF NOT EXISTS organizer_official_email_requests_org_idx ON app_v3.organizer_official_email_requests (organizer_id);
CREATE INDEX IF NOT EXISTS organizer_official_email_requests_requestor_idx ON app_v3.organizer_official_email_requests
(requested_by_user_id);

CREATE TABLE IF NOT EXISTS app_v3.organization_audit_logs (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    organizer_id integer NOT NULL,
    actor_user_id uuid,
    action text NOT NULL,
    from_user_id uuid,
    to_user_id uuid,
    metadata jsonb DEFAULT '{}'::jsonb,
    ip text,
    user_agent text,
    created_at timestamptz DEFAULT now(),
    CONSTRAINT organization_audit_logs_organizer_fkey FOREIGN KEY (organizer_id) REFERENCES app_v3.organizers(id) ON DELETE
CASCADE
);
CREATE INDEX IF NOT EXISTS organization_audit_logs_org_idx ON app_v3.organization_audit_logs (organizer_id);
CREATE INDEX IF NOT EXISTS organization_audit_logs_actor_idx ON app_v3.organization_audit_logs (actor_user_id);

CREATE TABLE IF NOT EXISTS app_v3.ticket_checkins (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    ticket_id text NOT NULL,
    event_id integer NOT NULL,
    staff_user_id uuid,
    device_id text,
    created_at timestamptz DEFAULT now(),
    CONSTRAINT ticket_checkins_ticket_fkey FOREIGN KEY (ticket_id) REFERENCES app_v3.tickets(id) ON DELETE CASCADE,
    CONSTRAINT ticket_checkins_event_fkey FOREIGN KEY (event_id) REFERENCES app_v3.events(id) ON DELETE CASCADE
);
CREATE INDEX IF NOT EXISTS ticket_checkins_ticket_idx ON app_v3.ticket_checkins (ticket_id);
CREATE INDEX IF NOT EXISTS ticket_checkins_event_idx ON app_v3.ticket_checkins (event_id);
CREATE INDEX IF NOT EXISTS ticket_checkins_staff_idx ON app_v3.ticket_checkins (staff_user_id);

```

prisma/migrations/20250515120000_purchase_anchor_observability/migration.sql

```

DO $$
BEGIN
    CREATE TYPE app_v3."PaymentEventSource" AS ENUM ('WEBHOOK', 'JOB', 'API');

```

```

EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

ALTER TABLE app_v3.payment_events
  ADD COLUMN IF NOT EXISTS stripe_event_id text,
  ADD COLUMN IF NOT EXISTS purchase_id uuid,
  ADD COLUMN IF NOT EXISTS source app_v3."PaymentEventSource" NOT NULL DEFAULT 'WEBHOOK',
  ADD COLUMN IF NOT EXISTS dedupe_key text,
  ADD COLUMN IF NOT EXISTS attempt integer NOT NULL DEFAULT 1;

CREATE UNIQUE INDEX IF NOT EXISTS payment_events_stripe_event_id_key
  ON app_v3.payment_events(stripe_event_id)
  WHERE stripe_event_id IS NOT NULL;

CREATE UNIQUE INDEX IF NOT EXISTS payment_events_purchase_id_key
  ON app_v3.payment_events(purchase_id)
  WHERE purchase_id IS NOT NULL;

CREATE INDEX IF NOT EXISTS payment_events_dedupe_key_idx
  ON app_v3.payment_events(dedupe_key);

ALTER TABLE app_v3.sale_summaries
  ADD COLUMN IF NOT EXISTS purchase_id uuid;

CREATE UNIQUE INDEX IF NOT EXISTS sale_summaries_purchase_id_key
  ON app_v3.sale_summaries(purchase_id)
  WHERE purchase_id IS NOT NULL;

```

[prisma/migrations/20250520120000_padel_pairing_state_machine/migration.sql](#)

```

-- Add gender enum and field to profiles
DO $$ 
BEGIN
  CREATE TYPE app_v3."Gender" AS ENUM ('MALE','FEMALE');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

ALTER TABLE app_v3.profiles
  ADD COLUMN IF NOT EXISTS gender app_v3."Gender";

-- Eligibility type for padel tournaments
DO $$ 
BEGIN
  CREATE TYPE app_v3."PadelEligibilityType" AS ENUM ('OPEN','MALE_ONLY','FEMALE_ONLY','MIXED');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

ALTER TABLE app_v3.padel_tournament_configs
  ADD COLUMN IF NOT EXISTS eligibility_type app_v3."PadelEligibilityType" NOT NULL DEFAULT 'OPEN';

-- Lifecycle/join mode enums for pairings
DO $$ 
BEGIN
  CREATE TYPE app_v3."PadelPairingLifecycleStatus" AS ENUM
('PENDING_ONE_PAID','PENDING_PARTNER_PAYMENT','CONFIRMED_BOTH_PAID','CONFIRMED_CAPTAIN_FULL','CANCELLED_INCOMPLETE');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

DO $$ 
BEGIN
  CREATE TYPE app_v3."PadelPairingJoinMode" AS ENUM ('INVITE_PARTNER','LOOKING_FOR_PARTNER');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

ALTER TABLE app_v3.padel_pairings
  ADD COLUMN IF NOT EXISTS player1_user_id uuid,
  ADD COLUMN IF NOT EXISTS player2_user_id uuid,
  ADD COLUMN IF NOT EXISTS lifecycle_status app_v3."PadelPairingLifecycleStatus" NOT NULL DEFAULT 'PENDING_ONE_PAID',
  ADD COLUMN IF NOT EXISTS pairing_join_mode app_v3."PadelPairingJoinMode" NOT NULL DEFAULT 'INVITE_PARTNER';

CREATE INDEX IF NOT EXISTS padel_pairings_player1_idx ON app_v3.padel_pairings(player1_user_id);
CREATE INDEX IF NOT EXISTS padel_pairings_player2_idx ON app_v3.padel_pairings(player2_user_id);

-- Partial unique indexes to enforce one active pairing per event+user (active = lifecycle_status <> CANCELLED_INCOMPLETE)
DO $$ 
BEGIN

```

```

CREATE UNIQUE INDEX padel_pairings_event_player1_active_idx
    ON app_v3.padel_pairings(event_id, player1_user_id)
    WHERE lifecycle_status <> 'CANCELLED_INCOMPLETE' AND player1_user_id IS NOT NULL;
EXCEPTION WHEN duplicate_table THEN NULL;
END$$;

DO $$
BEGIN
    CREATE UNIQUE INDEX padel_pairings_event_player2_active_idx
        ON app_v3.padel_pairings(event_id, player2_user_id)
        WHERE lifecycle_status <> 'CANCELLED_INCOMPLETE' AND player2_user_id IS NOT NULL;
EXCEPTION WHEN duplicate_table THEN NULL;
END$$;

```

prisma/migrations/20250520131000_padel_pairing_guarantee_deadlines/migration.sql

```

-- Enum for guarantee status
DO $$
BEGIN
    CREATE TYPE app_v3."PadelPairingGuaranteeStatus" AS ENUM
    ('NONE','ARMED','SCHEDULED','SUCCEEDED','REQUIRES_ACTION','FAILED','EXPIRED');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

-- New nullable columns for pairing metadata (prod-safe: all nullable, no defaults that would lock heavily except
guarantee_status default)
ALTER TABLE app_v3.padel_pairings
    ADD COLUMN IF NOT EXISTS player2_identity_id uuid,
    ADD COLUMN IF NOT EXISTS partner_invite_used_at timestamptz,
    ADD COLUMN IF NOT EXISTS partner_link_token text,
    ADD COLUMN IF NOT EXISTS deadline_at timestamptz,
    ADD COLUMN IF NOT EXISTS partner_swap_allowed_until_at timestamptz,
    ADD COLUMN IF NOT EXISTS grace_until_at timestamptz,
    ADD COLUMN IF NOT EXISTS partner_invited_at timestamptz,
    ADD COLUMN IF NOT EXISTS partner_accepted_at timestamptz,
    ADD COLUMN IF NOT EXISTS partner_paid_at timestamptz,
    ADD COLUMN IF NOT EXISTS captain_second_charged_at timestamptz,
    ADD COLUMN IF NOT EXISTS guarantee_status app_v3."PadelPairingGuaranteeStatus" NOT NULL DEFAULT 'NONE',
    ADD COLUMN IF NOT EXISTS setup_intent_id text,
    ADD COLUMN IF NOT EXISTS payment_method_id text,
    ADD COLUMN IF NOT EXISTS second_charge_payment_intent_id text,
    ADD COLUMN IF NOT EXISTS captain_consent_at timestamptz,
    ADD COLUMN IF NOT EXISTS captain_first_sale_id integer,
    ADD COLUMN IF NOT EXISTS partner_sale_id integer,
    ADD COLUMN IF NOT EXISTS captain_second_sale_id integer;

-- partner_link_expires_at reused existing column invite_expires_at via @map; no DDL change needed
-- partner_invite_token reused existing invite_token via @map; no DDL change needed

-- Ensure partial index on player2 respects NULL semantics (allow many NULLs, enforce when NOT NULL)
DO $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM pg_indexes WHERE schemaname = 'app_v3' AND indexname = 'padel_pairings_event_player2_active_idx'
    ) THEN
        EXECUTE 'DROP INDEX IF EXISTS app_v3.padel_pairings_event_player2_active_idx';
    END IF;
END$$;

DO $$
BEGIN
    CREATE UNIQUE INDEX padel_pairings_event_player2_active_idx
        ON app_v3.padel_pairings(event_id, player2_user_id)
        WHERE lifecycle_status <> 'CANCELLED_INCOMPLETE' AND player2_user_id IS NOT NULL;
EXCEPTION WHEN duplicate_table THEN NULL;
END$$;

```

prisma/migrations/20250521120000_padel_pairing_holds/migration.sql

```

-- Holds para pairings (anti-oversell)
DO $$
BEGIN
    CREATE TYPE app_v3."PadelPairingHoldStatus" AS ENUM ('ACTIVE','CANCELLED','EXPIRED');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

```

```

END$$;

CREATE TABLE IF NOT EXISTS app_v3.padel_pairing_holds (
    id serial PRIMARY KEY,
    pairing_id integer NOT NULL,
    event_id integer NOT NULL,
    holds integer NOT NULL DEFAULT 2,
    status app_v3."PadelPairingHoldStatus" NOT NULL DEFAULT 'ACTIVE',
    expires_at timestamptz,
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now(),
    CONSTRAINT padel_pairing_holds_pairing_fk FOREIGN KEY (pairing_id) REFERENCES app_v3.padel_pairings(id) ON DELETE CASCADE
);

-- índice para lookup
CREATE INDEX IF NOT EXISTS padel_pairing_holds_pairing_idx ON app_v3.padel_pairing_holds(pairing_id);
CREATE INDEX IF NOT EXISTS padel_pairing_holds_event_idx ON app_v3.padel_pairing_holds(event_id);

-- Apenas um hold ACTIVE por pairing (permite múltiplos expirados/cancelados)
DO $$
BEGIN
    CREATE UNIQUE INDEX padel_pairing_holds_active_unique ON app_v3.padel_pairing_holds(pairing_id) WHERE status = 'ACTIVE';
EXCEPTION WHEN duplicate_table THEN NULL;
END$$;

```

prisma/migrations/20250521154000_tournament_entries/migration.sql

```

-- Enum para torneios
DO $$
BEGIN
    CREATE TYPE app_v3."TournamentEntryRole" AS ENUM ('CAPTAIN', 'PARTNER');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

DO $$
BEGIN
    CREATE TYPE app_v3."TournamentEntryStatus" AS ENUM ('PENDING', 'CONFIRMED', 'CANCELLED');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

-- Tabela tournament_entries
CREATE TABLE IF NOT EXISTS app_v3.tournament_entries (
    id serial PRIMARY KEY,
    event_id integer NOT NULL,
    user_id uuid NOT NULL,
    pairing_id integer,
    role app_v3."TournamentEntryRole" NOT NULL DEFAULT 'PARTNER',
    status app_v3."TournamentEntryStatus" NOT NULL DEFAULT 'PENDING',
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now()
);

-- FKs
ALTER TABLE app_v3.tournament_entries
    ADD CONSTRAINT tournament_entries_event_fk FOREIGN KEY (event_id) REFERENCES app_v3.events(id) ON DELETE CASCADE;
ALTER TABLE app_v3.tournament_entries
    ADD CONSTRAINT tournament_entries_user_fk FOREIGN KEY (user_id) REFERENCES app_v3.profiles(id) ON DELETE CASCADE;
ALTER TABLE app_v3.tournament_entries
    ADD CONSTRAINT tournament_entries_pairing_fk FOREIGN KEY (pairing_id) REFERENCES app_v3.padel_pairings(id) ON DELETE CASCADE;

-- Índices
CREATE UNIQUE INDEX IF NOT EXISTS tournament_entries_event_user_unique ON app_v3.tournament_entries(event_id, user_id);
CREATE INDEX IF NOT EXISTS tournament_entries_pairing_idx ON app_v3.tournament_entries(pairing_id);

```

prisma/migrations/20250521180000_ownership_tournament_links/migration.sql

```

-- Email identities
CREATE TABLE IF NOT EXISTS app_v3.email_identities (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    email_normalized citext UNIQUE NOT NULL,
    email_verified_at timestamptz,
    user_id uuid,
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now(),

```

```

CONSTRAINT email_identities_user_fk FOREIGN KEY (user_id) REFERENCES app_v3.profiles(id) ON DELETE SET NULL
);

-- Ownership columns
ALTER TABLE app_v3.tickets
ADD COLUMN IF NOT EXISTS owner_user_id uuid,
ADD COLUMN IF NOT EXISTS owner_identity_id uuid,
ADD COLUMN IF NOT EXISTS tournament_entry_id integer;

ALTER TABLE app_v3.sale_summaries
ADD COLUMN IF NOT EXISTS owner_user_id uuid,
ADD COLUMN IF NOT EXISTS owner_identity_id uuid;

ALTER TABLE app_v3.tournament_entries
ADD COLUMN IF NOT EXISTS owner_user_id uuid,
ADD COLUMN IF NOT EXISTS owner_identity_id uuid;

-- FKs for ownership (soft, no cascade)
ALTER TABLE app_v3.tickets
ADD CONSTRAINT tickets_owner_user_fk FOREIGN KEY (owner_user_id) REFERENCES app_v3.profiles(id) ON DELETE SET NULL,
ADD CONSTRAINT tickets_owner_identity_fk FOREIGN KEY (owner_identity_id) REFERENCES app_v3.email_identities(id) ON DELETE SET NULL;

ALTER TABLE app_v3.sale_summaries
ADD CONSTRAINT sale_summaries_owner_user_fk FOREIGN KEY (owner_user_id) REFERENCES app_v3.profiles(id) ON DELETE SET NULL,
ADD CONSTRAINT sale_summaries_owner_identity_fk FOREIGN KEY (owner_identity_id) REFERENCES app_v3.email_identities(id) ON DELETE SET NULL;

ALTER TABLE app_v3.tournament_entries
ADD CONSTRAINT tournament_entries_owner_user_fk FOREIGN KEY (owner_user_id) REFERENCES app_v3.profiles(id) ON DELETE SET NULL,
ADD CONSTRAINT tournament_entries_owner_identity_fk FOREIGN KEY (owner_identity_id) REFERENCES app_v3.email_identities(id) ON DELETE SET NULL;

-- FK ticket -> tournament_entries
ALTER TABLE app_v3.tickets
ADD CONSTRAINT tickets_tournament_entry_fk FOREIGN KEY (tournament_entry_id) REFERENCES app_v3.tournament_entries(id) ON DELETE SET NULL;

CREATE INDEX IF NOT EXISTS tickets_tournament_entry_idx ON app_v3.tickets(tournament_entry_id);

-- Check constraints: only one owner set
DO $$ 
BEGIN
    ALTER TABLE app_v3.tickets
    ADD CONSTRAINT tickets_owner_exclusive_chk CHECK (NOT (owner_user_id IS NOT NULL AND owner_identity_id IS NOT NULL));
EXCEPTION WHEN duplicate_object THEN NULL; END$$;

DO $$ 
BEGIN
    ALTER TABLE app_v3.sale_summaries
    ADD CONSTRAINT sale_summaries_owner_exclusive_chk CHECK (NOT (owner_user_id IS NOT NULL AND owner_identity_id IS NOT NULL));
EXCEPTION WHEN duplicate_object THEN NULL; END$$;

DO $$ 
BEGIN
    ALTER TABLE app_v3.tournament_entries
    ADD CONSTRAINT tournament_entries_owner_exclusive_chk CHECK (NOT (owner_user_id IS NOT NULL AND owner_identity_id IS NOT NULL));
EXCEPTION WHEN duplicate_object THEN NULL; END$$;

```

prisma/migrations/20250521181000_email_identity_claim/migration.sql

```

-- Table email_identities
CREATE TABLE IF NOT EXISTS app_v3.email_identities (
    id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
    email_normalized citext UNIQUE NOT NULL,
    email_verified_at timestamptz,
    user_id uuid,
    created_at timestamptz NOT NULL DEFAULT now(),
    updated_at timestamptz NOT NULL DEFAULT now(),
    CONSTRAINT email_identities_user_fk FOREIGN KEY (user_id) REFERENCES app_v3.profiles(id) ON DELETE SET NULL
);

-- Ownership columns (safety: only add if missing)
ALTER TABLE app_v3.tickets
ADD COLUMN IF NOT EXISTS owner_user_id uuid,

```

```

ADD COLUMN IF NOT EXISTS owner_identity_id uuid,
ADD COLUMN IF NOT EXISTS tournament_entry_id integer;

ALTER TABLE app_v3.sale_summaries
ADD COLUMN IF NOT EXISTS owner_user_id uuid,
ADD COLUMN IF NOT EXISTS owner_identity_id uuid;

ALTER TABLE app_v3.tournament_entries
ADD COLUMN IF NOT EXISTS owner_user_id uuid,
ADD COLUMN IF NOT EXISTS owner_identity_id uuid;

-- Fks
ALTER TABLE app_v3.tickets
ADD CONSTRAINT tickets_owner_identity_fk FOREIGN KEY (owner_identity_id) REFERENCES app_v3.email_identities(id) ON DELETE SET NULL;

ALTER TABLE app_v3.sale_summaries
ADD CONSTRAINT sale_summaries_owner_user_fk FOREIGN KEY (owner_user_id) REFERENCES app_v3.profiles(id) ON DELETE SET NULL,
ADD CONSTRAINT sale_summaries_owner_identity_fk FOREIGN KEY (owner_identity_id) REFERENCES app_v3.email_identities(id) ON DELETE SET NULL;

ALTER TABLE app_v3.tournament_entries
ADD CONSTRAINT tournament_entries_owner_user_fk FOREIGN KEY (owner_user_id) REFERENCES app_v3.profiles(id) ON DELETE SET NULL,
ADD CONSTRAINT tournament_entries_owner_identity_fk FOREIGN KEY (owner_identity_id) REFERENCES app_v3.email_identities(id) ON DELETE SET NULL;

-- ticket -> tournament_entry link
ALTER TABLE app_v3.tickets
ADD CONSTRAINT tickets_tournament_entry_fk FOREIGN KEY (tournament_entry_id) REFERENCES app_v3.tournament_entries(id) ON DELETE SET NULL;

CREATE INDEX IF NOT EXISTS tickets_tournament_entry_idx ON app_v3.tickets(tournament_entry_id);

-- Exclusividade de owner (um ou outro)
DO $$ 
BEGIN
    ALTER TABLE app_v3.tickets
    ADD CONSTRAINT tickets_owner_exclusive_chk CHECK (NOT (owner_user_id IS NOT NULL AND owner_identity_id IS NOT NULL));
EXCEPTION WHEN duplicate_object THEN NULL; END$$;

DO $$ 
BEGIN
    ALTER TABLE app_v3.sale_summaries
    ADD CONSTRAINT sale_summaries_owner_exclusive_chk CHECK (NOT (owner_user_id IS NOT NULL AND owner_identity_id IS NOT NULL));
EXCEPTION WHEN duplicate_object THEN NULL; END$$;

DO $$ 
BEGIN
    ALTER TABLE app_v3.tournament_entries
    ADD CONSTRAINT tournament_entries_owner_exclusive_chk CHECK (NOT (owner_user_id IS NOT NULL AND owner_identity_id IS NOT NULL));
EXCEPTION WHEN duplicate_object THEN NULL; END$$;

-- Re-add tickets_owner_user_fk only if missing
DO $$ 
BEGIN
    IF NOT EXISTS (
        SELECT 1 FROM pg_constraint WHERE conname = 'tickets_owner_user_fk'
    ) THEN
        ALTER TABLE app_v3.tickets
        ADD CONSTRAINT tickets_owner_user_fk FOREIGN KEY (owner_user_id) REFERENCES app_v3.profiles(id) ON DELETE SET NULL;
    END IF;
END$$;

```

prisma/migrations/20250526001_tournament_structure/migration.sql

```

-- Fase 4: estrutura base de torneios (Tournament/Stage/Group/Match)
-- Apenas operações aditivas (sem DROP/RENAME destrutivo).

SET search_path TO app_v3, public;

-- Enums
CREATE TYPE app_v3."TournamentFormat" AS ENUM (
    'GROUPS_PLUS_PLAYOFF',
    'DRAW_A_B',
    'GROUPS_PLUS_FINALS_ALL_PLACES',

```

```

'CHAMPIONSHIP_ROUND_ROBIN',
'NONSTOP_ROUND_ROBIN',
'MANUAL'
);

CREATE TYPE app_v3."TournamentStageType" AS ENUM (
    'GROUPS',
    'PLAYOFF',
    'CONSOLATION',
    'NONSTOP'
);

CREATE TYPE app_v3."TournamentMatchStatus" AS ENUM (
    'PENDING',
    'SCHEDULED',
    'IN_PROGRESS',
    'DONE',
    'CANCELLED'
);

-- Table: tournaments
CREATE TABLE app_v3."tournaments" (
    "id" SERIAL PRIMARY KEY,
    "event_id" INTEGER NOT NULL UNIQUE,
    "format" app_v3."TournamentFormat" NOT NULL,
    "generation_seed" TEXT,
    "inscription_deadline_at" TIMESTAMPTZ(6),
    "tie_break_rules" JSONB NOT NULL DEFAULT '{}':jsonb,
    "config" JSONB NOT NULL DEFAULT '{}':jsonb,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT "tournaments_event_id_fkey" FOREIGN KEY ("event_id") REFERENCES app_v3."events"("id") ON DELETE CASCADE ON UPDATE CASCADE
);
;

-- Table: tournament_stages
CREATE TABLE app_v3."tournament_stages" (
    "id" SERIAL PRIMARY KEY,
    "tournament_id" INTEGER NOT NULL,
    "name" TEXT,
    "stage_type" app_v3."TournamentStageType" NOT NULL DEFAULT 'GROUPS',
    "order" INTEGER NOT NULL DEFAULT 0,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT "tournament_stages_tournament_id_fkey" FOREIGN KEY ("tournament_id") REFERENCES app_v3."tournaments"("id") ON DELETE CASCADE ON UPDATE CASCADE
);
;
CREATE INDEX "tournament_stages_tournament_id_idx" ON app_v3."tournament_stages"("tournament_id");

-- Table: tournament_groups
CREATE TABLE app_v3."tournament_groups" (
    "id" SERIAL PRIMARY KEY,
    "stage_id" INTEGER NOT NULL,
    "name" TEXT NOT NULL,
    "order" INTEGER NOT NULL DEFAULT 0,
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT "tournament_groups_stage_id_fkey" FOREIGN KEY ("stage_id") REFERENCES app_v3."tournament_stages"("id") ON DELETE CASCADE ON UPDATE CASCADE
);
;
CREATE INDEX "tournament_groups_stage_id_idx" ON app_v3."tournament_groups"("stage_id");

-- Table: tournament_matches
CREATE TABLE app_v3."tournament_matches" (
    "id" SERIAL PRIMARY KEY,
    "stage_id" INTEGER NOT NULL,
    "group_id" INTEGER,
    "pairing1_id" INTEGER,
    "pairing2_id" INTEGER,
    "round_number" INTEGER,
    "round_label" TEXT,
    "next_match_id" INTEGER,
    "next_slot" INTEGER,
    "court_id" INTEGER,
    "start_at" TIMESTAMPTZ(6),
    "score" JSONB NOT NULL DEFAULT '{}':jsonb,
    "status" app_v3."TournamentMatchStatus" NOT NULL DEFAULT 'PENDING',
    "created_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP
);
;
```

```

"updated_at" TIMESTAMP(3) NOT NULL DEFAULT CURRENT_TIMESTAMP,
CONSTRAINT "tournament_matches_stage_id_fkey" FOREIGN KEY ("stage_id") REFERENCES app_v3."tournament_stages"("id") ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT "tournament_matches_group_id_fkey" FOREIGN KEY ("group_id") REFERENCES app_v3."tournament_groups"("id") ON DELETE SET NULL ON UPDATE CASCADE
);
CREATE INDEX "tournament_matches_stage_id_idx" ON app_v3."tournament_matches"("stage_id");
CREATE INDEX "tournament_matches_group_id_idx" ON app_v3."tournament_matches"("group_id");

```

prisma/migrations/20250528090000_tournament_generation_audit/migration.sql

```

-- Safe additions for tournament generation metadata + audit log
DO $$
BEGIN
IF NOT EXISTS (
    SELECT 1 FROM information_schema.columns
    WHERE table_schema = 'app_v3' AND table_name = 'tournaments' AND column_name = 'generated_at'
) THEN
    ALTER TABLE app_v3.tournaments ADD COLUMN generated_at timestamp;
END IF;

IF NOT EXISTS (
    SELECT 1 FROM information_schema.columns
    WHERE table_schema = 'app_v3' AND table_name = 'tournaments' AND column_name = 'generated_by_user_id'
) THEN
    ALTER TABLE app_v3.tournaments ADD COLUMN generated_by_user_id uuid REFERENCES app_v3.profiles(id) ON UPDATE NO ACTION;
    CREATE INDEX IF NOT EXISTS tournaments_generated_by_user_idx ON app_v3.tournaments (generated_by_user_id);
END IF;
END $$;

-- Audit log table
DO $$
BEGIN
IF NOT EXISTS (
    SELECT 1 FROM information_schema.tables
    WHERE table_schema = 'app_v3' AND table_name = 'tournament_audit_logs'
) THEN
    CREATE TABLE app_v3.tournament_audit_logs (
        id serial PRIMARY KEY,
        tournament_id integer NOT NULL REFERENCES app_v3.tournaments(id) ON DELETE CASCADE,
        user_id uuid REFERENCES app_v3.profiles(id) ON UPDATE NO ACTION,
        action text NOT NULL,
        payload_before jsonb,
        payload_after jsonb,
        created_at timestamp(6) DEFAULT now()
    );
END IF;
END $$;

CREATE INDEX IF NOT EXISTS tournament_audit_logs_tournament_idx ON app_v3.tournament_audit_logs (tournament_id);
CREATE INDEX IF NOT EXISTS tournament_audit_logs_user_idx ON app_v3.tournament_audit_logs (user_id);

```

prisma/migrations/20250529120000_match_notifications/migration.sql

```

DO $$
BEGIN
IF NOT EXISTS (
    SELECT 1 FROM information_schema.tables
    WHERE table_schema = 'app_v3' AND table_name = 'match_notifications'
) THEN
    CREATE TABLE app_v3.match_notifications (
        id serial PRIMARY KEY,
        match_id integer NOT NULL,
        dedupe_key text NOT NULL,
        created_at timestamp(6) DEFAULT now(),
        payload jsonb,
        UNIQUE (dedupe_key)
    );
    CREATE INDEX match_notifications_match_idx ON app_v3.match_notifications (match_id);
END IF;
END $$;

```

prisma/migrations/20250601090000_sale_summary_status/migration.sql

```
DO $$  
BEGIN  
    IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'SaleSummaryStatus' AND typnamespace = (SELECT oid FROM pg_namespace WHERE nspname='app_v3')) THEN  
        CREATE TYPE app_v3."SaleSummaryStatus" AS ENUM ('PAID','REFUNDED','DISPUTED','FAILED');  
    END IF;  
    IF NOT EXISTS (  
        SELECT 1 FROM information_schema.columns  
        WHERE table_schema='app_v3' AND table_name='sale_summaries' AND column_name='status'  
    ) THEN  
        ALTER TABLE app_v3.sale_summaries ADD COLUMN status app_v3."SaleSummaryStatus" DEFAULT 'PAID';  
    END IF;  
END $$;
```

prisma/migrations/20250601100000_notification_outbox/migration.sql

```
DO $$  
BEGIN  
    IF NOT EXISTS (  
        SELECT 1 FROM information_schema.tables  
        WHERE table_schema='app_v3' AND table_name='notification_outbox'  
    ) THEN  
        CREATE TABLE app_v3.notification_outbox (  
            id uuid PRIMARY KEY DEFAULT gen_random_uuid(),  
            user_id uuid NULL,  
            notification_type text NOT NULL,  
            template_version text NULL,  
            dedupe_key text NOT NULL,  
            status text NOT NULL DEFAULT 'PENDING',  
            payload jsonb NOT NULL DEFAULT '{}':jsonb,  
            retries integer NOT NULL DEFAULT 0,  
            last_error text NULL,  
            created_at timestamp(6) NOT NULL DEFAULT now(),  
            sent_at timestamp(6) NULL  
        );  
        CREATE UNIQUE INDEX notification_outbox_dedupe_idx ON app_v3.notification_outbox (dedupe_key);  
        CREATE INDEX notification_outbox_status_idx ON app_v3.notification_outbox (status);  
        CREATE INDEX notification_outbox_user_idx ON app_v3.notification_outbox (user_id);  
    END IF;  
END $$;
```

prisma/prisma/dev.db

```
SQLite format 3 @ 0% 0.zp 0 0  
0 0  
y 0"0 0 G/0)indexTicketReservation_eventId_idxTicketReservation CREATE INDEX "TicketReservation_eventId_idx" ON  
"TicketReservation"("eventId")0 I/0-indexTicketReservation_ticketId_idxTicketReservation  
CREATE INDEX "TicketReservation_ticketId_idx" ON "TicketReservation"  
("ticketId")0 //0tableTicketReservationTicketReservation CREATE TABLE "TicketReservation" (  
    "id" TEXT NOT NULL PRIMARY KEY,  
    "ticketId" TEXT NOT NULL,  
    "eventId" INTEGER NOT NULL,  
    "userId" TEXT,  
    "quantity" INTEGER NOT NULL,  
    "status" TEXT NOT NULL DEFAULT 'ACTIVE',  
    "expiresAt" DATETIME NOT NULL,  
    "createdAt" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    "updatedAt" DATETIME NOT NULL,  
    CONSTRAINT "TicketReservation_ticketId_fkey" FOREIGN KEY ("ticketId") REFERENCES "Ticket" ("id") ON DELETE CASCADE ON UPDATE  
CASCADE,  
    CONSTRAINT "TicketReservation_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "Event" ("id") ON DELETE CASCADE ON UPDATE  
CASCADE  
)A U/ indexesqlite_autoindex_TicketReservation_1TicketReservation@d 0'tableEventEvent CREATE TABLE "Event" (  
    "id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,  
    "slug" TEXT NOT NULL,  
    "title" TEXT NOT NULL,  
    "description" TEXT NOT NULL,  
    "startDate" DATETIME NOT NULL,  
    "endDate" DATETIME NOT NULL,  
    "locationName" TEXT NOT NULL,  
    "address" TEXT NOT NULL,
```

```

"isFree" BOOLEAN NOT NULL DEFAULT false,
"basePrice" INTEGER,
"timezone" TEXT NOT NULL DEFAULT 'Europe/Lisbon',
"coverImageUrl" TEXT,
"organizerName" TEXT,
"organizerId" TEXT,
"createdAt" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updatedAt" DATETIME NOT NULL
)+

? indexsqlite_autoindex_Ticket_1TicketV ) {indexEvent_slug_keyEvent CREATE UNIQUE INDEX "Event_slug_key" ON "Event"
("slug")@q }) @tableTicketPurchaseTicketPurchase      CREATE TABLE "TicketPurchase" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "ticketId" TEXT NOT NULL,
    "eventId" INTEGER NOT NULL,
    "quantity" INTEGER NOT NULL,
    "pricePaid" INTEGER NOT NULL,
    "currency" TEXT NOT NULL DEFAULT 'EUR',
    "createdat" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, "userId" TEXT,
    CONSTRAINT "TicketPurchase_ticketId_fkey" FOREIGN KEY ("ticketId") REFERENCES "Ticket" ("id") ON DELETE CASCADE ON UPDATE
CASCADE,
    CONSTRAINT "TicketPurchase_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "Event" ("id") ON DELETE CASCADE ON UPDATE CASCADE
); 0) indexsqlite_autoindex_TicketPurchase_1TicketPurchase
@S  @tableTicketTicketCREATE TABLE "Ticket" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "eventId" INTEGER NOT NULL,
    "name" TEXT NOT NULL,
    "description" TEXT,
    "price" INTEGER NOT NULL,
    "currency" TEXT NOT NULL DEFAULT 'EUR',
    "available" BOOLEAN NOT NULL DEFAULT true,
    "isVisible" BOOLEAN NOT NULL DEFAULT true,
    "totalQuantity" INTEGER,
    "soldQuantity" INTEGER NOT NULL DEFAULT 0,
    "startsAt" DATETIME,
    "endsAt" DATETIME,
    "sortOrder" INTEGER NOT NULL DEFAULT 0,
    "createdat" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updatedAt" DATETIME NOT NULL,
    CONSTRAINT "Ticket_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "Event" ("id") ON DELETE CASCADE ON UPDATE CASCADE
)P ++ Ytablesqlite_sequencessqlite_sequence CREATE TABLE
sqlite_sequence(name,seq)C W1 indexsqlite_autoindex__prisma_migrations_1_prisma_migrations @Z 11 @_table_prisma_migrations_prisma_m:
TABLE "_prisma_migrations" (
    "id"          TEXT PRIMARY KEY NOT NULL,
    "checksum"    TEXT NOT NULL,
    "finished_at" DATETIME,
    "migration_name" TEXT NOT NULL,
    "logs"         TEXT,
    "rolled_back_at" DATETIME,
    "started_at"   DATETIME NOT NULL DEFAULT current_timestamp,
    "applied_steps_count" INTEGER UNSIGNED NOT NULL DEFAULT
)
{#1 @
@
L      @      @%
U@
c      8e2a0f39-8903-4694-ac35-
ad2ca88efacd6b61230bd582a68b0fb29688c0cc38bcf54e6db18fad685cebf1de4c1ec6c7b4 @@@ 20251119154902_add_stripe_payment_intent_id @@@ ?

U@
C      31bcea1f-4eef-4b8a-b893-
d1c356944610f4e3b36193f942d0934aa1e0826bf2cffc40058a6f93c18c32b50bd81e4dd466 @@@ 20251119110455_add_qr_token @@@
U@
O      d1fa7e81-4046-4348-b74e-
6b6050a5803322eedaaafc91f0919b16c8f7dde0c291604f2e545f6da4f5361bad77ebd362dd @@@ 20251118154908_add_event_interest @@@
U@
W      524df5b5-faf3-4a6e-a5f0-
8a0f41a3edef85a4bcec6dde679d87b57f34d71399cc4ecd8c524c4dbea8bfedd46377b5c4cc @@@ 20251117130224_add_ticket_reservation @@@
U@
I      d89b77a5-efec-4e57-bc5a-
6e60f0dc40a1a2ad2c68dcf44d63e738c6290e4e1bd66dc14e6eb8e33a48b02fe96718743b76 @@@ 20251115203927_add_organizer_id_to_event @@@
U@
S      7ddb3655-bb21-4f09-aa46-
981cec865b387a2a8789f745f0726c31f1ef8d16aa3b71fd15b9f62c070419efe95fd4951b91 @@@ 20251115183127_add_creator_to_event @@@
U@
G      493fdf35-b8eb-42bd-81ed-
f00e4b6c8e8dc2f24c975366565411662b380b3b7a669722cc7408d193aa7116bd8604c70d8f @@@ 20251115180904_add_user_id_to_ticket_purchase @@@
U@
O      e47d0558-d942-463f-98dc-

```

f010952afa014ad736e0e301a2ebc125888072cc9c7cf0301c545b96c87df90dce860d9e00eb 000 20251115073501_ticket_waves_stock 000 0
U0
0 ae406dd7-5031-42f9-8044-
09066b17ca043c91c4db4c9f8c4d1f804a79f41c9277ecc4b582c44e49ab0486c83eaddffdfa 000 20251115072435_ticket_waves_stock 000 0#
U0
- e734bc97-1e44-49b8-8c65-
1f17b03f07cc2d8f57aaa25c50f1b10594eea23492b5db87331f7f6cb2df6f704f8eff4a0ddc 000 20251115071940_ticket_waves_and_purchases 000
0
U0
Y 3557d53d-719c-4b91-b1cf-18aa7eeb72a974a2338697d69fd6229d42c79036baaed3d138ef8c3d1128ea23902105a70287 000
20251115065840_init_events_and_tickets 000

> g 0 0 > 0 0 0] 0(U 8e2a0f39-8903-4694-ac35-ad2ca88efacd (U 31bcea1f-4eef-4b8a-b893-d1c356944610
(U d1fa7e81-4046-4348-b74e-6b6050a58033 (U 524df5b5-faf3-4a6e-a5f0-8a0f41a3edef(U d89b77a5-efec-4e57-bc5a-
6e60f0dc40a1 (U 7ddb3655-bb21-4f09-aa46-981cec865b38 (U 493fdf35-b8eb-42bd-81ed-f00e4b6c8e8d (U e47d0558-d942-463f-98dc-
f010952fa01 (U ae406dd7-5031-42f9-8044-09066b17ca04 (U e734bc97-1e44-49b8-8c65-1f17b03f07cc ' U 3557d53d-719c-4b91-b1cf-
18aa7eeb72a9
0 0 0 + festival-bandas da
0 0 0 new_Event Event

K S
K0 ++# #
'0 U festival-bandasFestival BandasA ORYA prepara-se para levar a energia à Póvoa de Varzim com um novo conceito: um festival noturno de bandas ao vivo, na icónica Póvoa Arena.

Durante dois dias, várias bandas de covers sobem ao palco para disputar o título de melhor performance do festival, com votação do público ao vivo e um prémio financeiro para os vencedores.

Entre atuações, o ambiente não pára – haverá momentos de humor, street food e bebidas, criando uma atmosfera descontraída, perfeita para o início do verão.

- ☛ Bandas de covers (2 noites de música ao vivo)
- ☛ Cerveja a 3€ – bares e bancas espalhadas pela Arena
- ☛ Street food e espaços de convívio
- ☛ Humor e animação entre atuações
- ☛ Votação do público + prémio para a banda vencedora

Tudo isto num formato simples, noturno e cheio de boas vibrações.

⌚ Horário: das 21h00 às 24h00

📍 Local: Póvoa Arena, Póvoa de Varzim

📅 Data: a anunciar

A música, a energia e o público no centro – é isso que faz da ORYA muito mais do que uma app.

É uma experiência. 0000 000P Povoa Arena 0Europe/Lisbon/uploads/1763577113967-47df1a2208a2ecc7f8a8f3a90187304.png0RYA Team13e1b3a8-3beb-4419-8a98-86089860ffd0 00c/ 00c/ 0*
'0 U dadawd 000H 0 32~,Europe/Lisbon/uploads/1763552195461-a4ba48138328a072752c8dfbeac9efb.webp0RYA Team13e1b3a8-3beb-4419-8a98-86089860ffd0 0000 0000 0000
M 0 0 0 0 KM ? cmi6473gj0001lmr2kg95pzd1 ? cmi5yzre40005za6fr4uxx21h ? cmi5yhkg40001za6fiz27jhzc ? cmi5xi2oq000bzalhd24atdso ? cmi5xi2cmi5xg1bp0008zalh6xeqrtd
z 0 zA ? cmi6ca2he0000p59j6b9fiukr Wave 1 0EUR 00c/ 00c/ A ?
cmi5xfzdm0006zalhpz9wc25tWave 1~,EUR 0000 0000
n n?
?? UUcmi5yhq8p0003za6f6lvstnji cmi5xfzdm0006zalhpz9wc25t~,EUR 00 0 13e1b3a8-3beb-4419-8a98-86089860ffd03aba5489-4d24-44a8-ad2e-5f0bf9f86f4f
0 0 ? cmi5yhq8p0003za6f6lvstnji
0 0
0
0yx
?? U cmi6473gj0001lmr2kg95pzd1cmi5xfzdm0006zalhpz9wc25t13e1b3a8-3beb-4419-8a98-86089860ffd0ACTIVE 0000 000 C 000
Cz
?? U cmi5yzre40005za6fr4uxx21hcmi5xfzdm0006zalhpz9wc25t13e1b3a8-3beb-4419-8a98-86089860ffd0CANCELED 00 00 00 01 00 00z
?? U cmi5yhkg40001za6fiz27jhccmi5xfzdm0006zalhpz9wc25t13e1b3a8-3beb-4419-8a98-86089860ffd0CANCELED 00 00 00 00 00 00z

```

??      U      cmi5xi2oq000czalh8m60ug63cmi5xfzdm0006zalhpz9wc25t13e1b3a8-3beb-4419-8a98-86089860ffd0CANCELED 000s0 000s 000s0x?
U      cmi5xi2oq000bzalhd24atdsocmi5xfzdm0006zalhpz9wc25t13e1b3a8-3beb-4419-8a98-86089860ffd0ACTIVE 000s0 000s 000z
??      U      cmi5xi2oq000bzalhd24atdsocmi5xfzdm0006zalhpz9wc25t13e1b3a8-3beb-4419-8a98-86089860ffd0CANCELED 000z
000s 000z z
??      U      cmi5xg1bp0008zalh6xeqrtdcmi5xfzdm0006zalhpz9wc25t13e1b3a8-3beb-4419-8a98-86089860ffd0CANCELED 000 00000 000
000? ? cmi6ca2he0000p59j6b9fiukr ? cmi5xfzdm0006zalhpz9wc25t ? cmi5xfzdm0006zalhpz9wc25t ? cmi5xfzdm0006zalhpz9wc25t ? cmi5xfzdm0006zalhpz9wc25t
M 000? k M ? cmi5xfzdm0006zalhpz9wc25t ? cmi5xfzdm0006zalhpz9wc25t ? cmi5xfzdm0006zalhpz9wc25t ? cmi5xfzdm0006zalhpz9wc25t ? cmi5xfzdm0006zalhpz9wc25t
cmi5xfzdm0006zalhpz9wc25t
0000000
000] 4 ( U 13e1b3a8-3beb-4419-8a98-86089860ffd0 ( U 13e1b3a8-3beb-4419-8a98-86089860ffd0 ( U 13e1b3a8-3beb-4419-8a98-
86089860ffd0 ( U 13e1b3a8-3beb-4419-8a98-86089860ffd0 ( U 13e1b3a8-3beb-4419-8a98-86089860ffd0 ' U 13e1b3a8-3beb-4419-8a98-
86089860ffd0
00##
0
0
00
y 0"00 ~0 E/0%indexTicketReservation_userId_idxTicketReservation CREATE INDEX "TicketReservation_userId_idx" ON
"TicketReservation"("userId")0 G/0)indexTicketReservation_eventId_idxTicketReservation CREATE INDEX
"TicketReservation_eventId_idx" ON "TicketReservation"("eventId")0 I/0-indexTicketReservation_ticketId_idxTicketReservation
CREATE INDEX "TicketReservation_ticketId_idx" ON "TicketReservation"
("ticketId")0 //0tableTicketReservationTicketReservation CREATE TABLE "TicketReservation" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "ticketId" TEXT NOT NULL,
    "eventId" INTEGER NOT NULL,
    "userId" TEXT,
    "quantity" INTEGER NOT NULL,
    "status" TEXT NOT NULL DEFAULT 'ACTIVE',
    "expiresAt" DATETIME NOT NULL,
    "createdAt" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updatedAt" DATETIME NOT NULL,
    CONSTRAINT "TicketReservation_ticketId_fkey" FOREIGN KEY ("ticketId") REFERENCES "Ticket" ("id") ON DELETE CASCADE ON UPDATE
CASCADE,
    CONSTRAINT "TicketReservation_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "Event" ("id") ON DELETE CASCADE ON UPDATE
CASCADE
)A U/ indexessqlite_autoindex_TicketReservation_1TicketReservation 00d 0'tableEventEvent CREATE TABLE "Event" (
    "id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    "slug" TEXT NOT NULL,
    "title" TEXT NOT NULL,
    "description" TEXT NOT NULL,
    "startDate" DATETIME NOT NULL,
    "endDate" DATETIME NOT NULL,
    "locationName" TEXT NOT NULL,
    "address" TEXT NOT NULL,
    "isFree" BOOLEAN NOT NULL DEFAULT false,
    "basePrice" INTEGER,
    "timezone" TEXT NOT NULL DEFAULT 'Europe/Lisbon',
    "coverImageUrl" TEXT,
    "organizerName" TEXT,
    "organizerId" TEXT,
    "createdAt" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updatedAt" DATETIME NOT NULL
)
+
? indexssqlite_autoindex_Ticket_1Ticket*) {indexEvent_slug_keyEvent CREATE UN0 } )0tableTicketPurchaseTicketPurchase CREATE
TABLE "TicketPurchase" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "ticketId" TEXT NOT NULL,
    "eventId" INTEGER NOT NULL,
    "quantity" INTEGER NOT NULL,
    "pricePaid" INTEGER NOT NULL,
    "currency" TEXT NOT NULL DEFAULT 'EUR',
    "createdAt" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, "userId" TEXT, "qrToken" TEXT, "stripePaymentIntentId" TEXT,
    CONSTRAINT "TicketPurchase_ticketId_fkey" FOREIGN KEY ("ticketId") REFERENCES "Ticket" ("id") ON DELETE CASCADE ON UPDATE
CASCADE,
    CONSTRAINT "TicketPurchase_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "Event" ("id") ON DELETE CASCADE ON UPDATE CASCADE
); 0) indexssqlite_autoindex_TicketPurchase_1TicketPurchase
0S 0'tableTicketTicketCREATE TABLE "Ticket" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "eventId" INTEGER NOT NULL,
    "name" TEXT NOT NULL,
    "description" TEXT,
    "price" INTEGER NOT NULL,
    "currency" TEXT NOT NULL DEFAULT 'EUR',
    "available" BOOLEAN NOT NULL DEFAULT true,
    "isVisible" BOOLEAN NOT NULL DEFAULT true,
    "totalQuantity" INTEGER,
    "soldQuantity" INTEGER NOT NULL DEFAULT 0,
    "startsAt" DATETIME,
    "endsAt" DATETIME,

```

```

"sortOrder" INTEGER NOT NULL DEFAULT 0,
"createdAt" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
"updatedAt" DATETIME NOT NULL,
CONSTRAINT "Ticket_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "Event" ("id") ON DELETE CASCADE ON UPDATE CASCADE
)P ++ Ytablessqlite_sequencessqlite_sequence CREATE TABLE
sqlite_sequence(name,seq)C W1 indexsqlite_autoindex__prisma_migrations_1_prisma_migrations @Z 11 @_table_prisma_migrations_prisma_m:
TABLE "_prisma_migrations" (
    "id" TEXT PRIMARY KEY NOT NULL,
    "checksum" TEXT NOT NULL,
    "finished_at" DATETIME,
    "migration_name" TEXT NOT NULL,
    "logs" TEXT,
    "rolled_back_at" DATETIME,
    "started_at" DATETIME NOT NULL DEFAULT current_timestamp,
    "applied_steps_count" INTEGER UNSIGNED NOT NULL DEFAULT 0
)
@ {{000000 ACTIVE 00000 CANCELED 00 00 CANCELED 00 00 CANCELED 000s0 ACTIVE 000 CANCELED 000z
CANCELED 000
}
@

@
A
@

R      70 X00E0
@. 1) @cindexTicketPurchase_stripePaymentIntentId_keyTicketPurchase CREATE UNIQUE INDEX "TicketPurchase_stripePaymentIntentId_key"
ON "TicketPurchase"("stripePaymentIntentId")@ A) @cindexTicketPurchase_qrToken_keyTicketPurchase CREATE UNIQUE INDEX
"TicketPurchase_qrToken_key" ON "TicketPurchase"("qrToken")@ M' @lindexEventInterest_eventId_userId_keyEventInterest CREATE UNIQUE
INDEX "EventInterest_eventId_userId_key" ON "EventInterest"("eventId",
"userId")v = '@indexEventInterest_userId_idxEventInterest CREATE INDEX "EventInterest_userId_idx" ON "EventInterest"
("userId")@f '' @_tableEventInterestEventInterest CREATE TABLE "EventInterest" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "eventId" INTEGER NOT NULL,
    "userId" TEXT NOT NULL,
    "createdAt" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT "EventInterest_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "Event" ("id") ON DELETE CASCADE ON UPDATE CASCADE
)9 M' indexsqlite_autoindex_EventInterest_1EventInterest @ Y/ @SindexTicketReservation_status_expiresAt_idxTicketReservation CREATE
INDEX "TicketReservation_status_expiresAt_idx" ON "TicketReservation"("status",
"expiresAt")@ E/ @%indexTicketReservation_userId_idxTicketReservation CREATE INDEX "TicketReservation_userId_idx" ON
"TicketReservation"("userId")@ G/ @)indexTicketReservation_eventId_idxTicketReservation CREATE INDEX
"TicketReservation_eventId_idx" ON "TicketReservation"("eventId")@ I/ @-indexTicketReservation_ticketId_idxTicketReservation
CREATE INDEX "TicketReservation_ticketId_idx" ON "TicketReservation"
("ticketId")A U/ indexsqlite_autoindex_TicketReservation_1TicketReservation @) // @_tableTicketReservationTicketReservation CREATE
TABLE "TicketReservation" (
    "id" TEXT NOT NULL PRIMARY KEY,
    "ticketId" TEXT NOT NULL,
    "eventId" INTEGER NOT NULL,
    "userId" TEXT,
    "quantity" INTEGER NOT NULL,
    "status" TEXT NOT NULL DEFAULT 'ACTIVE',
    "expiresAt" DATETIME NOT NULL,
    "createdAt" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updatedAt" DATETIME NOT NULL,
    CONSTRAINT "TicketReservation_ticketId_fkey" FOREIGN KEY ("ticketId") REFERENCES "Ticket" ("id") ON DELETE CASCADE ON UPDATE
CASCADE,
    CONSTRAINT "TicketReservation_eventId_fkey" FOREIGN KEY ("eventId") REFERENCES "Event" ("id") ON DELETE CASCADE ON UPDATE
CASCADE
)V ) {indexEvent_slug_keyEvent CREATE UNIQUE INDEX "Event_slug_key" ON "Event"("slug")@d   @_tableEventEvent CREATE TABLE "Event"
(
    "id" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
    "slug" TEXT NOT NULL,
    "title" TEXT NOT NULL,
    "description" TEXT NOT NULL,
    "startDate" DATETIME NOT NULL,
    "endDate" DATETIME NOT NULL,
    "locationName" TEXT NOT NULL,
    "address" TEXT NOT NULL,
    "isFree" BOOLEAN NOT NULL DEFAULT false,
    "basePrice" INTEGER,
    "timezone" TEXT NOT NULL DEFAULT 'Europe/Lisbon',
    "coverImageUrl" TEXT,
    "organizerName" TEXT,
    "organizerId" TEXT,
    "createdAt" DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    "updatedAt" DATETIME NOT NULL
)

```

```
  ' U 3aba5489-4d24-44a8-ad2e-5f0bf9f86f4f  
  
```

prisma/schema.prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  schemas  = ["app_v3", "auth"]
}

/// Perfil base do utilizador ORYA (1-1 com auth.users)
model Profile {
  id                      String      @id @db.Uuid
  username                String?    @unique @db.Citext
  fullName                String?    @map("full_name")
  avatarUrl               String?    @map("avatar_url")
  bio                     String?
  city                    String?
  gender                  Gender?
  favouriteCategories    String[]   @default([])
  @map("favourite_categories")
  onboardingDone          Boolean   @default(false)
  @map("onboarding_done")
  /// roles do user: ["user"], ["user","organizer"], ["user","admin"], etc.
  roles                  String[]   @default(["user"])
  createdAt               DateTime  @default(now()) @map("created_at")
  updatedAt               DateTime  @default(now()) @updatedAt
  @map("updated_at")
  allowEmailNotifications Boolean   @default(true)
  @map("allow_email_notifications")
  allowEventReminders     Boolean   @default(true)
  @map("allow_event_reminders")
  allowFriendRequests     Boolean   @default(true)
  @map("allow_friend_requests")
  deletedAt               DateTime? @map("deleted_at") @db.Timestamp(6)
  isDeleted               Boolean   @default(false) @map("is_deleted")
  visibility              Visibility @default(PUBLIC)
  isVerified              Boolean   @default(false)
  contactPhone            String?   @map("contact_phone")
  status                  AccountStatus? @default(ACTIVE)
  deletionRequestedAt    DateTime? @map("deletion_requested_at")
  @db.Timestamp(6)
  deletionScheduledFor   DateTime? @map("deletion_scheduled_for")
  @db.Timestamp(6)
  deletedAtFinal          DateTime? @map("deleted_at_final")
  @db.Timestamp(6)
  follows_follows_follower_idToprofiles follows[] @relation("follows_follower_idToprofiles")
  @relation("follows_follower_idToprofiles")
  follows_follows_following_idToprofiles follows[] @relation("follows_following_idToprofiles")
  @relation("follows_following_idToprofiles")
  notificationsSent       Notification[] @relation("NotificationSender")
  notifications           Notification[]
  notificationPreference  NotificationPreference?
  organizerInvitesSent   OrganizerMemberInvite[]
  @relation("OrganizerMemberInviteInviter")
  organizerInvitesReceived OrganizerMemberInvite[]
  @relation("OrganizerMemberInviteTarget")
  organizer_members_organizer_members_invited_by_user_idToprofiles OrganizerMember[]
  @relation("organizer_members_invited_by_user_idToprofiles")
  organizerMembers        OrganizerMember[]
  organizers              Organizer[]
  padelPairingSlots       PadelPairingSlot[]
  padelPairingsPlayer1    PadelPairing[] @relation("PadelPairingPlayer1")
  padelPairingsPlayer2    PadelPairing[] @relation("PadelPairingPlayer2")
  tournamentEntries       TournamentEntry[]
  tournamentsGenerated    Tournament[]
  tournamentAuditLogs     TournamentAuditLog[]
  users                   users @relation(fields: [id], references:
  [id], onDelete: Cascade, onUpdate: NoAction)
  saleSummaries           SaleSummary[]
  notificationOutbox      NotificationOutbox[]
```

```

@@map("profiles")
@@schema("app_v3")
}

/// Entidade organizadora (marca, página de eventos)
model Organizer {
    id          Int      @id @default(autoincrement())
    displayName String   @map("display_name")
    stripeAccountId String? @map("stripe_account_id")
    status       OrganizerStatus @default(PENDING)
    createdAt   DateTime @default(now()) @map("created_at")
    updatedAt   DateTime @default(now()) @updatedAt @map("updated_at")
    userId      String? @map("user_id") @db.Uuid
    feeMode     FeeMode @default(ADDED) @map("fee_mode")
    platformFeeBps Int     @default(200) @map("platform_fee_bps")
    platformFeeFixedCents Int    @default(0) @map("platform_fee_fixed_cents")
    stripeChargesEnabled Boolean @default(false) @map("stripe_charges_enabled")
    stripePayoutsEnabled Boolean @default(false) @map("stripe_payouts_enabled")
    entityType   String? @map("entity_type")
    businessName String? @map("business_name")
    city         String? @map("city")
    payoutIban   String? @map("payout_iban")
    username     String? @db.Ciext
    language     String? @default("pt")
    publiclistingEnabled Boolean? @default(true) @map("public_listing_enabled")
    alertsEmail  String? @map("alerts_email")
    alertsSalesEnabled Boolean? @default(true) @map("alerts_sales_enabled")
    alertsPayoutEnabled Boolean? @default(false) @map("alerts_payout_enabled")
    brandingAvatarUrl String? @map("branding_avatar_url")
    brandingPrimaryColor String? @map("branding_primary_color")
    brandingSecondaryColor String? @map("branding_secondary_color")
    refundFeePayer RefundFeePayer @default(CUSTOMER) @map("refund_fee_payer")
    organizationKind OrganizationKind @default(PESSOA_SINGULAR) @map("organization_kind")
    padelDefaultShortName String? @map("padel_default_short_name")
    padelDefaultCity String? @map("padel_default_city")
    padelDefaultAddress String? @map("padel_default_address")
    padelDefaultCourts Int     @default(0) @map("padel_default_courts")
    padelDefaultHours String? @map("padel_default_hours")
    padelDefaultRuleSetId Int? @map("padel_default_rule_set_id")
    padelFavoriteCategories Int[] @default([]) @map("padel_favorite_categories")
    publicName   String? @map("public_name")
    address     String? @map("address")
    showAddressPublicly Boolean @default(false) @map("show_address_publicly")
    is_platform_owned Boolean @default(false)
    orgType     OrgType @default(EXTERNAL) @map("org_type")
    officialEmail String? @map("official_email")
    officialEmailVerifiedAt DateTime? @map("official_email_verified_at") @db.Timestamptz(6)
    officialEmailRequests OrganizerOfficialEmailRequest[]
    events       Event[]
    notifications Notification[] @relation("NotificationOrganizer")
    memberInvites OrganizerMemberInvite[]
    members      OrganizerMember[]
    padelDefaultRuleSet PadelRuleSet? @relation("OrganizerPadelDefaultRuleSet", fields:
[padelDefaultRuleSetId], references: [id], onUpdate: NoAction, map: "organizers_padel_default_rule_set_fk")
    user         Profile? @relation(fields: [userId], references: [id], onUpdate: NoAction)
    padelCategories PadelCategory[]
    padelClubs    PadelClub[]
    padelPairings PadelPairing[]
    padelPlayers  PadelPlayerProfile[]
    padelRankingEntries PadelRankingEntry[]
    padelRuleSets PadelRuleSet[] @relation("OrganizerPadelRuleSets")
    padelTournaments PadelTournamentConfig[]
    staffAssignments StaffAssignment[]

    @@index([userId])
    @@map("organizers")
    @@schema("app_v3")
}

model OrganizerOfficialEmailRequest {
    id          Int      @id @default(autoincrement())
    organizerId Int      @map("organizer_id")
    requestedByUserId String? @map("requested_by_user_id") @db.Uuid
    newEmail    String   @map("new_email")
    token       String   @unique
    status      String   @default("PENDING")
    expiresAt   DateTime? @map("expires_at") @db.Timestamptz(6)
}

```

```

createdAt      DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
confirmedAt    DateTime? @map("confirmed_at") @db.Timestamptz(6)
cancelledAt    DateTime? @map("cancelled_at") @db.Timestamptz(6)

organizer Organizer @relation(fields: [organizerId], references: [id], onDelete: Cascade, onUpdate: NoAction, map:
"organizer_official_email_requests_organizer_fk")

@@index([organizerId], map: "organizer_official_email_requests_org_idx")
@@map("organizer_official_email_requests")
@@schema("app_v3")
}

model GlobalUsername {
  username String @id @db.Citext
  ownerType String @map("owner_type")
  ownerId String @map("owner_id")
  createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
  updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)

  @unique([ownerType, ownerId], map: "global_usernames_owner_unique")
  @@map("global_usernames")
  @@schema("app_v3")
}

model Notification {
  id          String          @id @default(dbgenerated("gen_random_uuid()")) @db.Uuid
  userId      String          @map("user_id") @db.Uuid
  type        NotificationType
  title       String?
  body        String?
  payload     Json?
  ctaUrl     String?         @map("cta_url")
  ctaLabel    String?         @map("cta_label")
  priority   NotificationPriority @default(NORMAL)
  fromUserId String?         @map("from_user_id") @db.Uuid
  organizerId Int?          @map("organizer_id")
  eventId     Int?           @map("event_id")
  ticketId   String?         @map("ticket_id")
  inviteId   String?         @map("invite_id") @db.Uuid
  isRead     Boolean          @default(false) @map("is_read")
  readAt     DateTime?        @map("read_at") @db.Timestamptz(6)
  seenAt     DateTime?        @map("seen_at") @db.Timestamptz(6)
  expiresAt  DateTime?        @map("expires_at") @db.Timestamptz(6)
  createdAt  DateTime          @default(now()) @map("created_at") @db.Timestamptz(6)
  event      Event?          @relation("NotificationEvent", fields: [eventId], references: [id], onUpdate: NoAction)
  fromUser   Profile?        @relation("NotificationSender", fields: [fromUserId], references: [id], onUpdate: NoAction)
  invite     OrganizerMemberInvite? @relation("NotificationInvite", fields: [inviteId], references: [id], onUpdate: NoAction)
  organizer  Organizer?      @relation("NotificationOrganizer", fields: [organizerId], references: [id], onUpdate:
NoAction)
  ticket     Ticket?         @relation("NotificationTicket", fields: [ticketId], references: [id], onUpdate: NoAction)
  user      Profile          @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@index([userId, createdAt])
  @@index([userId, readAt])
  @@index([fromUserId])
  @@index([organizerId])
  @@index([eventId])
  @@index([ticketId])
  @@index([inviteId])
  @@map("notifications")
  @@schema("app_v3")
}

model NotificationPreference {
  userId        String @id @map("userId") @db.Uuid
  allowEmailNotifications Boolean @default(true) @map("allow_email_notifications")
  allowEventReminders Boolean @default(true) @map("allow_event_reminders")
  allowFriendRequests Boolean @default(true) @map("allow_friend_requests")
  allowSalesAlerts Boolean @default(true) @map("allow_sales_alerts")
  allowSystemAnnouncements Boolean @default(true) @map("allow_system_announcements")
  createdAt    DateTime @default(now()) @map("created_at")
  updatedAt    DateTime @default(now()) @updatedAt @map("updated_at")
  user         Profile @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@map("notification_preferences")
  @@schema("app_v3")
}

```

```

model NotificationOutbox {
    id          String      @id @default(dbgenerated("gen_random_uuid()")) @db.Uuid
    userId      String?    @map("user_id") @db.Uuid
    notificationType String   @map("notification_type")
    templateVersion String? @map("template_version")
    dedupeKey    String     @unique @map("dedupe_key")
    status       String     @default("PENDING")
    payload      Json       @default("{}")
    retries      Int        @default(0)
    lastError    String?   @map("last_error")
    createdAt    DateTime   @default(now()) @map("created_at") @db.Timestamptz(6)
    sentAt      DateTime?  @map("sent_at") @db.Timestamptz(6)
    user        Profile?   @relation(fields: [userId], references: [id], onUpdate: NoAction, onDelete: NoAction)

    @@index([status], map: "notification_outbox_status_idx")
    @@index([userId], map: "notification_outbox_user_idx")
    @@map("notification_outbox")
    @@schema("app_v3")
}

model OrganizerMember {
    id          String      @id @default(dbgenerated("gen_random_uuid()"))
    @db.Uuid
    organizerId Int         @map("organizer_id")
    userId      String     @map("user_id") @db.Uuid
    role        OrganizerMemberRole
    invitedByUserId String? @map("invited_by_user_id") @db.Uuid
    createdAt    DateTime?  @default(now()) @map("created_at")
    @db.Timestamptz(6)
    updatedAt    DateTime?  @default(now()) @map("updated_at")
    @db.Timestamptz(6)
    lastUsedAt  DateTime?  @map("last_used_at") @db.Timestamptz(6)
    profiles_organizer_members_invited_by_user_idToprofiles Profile?
    @relation("organizer_members_invited_by_user_idToprofiles", fields: [invitedByUserId], references: [id], onDelete: NoAction,
    onUpdate: NoAction)
    organizer    Organizer   @relation(fields: [organizerId], references: [id],
    onDelete: Cascade, onUpdate: NoAction)
    user        Profile     @relation(fields: [userId], references: [id],
    onDelete: Cascade, onUpdate: NoAction)

    @@unique([organizerId, userId], map: "organizer_members_org_user_uniq")
    @@unique([organizerId, userId], map: "organizer_members_org_user_uq")
    @@index([organizerId, role], map: "organizer_members_org_role_idx")
    @@index([userId], map: "organizer_members_user_idx")
    @@index([userId, lastUsedAt(sort: Desc), createdAt], map: "organizer_members_user_last_used_idx")
    @@map("organizer_members")
    @@schema("app_v3")
}

model OrganizerMemberInvite {
    id          String      @id @default(dbgenerated("gen_random_uuid()")) @db.Uuid
    organizerId Int         @map("organizer_id")
    invitedByUserId String   @map("invited_by_user_id") @db.Uuid
    targetIdentifier String   @map("target_identifier") @db.Citext
    targetUserId  String?   @map("target_user_id") @db.Uuid
    role        OrganizerMemberRole
    token       String     @unique @db.Uuid
    expiresAt   DateTime   @map("expires_at") @db.Timestamptz(6)
    acceptedAt  DateTime?  @map("accepted_at") @db.Timestamptz(6)
    declinedAt  DateTime?  @map("declined_at") @db.Timestamptz(6)
    cancelledAt DateTime?  @map("cancelled_at") @db.Timestamptz(6)
    createdAt    DateTime?  @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt    DateTime?  @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    notifications Notification[] @relation("NotificationInvite")
    invitedBy    Profile    @relation("OrganizerMemberInviteInviter", fields: [invitedByUserId], references: [id],
    onDelete: Cascade, onUpdate: NoAction)
    organizer    Organizer   @relation(fields: [organizerId], references: [id], onDelete: Cascade, onUpdate: NoAction)
    targetUser    Profile?   @relation("OrganizerMemberInviteTarget", fields: [targetUserId], references: [id],
    onDelete: Cascade, onUpdate: NoAction)

    @@index([targetIdentifier], map: "organizer_member_invites_identifier_idx")
    @@index([organizerId], map: "organizer_member_invites_org_idx")
    @@index([targetUserId], map: "organizer_member_invites_target_idx")
    @@map("organizer_member_invites")
    @@schema("app_v3")
}

/// Evento/Experiência – único tipo com campo `type` a separar

```

```

model Event {
    id          Int          @id @default(autoincrement())
    slug        String       @unique
    title       String
    description String
    /// "EXPERIENCE" (user) ou "ORGANIZER_EVENT" (organizador)
    type        EventType    @default(EXPERIENCE)
    /// Template de apoio (festa, desporto, voluntariado, palestra...)
    templateType EventTemplateType? @map("template_type")
    /// Liga ao Organizer quando é evento de organizador
    organizerId Int?        @map("organizer_id")
    startsAt    DateTime     @map("starts_at")
    endsAt     DateTime     @map("ends_at")
    locationName String      @map("location_name")
    locationCity String?    @map("location_city")
    address      String?
    latitude     Float?      @map("lat")
    longitude    Float?      @map("lng")
    isFree       Boolean     @default(false) @map("is_free")
    status        EventStatus @default(DRAFT)
    timezone     String      @default("Europe/Lisbon")
    coverImageUrl String?    @map("cover_image_url")
    createdat    DateTime     @default(now()) @map("created_at")
    updatedAt    DateTime     @updatedAt @map("updated_at")
    /// auth.users.id de quem criou (user base)
    ownerUserId  String      @map("owner_user_id") @db.Uuid
    deletedAt    DateTime?   @map("deleted_at") @db.Timestamp(6)
    isDeleted    Boolean     @default(false) @map("is_deleted")
    resaleMode   ResaleMode  @default(ALWAYS) @map("resale_mode")
    feeMode_override FeeMode? 
    platform_fee_bps_override Int?
    platform_fee_fixed_cents_override Int?
    feeMode      FeeMode     @default(INCLUDED) @map("fee_mode")
    isTest       Boolean     @default(false) @map("is_test")
    payoutMode   PayoutMode  @default(ORGANIZER) @map("payout_mode")
    categories   EventCategory[]
    interests    EventInterest[]
    salesAgg     EventSalesAgg?
    views        EventView[]
    organizer    Organizer?  @relation(fields: [organizerId], references: [id])
    participants ExperienceParticipant[]
    notifications Notification[] @relation("NotificationEvent")
    padelMatches PadelMatch[]
    padelPairings PadelPairing[]
    padelRankingEntries PadelRankingEntry[]
    padelTeams   PadelTeam[]
    padelTournamentConfig PadelTournamentConfig?
    tournamentEntries TournamentEntry[]
    tournament   Tournament? @relation("EventTournament")
    promoCodes   PromoCode[]
    saleLines    SaleLine[]
    saleSummaries SaleSummary[]
    staffAssignments StaffAssignment[]
    reservations TicketReservation[]
    ticketTypes  TicketType[]
    tickets      Ticket[]

    @@index([type, status])
    @@index([ownerUserId])
    @@index([organizerId])
    @map("events")
    @@schema("app_v3")
}

/// Junção N-N de eventos a categorias
model EventCategory {
    id          Int          @id @default(autoincrement())
    eventId     Int          @map("event_id")
    category    EventCategoryType
    createdAt  DateTime     @default(now()) @map("created_at") @db.Timestamptz(6)
    event       Event        @relation(fields: [eventId], references: [id], onDelete: Cascade, onUpdate: NoAction, map: "event_categories_event_fk")
    @@unique([eventId, category], map: "event_categories_unique")
    @@index([category])
    @map("event_categories")
    @@schema("app_v3")
}

```

```

/// Tipos de bilhete (antigas "waves")
model TicketType {
    id          Int           @id @default(autoincrement())
    eventId     Int
    name        String
    description String?
    price       Int
    currency    String        @default("EUR")
    totalQuantity Int?        @map("total_quantity")
    soldQuantity Int          @default(0) @map("sold_quantity")
    status      TicketTypeStatus @default(ON_SALE)
    startsAt    DateTime?     @map("starts_at")
    endsAt     DateTime?     @map("ends_at")
    sortOrder   Int          @default(0) @map("sort_order")
    createdAt   DateTime      @default(now()) @map("created_at")
    updatedAt   DateTime      @updatedAt @map("updated_at")
    saleLines   SaleLine[]
    reservations TicketReservation[]
    event       Event         @relation(fields: [eventId], references: [id], onDelete: Cascade)
    tickets     Ticket[]

    @@index([eventId])
    @@map("ticket_types")
    @@schema("app_v3")
}

/// Bilhete individual (por utilizador)
model Ticket {
    id          String        @id @default(cuid())
    eventId     Int
    ticketTypeId Int          @map("ticket_type_id")
    purchasedAt DateTime      @default(now()) @map("purchased_at")
    status      TicketStatus  @default(ACTIVE)
    qrSecret    String        @unique @map("qr_secret")
    rotatingSeed String?      @map("rotating_seed") @db.Uuid
    pricePaid   Int          @map("price_paid")
    currency    String        @default("EUR")
    stripePaymentIntentId String? @map("stripe_payment_intent_id")
    usedAt     DateTime?     @map("used_at")
    userId      String        @map("user_id") @db.Uuid
    platformFeeCents Int        @default(0) @map("platform_fee_cents")
    totalPaidCents Int        @default(0) @map("total_paid_cents")
    pairingId   Int?         @map("pairing_id")
    padelSplitShareCents Int?    @map("padel_split_share_cents")
    padelPairingVersion Int?    @map("padel_pairing_version")
    purchaseId  String        @map("purchase_id") @db.Text
    salesSummaryId Int?       @map("sale_summary_id")
    emissionIndex Int          @default(0) @map("emission_index")
    guestLink    GuestTicketLink?
    notifications Notification[] @relation("NotificationTicket")
    pairingSlot   PadelPairingSlot? @relation("TicketPairingSlot")
    createdPadelPairings PadelPairing[] @relation("PadelPairingCreatedByTicket")
    resales      TicketResale[]
    transfers    TicketTransfer[]
    event       Event         @relation(fields: [eventId], references: [id], onDelete: Cascade)
    pairing     PadelPairing? @relation("PairingTickets", fields: [pairingId], references: [id])
    ticketType  TicketType   @relation(fields: [ticketTypeId], references: [id], onDelete: Cascade)
    ownerUserId String?      @map("owner_user_id") @db.Uuid
    ownerId     String?      @map("owner_identity_id") @db.Uuid
    tournamentEntryId Int?    @map("tournament_entry_id")
    tournamentEntry TournamentEntry? @relation("TicketTournamentEntry", fields: [tournamentEntryId], references: [id], onDelete: SetNull)
    salesSummary SaleSummary? @relation(fields: [salesSummaryId], references: [id], onDelete: SetNull)

    @@unique([purchaseId, ticketTypeId, emissionIndex], map: "tickets_purchase_ticket_idx")
    @@index([eventId])
    @@index([userId])
    @@index([ticketTypeId])
    @@index([stripePaymentIntentId])
    @@index([pairingId])
    @@index([salesSummaryId])
    @@map("tickets")
    @@schema("app_v3")
}

/// Ligação de bilhetes comprados como convidado
/// This model contains an expression index which requires additional setup for migrations. Visit https://pris.ly/d/expression-

```

```

indexes for more info.

model GuestTicketLink {
    ticketId      String   @id @map("ticket_id")
    guestEmail    String   @map("guest_email")
    guestName     String   @map("guest_name")
    guestPhone    String?  @map("guest_phone")
    migratedToUserId String? @map("migrated_to_user_id") @db.Uuid
    migratedAt    DateTime? @map("migrated_at") @db.Timestamptz(6)
    createdAt     DateTime? @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt     DateTime? @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    users         users?   @relation(fields: [migratedToUserId], references: [id], onUpdate: NoAction)
    ticket        Ticket   @relation(fields: [ticketId], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @@index([migratedToUserId], map: "guest_ticket_links_migrated_idx")
    @@map("guest_ticket_links")
    @@schema("app_v3")
}

/// Reservas temporárias de stock de bilhetes
model TicketReservation {
    id          String   @id @default(cuid())
    eventId     Int
    ticketTypeId Int      @map("ticket_type_id")
    quantity    Int
    status      ReservationStatus @default(ACTIVE)
    expiresAt   DateTime  @map("expires_at")
    createdAt    DateTime  @default(now()) @map("created_at")
    updatedAt    DateTime  @updatedAt @map("updated_at")
    userId      String?  @map("user_id") @db.Uuid
    event       Event    @relation(fields: [eventId], references: [id], onDelete: Cascade)
    ticketType  TicketType @relation(fields: [ticketTypeId], references: [id], onDelete: Cascade)

    @@index([eventId])
    @@index([ticketTypeId])
    @@index([userId])
    @@index([status, expiresAt])
    @@map("ticket_reservations")
    @@schema("app_v3")
}

/// Participantes em experiências (sem QR / bilhete)
model ExperienceParticipant {
    id          String   @id @default(cuid())
    eventId     Int
    createdAt   DateTime @default(now()) @map("created_at")
    userId      String   @db.Uuid
    volunteerMinutes Int?
    event       Event    @relation(fields: [eventId], references: [id], onDelete: Cascade)

    @@unique([eventId, userId])
    @@index([userId])
    @@map("experience_participants")
    @@schema("app_v3")
}

/// Interesse / "quero ir" em eventos de organizador
model EventInterest {
    id          String   @id @default(cuid())
    eventId     Int
    createdAt   DateTime @default(now()) @map("created_at")
    userId      String   @db.Uuid
    event       Event    @relation(fields: [eventId], references: [id], onDelete: Cascade)

    @@unique([eventId, userId])
    @@index([userId])
    @@map("event_interests")
    @@schema("app_v3")
}

/// Staff associado a organizadores / eventos
model StaffAssignment {
    id          Int      @id @default(autoincrement())
    organizerId Int     @map("organizer_id")
    scope       StaffScope
    eventId     Int?    @map("event_id")
    createdAt   DateTime @default(now()) @map("created_at")
    revokedAt   DateTime? @map("revoked_at")
    userId      String   @map("user_id") @db.Uuid
}

```

```

acceptedAt DateTime? @map("accepted_at") @db.Timestamp(6)
status StaffStatus @default(PENDING)
role StaffRole @default(STAFF)
event Event? @relation(fields: [eventId], references: [id])
organizer Organizer @relation(fields: [organizerId], references: [id], onDelete: Cascade)

@@index([organizerId])
@@index([userId])
@@index([eventId])
@@map("staff_assignments")
@@schema("app_v3")
}

/// Transferência de bilhetes entre utilizadores
model TicketTransfer {
    id String @id @default(cuid())
    ticketId String @map("ticket_id")
    status TransferStatus @default(PENDING)
    createdAt DateTime @default(now()) @map("created_at")
    completedAt DateTime? @map("completed_at")
    fromUserId String @map("from_user_id") @db.Uuid
    toUserId String @map("to_user_id") @db.Uuid
    ticket Ticket @relation(fields: [ticketId], references: [id], onDelete: Cascade)

    @@index([ticketId])
    @@index([fromUserId])
    @@index([toUserId])
    @@map("ticket_transfers")
    @@schema("app_v3")
}

/// Revenda de bilhetes pelo utilizador
model TicketResale {
    id String @id @default(cuid())
    ticketId String @map("ticket_id")
    price Int
    currency String @default("EUR")
    status ResaleStatus @default(LISTED)
    createdAt DateTime @default(now()) @map("created_at")
    completedAt DateTime? @map("completed_at")
    sellerUserId String @map("seller_user_id") @db.Uuid
    ticket Ticket @relation(fields: [ticketId], references: [id], onDelete: Cascade)

    @@index([ticketId])
    @@index([sellerUserId])
    @@map("ticket_resales")
    @@schema("app_v3")
}

/// Pageviews por evento (simples, para futuras stats)
model EventView {
    id String @id @default(cuid())
    eventId Int
    sessionId String? @map("session_id")
    createdAt DateTime @default(now()) @map("created_at")
    userId String? @db.Uuid
    event Event @relation(fields: [eventId], references: [id], onDelete: Cascade)

    @@index([eventId])
    @@index([userId])
    @@map("event_views")
    @@schema("app_v3")
}

/// Agregado diário de vendas por evento
model EventSalesAgg {
    id Int @id @default(autoincrement())
    eventId Int @unique
    date DateTime @map("date")
    ticketsSold Int @default(0) @map("tickets_sold")
    revenue Int @default(0) @map("revenue")
    event Event @relation(fields: [eventId], references: [id], onDelete: Cascade)

    @@map("event_sales_agg")
    @@schema("app_v3")
}

/// Registo de eventos de pagamento (auditoria webhooks/intent)

```

```

model PaymentEvent {
    id          Int      @id @default(autoincrement())
    stripePaymentIntentId String @unique @map("stripe_payment_intent_id")
    status       String   @default("PROCESSING")
    stripeEventId String?  @unique @map("stripe_event_id")
    purchaseId  String?  @unique @map("purchase_id") @db.Text
    source       PaymentEventSource @default(WEBHOOK)
    dedupeKey   String?  @map("dedupe_key")
    attempt     Int      @default(1)
    eventId     Int?    @map("event_id")
    userId      String?  @map("user_id") @db.Uuid
    amountCents Int?    @map("amount_cents")
    platformFeeCents Int?  @map("platform_fee_cents")
    stripeFeeCents Int?  @map("stripe_fee_cents")
    errorMessage String? @map("error_message")
    createdAt   DateTime? @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt   DateTime? @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    mode        PaymentMode @default(LIVE)
    isTest      Boolean   @default(false) @map("is_test")

    @@index([stripePaymentIntentId])
    @@index([stripeEventId])
    @@index([purchaseId])
    @@index([eventId])
    @@map("payment_events")
    @@schema("app_v3")
}

model Operation {
    id          Int      @id @default(autoincrement())
    operationType String  @map("operation_type")
    dedupeKey   String   @unique(map: "operations_dedupe_key_unique") @map("dedupe_key")
    status       OperationStatus @default(PENDING)
    attempts    Int      @default(0)
    lastError   String?  @map("last_error")
    lockedAt   DateTime? @map("locked_at") @db.Timestamptz(6)
    nextRetryAt DateTime? @map("next_retry_at") @db.Timestamptz(6)
    payload     Json?   @default("{}")
    purchaseId  String?  @map("purchase_id") @db.Text
    paymentIntentId String? @map("payment_intent_id")
    stripeEventId String? @map("stripe_event_id")
    eventId     Int?    @map("event_id")
    organizerId Int?    @map("organizer_id")
    pairingId   Int?    @map("pairing_id")
    createdAt   DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt   DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)

    @@index([status], map: "operations_status_idx")
    @@index([purchaseId], map: "operations_purchase_idx")
    @@index([paymentIntentId], map: "operations_payment_intent_idx")
    @@index([stripeEventId], map: "operations_stripe_event_idx")
    @@map("operations")
    @@schema("app_v3")
}

/// Configurações simples chave-valor (taxas, etc.)
model PlatformSetting {
    id          Int      @id @default(autoincrement())
    key         String   @unique(map: "platform_settings_key_idx")
    value       String
    createdAt  DateTime? @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt  DateTime? @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)

    @@map("platform_settings")
    @@schema("app_v3")
}

/// Códigos promocionais simples
/// This model contains an expression index which requires additional setup for migrations. Visit https://pris.ly/d/expression-indexes for more info.
model PromoCode {
    id          Int      @id @default(autoincrement())
    code        String
    type        PromoType
    value       Int
    maxUses    Int?    @map("max_uses")
    perUserLimit Int?  @map("per_user_limit")
    validFrom  DateTime? @map("valid_from") @db.Timestamptz(6)
}

```

```

validUntil DateTime? @map("valid_until") @db.Timestamptz(6)
active Boolean @default(true)
eventId Int? @map("event_id")
createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
autoApply Boolean? @default(false) @map("auto_apply")
minQuantity Int? @map("min_quantity")
minTotalCents Int? @map("min_total_cents")
event Event? @relation(fields: [eventId], references: [id], onUpdate: NoAction, map:
"promo_codes_event_fk")
redemptions PromoRedemption[]
saleLines SaleLine[]
saleSummaries SaleSummary[]

@@index([eventId])
@@index([eventId, active], map: "promo_codes_event_active_idx")
@@index([validFrom, validUntil], map: "promo_codes_valid_idx")
@map("promo_codes")
@@schema("app_v3")
}

/// This model contains an expression index which requires additional setup for migrations. Visit https://pris.ly/d/expression-indexes for more info.
model PromoRedemption {
    id Int @id @default(autoincrement())
    promoCodeId Int @map("promo_code_id")
    userId String? @map("user_id") @db.Uuid
    guestEmail String? @map("guest_email")
    usedAt DateTime @default(now()) @map("used_at") @db.Timestamptz(6)
    purchaseId String? @map("purchase_id") @db.Text
    promoCode PromoCode @relation(fields: [promoCodeId], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @unique([purchaseId, promoCodeId], map: "promo_redemptions_purchase_code_unique")
    @@index([promoCodeId], map: "promo_redemptions_code_idx")
    @@index([promoCodeId], map: "promo_redemptions_promo_idx")
    @@index([userId], map: "promo_redemptions_user_idx")
    @@index([purchaseId], map: "promo_redemptions_purchase_idx")
    @map("promo_redemptions")
    @@schema("app_v3")
}

model Refund {
    id Int @id @default(autoincrement())
    dedupeKey String @unique(map: "refunds_dedupe_key_unique") @map("dedupe_key")
    purchaseId String? @map("purchase_id") @db.Text
    paymentIntentId String? @map("payment_intent_id")
    eventId Int @map("event_id")
    baseAmountCents Int @map("base_amount_cents")
    feesExcludedCents Int @map("fees_excluded_cents")
    reason RefundReason
    refundedBy String? @map("refunded_by")
    refundedAt DateTime? @default(now()) @map("refunded_at") @db.Timestamptz(6)
    stripeRefundId String? @map("stripe_refund_id")
    auditPayload Json? @map("audit_payload")
    createdAt DateTime? @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime? @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)

    @@index([eventId], map: "refunds_event_idx")
    @@index([purchaseId], map: "refunds_purchase_idx")
    @@index([paymentIntentId], map: "refunds_payment_intent_idx")
    @map("refunds")
    @@schema("app_v3")
}

/// Venda agregada por PaymentIntent (fonte de verdade de valores)
model SaleSummary {
    id Int @id @default(autoincrement())
    paymentIntentId String @unique @map("payment_intent_id")
    eventId Int @map("event_id")
    userId String? @map("user_id") @db.Uuid
    promoCodeId Int? @map("promo_code_id")
    subtotalCents Int @map("subtotal_cents")
    discountCents Int @map("discount_cents")
    platformFeeCents Int @map("platform_fee_cents")
    totalCents Int @map("total_cents")
    netCents Int @map("net_cents")
    feeMode FeeMode? @map("fee_mode")
    currency String @default("EUR")
}

```

```

createdAt      DateTime      @default(now()) @map("created_at") @db.Timestamptz(6)
updatedAt      DateTime      @default(now()) @updatedat @map("updated_at") @db.Timestamptz(6)
purchaseId     String?      @unique @map("purchase_id") @db.Text
promoCodeSnapshot String?    @map("promo_code_snapshot")
promoLabelSnapshot String?   @map("promo_label_snapshot")
promoTypeSnapshot PromoType? @map("promo_type_snapshot")
promoValueSnapshot Int?     @map("promo_value_snapshot")
stripeFeeCents  Int         @default(0) @map("stripe_fee_cents")
status          SaleSummaryStatus @default(PAID)
lines           SaleLine[]
tickets         Ticket[]
tournamentEntries TournamentEntry[]
event           Event        @relation(fields: [eventId], references: [id], onDelete: Cascade, onUpdate: NoAction)
promoCode       PromoCode?   @relation(fields: [promoCodeId], references: [id], onUpdate: NoAction)
user            Profile?    @relation(fields: [userId], references: [id], onUpdate: NoAction)
ownerUserId     String?     @map("owner_user_id") @db.Uuid
ownerIdentityId String?    @map("owner_identity_id") @db.Uuid

@@index([eventId], map: "sale_summaries_event_idx")
@@index([promoCodeId], map: "sale_summaries_promo_idx")
@@index([userId], map: "sale_summaries_user_idx")
@@map("sale_summaries")
@@schema("app_v3")
}

/// Linhas de venda por tipo de bilhete (para analytics, promo e fees)
model SaleLine {
  id              Int          @id @default(autoincrement())
  saleSummaryId   Int          @map("sale_summary_id")
  eventId         Int          @map("event_id")
  ticketTypeId    Int          @map("ticket_type_id")
  promoCodeId     Int?         @map("promo_code_id")
  quantity        Int
  unitPriceCents  Int          @map("unit_price_cents")
  discountPerUnitCents Int        @default(0) @map("discount_per_unit_cents")
  grossCents      Int          @map("gross_cents")
  netCents        Int          @map("net_cents")
  platformFeeCents Int         @default(0) @map("platform_fee_cents")
  createdat       DateTime     @default(now()) @map("created_at") @db.Timestamptz(6)
  promoCodeSnapshot String?    @map("promo_code_snapshot")
  promoLabelSnapshot String?   @map("promo_label_snapshot")
  promoTypeSnapshot PromoType? @map("promo_type_snapshot")
  promoValueSnapshot Int?     @map("promo_value_snapshot")
  event           Event        @relation(fields: [eventId], references: [id], onDelete: Cascade, onUpdate: NoAction, map:
"sale_lines_event_fkey")
  promoCode       PromoCode?   @relation(fields: [promoCodeId], references: [id], onUpdate: NoAction, map:
"sale_lines_promo_code_fkey")
  saleSummary     SaleSummary  @relation(fields: [saleSummaryId], references: [id], onDelete: Cascade, onUpdate: NoAction,
map: "sale_lines_summary_fkey")
  ticketType      TicketType   @relation(fields: [ticketTypeId], references: [id], onDelete: Cascade, onUpdate: NoAction,
map: "sale_lines_ticket_type_fkey")

@@index([eventId], map: "sale_lines_event_idx")
@@index([promoCodeId], map: "sale_lines_promo_code_idx")
@@index([saleSummaryId], map: "sale_lines_summary_idx")
@@index([ticketTypeId], map: "sale_lines_ticket_type_idx")
@@map("sale_lines")
@@schema("app_v3")
}

/// Padel MVP: perfis de jogadores do clube
model PadelPlayerProfile {
  id              Int          @id @default(autoincrement())
  organizerId    Int          @map("organizer_id")
  userId          String?      @map("user_id") @db.Uuid
  fullName        String        @map("full_name")
  email           String?      @db.CiText
  phone           String?
  gender          String?
  level           String?
  isActive        Boolean       @default(true) @map("is_active")
  notes           String?
  createdAt       DateTime     @default(now()) @map("created_at") @db.Timestamptz(6)
  updatedAt       DateTime     @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
  displayName     String?      @map("display_name")
  preferredSide   PadelPreferredSide? @map("preferred_side")
  clubName        String?      @map("club_name")
  birthDate       DateTime     @map("birth_date") @db.Timestamp(6)
}

```

```

pairingSlots PadelPairingSlot[]
organizer Organizer @relation(fields: [organizerId], references: [id], onDelete: Cascade, onUpdate: NoAction)
users users? @relation(fields: [userId], references: [id], onUpdate: NoAction)
rankings PadelRankingEntry[]
teams1 PadelTeam[] @relation("PadelTeamPlayer1")
teams2 PadelTeam[] @relation("PadelTeamPlayer2")

@map("padel_player_profiles")
@schema("app_v3")
}

/// Clubes de Padel por organizador
model PadelClub {
    id Int @id @default(autoincrement())
    organizerId Int @map("organizer_id")
    name String
    shortName String? @map("short_name")
    city String?
    address String?
    courtsCount Int @default(1) @map("courts_count")
    hours String?
    favoriteCategoryIds Int[] @default([]) @map("favorite_category_ids")
    slug String? @unique
    isActive Boolean @default(true) @map("is_active")
    createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    isDefault Boolean @default(false) @map("is_default")
    deletedAt DateTime? @map("deleted_at") @db.Timestamptz(6)
    courts PadelclubCourt[]
    staff PadelClubStaff[]
    organizer Organizer @relation(fields: [organizerId], references: [id], onDelete: Cascade, onUpdate: NoAction)
    tournaments PadelTournamentConfig[]

    @index([organizerId], map: "padel_clubs_organizer_idx")
    @map("padel_clubs")
    @schema("app_v3")
}

model PadelClubCourt {
    id Int @id @default(autoincrement())
    padelClubId Int @map("padel_club_id")
    name String
    description String?
    surface String?
    indoor Boolean @default(false)
    isActive Boolean @default(true) @map("is_active")
    displayOrder Int @default(0) @map("display_order")
    createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    club PadelClub @relation(fields: [padelClubId], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @index([padelClubId], map: "padel_club_courts_club_idx")
    @map("padel_club_courts")
    @schema("app_v3")
}

model PadelClubStaff {
    id Int @id @default(autoincrement())
    padelClubId Int @map("padel_club_id")
    userId String? @map("user_id") @db.Uuid
    email String? @db.Ciext
    role String
    inheritToEvents Boolean @default(true) @map("inherit_to_events")
    createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    isActive Boolean @default(true) @map("is_active")
    deletedAt DateTime? @map("deleted_at") @db.Timestamptz(6)
    club PadelClub @relation(fields: [padelClubId], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @index([padelClubId], map: "padel_club_staff_club_idx")
    @map("padel_club_staff")
    @schema("app_v3")
}

/// Categorias / níveis de Padel
model PadelCategory {
    id Int @id @default(autoincrement())

```

```

organizerId Int @map("organizer_id")
label String @map("gender_restriction")
genderRestriction String? @map("min_level")
minLevel String? @map("max_level")
maxLevel String?
isDefault Boolean @default(false) @map("is_default")
isActive Boolean @default(true) @map("is_active")
season String?
year Int?
createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
organizer Organizer @relation(fields: [organizerId], references: [id], onDelete: Cascade, onUpdate:
NoAction)
matches PadelMatch[]
pairings PadelPairing[]
teams PadelTeam[]
tournamentConfigs PadelTournamentConfig[]

@@map("padel_categories")
@@schema("app_v3")
}

/// Regras base + tabela de pontos por clube
model PadelRuleSet {
    id Int @id @default(autoincrement())
    organizerId Int @map("organizer_id")
    name String
    tieBreakRules Json @default("{}") @map("tie_break_rules")
    pointsTable Json @default("{}") @map("points_table")
    enabledFormats String[] @default(["TODOS CONTRA TODOS", "QUADRO ELIMINATORIO"]) @map("enabled_formats")
    season String?
    year Int?
    createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    defaultForOrganizers Organizer[] @relation("OrganizerPadelDefaultRuleSet")
    organizer Organizer @relation("OrganizerPadelRuleSets", fields: [organizerId], references: [id],
onDelete: Cascade, onUpdate: NoAction)
    tournaments PadelTournamentConfig[]

    @@map("padel_rule_sets")
    @@schema("app_v3")
}

/// Configuração específica do torneio Padel (por evento)
model PadelTournamentConfig {
    id Int @id @default(autoincrement())
    eventId Int @unique(map: "padel_tournament_configs_event_unique") @map("event_id")
    organizerId Int @map("organizer_id")
    format padel_format
    numberOfWorks Int @default(1) @map("number_of_courts")
    ruleSetId Int? @map("rule_set_id")
    defaultCategoryId Int? @map("default_category_id")
    enabledFormats String[] @default([]) @map("enabled_formats")
    createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    padelV2Enabled Boolean @default(false) @map("padel_v2_enabled")
    splitDeadlineHours Int? @map("split_deadline_hours")
    autoCancelUnpaid Boolean @default(true) @map("auto_cancel_unpaid")
    allowCaptainAssume Boolean @default(true) @map("allow_captain_assume")
    defaultPaymentMode PadelPaymentMode @map("default_payment_mode")
    refundFeePayer RefundFeePayer? @map("refund_fee_payer")
    padelClubId Int? @map("padel_club_id")
    clubHours String? @map("club_hours")
    partnerClubIds Int[] @default([]) @map("partner_club_ids")
    advancedSettings Json? @map("advanced_settings")
    eligibilityType PadelEligibilityType @default(OPEN) @map("eligibility_type")
    category PadelCategory? @relation(fields: [defaultCategoryId], references: [id], onUpdate: NoAction)
    event Event @relation(fields: [eventId], references: [id], onDelete: Cascade, onUpdate: NoAction)
    organizer Organizer @relation(fields: [organizerId], references: [id], onDelete: Cascade, onUpdate:
NoAction)
    club PadelClub? @relation(fields: [padelClubId], references: [id], onUpdate: NoAction)
    ruleSet PadelRuleSet? @relation(fields: [ruleSetId], references: [id], onUpdate: NoAction)

    @@index([padelClubId], map: "padel_tournament_configs_padel_club_idx")
    @@map("padel_tournament_configs")
    @@schema("app_v3")
}

```

```

/// Pairings (duplicas) para Padel v2
model PadelPairing {
    id          Int          @id @default(autoincrement())
    eventId     Int          @map("event_id")
    organizerId Int          @map("organizer_id")
    categoryId  Int?         @map("category_id")
    player1UserId String?    @map("player1_user_id") @db.Uuid
    player2UserId String?    @map("player2_user_id") @db.Uuid
    player2IdentityId String?    @map("player2_identity_id") @db.Uuid
    paymentMode  PadelPaymentMode
    pairingStatus PadelPairingStatus @default(INCOMPLETE) @map("pairing_status")
    lifecycleStatus PadelPairingLifecycleStatus @default(PENDING_ONE_PAID) @map("lifecycle_status")
    pairingJoinMode PadelPairingJoinMode @default(INVITE_PARTNER) @map("pairing_join_mode")
    createdById String?    @map("created_by_user_id") @db.Uuid
    createdByTicketId String?    @map("created_by_ticket_id")
    partnerInviteToken String?    @unique @map("invite_token")
    partnerInviteUsedAt DateTime?  @map("partner_invite_used_at") @db.Timestamptz(6)
    partnerLinkToken String?    @map("partner_link_token")
    partnerLinkExpiresAt DateTime?  @map("invite_expires_at") @db.Timestamptz(6)
    lockedUntil   DateTime?  @map("locked_until") @db.Timestamptz(6)
    isPublicOpen  Boolean     @default(false) @map("is_public_open")
    deadlineAt    DateTime?  @map("deadline_at") @db.Timestamptz(6)
    partnerSwapAllowedUntilAt DateTime?  @map("partner_swap_allowed_until_at") @db.Timestamptz(6)
    graceUntilAt  DateTime?  @map("grace_until_at") @db.Timestamptz(6)
    partnerInvitedAt DateTime?  @map("partner_invited_at") @db.Timestamptz(6)
    partnerAcceptedAt DateTime?  @map("partner_accepted_at") @db.Timestamptz(6)
    partnerPaidAt  DateTime?  @map("partner_paid_at") @db.Timestamptz(6)
    captainSecondChargedAt DateTime?  @map("captain_second_charged_at") @db.Timestamptz(6)
    guaranteeeStatus PadelPairingGuaranteeStatus @default(NONE) @map("guaranteee_status")
    setupIntentId String?    @map("setup_intent_id")
    paymentMethodId String?    @map("payment_method_id")
    secondChargePaymentIntentId String?    @map("second_charge_payment_intent_id")
    captainConsentAt DateTime?  @map("captain_consent_at") @db.Timestamptz(6)
    captainFirstSaleId Int?      @map("captain_first_sale_id")
    partnerSaleId  Int?      @map("partner_sale_id")
    captainSecondSaleId Int?      @map("captain_second_sale_id")
    createdAt     DateTime     @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt     DateTime     @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    matchesA     PadelMatch[]  @relation("MatchPairingA")
    matchesB     PadelMatch[]  @relation("MatchPairingB")
    matchesWinner PadelMatch[]  @relation("MatchWinnerPairing")
    slots        PadelPairingSlot[]
    category     PadelCategory? @relation(fields: [categoryId], references: [id])
    createdByTicket Ticket?    @relation("PadelPairingCreatedByTicket", fields: [createdByTicketId], references: [id], map: "padel_pairings_created_ticket_fkey")
    event        Event        @relation(fields: [eventId], references: [id], onDelete: Cascade)
    organizer    Organizer    @relation(fields: [organizerId], references: [id], onDelete: Cascade)
    tickets      Ticket[]    @relation("PairingTickets")
    player1      Profile?    @relation("PadelPairingPlayer1", fields: [player1UserId], references: [id], onUpdate: NoAction)
    player2      Profile?    @relation("PadelPairingPlayer2", fields: [player2UserId], references: [id], onUpdate: NoAction)
    holds        PadelPairingHold[] @relation("PadelPairingHold", fields: [pairingId], references: [id])
    tournamentEntries TournamentEntry[]

    @@index([eventId])
    @@index([organizerId])
    @@index([categoryId])
    @@index([player1UserId])
    @@index([player2UserId])
    @@map("padel_pairings")
    @@schema("app_v3")
}

model PadelPairingHold {
    id          Int          @id @default(autoincrement())
    pairingId  Int          @map("pairing_id")
    eventId     Int          @map("event_id")
    holds      Int          @default(2)
    status      PadelPairingHoldStatus @default(ACTIVE)
    expiresAt  DateTime     @map("expires_at") @db.Timestamptz(6)
    createdAt   DateTime     @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt   DateTime     @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    pairing     PadelPairing  @relation(fields: [pairingId], references: [id], onDelete: Cascade)

    @@unique([pairingId, status], map: "padel_pairing_holds_active_unique")
    @@index([pairingId])
    @@index([eventId])
}

```

```

@@map("padel_pairing_holds")
@schema("app_v3")
}

model TournamentEntry {
    id      Int          @id @default(autoincrement())
    eventId Int          @map("event_id")
    userId  String       @map("user_id") @db.Uuid
    pairingId Int?       @map("pairing_id")
    role    TournamentEntryRole @default(PARTNER)
    status   TournamentEntryStatus @default(PENDING)
    ownerUserId String?   @map("owner_user_id") @db.Uuid
    ownerIdentityId String?   @map("owner_identity_id") @db.Uuid
    purchaseId String?   @map("purchase_id") @db.Text
    saleSummaryId Int?     @map("sale_summary_id")
    emissionIndex Int      @default(0) @map("emission_index")
    createdAt DateTime    @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime    @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    event    Event         @relation(fields: [eventId], references: [id], onDelete: Cascade)
    user    Profile        @relation(fields: [userId], references: [id], onDelete: Cascade)
    pairing  PadelPairing? @relation(fields: [pairingId], references: [id], onDelete: Cascade)
    tickets  Ticket[]     @relation("TicketTournamentEntry")
    saleSummary SaleSummary? @relation(fields: [saleSummaryId], references: [id], onDelete: SetNull)

    @unique([eventId, userId], map: "tournament_entries_event_user_unique")
    @unique([purchaseId, emissionIndex], map: "tournament_entries_purchase_idx")
    @index([pairingId])
    @map("tournament_entries")
    @schema("app_v3")
}

/// Slots de cada pairing (um por jogador)
model PadelPairingSlot {
    id      Int          @id @default(autoincrement())
    pairingId Int          @map("pairing_id")
    ticketId String?   @unique @map("ticket_id")
    profileId String?   @map("profile_id") @db.Uuid
    slot_role PadelPairingSlotRole
    slotStatus PadelPairingSlotStatus @default(PENDING) @map("slot_status")
    paymentStatus PadelPairingPaymentStatus @default(UNPAID) @map("payment_status")
    invitedContact String?   @map("invited_contact")
    isPublicOpen Boolean    @default(false) @map("is_public_open")
    createdAt DateTime    @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime    @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    playerProfileId Int?     @map("player_profile_id")
    pairing    PadelPairing   @relation(fields: [pairingId], references: [id], onDelete: Cascade)
    playerProfile PadelPlayerProfile? @relation(fields: [playerProfileId], references: [id], onUpdate: NoAction, map:
    "padel_pairing_slots_player_profile_fk")
    profile   Profile?     @relation(fields: [profileId], references: [id])
    ticket    Ticket?      @relation("TicketPairingSlot", fields: [ticketId], references: [id])

    @index([pairingId])
    @index([profileId])
    @index([playerProfileId], map: "padel_pairing_slots_player_profile_idx")
    @map("padel_pairing_slots")
    @schema("app_v3")
}

/// Equipas / duplas
model PadelTeam {
    id      Int          @id @default(autoincrement())
    eventId Int          @map("event_id")
    categoryId Int?       @map("category_id")
    player1Id Int?       @map("player1_id")
    player2Id Int?       @map("player2_id")
    isFromMatchmaking Boolean    @default(false) @map("is_from_matchmaking")
    createdAt DateTime    @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime    @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    matchesA PadelMatch[] @relation("PadelMatchTeamA")
    matchesB PadelMatch[] @relation("PadelMatchTeamB")
    category PadelCategory? @relation(fields: [categoryId], references: [id], onUpdate: NoAction)
    event    Event         @relation(fields: [eventId], references: [id], onDelete: Cascade, onUpdate: NoAction)
    player1  PadelPlayerProfile? @relation("PadelTeamPlayer1", fields: [player1Id], references: [id], onUpdate: NoAction)
    player2  PadelPlayerProfile? @relation("PadelTeamPlayer2", fields: [player2Id], references: [id], onUpdate: NoAction)

    @map("padel_teams")
    @schema("app_v3")
}

```

```

/// Jogos
model PadelMatch {
    id          Int      @id @default(autoincrement())
    eventId     Int      @map("event_id")
    categoryId  Int?    @map("category_id")
    courtNumber Int?    @map("court_number")
    startTime   DateTime @map("start_time") @db.Timestamptz(6)
    roundLabel  String? @map("round_label")
    teamAId    Int?    @map("team_a_id")
    teamBId    Int?    @map("team_b_id")
    score       Json     @default("{}")
    status      padel_match_status @default(PENDING)
    createdAt   DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt   DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    pairingAId Int?    @map("pairing_a_id")
    pairingBId Int?    @map("pairing_b_id")
    groupLabel  String? @map("group_label")
    roundType   String? @map("round_type")
    winnerPairingId Int? @map("winner_pairing_id")
    scoreSets   Json?   @map("score_sets")
    courtName   String? @map("court_name")
    category    PadelCategory? @relation(fields: [categoryId], references: [id], onUpdate: NoAction)
    event       Event    @relation(fields: [eventId], references: [id], onDelete: Cascade, onUpdate: NoAction)
    pairingA   PadelPairing? @relation("MatchPairingA", fields: [pairingAId], references: [id], onUpdate: NoAction, map:
    "padel_matches_pairing_a_fk")
    pairingB   PadelPairing? @relation("MatchPairingB", fields: [pairingBId], references: [id], onUpdate: NoAction, map:
    "padel_matches_pairing_b_fk")
    teamA      PadelTeam? @relation("PadelMatchTeamA", fields: [teamAId], references: [id], onUpdate: NoAction)
    teamB      PadelTeam? @relation("PadelMatchTeamB", fields: [teamBId], references: [id], onUpdate: NoAction)
    winnerPairing PadelPairing? @relation("MatchWinnerPairing", fields: [winnerPairingId], references: [id], onUpdate:
    NoAction, map: "padel_matches_winner_fk")

    @@index([pairingAId, pairingBId], map: "padel_matches_pairings_idx")
    @@map("padel_matches")
    @@schema("app_v3")
}

/// Ranking básico por torneio/clube
model PadelRankingEntry {
    id          Int      @id @default(autoincrement())
    organizerId Int      @map("organizer_id")
    playerId    Int      @map("player_id")
    eventId     Int      @map("event_id")
    points      Int      @default(0)
    position    Int?
    level       String?
    season      String?
    year        Int?
    createdAt   DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    event       Event    @relation(fields: [eventId], references: [id], onDelete: Cascade, onUpdate: NoAction)
    organizer   Organizer @relation(fields: [organizerId], references: [id], onDelete: Cascade, onUpdate: NoAction)
    player      PadelPlayerProfile @relation(fields: [playerId], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @@map("padel_ranking_entries")
    @@schema("app_v3")
}

/// Estrutura base de torneios (formatos/chaves)
model Tournament {
    id          Int      @id @default(autoincrement())
    eventId     Int      @unique @map("event_id")
    format      TournamentFormat
    generationSeed String? @map("generation_seed")
    generatedAt DateTime? @map("generated_at") @db.Timestamptz(6)
    generatedByUserId String? @map("generated_by_user_id") @db.Uuid
    inscriptionDeadlineAt DateTime? @map("inscription_deadline_at") @db.Timestamptz(6)
    tieBreakRules Json    @default("{}") @map("tie_break_rules")
    config      Json    @default("{}")
    createdAt   DateTime @default(now()) @map("created_at")
    updatedAt   DateTime @default(now()) @updatedAt @map("updated_at")
    stages      TournamentStage[]
    auditLogs   TournamentAuditLog[]
    generatedBy Profile? @relation("TournamentGeneratedBy", fields: [generatedByUserId], references: [id], onUpdate: NoAction, onDelete: SetNull)
    event       Event    @relation("EventTournament", fields: [eventId], references: [id], onDelete:
    Cascade)
}

```

```

@@map("tournaments")
@schema("app_v3")
}

model TournamentStage {
    id      Int          @id @default(autoincrement())
    tournamentId Int      @map("tournament_id")
    name     String?
    stageType TournamentStageType @default(GROUPS) @map("stage_type")
    order    Int          @default(0)
    createdAt DateTime     @default(now()) @map("created_at")
    updatedAt DateTime     @default(now()) @updatedAt @map("updated_at")
    tournament Tournament   @relation(fields: [tournamentId], references: [id], onDelete: Cascade)
    groups   TournamentGroup[]
    matches   TournamentMatch[]

    @@index([tournamentId])
    @@map("tournament_stages")
    @schema("app_v3")
}

model TournamentGroup {
    id      Int          @id @default(autoincrement())
    stageId Int          @map("stage_id")
    name     String?
    order    Int          @default(0)
    createdAt DateTime     @default(now()) @map("created_at")
    updatedAt DateTime     @default(now()) @updatedAt @map("updated_at")
    stage    TournamentStage @relation(fields: [stageId], references: [id], onDelete: Cascade)
    matches   TournamentMatch[]

    @@index([stageId])
    @@map("tournament_groups")
    @schema("app_v3")
}

model TournamentMatch {
    id      Int          @id @default(autoincrement())
    stageId Int          @map("stage_id")
    groupId  Int?        @map("group_id")
    pairing1Id Int?      @map("pairing1_id")
    pairing2Id Int?      @map("pairing2_id")
    round    Int?        @map("round_number")
    roundLabel String?   @map("round_label")
    nextMatchId Int?     @map("next_match_id")
    nextSlot   Int?      @map("next_slot")
    courtId   Int?      @map("court_id")
    startAt   DateTime?  @map("start_at") @db.Timestamptz(6)
    score     Json         @default("{}")
    status    TournamentMatchStatus @default(PENDING)
    createdAt  DateTime     @default(now()) @map("created_at")
    updatedAt  DateTime     @default(now()) @updatedAt @map("updated_at")
    stage    TournamentStage @relation(fields: [stageId], references: [id], onDelete: Cascade)
    group    TournamentGroup? @relation(fields: [groupId], references: [id], onDelete: SetNull)

    @@index([stageId])
    @@index([groupId])
    @@map("tournament_matches")
    @schema("app_v3")
}

model TournamentAuditLog {
    id      Int          @id @default(autoincrement())
    tournamentId Int      @map("tournament_id")
    userId   String?     @map("user_id") @db.Uuid
    action   String
    payloadBefore Json?   @map("payload_before")
    payloadAfter  Json?   @map("payload_after")
    createdAt  DateTime     @default(now()) @map("created_at") @db.Timestamptz(6)
    tournament Tournament @relation(fields: [tournamentId], references: [id], onDelete: Cascade)
    user     Profile?     @relation("TournamentAuditLogUser", fields: [userId], references: [id], onUpdate: NoAction, onDelete: SetNull)

    @@index([tournamentId])
    @@index([userId])
    @@map("tournament_audit_logs")
    @schema("app_v3")
}

```

```

model follows {
    id           Int      @id @default(autoincrement())
    follower_id  String   @db.Uuid
    following_id String   @db.Uuid
    created_at   DateTime @default(now()) @db.Timestamptz(6)
    profiles_follows_follower_idToprofiles Profile @relation("follows_follower_idToprofiles", fields: [follower_id], references: [id], onDelete: Cascade, onUpdate: NoAction)
    profiles_follows_following_idToprofiles Profile @relation("follows_following_idToprofiles", fields: [following_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @@unique([follower_id, following_id], map: "follows_unique")
    @@index([follower_id], map: "idx_follows_follower")
    @@index([following_id], map: "idx_follows_following")
    @@schema("app_v3")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-security for more info.
model audit_log_entries {
    instance_id String? @db.Uuid
    id          String   @id @db.Uuid
    payload     Json?
    created_at  DateTime? @db.Timestamptz(6)
    ip_address  String   @default("") @db.VarChar(64)

    @@index([instance_id], map: "audit_logs_instance_id_idx")
    @@schema("auth")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-security for more info.
model flow_state {
    id           String      @id @db.Uuid
    user_id     String?     @db.Uuid
    auth_code   String
    code_challenge_method code_challenge_method
    code_challenge String
    provider_type String
    provider_access_token String?
    provider_refresh_token String?
    created_at   DateTime?   @db.Timestamptz(6)
    updated_at   DateTime?   @db.Timestamptz(6)
    authentication_method String
    auth_code_issued_at DateTime?   @db.Timestamptz(6)
    saml_relay_states   saml_relay_states[]

    @@index([created_at(sort: Desc)])
    @@index([auth_code], map: "idx_auth_code")
    @@index([user_id, authentication_method], map: "idx_user_id_auth_method")
    @@schema("auth")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-security for more info.
model identities {
    provider_id  String
    user_id      String   @db.Uuid
    identity_data Json
    provider     String
    last_sign_in_at DateTime? @db.Timestamptz(6)
    created_at   DateTime? @db.Timestamptz(6)
    updated_at   DateTime? @db.Timestamptz(6)
    email        String?  @default(dbgenerated("lower((identity_data -> 'email')::text)"))
    id          String   @id @default(dbgenerated("gen_random_uuid()")) @db.Uuid
    users        users    @relation(fields: [user_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @@unique([provider_id, provider], map: "identities_provider_id_provider_unique")
    @@index([email])
    @@index([user_id])
    @@schema("auth")
}

```

```

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read
more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-
security for more info.
model instances {
    id          String      @id @db.Uuid
    uuid        String?     @db.Uuid
    raw_base_config String?
    created_at   DateTime?  @db.Timestamptz(6)
    updated_at   DateTime?  @db.Timestamptz(6)

    @@schema("auth")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read
more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-
security for more info.
model mfa_amr_claims {
    session_id      String      @db.Uuid
    created_at      DateTime   @db.Timestamptz(6)
    updated_at      DateTime   @db.Timestamptz(6)
    authentication_method String
    id              String      @id(map: "amr_id_pk") @db.Uuid
    sessions        sessions @relation(fields: [session_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @@unique([session_id, authentication_method], map: "mfa_amr_claims_session_id_authentication_method_pkey")
    @@schema("auth")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read
more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-
security for more info.
model mfa_challenges {
    id          String      @id @db.Uuid
    factor_id    String      @db.Uuid
    created_at   DateTime   @db.Timestamptz(6)
    verified_at  DateTime?  @db.Timestamptz(6)
    ip_address   String      @db.Inet
    otp_code     String?
    web_authn_session_data Json?
    mfa_factors  mfa_factors @relation(fields: [factor_id], references: [id], onDelete: Cascade, onUpdate: NoAction,
map: "mfa_challenges_auth_factor_id_fkey")

    @@index([created_at(sort: Desc)], map: "mfa_challenge_created_at_idx")
    @@schema("auth")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read
more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-
security for more info.
model mfa_factors {
    id          String      @id @db.Uuid
    user_id     String      @db.Uuid
    friendly_name String?
    factor_type factor_type
    status       factor_status
    created_at   DateTime   @db.Timestamptz(6)
    updated_at   DateTime   @db.Timestamptz(6)
    secret       String?
    phone        String?
    last_challenged_at DateTime?  @unique @db.Timestamptz(6)
    web_authn_credential Json?
    web_authn_aaguid String?     @db.Uuid
    last_webauthn_challenge_data Json?
    mfa_challenges mfa_challenges[]
    users        users @relation(fields: [user_id], references: [id], onDelete: Cascade, onUpdate:
NoAction)

    @@unique([user_id, phone], map: "unique_phone_factor_per_user")
    @@index([user_id, created_at], map: "factor_id_created_at_idx")
    @@index([user_id])
    @@schema("auth")
}

```

```

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-
constraints for more info.
model oauth_authorizations {
    id          String      @id @db.Uuid
    authorization_id String    @unique
    client_id    String      @db.Uuid
    user_id      String?     @db.Uuid
    redirect_uri String
    scope        String
    state        String?
    resource     String?
    code_challenge String?
    code_challenge_method code_challenge_method?
    response_type   oauth_response_type    @default(code)
    status         oauth_authorization_status @default(pending)
    authorization_code String?      @unique
    created_at     DateTime     @default(now()) @db.Timestamptz(6)
    expires_at     DateTime     @default(dbgenerated("now() + '00:03:00'::interval")) @db.Timestamptz(6)
    approved_at    DateTime?    @db.Timestamptz(6)
    nonce         String?
    oauth_clients  oauth_clients @relation(fields: [client_id], references: [id], onDelete: Cascade, onUpdate:
NoAction)
    users          users?      @relation(fields: [user_id], references: [id], onDelete: Cascade, onUpdate:
NoAction)

    @@schema("auth")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read
more: https://pris.ly/d/database-comments
model oauth_client_states {
    id          String      @id @db.Uuid
    provider_type String
    code_verifier String?
    created_at     DateTime @db.Timestamptz(6)

    @@index([created_at], map: "idx_oauth_client_states_created_at")
    @@schema("auth")
}

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-
constraints for more info.
model oauth_clients {
    id          String      @id @db.Uuid
    client_secret_hash String?
    registration_type oauth_registration_type
    redirect_uris    String
    grant_types      String
    client_name      String?
    client_uri       String?
    logo_uri         String?
    created_at       DateTime     @default(now()) @db.Timestamptz(6)
    updated_at       DateTime     @default(now()) @db.Timestamptz(6)
    deleted_at       DateTime?    @db.Timestamptz(6)
    client_type      oauth_client_type    @default(confidential)
    oauth_authorizations oauth_authorizations[]
    oauth_consents    oauth_consents[]
    sessions         sessions[]

    @@index([deleted_at])
    @@schema("auth")
}

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-
constraints for more info.
model oauth_consents {
    id          String      @id @db.Uuid
    user_id    String      @db.Uuid
    client_id  String      @db.Uuid
    scopes      String
    granted_at  DateTime     @default(now()) @db.Timestamptz(6)
    revoked_at  DateTime?    @db.Timestamptz(6)
    oauth_clients oauth_clients @relation(fields: [client_id], references: [id], onDelete: Cascade, onUpdate: NoAction)
    users       users      @relation(fields: [user_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @@unique([user_id, client_id], map: "oauth_consents_user_client_unique")
    @@index([user_id, granted_at(sort: Desc)], map: "oauth_consents_user_order_idx")
    @@schema("auth")
}

```

```

}

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-
constraints for more info.
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-
security for more info.
model one_time_tokens {
    id      String      @id @db.Uuid
    user_id String      @db.Uuid
    token_type one_time_token_type
    token_hash String
    relates_to String
    created_at DateTime     @default(now()) @db.Timestamp(6)
    updated_at DateTime     @default(now()) @db.Timestamp(6)
    users    users      @relation(fields: [user_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @@unique([user_id, token_type])
    @index([relates_to], map: "one_time_tokens_relates_to_hash_idx", type: Hash)
    @index([token_hash], map: "one_time_tokens_token_hash_hash_idx", type: Hash)
    @@schema("auth")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read
more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-
security for more info.
model refresh_tokens {
    instance_id String? @db.Uuid
    id          BigInt    @id @default(autoincrement())
    token       String?  @unique(map: "refresh_tokens_token_unique") @db.VarChar(255)
    user_id     String?  @db.VarChar(255)
    revoked     Boolean?
    created_at  DateTime? @db.Timestamptz(6)
    updated_at  DateTime? @db.Timestamptz(6)
    parent      String?  @db.VarChar(255)
    session_id String?  @db.Uuid
    sessions    sessions? @relation(fields: [session_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @@index([instance_id])
    @index([instance_id, user_id])
    @@index([parent])
    @index([session_id, revoked])
    @index([updated_at(sort: Desc)])
    @@schema("auth")
}

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-
constraints for more info.
/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read
more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-
security for more info.
model saml_providers {
    id      String      @id @db.Uuid
    sso_provider_id String      @db.Uuid
    entity_id String      @unique
    metadata_xml String
    metadata_url String?
    attribute_mapping Json?
    created_at  DateTime? @db.Timestamptz(6)
    updated_at  DateTime? @db.Timestamptz(6)
    name_id_format String?
    sso_providers sso_providers @relation(fields: [sso_provider_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

    @@index([sso_provider_id])
    @@schema("auth")
}

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-
constraints for more info.
/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read
more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-
security for more info.
model saml_relay_states {
    id      String      @id @db.Uuid
    sso_provider_id String      @db.Uuid
    request_id   String
}

```

```

for_email      String?
redirect_to    String?
created_at     DateTime?      @db.Timestamptz(6)
updated_at     DateTime?      @db.Timestamptz(6)
flow_state_id String?        @db.Uuid
flow_state     flow_state?   @relation(fields: [flow_state_id], references: [id], onDelete: Cascade, onUpdate: NoAction)
sso_providers sso_providers @relation(fields: [sso_provider_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

@@index([created_at(sort: Desc)])
@@index([for_email])
@@index([sso_provider_id])
@@schema("auth")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-security for more info.
model schema_migrations {
  version String @id @db.VarChar(255)

  @@schema("auth")
}

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-constraints for more info.
/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-security for more info.
model sessions {
  id          String        @id @db.Uuid
  user_id     String        @db.Uuid
  created_at  DateTime?    @db.Timestamptz(6)
  updated_at  DateTime?    @db.Timestamptz(6)
  factor_id   String?       @db.Uuid
  aal         aal_level?
  not_after   DateTime?    @db.Timestamptz(6)
  refreshed_at DateTime?    @db.Timestamp(6)
  user_agent  String?
  ip          String?       @db.Inet
  tag         String?
  oauth_client_id String?    @db.Uuid
  refresh_token_hmac_key String?
  refresh_token_counter BigInt?
  scopes       String?
  mfa_amr_claims mfa_amr_claims[]
  refresh_tokens refresh_tokens[]
  oauth_clients oauth_clients? @relation(fields: [oauth_client_id], references: [id], onDelete: Cascade, onUpdate: NoAction)
  users        users        @relation(fields: [user_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

  @@index([not_after(sort: Desc)])
  @@index([oauth_client_id])
  @@index([user_id])
  @@index([user_id, created_at], map: "user_id_created_at_idx")
  @@schema("auth")
}

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-constraints for more info.
/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-security for more info.
/// This model contains an expression index which requires additional setup for migrations. Visit https://pris.ly/d/expression-indexes for more info.
model sso_domains {
  id          String        @id @db.Uuid
  sso_provider_id String?    @db.Uuid
  domain      String?
  created_at  DateTime?    @db.Timestamptz(6)
  updated_at  DateTime?    @db.Timestamptz(6)
  sso_providers sso_providers @relation(fields: [sso_provider_id], references: [id], onDelete: Cascade, onUpdate: NoAction)

  @@index([sso_provider_id])
  @@schema("auth")
}

```

```

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-constraints for more info.
/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-security for more info.
/// This model contains an expression index which requires additional setup for migrations. Visit https://pris.ly/d/expression-indexes for more info.
model sso_providers {
  id          String      @id @db.Uuid
  resource_id String?
  created_at   DateTime?   @db.Timestamptz(6)
  updated_at   DateTime?   @db.Timestamptz(6)
  disabled     Boolean?
  saml_providers saml_providers[]
  saml_relay_states saml_relay_states[]
  sso_domains   sso_domains[]

  @@index([resource_id], map: "sso_providers_resource_id_pattern_idx")
  @@schema("auth")
}

/// This table contains check constraints and requires additional setup for migrations. Visit https://pris.ly/d/check-constraints for more info.
/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-security for more info.
/// This model contains an expression index which requires additional setup for migrations. Visit https://pris.ly/d/expression-indexes for more info.
model users {
  instance_id      String?      @db.Uuid
  id               String      @id @db.Uuid
  aud              String?
  role             String?
  email            String?
  encrypted_password String?
  email_confirmed_at DateTime?   @db.Timestamptz(6)
  invited_at       DateTime?   @db.Timestamptz(6)
  confirmation_token String?
  confirmation_sent_at DateTime?   @db.Timestamptz(6)
  recovery_token   String?
  recovery_sent_at DateTime?   @db.Timestamptz(6)
  email_change_token_new String?
  email_change     String?
  email_change_sent_at DateTime?   @db.Timestamptz(6)
  last_sign_in_at DateTime?   @db.Timestamptz(6)
  raw_app_meta_data Json?
  raw_user_meta_data Json?
  is_super_admin   Boolean?
  created_at       DateTime?   @db.Timestamptz(6)
  updated_at       DateTime?   @db.Timestamptz(6)
  phone            String?      @unique
  phone_confirmed_at DateTime?   @db.Timestamptz(6)
  phone_change     String?      @default("")
  phone_change_token String?      @default("") @db.VarChar(255)
  phone_change_sent_at DateTime?   @db.Timestamptz(6)
  confirmed_at     DateTime?   @default(dbgenerated("LEAST(email_confirmed_at, phone_confirmed_at)"))
@db.Timestamptz(6)
  email_change_token_current String?      @default("") @db.VarChar(255)
  email_change_confirm_status Int?        @default(0) @db.SmallInt
  banned_until       DateTime?   @db.Timestamptz(6)
  reauthentication_token String?      @default("") @db.VarChar(255)
  reauthentication_sent_at DateTime?   @db.Timestamptz(6)
  is_sso_user        Boolean?     @default(false)
  deleted_at         DateTime?   @db.Timestamptz(6)
  is_anonymous       Boolean?     @default(false)
  guest_ticket_links GuestTicketLink[]
  padel_player_profiles PadelPlayerProfile[]
  profiles           Profile?
  identities         identities[]
  mfa_factors        mfa_factors[]
  oauth_authorizations oauth_authorizations[]
  oauth_consents      oauth_consents[]
  one_time_tokens    one_time_tokens[]
  sessions           sessions[]
}

```

```

@@index([instance_id])
@@index([is_anonymous])
@@schema("auth")
}

model Entitlement {
    id      String      @id @default(dbgenerated("gen_random_uuid()")) @db.Uuid
    type   EntitlementType
    status EntitlementStatus @default(ACTIVE)
    ownerUserId String? @map("owner_user_id") @db.Uuid
    ownerIdentityId String? @map("owner_identity_id") @db.Uuid
    ownerKey String @map("owner_key")
    purchaseId String @map("purchase_id") @db.Text
    saleLineId Int @map("sale_line_id")
    eventId Int? @map("event_id")
    tournamentId Int? @map("tournament_id")
    seasonId Int? @map("season_id")
    snapshotTitle String @map("snapshot_title")
    snapshotCoverUrl String? @map("snapshot_cover_url")
    snapshotVenueName String? @map("snapshot_venue_name")
    snapshotStartAt DateTime @map("snapshot_start_at") @db.Timestamptz(6)
    snapshotTimezone String @map("snapshot_timezone")
    snapshotVersion Int @default(1) @map("snapshot_version")
    createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    checkins EntitlementCheckin[]
    qrTokens EntitlementQrToken[]
}

@@unique([purchaseId, saleLineId, ownerKey, type], map: "entitlements_purchase_sale_owner_type_key")
@@index([ownerKey, snapshotStartAt], map: "entitlements_owner_start_idx")
@@index([eventId], map: "entitlements_event_idx")
@@index([tournamentId], map: "entitlements_tournament_idx")
@@index([seasonId], map: "entitlements_season_idx")
@@index([status], map: "entitlements_status_idx")
@map("entitlements")
@@schema("app_v3")
}

model EntitlementCheckin {
    id      Int      @id @default(autoincrement())
    entitlementId String @map("entitlement_id") @db.Uuid
    eventId Int @map("event_id")
    deviceId String @map("device_id")
    resultCode CheckinResultCode @map("result_code")
    checkedInAt DateTime @default(now()) @map("checked_in_at") @db.Timestamptz(6)
    checkedInBy String? @map("checked_in_by") @db.Uuid
    purchaseId String @map("purchase_id") @db.Text
    createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    updatedAt DateTime @default(now()) @updatedAt @map("updated_at") @db.Timestamptz(6)
    entitlement Entitlement @relation(fields: [entitlementId], references: [id], onDelete: Cascade, onUpdate: NoAction,
map: "entitlement_checkins_entitlement_fkey")

@@unique([eventId, entitlementId], map: "entitlement_checkins_event_entitlement_key")
@@index([eventId], map: "entitlement_checkins_event_idx")
@@index([purchaseId], map: "entitlement_checkins_purchase_idx")
@map("entitlement_checkins")
@@schema("app_v3")
}

model EntitlementQrToken {
    id      BigInt    @id @default(autoincrement())
    tokenHash String @map("token_hash") @unique
    entitlementId String @map("entitlement_id") @db.Uuid
    expiresAt DateTime? @map("expires_at") @db.Timestamptz(6)
    createdAt DateTime @default(now()) @map("created_at") @db.Timestamptz(6)
    entitlement Entitlement @relation(fields: [entitlementId], references: [id], onDelete: Cascade, onUpdate: NoAction, map:
"entitlement_qr_tokens_entitlement_fkey")

@@index([entitlementId], map: "entitlement_qr_tokens_entitlement_idx")
@map("entitlement_qr_tokens")
@@schema("app_v3")
}

enum NotificationType {
    ORGANIZER_INVITE
    ORGANIZER_TRANSFER
    STAFF_INVITE
    STAFF_ROLE_CHANGE
}

```

```

EVENT_SALE
EVENT_PAYOUT_STATUS
STRIPE_STATUS
FRIEND_REQUEST
FRIEND_ACCEPT
EVENT_RemINDER
CHECKIN_READY
TICKET_SHARED
MARKETING_PROMO_ALERT
SYSTEM_ANNOUNCE
FOLLOWED_YOU
TICKET_TRANSFER_RECEIVED
TICKET_TRANSFER_ACCEPTED
TICKET_TRANSFER_DECLINED
CLUB_INVITE
NEW_EVENT_FROM_FOLLOWED_ORGANIZER

@@schema("app_v3")
}

enum NotificationPriority {
    LOW
    NORMAL
    HIGH

    @@schema("app_v3")
}

enum SaleSummaryStatus {
    PAID
    REFUNDED
    DISPUTED
    FAILED
    PROCESSING

    @@schema("app_v3")
}

enum OrgType {
    PLATFORM
    EXTERNAL

    @@schema("app_v3")
}

enum OrganizerMemberRole {
    OWNER
    CO_OWNER
    ADMIN
    STAFF
    VIEWER

    @@schema("app_v3")
}

enum PaymentMode {
    LIVE
    TEST

    @@schema("app_v3")
}

enum PaymentEventSource {
    WEBHOOK
    JOB
    API

    @@schema("app_v3")
}

enum OperationStatus {
    PENDING
    RUNNING
    SUCCEEDED
    FAILED
    DEAD_LETTER

    @@schema("app_v3")
}

```

```

}

enum PadelPreferredSide {
    ESQUERDA
    DIREITA
    QUALQUER

    @@schema("app_v3")
}

/// This model or at least one of its fields has comments in the database, and requires an additional setup for migrations: Read
more: https://pris.ly/d/database-comments
/// This model contains row level security and requires additional setup for migrations. Visit https://pris.ly/d/row-level-
security for more info.
enum OrganizerStatus {
    PENDING
    ACTIVE
    SUSPENDED

    @@schema("app_v3")
}

enum EventType {
    EXPERIENCE
    ORGANIZER_EVENT

    @@schema("app_v3")
}

enum Visibility {
    PUBLIC
    PRIVATE

    @@schema("app_v3")
}

enum AccountStatus {
    ACTIVE
    PENDING_DELETE
    DELETED

    @@schema("app_v3")
}

enum EventCategoryType {
    FESTA
    DESPORTO
    CONCERTO
    PALESTRA
    ARTE
    COMIDA
    DRINKS

    @@schema("app_v3")
}

enum EventTemplateType {
    PARTY
    SPORT
    VOLUNTEERING
    TALK
    OTHER
    PADEL

    @@schema("app_v3")
}

enum OrganizationKind {
    CLUBE_PADEL
    RESTAURANTE
    EMPRESA_EVENTOS
    ASSOCIACAO
    PESSOA_SINGULAR

    @@schema("app_v3")
}

enum EventStatus {

```

```

DRAFT
PUBLISHED
CANCELLED
FINISHED
DATE_CHANGED

@@schema("app_v3")
}

enum ResaleMode {
ALWAYS
AFTER SOLD OUT
DISABLED

@@schema("app_v3")
}

enum RefundFeePayer {
ORGANIZER
CUSTOMER

@@schema("app_v3")
}

enum RefundReason {
CANCELLED
DELETED
DATE_CHANGED

@@schema("app_v3")
}

enum PadelPaymentMode {
FULL
SPLIT

@@schema("app_v3")
}

enum PadelPairingLifecycleStatus {
PENDING_ONE_PAID
PENDING_PARTNER_PAYMENT
CONFIRMED_BOTH_PAID
CONFIRMED_CAPTAIN_FULL
CANCELLED_INCOMPLETE

@@schema("app_v3")
}

enum PadelPairingJoinMode {
INVITE_PARTNER
LOOKING_FOR_PARTNER

@@schema("app_v3")
}

enum PadelEligibilityType {
OPEN
MALE_ONLY
FEMALE_ONLY
MIXED

@@schema("app_v3")
}

enum PadelPairingStatus {
INCOMPLETE
COMPLETE
CANCELLED

@@schema("app_v3")
}

enum PadelPairingGuaranteeStatus {
NONE
ARMED
SCHEDULED
SUCCEEDED

```

```

REQUIRES_ACTION
FAILED
EXPIRED

@@schema("app_v3")
}

enum PadelPairingSlotStatus {
PENDING
FILLED
CANCELLED

@@schema("app_v3")
}

enum PadelPairingHoldStatus {
ACTIVE
CANCELLED
EXPIRED

@@schema("app_v3")
}

enum TournamentEntryRole {
CAPTAIN
PARTNER

@@schema("app_v3")
}

enum TournamentEntryStatus {
PENDING
CONFIRMED
CANCELLED

@@schema("app_v3")
}

enum TournamentFormat {
GROUPS_PLUS_PLAYOFF
DRAW_A_B
GROUPS_PLUS_FINALS_ALL_PLACES
CHAMPIONSHIP_ROUND_ROBIN
NONSTOP_ROUND_ROBIN
MANUAL

@@schema("app_v3")
}

enum TournamentStageType {
GROUPS
PLAYOFF
CONSOLATION
NONSTOP

@@schema("app_v3")
}

enum TournamentMatchStatus {
PENDING
SCHEDULED
IN_PROGRESS
DONE
CANCELLED

@@schema("app_v3")
}

enum PadelPairingPaymentStatus {
UNPAID
PAID

@@schema("app_v3")
}

enum PadelPairingSlotRole {
CAPTAIN
PARTNER

```

```
    @@schema("app_v3")
}

enum FeeMode {
  INCLUDED
  ADDED
  ON_TOP

    @@schema("app_v3")
}

enum PayoutMode {
  ORGANIZER
  PLATFORM

  @@schema("app_v3")
}

enum PromoType {
  PERCENTAGE
  FIXED

  @@schema("app_v3")
}

enum TicketTypeStatus {
  ON_SALE
  UPCOMING
  CLOSED
  SOLD_OUT

  @@schema("app_v3")
}

enum TicketStatus {
  ACTIVE
  USED
  REFUNDED
  TRANSFERRED
  RESALE_LISTED

  @@schema("app_v3")
}

enum ReservationStatus {
  ACTIVE
  COMPLETED
  EXPIRED
  CANCELED

  @@schema("app_v3")
}

enum StaffScope {
  GLOBAL
  EVENT

  @@schema("app_v3")
}

enum StaffRole {
  OWNER
  ADMIN
  STAFF
  CHECKIN

  @@schema("app_v3")
}

enum StaffStatus {
  PENDING
  ACCEPTED
  REVOKED

  @@schema("app_v3")
}
```

```

enum TransferStatus {
    PENDING
    ACCEPTED
    CANCELLED

    @@schema("app_v3")
}

enum ResaleStatus {
    LISTED
    SOLD
    CANCELLED

    @@schema("app_v3")
}

enum padel_format {
    TODOS_CONTRA_TODOS
    QUADRO_ELIMINATORIO
    GRUPOS_ELIMINATORIAS
    CAMPEONATO_LIGA
    QUADRO_AB
    NON_STOP

    @@schema("app_v3")
}

enum padel_match_status {
    PENDING
    IN_PROGRESS
    DONE
    CANCELLED

    @@schema("app_v3")
}

enum EntitlementStatus {
    ACTIVE
    USED
    REFUNDED
    REVOKED
    SUSPENDED

    @@schema("app_v3")
}

enum EntitlementType {
    EVENT_TICKET
    PADEL_ENTRY
    PASS
    SUBSCRIPTION_ACCESS
    FUTURE_TYPE

    @@schema("app_v3")
}

enum CheckinResultCode {
    OK
    ALREADY_USED
    INVALID
    REFUNDED
    REVOKED
    SUSPENDED
    NOT_ALLOWED
    OUTSIDE_WINDOW

    @@schema("app_v3")
}

enum Gender {
    MALE
    FEMALE

    @@schema("app_v3")
}

enum aal_level {
    aal1
}

```

```

aal2
aal3

@@schema("auth")
}

enum code_challenge_method {
  s256
  plain

@@schema("auth")
}

enum factor_status {
  unverified
  verified

@@schema("auth")
}

enum factor_type {
  totp
  webauthn
  phone

@@schema("auth")
}

enum oauth_authorization_status {
  pending
  approved
  denied
  expired

@@schema("auth")
}

enum oauth_client_type {
  public
  confidential

@@schema("auth")
}

enum oauth_registration_type {
  dynamic
  manual

@@schema("auth")
}

enum oauth_response_type {
  code

@@schema("auth")
}

enum one_time_token_type {
  confirmation_token
  reauthentication_token
  recovery_token
  email_change_token_new
  email_change_token_current
  phone_change_token

@@schema("auth")
}

```

prisma/schema.prisma (enum insertion)

```

enum SaleSummaryStatus {
  PAID
  PROCESSING
  REFUNDED
  DISPUTED

```

```
FAILED  
  @@schema("app_v3")  
}
```

README.md

This is a [Next.js](https://nextjs.org) project bootstrapped with [`create-next-app`](https://nextjs.org/docs/app/api-reference/cli/create-next-app).

Getting Started

First, run the development server:

```
```bash  
npm run dev
or
yarn dev
or
pnpm dev
or
bun dev
```

Open <http://localhost:3000> with your browser to see the result.

You can start editing the page by modifying `app/page.tsx`. The page auto-updates as you edit the file.

This project uses [next/font](#) to automatically optimize and load [Geist](#), a new font family for Vercel.

## Learn More

To learn more about Next.js, take a look at the following resources:

- [Next.js Documentation](#) - learn about Next.js features and API.
- [Learn Next.js](#) - an interactive Next.js tutorial.

You can check out [the Next.js GitHub repository](#) - your feedback and contributions are welcome!

## Deploy on Vercel

The easiest way to deploy your Next.js app is to use the [Vercel Platform](#) from the creators of Next.js.

Check out our [Next.js deployment documentation](#) for more details.

```

scripts/backfillSaleSummaries.js
```js  
#!/usr/bin/env node  
/**  
 * Constrói sale_summaries e sale_lines a partir de tickets + payment_events (sem tocar em intents já existentes).  
 * Uso: node scripts/backfillSaleSummaries.js  
*/  
  
const fs = require("fs");  
const path = require("path");  
const { PrismaClient } = require("@prisma/client");  
const { PrismaPg } = require("@prisma/adapter-pg");  
const { Pool } = require("pg");  
const Stripe = require("stripe");  
  
function loadEnvFile(file) {  
  if (!fs.existsSync(file)) return;  
  const content = fs.readFileSync(file, "utf8");  
  for (const line of content.split("\n")) {  
    const trimmed = line.trim();  
    if (!trimmed || trimmed.startsWith("#")) continue;  
    const eq = trimmed.indexOf("=");  
    if (eq === -1) continue;  
    const key = trimmed.slice(0, eq).trim();  
    let val = trimmed.slice(eq + 1).trim();  
    if ((val.startsWith('"'') && val.endsWith('"'')) || (val.startsWith(''''') && val.endsWith('''''))) {  
      val = val.slice(1, -1);  
    }  
    if (!(key in process.env)) {  
      process.env[key] = val;  
    }  
  }  
}  
loadEnvFile(path.resolve(__dirname, ".env"));
```

```

        }
    }
}

loadEnvFile(path.join(process.cwd(), ".env.local"));
loadEnvFile(path.join(process.cwd(), ".env"));

if (!process.env.DATABASE_URL) {
    console.error("Falta DATABASE_URL no ambiente.");
    process.exit(1);
}

const stripeSecret = process.env.STRIPE_SECRET_KEY;
const stripe =
    stripeSecret && stripeSecret.trim()
    ? new Stripe(stripeSecret, { apiVersion: "2024-09-30.acacia" })
    : null;

const pool = new Pool({
    connectionString: process.env.DATABASE_URL,
    ssl: process.env.NODE_ENV === "production" ? undefined : { rejectUnauthorized: false },
});
const adapter = new PrismaPg(pool);
const prisma = new PrismaClient({ adapter, log: ["error"] });

async function getStripeFeeReal(paymentIntentId) {
    if (!stripe) return null;
    try {
        const pi = await stripe.paymentIntents.retrieve(paymentIntentId, {
            expand: ["latest_charge.balance_transaction"],
        });
        const charge = pi.latest_charge;
        if (charge && typeof charge === "object" && charge.balance_transaction && typeof charge.balance_transaction === "object") {
            return charge.balance_transaction.fee ?? null;
        }
        return null;
    } catch (err) {
        console.warn(`[backfillSaleSummaries] Não foi possível obter fee real para ${paymentIntentId}:`, err.message);
        return null;
    }
}

async function main() {
    // Pegar intents sem sale_summary
    const tickets = await prisma.ticket.findMany({
        where: {
            stripePaymentIntentId: { not: null },
            status: { in: ["ACTIVE", "USED"] },
        },
        select: {
            id: true,
            stripePaymentIntentId: true,
            eventId: true,
            ticketTypeId: true,
            pricePaid: true,
            totalPaidCents: true,
            platformFeeCents: true,
            purchasedAt: true,
        },
    });
}

const existingSummaries = new Set(
(
    await prisma.saleSummary.findMany({
        select: { paymentIntentId: true },
    })
)
.map((s) => s.paymentIntentId)
.filter(Boolean),
);

const intentGroups = new Map();
tickets.forEach((t) => {
    if (!t.stripePaymentIntentId || existingSummaries.has(t.stripePaymentIntentId)) return;
    if (!intentGroups.has(t.stripePaymentIntentId)) {
        intentGroups.set(t.stripePaymentIntentId, []);
    }
    intentGroups.get(t.stripePaymentIntentId).push(t);
}

```

```

});

console.log(`Intents sem sale_summary: ${intentGroups.size}`);
let created = 0;
for (const [intentId, group] of intentGroups.entries()) {
  const eventId = group[0].eventId;
  const platformFee = group.reduce((sum, t) => sum + (t.platformFeeCents ?? 0), 0);
  const subtotal = group.reduce((sum, t) => sum + (t.pricePaid ?? 0), 0);
  const totalPaid = group.reduce((sum, t) => sum + (t.totalPaidCents ?? t.pricePaid ?? 0), 0);

  const pe = await prisma.paymentEvent.findFirst({
    where: { stripePaymentIntentId: intentId },
    select: { stripeFeeCents: true, platformFeeCents: true, amountCents: true, eventId: true },
  });
  const stripeFeeReal = pe?.stripeFeeCents ?? (await getStripeFeeReal(intentId));
  const stripeFee = stripeFeeReal ?? 0;
  const totalFees = platformFee + stripeFee;
  const totalCents = totalPaid + platformFee; // cliente pagou totalPaid; platform fee adicionada (caso ADDED)
  const net = Math.max(0, totalCents - totalFees);

  const linesByType = new Map();
  group.forEach((t) => {
    if (!linesByType.has(t.ticketTypeId)) {
      linesByType.set(t.ticketTypeId, { qty: 0, gross: 0, fee: 0 });
    }
    const entry = linesByType.get(t.ticketTypeId);
    entry.qty += 1;
    entry.gross += t.pricePaid ?? 0;
    entry.fee += t.platformFeeCents ?? 0;
    linesByType.set(t.ticketTypeId, entry);
  });

  try {
    const summary = await prisma.saleSummary.create({
      data: {
        paymentIntentId: intentId,
        eventId,
        userId: null,
        promoCodeId: null,
        subtotalCents: subtotal,
        discountCents: 0,
        platformFeeCents: platformFee,
        stripeFeeCents: stripeFee,
        totalCents: totalCents,
        netCents: net,
        feeMode: null,
        currency: "EUR",
        promoCodeSnapshot: null,
        promoLabelSnapshot: null,
        promoTypeSnapshot: null,
        promoValueSnapshot: null,
      },
    });
    for (const [ticketTypeId, entry] of linesByType.entries()) {
      await prisma.saleLine.create({
        data: {
          saleSummaryId: summary.id,
          eventId,
          ticketTypeId,
          promoCodeId: null,
          quantity: entry.qty,
          unitPriceCents: Math.round(entry.gross / entry.qty),
          discountPerUnitCents: 0,
          grossCents: entry.gross,
          netCents: Math.max(0, entry.gross - entry.fee),
          platformFeeCents: entry.fee,
          promoCodeSnapshot: null,
          promoLabelSnapshot: null,
          promoTypeSnapshot: null,
          promoValueSnapshot: null,
        },
      });
    }
    created += 1;
  } catch (err) {
    console.error(`[backfillSaleSummaries] Falha ao criar sale_summary para ${intentId}:`, err.message);
  }
}

```

```

    }

    console.log(`Sale_summaries criados: ${created}`);
}

main()
  .catch((e) => {
    console.error(e);
    process.exit(1);
})
  .finally(async () => {
    await prisma.$disconnect();
});

```

scripts/backfillStripeFees.js

```

#!/usr/bin/env node
/***
 * Backfill stripe_fee_cents usando balance_transaction real da Stripe.
 * Atualiza sale_summaries (stripe_fee_cents, net_cents) e payment_events (stripe_fee_cents).
 * Segura: processa em batches pequenos; se algum intent não tiver charge/fee, apenas regista aviso.
 */

const fs = require("fs");
const path = require("path");
const { PrismaClient } = require("@prisma/client");
const { PrismaPg } = require("@prisma/adapter-pg");
const { Pool } = require("pg");
const Stripe = require("stripe");

// Carregar env de .env.local e .env manualmente (sem depender de dotenv)
function loadEnvFile(file) {
  if (!fs.existsSync(file)) return;
  const content = fs.readFileSync(file, "utf8");
  for (const line of content.split("\n")) {
    const trimmed = line.trim();
    if (!trimmed || trimmed.startsWith("#")) continue;
    const eq = trimmed.indexOf "=";
    if (eq === -1) continue;
    const key = trimmed.slice(0, eq).trim();
    let val = trimmed.slice(eq + 1).trim();
    // remove quotes simples ou duplas envolventes
    if ((val.startsWith('') && val.endsWith('')) || (val.startsWith("') && val.endsWith("')))) {
      val = val.slice(1, -1);
    }
    if (!(key in process.env)) {
      process.env[key] = val;
    }
  }
}

loadEnvFile(path.join(process.cwd(), ".env.local"));
loadEnvFile(path.join(process.cwd(), ".env"));

if (!process.env.STRIPE_SECRET_KEY) {
  console.error("Falta STRIPE_SECRET_KEY no ambiente (.env.local)");
  process.exit(1);
}

const stripe = Stripe(process.env.STRIPE_SECRET_KEY, { apiVersion: "2024-09-30.acacia" });

const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  ssl:
    process.env.NODE_ENV === "production"
    ? undefined
    : { rejectUnauthorized: false },
});
const adapter = new PrismaPg(pool);
const prisma = new PrismaClient({ adapter, log: ["error"] });

async function main() {
  const batchSize = 200;
  const summaries = await prisma.saleSummary.findMany({
    where: { stripeFeeCents: 0 },
    select: {

```

```

    id: true,
    paymentIntentId: true,
    platformFeeCents: true,
    totalCents: true,
    netCents: true,
  },
  take: batchSize,
  orderBy: { id: "asc" },
});

console.log(`Encontradas ${summaries.length} sale_summaries sem stripe_fee_cents.`);
let ok = 0;
let skipped = 0;

for (const s of summaries) {
  if (!s.paymentIntentId) {
    console.warn(`SaleSummary ${s.id} sem paymentIntentId, a ignorar`);
    skipped += 1;
    continue;
  }

  try {
    const intent = await stripe.paymentIntents.retrieve(s.paymentIntentId, {
      expand: ["latest_charge.balance_transaction"],
    });
    let fee = null;
    const charge = intent.latest_charge;
    if (charge && typeof charge === "object" && charge.balance_transaction && typeof charge.balance_transaction === "object")
    {
      fee = charge.balance_transaction.fee ?? null;
    }
    if (fee == null) {
      console.warn(`PI ${s.paymentIntentId}: sem fee real (sem balance_transaction).`);
      skipped += 1;
      continue;
    }

    const platformFee = s.platformFeeCents ?? 0;
    const total = s.totalCents ?? 0;
    const net = Math.max(0, total - platformFee - fee);

    await prisma.saleSummary.update({
      where: { id: s.id },
      data: { stripeFeeCents: fee, netCents: net },
    });
    await prisma.paymentEvent.updateMany({
      where: { stripePaymentIntentId: s.paymentIntentId },
      data: { stripeFeeCents: fee },
    });
    ok += 1;
  } catch (err) {
    console.error(`Falha no PI ${s.paymentIntentId} (saleSummary ${s.id}):`, err);
    skipped += 1;
  }
}

console.log(`Backfill sale_summaries concluído. Atualizados: ${ok}. Skipped/erros: ${skipped}.`);

// Backfill payment_events stripe_fee_cents (quando não há sale_summary)
const pevents = await prisma.paymentEvent.findMany({
  where: { stripeFeeCents: null },
  select: { id: true, stripePaymentIntentId: true },
  take: batchSize,
  orderBy: { id: "asc" },
});

console.log(`Encontrados ${pevents.length} payment_events sem stripe_fee_cents.`);
let okPe = 0;
let skipPe = 0;
for (const p of pevents) {
  if (!p.stripePaymentIntentId) {
    skipPe++;
    continue;
  }

  try {
    const intent = await stripe.paymentIntents.retrieve(p.stripePaymentIntentId, {
      expand: ["latest_charge.balance_transaction"],
    });
  }

```

```

let fee = null;
const charge = intent.latest_charge;
if (charge && typeof charge === "object" && charge.balance_transaction && typeof charge.balance_transaction === "object")
{
    fee = charge.balance_transaction.fee ?? null;
}
if (fee == null) {
    console.warn(`PaymentEvent ${p.id} PI ${p.stripePaymentIntentId}: sem fee real.`);
    skipPe++;
    continue;
}
await prisma.paymentEvent.update({
    where: { id: p.id },
    data: { stripeFeeCents: fee },
});
okPe++;
} catch (err) {
    console.error(`Falha paymentEvent ${p.id} (${p.stripePaymentIntentId}):`, err);
    skipPe++;
}
}

console.log(`Backfill payment_events concluido. Atualizados: ${okPe}. Skipped/erros: ${skipPe}.`);
}

main()
.catch((e) => {
    console.error(e);
    process.exit(1);
})
.finally(async () => {
    await prisma.$disconnect();
});

```

scripts/check-slug.js

```

/* eslint-disable @typescript-eslint/no-require-imports */
// scripts/check-slug.js
// Helper para verificar se um slug existe na tabela app_v3.events

const { PrismaClient } = require("@prisma/client");
const { PrismaPg } = require("@prisma/adapter-pg");
require("dotenv").config();

const slug = process.argv[2];

if (!slug) {
    console.error("Uso: node scripts/check-slug.js <slug>");
    process.exit(1);
}

const prisma = new PrismaClient({
    adapter: new PrismaPg({ connectionString: process.env.DATABASE_URL }),
});

async function main() {
    const event = await prisma.event.findUnique({
        where: { slug },
        select: {
            id: true,
            slug: true,
            status: true,
            startsAt: true,
            endsAt: true,
            locationCity: true,
            organizerId: true,
        },
    });
    console.log(event);
}

main()
.catch((err) => {
    console.error(err);
    process.exit(1);
});

```

```

})
.finally(async () => {
  await prisma.$disconnect();
});

```

scripts/db/patch_purchase_anchor.sql

```

DO $$ 
BEGIN
  CREATE TYPE app_v3."PaymentEventSource" AS ENUM ('WEBHOOK', 'JOB', 'API');
EXCEPTION WHEN duplicate_object THEN NULL;
END$$;

ALTER TABLE app_v3.payment_events
  ADD COLUMN IF NOT EXISTS stripe_event_id text,
  ADD COLUMN IF NOT EXISTS purchase_id uuid,
  ADD COLUMN IF NOT EXISTS source app_v3."PaymentEventSource" NOT NULL DEFAULT 'WEBHOOK',
  ADD COLUMN IF NOT EXISTS dedupe_key text,
  ADD COLUMN IF NOT EXISTS attempt integer NOT NULL DEFAULT 1;

CREATE UNIQUE INDEX IF NOT EXISTS payment_events_stripe_event_id_key
  ON app_v3.payment_events(stripe_event_id)
  WHERE stripe_event_id IS NOT NULL;

CREATE UNIQUE INDEX IF NOT EXISTS payment_events_purchase_id_key
  ON app_v3.payment_events(purchase_id)
  WHERE purchase_id IS NOT NULL;

CREATE INDEX IF NOT EXISTS payment_events_dedupe_key_idx
  ON app_v3.payment_events(dedupe_key);

ALTER TABLE app_v3.sale_summaries
  ADD COLUMN IF NOT EXISTS purchase_id uuid;

CREATE UNIQUE INDEX IF NOT EXISTS sale_summaries_purchase_id_key
  ON app_v3.sale_summaries(purchase_id)
  WHERE purchase_id IS NOT NULL;

```

scripts/hard-delete-user.ts

```

import { config } from "dotenv";
import { createClient } from "@supabase/supabase-js";

// Load environment variables from .env.local (fallback to .env)
config({ path: ".env.local" });
config();

const url = process.env.SUPABASE_URL;
const serviceRole = process.env.SUPABASE_SERVICE_ROLE;
const email = process.argv[2];

if (!url || !serviceRole) {
  console.error("Missing SUPABASE_URL or SUPABASE_SERVICE_ROLE env.");
  process.exit(1);
}
if (!email) {
  console.error("Usage: ts-node hard-delete-user.ts user@example.com");
  process.exit(1);
}

async function main() {
  const supabase = createClient(url as string, serviceRole as string, {
    auth: { autoRefreshToken: false, persistSession: false },
  });

  const { data, error } = await supabase.auth.admin.listUsers({
    page: 1,
    perPage: 1000,
  });
  if (error) {
    console.error("listUsers error:", error);
    process.exit(1);
  }
}

```

```

const user = data?.users?.find(
  (u) => u.email?.toLowerCase() === email.toLowerCase(),
);
if (!user) {
  console.log("User not found for email", email);
  return;
}

const { error: delErr } = await supabase.auth.admin.deleteUser(user.id, false);

if (delErr) {
  console.error("deleteUser error:", delErr);
  process.exit(1);
}

console.log("Hard deleted user", user.id, "for email", email);
}

main();

```

scripts/seed_padel.ts

```

/*
 * Seed de teste para Padel (organizer + jogadores + evento + equipas/jogos).
 *
 * NOTA: não corre automaticamente. Usa:
 *   npx ts-node scripts/seed_padel.ts
 *
 * Antes de correr:
 *   - Define USER_ID_TEST (auth.users.id) de um utilizador existente.
 *   - Ajusta emails/usernames se necessário.
 */

import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {
  const userId = process.env.USER_ID_TEST;
  if (!userId) throw new Error("Define USER_ID_TEST com o id do utilizador (auth.users.id).");

  const organizer = await prisma.organizer.upsert({
    where: { username: "club-padel-demo" },
    update: {},
    create: {
      userId,
      username: "club-padel-demo",
      displayName: "Clube Padel Demo",
      businessName: "Clube Padel Demo",
      city: "Lisboa",
      entityType: "CLUBE",
      status: "ACTIVE",
    },
  });

  const playersData = [
    { fullName: "João Silva", email: "joao.silva@padel.com", level: "M3" },
    { fullName: "Maria Costa", email: "maria.costa@padel.com", level: "M3" },
    { fullName: "Ricardo Lopes", email: "ricardo.lopes@padel.com", level: "M4" },
    { fullName: "Ana Santos", email: "ana.santos@padel.com", level: "M4" },
  ];

  const players = await Promise.all(
    playersData.map((p) =>
      prisma.padelPlayerProfile.upsert({
        where: { organizerId_email: { organizerId: organizer.id, email: p.email! } },
        update: { fullName: p.fullName, level: p.level },
        create: { organizerId: organizer.id, fullName: p.fullName, email: p.email, level: p.level },
      }),
    ),
  );

  const event = await prisma.event.create({
    data: {
      slug: `torneio-padel-demo-${Math.random().toString(36).slice(2, 6)}`,
      title: "Torneio Padel Demo",
    }
  });
}
```

```

        description: "Seed de teste Padel",
        type: "ORGANIZER_EVENT",
        templateType: "PADEL",
        ownerUserId: userId,
        organizerId: organizer.id,
        startsAt: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000),
        endsAt: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000 + 3 * 60 * 60 * 1000),
        locationName: "Clube Demo",
        locationCity: "Lisboa",
        isFree: true,
        status: "PUBLISHED",
        resaleMode: "ALWAYS",
        feeMode: "INCLUDED",
        payoutMode: "ORGANIZER",
    },
});

await prisma.padelTournamentConfig.upsert({
    where: { eventId: event.id },
    create: {
        eventId: event.id,
        organizerId: organizer.id,
        format: "TODOS_CONTRA_TODOS",
        numberOfWorks: 2,
    },
    update: {},
});

const teamA = await prisma.padelTeam.create({
    data: { eventId: event.id, player1Id: players[0].id, player2Id: players[1].id },
});
const teamB = await prisma.padelTeam.create({
    data: { eventId: event.id, player1Id: players[2].id, player2Id: players[3].id },
});

await prisma.padelMatch.create({
    data: {
        eventId: event.id,
        teamAId: teamA.id,
        teamBId: teamB.id,
        status: "DONE",
        score: { sets: [{ teamA: 6, teamB: 4 }, { teamA: 6, teamB: 3 }] },
    },
});

console.log("Seed Padel criada com sucesso:", { organizerId: organizer.id, eventId: event.id });
}

main()
    .catch((e) => {
        console.error(e);
        process.exit(1);
    })
    .finally(async () => {
        await prisma.$disconnect();
    });

```

tsconfig.json

```
{
    "compilerOptions": {
        "target": "ES2020",
        "lib": [
            "dom",
            "dom.iterable",
            "esnext"
        ],
        "allowJs": true,
        "skipLibCheck": true,
        "strict": true,
        "forceConsistentCasingInFileNames": true,
        "noImplicitReturns": true,
        "noFallthroughCasesInSwitch": true,
        "noEmit": true,
        "esModuleInterop": true,
        "module": "esnext",
    }
}
```

```
"moduleResolution": "bundler",
"resolveJsonModule": true,
"isolatedModules": true,
"jsx": "react-jsx",
"incremental": true,
"plugins": [
  {
    "name": "next"
  }
],
"paths": {
  "@/*": [
    "./@"
  ]
}
},
"include": [
  "next-env.d.ts",
  "**/*.ts",
  "**/*.tsx",
  ".next/types/**/*.ts",
  ".next/dev/types/**/*.ts"
],
"exclude": [
  "node_modules",
  ".next"
]
}
```

types/next-shim.d.ts

```
import type { ForwardRefExoticComponent, RefAttributes, ComponentProps } from "react";
import type { AppRouterInstance } from "next/dist/shared/lib/app-router-context.shared-runtime";

declare module "next/navigation" {
  export function useRouter(): AppRouterInstance;
  export function redirect(url: string): never;
  export function useSearchParams(): URLSearchParams;
  export function usePathname(): string;
}

declare module "next/link" {
  type AnchorProps = ComponentProps<"a">;
  type LinkProps = {
    href: string;
    replace?: boolean;
    scroll?: boolean;
    prefetch?: boolean;
    shallow?: boolean;
    locale?: string | false;
  } & AnchorProps;

  const Link: ForwardRefExoticComponent<LinkProps & RefAttributes<HTMLAnchorElement>>;
  export default Link;
}
```

types/qrcode.d.ts

```
declare module "qrcode";
```