


Below are the files to add for full CRUD on **teams** in your dashboard. Paths assume `src/` at repo root and match your existing imports.

`src/app/components/DashboardPageComponents/AdminTeamsCRUD.tsx`

```
"use client";

import React, { useEffect, useMemo, useState } from "react";
import { Edit3, Plus, RefreshCw, Trash2 } from "lucide-react";

// Keep this small Team type local so it stays decoupled from server types
export type TeamRow = {
  id: number;
  name: string;
  logo: string | null;
  created_at: string | null;
};

function Logo({ src, alt }: { src: string | null; alt: string }) {
  if (!src) return (
    <div
      className="h-7 w-7 grid place-items-center rounded-full bg-zinc-800
text-[10px] text-white/60"
      title="No logo"
    >
      
    </div>
  );
  return (
    <img
      src={src}
      alt={alt}
      className="h-7 w-7 rounded-full object-contain ring-1 ring-white/10 bg-
white/5"
      loading="lazy"
    />
  );
}

function useTeams() {
  const [rows, setRows] = useState<TeamRow[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  const load = async () => {
    setLoading(true);
    setError(null);
    try {
      const res = await fetch("/api/teams", { credentials: "include" });
    }
  }
}
```

```

        if (!res.ok) throw new Error((await res.json()).error || `HTTP $
{res.status}`);
        const data = (await res.json()) as { teams: TeamRow[] };
        setRows(data.teams);
    } catch (e: any) {
        setError(e.message ?? String(e));
    } finally {
        setLoading(false);
    }
};

useEffect(() => {
    void load();
}, []);

return { rows, setRows, loading, error, load };
}

export default function AdminTeamsCRUD() {
    const { rows, setRows, loading, error, load } = useTeams();
    const [creating, setCreating] = useState(false);
    const [editingId, setEditingId] = useState<number | null>(null);
    const [q, setQ] = useState("");

    const filtered = useMemo(() => {
        const term = q.trim().toLowerCase();
        if (!term) return rows;
        return rows.filter((r) => r.name.toLowerCase().includes(term));
    }, [rows, q]);

    async function remove(id: number) {
        if (!confirm("Delete this team? This cannot be undone.")) return;
        try {
            const res = await fetch(`/api/teams/${id}`, { method: "DELETE",
credentials: "include" });
            if (!res.ok) throw new Error((await res.json()).error || `HTTP $
{res.status}`);
            setRows((prev) => prev.filter((r) => r.id !== id));
        } catch (e: any) {
            alert(e.message ?? String(e));
        }
    }

    return (
        <section className="space-y-4">
            <header className="flex flex-col gap-3 sm:flex-row sm:items-center
sm:justify-between">
                <h2 className="text-xl font-bold text-white">Teams</h2>
                <div className="flex flex-1 gap-2 sm:flex-none">
                    <input
                        placeholder="Search by name..."

```

```

        value={q}
        onChange={(e) => setQ(e.target.value)}
        className="flex-1 sm:flex-none sm:w-64 px-3 py-2 rounded-lg bg-
zinc-900 text-white border border-white/15"
      />
      <button
        onClick={load}
        className="inline-flex items-center gap-2 px-3 py-2 rounded-lg
border border-white/15 bg-zinc-900 text-white hover:bg-zinc-800"
      >
        <RefreshCw className="h-4 w-4" /> Refresh
      </button>
      <button
        onClick={() => {
          setCreating(true);
          setEditingId(null);
        }}
        className="inline-flex items-center gap-2 px-3 py-2 rounded-lg
border border-emerald-400/40 bg-emerald-700/30 text-white hover:bg-
emerald-700/50"
      >
        <Plus className="h-4 w-4" /> New team
      </button>
    </div>
  </header>

  {error && (
    <div
      className="rounded-lg border border-red-500/40 bg-red-900/30 p-3 text-
red-200">
      Error: {error}
    </div>
  )}

  {creating && (
    <TeamRowEditor
      key="create"
      onCancel={() => setCreating(false)}
      onSave={(created) => {
        setCreating(false);
        setRows((prev) => [created, ...prev]);
      }}
    />
  )}

  <div className="overflow-x-auto border border-white/10 rounded-xl">
    {loading ? (
      <p className="p-3 text-white/70">Loading...</p>
    ) : filtered.length === 0 ? (
      <p className="p-3 text-white/70">No teams found.</p>
    ) : (

```

```

<table className="min-w-full text-sm text-left text-white/90">
  <thead className="bg-zinc-900/70 text-white">
    <tr>
      <th className="px-3 py-2">Team</th>
      <th className="px-3 py-2">Created</th>
      <th className="px-3 py-2 text-right">Actions</th>
    </tr>
  </thead>
  <tbody>
    {filtered.map((r) => (
      <tr key={r.id} className="odd:bg-zinc-950/60 even:bg-
zinc-900/40 align-top">
        <td className="px-3 py-2">
          {editingId === r.id ? (
            <TeamRowEditor
              initial={r}
              onCancel={() => setEditingId(null)}
              onSave={
                (updated) => {
                  setEditingId(null);
                  setRows((prev) => prev.map((x) => (x.id ===
updated.id ? updated : x)));
                }
              }
            </>
          ) : (
            <div className="flex items-center gap-3">
              <Logo src={r.logo} alt={r.name} />
              <div>
                <div className="font-semibold">{r.name}</div>
                <div className="text-xs text-white/50">ID # {r.id}</
div>
              </div>
            </div>
          )}
        </td>
        <td className="px-3 py-2 whitespace-nowrap">
          {r.created_at ? new
Date(r.created_at).toLocaleString() : "-"}
        </td>
        <td className="px-3 py-2">
          {editingId !== r.id && (
            <div className="flex items-center gap-2 justify-end">
              <button
                onClick={() => {
                  setCreating(false);
                  setEditingId(r.id);
                }}
                className="inline-flex items-center gap-1 px-3
py-1.5 rounded-lg border border-white/15 bg-zinc-900 hover:bg-zinc-800"
              >
                <Edit3 className="h-4 w-4" /> Edit
              </button>
            </div>
          )}
        </td>
      </tr>
    )}
  </tbody>
</table>

```

```

        <button
          onClick={() => remove(r.id)}
          className="inline-flex items-center gap-1 px-3
py-1.5 rounded-lg border border-red-400/40 bg-red-900/30 hover:bg-red-900/50"
        >
          <Trash2 className="h-4 w-4" /> Delete
        </button>
      </div>
    )}
  </td>
</tr>
))}
</tbody>
</table>
)}
</div>
</section>
);
}

// ===== Inline editor =====
function TeamRowEditor({
  initial,
  onCancel,
  onSave,
}: {
  initial?: Partial<TeamRow>;
  onCancel: () => void;
  onSave: (saved: TeamRow) => void;
}) {
  const isEdit = Boolean(initial?.id);
  const [name, setName] = useState(initial?.name ?? "");
  const [logo, setLogo] = useState<string>(initial?.logo ?? "");
  const [saving, setSaving] = useState(false);

  const validationError = useMemo(() => {
    if (!name.trim()) return "Name is required";
    if (name.trim().length < 2) return "Name must be at least 2 characters";
    if (logo && !/^https?:\/\/\//i.test(logo)) return "Logo must be a full URL
(http://...)";
    return null;
  }, [name, logo]);

  async function save() {
    if (validationError) return;
    setSaving(true);
    try {
      const payload = { name: name.trim(), logo: logo.trim() || null };
      const res = await fetch(isEdit ? `/api/teams/${initial!.id}` : "/api/
teams", {
        method: isEdit ? "PATCH" : "POST",

```

```

        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(payload),
        credentials: "include",
    });
    if (!res.ok) throw new Error((await res.json()).error || `HTTP $
{res.status}`);
    const data = (await res.json()) as { team: TeamRow };
    onSave(data.team);
  } catch (e: any) {
    alert(e.message ?? String(e));
  } finally {
    setSaving(false);
  }
}

return (
  <div className="p-3 rounded-xl border border-white/15 bg-black/50 space-
y-3">
    <div className="grid grid-cols-1 md:grid-cols-2 gap-3">
      <label className="flex flex-col gap-1">
        <span className="text-sm text-white/80">Team name</span>
        <input
          value={name}
          onChange={(e) => setName(e.target.value)}
          className="px-3 py-2 rounded-lg bg-zinc-900 text-white border
border-white/10"
          placeholder="e.g. Athens City FC"
        />
      </label>
      <label className="flex flex-col gap-1">
        <span className="text-sm text-white/80">Logo URL (optional)</span>
        <input
          value={logo}
          onChange={(e) => setLogo(e.target.value)}
          className="px-3 py-2 rounded-lg bg-zinc-900 text-white border
border-white/10"
          placeholder="https://..."
        />
      </label>
    </div>

    {validationError && <p className="text-red-400 text-
sm">{validationError}</p>}

    <div className="flex gap-2 justify-end">
      <button
        type="button"
        onClick={onCancel}
        className="px-3 py-2 rounded-lg border border-white/15 text-white
bg-zinc-900 hover:bg-zinc-800"
      >

```

```

        Cancel
      </button>
      <button
        type="button"
        disabled={!validationError || saving}
        onClick={save}
        className="px-3 py-2 rounded-lg border border-emerald-400/40 text-
white bg-emerald-700/30 hover:bg-emerald-700/50 disabled:opacity-50"
      >
        {saving ? "Saving..." : isEdit ? "Save" : "Create"}
      </button>
    </div>
  </div>
);
}

```

src/app/api/teams/route.ts

```

import { NextResponse } from "next/server";
import { cookies } from "next/headers";
import { createServerClient } from "@supabase/ssr";
import { z } from "zod";

const TeamInput = z.object({
  name: z.string().trim().min(2).max(100),
  logo: z.string().url().nullable().optional(),
});

function createRouteClient() {
  const cookieStore = cookies();
  return createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.SUPABASE_ANON_KEY!,
    {
      cookies: {
        get(name: string) {
          return cookieStore.get(name)?.value;
        },
        set(name: string, value: string, options: any) {
          cookieStore.set({ name, value, ...options });
        },
        remove(name: string, options: any) {
          cookieStore.delete({ name, ...options });
        },
      },
    }
  );
}

```

```

async function assertAdmin() {
  const supabase = createRouteClient();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data.user) return NextResponse.json({ error:
    "Unauthorized" }, { status: 401 });
  const roles: string[] = Array.isArray(data.user.app_metadata?.roles)
    ? (data.user.app_metadata!.roles as string[])
    : [];
  if (!roles.includes("admin")) return NextResponse.json({ error:
    "Forbidden" }, { status: 403 });
  return supabase; // caller can continue using this instance
}

export async function GET() {
  const supabase = createRouteClient();
  const { data, error } = await supabase
    .from("teams")
    .select("id, name, logo, created_at")
    .order("name", { ascending: true });
  if (error) return NextResponse.json({ error: error.message }, { status:
    500 });
  return NextResponse.json({ teams: data ?? [] });
}

export async function POST(req: Request) {
  const supabase = await assertAdmin();
  if (supabase instanceof NextResponse) return supabase; // early return on
  401/403

  const json = await req.json().catch(() => null);
  const parsed = TeamInput.safeParse(json);
  if (!parsed.success) {
    return NextResponse.json({ error: parsed.error.issues[0]?.message ??
    "Invalid input" }, { status: 400 });
  }

  const payload = parsed.data;
  const { data, error } = await supabase
    .from("teams")
    .insert({ name: payload.name, logo: payload.logo ?? null })
    .select("id, name, logo, created_at")
    .single();

  if (error) return NextResponse.json({ error: error.message }, { status:
    400 });
  return NextResponse.json({ team: data }, { status: 201 });
}

```


src/app/api/teams/[id]/route.ts

```
import { NextResponse } from "next/server";
import { cookies } from "next/headers";
import { createServerClient } from "@supabase/ssr";
import { z } from "zod";

const TeamInput = z.object({
  name: z.string().trim().min(2).max(100).optional(),
  logo: z.string().url().nullable().optional(),
});

function createRouteClient() {
  const cookieStore = cookies();
  return createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.SUPABASE_ANON_KEY!,
    {
      cookies: {
        get(name: string) {
          return cookieStore.get(name)?.value;
        },
        set(name: string, value: string, options: any) {
          cookieStore.set({ name, value, ...options });
        },
        remove(name: string, options: any) {
          cookieStore.delete({ name, ...options });
        },
      },
    }
  );
}

async function assertAdmin() {
  const supabase = createRouteClient();
  const { data, error } = await supabase.auth.getUser();
  if (error || !data.user) return NextResponse.json({ error:
    "Unauthorized" }, { status: 401 });
  const roles: string[] = Array.isArray(data.user.app_metadata?.roles)
    ? (data.user.app_metadata!.roles as string[])
    : [];
  if (!roles.includes("admin")) return NextResponse.json({ error:
    "Forbidden" }, { status: 403 });
  return supabase;
}

export async function PATCH(req: Request, { params }: { params: { id:
  string } }) {
  const supabase = await assertAdmin();
  if (supabase instanceof NextResponse) return supabase;
}
```

```

const id = Number(params.id);
if (!Number.isFinite(id)) return NextResponse.json({ error: "Invalid id" }, { status: 400 });

const json = await req.json().catch(() => null);
const parsed = TeamInput.safeParse(json);
if (!parsed.success) {
  return NextResponse.json({ error: parsed.error.issues[0]?.message ?? "Invalid input" }, { status: 400 });
}

const payload = parsed.data;
const { data, error } = await supabase
  .from("teams")
  .update({ ...payload, updated_at: new Date().toISOString() })
  .eq("id", id)
  .select("id, name, logo, created_at")
  .single();

if (error) return NextResponse.json({ error: error.message }, { status: 400 });
return NextResponse.json({ team: data });
}

export async function DELETE(_req: Request, { params }: { params: { id: string } }) {
  const supabase = await assertAdmin();
  if (supabase instanceof NextResponse) return supabase;

  const id = Number(params.id);
  if (!Number.isFinite(id)) return NextResponse.json({ error: "Invalid id" }, { status: 400 });

  // Will fail with FK violations if team is referenced by matches/
  player_teams
  const { error } = await supabase.from("teams").delete().eq("id", id);
  if (error) return NextResponse.json({ error: error.message }, { status: 400 });
  return NextResponse.json({ ok: true });
}

```

Notes

- **Auth & RLS:** API routes use the *anon* client with user cookies, then explicitly require the `admin` role in `app_metadata.roles` before mutating. This mirrors your Dashboard access checks.
- **UI parity:** Styling and UX mirror your existing `MatchesDashboard` / `RowEditor` components so the dashboard feels cohesive.

- **Logos:** To keep things simple, the editor accepts a logo URL. If you want file uploads to Supabase Storage, we can extend the editor later.
- **Errors:** Foreign key constraints (e.g., deleting a team tied to a match) will surface as readable errors from Supabase.
- **Import:** `DashboardPage` already imports `AdminTeamsCRUD`, so once these files exist, the section will render.