```
// ============================
// TakeachefHome.com — MVP Monorepo (single app)
// Stack: Next.js 14 (App Router) + Prisma (Postgres) + Stripe Connect + NextAuth +
UploadThing + Zod + shadcn/ui
// ============================

// ---------- README (high level) ----------
/*
MVP Goals:
- Multi-role marketplace (Chef/Vendor, Customer, Admin)
- Listings with categories (Chefs, Events, Services, Rentals, Products)
- Booking + messaging + reviews
- Payments: Stripe Checkout + Stripe Connect (destination charges) for vendor payouts
- Admin moderation
- Mobile-first UI (shadcn/ui)

Quick Start:
1) Create Postgres DB (e.g., Neon/Supabase/Railway). Set DATABASE_URL.
2) Create Stripe account + Connect platform.
3) Create GitHub OAuth and/or Email provider for NextAuth.
4) Clone -> pnpm i -> pnpm prisma db push -> pnpm dev

Important ENV (create .env.local):
DATABASE_URL="postgresql://USER:PASS@HOST:PORT/DB?schema=public"
NEXTAUTH_URL="http://localhost:3000"
NEXTAUTH_SECRET="<openssl rand -base64 32>"
GITHUB_ID="..."
GITHUB_SECRET="..."
STRIPE_SECRET_KEY="sk_live_...or_test..."
STRIPE_WEBHOOK_SECRET="whsec_..."
STRIPE_CONNECT_CLIENT_ID="ca_..."
UPLOADTHING_TOKEN="..."
RESEND_API_KEY="..." // optional for email
SITE_URL="http://localhost:3000" // prod: https://takeachefhome.com

Runbook:
- pnpm dlx shadcn-ui@latest init
- pnpm dlx prisma generate
- pnpm dev
*/

// ---------- package.json ----------
export const packageJson = {
  "name": "takeachefhome",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "prisma:push": "prisma db push",
    "prisma:studio": "prisma studio"
  },
  "dependencies": {
    "@prisma/client": "^5.18.0",
    "@tanstack/react-query": "^5.56.2",
    "@uploadthing/react": "^6.6.0",
    "clsx": "^2.1.1",
    "lucide-react": "^0.456.0",
    "next": "14.2.6",
    "next-auth": "^5.0.0",
    "react": "18.3.1",
    "react-dom": "18.3.1",
    "zod": "^3.23.8",
    "stripe": "^16.8.0"
  },
  "devDependencies": {
    "@types/node": "^20.12.12",
    "@types/react": "^18.2.66",
    "@types/react-dom": "^18.2.22",
    "eslint": "^9.10.0",
    "eslint-config-next": "14.2.6",
    "postcss": "^8.4.33",
    "prisma": "^5.18.0",
    "tailwindcss": "^3.4.6",
    "typescript": "^5.5.4"
  }
};

// ---------- prisma/schema.prisma ----------
/*
model User {
  id            String   @id @default(cuid())
  name          String?
  email         String?  @unique
  image         String?
  role          Role     @default(CUSTOMER)
  stripeAccount String?  // Stripe Connect account id (for vendors)
  listings      Listing[]
  bookings      Booking[] @relation("UserBookings")
  messages      Message[]
  reviews       Review[]
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
}

enum Role {
  CUSTOMER
  VENDOR // chef/vendor/entertainer
  ADMIN
}

model Listing {
  id           String   @id @default(cuid())
  title        String
  slug         String   @unique
  description  String
  images       String[]
  category     Category
  priceCents   Int      // base price per event/hour/day/item
  unit         Unit     // PER_EVENT, PER_HOUR, PER_DAY, PER_ITEM
  location     String?
  ownerId      String
  owner        User     @relation(fields: [ownerId], references: [id])
  ratingAvg    Float?   @default(0)
  ratingCount  Int      @default(0)
  availability Json?    // future: calendar blocks
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt
  bookings     Booking[]
  reviews      Review[]
}

enum Category {
  CHEF
  EVENT
  SERVICE
  RENTAL
  PRODUCT
}
```

```
enum Unit {
  PER_EVENT
  PER_HOUR
  PER_DAY
  PER_ITEM
}

model Booking {
  id            String   @id @default(cuid())
  listingId     String
  listing       Listing  @relation(fields: [listingId], references: [id])
  customerId    String
  customer      User     @relation("UserBookings", fields: [customerId], references: [id])
  status        BookingStatus @default(PENDING)
  dateStart     DateTime
  dateEnd       DateTime?
  qty           Int      @default(1)
  subtotalCents Int
  feeCents      Int
  payoutCents   Int
  totalCents    Int
  paymentIntentId String?
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
}

enum BookingStatus {
  PENDING
  CONFIRMED
  CANCELED
  COMPLETED
}

model Message {
  id          String   @id @default(cuid())
  fromId      String
  toId        String
  listingId   String?
  body        String
  readAt      DateTime?
  createdAt   DateTime @default(now())
}
```

```
model Review {
  id         String   @id @default(cuid())
  listingId  String
  listing    Listing  @relation(fields: [listingId], references: [id])
  authorId   String
  author     User     @relation(fields: [authorId], references: [id])
  rating     Int
  comment    String?
  createdAt  DateTime @default(now())
}
*/

// ---------- app/(site)/layout.tsx ----------
export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html lang="en">
      <body className="min-h-screen bg-white text-gray-900">
        <header className="sticky top-0 z-50 border-b bg-white/80 backdrop-blur">
          <div className="mx-auto flex max-w-6xl items-center justify-between p-4">
            <a href="/" className="text-xl font-bold">TakeachefHome</a>
            <nav className="flex items-center gap-4">
              <a href="/browse" className="hover:underline">Browse</a>
              <a href="/create" className="hover:underline">Create Listing</a>
              <a href="/dashboard" className="hover:underline">Dashboard</a>
            </nav>
          </div>
        </header>
        <main className="mx-auto max-w-6xl p-4">{children}</main>
        <footer className="mt-12 border-t py-8 text-center text-sm">© {new Date().getFullYear()}
TakeachefHome.com</footer>
      </body>
    </html>
  );
}

// ---------- app/(site)/page.tsx (Homepage) ----------
export function HomePage() {
  return (
    <div className="space-y-10">
      <section className="rounded-2xl bg-gray-50 p-8">
        <h1 className="text-3xl font-semibold">Bring the chef, the vibe, and the experience —
home.</h1>
        <p className="mt-2 max-w-2xl text-gray-600">Book private chefs, events, services,
rentals, and more — all in one soulful marketplace.</p>
```

```
        <div className="mt-6 flex gap-3">
          <a href="/browse" className="rounded-xl border px-4 py-2">Browse Listings</a>
          <a href="/create" className="rounded-xl bg-black px-4 py-2 text-white">Create a
Listing</a>
        </div>
      </section>
      <section>
        <h2 className="mb-4 text-xl font-semibold">Popular Categories</h2>
        <div className="grid grid-cols-2 gap-4 md:grid-cols-4">
          {[
            { name: "Chefs", href: "/browse?cat=CHEF" },
            { name: "Events", href: "/browse?cat=EVENT" },
            { name: "Services", href: "/browse?cat=SERVICE" },
            { name: "Rentals", href: "/browse?cat=RENTAL" },
          ].map((c) => (
            <a key={c.name} href={c.href} className="rounded-2xl border p-6 hover:bg-gray-50">
              <p className="text-lg font-medium">{c.name}</p>
              <p className="text-sm text-gray-500">Explore curated {c.name.toLowerCase()} near
you.</p>
            </a>
          ))}
        </div>
      </section>
    </div>
  );
}

// ---------- app/browse/page.tsx ----------
// NOTE: Replace mock with DB fetch and URL param filters
export function BrowsePage() {
  const mock = [
    { id: "1", title: "Chef Savoir Faire — Luxe Dinner for 8", priceCents: 120000, category:
"CHEF", images: [] },
    { id: "2", title: "Soulful Brunch Pop-Up", priceCents: 60000, category: "EVENT", images: [] },
  ];
  return (
    <div>
      <h1 className="mb-4 text-2xl font-semibold">Browse</h1>
      <div className="grid grid-cols-1 gap-4 md:grid-cols-3">
        {mock.map((x) => (
          <a key={x.id} href={`/listing/${x.id}`} className="rounded-2xl border p-4">
            <div className="aspect-video rounded-xl bg-gray-100" />
            <div className="mt-3 font-medium">{x.title}</div>
            <div className="text-sm text-gray-600">${(x.priceCents / 100).toFixed(0)}+</div>
        </a>
      ))}
    </div>
  </div>
 );
}

// ---------- app/listing/[id]/page.tsx ----------
export function ListingPage({ params }: { params: { id: string } }) {
 const listing = { id: params.id, title: "Chef Savoir Faire — Luxe Dinner for 8", description: "A
soulful 4-course dining journey.", priceCents: 120000 };
 return (
  <div className="grid gap-8 md:grid-cols-5">
   <div className="md:col-span-3">
    <div className="aspect-video w-full rounded-2xl bg-gray-100" />
    <h1 className="mt-4 text-2xl font-semibold">{listing.title}</h1>
    <p className="mt-2 text-gray-700">{listing.description}</p>
   </div>
   <aside className="md:col-span-2">
    <div className="rounded-2xl border p-4">
     <div className="text-xl font-semibold">${(listing.priceCents/100).toFixed(0)}</div>
     <form className="mt-3 space-y-3">
      <input type="date" className="w-full rounded-lg border p-2" />
      <input type="time" className="w-full rounded-lg border p-2" />
      <button className="w-full rounded-lg bg-black p-2 text-white">Request
Booking</button>
     </form>
    </div>
   </aside>
  </div>
 );
}

// ---------- app/create/page.tsx (Create Listing) ----------
export function CreateListingPage() {
 return (
  <div className="max-w-2xl">
   <h1 className="mb-4 text-2xl font-semibold">Create a Listing</h1>
   <form className="space-y-4">
    <input placeholder="Title" className="w-full rounded-lg border p-2" />
    <textarea placeholder="Description" className="w-full rounded-lg border p-2" rows={6} />
    <div className="grid grid-cols-2 gap-4">
     <select className="rounded-lg border p-2">
      <option value="CHEF">Chef</option>
```

```
          <option value="EVENT">Event</option>
          <option value="SERVICE">Service</option>
          <option value="RENTAL">Rental</option>
          <option value="PRODUCT">Product</option>
        </select>
        <input type="number" placeholder="Base Price (USD)" className="rounded-lg border
p-2" />
      </div>
      <button className="rounded-xl bg-black px-4 py-2 text-white">Save Listing</button>
    </form>
   </div>
  );
}

// ---------- lib/db.ts ----------
/*
import { PrismaClient } from "@prisma/client";
export const prisma = globalThis.prisma || new PrismaClient();
if (process.env.NODE_ENV !== "production") globalThis.prisma = prisma;
*/

// ---------- lib/stripe.ts ----------
/*
import Stripe from "stripe";
export const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, { apiVersion:
"2024-06-20" });

export async function createCheckoutSession({ bookingId, amount, applicationFeeCents,
connectedAccountId }:{
  bookingId: string;
  amount: number;
  applicationFeeCents: number;
  connectedAccountId: string;
}){
  return await stripe.checkout.sessions.create({
    mode: "payment",
    payment_intent_data: {
      application_fee_amount: applicationFeeCents,
      transfer_data: { destination: connectedAccountId },
    },
    line_items: [{ price_data: { currency: "usd", product_data: { name: `Booking ${bookingId}` },
unit_amount: amount }, quantity: 1 }],
    success_url: `${process.env.SITE_URL}/booking/${bookingId}?status=success`,
    cancel_url: `${process.env.SITE_URL}/booking/${bookingId}?status=cancel`,
    metadata: { bookingId },
  });
}
*/

// ---------- app/api/listings/route.ts (POST create, GET browse) ----------
/*
import { prisma } from "@/lib/db";
import { NextRequest, NextResponse } from "next/server";
import { z } from "zod";

const CreateListing = z.object({
  title: z.string().min(3),
  description: z.string().min(20),
  category: z.enum(["CHEF","EVENT","SERVICE","RENTAL","PRODUCT"]),
  priceCents: z.number().int().positive(),
  unit: z.enum(["PER_EVENT","PER_HOUR","PER_DAY","PER_ITEM"]),
  images: z.array(z.string()).optional(),
  location: z.string().optional(),
});

export async function GET(req: NextRequest){
  const { searchParams } = new URL(req.url);
  const cat = searchParams.get("cat") as any;
  const data = await prisma.listing.findMany({
    where: { category: cat ?? undefined },
    orderBy: { createdAt: "desc" },
    take: 30,
  });
  return NextResponse.json({ data });
}

export async function POST(req: NextRequest){
  const body = await req.json();
  const parsed = CreateListing.safeParse(body);
  if(!parsed.success) return NextResponse.json({ error: parsed.error.flatten() }, { status: 400 });
  // TODO: get session user
  const ownerId = "SESSION_USER_ID";
  const slug = parsed.data.title.toLowerCase().replace(/[^a-z0-9]+/g, "-").replace(/(^-|-$)/g, "");
  const listing = await prisma.listing.create({ data: { ...parsed.data, slug, ownerId } });
  return NextResponse.json({ listing });
}
*/
```

```
// ---------- app/api/bookings/route.ts (POST create) ----------
/*
import { prisma } from "@/lib/db";
import { NextRequest, NextResponse } from "next/server";
import { z } from "zod";

const CreateBooking = z.object({
  listingId: z.string(),
  dateStart: z.string(),
  dateEnd: z.string().optional(),
  qty: z.number().int().positive().default(1),
});

export async function POST(req: NextRequest){
  const body = await req.json();
  const parsed = CreateBooking.safeParse(body);
  if(!parsed.success) return NextResponse.json({ error: parsed.error.flatten() }, { status: 400 });

  const listing = await prisma.listing.findUnique({ where: { id: parsed.data.listingId }, include: {
owner: true } });
  if(!listing || !listing.owner.stripeAccount) return NextResponse.json({ error: "Listing not
bookable" }, { status: 400 });

  // naive pricing: subtotal = listing.priceCents * qty
  const subtotal = listing.priceCents * (parsed.data.qty ?? 1);
  const fee = Math.round(subtotal * 0.12); // 12% platform fee
  const payout = subtotal - fee;

  const booking = await prisma.booking.create({ data: {
    listingId: listing.id,
    customerId: "SESSION_USER_ID",
    dateStart: new Date(parsed.data.dateStart),
    dateEnd: parsed.data.dateEnd ? new Date(parsed.data.dateEnd) : null,
    qty: parsed.data.qty ?? 1,
    subtotalCents: subtotal,
    feeCents: fee,
    payoutCents: payout,
    totalCents: subtotal,
  }});

  // TODO: create Stripe Checkout session using connect
  // const session = await createCheckoutSession({ bookingId: booking.id, amount:
booking.totalCents, applicationFeeCents: booking.feeCents, connectedAccountId:
listing.owner.stripeAccount! });
```

```
  return NextResponse.json({ booking /*, checkoutUrl: session.url */ });
}
*/

// ---------- app/api/webhooks/stripe/route.ts ----------
/*
import { NextRequest, NextResponse } from "next/server";
import { stripe } from "@/lib/stripe";
import { prisma } from "@/lib/db";

export async function POST(req: NextRequest){
  const sig = req.headers.get("stripe-signature");
  const raw = await req.text();
  let event;
  try {
    event = stripe.webhooks.constructEvent(raw, sig!,
process.env.STRIPE_WEBHOOK_SECRET!);
  } catch (err:any) {
    return NextResponse.json({ error: `Webhook Error: ${err.message}` }, { status: 400 });
  }

  if(event.type === "checkout.session.completed"){
    const session:any = event.data.object;
    const bookingId = session.metadata?.bookingId;
    if(bookingId){
      await prisma.booking.update({ where: { id: bookingId }, data: { status: "CONFIRMED",
paymentIntentId: session.payment_intent } });
    }
  }

  return NextResponse.json({ received: true });
}
*/

// ---------- TODO: Auth (NextAuth) skeleton ----------
/*
// app/api/auth/[...nextauth]/route.ts
import NextAuth from "next-auth";
import GitHub from "next-auth/providers/github";
export const { handlers, auth } = NextAuth({
  providers: [GitHub({ clientId: process.env.GITHUB_ID!, clientSecret:
process.env.GITHUB_SECRET! })],
  callbacks: { async session({ session, token }){ return session; } }
});
*/

// ---------- components/ListingCard.tsx ----------
export function ListingCard({ title, priceCents, href }: { title: string; priceCents: number; href:
string }){
  return (
    <a href={href} className="rounded-2xl border p-4 transition hover:shadow-sm">
      <div className="aspect-video rounded-xl bg-gray-100" />
      <div className="mt-3 font-medium">{title}</div>
      <div className="text-sm text-gray-600">${(priceCents/100).toFixed(0)}+</div>
    </a>
  );
}

// ---------- SECURITY & OPS CHECKLIST (what to stack next) ----------
/*
MUST-ADD NEXT:
1) Auth + RBAC: NextAuth with roles; protect create/dashboard routes.
2) Stripe Connect onboarding flow (account links) for vendors.
3) Image upload (UploadThing) + image optimization.
4) Input validation (Zod) on all APIs; rate limiting (Upstash Ratelimit).
5) Messaging (simple): messages table + UI thread; email notifications (Resend).
6) Reviews flow with post-purchase guard.
7) Admin dashboard: approve listings, ban users, refund flow.
8) SEO: metadata, OpenGraph, sitemap.xml, robots.txt, JSON-LD for listings.
9) Analytics & Error Tracking: Vercel Analytics + Sentry.
10) Content moderation: basic profanity/NSFW filter; report button.
11) Terms, Privacy, Refunds, Cookie consent (legal baseline).
12) Availability calendar (blocked dates), time zones.
13) Webhooks retry handling + idempotency keys for payments.
14) Email templates (booking request, confirmed, reminder, review invite).
15) Data privacy: PII minimal storage, audit logs for admin actions.
*/

// ---------- Admin notes ----------
/*
Platform Fee Strategy (tune later): 12% platform fee + Stripe fees. Featured Listings: $29/wk.
Vendor Pro: $39/mo, 0% extra fee.
Categories to pre-seed: CHEF, EVENT, SERVICE, RENTAL, PRODUCT.
*/


// ============================
// AUTH + RBAC ADD-ON (NextAuth)
// ============================

// prisma/schema.prisma (append NextAuth models + relations on User)
/*
// Add to existing User model:
//  accounts      Account[]
//  sessions      Session[]

model Account {
  id                String  @id @default(cuid())
  userId            String
  user              User    @relation(fields: [userId], references: [id], onDelete: Cascade)
  type              String
  provider          String
  providerAccountId String
  refresh_token     String?  @db.Text
  access_token      String?  @db.Text
  expires_at        Int?
  token_type        String?
  scope             String?
  id_token          String?  @db.Text
  session_state     String?

  @@unique([provider, providerAccountId])
}

model Session {
  id           String   @id @default(cuid())
  sessionToken String   @unique
  userId       String
  user         User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  expires      DateTime
}

model VerificationToken {
  identifier String
  token      String   @unique
  expires    DateTime

  @@unique([identifier, token])
}
*/
```

```
// lib/auth.ts
/*
import NextAuth, { DefaultSession } from "next-auth";
import { PrismaAdapter } from "@auth/prisma-adapter";
import GitHub from "next-auth/providers/github";
import Email from "next-auth/providers/nodemailer";
import { prisma } from "@/lib/db";

declare module "next-auth" {
  interface Session extends DefaultSession {
    user: {
      id: string;
      role: "CUSTOMER" | "VENDOR" | "ADMIN";
    } & DefaultSession["user"];
  }
}

export const authOptions = {
  adapter: PrismaAdapter(prisma),
  providers: [
    GitHub({ clientId: process.env.GITHUB_ID!, clientSecret: process.env.GITHUB_SECRET! }),
    Email({ server: process.env.EMAIL_SERVER!, from: process.env.EMAIL_FROM! }),
  ],
  session: { strategy: "database" as const },
  callbacks: {
    async session({ session, user }: any) {
      if (session.user) {
        session.user.id = user.id;
        session.user.role = user.role;
      }
      return session;
    },
  },
};
*/

// app/api/auth/[...nextauth]/route.ts
/*
import NextAuth from "next-auth";
import { authOptions } from "@/lib/auth";
const handler = NextAuth(authOptions as any);
export { handler as GET, handler as POST };
*/
```

```
// middleware.ts (RBAC gate)
/*
import { NextResponse } from "next/server";
import type { NextRequest } from "next/server";
import { getToken } from "next-auth/jwt";

const PROTECTED = ["/create", "/dashboard"]; // require auth
const ADMIN_ONLY = ["/admin"]; // require ADMIN

export async function middleware(req: NextRequest) {
  const { pathname } = req.nextUrl;
  const token = await getToken({ req, secret: process.env.NEXTAUTH_SECRET });

  if (PROTECTED.some((p) => pathname.startsWith(p))) {
    if (!token) return NextResponse.redirect(new URL("/api/auth/signin", req.url));
  }
  if (ADMIN_ONLY.some((p) => pathname.startsWith(p))) {
    if (!token || token.role !== "ADMIN") return NextResponse.redirect(new URL("/", req.url));
  }
  return NextResponse.next();
}

export const config = { matcher: ["/create/:path*", "/dashboard/:path*", "/admin/:path*"] };
*/

// lib/roles.ts
/*
export const canManageListings = (role: string) => role === "VENDOR" || role === "ADMIN";
*/

// README env additions (Auth)
/*
EMAIL_SERVER="smtp://USER:PASS@HOST:PORT"
EMAIL_FROM="TakeachefHome <noreply@takeachefhome.com>"
*/

// package.json deps add
/*
"@auth/prisma-adapter": "^1.7.4"
*/

// NEXT STEP: run
// pnpm i @auth/prisma-adapter next-auth
// pnpm prisma db push
// pnpm dev


// ============================
// STEP 2 — Stripe Connect Onboarding
// ============================

// lib/stripe.ts (real client)
/*
import Stripe from "stripe";
export const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, { apiVersion:
"2024-06-20" });

export async function createCheckoutSession({ bookingId, amount, applicationFeeCents,
connectedAccountId }:{
  bookingId: string;
  amount: number;
  applicationFeeCents: number;
  connectedAccountId: string;
}){
  return await stripe.checkout.sessions.create({
    mode: "payment",
    payment_intent_data: {
      application_fee_amount: applicationFeeCents,
      transfer_data: { destination: connectedAccountId },
    },
    line_items: [{ price_data: { currency: "usd", product_data: { name: `Booking ${bookingId}` },
unit_amount: amount }, quantity: 1 }],
    success_url: `${process.env.SITE_URL}/booking/${bookingId}?status=success`,
    cancel_url: `${process.env.SITE_URL}/booking/${bookingId}?status=cancel`,
    metadata: { bookingId },
  });
}
*/

// app/api/stripe/create-account/route.ts
/*
import { NextRequest, NextResponse } from "next/server";
import { stripe } from "@/lib/stripe";
import { prisma } from "@/lib/db";
import { getServerSession } from "next-auth";
import { authOptions } from "@/lib/auth";

export async function POST(req: NextRequest){
  const session = await getServerSession(authOptions as any);
  if(!session?.user?.id) return NextResponse.json({ error: "Unauthorized" }, { status: 401 });

  const user = await prisma.user.findUnique({ where: { id: session.user.id } });
  if(!user) return NextResponse.json({ error: "User not found" }, { status: 404 });

  // If already has a Connect account, just create an account link
  let accountId = user.stripeAccount;
  if(!accountId){
    const account = await stripe.accounts.create({ type: "express", email: user.email || undefined,
business_type: "individual" });
    accountId = account.id;
    await prisma.user.update({ where: { id: user.id }, data: { stripeAccount: accountId, role:
user.role === "CUSTOMER" ? "VENDOR" : user.role } });
  }

  const link = await stripe.accountLinks.create({
    account: accountId!,
    refresh_url: `${process.env.SITE_URL}/dashboard?connect=refresh`,
    return_url: `${process.env.SITE_URL}/dashboard?connect=return`,
    type: "account_onboarding",
  });
  return NextResponse.json({ url: link.url, accountId });
}
*/

// app/api/stripe/account-link/route.ts (generate fresh onboarding link)
/*
import { NextRequest, NextResponse } from "next/server";
import { stripe } from "@/lib/stripe";
import { prisma } from "@/lib/db";
import { getServerSession } from "next-auth";
import { authOptions } from "@/lib/auth";

export async function POST(req: NextRequest){
  const session = await getServerSession(authOptions as any);
  if(!session?.user?.id) return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
  const user = await prisma.user.findUnique({ where: { id: session.user.id } });
  if(!user?.stripeAccount) return NextResponse.json({ error: "No connected account" }, { status:
400 });

  const link = await stripe.accountLinks.create({
    account: user.stripeAccount,
    refresh_url: `${process.env.SITE_URL}/dashboard?connect=refresh`,
    return_url: `${process.env.SITE_URL}/dashboard?connect=return`,
    type: "account_onboarding",
  });
  return NextResponse.json({ url: link.url });
}
*/

// app/dashboard/page.tsx — vendor Stripe connect CTA
/*
import { prisma } from "@/lib/db";
import { getServerSession } from "next-auth";
import { authOptions } from "@/lib/auth";

export default async function Dashboard(){
  const session = await getServerSession(authOptions as any);
  if(!session) return <div>Please sign in.</div>;
  const me = await prisma.user.findUnique({ where: { id: session.user.id } });

  return (
    <div className="space-y-6">
      <h1 className="text-2xl font-semibold">Dashboard</h1>

      { (me?.role === "VENDOR" || me?.role === "ADMIN") && !me?.stripeAccount && (
        <div className="rounded-2xl border p-4">
          <h2 className="text-lg font-medium">Get paid with Stripe</h2>
          <p className="text-sm text-gray-600">Connect a Stripe Express account to receive
payouts for your bookings.</p>
          <form action="/api/stripe/create-account" method="post">
            <button className="mt-3 rounded-lg bg-black px-4 py-2 text-white">Start Stripe
Onboarding</button>
          </form>
        </div>
      )}

      {/* ...rest of vendor/customer dashboard ... */}
    </div>
  );
}
*/

// app/api/webhooks/stripe/route.ts — confirm bookings
/*
import { NextRequest, NextResponse } from "next/server";
import { stripe } from "@/lib/stripe";
import { prisma } from "@/lib/db";

export async function POST(req: NextRequest){
  const sig = req.headers.get("stripe-signature");
  const raw = await req.text();
  let event;
  try {
    event = stripe.webhooks.constructEvent(raw, sig!,
process.env.STRIPE_WEBHOOK_SECRET!);
  } catch (err:any) {
    return NextResponse.json({ error: `Webhook Error: ${err.message}` }, { status: 400 });
  }

  if(event.type === "checkout.session.completed"){
    const session:any = event.data.object;
    const bookingId = session.metadata?.bookingId;
    if(bookingId){
      await prisma.booking.update({ where: { id: bookingId }, data: { status: "CONFIRMED",
paymentIntentId: session.payment_intent } });
    }
  }
  return NextResponse.json({ received: true });
}
*/

// README additions — Stripe
/*
STRIPE_SECRET_KEY=sk_test_...
STRIPE_WEBHOOK_SECRET=whsec_...
STRIPE_CONNECT_CLIENT_ID=ca_...
SITE_URL=http://localhost:3000

Enable Connect (Express) in Stripe Dashboard. Set webhook endpoint:
https://yourdomain.com/api/webhooks/stripe with events: checkout.session.completed
*/


// ============================
// STEP 3 — Image Uploads (UploadThing)
// ============================

// Install (notes)
/*
 pnpm i uploadthing @uploadthing/react
```

```
*/

// app/api/uploadthing/core.ts — server config
/*
import { createUploadthing, type FileRouter } from "uploadthing/next";
import { UTApi } from "uploadthing/server";
import { getServerSession } from "next-auth";
import { authOptions } from "@/lib/auth";

const f = createUploadthing();

export const mediaRouter = {
  listingImage: f({ image: { maxFileSize: "8MB", maxFileCount: 6 } })
    .middleware(async () => {
      const session = await getServerSession(authOptions as any);
      if (!session?.user?.id) throw new Error("Unauthorized");
      return { userId: session.user.id };
    })
    .onUploadComplete(async ({ metadata, file }) => {
      // Optionally store file info in DB here
      console.log("Uploaded by", metadata.userId, file.url);
    }),
} satisfies FileRouter;

export type MediaRouter = typeof mediaRouter;
export const utapi = new UTApi();
*/

// app/api/uploadthing/route.ts — API handler
/*
import { createNextRouteHandler } from "uploadthing/next";
import { mediaRouter } from "./core";
export const { GET, POST } = createNextRouteHandler({ router: mediaRouter });
*/

// components/UploadBox.tsx — client uploader
/*
"use client";
import { UploadButton } from "@uploadthing/react";
import type { MediaRouter } from "@/app/api/uploadthing/core";

export function UploadBox({ onDone }:{ onDone: (urls: string[]) => void }){
  return (
    <div className="rounded-xl border p-4">
```

```
      <p className="text-sm text-gray-600">Upload up to 6 images</p>
      <UploadButton<MediaRouter>
        endpoint="listingImage"
        onClientUploadComplete={(res) => onDone(res?.map(r=>r.url) || [])}
        onUploadError={(e) => alert(e.message)}
      />
    </div>
  );
}
*/

// app/create/page.tsx — integrate uploader into Create Listing
/*
"use client";
import { useState } from "react";
import { UploadBox } from "@/components/UploadBox";

export default function CreateListingPage(){
  const [images, setImages] = useState<string[]>([]);
  return (
    <div className="max-w-2xl">
      <h1 className="mb-4 text-2xl font-semibold">Create a Listing</h1>
      <form className="space-y-4" onSubmit={(e)=>{e.preventDefault(); /* TODO: POST to
/api/listings with images */}}>
        <input name="title" placeholder="Title" className="w-full rounded-lg border p-2" />
        <textarea name="description" placeholder="Description" className="w-full rounded-lg
border p-2" rows={6} />
        <div className="grid grid-cols-2 gap-4">
          <select name="category" className="rounded-lg border p-2">
            <option value="CHEF">Chef</option>
            <option value="EVENT">Event</option>
            <option value="SERVICE">Service</option>
            <option value="RENTAL">Rental</option>
            <option value="PRODUCT">Product</option>
          </select>
          <input name="price" type="number" placeholder="Base Price (USD)"
className="rounded-lg border p-2" />
        </div>

        <UploadBox onDone={(urls)=> setImages(urls)} />
        {images.length>0 && (
          <div className="grid grid-cols-3 gap-2">
            {images.map((u)=> <img key={u} src={u} className="aspect-video w-full rounded-lg
object-cover" />)}
```

```
          </div>
        )}

        <input type="hidden" name="images" value={JSON.stringify(images)} />
        <button className="rounded-xl bg-black px-4 py-2 text-white">Save Listing</button>
      </form>
    </div>
  );
}
*/

// README/env additions — UploadThing
/*
UPLOADTHING_TOKEN=ut_...
NEXT_PUBLIC_UPLOADTHING_APP_ID=...
*/

// Notes
/*
- Use the returned URLs to store in `Listing.images` during POST /api/listings.
- Later: add image optimization (next/image) and aspect-ratio enforcement.
*/


// ==============================
// STEP 4 — Validation (Zod) + Rate Limiting (Upstash)
// ==============================

// Install (notes)
/*
 pnpm i zod @upstash/ratelimit @upstash/redis
*/

// lib/validators.ts — shared Zod schemas
/*
import { z } from "zod";

export const CreateListingSchema = z.object({
  title: z.string().min(3).max(120),
  description: z.string().min(30).max(8000),
  category: z.enum(["CHEF","EVENT","SERVICE","RENTAL","PRODUCT"]),
  priceCents: z.number().int().positive().max(10_000_000),
  unit: z.enum(["PER_EVENT","PER_HOUR","PER_DAY","PER_ITEM"]),
  images: z.array(z.string().url()).max(10).optional(),
  location: z.string().max(160).optional(),
});

export const CreateBookingSchema = z.object({
  listingId: z.string().min(1),
  dateStart: z.string().datetime(),
  dateEnd: z.string().datetime().optional(),
  qty: z.number().int().positive().max(500).default(1),
});
*/

// lib/rate.ts — Upstash rate limiter (per user/ip)
/*
import { Ratelimit } from "@upstash/ratelimit";
import { Redis } from "@upstash/redis";
import { NextRequest } from "next/server";

const redis = new Redis({
  url: process.env.UPSTASH_REDIS_REST_URL!,
  token: process.env.UPSTASH_REDIS_REST_TOKEN!,
});

export const limiter = new Ratelimit({
  redis,
  limiter: Ratelimit.slidingWindow(10, "10 s"), // 10 writes / 10s
  analytics: true,
  prefix: "tch:rl",
});

export async function limitOrThrow(req: NextRequest, keyHint?: string){
  const ip = req.ip ?? req.headers.get("x-forwarded-for")?.split(",")[0] ?? "anon";
  const key = `${keyHint || "api"}:${ip}`;
  const { success, limit, remaining, reset } = await limiter.limit(key);
  if(!success) {
    const err: any = new Error("Too many requests");
    err.status = 429;
    err.meta = { limit, remaining, reset };
    throw err;
  }
}
*/

// app/api/listings/route.ts — add Zod + rate limit
/*
```

```
import { prisma } from "@/lib/db";
import { NextRequest, NextResponse } from "next/server";
import { CreateListingSchema } from "@/lib/validators";
import { limitOrThrow } from "@/lib/rate";

export async function GET(req: NextRequest){
  const { searchParams } = new URL(req.url);
  const cat = searchParams.get("cat") as any;
  const data = await prisma.listing.findMany({
    where: { category: cat ?? undefined },
    orderBy: { createdAt: "desc" },
    take: 30,
  });
  return NextResponse.json({ data });
}

export async function POST(req: NextRequest){
  try {
    await limitOrThrow(req, "listings:create");
    const body = await req.json();
    const parsed = CreateListingSchema.safeParse(body);
    if(!parsed.success) return NextResponse.json({ error: parsed.error.format() }, { status: 400 });

    const ownerId = "SESSION_USER_ID"; // TODO: get from session
    const slug = parsed.data.title.toLowerCase().replace(/[^a-z0-9]+/g, "-").replace(/(^-|-$)/g, "");
    const listing = await prisma.listing.create({ data: { ...parsed.data, slug, ownerId } });
    return NextResponse.json({ listing });
  } catch (e:any) {
    const status = e?.status || 500;
    return NextResponse.json({ error: e?.message || "Server error" }, { status });
  }
}
*/

// app/api/bookings/route.ts — add Zod + rate limit
/*
import { prisma } from "@/lib/db";
import { NextRequest, NextResponse } from "next/server";
import { CreateBookingSchema } from "@/lib/validators";
import { limitOrThrow } from "@/lib/rate";

export async function POST(req: NextRequest){
  try {
    await limitOrThrow(req, "bookings:create");
    const body = await req.json();
    const parsed = CreateBookingSchema.safeParse(body);
    if(!parsed.success) return NextResponse.json({ error: parsed.error.format() }, { status: 400 });

    const listing = await prisma.listing.findUnique({ where: { id: parsed.data.listingId }, include: {
owner: true } });
    if(!listing || !listing.owner.stripeAccount) return NextResponse.json({ error: "Listing not
bookable" }, { status: 400 });

    const subtotal = listing.priceCents * (parsed.data.qty ?? 1);
    const fee = Math.round(subtotal * 0.12);
    const payout = subtotal - fee;

    const booking = await prisma.booking.create({ data: {
      listingId: listing.id,
      customerId: "SESSION_USER_ID",
      dateStart: new Date(parsed.data.dateStart),
      dateEnd: parsed.data.dateEnd ? new Date(parsed.data.dateEnd) : null,
      qty: parsed.data.qty ?? 1,
      subtotalCents: subtotal,
      feeCents: fee,
      payoutCents: payout,
      totalCents: subtotal,
    }});

    return NextResponse.json({ booking });
  } catch (e:any) {
    const status = e?.status || 500;
    return NextResponse.json({ error: e?.message || "Server error" }, { status });
  }
}
*/

// README/env additions — Upstash
/*
UPSTASH_REDIS_REST_URL=https://us1-merry-you-12345.upstash.io
UPSTASH_REDIS_REST_TOKEN=xxxx
*/


// ==============================
// STEP 5 — Messaging + Email Notifications (Resend)
// ==============================
// Install (notes)
/*
 pnpm i resend
*/

// lib/mail.ts — Resend client + helpers
/*
import { Resend } from "resend";
const resend = new Resend(process.env.RESEND_API_KEY!);

export async function sendBookingRequestEmail({ to, listingTitle, threadUrl }:{ to: string;
listingTitle: string; threadUrl: string; }){
  try {
    await resend.emails.send({
      from: process.env.EMAIL_FROM || "TakeachefHome <noreply@takeachefhome.com>",
      to,
      subject: `New inquiry for: ${listingTitle}`,
      html: `<p>You have a new inquiry for <b>${listingTitle}</b>.</p><p><a
href="${threadUrl}">Open conversation</a></p>`,
    });
  } catch(e){ console.error(e); }
}

export async function sendMessageEmail({ to, threadUrl }:{ to: string; threadUrl: string; }){
  try {
    await resend.emails.send({
      from: process.env.EMAIL_FROM || "TakeachefHome <noreply@takeachefhome.com>",
      to,
      subject: `New message on TakeachefHome`,
      html: `<p>You have a new message.</p><p><a href="${threadUrl}">Open
conversation</a></p>`,
    });
  } catch(e){ console.error(e); }
}
*/

// prisma/schema.prisma — (optional) add Thread model or reuse Message with threadId
/*
model Thread {
  id        String   @id @default(cuid())
  listingId String?
  listing   Listing? @relation(fields: [listingId], references: [id])
  aId       String   // user A
  bId       String   // user B
  messages  Message[]
  createdAt DateTime @default(now())
}

model Message {
  id         String   @id @default(cuid())
  threadId   String
  thread     Thread   @relation(fields: [threadId], references: [id])
  fromId     String
  toId       String
  body       String
  readAt     DateTime?
  createdAt  DateTime @default(now())
}
*/

// After adding models: pnpm prisma db push

// app/api/threads/route.ts — create or find thread (buyer ↔ vendor)
/*
import { prisma } from "@/lib/db";
import { NextRequest, NextResponse } from "next/server";

export async function POST(req: NextRequest){
  const { listingId, otherUserId } = await req.json();
  const me = "SESSION_USER_ID"; // TODO: from session
  if(!me) return NextResponse.json({ error: "Unauthorized" }, { status: 401 });

  // Find existing thread between me and other user for this listing
  let thread = await prisma.thread.findFirst({ where: {
    listingId, OR: [ { aId: me, bId: otherUserId }, { aId: otherUserId, bId: me } ]
  }});
  if(!thread){
    thread = await prisma.thread.create({ data: { listingId, aId: me, bId: otherUserId } });
  }
  return NextResponse.json({ thread });
}
*/

// app/api/messages/route.ts — send message + notify
/*
import { prisma } from "@/lib/db";
import { NextRequest, NextResponse } from "next/server";
import { sendMessageEmail } from "@/lib/mail";
```

```
export async function POST(req: NextRequest){
  const { threadId, body } = await req.json();
  const me = "SESSION_USER_ID"; // TODO: from session
  if(!me) return NextResponse.json({ error: "Unauthorized" }, { status: 401 });

  const thread = await prisma.thread.findUnique({ where: { id: threadId }, include: { listing: true }
});
  if(!thread) return NextResponse.json({ error: "Thread not found" }, { status: 404 });

  const toId = thread.aId === me ? thread.bId : thread.aId;
  const msg = await prisma.message.create({ data: { threadId, fromId: me, toId, body } });

  // Notify recipient
  const toUser = await prisma.user.findUnique({ where: { id: toId } });
  if (toUser?.email) await sendMessageEmail({ to: toUser.email, threadUrl:
`${process.env.SITE_URL}/messages/${threadId}` });

  return NextResponse.json({ message: msg });
}
*/

// app/messages/[id]/page.tsx — simple thread UI
/*
import { prisma } from "@/lib/db";

export default async function ThreadPage({ params }: { params: { id: string } }){
  const thread = await prisma.thread.findUnique({ where: { id: params.id }, include: { messages: {
orderBy: { createdAt: "asc" } } } });
  if(!thread) return <div>Thread not found</div>;
  return (
    <div className="mx-auto max-w-2xl space-y-4">
      <h1 className="text-2xl font-semibold">Conversation</h1>
      <div className="rounded-xl border p-4">
        {thread.messages.map(m => (
          <div key={m.id} className="mb-3">
            <div className="text-xs text-gray-500">{new Date(m.createdAt).toLocaleString()}</div>
            <div>{m.body}</div>
          </div>
        ))}
      </div>
      <form action="/api/messages" method="post" className="flex gap-2">
        <input type="hidden" name="threadId" value={thread.id} />
        <input name="body" placeholder="Write a message" className="flex-1 rounded-lg border
p-2" />
        <button className="rounded-lg bg-black px-4 py-2 text-white">Send</button>
      </form>
    </div>
  );
}
*/

// Booking request notification example (called when a booking is created)
/*
import { sendBookingRequestEmail } from "@/lib/mail";
// after prisma.booking.create(...)
const owner = await prisma.user.findUnique({ where: { id: listing.ownerId } });
if(owner?.email){
  await sendBookingRequestEmail({ to: owner.email, listingTitle: listing.title, threadUrl:
`${process.env.SITE_URL}/messages/${/* threadId */ ""}` });
}
*/

// README/env additions — Resend
/*
RESEND_API_KEY=re_...
EMAIL_FROM=TakeachefHome <noreply@takeachefhome.com>
*/



// ============================
// STEP 6 — Reviews with Post-Purchase Guard
// ============================

// Assumptions: `Review` model already exists and references Listing + User.
// Goal: Only users with a COMPLETED booking on a listing can leave a review once per
booking.

// prisma/schema.prisma — add unique constraint to prevent dupes per (author, listing)
optionally per booking
/*
model Review {
  id         String   @id @default(cuid())
  listingId  String
  listing    Listing  @relation(fields: [listingId], references: [id])
  authorId   String
  author     User     @relation(fields: [authorId], references: [id])
  rating     Int
  comment    String?
  createdAt  DateTime @default(now())
  bookingId  String?  // optional, link to booking

  @@index([listingId])
  @@unique([authorId, listingId]) // one review per user per listing (simple rule)
}
*/

// lib/reviews.ts — helpers to compute averages
/*
import { prisma } from "@/lib/db";

export async function recomputeListingRating(listingId: string){
  const agg = await prisma.review.aggregate({ where: { listingId }, _avg: { rating: true }, _count: {
rating: true } });
  await prisma.listing.update({ where: { id: listingId }, data: { ratingAvg: agg._avg.rating ?? 0,
ratingCount: agg._count.rating } });
}
*/

// app/api/reviews/route.ts — POST create with guard
/*
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/db";
import { z } from "zod";
import { recomputeListingRating } from "@/lib/reviews";

const CreateReview = z.object({ listingId: z.string(), rating: z.number().int().min(1).max(5),
comment: z.string().max(2000).optional() });

export async function POST(req: NextRequest){
  const me = "SESSION_USER_ID"; // TODO: from session
  if(!me) return NextResponse.json({ error: "Unauthorized" }, { status: 401 });

  const body = await req.json();
  const parsed = CreateReview.safeParse(body);
  if(!parsed.success) return NextResponse.json({ error: parsed.error.format() }, { status: 400 });

  // Guard: user must have at least one COMPLETED booking for this listing
  const completed = await prisma.booking.findFirst({ where: { listingId: parsed.data.listingId,
customerId: me, status: "COMPLETED" } });
```

```
  if(!completed) return NextResponse.json({ error: "You can review only after a completed
booking." }, { status: 403 });

  // Upsert-like: rely on @@unique([authorId, listingId]) to prevent dupes
  const review = await prisma.review.create({ data: { listingId: parsed.data.listingId, authorId: me,
rating: parsed.data.rating, comment: parsed.data.comment, bookingId: completed.id } });
  await recomputeListingRating(parsed.data.listingId);

  return NextResponse.json({ review });
}
*/

// app/listing/[id]/page.tsx — show reviews + create form (if eligible)
/*
import { prisma } from "@/lib/db";

export default async function ListingPage({ params }: { params: { id: string } }){
  const listing = await prisma.listing.findUnique({ where: { id: params.id }, include: { reviews: {
include: { author: true }, orderBy: { createdAt: "desc" } } } });
  if(!listing) return <div>Listing not found</div>;

  return (
    <div className="grid gap-8 md:grid-cols-5">
      <div className="md:col-span-3">
        {/* media + description ... */}
        <h2 className="mt-8 text-xl font-semibold">Reviews ({listing.ratingCount})</h2>
        <div className="space-y-4">
          {listing.reviews.map(r => (
            <div key={r.id} className="rounded-xl border p-4">
              <div className="flex items-center justify-between">
                <div className="font-medium">{r.author.name || "Guest"}</div>
                <div className="text-sm">{"★".repeat(r.rating)}{"☆".repeat(5-r.rating)}</div>
              </div>
              {r.comment && <p className="mt-1 text-gray-700">{r.comment}</p>}
              <div className="mt-1 text-xs text-gray-500">{new
Date(r.createdAt).toLocaleDateString()}</div>
            </div>
          ))}
        </div>
      </div>
      <aside className="md:col-span-2">
        {/* booking card ... */}
        <div className="mt-6 rounded-2xl border p-4">
          <h3 className="mb-2 font-medium">Leave a review</h3>
          <form action="/api/reviews" method="post" className="space-y-2">
            <input type="hidden" name="listingId" value={listing.id} />
            <select name="rating" className="w-full rounded-lg border p-2">
              {[1,2,3,4,5].map(n => <option key={n} value={n}>{n} star{n>1?"s":""}</option>)}
            </select>
            <textarea name="comment" placeholder="Share details of your experience"
className="w-full rounded-lg border p-2" rows={3} />
            <button className="w-full rounded-lg bg-black p-2 text-white">Submit</button>
          </form>
          <p className="mt-2 text-xs text-gray-500">Reviews require a completed booking.</p>
        </div>
      </aside>
    </div>
  );
}
*/

// Booking completion hook — mark COMPLETED then invite review
/*
// After service date, a cron/queue can mark eligible bookings as COMPLETED.
// Then trigger a Resend email inviting the review with direct link.
*/


// ============================
// STEP 7 — Admin Dashboard (approve/feature/suspend/refund)
// ============================

// prisma/schema.prisma — add moderation fields
/*
model User {
  id            String   @id @default(cuid())
  name          String?
  email         String?  @unique
  image         String?
  role          Role     @default(CUSTOMER)
  stripeAccount String?
  suspended     Boolean  @default(false)
  // ... rest unchanged
}

model Listing {
  id           String   @id @default(cuid())
  title        String
  slug         String   @unique
  description  String
  images       String[]
  category     Category
  priceCents   Int
  unit         Unit
  location     String?
  ownerId      String
  owner        User     @relation(fields: [ownerId], references: [id])
  ratingAvg    Float?   @default(0)
  ratingCount  Int      @default(0)
  availability Json?
  approved     Boolean  @default(false)
  featuredAt   DateTime?
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt
  bookings     Booking[]
  reviews      Review[]
}
*/

// app/admin/page.tsx — Admin dashboard UI
/*
import { prisma } from "@/lib/db";

export default async function AdminPage(){
  const [pending, recentUsers] = await Promise.all([
    prisma.listing.findMany({ where: { approved: false }, include: { owner: true }, orderBy: {
createdAt: "desc" } }),
    prisma.user.findMany({ orderBy: { createdAt: "desc" }, take: 20 })
  ]);

  return (
    <div className="space-y-8">
      <h1 className="text-2xl font-semibold">Admin</h1>

      <section>
        <h2 className="mb-3 text-lg font-medium">Pending Listings</h2>
        <div className="overflow-x-auto rounded-xl border">
          <table className="w-full text-sm">
            <thead className="bg-gray-50"><tr><th className="p-2 text-left">Title</th><th
className="p-2">Owner</th><th className="p-2">Category</th><th
className="p-2">Created</th><th className="p-2">Actions</th></tr></thead>
            <tbody>
          {pending.map((l)=> (
            <tr key={l.id} className="border-t">
              <td className="p-2">{l.title}</td>
              <td className="p-2">{l.owner?.email}</td>
              <td className="p-2 text-center">{l.category}</td>
              <td className="p-2 text-center">{new Date(l.createdAt).toLocaleDateString()}</td>
              <td className="p-2 text-center">
                <form action="/api/admin/listings/approve" method="post" className="inline">
                  <input type="hidden" name="id" value={l.id} />
                  <button className="rounded bg-green-600 px-3 py-1
text-white">Approve</button>
                </form>
                <form action="/api/admin/listings/feature" method="post" className="inline ml-2">
                  <input type="hidden" name="id" value={l.id} />
                  <button className="rounded bg-amber-600 px-3 py-1
text-white">Feature</button>
                </form>
              </td>
            </tr>
          ))}
          </tbody>
        </table>
      </div>
    </section>

    <section>
      <h2 className="mb-3 text-lg font-medium">Users</h2>
      <div className="overflow-x-auto rounded-xl border">
        <table className="w-full text-sm">
          <thead className="bg-gray-50"><tr><th className="p-2 text-left">Email</th><th
className="p-2">Role</th><th className="p-2">Stripe</th><th
className="p-2">Status</th><th className="p-2">Actions</th></tr></thead>
          <tbody>
          {recentUsers.map((u)=> (
            <tr key={u.id} className="border-t">
              <td className="p-2">{u.email}</td>
              <td className="p-2 text-center">{u.role}</td>
              <td className="p-2 text-center">{u.stripeAccount ? 'Connected' : '—'}</td>
              <td className="p-2 text-center">{u.suspended ? 'Suspended' : 'Active'}</td>
              <td className="p-2 text-center">
                <form action="/api/admin/users/toggle-suspend" method="post"
className="inline">
                  <input type="hidden" name="id" value={u.id} />
```

```
                    <button className="rounded bg-black px-3 py-1 text-white">{u.suspended ?
'Unsuspend' : 'Suspend'}</button>
                  </form>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </section>

    <section>
      <h2 className="mb-3 text-lg font-medium">Refunds</h2>
      <p className="mb-2 text-sm text-gray-600">Issue refunds for specific bookings (full
refunds only in MVP).</p>
      <form action="/api/admin/refund" method="post" className="flex max-w-md gap-2">
        <input name="bookingId" placeholder="Booking ID" className="flex-1 rounded-lg border
p-2" />
        <button className="rounded bg-red-600 px-4 py-2 text-white">Refund</button>
      </form>
    </section>
  </div>
 );
}
*/

// API routes — admin actions
/*
// app/api/admin/listings/approve/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/db";
export async function POST(req: NextRequest){
 const form = await req.formData();
 const id = String(form.get("id"));
 await prisma.listing.update({ where: { id }, data: { approved: true } });
 return NextResponse.redirect(new URL("/admin", req.url));
}

// app/api/admin/listings/feature/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/db";
export async function POST(req: NextRequest){
 const form = await req.formData();
 const id = String(form.get("id"));
```

```
  await prisma.listing.update({ where: { id }, data: { featuredAt: new Date(), approved: true } });
  return NextResponse.redirect(new URL("/admin", req.url));
}

// app/api/admin/users/toggle-suspend/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/db";
export async function POST(req: NextRequest){
  const form = await req.formData();
  const id = String(form.get("id"));
  const u = await prisma.user.update({ where: { id }, data: { suspended: { set: undefined } } });
  // toggle because set undefined won't work; retrieve and flip
  await prisma.user.update({ where: { id }, data: { suspended: !u.suspended } });
  return NextResponse.redirect(new URL("/admin", req.url));
}

// app/api/admin/refund/route.ts
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/db";
import { stripe } from "@/lib/stripe";
export async function POST(req: NextRequest){
  const form = await req.formData();
  const bookingId = String(form.get("bookingId"));
  const booking = await prisma.booking.findUnique({ where: { id: bookingId } });
  if(!booking?.paymentIntentId) return NextResponse.json({ error: "No payment found" }, { status:
400 });
  await stripe.refunds.create({ payment_intent: booking.paymentIntentId });
  await prisma.booking.update({ where: { id: bookingId }, data: { status: "CANCELED" } });
  return NextResponse.redirect(new URL("/admin", req.url));
}
*/

// Middleware note: `/admin/*` already protected by ADMIN in middleware.ts

// README — Admin setup
/*
- Assign ADMIN role manually (Prisma Studio) to your user for initial access.
- Approving a listing flips `approved: true` and optional `featuredAt` timestamp.
- Suspended users should be prevented from creating listings/booking (add checks on APIs).
*/


// ============================
// STEP 8 — SEO Pack (OpenGraph, JSON-LD, sitemap, robots)
// ==============================

// app/(site)/layout.tsx — default metadata
/*
export const metadata = {
  metadataBase: new URL(process.env.SITE_URL || "http://localhost:3000"),
  title: {
    default: "TakeachefHome — Book Chefs, Events, Services, Rentals",
    template: "%s • TakeachefHome",
  },
  description: "Bring the chef, the vibe, and the experience — home. A soulful marketplace for
chefs, events, services, rentals, and more.",
  openGraph: {
    type: "website",
    title: "TakeachefHome",
    siteName: "TakeachefHome",
    url: "/",
  },
  twitter: { card: "summary_large_image", site: "@takeachefhome" },
};
*/

// app/listing/[id]/page.tsx — per-listing metadata + JSON-LD
/*
import { prisma } from "@/lib/db";
import type { Metadata } from "next";

export async function generateMetadata({ params }: { params: { id: string } }):
Promise<Metadata> {
  const listing = await prisma.listing.findUnique({ where: { id: params.id } });
  if(!listing) return { title: "Listing not found" };
  const title = `${listing.title}`;
  const desc = `${listing.description?.slice(0, 160)}`;
  const image = listing.images?.[0] || "/og-default.jpg";
  return {
    title,
    description: desc,
    openGraph: {
      title,
      description: desc,
      images: [{ url: image }],
    },
  };
}
```

```
function ListingJsonLd({ listing }:{ listing: any }){
  const data = {
    "@context": "https://schema.org",
    "@type": listing.category === "PRODUCT" ? "Product" : "Service",
    name: listing.title,
    description: listing.description,
    image: listing.images?.length ? listing.images : undefined,
    areaServed: listing.location || undefined,
    offers: {
      "@type": "Offer",
      price: (listing.priceCents/100).toFixed(2),
      priceCurrency: "USD",
      availability: listing.approved ? "https://schema.org/InStock" : "https://schema.org/PreOrder",
    },
    aggregateRating: listing.ratingCount ? {
      "@type": "AggregateRating",
      ratingValue: listing.ratingAvg?.toFixed(1),
      reviewCount: listing.ratingCount,
    } : undefined,
  };
  return <script type="application/ld+json" dangerouslySetInnerHTML={{ __html:
JSON.stringify(data) }} />
}

export default async function ListingPage({ params }: { params: { id: string } }){
  const listing = await prisma.listing.findUnique({ where: { id: params.id }, include: { reviews: {
include: { author: true } } } });
  if(!listing) return <div>Listing not found</div>;
  return (
    <div className="grid gap-8 md:grid-cols-5">
      <ListingJsonLd listing={listing} />
      {/* rest of UI */}
    </div>
  );
}
*/

// app/opengraph-image.tsx — dynamic OG image (optional simple)
/*
import { ImageResponse } from "next/og";
export const size = { width: 1200, height: 630 };
export const contentType = "image/png";
export default async function OG(){
  return new ImageResponse(
    (
      <div style={{ display: "flex", width: "100%", height: "100%", background: "#0a0a0a", color:
"#fff", alignItems: "center", justifyContent: "center", fontSize: 64, fontWeight: 700 }}>
        TakeachefHome
      </div>
    ),
    { ...size }
  );
}
*/

// app/sitemap.ts — sitemap
/*
import { prisma } from "@/lib/db";
export default async function sitemap(){
  const listings = await prisma.listing.findMany({ where: { approved: true }, select: { id: true,
updatedAt: true } });
  const base = process.env.SITE_URL || "http://localhost:3000";
  return [
    { url: `${base}/`, lastModified: new Date() },
    { url: `${base}/browse`, lastModified: new Date() },
    ...listings.map(l => ({ url: `${base}/listing/${l.id}`, lastModified: l.updatedAt }))
  ];
}
*/

// app/robots.ts — robots
/*
export default function robots(){
  const base = process.env.SITE_URL || "http://localhost:3000";
  return {
    rules: { userAgent: "*", allow: "/", disallow: ["/admin", "/dashboard", "/messages"] },
    sitemap: `${base}/sitemap.xml`,
  };
}
*/

// README — SEO notes
/*
- Ensure `SITE_URL` is set in prod for correct absolute URLs.
- Add `twitter:site` handle when ready.
- Consider per-category JSON-LD types (Event, Rental, Service, Product) as we specialize
listings.
```

```
*/


// ============================
// STEP 9 — Analytics (Vercel) + Error Tracking (Sentry)
// ============================

// Install (notes)
/*
 pnpm i @vercel/analytics @sentry/nextjs
*/

// app/(site)/layout.tsx — add Vercel Analytics component
/*
import { Analytics } from "@vercel/analytics/react";

export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html lang="en">
      <body className="min-h-screen bg-white text-gray-900">
        <header className="sticky top-0 z-50 border-b bg-white/80 backdrop-blur">
          <div className="mx-auto flex max-w-6xl items-center justify-between p-4">
            <a href="/" className="text-xl font-bold">TakeachefHome</a>
            <nav className="flex items-center gap-4">
              <a href="/browse" className="hover:underline">Browse</a>
              <a href="/create" className="hover:underline">Create Listing</a>
              <a href="/dashboard" className="hover:underline">Dashboard</a>
              <a href="/api/auth/signin" className="rounded-lg border px-3 py-1">Sign in</a>
            </nav>
          </div>
        </header>
        <main className="mx-auto max-w-6xl p-4">{children}</main>
        <footer className="mt-12 border-t py-8 text-center text-sm">© {new Date().getFullYear()}
TakeachefHome.com</footer>
        <Analytics />
      </body>
    </html>
  );
}
*/

// Sentry minimal setup — sentry.client.config.ts
/*
import * as Sentry from "@sentry/nextjs";
```

```
Sentry.init({
  dsn: process.env.SENTRY_DSN,
  tracesSampleRate: 0.2, // RUM
  replaysSessionSampleRate: 0.1,
  replaysOnErrorSampleRate: 1.0,
});
*/

// Sentry minimal setup — sentry.server.config.ts
/*
import * as Sentry from "@sentry/nextjs";

Sentry.init({
  dsn: process.env.SENTRY_DSN,
  tracesSampleRate: 0.2,
});
*/

// next.config.mjs — wrap with Sentry (optional for source maps)
/*
import { withSentryConfig } from "@sentry/nextjs";

/ ** @type {import('next').NextConfig} * /
const nextConfig = {
  reactStrictMode: true,
  experimental: { serverActions: { bodySizeLimit: "2mb" } },
};

export default withSentryConfig(nextConfig, {
  silent: true,
  org: process.env.SENTRY_ORG,
  project: process.env.SENTRY_PROJECT,
});
*/

// app/global-error.tsx — global error boundary (Next.js App Router)
/*
'use client';
export default function GlobalError({ error, reset }: { error: Error; reset: () => void }) {
  return (
    <html>
      <body className="grid min-h-screen place-items-center p-8 text-center">
        <div>
```

```
          <h2 className="text-2xl font-semibold">Something went wrong</h2>
          <p className="mt-2 text-gray-600">Our team has been notified. Try again or go back
home.</p>
          <div className="mt-4 flex justify-center gap-3">
            <button onClick={() => reset()} className="rounded bg-black px-4 py-2 text-white">Try
again</button>
            <a href="/" className="rounded border px-4 py-2">Home</a>
          </div>
        </div>
      </body>
    </html>
  );
}
*/

// lib/logger.ts — helper to capture exceptions to Sentry
/*
export async function capture(err: any, context?: Record<string, any>){
  try {
    const Sentry = await import("@sentry/nextjs");
    Sentry.captureException(err, { extra: context });
  } catch {}
}
*/

// Example usage in API route
/*
import { capture } from "@/lib/logger";
export async function POST(req: Request){
  try {
    // ...
  } catch(e){
    await capture(e, { route: "/api/whatever" });
    return new Response(JSON.stringify({ error: "Server error" }), { status: 500 });
  }
}
*/

// README/env additions — Analytics + Sentry
/*
VERCEL_ANALYTICS_ID (Vercel auto)
SENTRY_DSN=https://xxxxxxx.ingest.sentry.io/xxxxx
SENTRY_AUTH_TOKEN=... (for uploading source maps in CI)
SENTRY_ORG=takeachefhome
```

```
SENTRY_PROJECT=web
*/


// ==============================
// STEP 10 — Availability Calendar (blocked dates + timezone)
// ==============================

// Install (notes)
/*
 pnpm i date-fns
*/

// prisma/schema.prisma — add AvailabilityBlock + optional timeZone on Listing
/*
model Listing {
  id           String   @id @default(cuid())
  title        String
  slug         String   @unique
  description  String
  images       String[]
  category     Category
  priceCents   Int
  unit         Unit
  location     String?
  timeZone     String?   // e.g., "America/New_York"
  ownerId      String
  owner        User     @relation(fields: [ownerId], references: [id])
  ratingAvg    Float?   @default(0)
  ratingCount  Int      @default(0)
  availability Json?
  approved     Boolean  @default(false)
  featuredAt   DateTime?
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt
  bookings     Booking[]
  reviews      Review[]
  blocks       AvailabilityBlock[]
}

model AvailabilityBlock {
  id         String   @id @default(cuid())
  listingId  String
  listing    Listing  @relation(fields: [listingId], references: [id])
  start      DateTime
  end        DateTime
  note       String?
  createdAt  DateTime @default(now())

  @@index([listingId, start, end])
}
*/

// lib/availability.ts — helpers for overlap checks
/*
import { prisma } from "@/lib/db";

function overlaps(aStart: Date, aEnd: Date, bStart: Date, bEnd: Date){
  return aStart < bEnd && bStart < aEnd;
}

export async function isListingAvailable(listingId: string, start: Date, end?: Date){
  const [blocks, bookings] = await Promise.all([
    prisma.availabilityBlock.findMany({ where: { listingId } }),
    prisma.booking.findMany({ where: { listingId, status: { in: ["PENDING","CONFIRMED"] } } })
  ]);
  const e = end ?? new Date(start.getTime() + 3*60*60*1000); // default 3h
  for(const b of blocks){ if(overlaps(b.start, b.end, start, e)) return false; }
  for(const k of bookings){ const ks = k.dateStart; const ke = k.dateEnd ?? new
Date(ks.getTime() + 3*60*60*1000); if(overlaps(ks, ke, start, e)) return false; }
  return true;
}
*/

// app/api/availability/route.ts — vendor adds/removes blocks
/*
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/db";
import { z } from "zod";

const UpsertBlock = z.object({ listingId: z.string(), start: z.string().datetime(), end:
z.string().datetime(), note: z.string().max(120).optional() });
const DeleteBlock = z.object({ id: z.string() });

export async function POST(req: NextRequest){
  const me = "SESSION_USER_ID"; // TODO from session
  const body = await req.json();
  const parsed = UpsertBlock.safeParse(body);
  if(!me) return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
  if(!parsed.success) return NextResponse.json({ error: parsed.error.format() }, { status: 400 });

  // ownership check
  const listing = await prisma.listing.findUnique({ where: { id: parsed.data.listingId } });
  if(!listing || listing.ownerId !== me) return NextResponse.json({ error: "Forbidden" }, { status:
403 });

  const block = await prisma.availabilityBlock.create({ data: { listingId: listing.id, start: new
Date(parsed.data.start), end: new Date(parsed.data.end), note: parsed.data.note } });
  return NextResponse.json({ block });
}

export async function DELETE(req: NextRequest){
  const me = "SESSION_USER_ID"; // TODO
  const body = await req.json();
  const parsed = DeleteBlock.safeParse(body);
  if(!me) return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
  if(!parsed.success) return NextResponse.json({ error: parsed.error.format() }, { status: 400 });

  const block = await prisma.availabilityBlock.findUnique({ where: { id: parsed.data.id }, include: {
listing: true } });
  if(!block || block.listing.ownerId !== me) return NextResponse.json({ error: "Forbidden" }, {
status: 403 });
  await prisma.availabilityBlock.delete({ where: { id: block.id } });
  return NextResponse.json({ ok: true });
}
*/

// app/dashboard/availability/page.tsx — simple calendar UI (month grid)
/*
"use client";
import { useEffect, useState } from "react";
import { addDays, startOfMonth, endOfMonth, startOfWeek, endOfWeek, isSameMonth,
isSameDay, format } from "date-fns";

function buildCalendar(cursor: Date){
  const start = startOfWeek(startOfMonth(cursor));
  const end = endOfWeek(endOfMonth(cursor));
  const days: Date[] = [];
  let d = start;
  while(d <= end){ days.push(d); d = addDays(d, 1); }
  return days;
}
```

```
export default function AvailabilityPage(){
  const [cursor, setCursor] = useState(new Date());
  const [selection, setSelection] = useState<{ start?: string; end?: string }>({});

  const days = buildCalendar(cursor);

  async function addBlock(){
    if(!selection.start || !selection.end) return alert("Select start and end");
    const res = await fetch("/api/availability", { method: "POST", headers: { "Content-Type":
"application/json" }, body: JSON.stringify({ listingId: "SELECTED_LISTING_ID", start:
selection.start, end: selection.end }) });
    if(!res.ok) return alert("Failed to save");
    alert("Blocked!");
  }

  return (
    <div className="space-y-4">
      <h1 className="text-2xl font-semibold">Availability</h1>
      <div className="flex items-center gap-3">
        <button className="rounded border px-3 py-1" onClick={()=> setCursor(addDays(cursor,
-30))}>Prev</button>
        <div className="font-medium">{format(cursor, "MMMM yyyy")}</div>
        <button className="rounded border px-3 py-1" onClick={()=> setCursor(addDays(cursor,
30))}>Next</button>
      </div>
      <div className="grid grid-cols-7 gap-2">
        {days.map((d)=> (
          <button key={d.toISOString()} onClick={()=> {
            const iso = d.toISOString();
            setSelection((s)=> s.start && !s.end ? { ...s, end: iso } : { start: iso, end: undefined });
          }} className={`aspect-square rounded-lg border p-2 text-sm ${isSameMonth(d, cursor)?
'bg-white' : 'bg-gray-50'} ${selection.start && isSameDay(new Date(selection.start), d) ? 'ring-2
ring-black' : ''}`}>
            {format(d, "d")}
          </button>
        ))}
      </div>
      <div className="flex items-center gap-2">
        <button className="rounded bg-black px-4 py-2 text-white" onClick={addBlock}>Block
Selected</button>
        <div className="text-sm text-gray-600">Click start day, then end day.</div>
      </div>
    </div>
  );
}
*/

// app/api/bookings/route.ts — enforce availability check before creating booking
/*
import { isListingAvailable } from "@/lib/availability";
// ... after validating payload and loading listing
const start = new Date(parsed.data.dateStart);
const end = parsed.data.dateEnd ? new Date(parsed.data.dateEnd) : undefined;
const ok = await isListingAvailable(listing.id, start, end);
if(!ok) return NextResponse.json({ error: "Selected date/time is unavailable." }, { status: 409 });
*/

// README — Availability notes
/*
- Simple blocked-date model for MVP. Later: per-hour windows, Google Calendar/ICS sync,
multi-day rentals.
- Store `Listing.timeZone` to interpret local times; MVP uses ISO UTC and treats day-level
blocks.
*/


// ============================
// STEP 11 — Legal Pages (Terms, Privacy, Refunds, Cookies) + Cookie Banner
// ============================

// app/(site)/terms/page.tsx
/*
export default function Terms(){
  return (
    <div className="prose max-w-none">
      <h1>Terms of Service</h1>
      <p>Welcome to TakeachefHome.com. By accessing or using our platform, you agree to
these Terms.</p>
      <h2>1. Marketplace Role</h2>
      <p>We connect customers with independent vendors (chefs, event providers, rentals). We
do not control vendor operations and are not a party to vendor-customer contracts.</p>
      <h2>2. Accounts & Eligibility</h2>
      <p>You must provide accurate information and maintain security of your account. Vendors
must comply with applicable health, safety, and licensing laws.</p>
      <h2>3. Payments & Fees</h2>
      <p>Payments are processed via Stripe. Platform fees and Stripe fees are disclosed at
checkout. Refunds are governed by the Refund Policy.</p>
```

```
      <h2>4. Cancellations</h2>
      <p>Vendor-specific policies apply. The platform reserves the right to suspend accounts for
policy violations.</p>
      <h2>5. Content & Conduct</h2>
      <p>No illegal, hateful, or explicit content. No harassment or fraud. We reserve the right to
remove content or listings.</p>
      <h2>6. Disclaimers & Liability</h2>
      <p>Service is provided “as is”. To the maximum extent permitted by law, we disclaim
warranties and limit liability.</p>
      <h2>7. Governing Law</h2>
      <p>These Terms are governed by the laws of the United States and the state indicated in
our corporate registration.</p>
      <h2>8. Contact</h2>
      <p>legal@takeachefhome.com</p>
    </div>
  );
}
*/

// app/(site)/privacy/page.tsx
/*
export default function Privacy(){
  return (
    <div className="prose max-w-none">
      <h1>Privacy Policy</h1>
      <p>We collect account information, listing data, messages, and payments metadata to
operate the marketplace.</p>
      <h2>Data We Collect</h2>
      <ul>
        <li>Account: name, email, role, profile image</li>
        <li>Transactions: booking details, Stripe payment status (no full card numbers)</li>
        <li>Usage: logs, analytics, device info</li>
      </ul>
      <h2>How We Use Data</h2>
      <p>To run the service, prevent fraud, provide support, and improve features. Emails may be
sent for critical notifications.</p>
      <h2>Sharing</h2>
      <p>With processors (Stripe, Resend, UploadThing, Sentry, Vercel). We do not sell personal
data.</p>
      <h2>Data Rights</h2>
      <p>You may request access, correction, or deletion by contacting
privacy@takeachefhome.com.</p>
      <h2>Security</h2>
      <p>We apply best practices and minimize PII. No method is 100% secure.</p>
    </div>
  );
}
*/

// app/(site)/refunds/page.tsx
/*
export default function Refunds(){
  return (
    <div className="prose max-w-none">
      <h1>Refund Policy</h1>
      <p>Refund eligibility depends on vendor policy and booking status. Platform-issued refunds
are limited to payment errors and fraud investigations.</p>
      <h2>Standard Window</h2>
      <p>Customer-initiated cancellations ≥7 days before service date: vendor may offer partial
refund minus fees. Within 7 days: vendor discretion.</p>
      <h2>No-Shows & Service Issues</h2>
      <p>Report within 24 hours of the event with documentation. We may hold payouts during
investigation.</p>
      <h2>How to Request</h2>
      <p>Email support@takeachefhome.com with booking ID.</p>
    </div>
  );
}
*/

// app/(site)/cookies/page.tsx
/*
export default function Cookies(){
  return (
    <div className="prose max-w-none">
      <h1>Cookie Policy</h1>
      <p>We use cookies for authentication, analytics, and essential site features. You can control
cookies in your browser settings.</p>
      <h2>Types</h2>
      <ul>
        <li>Essential (auth/session)</li>
        <li>Analytics (Vercel Analytics)</li>
        <li>Performance & error tracking (Sentry)</li>
      </ul>
    </div>
  );
}
*/
```

```
// components/CookieBanner.tsx — simple consent banner (informational)
/*
"use client";
import { useEffect, useState } from "react";

export function CookieBanner(){
  const [show, setShow] = useState(false);
  useEffect(()=>{ if(!localStorage.getItem("tch_cookie_ack")) setShow(true); },[]);
  if(!show) return null;
  return (
    <div className="fixed bottom-4 left-1/2 z-50 w-[92vw] max-w-3xl -translate-x-1/2 rounded-xl
border bg-white p-4 shadow">
      <p className="text-sm text-gray-700">We use cookies for essential features, analytics,
and to improve your experience. See our <a href="/cookies" className="underline">Cookie
Policy</a>.</p>
      <div className="mt-2 flex justify-end gap-2">
        <button className="rounded border px-3 py-1" onClick={()=> {
localStorage.setItem("tch_cookie_ack","dismissed"); setShow(false); }}>Dismiss</button>
        <a href="/privacy" className="rounded bg-black px-3 py-1 text-white">Privacy</a>
      </div>
    </div>
  );
}
*/

// app/(site)/layout.tsx — include CookieBanner
/*
import { CookieBanner } from "@/components/CookieBanner";
export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html lang="en">
      <body className="min-h-screen bg-white text-gray-900">
        {/* header ... */}
        <main className="mx-auto max-w-6xl p-4">{children}</main>
        <footer className="mt-12 border-t py-8 text-center text-sm">© {new Date().getFullYear()}
TakeachefHome.com</footer>
        <CookieBanner />
      </body>
    </html>
  );
}
*/
```

```
// README — Legal
/*
- Replace placeholder legal text with counsel-reviewed language before launch.
- Add contact emails: legal@, privacy@, support@ (configure in your mail provider).
*/


// ==============================
// STEP 12 — Vercel DEMO MODE (investor-ready)
// ==============================
// Goal: one-click deploy to Vercel with safe test keys, fake emails, optional fake checkout,
// seeded data, and an Investor Tour page.

// .env keys (Demo)
/*
DEMO_MODE=true
SITE_URL=<<Vercel URL after first deploy>>
NEXTAUTH_URL=<<same as SITE_URL>>
NEXTAUTH_SECRET=demo_demo_demo
DATABASE_URL=postgresql://USER:PASS@HOST:PORT/DB?schema=public
GITHUB_ID=xxxx
GITHUB_SECRET=xxxx
# Stripe (test)
STRIPE_SECRET_KEY=sk_test_xxx
STRIPE_WEBHOOK_SECRET=whsec_test
STRIPE_CONNECT_CLIENT_ID=ca_test_xxx
# UploadThing (optional; demo can run without uploads)
UPLOADTHING_TOKEN=
NEXT_PUBLIC_UPLOADTHING_APP_ID=
# Resend (omit in demo)
RESEND_API_KEY=
UPSTASH_REDIS_REST_URL=
UPSTASH_REDIS_REST_TOKEN=
SENTRY_DSN=
*/

// lib/env.ts — tiny helper
/*
export const DEMO = process.env.DEMO_MODE === "true";
export const SITE = process.env.SITE_URL || "http://localhost:3000";
*/

// lib/mail.ts — stub emails in demo
/*
```

```
import { DEMO } from "@/lib/env";
export async function sendBookingRequestEmail({ to, listingTitle, threadUrl }:{ to: string;
listingTitle: string; threadUrl: string; }){
  if (DEMO) { console.log("[DEMO EMAIL] booking ->", { to, listingTitle, threadUrl }); return; }
  const { Resend } = await import("resend");
  const resend = new Resend(process.env.RESEND_API_KEY!);
  await resend.emails.send({ from: process.env.EMAIL_FROM || "TakeachefHome
<noreply@takeachefhome.com>", to, subject: `New inquiry for: ${listingTitle}` , html: `<a
href="${threadUrl}">Open conversation</a>` });
}
export async function sendMessageEmail({ to, threadUrl }:{ to: string; threadUrl: string; }){
  if (DEMO) { console.log("[DEMO EMAIL] message ->", { to, threadUrl }); return; }
  const { Resend } = await import("resend");
  const resend = new Resend(process.env.RESEND_API_KEY!);
  await resend.emails.send({ from: process.env.EMAIL_FROM || "TakeachefHome
<noreply@takeachefhome.com>", to, subject: `New message on TakeachefHome`, html: `<a
href="${threadUrl}">Open conversation</a>` });
}
*/

// lib/stripe.ts — fake checkout when DEMO=true
/*
import { DEMO, SITE } from "@/lib/env";

export async function createCheckoutSession({ bookingId, amount, applicationFeeCents,
connectedAccountId }:{ bookingId: string; amount: number; applicationFeeCents: number;
connectedAccountId: string; }){
  if (DEMO) {
    // Simulate a session object
    return { id: `cs_demo_${bookingId}`, url: `${SITE}/demo/paid?bookingId=${bookingId}` } as
any;
  }
  const Stripe = (await import("stripe")).default;
  const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, { apiVersion: "2024-06-20" });
  return await stripe.checkout.sessions.create({
    mode: "payment",
    payment_intent_data: {
      application_fee_amount: applicationFeeCents,
      transfer_data: { destination: connectedAccountId },
    },
    line_items: [{ price_data: { currency: "usd", product_data: { name: `Booking ${bookingId}` },
unit_amount: amount }, quantity: 1 }],
    success_url: `${SITE}/booking/${bookingId}?status=success`,
    cancel_url: `${SITE}/booking/${bookingId}?status=cancel`,
    metadata: { bookingId },
  });
}
*/

// app/demo/paid/route.ts — convert booking to CONFIRMED when DEMO fake checkout hits
/*
import { NextRequest, NextResponse } from "next/server";
import { prisma } from "@/lib/db";
import { DEMO } from "@/lib/env";

export async function GET(req: NextRequest){
  if (!DEMO) return NextResponse.redirect(new URL("/", req.url));
  const bookingId = new URL(req.url).searchParams.get("bookingId");
  if (!bookingId) return NextResponse.redirect(new URL("/", req.url));
  await prisma.booking.update({ where: { id: bookingId }, data: { status: "CONFIRMED",
paymentIntentId: `pi_demo_${bookingId}` } });
  return NextResponse.redirect(new URL(`/booking/${bookingId}?status=success`, req.url));
}
*/

// app/(site)/tour/page.tsx — Investor Tour page
/*
export default function Tour(){
  return (
    <div className="prose max-w-none">
      <h1>Investor Tour</h1>
      <ol>
        <li>Sign in (GitHub or Email).</li>
        <li>Open <a href="/browse">Browse</a> → pick a listing.</li>
        <li>Click <b>Book</b> → confirm → (in demo) payment auto-confirms.</li>
        <li>Send a message to the vendor → email is logged to console (demo).</li>
        <li>Admin: <a href="/admin">/admin</a> to approve/feature listings and issue refunds
(demo-safe).</li>
      </ol>
      <p className="text-sm text-gray-600">Demo Mode: no real money, emails are logged,
some flows are simulated.</p>
    </div>
  );
}
*/

// components/DemoBanner.tsx — top flag
/*
```

```
export default function DemoBanner(){
  if (process.env.NEXT_PUBLIC_DEMO !== 'true' && process.env.DEMO_MODE !== 'true')
return null;
  return (
    <div className="w-full bg-yellow-400/90 p-2 text-center text-sm font-medium">DEMO
ENVIRONMENT — No real payments. Data may reset.</div>
  );
}
*/

// app/(site)/layout.tsx — mount DemoBanner under header
/*
import DemoBanner from "@/components/DemoBanner";
export default function RootLayout({ children }: { children: React.ReactNode }) {
  return (
    <html lang="en">
      <body>
        {/* header */}
        <DemoBanner />
        <main className="mx-auto max-w-6xl p-4">{children}</main>
      </body>
    </html>
  );
}
*/

// app/api/demo/seed/route.ts — seed users/listings (idempotent)
/*
import { NextResponse } from "next/server";
import { prisma } from "@/lib/db";
import { DEMO } from "@/lib/env";

export async function POST(){
  if (!DEMO) return NextResponse.json({ error: "Forbidden" }, { status: 403 });
  // Upsert two vendors and two listings each
  const vendorA = await prisma.user.upsert({ where: { email: "chef.a@demo.local" }, update: {},
create: { email: "chef.a@demo.local", role: "VENDOR", name: "Chef A" } });
  const vendorB = await prisma.user.upsert({ where: { email: "chef.b@demo.local" }, update: {},
create: { email: "chef.b@demo.local", role: "VENDOR", name: "Chef B" } });
  const mk = async (ownerId: string, title: string) => prisma.listing.upsert({ where: { slug:
title.toLowerCase().replace(/[^a-z0-9]+/g, '-') }, update: {}, create: { title, slug:
title.toLowerCase().replace(/[^a-z0-9]+/g, '-'), description: "A soulful private dining experience.",
images: ["/og-default.jpg"], category: "CHEF", priceCents: 15000, unit: "PER_EVENT", ownerId,
approved: true } });
```

```
  await Promise.all([ mk(vendorA.id, "Gullah Coastal Dinner"), mk(vendorA.id, "Soul Brunch
Deluxe"), mk(vendorB.id, "BBQ & Blues Night"), mk(vendorB.id, "Date Night Tasting") ]);
  return NextResponse.json({ ok: true });
}
*/

// README — Vercel Demo Steps
/*
1) Push to GitHub → Import in Vercel.
2) Set env: DEMO_MODE=true (and NEXT_PUBLIC_DEMO=true if you want),
NEXTAUTH_SECRET, DATABASE_URL, NEXTAUTH_URL/SITE_URL (use your Vercel
preview URL on first deploy), GITHUB_ID/SECRET. Stripe keys optional in demo.
3) Deploy once → copy the deployed URL → set SITE_URL/NEXTAUTH_URL to that exact
HTTPS URL → redeploy.
4) Seed data: run `POST /api/demo/seed` (use a REST client or browser via a tiny form).
5) Visit `/tour` and walk investors through the flow.
*/
```