# Overview

A full‑stack (server + React client) portfolio site with:

- **Express** server (TypeScript, ESM) that exposes two APIs: POST /api/contact and POST /api/chat.

- **Vite + React + Tailwind** frontend (in client/) rendering your portfolio sections (Hero, About, Experience, Projects, Skills, Education), plus an **AI assistant chat** and **Contact** form.

- **Shared Zod/Drizzle schema** in shared/ for input validation and (optionally) DB types.

- **In‑memory storage** by default; Drizzle + Postgres is scaffolded but not wired in yet.

**Secrets**: There’s a .env file with a GEMINI\_API\_KEY. Treat this as sensitive. Don’t commit it, and rotate it if it’s accidentally shared.

## Quickstart — Run Locally

### Prereqs

* **Node.js 18+** (ESM + Vite).
* (Optional) **PostgreSQL** URL if you later decide to use Drizzle for persistence.

### 1) Configure environment

Create .env (already present) and set:

GEMINI\_API\_KEY=\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
# Optional (only if you plan to use Drizzle + Neon/Postgres):  
# DATABASE\_URL=postgres://<user>:<pass>@<host>/<db>

### 2) Install deps

npm install

### 3) Development mode (HMR + API)

npm run dev

* Starts the **Express** server and attaches **Vite** in middleware mode.
* Opens on **http://localhost:5000** (both API + client are served on the same port).

**Windows** tip: If NODE\_ENV=development inline env var causes issues in cmd, either use PowerShell ($env:NODE\_ENV='development'; npm run dev) or run node dev-win.js.

### 4) Production build + run

npm run build # builds client to dist/public and bundles server to dist/index.js  
npm start # runs Node on the bundled server

### 5) Smoke‑test the APIs

# Contact  
curl -X POST http://localhost:5000/api/contact \  
 -H 'Content-Type: application/json' \  
 -d '{"name":"Test User","email":"t@e.com","subject":"Hello","message":"Nice portfolio!"}'  
  
# Chat (AI)  
curl -X POST http://localhost:5000/api/chat \  
 -H 'Content-Type: application/json' \  
 -d '{"message":"Summarize Yash\'s experience"}'

## Project Layout (curated tree)

MyPortfolio/  
├── client/ # Vite + React app (Tailwind, shadcn/ui, wouter, React Query)  
│ ├── index.html  
│ └── src/  
│ ├── App.tsx # Router + providers (Theme, Tooltip, QueryClient, Toaster)  
│ ├── main.tsx # React root  
│ ├── index.css # Tailwind base + design tokens + animations  
│ ├── pages/  
│ │ ├── home.tsx # Assembles the full one‑page layout  
│ │ └── not-found.tsx  
│ ├── components/  
│ │ ├── navigation.tsx # Sticky nav; theme toggle; mobile menu  
│ │ ├── hero-section.tsx # Landing hero + down‑chevron smooth scroll  
│ │ ├── about-section.tsx # About + highlights  
│ │ ├── experience-section.tsx# Timeline with achievements  
│ │ ├── projects-section.tsx # Cards with badges & links  
│ │ ├── skills-section.tsx # Categorized skill badges  
│ │ ├── education-section.tsx # Degrees/education  
│ │ ├── ai-chat-section.tsx # Chat UI; calls /api/chat  
│ │ ├── contact-section.tsx # Validated form; calls /api/contact  
│ │ ├── footer.tsx  
│ │ ├── theme-provider.tsx # Light/Dark toggling with localStorage  
│ │ └── ui/… # shadcn/ui components (Radix based)  
│ ├── hooks/  
│ │ ├── use-toast.ts # Toast system hook  
│ │ └── use-mobile.tsx # Mobile detection  
│ └── lib/  
│ ├── queryClient.ts # Fetch helper + React Query client  
│ └── utils.ts # cn() (clsx + tailwind-merge)  
│  
├── server/ # Express (TypeScript, ESM)  
│ ├── index.ts # App bootstrap, logging middleware, Vite attach/serveStatic, listen  
│ ├── routes.ts # /api/contact and /api/chat endpoints  
│ ├── storage.ts # In‑memory storage (users, contact, chat logs)  
│ ├── gemini.ts # Google GenAI client + resume context + response generator  
│ └── vite.ts # Dev: create Vite server in middleware; Prod: serve dist/public  
│  
├── shared/  
│ └── schema.ts # Drizzle schema + Zod validators (users, contact\_messages, chat\_messages)  
│  
├── vite.config.ts # Vite config (root=client, build to dist/public, Replit plugins in dev)  
├── tailwind.config.ts # Content globs, CSS vars, animations, plugins  
├── postcss.config.js  
├── tsconfig.json # Paths: @/\* → client/src/\*, @shared/\* → shared/\*  
├── components.json # shadcn/ui config and aliases  
├── package.json # Scripts & deps (server + client in one repo)  
├── dev-win.js # Windows‑friendly dev starter  
└── .env # GEMINI\_API\_KEY (secret)

## End‑to‑End Flow (What calls what)

### Page render

1. **client/src/main.tsx** mounts **App** → wraps providers (**ThemeProvider**, **TooltipProvider**, **React Query**, **Toaster**).
2. **App.tsx** uses **wouter** to route:
   * / → **pages/home.tsx** which composes all sections.

### Navigation & layout

* **navigation.tsx**: sticky navbar with dark/light toggle via **ThemeProvider** (persisted in localStorage). Smooth scroll to section IDs.
* Sections are pure presentational React components with Tailwind utilities, **lucide-react** icons and **shadcn/ui** primitives.

### Contact form → server

1. **contact-section.tsx** gathers name/email/subject/message.
2. On submit it **mutates** via apiRequest('POST', '/api/contact', data) using React Query.
3. **Server**: routes.ts
   * Validates body with **insertContactMessageSchema** (Zod) imported from **shared/schema.ts**.
   * Persists using **storage.createContactMessage** (in‑memory Map).
   * Returns a success message (and logs to console). In a real app you’d send an email here.

### AI chat → Gemini

1. **ai-chat-section.tsx** renders a chat UI and calls POST /api/chat with { message }.
2. **Server**: routes.ts
   * Validates with **chatRequestSchema**.
   * Calls **generateChatResponse** from server/gemini.ts.
3. **server/gemini.ts**
   * Initializes GoogleGenAI with process.env.GEMINI\_API\_KEY.
   * Builds a **resume‑aware system prompt** (YOUR profile) so replies stay on‑brand.
   * Streams/returns text which is saved via storage.createChatMessage and sent back.

### Dev vs Prod serving

* **Dev (npm run dev)**: server/index.ts calls setupVite(app, server) from server/vite.ts.
  + Vite runs in **middleware mode** (HMR, fast refresh). All non‑/api routes fall back to **client index.html**.
* **Prod (npm run build && npm start)**:
  + vite build outputs the client to dist/public.
  + esbuild bundles the server to dist/index.js.
  + serveStatic(app) serves dist/public and falls back to index.html for SPA routes.

### Logging & errors

* server/index.ts wraps res.json to capture/print compact API logs like:
  + POST /api/chat 200 in 84ms :: {"success":true,...} (truncated to keep logs tidy).
* Global error handler converts thrown errors into JSON with proper status.

## Validation & Types

* **Zod** schemas (shared/schema.ts) validate request payloads on the server.
  + insertContactMessageSchema: { name, email, subject, message } with length rules.
  + chatRequestSchema: { message: string(1..1000) }.
* **Drizzle** models in the same file mirror the shapes for a Postgres DB (users, contact\_messages, chat\_messages). The app currently uses **in‑memory** storage; a DB hookup is optional.

## Styling & UI System

* **Tailwind CSS** with custom design tokens (CSS variables) and a few keyframe animations in index.css.
* **shadcn/ui** + **Radix** primitives in client/src/components/ui/\* for consistent, accessible components (Button, Card, Dialog, Tooltip, etc.).
* **ThemeProvider** toggles class="dark" on <html> and persists in localStorage.

## Key Config Files

* **vite.config.ts**
  + root: client, builds to dist/public.
  + Replit dev plugins guarded by environment checks.
  + Static file restrictions via server.fs.deny.
* **tailwind.config.ts**
  + content globs restricted to client/ only.
  + Extends radii, colors (driven by CSS vars), and accordion animations.
* **tsconfig.json**
  + Path aliases: @/\* → client/src/\*, @shared/\* → shared/\*.
  + moduleResolution: "bundler", strict: true, noEmit: true.
* **components.json** (shadcn/ui) defines aliases matching the above.

## Scripts (from package.json)

* dev: NODE\_ENV=development tsx server/index.ts
* build: vite build (client) **and** esbuild server/index.ts (server → dist/index.js)
* start: NODE\_ENV=production node dist/index.js
* check: TypeScript typecheck
* db:push: drizzle-kit push (requires DATABASE\_URL)

## Deployment Notes

* Because the server and client are coupled, deploy to platforms that support a long‑running Node server (e.g., Render, Railway, Fly.io, a VM, Docker on your VPS).
* Set env vars in the platform console:
  + GEMINI\_API\_KEY (required)
  + PORT (platform default or 5000)
  + Optional DATABASE\_URL if/when you wire up Drizzle + Neon/Postgres.
* Build on the platform using the same scripts: npm run build → npm start.

## Security & Hardening Checklist

* ✅ **Input validation** with Zod (already in place)
* 🔒 **Keep secrets out of the repo**: do not commit .env; rotate the current key if it was ever shared.
* 🛡️ **Rate‑limit** /api/chat and /api/contact to avoid abuse.
* 🧵 **CORS** (not needed in same‑origin; add if you split client/server origins).
* 📜 **Audit logging**: Current logs are good for dev; consider structured logs (pino) for prod.
* 🧪 **Add tests** around routes.ts + gemini.ts (e.g., Vitest + supertest).

## Optional: Wire a Real Database (Drizzle + Neon/Postgres)

Currently storage.ts uses in‑memory Maps. To persist data: 1. Provision a Postgres (e.g., **Neon**). Set DATABASE\_URL in .env. 2. Run migrations: bash npm run db:push 3. Replace MemStorage with a database‑backed storage (Drizzle queries matching shared/schema.ts).

## What Each Important File Does (cheat sheet)

* client/src/App.tsx — wires providers + SPA router.
* client/src/pages/home.tsx — builds the entire one‑page layout by composing sections.
* client/src/components/\*-section.tsx — individual, stateless sections with Tailwind classes and icons.
* client/src/components/ai-chat-section.tsx — chat UI; React Query useMutation calls /api/chat; shows typing state; scrolls to latest message.
* client/src/components/contact-section.tsx — form UI; validation & toast feedback; posts to /api/contact.
* client/src/lib/queryClient.ts — apiRequest() fetch wrapper (throws on non‑OK), singleton QueryClient with sensible defaults.
* server/index.ts — Express app bootstrap, request logging middleware, attaches Vite in dev or serves static in prod, and starts HTTP server on PORT.
* server/routes.ts — defines /api/contact and /api/chat; Zod validation; calls storage + Gemini adapter; robust error handling for Zod vs generic errors.
* server/gemini.ts — initializes Google GenAI with API key; holds your resume context; generateChatResponse() returns on‑brand answers.
* server/storage.ts — simple in‑memory repo with strongly typed methods for users, contact messages, and chat messages.
* server/vite.ts — dev: createViteServer (middleware mode) + HTML transform; prod: static serving of dist/public.
* shared/schema.ts — Drizzle table definitions + createInsertSchema() Zod validators; exported types for compile‑time safety.

## Ideas & Next Steps

* **Projects Section**: Fetch real project links from a JSON/YAML or CMS instead of hard‑coding.
* **SEO**: Add <head> meta tags in client/index.html (title/description/OG), sitemap/robots if you deploy to a custom domain.
* **Analytics**: Add a privacy‑friendly tool (Plausible, Umami) to track visits.
* **Accessibility**: Audit landmarks, color contrast, focus states. shadcn/ui gives a solid baseline.
* **CI/CD**: Add GitHub Actions to run npm ci && npm run check && npm run build on pushes.

## FAQ

**Q: Where do I change my content (text, badges, links)?**  
A: In the section components under client/src/components/\*.tsx and client/src/pages/home.tsx.

**Q: Can I split this into separate repos (client/server)?**  
A: Yes—update API URLs in apiRequest(), enable CORS, and deploy separately.

**Q: Why in‑memory storage?**  
A: It keeps the demo simple. Use Drizzle + Neon when you need persistence.

**Q: Can the AI use my real PDF resume?**  
A: Yes—parse or embed the PDF and feed it as context in gemini.ts, or store structured data server‑side and reference it in the prompt.

## Handy Commands

# Dev  
npm run dev  
  
# Typecheck  
npm run check  
  
# Build + start  
npm run build && npm start  
  
# Drizzle (optional, requires DATABASE\_URL)  
npm run db:push

*Document generated from your uploaded MyPortfolio.zip — tailored to your exact codebase.*