Caleigh Goliath

Technical Report: AI Resume Builder

Deployed Application: https://ai-resume-builder-phi-nine.vercel.app/
Source Code Repository: https://github.com/Kgoliath/ai-resume-builder

1. Architecture Decisions and Technology Stack

The AI Resume Builder is a full-stack MERN-like application (though using localStorage instead of a database) designed for scalability, performance, and a seamless user experience. The architecture cleanly separates the responsive React frontend from the robust Node.js/Express backend API, communicating via REST.

Frontend Stack:

- **Framework:** React 18 with Vite was chosen for its exceptional development experience (fast HMR) and optimized production builds.
- **UI Library:** Material-UI (MUI) provided a comprehensive set of pre-styled, accessible, and responsive components, accelerating development and ensuring a professional, consistent look and feel without sacrificing customizability for our templates.
- **State Management:** React's built-in useState and useContext hooks efficiently manage complex form state and application data, avoiding unnecessary complexity from external libraries.
- **Storage:** The browser's localStorage API ensures full user privacy by persisting all resume data directly on the user's device.
- **PDF Generation:** The html2pdf.js library was selected for its ability to perfectly capture the styled HTML preview and convert it to a high-fidelity, print-ready PDF.

Backend Stack:

- **Runtime:** Node.js with Express.js offers a lightweight, efficient, and scalable environment for handling API requests.
- **AI Provider:** The Google Gemini API was integrated for its powerful content generation capabilities and competitive pricing.
- **Security:** API keys are securely managed using dotenv on the backend, preventing exposure to the client and ensuring secure communication with the AI service.

Deployment:

- **Frontend:** Deployed on **Vercel** for its seamless Git integration, global CDN, and serverless functions capability.

- **Backend:** Deployed on **Railway** for its effortless Node.js deployment, integrated environment management, and reliable uptime.

2. API Integration Methodology

The backend serves as a secure gateway to the Google Gemini AI, abstracting the complexity and protecting sensitive API keys.

Implementation Details:

1. **Prompt Engineering:** The core of the AI's effectiveness. Specific, structured prompts were crafted for each resume section:
   - **Work Experience:** "Generate 3 professional bullet points for a [Job Title] focusing on [Skills]. Use strong action verbs and quantify results."
   - **Summary:** "Write a compelling professional summary for a [Job Title] with [Years] years of experience in [Industry]."
   - **ATS Analysis:** "Analyze this resume against the job description '[Job Description]'. List missing keywords and suggest improvements."
2. **Secure Communication:** The frontend makes HTTP POST requests to the backend endpoints (/api/generate, /api/analyze-ats). The backend then adds the secure Gemini API key to the request header before proxying it to Google's servers. This pattern ensures the API key is never exposed client-side.
3. **Data Flow:** User input → Frontend React state → API call to backend → Backend calls Gemini AI → Response is sanitized and sent back to frontend → Frontend updates state and UI.
4. **Error Handling:** Both frontend and backend implement comprehensive try-catch blocks. Users are notified of errors (e.g., "AI service unavailable") with helpful guidance, while detailed errors are logged on the server for debugging.

3. Template Design Approach

The requirement for three distinct, customizable templates was met with a component-based architecture that strictly separates data from presentation.

Design and Implementation:

- **Data-Driven Components:** All resume data is stored in a central React state object. The ResumePreview component acts as a layout manager, passing this data down as props to the individual section components (e.g., PersonalInfo, WorkExperience).
- **Template System:** A templates directory contains three components: ModernTemplate, ClassicTemplate, and ProfessionalTemplate.

Each is a self-contained React component that consumes the resume data props and applies its own unique styling using CSS-in-JS (via MUI's sx prop or dedicated CSS files).

- **Customization:** The template switching logic is simple yet powerful. The selected template name is held in state, and a dynamic renderer conditionally displays the chosen template component, providing users with instant visual feedback.
- **ATS Optimization:** Each template was designed with ATS compatibility as a primary concern. This means using standard section headers (e.g., "Work Experience"), clear hierarchies, avoiding graphics and columns that parsers might misread, and ensuring all text is selectable and real.

4. Performance Optimization Techniques

Multiple strategies were employed to ensure a fast and responsive application:

- **Frontend Bundling:** Vite provides superior performance out of the box, with efficient code splitting and tree-shaking to minimize bundle size.
- **Lazy Loading:** React's lazy() and Suspense are used to defer loading the heavy ResumePreview and template components until the user navigates to the preview tab, drastically improving initial load time.
- **Memoization:** The useMemo and useCallback hooks are utilized to prevent unnecessary re-renders of expensive components, such as the resume preview and ATS analysis results, when parent state changes don't affect them.
- **Backend Optimization:** The Express server is lean and focused. Middleware is used efficiently for logging, CORS, and JSON parsing. API responses are kept concise to minimize network transfer time.

5. Known Limitations and Future Enhancements

Known Limitations:

- **Export Formats:** The application currently supports export exclusively in **PDF** format.
- **Data Portability:** As data is stored in localStorage, it is tied to a specific browser and device. Users cannot access their resumes from another machine.
- **AI Context Window:** Extensive job descriptions or very long resumes may exceed the Gemini model's context window, potentially leading to truncated or less accurate analysis.

Future Enhancements:

1. **Multi-Format Export:** Implement DOCX export using a library like docx and HTML export for web portfolios.
2. **User Accounts:** Integrate a database (e.g., MongoDB) and authentication to allow users to save, manage, and access multiple resumes from anywhere.
3. **Enhanced AI Feedback Loop:** Store user edits and selections to fine-tune future AI suggestions, creating a truly adaptive system.
4. **Template Editor:** Allow users to customize colors and fonts within each template for greater personalization.
5. **Browser Compatibility Testing:** Expand testing to cover a wider range of legacy browsers to ensure full cross-browser compatibility.