RESUMIND — AI Resume Analyzer

> Upload your resume, get an ATS score, and receive detailed AI-powered feedback tailored to a specific job — all without a backend server.

Table of Contents

---

1. Overview

RESUMIND is a fully client-side Single-Page Application (SPA) that lets users:

1. Upload a PDF resume
2. Optionally specify a target company, job title, and job description
3. Receive an instant AI analysis powered by **Claude 3.7 Sonnet**
4. View an ATS compatibility score, category-level breakdowns, and actionable tips

The app has **no custom backend**. All infrastructure — authentication, file storage, key-value persistence, and AI inference — is provided by **Puter.js**, a free in-browser cloud platform. This means there are no API keys to manage, no server to deploy, and no database to provision.

---

2. Live Demo

Platform	URL
Vercel	*your Vercel deployment URL*
Docker	`http://localhost:80` after `docker build`
Local dev	`http://localhost:5173` after `npm run dev`

---

3. Features

Feature	Detail
PDF Upload	Drag-and-drop or click-to-browse; accepts PDF up to 20 MB
PDF → Image Preview	First page is rendered to a PNG via pdfjs-dist v5 in-browser; no server needed
Job-Targeted Analysis	Provide a company name, job title, and job description to get role-specific feedback
ATS Score	0–100 score for how well the resume passes Applicant Tracking Systems
Category Scores	Tone & Style, Content, Structure, Skills — each scored 0–100
Actionable Tips	Every category lists `good` points and `improve` suggestions with detailed explanations
Overall Score Gauge	Animated half-circle gauge showing the overall resume score
Resume Dashboard	Home page lists all previously analysed resumes with their scores
Persistent Storage	Resumes and feedback are stored in Puter KV + Puter FS — survive page reloads
Auth	One-click sign-in / sign-out via Puter (no passwords)
Data Wipe	`/wipe` route lets a user delete all their uploaded files and KV records
Fully Responsive	Works on desktop, tablet, and mobile

4. How It Works

End-to-end flow

```
User opens app
    │
    ▼
Puter.js loads (injected via <script> in root.tsx)
    │
    ▼
usePuterStore (Zustand) initialises:
  • polls window.puter until available
  • calls puter.auth.isSignedIn()
  • hydrates auth state
    │
    ├─ Not signed in → redirect to /auth
    │
    └─ Signed in → show /home
          │
          └─ kv.list('resume:*') → render ResumeCards
```

Upload & analysis flow

```
User fills form (/upload)
    │
    ▼
1. fs.upload([pdfFile])           — PDF stored in Puter cloud FS
    │
    ▼
2. convertPdfToImage(file)        — pdfjs-dist renders page 1 → canvas → PNG
blob
    │
    ▼
3. fs.upload([imageFile])         — PNG thumbnail stored in Puter cloud FS
    │
    ▼
4. kv.set('resume:<uuid>', data)  — metadata saved (paths, job info, empty
feedback)
    │
    ▼
5. ai.feedback(pdfPath, prompt)   — Claude 3.7 Sonnet reads the PDF via Puter AI
    │                                and returns structured JSON feedback
    ▼
6. JSON.parse(feedbackText)       — validate AI response
    │
    ▼
7. kv.set('resume:<uuid>', data)  — feedback saved alongside metadata
    │
    ▼
8. navigate('/resume/<uuid>')     — redirect to result page
```

Resume view flow

```
/resume/:id
    │
    ▼
kv.get('resume:<id>')             — load metadata + feedback
    │
    ├─ fs.read(resumePath)        — fetch PDF blob → object URL
```

```
    │
    └─ fs.read(imagePath)         — fetch PNG blob → object URL
          │
          ▼
    Render: PDF preview (clickable) + Summary + ATS + Details accordion
          │
          ▼
    useEffect cleanup: URL.revokeObjectURL() — no memory leaks
```

---

5. Tech Stack

Layer	Technology	Version	Purpose
UI Framework	React	19	Component rendering
Routing	React Router	7	SPA navigation, file-based routes
Language	TypeScript	5.8	Type safety
Styling	TailwindCSS	4	Utility-first CSS
CSS Animations	tw-animate-css	—	`animate-in`, `fade-in` helpers
State Management	Zustand	5	Global Puter store
Cloud Platform	Puter.js	v2	Auth, FS, KV, AI (no backend needed)
AI Model	Claude 3.7 Sonnet	—	Resume analysis via Puter AI
PDF Rendering	pdfjs-dist	5	In-browser PDF → image conversion
File Drop	react-dropzone	14	Drag-and-drop upload UI
Class Utilities	clsx + tailwind-merge	—	Conditional class merging
Build Tool	Vite	6	Bundling, HMR, code splitting
Package Manager	npm	—	Dependency management

---

6. Project Structure

```
ai_resume_analyzer/
│
├── app/                          # All application source code
│   ├── app.css                   # Global CSS, Tailwind theme, custom utilities
│   ├── root.tsx                  # App shell: <html>, loads Puter.js, error
boundary
│   ├── routes.ts                 # Route table (React Router file-based
```

```
routing)
│   │
│   ├── routes/
│   │   ├── home.tsx              # Dashboard: lists all saved resumes
│   │   ├── upload.tsx            # Upload form + full analysis pipeline
│   │   ├── resume.tsx            # Resume detail: PDF preview + AI feedback
│   │   ├── auth.tsx              # Login / logout page
│   │   └── wipe.tsx              # Dev tool: delete all files + flush KV
│   │
│   ├── components/
│   │   ├── Navbar.tsx            # Top navigation bar
│   │   ├── FileUploader.tsx      # Drag-and-drop PDF uploader (react-dropzone)
│   │   ├── ResumeCard.tsx        # Card shown on the dashboard
│   │   ├── Summary.tsx           # Overall score gauge + category score rows
│   │   ├── ATS.tsx               # ATS score block with tips list
│   │   ├── Detail.tsx            # Accordion with per-category tips
│   │   ├── Accordion.tsx         # Reusable headless accordion (context-based)
│   │   ├── ScoreGauge.tsx        # Animated half-circle SVG gauge
│   │   ├── ScoreCircle.tsx       # Small circular progress ring (used in cards)
│   │   └── ScoreBadge.tsx        # Colour-coded "Strong / Good Start / Needs
Work" badge
│   │
│   ├── lib/
│   │   ├── puter.ts              # Zustand store wrapping the entire Puter.js
API
│   │   ├── pdf2img.ts            # PDF → PNG conversion using pdfjs-dist v5
│   │   └── utils.ts             # cn() class merger, formatSize(),
generateUUID()
│   │
│   └── types/
│       ├── PdfConversionResult.ts  # Return type of convertPdfToImage()
│       └── modules.d.ts           # Module declaration for pdfjs worker ?url
import
│
├── types/                        # Global ambient type declarations
│   ├── index.d.ts                # Resume, Feedback, Job interfaces
│   └── puter.d.ts               # FSItem, PuterUser, KVItem, ChatMessage,
AIResponse
│
├── constants.ts                  # prepareInstructions() — builds the AI prompt
│
├── public/                       # Static assets served at the root URL
│   └── images/, icons/           # SVGs, GIFs, PNGs used throughout the UI
│
├── vercel.json                   # Vercel deployment config (output dir + SPA
rewrites)
├── nginx.conf                    # nginx config for Docker deployment
├── Dockerfile                    # Multi-stage Docker build (node builder +
nginx)
├── react-router.config.ts        # SPA mode (ssr: false)
├── vite.config.ts                # Vite config: base URL, pdfjs code-split
chunk
├── tsconfig.json                 # TypeScript config with ~ path alias → app/
└── package.json                  # Scripts, dependencies
```

---

7. Getting Started

Prerequisites

Tool	Minimum version
Node.js	20
npm	10

No accounts, API keys, or external services are required for local development — Puter.js handles everything in the browser.

Installation

```
git clone https://github.com/MohamedAfrasDev/AI-Resume-Analyzer.git
cd AI-Resume-Analyzer
npm install
```

Development server

```
npm run dev
```

Open http://localhost:5173. The app has full Hot Module Replacement (HMR).

> **First run:** When you open the app, Puter.js will prompt you to sign in (or create a free account). Your Puter account is the only thing required — no other credentials needed.

Production preview (local)

```
npm run build       # Compile to build/client/
npm run preview     # Serve build/client/ at http://localhost:3000
```

Type checking

```
npm run typecheck   # react-router typegen + tsc --noEmit
```

---

8. Environment Variables

All variables are **build-time** (prefixed `VITE_`) and optional. The app works with zero configuration.

Variable	Default	Description
`VITE_BASE_URL`	`/`	The URL subpath the app is served from. Set to `/AI-Resume-Analyzer/` **only** when deploying to GitHub Pages where the repository name is the subpath. Leave unset for Vercel, Docker, and all other platforms.

Set them in a `.env.local` file during development:

```
# .env.local  (never committed — already in .gitignore)
VITE_BASE_URL=/
```

Or pass them at build time:

```
VITE_BASE_URL=/AI-Resume-Analyzer/ npm run build   # GitHub Pages build
```

---

9. Routes

All routes are defined in app/routes.ts and are purely client-side (SPA mode).

`GET /`

File: app/routes/home.tsx

The main dashboard. Requires authentication — unauthenticated users are redirected to `/auth?next=/`.

On mount, it calls `kv.list('resume:*', true)` to load every saved resume record, parses the JSON values, and renders a grid of `ResumeCard` components.

State:

- `resumes: Resume[]` — list loaded from Puter KV
- `loadingResumes: boolean` — shows a scanning GIF while fetching

`GET /auth`

File: app/routes/auth.tsx

A minimal sign-in / sign-out page. Reads `?next=<path>` from the query string and redirects there after a successful sign-in.

Security: The `next` parameter is validated — it must start with `/` and not with `//`, preventing open-redirect attacks where an attacker could craft `/auth?next=https://evil.com`.

When already authenticated, the page immediately redirects to `next`.

`GET /upload`

File: app/routes/upload.tsx

The upload and analysis page. Contains the full pipeline described in §4.

Form fields: | Field | Required | Purpose | |---|---|---| | Company Name | No | Stored with the resume record for display | | Job Title | No | Passed to the AI prompt for role-specific analysis | | Job

Description | No | Passed to the AI prompt for deeper keyword matching | | Resume (PDF) | **Yes** | The file to analyse; max 20 MB |

Processing states (shown as status text while `isProcessing === true`):

1. `Uploading the file...`
2. `Converting to image...`
3. `Uploading the image...`
4. `Preparing data...`
5. `Analyzing...`
6. `Analysis complete, redirecting...`

Any step can fail gracefully — an error message is shown and the form is restored.

`GET /resume/:id`

File: app/routes/resume.tsx

The feedback detail page for a single resume.

- Loads `resume:<id>` from Puter KV
- Fetches the PDF and PNG from Puter FS and creates temporary object URLs
- Renders a two-column layout: **left** = clickable PDF preview, **right** = feedback panel
- Object URLs are revoked on component unmount to prevent memory leaks

Feedback panel sections:

1. `Summary` — overall score gauge + per-category score rows
2. `ATS` — ATS score with a colour-coded card and tip list
3. `Details` — collapsible accordion with full tips for each category

---

`GET /wipe`

File: app/routes/wipe.tsx

A utility/admin page for development and debugging. Requires authentication.

Lists all files in the user's Puter root directory and provides a **"Wipe App Data"** button that:

1. Shows a `window.confirm` dialog (destructive action protection)
2. Deletes every file with `Promise.all` (parallel, fully awaited)
3. Calls `kv.flush()` to erase all KV records
4. Reloads the file list

> ⚠ This permanently deletes **all** uploaded resumes and feedback for the authenticated user. Use with caution.

---

10. Components

`<Navbar />`

File: `app/components/Navbar.tsx`

Simple top bar with a gradient "RESUMIND" brand link (→ `/`) and an "Upload Resume" button (→ `/upload`).

`<FileUploader onFileSelect />`

File: `app/components/FileUploader.tsx`

A drag-and-drop PDF upload area built on `react-dropzone`.

Props: | Prop | Type | Description | |---|---|---| | `onFileSelect` | `(file: File \| null) => void` | Called whenever a file is accepted or removed |

Behaviour:

- Accepts only `application/pdf` with `.pdf` extension
- Maximum file size: **20 MB** (displayed in the UI)
- Single file only (`multiple: false`)
- Shows the selected file name and size once a file is chosen

---

`<ResumeCard resume />`

File: `app/components/ResumeCard.tsx`

A card displayed in the home page grid for each saved resume.

Props: | Prop | Type | Description | |---|---|---| | `resume` | `Resume` | The full resume record from Puter KV |

On mount, reads the PNG thumbnail from Puter FS (`imagePath`) and creates a temporary object URL which is revoked on unmount. Shows the company name, job title, a `ScoreCircle`, and the resume thumbnail.

`<Summary feedback />`

File: `app/components/Summary.tsx`

Top section of the resume review page. Shows:

- A `ScoreGauge` for the overall score
- Four `Category` rows (Tone & Style, Content, Structure, Skills), each with a score number and a `ScoreBadge`

Props: | Prop | Type | Description | |---|---|---| | `feedback` | `Feedback` | Full AI feedback object |

`<ATS score suggestions />`

File: `app/components/ATS.tsx`

Displays the ATS (Applicant Tracking System) compatibility score.

Props: | Prop | Type | Description | |---|---|---| | `score` | `number` | ATS score 0–100 | | `suggestions` | `{ type: "good" \| "improve"; tip: string }[]` | List of ATS tips |

The card background colour changes based on the score:

- **> 69** → green gradient
- **> 49** → yellow gradient
- **≤ 49** → red gradient

Each tip is shown with a check (✓) or warning (⚠) icon.

`<Details feedback />`

File: `app/components/Detail.tsx`

An accordion showing detailed tips for all four scored categories. Each accordion item contains:

1. A **summary grid** — icon + short tip for each item in the category
2. A **detailed list** — colour-coded cards (green = good, yellow = needs improvement) with the tip headline and a full explanation paragraph

Props: | Prop | Type | Description | |---|---|---| | `feedback` | `Feedback` | Full AI feedback object |

`<Accordion>` / `<AccordionItem>` / `<AccordionHeader>` / `<AccordionContent>`

File: `app/components/Accordion.tsx`

A fully headless, context-based accordion system with no external dependencies.

`<Accordion>` props: | Prop | Type | Default | Description | |---|---|---|---| | `defaultOpen` | `string` | — | ID of the item to open by default | | `allowMultiple` | `boolean` | `false` | Allow multiple items open simultaneously | | `className` | `string` | `""` | Extra classes on the wrapper `<div>` |

`<AccordionHeader>` props: | Prop | Type | Default | Description | |---|---|---|---| | `itemId` | `string` | — | Must match the parent `<AccordionItem id>` | | `icon` | `ReactNode` | chevron SVG | Custom icon to replace the default chevron | | `iconPosition` | `"left" \| "right"` | `"right"` | Which side the icon appears on |

Animation: The content panel transitions using `max-h-[2000px]` → `max-h-0` and `opacity-100` → `opacity-0`, producing a smooth expand/collapse without JavaScript measurements.

`<ScoreGauge score />`

File: `app/components/ScoreGauge.tsx`

An animated half-circle SVG gauge displayed prominently in the `Summary` component.

Prop	Type	Default	Description
`score`	`number`	`75`	Score 0–100

Uses `useEffect` + `useRef` to measure the SVG path length, then drives the `strokeDashoffset` with that value so the gradient arc fills proportionally to the score.

`<ScoreCircle score />`

File: `app/components/ScoreCircle.tsx`

A compact circular progress ring used inside `ResumeCard`.

Prop	Type	Default	Description
`score`	`number`	`75`	Score 0–100

Uses React's `useId()` to generate a unique `id` for the SVG `<linearGradient>` — critical when multiple cards are rendered on the same page, since duplicate SVG `id`s cause all but the first gradient to silently break.

`<ScoreBadge score />`

File: `app/components/ScoreBadge.tsx`

A pill badge showing a human-readable score label.

Score range	Colour	Label
> 69	Green	Strong
50–69	Yellow	Good Start
≤ 49	Red	Needs Work

---

11. Core Libraries

`app/lib/puter.ts` — Puter Store

A **Zustand store** that wraps the entire `window.puter` API surface and exposes it with TypeScript types, error handling, and loading states.

Initialisation

`init()` is called once in `root.tsx`'s `useEffect`. It polls `window.puter` every 100 ms (up to 10 seconds) until the Puter.js CDN script has loaded, then calls `checkAuthStatus()` to hydrate auth state.

Store shape

```
usePuterStore() → {
  isLoading: boolean,          // true while auth status is being determined
  error: string | null,        // last error message, if any
  puterReady: boolean,         // true once window.puter is available

  auth: {
    user: PuterUser | null,
    isAuthenticated: boolean,
    signIn():    Promise<void>,
    signOut():   Promise<void>,
    refreshUser(): Promise<void>,
    checkAuthStatus(): Promise<boolean>,
    getUser(): PuterUser | null,
  },

  fs: {
    write(path, data):   Promise<File | undefined>,
    read(path):          Promise<Blob | undefined>,
    readDir(path):       Promise<FSItem[] | undefined>,
    upload(files):       Promise<FSItem | undefined>,
    delete(path):        Promise<void>,
  },

  ai: {
    chat(prompt, imageURL?, testMode?, options?): Promise<AIResponse |
undefined>,
    feedback(path, message): Promise<AIResponse | undefined>,  // used for
resume analysis
    img2txt(image, testMode?): Promise<string | undefined>,
  },

  kv: {
    get(key):                     Promise<string | null | undefined>,
    set(key, value):              Promise<boolean | undefined>,
    delete(key):                  Promise<boolean | undefined>,
    list(pattern, returnValues?): Promise<string[] | KVItem[] | undefined>,
    flush():                      Promise<boolean | undefined>,
  },

  init():       void,
  clearError(): void,
```

```
}
```

The `ai.feedback()` method

This is the main AI integration point. It calls `puter.ai.chat()` with a structured message payload:

```
{
  role: "user",
  content: [
    { type: "file", puter_path: path },   // The uploaded PDF file
    { type: "text", text: message },       // The analysis instructions
  ]
}
```

The model is pinned to `"claude-3-7-sonnet"` and the response is expected to be a JSON string.

`app/lib/pdf2img.ts` — PDF to Image

Converts the first page of a PDF file into a PNG `File` object, entirely in the browser using **pdfjs-dist v5**.

```
convertPdfToImage(file: File): Promise<PdfConversionResult>
```

Returns:

```
{
  imageUrl: string,       // object URL of the generated PNG (call
revokeObjectURL when done)
  file: File | null,      // PNG File ready to upload
  error?: string,         // set only on failure
}
```

Implementation details:

- The pdfjs worker is imported as a Vite `?url` asset (`import workerUrl from 'pdfjs-dist/build/pdf.worker.min.mjs?url'`) — no manual file copying needed and the URL survives content-hash renaming at build time
- Worker is configured lazily on the first call (`workerConfigured` flag)
- Renders at `scale: 1.5` for a good balance of quality and file size
- Returns a detailed error string (never throws) so the upload flow can show a user-facing message

---

`app/lib/utils.ts` — Utilities

```
// Merge Tailwind classes without conflicts (clsx + tailwind-merge)
cn(...inputs: ClassValue[]): string

// Format a byte count as a human-readable string
// e.g. formatSize(20971520) → "20 MB"
```

```
formatSize(bytes: number): string

// Generate a cryptographically random UUID (uses crypto.randomUUID)
generateUUID(): string
```

`constants.ts` — AI Prompt Builder

```
prepareInstructions({ jobTitle, jobDescription }): string
```

Builds the system prompt sent to Claude. It instructs the model to:

1. Act as an expert resume reviewer
2. Incorporate the target job title and description if provided
3. Return **only** a raw JSON object (no markdown, no preamble) matching the `Feedback` schema

The strict JSON-only instruction is essential because the response is passed directly to `JSON.parse()`. If the AI adds any surrounding text, parsing fails and the user sees an error.

---

12. Type Definitions

All global ambient types live in the `types/` directory and are available throughout the project without any imports.

`types/index.d.ts`

```
interface Resume {
  id: string;
  companyName?: string;
  jobTitle?: string;
  imagePath: string;      // Puter FS path of the PNG thumbnail
  resumePath: string;     // Puter FS path of the original PDF
  feedback: Feedback;
}

interface Feedback {
  overallScore: number;
  ATS: {
    score: number;
    tips: { type: "good" | "improve"; tip: string }[];
  };
  toneAndStyle: {
    score: number;
    tips: { type: "good" | "improve"; tip: string; explanation: string }[];
  };
  content: {
    score: number;
    tips: { type: "good" | "improve"; tip: string; explanation: string }[];
  };
  structure: {
    score: number;
    tips: { type: "good" | "improve"; tip: string; explanation: string }[];
  };
  skills: {
```

```
    score: number;
    tips: { type: "good" | "improve"; tip: string; explanation: string }[];
  };
}
```

types/puter.d.ts

```
interface FSItem {
  id: string; uid: string; name: string; path: string;
  is_dir: boolean; size: number | null; writable: boolean;
  created: number; modified: number; accessed: number;
}

interface PuterUser { uuid: string; username: string; }

interface KVItem { key: string; value: string; }

interface ChatMessage {
  role: "user" | "assistant" | "system";
  content: string | ChatMessageContent[];
}

interface AIResponse {
  message: { role: string; content: string | any[] };
  finish_reason: string;
  usage: { type: string; model: string; amount: number; cost: number }[];
}
```

---

13. AI Prompt & Feedback Schema

The prompt in `constants.ts` instructs Claude to return this exact JSON structure. Any deviation causes a parse error.

```
{
  "overallScore": 78,
  "ATS": {
    "score": 82,
    "tips": [
      { "type": "good",    "tip": "Clean single-column layout is ATS-
friendly" },
      { "type": "improve", "tip": "Add more keywords from the job description" }
    ]
  },
  "toneAndStyle": {
    "score": 75,
    "tips": [
      {
        "type": "good",
        "tip": "Action verbs used consistently",
        "explanation": "Every bullet point starts with a strong verb like
'Built', 'Led', or 'Reduced' — exactly what recruiters and ATS systems look
for."
      },
      {
        "type": "improve",
        "tip": "Avoid first-person pronouns",
        "explanation": "Phrases like 'I managed...' should be 'Managed...' —
omitting the pronoun is standard resume convention."
      }
```

```
    ]
  },
  "content":   { "score": 80, "tips": [ ... ] },
  "structure": { "score": 70, "tips": [ ... ] },
  "skills":    { "score": 85, "tips": [ ... ] }
}
```

Score thresholds used consistently across all UI components:

Range	Colour	Badge label
> 69	🟢 Green	Strong
50–69	🟡 Yellow	Good Start
≤ 49	🔴 Red	Needs Work

---

14. State Management

The entire application state is a single **Zustand store** defined in [`app/lib/puter.ts`](). There is no Redux, no Context API (other than the headless Accordion), and no server state library.

Why Zustand?

- The store needs to be consumed in many components that are not in a parent–child relationship (Navbar, upload form, resume viewer, home dashboard all need auth and FS access)
- Puter.js is loaded asynchronously from a CDN script tag, so the store's `init()` pattern (polling for `window.puter`) is a natural fit
- Zustand's minimal boilerplate avoids wrapping the entire tree in a Provider

Auth state lifecycle

```
Page load
  → init() polls window.puter
  → checkAuthStatus() called
  → isLoading: true

puter.auth.isSignedIn() resolves
  → isLoading: false
  → auth.isAuthenticated: true/false

Any component with useEffect([isLoading, auth.isAuthenticated])
  → redirects to /auth if not authenticated
```

Error handling pattern

Every Puter API wrapper in the store:

1. Checks `getPuter()` — sets error + returns if Puter.js isn't loaded
2. Wraps the call in try/catch

3. Calls `setError(msg)` on failure (which also resets `isLoading: false`)

Components can read `error` from the store and call `clearError()` to dismiss.

---

15. Deployment

Option A — Vercel (recommended)

Push to GitHub. Vercel auto-detects the project and uses `vercel.json`:

```
{
  "buildCommand": "npm run build",
  "outputDirectory": "build/client",
  "rewrites": [{ "source": "/(.*)", "destination": "/index.html" }],
  "headers": [{ "source": "/assets/(.*)", "Cache-Control": "public, max-
age=31536000, immutable" }]
}
```

No environment variables are needed for a standard deployment.

```
git push origin main   # Vercel deploys automatically
```

Option B — Docker + nginx

A production-grade multi-stage build:

```
# Stage 1: install deps
FROM node:20-alpine AS deps
# Stage 2: build SPA → build/client/
FROM node:20-alpine AS builder
# Stage 3: serve static files
FROM nginx:1.27-alpine AS production

# Build the image
docker build -t resumind .

# Run on port 8080
docker run -p 8080:80 resumind
```

The app is served at `http://localhost:8080`.

The custom `nginx.conf` provides:

- SPA fallback (`try_files $uri /index.html`)
- `Cache-Control: immutable` for hashed assets
- gzip compression
- Security headers (`X-Frame-Options`, `X-Content-Type-Options`, `Referrer-Policy`)

Option C — GitHub Pages

GitHub Pages serves repositories at `https://<user>.github.io/<repo>/`. You must set the base URL at build time:

```
VITE_BASE_URL=/AI-Resume-Analyzer/ npm run build
```

Then deploy the `build/client/` directory to the `gh-pages` branch. With GitHub Actions:

```
# .github/workflows/deploy.yml
name: Deploy to GitHub Pages
on:
  push:
    branches: [main]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: 20 }
      - run: npm ci
      - run: npm run build
        env:
          VITE_BASE_URL: /AI-Resume-Analyzer/
      - uses: peaceiris/actions-gh-pages@v4
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./build/client
```

Option D — Any static host (Netlify, Cloudflare Pages, S3…)

Build with `npm run build`. Upload `build/client/`. Configure your host to:

1. Serve `index.html` for all 404 paths (SPA routing)
2. Set `Cache-Control: public, max-age=31536000, immutable` on `/assets/*`

---

16. Configuration Reference

`vite.config.ts`

Option	Value	Notes
`base`	`process.env.VITE_BASE_URL ?? "/"`	Configurable via env var
`manualChunks.pdfjs`	`["pdfjs-dist"]`	Splits pdfjs into its own chunk; only loaded on `/upload`
`isSsrBuild` guard	skips manualChunks in SSR pass	pdfjs is external in the SSR bundle

react-router.config.ts

```
export default { ssr: false }   // SPA mode — no server bundle
```

tsconfig.json

Setting	Value	Notes
`paths["~/*"]`	`["./app/*"]`	~ alias maps to `app/`
`strict`	`true`	Full strict TypeScript
`moduleResolution`	`bundler`	Required for Vite
`verbatimModuleSyntax`	`true`	Enforces `import type` for type-only imports

---

17. Security Model

Authentication

Puter.js handles auth via its own OAuth-style popup. The app never sees or stores passwords or tokens — they live entirely within the Puter runtime.

Open redirect prevention

The `/auth` route extracts the `?next=` query parameter and validates it before redirecting:

```
const rawNext = new URLSearchParams(location.search).get('next') || '/';
// Must start with "/" and not "//" (protocol-relative URL)
const next = rawNext.startsWith('/') && !rawNext.startsWith('//') ? rawNext :
'/';
```

An attacker sending `/auth?next=https://evil.com` is neutralised — the user is redirected to `/` instead.

Data isolation

Puter KV and FS are scoped per Puter user account. One user cannot read another user's uploaded resumes or feedback.

AI safety

The app uses `puter.ai.chat()` which proxies requests through Puter's managed AI service. No direct Anthropic API keys are exposed to the client.

Dependency security

All npm dependencies are audited at install time. As of the last release: **0 known vulnerabilities**.

```
npm audit   # should report: found 0 vulnerabilities
```

Destructive action confirmation

The `/wipe` route requires an explicit `window.confirm()` before deleting any data.

---

18. Known Limitations

Limitation	Detail
PDF only	Only PDF files are accepted. DOCX, images, and other formats are not supported.
First page only	The PDF-to-image preview renders only page 1. Multi-page PDFs are still fully analysed by the AI (it receives the whole file), but only page 1 is shown as a visual preview.
AI response variability	Claude may occasionally return non-JSON output (e.g. if a safety filter triggers). The app handles this with a try/catch and shows a friendly error.
No offline support	Puter.js requires internet access. The app does not work offline.
Puter free tier limits	Puter's free tier has storage and AI usage limits. Heavy usage may hit quotas. Check puter.com for current limits.
No resume deletion	Individual resumes cannot be deleted from the dashboard. Use `/wipe` to delete everything, or access Puter's file manager directly at puter.com.
pdfjs worker size	The pdfjs worker file is ~1.2 MB (uncompressed). It is cached after the first visit, so subsequent loads are instant.

Scripts Reference

```
npm run dev          # Start Vite dev server with HMR at localhost:5173
npm run build        # Compile TypeScript + bundle to build/client/
npm run preview      # Serve build/client/ locally at localhost:3000
npm run typecheck    # Generate React Router types + run tsc --noEmit
```

Contributing

1. Fork the repository
2. Create a feature branch: `git checkout -b feat/your-feature`
3. Run `npm run typecheck` and fix any errors before committing

4. Open a Pull Request against `main`

Built with React, React Router, Puter.js, and Claude AI.