Full Project Architecture & Tech Stack Overview
==========

Included:
- Prisma schema
- Next.js API routes: /api/projects/new.ts, /api/projects/list.ts
- Frontend: app/projects/page.tsx (Next.js + Clerk)
- Notes & configuration snippets (Supabase, Prisma client, Environment vars)

--- Prisma Schema (schema.prisma) ---
```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Project {
  id          String   @id @default(uuid())
  configHash  String   @unique
  stack       String
  version     String
  features    Json?
  zipUrl      String
  pdfUrl      String
  createdAt   DateTime @default(now())
  expiresAt   DateTime
  userLinks   UserProject[]
}

model UserProject {
  id        String   @id @default(uuid())
  userId    String
  project   Project  @relation(fields: [projectId], references: [id], onDelete: Cascade)
  projectId String
  createdAt DateTime @default(now())

  @@index([userId])
}
```

--- Environment Variables (example) ---
```
# .env
DATABASE_URL="postgresql://user:pass@db:5432/dbname"
SUPABASE_URL="https://your-supabase-url.supabase.co"
SUPABASE_KEY="your-service-role-key"
PRISMA_CLIENT_ENGINE_TYPE="binary"
NEXT_PUBLIC_SUPABASE_URL="https://your-supabase-url.supabase.co"
NEXT_PUBLIC_SUPABASE_ANON_KEY="your-anon-key"
CLERK_API_KEY="clerk_api_key"
CLERK_SECRET_KEY="clerk_secret"
EXPIRY_DAYS=7
```

--- /api/projects/new.ts ---
```
import { NextApiRequest, NextApiResponse } from "next";
import { getAuth } from "@clerk/nextjs/server";
import { PrismaClient } from "@prisma/client";
import crypto from "crypto";
import dayjs from "dayjs";
import { createClient } from "@supabase/supabase-js";
```

```
const prisma = new PrismaClient();
const supabase = createClient(process.env.SUPABASE_URL!, process.env.SUPABASE_KEY!);
const DAILY_LIMIT = 3;

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  try {
    const { userId } = getAuth(req);
    if (!userId) return res.status(401).json({ error: "Unauthorized" });

    if (req.method !== "POST") return res.status(405).json({ error: "Method not allowed" });

    const { stack, version, features } = req.body;
    if (!stack || !version) return res.status(400).json({ error: "stack and version required" });

    // rate limit: count today
    const startOfDay = dayjs().startOf("day").toDate();
    const userCount = await prisma.userProject.count({
      where: { userId, createdAt: { gte: startOfDay } }
    });
    if (userCount >= DAILY_LIMIT) return res.status(429).json({ error: "Daily limit reached" });

    const hash = crypto.createHash("sha256")
      .update(JSON.stringify({ stack, version, features }))
      .digest("hex");

    const existing = await prisma.project.findUnique({ where: { configHash: hash } });
    const expiresAt = dayjs().add(Number(process.env.EXPIRY_DAYS || 7), "day").toDate();

    if (existing) {
      // update expiry
      await prisma.project.update({
        where: { id: existing.id },
        data: { expiresAt }
      });
      // link to user if not already linked
      await prisma.userProject.create({
        data: { userId, projectId: existing.id }
      }).catch(()=>{});
      return res.status(200).json({ zipUrl: existing.zipUrl, pdfUrl: existing.pdfUrl, projectId:
existing.id });
    }

    // Trigger generation: call Express backend or internal generator
    // Example: POST to external generator service (pseudo)
    // const generatorRes = await fetch(process.env.GENERATOR_URL + "/generate", { method: "POST",
body: JSON.stringify({stack, version, features}), headers: {"Content-Type":"application/json"}});
    // const { zipBuffer, pdfBuffer, zipName, pdfName } = await generatorRes.json();

    // For this template, assume generator returned signed URLs and file buffers.
    // Upload to Supabase Storage (example)
    const zipName = `projects/${hash}.zip`;
    const pdfName = `projects/${hash}.pdf`;

    // Placeholder: you would upload actual buffers here.
    // await supabase.storage.from("projects").upload(zipName, zipBuffer, { upsert: true });
    // await supabase.storage.from("projects").upload(pdfName, pdfBuffer, { upsert: true });

    const { data: zipPublic } = supabase.storage.from("projects").getPublicUrl(zipName);
    const { data: pdfPublic } = supabase.storage.from("projects").getPublicUrl(pdfName);
```

```
    const created = await prisma.project.create({
      data: {
        configHash: hash,
        stack,
        version,
        features,
        zipUrl: zipPublic.publicUrl || "",
        pdfUrl: pdfPublic.publicUrl || "",
        expiresAt
      }
    });

    await prisma.userProject.create({
      data: { userId, projectId: created.id }
    });

    return res.status(201).json({ zipUrl: created.zipUrl, pdfUrl: created.pdfUrl, projectId:
created.id });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: "Internal server error" });
  } finally {
    // prisma.$disconnect(); // avoid disconnecting in serverless hot reloader
  }
}

--- /api/projects/list.ts ---
import { NextApiRequest, NextApiResponse } from "next";
import { getAuth } from "@clerk/nextjs/server";
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  try {
    const { userId } = getAuth(req);
    if (!userId) return res.status(401).json({ error: "Unauthorized" });

    const links = await prisma.userProject.findMany({
      where: { userId },
      include: { project: true },
      orderBy: { createdAt: "desc" }
    });

    const formatted = links.map(lp => ({
      id: lp.id,
      projectId: lp.projectId,
      zipUrl: lp.project.zipUrl,
      pdfUrl: lp.project.pdfUrl,
      stack: lp.project.stack,
      version: lp.project.version,
      features: lp.project.features,
      createdAt: lp.createdAt,
      expiresAt: lp.project.expiresAt
    }));

    res.status(200).json({ projects: formatted });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Internal server error" });
  }
```

```
}

--- Frontend: app/projects/page.tsx (Next 13+ /app router) ---
"use client";

import React, { useEffect, useState } from "react";
import { useAuth, ClerkLoaded, useUser } from "@clerk/nextjs";
import axios from "axios";

type Project = {
  id: string;
  projectId: string;
  zipUrl: string;
  pdfUrl: string;
  stack: string;
  version: string;
  features: any;
  createdAt: string;
  expiresAt: string;
};

export default function ProjectsPage() {
  const [projects, setProjects] = useState<Project[]>([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState<string | null>(null);

  async function fetchProjects() {
    setLoading(true);
    try {
      const res = await axios.get("/api/projects/list");
      setProjects(res.data.projects || []);
    } catch (e:any) {
      setError(e.message);
    } finally { setLoading(false); }
  }

  useEffect(() => { fetchProjects(); }, []);

  return (
    <div className="p-6">
      <h1 className="text-2xl font-bold mb-4">Your Generated Projects</h1>
      {loading && <p>Loading...</p>}
      {error && <p className="text-red-500">{error}</p>}
      <div className="space-y-4">
        {projects.map(p => (
          <div key={p.id} className="border rounded p-4">
            <div className="flex justify-between items-start">
              <div>
                <h2 className="font-semibold">{p.stack} — v{p.version}</h2>
                <p className="text-sm">Features: {JSON.stringify(p.features)}</p>
                <p className="text-xs text-gray-500">Expires: {new
Date(p.expiresAt).toLocaleString()}</p>
              </div>
              <div className="space-x-2">
                <a href={p.zipUrl} target="_blank" rel="noreferrer" className="underline">Download
ZIP</a>
                <a href={p.pdfUrl} target="_blank" rel="noreferrer" className="underline">Download
PDF</a>
              </div>
            </div>
          </div>
```

```
        ))}
      </div>
    </div>
  );
}
```

--- Notes & Next Steps ---
1. Generator service: implement an Express service that accepts a config, builds a project scaffolding (use templates or Yeoman-like generators), zips the folder, and returns buffers to be uploaded to Supabase Storage.
2. Use Supabase service_role key on server-side only.
3. Cron job: use Supabase Edge Functions, Vercel cron, or a lightweight worker to delete expired projects and remove files from storage.
4. Testing: seed Prisma, test rate-limiting, and confirm Clerk headers in Next.js requests.
5. Security: ensure Supabase keys never reach the client, and consider signed URLs with limited expiry for downloads.

End of document.