# excel-delivery — selected files

## .env

# Environment variables declared in this file are automatically made available to Prisma.  
# See the documentation for more detail: https://pris.ly/d/prisma-schema#accessing-environment-variables-from-the-schema  
  
# Prisma supports the native connection string format for PostgreSQL, MySQL, SQLite, SQL Server, MongoDB and CockroachDB.  
# See the documentation for all the connection string options: https://pris.ly/d/connection-strings  
  
# The following `prisma+postgres` URL is similar to the URL produced by running a local Prisma Postgres   
# server with the `prisma dev` CLI command, when not choosing any non-default ports or settings. The API key, unlike the   
# one found in a remote Prisma Postgres URL, does not contain any sensitive information.  
ALLOW\_SIGNUPS=true  
DATABASE\_URL=postgresql://postgres:FRGKAgQ5y8fRSomq@db.tqlfmswpqbyyhjhjbxpj.supabase.co:5432/postgres?sslmode=require  
NEXTAUTH\_URL=http://localhost:3000  
NEXTAUTH\_SECRET=87a235abe2594ab486f300ecfb3265d5a2cebf84746c1713e0642b6c87361b0b  
SUPABASE\_URL=https://tqlfmswpqbyyhjhjbxpj.supabase.co  
SUPABASE\_SERVICE\_ROLE=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InRxbGZtc3dwcWJ5eWhqaGpieHBqIiwicm9sZSI6InNlcnZpY2Vfcm9sZSIsImlhdCI6MTc1ODEzNDQ4NSwiZXhwIjoyMDczNzEwNDg1fQ.saKNirIWE5BPmNsDjfGaVl7-RZN8Bj1NrZ-S0CJyv6M  
SUPABASE\_BUCKET=excel  
PASSWORD\_PEPPER=939850fb87b0b5a51cec0adbc7f1099a912c8e05b18cfc58c08a5caf5f67b39a  
APP\_URL=http://localhost:3000

## middleware.ts

// middleware.ts  
import { NextResponse } from "next/server";  
import type { NextRequest } from "next/server";  
import { getToken } from "next-auth/jwt";  
  
export async function middleware(req: NextRequest) {  
 const { pathname } = req.nextUrl;  
 const url = (p: string) => new URL(p, req.url);  
  
 // Read JWT (needs NEXTAUTH\_SECRET in .env)  
 const token = await getToken({ req, secret: process.env.NEXTAUTH\_SECRET });  
  
 const needsAuth =  
 pathname === "/dashboard" ||  
 pathname.startsWith("/dashboard/") ||  
 pathname === "/files" ||  
 pathname.startsWith("/files/") ||  
 pathname === "/assignments" ||  
 pathname.startsWith("/assignments/") ||  
 pathname === "/support" ||  
 pathname.startsWith("/support/") ||  
 pathname === "/admin" ||  
 pathname.startsWith("/admin/");  
  
 // Not logged in → go to login  
 if (needsAuth && !token) {  
 const login = url("/login");  
 login.searchParams.set("callbackUrl", pathname);  
 return NextResponse.redirect(login);  
 }  
  
 // Admin only  
 if (pathname === "/admin" || pathname.startsWith("/admin/")) {  
 if (token?.role !== "ADMIN") {  
 return NextResponse.redirect(url("/dashboard"));  
 }  
 }  
  
 // If ADMIN hits /dashboard, route to /admin  
 if (pathname === "/dashboard" && token?.role === "ADMIN") {  
 return NextResponse.redirect(url("/admin"));  
 }  
  
 return NextResponse.next();  
}  
  
export const config = {  
 matcher: [  
 "/dashboard",  
 "/dashboard/:path\*",  
 "/files",  
 "/files/:path\*",  
 "/assignments",  
 "/assignments/:path\*",  
 "/support",  
 "/support/:path\*",  
 "/admin",  
 "/admin/:path\*",  
 ],  
};

## prisma\migrations\20250929175119\_init\migration.sql

-- CreateEnum  
CREATE TYPE "public"."Role" AS ENUM ('USER', 'ADMIN');  
  
-- CreateEnum  
CREATE TYPE "public"."AuditAction" AS ENUM ('USER\_CREATED', 'SUBSCRIPTION\_TOGGLED', 'FILE\_UPLOADED', 'FILE\_ASSIGNED', 'APIKEY\_CREATED', 'APIKEY\_REVOKED', 'DOWNLOAD\_GRANTED');  
  
-- CreateTable  
CREATE TABLE "public"."User" (  
 "id" TEXT NOT NULL,  
 "email" TEXT NOT NULL,  
 "name" TEXT,  
 "role" "public"."Role" NOT NULL DEFAULT 'USER',  
 "passwordHash" TEXT,  
 "subscriptionActive" BOOLEAN NOT NULL DEFAULT false,  
 "subscriptionEndsAt" TIMESTAMP(3),  
 "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 "updatedAt" TIMESTAMP(3) NOT NULL,  
  
 CONSTRAINT "User\_pkey" PRIMARY KEY ("id")  
);  
  
-- CreateTable  
CREATE TABLE "public"."File" (  
 "id" TEXT NOT NULL,  
 "storagePath" TEXT NOT NULL,  
 "originalName" TEXT NOT NULL,  
 "mimeType" TEXT NOT NULL,  
 "sizeBytes" INTEGER NOT NULL,  
 "uploadedById" TEXT NOT NULL,  
 "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 "sha256" TEXT,  
  
 CONSTRAINT "File\_pkey" PRIMARY KEY ("id")  
);  
  
-- CreateTable  
CREATE TABLE "public"."FileAssignment" (  
 "id" TEXT NOT NULL,  
 "userId" TEXT NOT NULL,  
 "fileId" TEXT NOT NULL,  
 "assignedById" TEXT NOT NULL,  
 "note" TEXT,  
 "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
  
 CONSTRAINT "FileAssignment\_pkey" PRIMARY KEY ("id")  
);  
  
-- CreateTable  
CREATE TABLE "public"."ApiKey" (  
 "id" TEXT NOT NULL,  
 "label" TEXT NOT NULL,  
 "keyHash" TEXT NOT NULL,  
 "isActive" BOOLEAN NOT NULL DEFAULT true,  
 "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
 "lastUsedAt" TIMESTAMP(3),  
  
 CONSTRAINT "ApiKey\_pkey" PRIMARY KEY ("id")  
);  
  
-- CreateTable  
CREATE TABLE "public"."AuditLog" (  
 "id" TEXT NOT NULL,  
 "actorId" TEXT,  
 "action" "public"."AuditAction" NOT NULL,  
 "targetId" TEXT,  
 "target" TEXT,  
 "meta" JSONB,  
 "createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURRENT\_TIMESTAMP,  
  
 CONSTRAINT "AuditLog\_pkey" PRIMARY KEY ("id")  
);  
  
-- CreateIndex  
CREATE UNIQUE INDEX "User\_email\_key" ON "public"."User"("email");  
  
-- CreateIndex  
CREATE UNIQUE INDEX "FileAssignment\_userId\_fileId\_key" ON "public"."FileAssignment"("userId", "fileId");  
  
-- AddForeignKey  
ALTER TABLE "public"."File" ADD CONSTRAINT "File\_uploadedById\_fkey" FOREIGN KEY ("uploadedById") REFERENCES "public"."User"("id") ON DELETE RESTRICT ON UPDATE CASCADE;  
  
-- AddForeignKey  
ALTER TABLE "public"."FileAssignment" ADD CONSTRAINT "FileAssignment\_userId\_fkey" FOREIGN KEY ("userId") REFERENCES "public"."User"("id") ON DELETE RESTRICT ON UPDATE CASCADE;  
  
-- AddForeignKey  
ALTER TABLE "public"."FileAssignment" ADD CONSTRAINT "FileAssignment\_assignedById\_fkey" FOREIGN KEY ("assignedById") REFERENCES "public"."User"("id") ON DELETE RESTRICT ON UPDATE CASCADE;  
  
-- AddForeignKey  
ALTER TABLE "public"."FileAssignment" ADD CONSTRAINT "FileAssignment\_fileId\_fkey" FOREIGN KEY ("fileId") REFERENCES "public"."File"("id") ON DELETE RESTRICT ON UPDATE CASCADE;  
  
-- AddForeignKey  
ALTER TABLE "public"."AuditLog" ADD CONSTRAINT "AuditLog\_actorId\_fkey" FOREIGN KEY ("actorId") REFERENCES "public"."User"("id") ON DELETE SET NULL ON UPDATE CASCADE;

## prisma\migrations\20251015132023\_user\_account\_status\migration.sql

/\*  
 Warnings:  
  
 - You are about to drop the column `subscriptionEndsAt` on the `User` table. All the data in the column will be lost.  
 - Made the column `name` on table `User` required. This step will fail if there are existing NULL values in that column.  
 - Made the column `passwordHash` on table `User` required. This step will fail if there are existing NULL values in that column.  
  
\*/  
-- CreateEnum  
CREATE TYPE "public"."AccountStatus" AS ENUM ('PENDING', 'ACTIVE', 'SUSPENDED');  
  
-- AlterTable  
ALTER TABLE "public"."User" DROP COLUMN "subscriptionEndsAt",  
ADD COLUMN "status" "public"."AccountStatus" NOT NULL DEFAULT 'PENDING',  
ALTER COLUMN "name" SET NOT NULL,  
ALTER COLUMN "passwordHash" SET NOT NULL;

## prisma\migrations\20251015142120\_add\_auditlog\_backrelation\migration.sql

/\*  
 Warnings:  
  
 - You are about to drop the column `mimeType` on the `File` table. All the data in the column will be lost.  
 - You are about to drop the column `sha256` on the `File` table. All the data in the column will be lost.  
 - You are about to drop the column `sizeBytes` on the `File` table. All the data in the column will be lost.  
 - You are about to drop the column `storagePath` on the `File` table. All the data in the column will be lost.  
 - A unique constraint covering the columns `[fileId,userId]` on the table `FileAssignment` will be added. If there are existing duplicate values, this will fail.  
 - Added the required column `title` to the `File` table without a default value. This is not possible if the table is not empty.  
 - Added the required column `updatedAt` to the `File` table without a default value. This is not possible if the table is not empty.  
  
\*/  
-- DropForeignKey  
ALTER TABLE "public"."File" DROP CONSTRAINT "File\_uploadedById\_fkey";  
  
-- DropForeignKey  
ALTER TABLE "public"."FileAssignment" DROP CONSTRAINT "FileAssignment\_fileId\_fkey";  
  
-- DropForeignKey  
ALTER TABLE "public"."FileAssignment" DROP CONSTRAINT "FileAssignment\_userId\_fkey";  
  
-- DropIndex  
DROP INDEX "public"."FileAssignment\_userId\_fileId\_key";  
  
-- AlterTable  
ALTER TABLE "public"."File" DROP COLUMN "mimeType",  
DROP COLUMN "sha256",  
DROP COLUMN "sizeBytes",  
DROP COLUMN "storagePath",  
ADD COLUMN "mime" TEXT,  
ADD COLUMN "size" INTEGER,  
ADD COLUMN "title" TEXT NOT NULL,  
ADD COLUMN "updatedAt" TIMESTAMP(3) NOT NULL,  
ADD COLUMN "url" TEXT,  
ALTER COLUMN "originalName" DROP NOT NULL,  
ALTER COLUMN "uploadedById" DROP NOT NULL;  
  
-- CreateIndex  
CREATE INDEX "AuditLog\_actorId\_idx" ON "public"."AuditLog"("actorId");  
  
-- CreateIndex  
CREATE INDEX "File\_uploadedById\_idx" ON "public"."File"("uploadedById");  
  
-- CreateIndex  
CREATE INDEX "File\_createdAt\_idx" ON "public"."File"("createdAt");  
  
-- CreateIndex  
CREATE INDEX "FileAssignment\_userId\_createdAt\_idx" ON "public"."FileAssignment"("userId", "createdAt");  
  
-- CreateIndex  
CREATE INDEX "FileAssignment\_assignedById\_idx" ON "public"."FileAssignment"("assignedById");  
  
-- CreateIndex  
CREATE UNIQUE INDEX "FileAssignment\_fileId\_userId\_key" ON "public"."FileAssignment"("fileId", "userId");  
  
-- AddForeignKey  
ALTER TABLE "public"."File" ADD CONSTRAINT "File\_uploadedById\_fkey" FOREIGN KEY ("uploadedById") REFERENCES "public"."User"("id") ON DELETE SET NULL ON UPDATE CASCADE;  
  
-- AddForeignKey  
ALTER TABLE "public"."FileAssignment" ADD CONSTRAINT "FileAssignment\_fileId\_fkey" FOREIGN KEY ("fileId") REFERENCES "public"."File"("id") ON DELETE CASCADE ON UPDATE CASCADE;  
  
-- AddForeignKey  
ALTER TABLE "public"."FileAssignment" ADD CONSTRAINT "FileAssignment\_userId\_fkey" FOREIGN KEY ("userId") REFERENCES "public"."User"("id") ON DELETE CASCADE ON UPDATE CASCADE;

## prisma\schema.prisma

generator client {  
 provider = "prisma-client-js"  
}  
  
datasource db {  
 provider = "postgresql"  
 url = env("DATABASE\_URL")  
}  
  
enum Role {  
 USER  
 ADMIN  
}  
  
enum AuditAction {  
 USER\_CREATED  
 SUBSCRIPTION\_TOGGLED  
 FILE\_UPLOADED  
 FILE\_ASSIGNED  
 APIKEY\_CREATED  
 APIKEY\_REVOKED  
 DOWNLOAD\_GRANTED  
}  
  
enum AccountStatus {  
 PENDING  
 ACTIVE  
 SUSPENDED  
}  
  
model User {  
 id String @id @default(cuid())  
 name String  
 email String @unique  
 passwordHash String  
 role Role @default(USER)  
 subscriptionActive Boolean @default(false)  
 status AccountStatus @default(PENDING)  
 createdAt DateTime @default(now())  
 updatedAt DateTime @updatedAt  
  
 // Relations  
 uploads File[] @relation("UserUploads")  
 assignments FileAssignment[] @relation("UserAssignments")  
 assignmentsGiven FileAssignment[] @relation("AssignmentsAssignedBy")  
 auditLogs AuditLog[] @relation("AuditActor") // 👈 add this  
}  
  
model File {  
 id String @id @default(cuid())  
 title String  
 originalName String?  
 url String? // for external or pre-signed location (we’ll wire actual uploads next day)  
 mime String?  
 size Int?  
 uploadedBy User? @relation("UserUploads", fields: [uploadedById], references: [id])  
 uploadedById String?  
 createdAt DateTime @default(now())  
 updatedAt DateTime @updatedAt  
  
 // Assignments  
 assignments FileAssignment[]  
  
 @@index([uploadedById])  
 @@index([createdAt])  
}  
  
model FileAssignment {  
 id String @id @default(cuid())  
 file File @relation(fields: [fileId], references: [id], onDelete: Cascade)  
 fileId String  
 user User @relation("UserAssignments", fields: [userId], references: [id], onDelete: Cascade)  
 userId String  
 assignedBy User @relation("AssignmentsAssignedBy", fields: [assignedById], references: [id])  
 assignedById String  
 note String?  
 createdAt DateTime @default(now())  
  
 @@unique([fileId, userId]) // prevent double-assigning the same file to same user  
 @@index([userId, createdAt])  
 @@index([assignedById])  
}  
  
model ApiKey {  
 id String @id @default(cuid())  
 label String  
 keyHash String  
 isActive Boolean @default(true)  
 createdAt DateTime @default(now())  
 lastUsedAt DateTime?  
}  
  
model AuditLog {  
 id String @id @default(cuid())  
 actorId String?  
 actor User? @relation("AuditActor", fields: [actorId], references: [id])  
 action AuditAction  
 targetId String?  
 target String?  
 meta Json?  
 createdAt DateTime @default(now())  
  
 @@index([actorId]) // optional  
}

## prisma\seed.ts

import { PrismaClient } from "@prisma/client";  
import bcrypt from "bcrypt";  
  
const prisma = new PrismaClient();  
  
async function main() {  
 const email = "admin@example.com";  
 const password = "Admin123!";  
 const passwordHash = await bcrypt.hash(password, 12);  
  
 await prisma.user.upsert({  
 where: { email },  
 update: { role: "ADMIN", passwordHash, subscriptionActive: true },  
 create: { email, role: "ADMIN", name: "Admin", passwordHash, subscriptionActive: true },  
 });  
  
 console.log("Admin ready:", email, password);  
}  
  
main().finally(() => prisma.$disconnect());

## src\app\(admin)\admin\audit\page.tsx

"use client";  
import { useEffect, useState } from "react";  
  
type Row = {  
 id: string;  
 createdAt: string;  
 action: string;  
 target?: string | null;  
 targetId?: string | null;  
 actor?: { id: string; email: string; name?: string | null } | null;  
 meta?: any;  
};  
  
export default function AdminAuditPage() {  
 const [rows, setRows] = useState<Row[]>([]);  
 const [cursor, setCursor] = useState<string | null>(null);  
 const [loading, setLoading] = useState(false);  
 const [hasMore, setHasMore] = useState(true);  
  
 async function load(next?: string | null) {  
 setLoading(true);  
 const q = new URLSearchParams({ limit: "50" });  
 if (next) q.set("cursor", next);  
 const r = await fetch(`/api/admin/audit?${q.toString()}`, { cache: "no-store" });  
 const { items, nextCursor } = await r.json();  
 setRows(prev => next ? [...prev, ...items] : items);  
 setCursor(nextCursor);  
 setHasMore(!!nextCursor);  
 setLoading(false);  
 }  
  
 useEffect(() => { load(null); }, []);  
  
 return (  
 <div className="grid gap-4">  
 <h1 className="text-xl font-semibold">Audit Log</h1>  
  
 <section className="rounded-[var(--radius)] border border-[color:var(--border)] bg-[color:var(--card)] p-4 overflow-x-auto">  
 <table className="w-full text-sm">  
 <thead>  
 <tr className="text-left text-[color:var(--muted)] border-b border-[color:var(--border)]">  
 <th className="py-2 pr-3">Time</th>  
 <th className="py-2 pr-3">Action</th>  
 <th className="py-2 pr-3">Actor</th>  
 <th className="py-2 pr-3">Target</th>  
 <th className="py-2 pr-3">Meta</th>  
 </tr>  
 </thead>  
 <tbody>  
 {rows.map((r) => (  
 <tr key={r.id} className="border-b last:border-0 border-[color:var(--border)] align-top">  
 <td className="py-2 pr-3 whitespace-nowrap">{new Date(r.createdAt).toLocaleString()}</td>  
 <td className="py-2 pr-3 font-medium">{r.action}</td>  
 <td className="py-2 pr-3">  
 {r.actor ? `${r.actor.email}${r.actor.name ? ` (${r.actor.name})` : ""}` : "—"}  
 </td>  
 <td className="py-2 pr-3">  
 {r.target ?? "—"} {r.targetId ? <span className="text-[color:var(--muted)]">#{r.targetId}</span> : null}  
 </td>  
 <td className="py-2 pr-3">  
 <pre className="max-w-[40ch] whitespace-pre-wrap break-words text-xs bg-black/5 rounded p-2">  
 {r.meta ? JSON.stringify(r.meta, null, 2) : "—"}  
 </pre>  
 </td>  
 </tr>  
 ))}  
 </tbody>  
 </table>  
  
 <div className="mt-3 flex justify-end">  
 {hasMore && (  
 <button  
 onClick={() => load(cursor)}  
 disabled={loading}  
 className="rounded border border-[color:var(--border)] px-3 py-2 text-sm"  
 >  
 {loading ? "Loading…" : "Load more"}  
 </button>  
 )}  
 </div>  
 </section>  
 </div>  
 );  
}

## src\app\(admin)\admin\files\page.tsx

"use client";  
import { useEffect, useState } from "react";  
  
type FileRow = {  
 id: string;  
 title: string;  
 createdAt: string;  
 url?: string | null;  
 assignments?: { user: { id: string; email: string; name?: string | null } }[];  
};  
  
type UserRow = { id: string; email: string; name: string | null; status?: string };  
  
export default function AdminFilesPage() {  
 const [files, setFiles] = useState<FileRow[]>([]);  
 const [users, setUsers] = useState<UserRow[]>([]);  
 const [loading, setLoading] = useState(true);  
  
 async function load() {  
 setLoading(true);  
 const f = await fetch("/api/files?scope=all", { cache: "no-store" });  
 if (f.ok) setFiles((await f.json()).files);  
 const u = await fetch("/api/admin/users", { cache: "no-store" });  
 if (u.ok) setUsers((await u.json()).users);  
 setLoading(false);  
 }  
 useEffect(() => { load(); }, []);  
  
 async function assign(fileId: string, userId: string) {  
 if (!userId) return;  
 const res = await fetch("/api/assignments", {  
 method: "POST",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify({ fileId, userId }),  
 });  
 if (!res.ok) alert("Failed to assign"); else load();  
 }  
  
 return (  
 <div className="grid gap-4">  
 <h2 className="text-xl font-semibold">All Files</h2>  
 {loading ? "Loading…" : (  
 <table className="w-full text-sm">  
 <thead>  
 <tr className="text-left text-[color:var(--muted)] border-b border-[color:var(--border)]">  
 <th className="py-2 pr-3">Title</th>  
 <th className="py-2 pr-3">Created</th>  
 <th className="py-2 pr-3">Assigned to</th>  
 <th className="py-2 pr-3">Assign</th>  
 <th className="py-2 pr-3">Actions</th>  
 </tr>  
 </thead>  
 <tbody>  
 {files.map(f => {  
 const assigned = (f.assignments || []).map(a => a.user.email).join(", ");  
 return (  
 <tr key={f.id} className="border-b last:border-0 border-[color:var(--border)]">  
 <td className="py-2 pr-3">  
 {f.title}  
 </td>  
 <td className="py-2 pr-3">{new Date(f.createdAt).toLocaleString()}</td>  
 <td className="py-2 pr-3">  
 {assigned ? assigned : <span className="text-[color:var(--muted)]">No user assigned</span>}  
 </td>  
 <td className="py-2 pr-3">  
 <form  
 onSubmit={e => {  
 e.preventDefault();  
 const userId = (new FormData(e.currentTarget).get("userId") as string) || "";  
 assign(f.id, userId);  
 }}  
 className="flex gap-2"  
 >  
 <select name="userId" className="border rounded px-2 py-1">  
 <option value="">Select user…</option>  
 {users  
 .filter(u => u.status === "ACTIVE")  
 .map(u => (  
 <option key={u.id} value={u.id}>  
 {u.email}{u.name ? ` (${u.name})` : ""}  
 </option>  
 ))}  
 </select>  
 <button className="rounded bg-[color:var(--brand)] text-black px-3 py-1">Assign</button>  
 </form>  
 </td>  
 <td className="py-2 pr-3">  
 {f.url ? (  
 <a href={f.url} target="\_blank" rel="noreferrer" className="rounded border px-3 py-1 hover:bg-black/5">  
 Download  
 </a>  
 ) : (  
 <span className="text-[color:var(--muted)]">No file URL</span>  
 )}  
 </td>  
 </tr>  
 );  
 })}  
 </tbody>  
 </table>  
 )}  
 </div>  
 );  
}

## src\app\(admin)\admin\page.tsx

// src/app/(admin)/admin/page.tsx  
"use client";  
  
import { useEffect, useMemo, useState } from "react";  
import Link from "next/link";  
import DashboardCard from "@/components/DashboardCard";  
import TrendMini from "@/components/TrendMini";  
  
type Stats = { users: number; pending: number; files: number };  
type FileRow = { id: string; createdAt: string };  
type AuditRow = {  
 id: string;  
 createdAt: string;  
 action: string;  
 target?: string | null;  
 targetId?: string | null;  
 actor?: { email: string; name?: string | null } | null;  
 meta?: any;  
};  
  
export default function AdminDashboard() {  
 const [stats, setStats] = useState<Stats | null>(null);  
 const [audit, setAudit] = useState<AuditRow[]>([]);  
 const [files, setFiles] = useState<FileRow[]>([]);  
  
 useEffect(() => {  
 (async () => {  
 const s = await fetch("/api/stats", { cache: "no-store" });  
 if (s.ok) setStats(await s.json());  
  
 const a = await fetch("/api/admin/audit?limit=10", { cache: "no-store" });  
 if (a.ok) setAudit((await a.json()).items);  
  
 const f = await fetch("/api/files?scope=all", { cache: "no-store" });  
 if (f.ok) {  
 const json = await f.json();  
 setFiles(json.files.map((x: any) => ({ id: x.id, createdAt: x.createdAt })));  
 }  
 })();  
 }, []);  
  
 // Make a 30-day series from file createdAt timestamps  
 const uploadsSeries = useMemo(() => {  
 const byDay = new Map<string, number>();  
 const fmt = (d: Date) => d.toISOString().slice(0, 10);  
 const today = new Date();  
 for (let i = 29; i >= 0; i--) {  
 const d = new Date(today);  
 d.setDate(today.getDate() - i);  
 byDay.set(fmt(d), 0);  
 }  
 for (const f of files) {  
 const k = fmt(new Date(f.createdAt));  
 if (byDay.has(k)) byDay.set(k, (byDay.get(k) || 0) + 1);  
 }  
 return Array.from(byDay.entries()).map(([k, v]) => ({ x: k, y: v }));  
 }, [files]);  
  
 const totalUploads = uploadsSeries.reduce((a, b) => a + b.y, 0);  
  
 return (  
 <div className="grid gap-4">  
 {/\* Top stats row \*/}  
 <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-4">  
 <DashboardCard title="Users" value={stats?.users ?? "—"} subtitle="Total registered" />  
 <DashboardCard title="Pending approvals" value={stats?.pending ?? "—"} subtitle="Awaiting activation" />  
 <DashboardCard title="Files" value={stats?.files ?? "—"} subtitle="Total uploaded" />  
 <DashboardCard title="Quick actions" subtitle="Common admin tasks">  
 <div className="mt-3 flex flex-wrap gap-2">  
 <Link href="/admin/users" className="rounded border px-3 py-1 text-sm">Manage users</Link>  
 <Link href="/admin/uploads" className="rounded border px-3 py-1 text-sm">Upload file</Link>  
 <Link href="/admin/audit" className="rounded border px-3 py-1 text-sm">View logs</Link>  
 </div>  
 </DashboardCard>  
 </div>  
  
 {/\* Uploads trend + recent audit \*/}  
 <div className="grid gap-4 lg:grid-cols-3">  
 <DashboardCard title="Uploads — last 30 days">  
 <div className="mb-2 flex items-center justify-between">  
 <span className="text-sm text-[color:var(--muted)]">Daily count</span>  
 <span className="text-xs text-[color:var(--muted)]">{totalUploads} total</span>  
 </div>  
 <TrendMini data={uploadsSeries} />  
 </DashboardCard>  
  
 <section className="lg:col-span-2 rounded-[var(--radius)] border border-[color:var(--border)] bg-[color:var(--card)] shadow-sm p-4 overflow-x-auto">  
 <div className="flex items-center justify-between mb-2">  
 <h2 className="font-semibold">Recent activity</h2>  
 <Link href="/admin/audit" className="text-sm underline">View all</Link>  
 </div>  
 {audit.length === 0 ? (  
 <p className="text-sm text-[color:var(--muted)]">No activity yet.</p>  
 ) : (  
 <table className="w-full text-sm">  
 <thead>  
 <tr className="text-left text-[color:var(--muted)] border-b border-[color:var(--border)]">  
 <th className="py-2 pr-3">Time</th>  
 <th className="py-2 pr-3">Action</th>  
 <th className="py-2 pr-3">Actor</th>  
 <th className="py-2 pr-3">Target</th>  
 <th className="py-2 pr-3">Meta</th>  
 </tr>  
 </thead>  
 <tbody>  
 {audit.map((r) => (  
 <tr key={r.id} className="border-b last:border-0 border-[color:var(--border)] align-top">  
 <td className="py-2 pr-3 whitespace-nowrap">{new Date(r.createdAt).toLocaleString()}</td>  
 <td className="py-2 pr-3 font-medium">{r.action}</td>  
 <td className="py-2 pr-3">{r.actor ? (r.actor.email + (r.actor.name ? ` (${r.actor.name})` : "")) : "—"}</td>  
 <td className="py-2 pr-3">  
 {r.target ?? "—"} {r.targetId ? <span className="text-[color:var(--muted)]">#{r.targetId}</span> : null}  
 </td>  
 <td className="py-2 pr-3">  
 <pre className="max-w-[40ch] whitespace-pre-wrap break-words text-xs bg-black/5 rounded p-2">  
 {r.meta ? JSON.stringify(r.meta, null, 2) : "—"}  
 </pre>  
 </td>  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 )}  
 </section>  
 </div>  
 </div>  
 );  
}

## src\app\(admin)\admin\settings\page.tsx

export default function AdminSettingsPage() {  
 return (  
 <div className="rounded-[var(--radius)] border border-[color:var(--border)] bg-[color:var(--card)] p-4">  
 <h2 className="font-semibold">Settings</h2>  
 <p className="text-sm text-[color:var(--muted)]">Platform settings.</p>  
 </div>  
 );  
}

## src\app\(admin)\admin\uploads\page.tsx

"use client";  
import { useState } from "react";  
  
export default function AdminUploadPage() {  
 const [file, setFile] = useState<File | null>(null);  
 const [title, setTitle] = useState("");  
 const [status, setStatus] = useState<string | null>(null);  
  
 async function handleUpload(e: React.FormEvent) {  
 e.preventDefault();  
 if (!file) return setStatus("No file selected.");  
  
 const fd = new FormData();  
 fd.append("file", file);  
 fd.append("title", title || file.name);  
  
 const res = await fetch("/api/uploads", { method: "POST", body: fd });  
 const json = await res.json();  
 if (!res.ok) return setStatus(`❌ ${json.error}`);  
 setStatus(`✅ Uploaded: ${json.file.title}`);  
 setFile(null);  
 setTitle("");  
 }  
  
 return (  
 <main className="grid gap-4 max-w-lg">  
 <h1 className="text-xl font-semibold">Upload Excel File</h1>  
 <form onSubmit={handleUpload} className="grid gap-3">  
 <input  
 type="text"  
 placeholder="Title (optional)"  
 value={title}  
 onChange={e => setTitle(e.target.value)}  
 className="border rounded px-3 py-2"  
 />  
 <input  
 type="file"  
 accept=".xlsx,.xls"  
 onChange={e => setFile(e.target.files?.[0] || null)}  
 className="border rounded px-3 py-2"  
 />  
 <button className="rounded bg-[color:var(--brand)] text-black px-4 py-2">  
 Upload  
 </button>  
 </form>  
 {status && <p className="text-sm">{status}</p>}  
 </main>  
 );  
}

## src\app\(admin)\admin\users\page.tsx

"use client";  
  
import { useEffect, useMemo, useState } from "react";  
  
type User = {  
 id: string;  
 name: string | null;  
 email: string;  
 role: "USER" | "ADMIN";  
 subscriptionActive: boolean;  
 status: "PENDING" | "ACTIVE" | "SUSPENDED"; // 👈 NEW  
 createdAt: string;  
};  
  
export default function AdminUsersPage() {  
 const [users, setUsers] = useState<User[]>([]);  
 const [loading, setLoading] = useState(true);  
 const [err, setErr] = useState<string | null>(null);  
 const [filter, setFilter] = useState<"ALL" | "PENDING" | "ACTIVE" | "SUSPENDED">("ALL");  
  
 // create form state  
 const [form, setForm] = useState({  
 name: "",  
 email: "",  
 password: "",  
 role: "USER" as "USER" | "ADMIN",  
 subscriptionActive: false,  
 });  
 const canCreate = useMemo(() => {  
 return form.name.trim().length >= 2 && /\S+@\S+\.\S+/.test(form.email) && form.password.length >= 6;  
 }, [form]);  
  
 async function load() {  
 setLoading(true); setErr(null);  
 try {  
 const q = filter === "ALL" ? "" : `?status=${filter}`;  
 const res = await fetch(`/api/admin/users${q}`, { cache: "no-store" });  
 const json = await res.json();  
 if (!res.ok) throw new Error(json?.error || "Failed to load users");  
 setUsers(json.users);  
 } catch (e: any) { setErr(e.message); }  
 finally { setLoading(false); }  
 }  
 useEffect(() => { load(); }, [filter]);  
  
 useEffect(() => { load(); }, []);  
  
 async function toggleSubscription(u: User) {  
 const optimistic = users.map(x => x.id === u.id ? { ...x, subscriptionActive: !u.subscriptionActive } : x);  
 setUsers(optimistic);  
 const res = await fetch(`/api/admin/users/${u.id}`, {  
 method: "PATCH",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify({ subscriptionActive: !u.subscriptionActive }),  
 });  
 if (!res.ok) load(); // rollback by reload  
 }  
  
 async function changeRole(u: User, role: "USER" | "ADMIN") {  
 const optimistic = users.map(x => x.id === u.id ? { ...x, role } : x);  
 setUsers(optimistic);  
 const res = await fetch(`/api/admin/users/${u.id}`, {  
 method: "PATCH",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify({ role }),  
 });  
 if (!res.ok) load();  
 }  
  
 async function remove(u: User) {  
 if (!confirm(`Delete user ${u.email}? This cannot be undone.`)) return;  
 const optimistic = users.filter(x => x.id !== u.id);  
 setUsers(optimistic);  
 const res = await fetch(`/api/admin/users/${u.id}`, { method: "DELETE" });  
 if (!res.ok) load();  
 }  
  
 async function createUser(e: React.FormEvent) {  
 e.preventDefault();  
 const res = await fetch("/api/admin/users", {  
 method: "POST",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify(form),  
 });  
 const json = await res.json();  
 if (!res.ok) { alert(json?.error || "Failed to create user"); return; }  
 setForm({ name: "", email: "", password: "", role: "USER", subscriptionActive: false });  
 load();  
 }  
  
 async function setStatus(u: User, status: User["status"]) {  
 const optimistic = users.map(x => x.id === u.id ? { ...x, status } : x);  
 setUsers(optimistic);  
 const res = await fetch(`/api/admin/users/${u.id}`, {  
 method: "PATCH",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify({ status }),  
 });  
 if (!res.ok) load();  
 }  
  
 return (  
 <div className="grid gap-6">  
 {/\* Filter bar \*/}  
 <div className="flex gap-2">  
 {(["ALL","PENDING","ACTIVE","SUSPENDED"] as const).map(f => (  
 <button  
 key={f}  
 onClick={() => setFilter(f)}  
 className={[  
 "rounded-md border px-3 py-2 text-sm",  
 filter === f  
 ? "bg-[color:var(--brand)] text-black border-transparent"  
 : "border-[color:var(--border)]"  
 ].join(" ")}  
 >  
 {f}  
 </button>  
 ))}  
 <div className="ml-auto">  
 <button onClick={load} className="rounded-md border border-[color:var(--border)] px-3 py-2 text-sm">Refresh</button>  
 </div>  
 </div>  
  
 {/\* Create user \*/}  
 <section className="rounded-[var(--radius)] border border-[color:var(--border)] bg-[color:var(--card)] p-4">  
 <h2 className="font-semibold mb-3">Create User</h2>  
 <form onSubmit={createUser} className="grid gap-3 md:grid-cols-5">  
 <input  
 className="rounded-md border border-[color:var(--border)] bg-white/90 px-3 py-2 md:col-span-1"  
 placeholder="Name"  
 value={form.name}  
 onChange={e => setForm({ ...form, name: e.target.value })}  
 required  
 />  
 <input  
 type="email"  
 className="rounded-md border border-[color:var(--border)] bg-white/90 px-3 py-2 md:col-span-2"  
 placeholder="Email"  
 value={form.email}  
 onChange={e => setForm({ ...form, email: e.target.value })}  
 required  
 />  
 <input  
 type="password"  
 className="rounded-md border border-[color:var(--border)] bg-white/90 px-3 py-2 md:col-span-1"  
 placeholder="Password"  
 value={form.password}  
 onChange={e => setForm({ ...form, password: e.target.value })}  
 required  
 />  
 <div className="flex items-center gap-2 md:col-span-1">  
 <select  
 className="rounded-md border border-[color:var(--border)] bg-white/90 px-3 py-2"  
 value={form.role}  
 onChange={e => setForm({ ...form, role: e.target.value as any })}  
 >  
 <option value="USER">USER</option>  
 <option value="ADMIN">ADMIN</option>  
 </select>  
 <label className="inline-flex items-center gap-2 text-sm">  
 <input  
 type="checkbox"  
 checked={form.subscriptionActive}  
 onChange={e => setForm({ ...form, subscriptionActive: e.target.checked })}  
 />  
 Subscription  
 </label>  
 </div>  
 <div className="md:col-span-5">  
 <button  
 type="submit"  
 disabled={!canCreate}  
 className="rounded-md bg-[color:var(--brand)] hover:bg-[color:var(--brand-600)] text-black font-medium px-4 py-2 disabled:opacity-50"  
 >  
 Create  
 </button>  
 </div>  
 </form>  
 </section>  
  
 {/\* Users table \*/}  
 <section className="rounded-[var(--radius)] border border-[color:var(--border)] bg-[color:var(--card)] p-4 overflow-x-auto">  
 {/\* ...heading... \*/}  
 {loading ? (  
 <div className="text-sm text-[color:var(--muted)]">Loading…</div>  
 ) : users.length === 0 ? (  
 <div className="text-sm text-[color:var(--muted)]">No users found.</div>  
 ) : (  
 <table className="w-full text-sm">  
 <thead>  
 <tr className="text-left text-[color:var(--muted)] border-b border-[color:var(--border)]">  
 <th className="py-2 pr-3">Name</th>  
 <th className="py-2 pr-3">Email</th>  
 <th className="py-2 pr-3">Role</th>  
 <th className="py-2 pr-3">Status</th>  
 <th className="py-2 pr-3">Subscription</th>  
 <th className="py-2 pr-3">Created</th>  
 <th className="py-2 pr-3">Actions</th>  
 </tr>  
 </thead>  
 <tbody>  
 {users.map(u => (  
 <tr key={u.id} className="border-b last:border-0 border-[color:var(--border)]">  
 <td className="py-2 pr-3">{u.name ?? "—"}</td>  
 <td className="py-2 pr-3">{u.email}</td>  
 <td className="py-2 pr-3">  
 <select  
 value={u.role}  
 onChange={e => changeRole(u, e.target.value as "USER" | "ADMIN")}  
 className="rounded-md border border-[color:var(--border)] bg-white/90 px-2 py-1"  
 >  
 <option value="USER">USER</option>  
 <option value="ADMIN">ADMIN</option>  
 </select>  
 </td>  
 <td className="py-2 pr-3">  
 <span className={[  
 "inline-flex items-center rounded-full px-2 py-0.5 border",  
 u.status === "ACTIVE" ? "border-green-300 text-green-700 bg-green-50" :  
 u.status === "PENDING" ? "border-amber-300 text-amber-700 bg-amber-50" :  
 "border-red-300 text-red-700 bg-red-50"  
 ].join(" ")}>  
 {u.status}  
 </span>  
 </td>  
 <td className="py-2 pr-3">  
 <label className="inline-flex items-center gap-2">  
 <input  
 type="checkbox"  
 checked={u.subscriptionActive}  
 onChange={() => toggleSubscription(u)}  
 disabled={u.status !== "ACTIVE"} // optional: only active users can have subscription  
 />  
 {u.subscriptionActive ? "Active" : "Inactive"}  
 </label>  
 </td>  
 <td className="py-2 pr-3">{new Date(u.createdAt).toLocaleDateString()}</td>  
 <td className="py-2 pr-3 flex gap-2">  
 {u.status !== "ACTIVE" && (  
 <button  
 onClick={() => setStatus(u, "ACTIVE")}  
 className="rounded-md bg-green-600/90 text-white px-3 py-1 hover:bg-green-700"  
 >  
 Approve  
 </button>  
 )}  
 {u.status === "ACTIVE" && (  
 <button  
 onClick={() => setStatus(u, "SUSPENDED")}  
 className="rounded-md bg-yellow-500/90 text-black px-3 py-1 hover:bg-yellow-500"  
 >  
 Suspend  
 </button>  
 )}  
 <button  
 onClick={() => remove(u)}  
 className="rounded-md border border-red-200 text-red-700 px-3 py-1 hover:bg-red-50"  
 >  
 Delete  
 </button>  
 </td>  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 )}  
 </section>  
 </div>  
 );  
}

## src\app\(admin)\layout.tsx

// src/app/(admin)/layout.tsx  
import { redirect } from "next/navigation";  
import AppShell from "@/components/AppShell";  
import { currentUser } from "@/lib/auth-helpers";  
  
export default async function AdminLayout({  
 children,  
}: {  
 children: React.ReactNode;  
}) {  
 const user = await currentUser();  
 if (!user || user.role !== "ADMIN") {  
 redirect("/login?next=/admin");  
 }  
  
 // AppShell will render the Sidebar and content area  
 return <AppShell>{children}</AppShell>;  
}

## src\app\(user)\dashboard\page.tsx

"use client";  
  
import { useEffect, useMemo, useState } from "react";  
import DashboardCard from "@/components/DashboardCard";  
import TrendMini from "@/components/TrendMini";  
  
type Stats = { myFiles:number; myAssigned:number };  
type FileRow = { id:string; title:string; originalName?:string|null; url?:string|null; createdAt:string };  
  
export default function UserDashboard() {  
 const [stats, setStats] = useState<Stats | null>(null);  
 const [assigned, setAssigned] = useState<FileRow[]>([]);  
  
 useEffect(() => {  
 (async () => {  
 const s = await fetch("/api/stats", { cache: "no-store" });  
 if (s.ok) setStats(await s.json());  
 const a = await fetch("/api/files?scope=assigned", { cache: "no-store" });  
 if (a.ok) setAssigned((await a.json()).files);  
 })();  
 }, []);  
  
 const assignedSeries = useMemo(() => {  
 const byDay = new Map<string, number>();  
 const fmt = (d:Date)=> d.toISOString().slice(0,10);  
 const today = new Date();  
 for (let i=29;i>=0;i--){  
 const d = new Date(today); d.setDate(today.getDate()-i);  
 byDay.set(fmt(d), 0);  
 }  
 for (const f of assigned) {  
 const k = fmt(new Date(f.createdAt));  
 if (byDay.has(k)) byDay.set(k, (byDay.get(k) || 0) + 1);  
 }  
 return Array.from(byDay.entries()).map(([k,v])=>({ x:k, y:v }));  
 }, [assigned]);  
  
 return (  
 <div className="grid gap-4">  
 <div className="grid gap-4 md:grid-cols-2 lg:grid-cols-4">  
 <DashboardCard title="Files assigned to me" value={stats?.myAssigned ?? "—"} subtitle="Total assignments" />  
 <DashboardCard title="My uploads" value={stats?.myFiles ?? "—"} subtitle="Files I uploaded" />  
 <DashboardCard title="Trend (30 days)">  
 <TrendMini data={assignedSeries} />  
 </DashboardCard>  
 <DashboardCard title="Shortcuts" subtitle="Common actions">  
 <div className="mt-3 flex flex-wrap gap-2">  
 <a href="/files" className="rounded border px-3 py-1 text-sm">My files</a>  
 <a href="/support" className="rounded border px-3 py-1 text-sm">Support</a>  
 </div>  
 </DashboardCard>  
 </div>  
  
 <section className="rounded-[var(--radius)] border border-[color:var(--border)] bg-[color:var(--card)] shadow-sm p-4 overflow-x-auto">  
 <div className="flex items-center justify-between mb-2">  
 <h2 className="font-semibold">Recent assigned files</h2>  
 <a href="/files" className="text-sm underline">View all</a>  
 </div>  
 {assigned.length === 0 ? (  
 <p className="text-sm text-[color:var(--muted)]">No files yet.</p>  
 ) : (  
 <table className="w-full text-sm">  
 <thead>  
 <tr className="text-left text-[color:var(--muted)] border-b border-[color:var(--border)]">  
 <th className="py-2 pr-3">Title</th>  
 <th className="py-2 pr-3">Assigned</th>  
 <th className="py-2 pr-3">Actions</th>  
 </tr>  
 </thead>  
 <tbody>  
 {assigned.slice(0,6).map(f => (  
 <tr key={f.id} className="border-b last:border-0 border-[color:var(--border)]">  
 <td className="py-2 pr-3">  
 {f.title}  
 {f.originalName ? <span className="text-[color:var(--muted)]"> · {f.originalName}</span> : null}  
 </td>  
 <td className="py-2 pr-3">{new Date(f.createdAt).toLocaleString()}</td>  
 <td className="py-2 pr-3">  
 {f.url ? (  
 <a href={f.url} target="\_blank" rel="noreferrer" className="rounded border px-3 py-1 hover:bg-black/5">  
 Download  
 </a>  
 ) : (  
 <span className="text-[color:var(--muted)]">No URL</span>  
 )}  
 </td>  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 )}  
 </section>  
 </div>  
 );  
}

## src\app\(user)\files\page.tsx

"use client";  
import { useEffect, useState } from "react";  
  
type FileRow = {  
 id: string;  
 title: string;  
 originalName?: string | null;  
 url?: string | null;  
 mime?: string | null;  
 size?: number | null;  
 createdAt: string;  
};  
  
export default function MyAssignedFilesPage() {  
 const [rows, setRows] = useState<FileRow[]>([]);  
 const [loading, setLoading] = useState(true);  
  
 async function load() {  
 setLoading(true);  
 const r = await fetch("/api/files?scope=assigned", { cache: "no-store" });  
 if (r.ok) setRows((await r.json()).files);  
 setLoading(false);  
 }  
 useEffect(() => { load(); }, []);  
  
 return (  
 <section className="rounded-[var(--radius)] border border-[color:var(--border)] bg-[color:var(--card)] p-4 overflow-x-auto">  
 <div className="flex items-center justify-between mb-2">  
 <h2 className="font-semibold">Files assigned to me</h2>  
 <button onClick={load} className="rounded border border-[color:var(--border)] px-3 py-1 text-sm">Refresh</button>  
 </div>  
 {loading ? (  
 <div className="text-sm text-[color:var(--muted)]">Loading…</div>  
 ) : rows.length === 0 ? (  
 <div className="text-sm text-[color:var(--muted)]">No assigned files yet.</div>  
 ) : (  
 <table className="w-full text-sm">  
 <thead>  
 <tr className="text-left text-[color:var(--muted)] border-b border-[color:var(--border)]">  
 <th className="py-2 pr-3">Title</th>  
 <th className="py-2 pr-3">Uploaded</th>  
 <th className="py-2 pr-3">Actions</th>  
 </tr>  
 </thead>  
 <tbody>  
 {rows.map(f => (  
 <tr key={f.id} className="border-b last:border-0 border-[color:var(--border)]">  
 <td className="py-2 pr-3">  
 {f.title}  
 {f.originalName ? <span className="text-[color:var(--muted)]"> · {f.originalName}</span> : null}  
 </td>  
 <td className="py-2 pr-3">{new Date(f.createdAt).toLocaleString()}</td>  
 <td className="py-2 pr-3">  
 {f.url ? (  
 <a  
 href={f.url}  
 target="\_blank"  
 rel="noreferrer"  
 className="rounded border px-3 py-1 hover:bg-black/5"  
 >  
 Download  
 </a>  
 ) : (  
 <span className="text-[color:var(--muted)]">No file URL</span>  
 )}  
 </td>  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 )}  
 </section>  
 );  
}

## src\app\(user)\layout.tsx

// src/app/(user)/layout.tsx  
import { redirect } from "next/navigation";  
import AppShell from "@/components/AppShell";  
import { currentUser } from "@/lib/auth-helpers";  
  
export default async function UserLayout({  
 children,  
}: {  
 children: React.ReactNode;  
}) {  
 const user = await currentUser();  
 if (!user) {  
 redirect("/login?next=/dashboard");  
 }  
  
 return <AppShell>{children}</AppShell>;  
}

## src\app\api\admin\apikeys\[id]\route.ts

import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { requireRole } from "@/lib/auth-helpers";  
  
type Params = Promise<{ id: string }>;  
  
export async function PATCH(req: Request, ctx: { params: Params }) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const { id } = await ctx.params;  
 const { isActive } = await req.json();  
 const updated = await prisma.apiKey.update({ where: { id }, data: { isActive: !!isActive } });  
 return NextResponse.json({ ok: true, key: { id: updated.id, isActive: updated.isActive } });  
}  
  
export async function DELETE(\_req: Request, ctx: { params: Params }) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const { id } = await ctx.params;  
 await prisma.apiKey.delete({ where: { id } });  
 return NextResponse.json({ ok: true });  
}

## src\app\api\admin\apikeys\route.ts

import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { requireRole } from "@/lib/auth-helpers";  
import bcrypt from "bcrypt";  
  
export async function GET() {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const keys = await prisma.apiKey.findMany({  
 orderBy: { createdAt: "desc" },  
 select: { id: true, label: true, isActive: true, createdAt: true, lastUsedAt: true }  
 });  
 return NextResponse.json({ keys });  
}  
  
export async function POST(req: Request) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const { label } = await req.json().catch(() => ({ label: "Integration Key" }));  
 const plain = cryptoRandom(48);  
 const keyHash = await bcrypt.hash(plain, 10);  
  
 const key = await prisma.apiKey.create({  
 data: { label: label || "Integration Key", keyHash },  
 select: { id: true, label: true, isActive: true, createdAt: true }  
 });  
  
 // IMPORTANT: return the plain key only once to admin UI  
 return NextResponse.json({ key, plain }, { status: 201 });  
}  
  
function cryptoRandom(len = 48) {  
 const alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  
 return Array.from({ length: len }, () => alphabet[Math.floor(Math.random() \* alphabet.length)]).join("");  
}

## src\app\api\admin\audit\route.ts

import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { requireRole } from "@/lib/auth-helpers";  
  
export async function GET(req: Request) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const { searchParams } = new URL(req.url);  
 const limit = Math.min(Number(searchParams.get("limit") ?? 50), 200);  
 const cursor = searchParams.get("cursor") || undefined;  
  
 const items = await prisma.auditLog.findMany({  
 take: limit + 1,  
 ...(cursor ? { cursor: { id: cursor }, skip: 1 } : {}),  
 orderBy: { createdAt: "desc" },  
 select: {  
 id: true,  
 createdAt: true,  
 action: true,  
 target: true,  
 targetId: true,  
 actor: { select: { id: true, email: true, name: true } },  
 meta: true,  
 },  
 });  
  
 let nextCursor: string | null = null;  
 if (items.length > limit) {  
 const next = items.pop();  
 nextCursor = next!.id;  
 }  
  
 return NextResponse.json({ items, nextCursor });  
}

## src\app\api\admin\users\[id]\route.ts

// src/app/api/admin/users/[id]/route.ts  
import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { requireRole } from "@/lib/auth-helpers";  
import { z } from "zod";  
  
type Params = Promise<{ id: string }>;  
  
export async function PATCH(req: Request, context: { params: Params }) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const { id } = await context.params; // 👈 await params  
  
 const UpdateSchema = z.object({  
 name: z.string().min(2).optional(),  
 role: z.enum(["USER", "ADMIN"]).optional(),  
 subscriptionActive: z.boolean().optional(),  
 status: z.enum(["PENDING", "ACTIVE", "SUSPENDED"]).optional(), // 👈 NEW  
});  
  
 const body = await req.json();  
 const data = UpdateSchema.parse(body);  
  
 const user = await prisma.user.update({  
 where: { id },  
 data,  
 select: { id: true, name: true, email: true, role: true, subscriptionActive: true, createdAt: true },  
 });  
  
 return NextResponse.json({ user });  
}  
  
export async function DELETE(\_req: Request, context: { params: Params }) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const { id } = await context.params; // 👈 await params  
 await prisma.user.delete({ where: { id } });  
  
 return NextResponse.json({ ok: true });  
}

## src\app\api\admin\users\route.ts

import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { requireRole } from "@/lib/auth-helpers";  
import { z } from "zod";  
import bcrypt from "bcrypt";  
  
export async function GET(req: Request) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const { searchParams } = new URL(req.url);  
 const status = searchParams.get("status") as "PENDING" | "ACTIVE" | "SUSPENDED" | null;  
  
 const users = await prisma.user.findMany({  
 where: status ? { status } : undefined,  
 orderBy: { createdAt: "desc" },  
 select: { id: true, name: true, email: true, role: true, subscriptionActive: true, status: true, createdAt: true },  
 });  
  
 return NextResponse.json({ users });  
}  
  
  
const CreateSchema = z.object({  
 name: z.string().min(2),  
 email: z.string().email(),  
 password: z.string().min(6),  
 role: z.enum(["USER", "ADMIN"]).default("USER"),  
 subscriptionActive: z.boolean().default(false),  
});  
  
export async function POST(req: Request) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const body = await req.json();  
 const data = CreateSchema.parse(body);  
  
 const exists = await prisma.user.findUnique({ where: { email: data.email } });  
 if (exists) return NextResponse.json({ error: "Email already exists" }, { status: 409 });  
  
 const passwordHash = await bcrypt.hash(data.password, 10);  
  
 const user = await prisma.user.create({  
 data: {  
 name: data.name,  
 email: data.email,  
 passwordHash,  
 role: data.role,  
 subscriptionActive: data.subscriptionActive,  
 },  
 select: { id: true, name: true, email: true, role: true, subscriptionActive: true, createdAt: true },  
 });  
  
 return NextResponse.json({ user }, { status: 201 });  
}

## src\app\api\assignments\route.ts

import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { currentUser, requireRole } from "@/lib/auth-helpers";  
import { z } from "zod";  
  
export async function GET() {  
 const me = await currentUser();  
 if (!me) return NextResponse.json({ error: "unauthenticated" }, { status: 401 });  
  
 const items = await prisma.fileAssignment.findMany({  
 where: { userId: (me as any).id },  
 orderBy: { createdAt: "desc" },  
 select: {  
 id: true, note: true, createdAt: true,  
 file: { select: { id: true, title: true, url: true, originalName: true } },  
 assignedBy: { select: { id: true, name: true, email: true } },  
 },  
 });  
  
 return NextResponse.json({ assignments: items });  
}  
  
const CreateSchema = z.object({  
 fileId: z.string().min(1),  
 userId: z.string().min(1),  
 note: z.string().optional(),  
});  
  
export async function POST(req: Request) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const body = await req.json();  
 const data = CreateSchema.parse(body);  
  
 const created = await prisma.fileAssignment.create({  
 data: {  
 fileId: data.fileId,  
 userId: data.userId,  
 note: data.note,  
 assignedById: (guard.user as any).id,  
 },  
 select: { id: true, createdAt: true },  
 });  
  
 return NextResponse.json({ assignment: created }, { status: 201 });  
}

## src\app\api\auth\[...nextauth]\route.ts

import NextAuth from "next-auth";  
import { authOptions } from "@/lib/auth";  
  
const handler = NextAuth(authOptions);  
export { handler as GET, handler as POST };

## src\app\api\auth\register\route.ts

// src/app/api/auth/register/route.ts  
import { NextResponse } from 'next/server';  
import { prisma } from '@/lib/prisma';  
import bcrypt from 'bcrypt';  
import { z } from 'zod';  
  
const SignupSchema = z.object({  
 name: z.string().min(2, 'Name is too short'),  
 email: z.string().email('Invalid email'),  
 password: z.string().min(6, 'Password must be at least 6 characters'),  
});  
  
export async function POST(req: Request) {  
 try {  
 if (process.env.ALLOW\_SIGNUPS !== 'true') {  
 return NextResponse.json({ error: 'Signups are disabled' }, { status: 403 });  
 }  
  
 const body = await req.json();  
 const parsed = SignupSchema.safeParse(body);  
 if (!parsed.success) {  
 return NextResponse.json({ error: parsed.error.flatten() }, { status: 400 });  
 }  
  
 const { name, email, password } = parsed.data;  
  
 // Check if user exists  
 const existing = await prisma.user.findUnique({ where: { email } });  
 if (existing) {  
 return NextResponse.json({ error: 'Email already in use' }, { status: 409 });  
 }  
  
 // Hash password and create user  
 const hash = await bcrypt.hash(password, 10);  
 const user = await prisma.user.create({  
 data: {  
 name,  
 email,  
 passwordHash: hash,  
 role: 'USER', // requires role enum in your schema (from earlier day)  
 subscriptionActive: false,  
 status: 'PENDING', // per Day 4 requirement  
 },  
 select: { id: true, email: true, name: true },  
 });  
  
 return NextResponse.json({ ok: true, user });  
 } catch (err) {  
 console.error(err);  
 return NextResponse.json({ error: 'Server error' }, { status: 500 });  
 }  
}

## src\app\api\files\route.ts

import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { currentUser } from "@/lib/auth-helpers";  
  
export async function GET(req: Request) {  
 const me = await currentUser();  
 if (!me) return NextResponse.json({ error: "unauthenticated" }, { status: 401 });  
  
 const { searchParams } = new URL(req.url);  
 const scope = (searchParams.get("scope") || "mine") as "mine" | "assigned" | "all";  
 const isAdmin = (me as any).role === "ADMIN";  
 const meId = (me as any).id;  
  
 let where: any = {};  
 if (isAdmin) {  
 if (scope === "mine") where = { uploadedById: meId };  
 else if (scope === "assigned") {  
 const ids = (await prisma.fileAssignment.findMany({  
 where: { userId: meId }, select: { fileId: true }  
 })).map(a => a.fileId);  
 where = { id: { in: ids } };  
 } else { // all  
 where = {};  
 }  
 } else {  
 if (scope === "assigned") {  
 const ids = (await prisma.fileAssignment.findMany({  
 where: { userId: meId }, select: { fileId: true }  
 })).map(a => a.fileId);  
 where = { id: { in: ids } };  
 } else {  
 where = { uploadedById: meId };  
 }  
 }  
  
 const files = await prisma.file.findMany({  
 where,  
 orderBy: { createdAt: "desc" },  
 select: {  
 id: true, title: true, originalName: true, url: true, mime: true, size: true, createdAt: true,  
 uploadedBy: { select: { id: true, name: true, email: true } },  
 // For admin, include assignment recipients  
 ...(isAdmin ? {  
 assignments: {  
 select: { user: { select: { id: true, email: true, name: true } } }  
 }  
 } : {})  
 },  
 });  
  
 return NextResponse.json({ files });  
}  
  
  
// 🚫 No public POST here anymore unless admin explicitly uses it  
export async function POST(req: Request) {  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const body = await req.json().catch(() => ({}));  
 const file = await prisma.file.create({  
 data: {  
 title: body.title ?? "Untitled",  
 url: body.url,  
 originalName: body.originalName,  
 mime: body.mime,  
 size: body.size,  
 uploadedById: (guard.user as any).id,  
 },  
 select: { id: true, title: true, createdAt: true }  
 });  
  
 return NextResponse.json({ file }, { status: 201 });  
}

## src\app\api\integrations\uploads\route.ts

// src/app/api/integrations/uploads/route.ts  
import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { verifyApiKey } from "@/lib/apiKeyAuth";  
import { logAudit } from "@/lib/audit";  
import { z } from "zod";  
import { extractEmailsFromText, resolveAssigneeIdsByEmails } from "@/lib/assignmentRules";  
  
const JsonSchema = z.object({  
 title: z.string().min(1).default("Untitled"),  
 url: z.string().min(1).optional(),  
 originalName: z.string().optional(),  
 mime: z.string().optional(),  
 size: z.number().int().nonnegative().optional(),  
 uploadedByEmail: z.string().email().optional(),  
});  
  
async function getAssignerId(uploadedById?: string | null) {  
 // If the uploader is an ADMIN, use them as assigner; else fallback to first ACTIVE admin.  
 if (uploadedById) {  
 const u = await prisma.user.findUnique({ where: { id: uploadedById }, select: { role: true } });  
 if (u?.role === "ADMIN") return uploadedById;  
 }  
 const admin = await prisma.user.findFirst({ where: { role: "ADMIN", status: "ACTIVE" }, select: { id: true } });  
 return admin?.id ?? null;  
}  
  
export async function POST(req: Request) {  
 const apiKey = req.headers.get("x-api-key");  
 const key = await verifyApiKey(apiKey);  
 if (!key) return NextResponse.json({ error: "invalid\_api\_key" }, { status: 401 });  
  
 // Parse + validate  
 let payload: any = {};  
 try { payload = await req.json(); } catch { payload = {}; }  
 const parsed = JsonSchema.safeParse(payload);  
 if (!parsed.success) {  
 return NextResponse.json({ error: "invalid\_payload", issues: parsed.error.flatten() }, { status: 400 });  
 }  
 const body = parsed.data;  
  
 // Optional attribution (who uploaded)  
 let uploadedById: string | undefined;  
 if (body.uploadedByEmail) {  
 const u = await prisma.user.findUnique({ where: { email: body.uploadedByEmail }, select: { id: true } });  
 uploadedById = u?.id;  
 }  
  
 // 1) Create the File record  
 const file = await prisma.file.create({  
 data: {  
 title: body.title,  
 url: body.url,  
 originalName: body.originalName,  
 mime: body.mime,  
 size: body.size,  
 uploadedById,  
 },  
 select: { id: true, title: true, createdAt: true, uploadedById: true },  
 });  
  
 // 2) Extract target emails (from filename/title/url)  
 const candidates = Array.from(new Set([  
 ...extractEmailsFromText(body.originalName),  
 ...extractEmailsFromText(body.title),  
 ...extractEmailsFromText(body.url),  
 ]));  
  
 const assigneeIds = await resolveAssigneeIdsByEmails(candidates);  
  
 // 3) Determine assigner (admin)  
 const assignedById = await getAssignerId(uploadedById);  
  
 // 4) Create assignments if we found any matching users  
 let assignedCount = 0;  
 if (assignedById && assigneeIds.length > 0) {  
 await prisma.fileAssignment.createMany({  
 data: assigneeIds.map(userId => ({  
 fileId: file.id,  
 userId,  
 assignedById,  
 note: "Auto-assigned via filename email",  
 })),  
 skipDuplicates: true,  
 });  
 assignedCount = assigneeIds.length;  
  
 await logAudit({  
 actorId: assignedById,  
 action: "FILE\_ASSIGNED",  
 targetId: file.id,  
 target: "File",  
 meta: { via: "integration", strategy: "emails\_in\_filename", emails: candidates, matched: assigneeIds.length },  
 });  
 } else {  
 // no matches — log that info (we still succeed the upload)  
 await logAudit({  
 actorId: uploadedById ?? null,  
 action: "FILE\_ASSIGNED",  
 targetId: file.id,  
 target: "File",  
 meta: { via: "integration", strategy: "emails\_in\_filename", emails: candidates, matched: 0, note: "No assignees matched" },  
 });  
 }  
  
 // 5) Audit the upload itself  
 await logAudit({  
 actorId: uploadedById ?? null,  
 action: "FILE\_UPLOADED",  
 targetId: file.id,  
 target: "File",  
 meta: { via: "integration", apiKeyId: key.id, title: body.title, originalName: body.originalName ?? null },  
 });  
  
 return NextResponse.json({ ok: true, file, assignments: assignedCount, matchedEmails: candidates }, { status: 201 });  
}

## src\app\api\me\route.ts

import { NextResponse } from "next/server";  
import { getServerSession } from "next-auth";  
import { authOptions } from "@/lib/auth";  
  
export async function GET() {  
 const session = await getServerSession(authOptions);  
 if (!session) return NextResponse.json({ ok: false }, { status: 401 });  
 return NextResponse.json({ ok: true, user: session.user });  
}

## src\app\api\static\uploads\[...path]\route.ts

import { NextResponse } from "next/server";  
import { createReadStream, stat } from "fs/promises";  
import { createReadStream as fsRead } from "fs";  
import path from "path";  
  
export async function GET(req: Request, { params }: { params: Promise<{ path: string[] }> }) {  
 const { path: parts } = await params;  
 const fullPath = path.join(process.cwd(), "uploads", ...parts);  
 try {  
 const s = await stat(fullPath);  
 if (!s.isFile()) throw new Error("not a file");  
 const stream = fsRead(fullPath);  
 return new Response(stream as any, {  
 headers: { "Content-Type": "application/octet-stream" },  
 });  
 } catch {  
 return NextResponse.json({ error: "not\_found" }, { status: 404 });  
 }  
}

## src\app\api\stats\route.ts

import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { currentUser } from "@/lib/auth-helpers";  
  
export async function GET() {  
 const me = await currentUser();  
 if (!me) return NextResponse.json({ error: "unauthenticated" }, { status: 401 });  
  
 const role = (me as any).role;  
  
 if (role === "ADMIN") {  
 const [users, pending, files] = await Promise.all([  
 prisma.user.count(),  
 prisma.user.count({ where: { status: "PENDING" } }),  
 prisma.file.count(),  
 ]);  
 return NextResponse.json({ users, pending, files });  
 }  
  
 // USER stats  
 const [myFiles, myAssigned] = await Promise.all([  
 prisma.file.count({ where: { uploadedById: (me as any).id } }),  
 prisma.fileAssignment.count({ where: { userId: (me as any).id } }),  
 ]);  
 return NextResponse.json({ myFiles, myAssigned });  
}

## src\app\api\uploads\route.ts

import { NextResponse } from "next/server";  
import { prisma } from "@/lib/prisma";  
import { requireRole } from "@/lib/auth-helpers";  
import { mkdir, writeFile } from "fs/promises";  
import path from "path";  
import { logAudit } from "@/lib/audit";  
import { extractEmailsFromText, resolveAssigneeIdsByEmails } from "@/lib/assignmentRules";  
  
export const runtime = "nodejs"; // needed for file IO  
  
export async function POST(req: Request) {  
 // Admin-only  
 const guard = await requireRole("ADMIN");  
 if (!guard.ok) return NextResponse.json({ error: "unauthorized" }, { status: guard.status });  
  
 const contentType = req.headers.get("content-type") || "";  
 if (!contentType.includes("multipart/form-data")) {  
 return NextResponse.json({ error: "invalid\_content\_type" }, { status: 400 });  
 }  
  
 const form = await req.formData();  
  
 // The <input name="file" type="file" />  
 const file = form.get("file") as File | null;  
 if (!file) return NextResponse.json({ error: "missing\_file" }, { status: 400 });  
  
 // Optional <input name="title" />  
 const titleFromForm = (form.get("title") as string) || "";  
  
 // ---- Save the file to disk ------------------------------------------  
 const buffer = Buffer.from(await file.arrayBuffer());  
 const uploadDir = path.join(process.cwd(), "uploads"); // served via your rewrite/proxy  
 await mkdir(uploadDir, { recursive: true });  
  
 const safeBase = (file.name || "file").replace(/[^\w.\-@]/g, "\_");  
 const filename = `${Date.now()}-${safeBase}`;  
 const fullPath = path.join(uploadDir, filename);  
 await writeFile(fullPath, buffer);  
  
 const publicUrl = `/uploads/${filename}`; // what we store in DB and render in UI  
  
 // ---- Create DB record ------------------------------------------------  
 const record = await prisma.file.create({  
 data: {  
 title: titleFromForm || file.name || "Untitled",  
 originalName: file.name,  
 mime: file.type,  
 size: buffer.length,  
 uploadedById: (guard.user as any).id,  
 url: publicUrl,  
 },  
 select: { id: true, title: true, originalName: true, url: true, createdAt: true },  
 });  
  
 // ---- Targeted auto-assignment from filename/title/url ----------------  
 // Extract candidate emails from any available text  
 const candidates = Array.from(  
 new Set([  
 ...extractEmailsFromText(record.originalName),  
 ...extractEmailsFromText(record.title),  
 ...extractEmailsFromText(record.url),  
 ])  
 );  
  
 // Match only ACTIVE USERs  
 const assigneeIds = await resolveAssigneeIdsByEmails(candidates);  
  
 // Create assignments (assignedBy = current admin)  
 let assignedCount = 0;  
 if (assigneeIds.length > 0) {  
 await prisma.fileAssignment.createMany({  
 data: assigneeIds.map((userId) => ({  
 fileId: record.id,  
 userId,  
 assignedById: (guard.user as any).id,  
 note: "Auto-assigned via filename email (manual upload)",  
 })),  
 skipDuplicates: true,  
 });  
 assignedCount = assigneeIds.length;  
  
 await logAudit({  
 actorId: (guard.user as any).id,  
 action: "FILE\_ASSIGNED",  
 targetId: record.id,  
 target: "File",  
 meta: {  
 via: "admin\_manual\_upload",  
 strategy: "emails\_in\_filename",  
 emails: candidates,  
 matched: assigneeIds.length,  
 },  
 });  
 } else {  
 // Log that no matches were found (still successful upload)  
 await logAudit({  
 actorId: (guard.user as any).id,  
 action: "FILE\_ASSIGNED",  
 targetId: record.id,  
 target: "File",  
 meta: {  
 via: "admin\_manual\_upload",  
 strategy: "emails\_in\_filename",  
 emails: candidates,  
 matched: 0,  
 note: "No assignees matched",  
 },  
 });  
 }  
  
 // ---- Audit the upload itself ----------------------------------------  
 await logAudit({  
 actorId: (guard.user as any).id,  
 action: "FILE\_UPLOADED",  
 targetId: record.id,  
 target: "File",  
 meta: {  
 title: record.title,  
 originalName: record.originalName,  
 size: buffer.length,  
 mime: file.type,  
 via: "admin\_manual\_upload",  
 url: publicUrl,  
 },  
 });  
  
 return NextResponse.json(  
 { ok: true, file: record, assignments: assignedCount, matchedEmails: candidates },  
 { status: 201 }  
 );  
}

## src\app\globals.css

@import "tailwindcss";  
  
:root{  
 /\* existing tokens … \*/  
 --brand:#25C3F4;  
 --text:#0A0F2C;  
 --muted:#6B7280;  
 --bg:#F9FAFB;  
 --card:#FFFFFF;  
 --border:#E5E7EB;  
 --radius:14px;  
  
 /\* NEW: high-contrast sidebar \*/  
 --sidebar-bg:#0D2435;  
 --sidebar-text:#ECF5F8;  
 --sidebar-muted:#A7BECC;  
 --sidebar-active-bg:rgba(37,195,244,.15);  
 --sidebar-active-text:#FFFFFF;  
 --sidebar-border:rgba(255,255,255,.08);  
}  
  
aside a:hover{ background-color: rgba(255,255,255,.06); }  
  
html,body{background:var(--bg);color:var(--text)}

## src\app\layout.tsx

import type { Metadata } from "next";  
import "./globals.css";  
import Providers from "./providers";  
  
export const metadata: Metadata = { title: "Excel Delivery", description: "..." };  
  
export default function RootLayout({ children }: { children: React.ReactNode }) {  
 return (  
 <html lang="en">  
 <body>  
 <Providers>{children}</Providers>  
 </body>  
 </html>  
 );  
}

## src\app\login\page.tsx

"use client";  
  
import { Suspense, useEffect, useState } from "react";  
import { signIn } from "next-auth/react";  
import { useRouter, useSearchParams } from "next/navigation";  
  
export const dynamic = "force-dynamic"; // avoid static prerender for this page  
  
function QueryEffect({ setError }: { setError: (msg: string) => void }) {  
 const sp = useSearchParams();  
  
 useEffect(() => {  
 const authError = sp.get("error");  
 const notice = sp.get("notice");  
 if (authError === "AccountPending") setError("Your account is awaiting admin approval.");  
 else if (authError === "AccountSuspended") setError("Your account is suspended. Contact support.");  
 else if (notice === "pending") setError("Signup successful. Wait for admin approval before logging in.");  
 }, [sp, setError]);  
  
 return null;  
}  
  
export default function LoginPage() {  
 const [email, setEmail] = useState("");  
 const [password, setPassword] = useState("");  
 const [loading, setLoading] = useState(false);  
 const [error, setError] = useState("");  
 const router = useRouter();  
  
 async function onSubmit(e: React.FormEvent) {  
 e.preventDefault();  
 setError("");  
 setLoading(true);  
 const res = await signIn("credentials", { redirect: false, email, password });  
 setLoading(false);  
 if (res?.error) {  
 if (res.error === "AccountPending") return setError("Your account is awaiting admin approval.");  
 if (res.error === "AccountSuspended") return setError("Your account is suspended. Contact support.");  
 return setError("Wrong email or password");  
 }  
 router.push("/dashboard");  
 }  
  
 return (  
 <main className="min-h-screen flex items-center justify-center p-6">  
 {/\* Wrap the hook-driven side-effect inside Suspense \*/}  
 <Suspense fallback={null}>  
 <QueryEffect setError={setError} />  
 </Suspense>  
  
 <div className="w-full max-w-sm">  
 <form onSubmit={onSubmit} className="space-y-3 border rounded-2xl p-6 bg-white">  
 <h1 className="text-xl font-semibold">Log in</h1>  
  
 <label className="block">  
 <span className="text-sm">Email</span>  
 <input  
 className="w-full border rounded p-2"  
 value={email}  
 onChange={e => setEmail(e.target.value)}  
 required  
 type="email"  
 autoComplete="email"  
 />  
 </label>  
  
 <label className="block">  
 <span className="text-sm">Password</span>  
 <input  
 className="w-full border rounded p-2"  
 value={password}  
 onChange={e => setPassword(e.target.value)}  
 required  
 type="password"  
 autoComplete="current-password"  
 />  
 </label>  
  
 {error && <p className="text-red-600 text-sm">{error}</p>}  
  
 <button  
 disabled={loading}  
 className="w-full rounded bg-black text-white py-2 disabled:opacity-60"  
 >  
 {loading ? "Logging in..." : "Log in"}  
 </button>  
  
 <p className="text-sm mt-2 text-center">  
 New here? <a className="underline" href="/register">Create account</a>  
 </p>  
 </form>  
 </div>  
 </main>  
 );  
}

## src\app\page.tsx

import Image from "next/image";  
  
export default function Home() {  
 return (  
 <div className="font-sans grid grid-rows-[20px\_1fr\_20px] items-center justify-items-center min-h-screen p-8 pb-20 gap-16 sm:p-20">  
 <main className="flex flex-col gap-[32px] row-start-2 items-center sm:items-start">  
 <Image  
 className="dark:invert"  
 src="/next.svg"  
 alt="Next.js logo"  
 width={180}  
 height={38}  
 priority  
 />  
 <ol className="font-mono list-inside list-decimal text-sm/6 text-center sm:text-left">  
 <li className="mb-2 tracking-[-.01em]">  
 Get started by editing{" "}  
 <code className="bg-black/[.05] dark:bg-white/[.06] font-mono font-semibold px-1 py-0.5 rounded">  
 src/app/page.tsx  
 </code>  
 .  
 </li>  
 <li className="tracking-[-.01em]">  
 Save and see your changes instantly.  
 </li>  
 </ol>  
  
 <div className="flex gap-4 items-center flex-col sm:flex-row">  
 <a  
 className="rounded-full border border-solid border-transparent transition-colors flex items-center justify-center bg-foreground text-background gap-2 hover:bg-[#383838] dark:hover:bg-[#ccc] font-medium text-sm sm:text-base h-10 sm:h-12 px-4 sm:px-5 sm:w-auto"  
 href="https://vercel.com/new?utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  
 target="\_blank"  
 rel="noopener noreferrer"  
 >  
 <Image  
 className="dark:invert"  
 src="/vercel.svg"  
 alt="Vercel logomark"  
 width={20}  
 height={20}  
 />  
 Deploy now  
 </a>  
 <a  
 className="rounded-full border border-solid border-black/[.08] dark:border-white/[.145] transition-colors flex items-center justify-center hover:bg-[#f2f2f2] dark:hover:bg-[#1a1a1a] hover:border-transparent font-medium text-sm sm:text-base h-10 sm:h-12 px-4 sm:px-5 w-full sm:w-auto md:w-[158px]"  
 href="https://nextjs.org/docs?utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  
 target="\_blank"  
 rel="noopener noreferrer"  
 >  
 Read our docs  
 </a>  
 </div>  
 </main>  
 <footer className="row-start-3 flex gap-[24px] flex-wrap items-center justify-center">  
 <a  
 className="flex items-center gap-2 hover:underline hover:underline-offset-4"  
 href="https://nextjs.org/learn?utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  
 target="\_blank"  
 rel="noopener noreferrer"  
 >  
 <Image  
 aria-hidden  
 src="/file.svg"  
 alt="File icon"  
 width={16}  
 height={16}  
 />  
 Learn  
 </a>  
 <a  
 className="flex items-center gap-2 hover:underline hover:underline-offset-4"  
 href="https://vercel.com/templates?framework=next.js&utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  
 target="\_blank"  
 rel="noopener noreferrer"  
 >  
 <Image  
 aria-hidden  
 src="/window.svg"  
 alt="Window icon"  
 width={16}  
 height={16}  
 />  
 Examples  
 </a>  
 <a  
 className="flex items-center gap-2 hover:underline hover:underline-offset-4"  
 href="https://nextjs.org?utm\_source=create-next-app&utm\_medium=appdir-template-tw&utm\_campaign=create-next-app"  
 target="\_blank"  
 rel="noopener noreferrer"  
 >  
 <Image  
 aria-hidden  
 src="/globe.svg"  
 alt="Globe icon"  
 width={16}  
 height={16}  
 />  
 Go to nextjs.org →  
 </a>  
 </footer>  
 </div>  
 );  
}

## src\app\providers.tsx

"use client";  
import { SessionProvider } from "next-auth/react";  
  
export default function Providers({ children }: { children: React.ReactNode }) {  
 return <SessionProvider>{children}</SessionProvider>;  
}

## src\app\register\page.tsx

// src/app/register/page.tsx  
'use client';  
  
import { FormEvent, useState } from 'react';  
import { useRouter } from 'next/navigation';  
import { signIn } from 'next-auth/react';  
  
export default function RegisterPage() {  
 const router = useRouter();  
 const [loading, setLoading] = useState(false);  
 const [error, setError] = useState<string | null>(null);  
  
 async function onSubmit(e: FormEvent<HTMLFormElement>) {  
 e.preventDefault();  
 setError(null);  
 setLoading(true);  
  
 const form = new FormData(e.currentTarget);  
 const name = String(form.get('name') || '');  
 const email = String(form.get('email') || '');  
 const password = String(form.get('password') || '');  
  
 try {  
 const res = await fetch('/api/auth/register', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ name, email, password }),  
 });  
  
 const data = await res.json();  
 if (!res.ok) {  
 // Display first form error if present  
 const zerr = (data?.error?.fieldErrors && Object.values(data.error.fieldErrors)[0]?.[0]) as string | undefined;  
 setError(zerr || data?.error || 'Registration failed');  
 setLoading(false);  
 return;  
 }  
  
 // Auto-login after registration  
 //const login = await signIn('credentials', {  
 //redirect: false,  
 //email,  
 //password,  
 //});  
  
 // in src/app/register/page.tsx (after successful POST)  
 setLoading(false);  
 router.replace('/login?notice=pending'); // send them to login with a notice  
  
  
 if (login?.error) {  
 setError(login.error);  
 setLoading(false);  
 return;  
 }  
  
 router.replace('/dashboard');  
 } catch (err) {  
 setError('Something went wrong');  
 setLoading(false);  
 }  
 }  
  
 return (  
 <main className="min-h-[80vh] flex items-center justify-center p-6">  
 <div className="w-full max-w-md rounded-2xl border p-6 shadow-sm">  
 <h1 className="text-2xl font-semibold mb-1">Create account</h1>  
 <p className="text-sm text-gray-500 mb-6">Access your dashboard after signing up.</p>  
  
 {error && (  
 <div className="mb-4 text-sm border rounded-md p-3 bg-red-50 border-red-200 text-red-700">  
 {error}  
 </div>  
 )}  
  
 <form onSubmit={onSubmit} className="space-y-4">  
 <div>  
 <label className="block text-sm mb-1" htmlFor="name">Name</label>  
 <input id="name" name="name" required className="w-full rounded-md border p-2" />  
 </div>  
  
 <div>  
 <label className="block text-sm mb-1" htmlFor="email">Email</label>  
 <input id="email" name="email" type="email" required className="w-full rounded-md border p-2" />  
 </div>  
  
 <div>  
 <label className="block text-sm mb-1" htmlFor="password">Password</label>  
 <input id="password" name="password" type="password" required className="w-full rounded-md border p-2" />  
 </div>  
  
 <button  
 type="submit"  
 disabled={loading}  
 className="w-full rounded-md bg-black text-white py-2 disabled:opacity-60"  
 >  
 {loading ? 'Creating...' : 'Create account'}  
 </button>  
 </form>  
  
 <p className="text-sm mt-4">  
 Already have an account? <a className="underline" href="/login">Sign in</a>  
 </p>  
 </div>  
 </main>  
 );  
}

## src\components\AppShell.tsx

// src/components/AppShell.tsx  
import React from "react";  
import Sidebar from "@/components/Sidebar";  
import { currentUser } from "@/lib/auth-helpers";  
  
type Role = "ADMIN" | "USER";  
  
export default async function AppShell({  
 children,  
}: {  
 children: React.ReactNode;  
}) {  
 // Server-side: fetch the current user once so Sidebar knows the role.  
 const user = await currentUser();  
 const role: Role = (user?.role as Role) || "USER";  
 const name = user?.name ?? null;  
  
 return (  
 <div className="min-h-screen grid grid-cols-[240px\_1fr]">  
 <aside className="bg-[#0D2435] text-white">  
 <Sidebar role={role} name={name} />  
 </aside>  
 <main className="bg-[#F9FAFB] p-6">{children}</main>  
 </div>  
 );  
}

## src\components\DashboardCard.tsx

"use client";  
import { ReactNode } from "react";  
  
export default function DashboardCard({  
 title, value, subtitle, right, children  
}:{  
 title: string;  
 value?: string | number;  
 subtitle?: string;  
 right?: ReactNode;  
 children?: ReactNode;  
}) {  
 return (  
 <section className="rounded-[var(--radius)] border border-[color:var(--border)] bg-[color:var(--card)] shadow-sm p-4">  
 <div className="flex items-start justify-between gap-2">  
 <div>  
 <h3 className="text-xs font-medium text-[color:var(--muted)]">{title}</h3>  
 {value !== undefined && (  
 <div className="text-3xl font-semibold mt-2">{value}</div>  
 )}  
 {subtitle && <p className="text-xs text-[color:var(--muted)] mt-1">{subtitle}</p>}  
 </div>  
 {right}  
 </div>  
 {children}  
 </section>  
 );  
}

## src\components\Sidebar.tsx

// src/components/Sidebar.tsx  
"use client";  
  
import { usePathname, useRouter } from "next/navigation";  
import { signOut } from "next-auth/react";  
import { getMenu } from "@/lib/menu";  
  
type Role = "ADMIN" | "USER";  
  
function isActive(pathname: string, href: string) {  
 if (href === "/") return pathname === "/";  
 if (pathname === href) return true;  
 return pathname.startsWith(href + "/");  
}  
  
export default function Sidebar({  
 role,  
 name,  
}: {  
 role: Role;  
 name?: string | null;  
}) {  
 const pathname = usePathname() || "/";  
 const router = useRouter();  
 const items = getMenu(role);  
  
 return (  
 <aside  
 className="hidden md:flex md:flex-col w-60 shrink-0 border-r"  
 style={{  
 backgroundColor: "var(--sidebar-bg)",  
 color: "var(--sidebar-text)",  
 borderColor: "var(--sidebar-border)",  
 }}  
 >  
 <div className="px-4 py-3 border-b" style={{ borderColor: "var(--sidebar-border)" }}>  
 <div className="text-xs uppercase tracking-wide" style={{ color: "var(--sidebar-muted)" }}>  
 Signed in  
 </div>  
 <div className="font-medium">{name || (role === "ADMIN" ? "Admin" : "User")}</div>  
 </div>  
  
 <nav className="p-2 space-y-1">  
 {items.map((i) => {  
 const active = isActive(pathname, i.href);  
 return (  
 <button  
 key={i.href}  
 type="button"  
 onClick={() => router.push(i.href)} // always navigate  
 className="w-full text-left flex items-center gap-2 rounded-md px-3 py-2 text-sm"  
 style={{  
 backgroundColor: active ? "var(--sidebar-active-bg)" : "transparent",  
 color: active ? "var(--sidebar-active-text)" : "var(--sidebar-text)",  
 }}  
 >  
 <span>{i.label}</span>  
 </button>  
 );  
 })}  
 </nav>  
  
 {/\* NOTE: Removed invalid Tailwind class `border-top` to avoid hydration mismatch \*/}  
 <div className="mt-auto p-3 border-t" style={{ borderColor: "var(--sidebar-border)" }}>  
 <button  
 type="button"  
 onClick={() => signOut({ callbackUrl: "/login" })}  
 className="w-full text-left rounded-md px-3 py-2 text-sm hover:bg-white/10"  
 title="Sign out"  
 >  
 Sign out  
 </button>  
 <div className="mt-2 text-xs" style={{ color: "var(--sidebar-muted)" }}>  
 v0.1  
 </div>  
 </div>  
 </aside>  
 );  
}

## src\components\TrendMini.tsx

"use client";  
import { ResponsiveContainer, LineChart, Line, Tooltip } from "recharts";  
  
export default function TrendMini({ data }:{ data:{x:string|number;y:number}[] }) {  
 return (  
 <div className="mt-2 h-10">  
 <ResponsiveContainer width="100%" height="100%">  
 <LineChart data={data}>  
 <Line type="monotone" dataKey="y" stroke="var(--brand)" strokeWidth={2} dot={false} />  
 <Tooltip contentStyle={{ fontSize: 12 }} />  
 </LineChart>  
 </ResponsiveContainer>  
 </div>  
 );  
}

## src\components\ui\Button.tsx

import { ButtonHTMLAttributes } from "react";  
  
export function Button(props: ButtonHTMLAttributes<HTMLButtonElement>) {  
 const { className = "", ...rest } = props;  
 return (  
 <button  
 {...rest}  
 className={[  
 "inline-flex items-center justify-center rounded-md px-3 py-2 text-sm font-medium",  
 "bg-[color:var(--brand)] text-black hover:bg-[color:var(--brand-600)]",  
 "disabled:opacity-60 disabled:cursor-not-allowed",  
 className,  
 ].join(" ")}  
 />  
 );  
}

## src\lib\apiKeyAuth.ts

import { prisma } from "@/lib/prisma";  
import bcrypt from "bcrypt";  
  
export async function verifyApiKey(plainKey?: string | null) {  
 if (!plainKey) return null;  
 const all = await prisma.apiKey.findMany({ where: { isActive: true }, select: { id: true, keyHash: true } });  
 for (const k of all) {  
 const ok = await bcrypt.compare(plainKey, k.keyHash);  
 if (ok) {  
 await prisma.apiKey.update({ where: { id: k.id }, data: { lastUsedAt: new Date() } });  
 return k;  
 }  
 }  
 return null;  
}

## src\lib\assignmentRules.ts

import { prisma } from "@/lib/prisma";  
  
/\*\* Pull emails from any text (filename, title, URL, etc.) \*/  
export function extractEmailsFromText(text?: string | null): string[] {  
 if (!text) return [];  
 try {  
 if (/^https?:\/\//i.test(text)) {  
 const u = new URL(text);  
 text = decodeURIComponent(u.pathname.split("/").pop() || "");  
 }  
 } catch { /\* ignore URL parse errors \*/ }  
  
 const emailRe = /[A-Z0-9.\_%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}/gi;  
 const found = text.match(emailRe) || [];  
 return Array.from(new Set(found.map((e) => e.trim().toLowerCase())));  
}  
  
/\*\* Resolve ACTIVE non-admin users whose emails match \*/  
export async function resolveAssigneeIdsByEmails(emails: string[]) {  
 if (emails.length === 0) return [];  
 const users = await prisma.user.findMany({  
 where: { email: { in: emails }, role: "USER", status: "ACTIVE" },  
 select: { id: true },  
 });  
 return users.map((u) => u.id);  
}

## src\lib\audit.ts

import { prisma } from "@/lib/prisma";  
  
type LogParams = {  
 actorId?: string | null;  
 action:  
 | "USER\_CREATED"  
 | "SUBSCRIPTION\_TOGGLED"  
 | "FILE\_UPLOADED"  
 | "FILE\_ASSIGNED"  
 | "APIKEY\_CREATED"  
 | "APIKEY\_REVOKED"  
 | "DOWNLOAD\_GRANTED";  
 targetId?: string | null;  
 target?: string | null;  
 meta?: Record<string, any> | null;  
};  
  
export async function logAudit(p: LogParams) {  
 await prisma.auditLog.create({  
 data: {  
 actorId: p.actorId ?? null,  
 action: p.action,  
 targetId: p.targetId ?? null,  
 target: p.target ?? null,  
 meta: p.meta ?? null,  
 },  
 });  
}

## src\lib\auth-helpers.ts

import { getServerSession } from "next-auth";  
import { authOptions } from "@/lib/auth";  
  
export async function currentUser() {  
 const session = await getServerSession(authOptions);  
 return session?.user ?? null; // has .role and .id  
}  
  
export async function requireRole(role: "ADMIN" | "USER") {  
 const user = await currentUser();  
 if (!user) return { ok: false as const, status: 401 as const };  
 if (role === "ADMIN" && (user as any).role !== "ADMIN")  
 return { ok: false as const, status: 403 as const };  
 return { ok: true as const, user };  
}

## src\lib\auth.ts

// src/lib/auth.ts  
import type { NextAuthOptions } from "next-auth";  
import Credentials from "next-auth/providers/credentials";  
import { prisma } from "@/lib/prisma";  
import bcrypt from "bcrypt";  
import { z } from "zod";  
  
export const authOptions: NextAuthOptions = {  
 session: { strategy: "jwt" },  
 providers: [  
 Credentials({  
 name: "Email & Password",  
 credentials: { email: {}, password: {} },  
 async authorize(raw) {  
 const creds = z.object({  
 email: z.string().email(),  
 password: z.string().min(6),  
 }).parse(raw);  
  
 //const user = await prisma.user.findUnique({ where: { email: creds.email } });  
 //if (!user || !user.passwordHash) return null;  
  
 //const ok = await bcrypt.compare(creds.password, user.passwordHash);  
 //if (!ok) return null;  
  
 //return { id: user.id, email: user.email, name: user.name ?? "", role: user.role };  
  
 // ...  
 const user = await prisma.user.findUnique({ where: { email: creds.email } });  
 if (!user || !user.passwordHash) return null;  
  
 const ok = await bcrypt.compare(creds.password, user.passwordHash);  
 if (!ok) return null;  
  
 // 👇 block if not approved  
 if (user.status !== 'ACTIVE') {  
 // Option A: return null (generic error)  
 // return null;  
  
 // Option B: throw a custom error to show message on /login  
 throw new Error(user.status === 'PENDING' ? 'AccountPending' : 'AccountSuspended');  
 }  
  
 return { id: user.id, email: user.email, name: user.name ?? "", role: user.role };  
 },  
 }),  
 ],  
 callbacks: {  
 async jwt({ token, user }) {  
 if (user) {  
 (token as any).id = (user as any).id;  
 (token as any).role = (user as any).role;  
 }  
 return token;  
 },  
 async session({ session, token }) {  
 if (session.user) {  
 (session.user as any).id = (token as any).id;  
 (session.user as any).role = (token as any).role;  
 }  
 return session;  
 },  
 },  
 pages: { signIn: "/login" },  
};

## src\lib\menu.ts

// src/lib/menu.ts  
export type Role = "ADMIN" | "USER";  
export type MenuItem = { label: string; href: string };  
  
export function getMenu(role: Role): MenuItem[] {  
 if (role === "ADMIN") {  
 // Admin Dashboard should go to /dashboard per your requirement  
 return [  
 { label: "Dashboard", href: "/dashboard" },  
 { label: "Users", href: "/admin/users" },  
 { label: "Files", href: "/admin/files" },  
 { label: "Settings", href: "/admin/settings" },  
 { label: "Audit", href: "/admin/audit" },  
 ];  
 }  
  
 // USER  
 return [  
 { label: "Dashboard", href: "/dashboard" },  
 { label: "My Files", href: "/files" },  
 { label: "Assignments", href: "/assignments" },  
 { label: "Support", href: "/support" },  
 ];  
}

## src\lib\prisma.ts

import { PrismaClient } from '@prisma/client';  
  
const globalForPrisma = global as unknown as { prisma: PrismaClient };  
  
export const prisma =  
 globalForPrisma.prisma ||  
 new PrismaClient({  
 log: ['error', 'warn'],  
 });  
  
if (process.env.NODE\_ENV !== 'production') globalForPrisma.prisma = prisma;

## src\types\next-auth.d.ts

// src/types/next-auth.d.ts  
import NextAuth, { DefaultSession } from "next-auth";  
  
declare module "next-auth" {  
 interface Session {  
 user: DefaultSession["user"] & {  
 id?: string;  
 role?: "ADMIN" | "USER";  
 };  
 }  
}  
  
declare module "next-auth/jwt" {  
 interface JWT {  
 id?: string;  
 role?: "ADMIN" | "USER";  
 }  
}