Resume Builder

A PROJECT REPORT

Submitted in partial fulfilment of the requirement for the award of the degree of Master of Computer Applications (MCA) by

Divyanshu Singhal

23FS20MCA00072

MANIPAL UNIVERSITY JAIPUR

DEPARTMENT OF COMPUTER APPLICATIONS

MANIPAL UNIVERSITY JAIPUR

JAIPUR-303007

RAJASTHAN, INDIA

MAY 2025

DEPARTMENT OF MASTER OF COMPUTER APPLICATION

MANIPAL UNIVERSITY JAIPUR, JAIPUR – 303 007 (RAJASTHAN), INDIA

16 May 2025

CERTIFICATE

This is to certify that the project titled **Resume Builder** is a record of the bonafide work done by **Divyanshu Singhal** (23FS20MCA00072) submitted in partial fulfilment of the requirements for the award of the Degree of Master of Computer Application in **Manipal University Jaipur, during the academic year 2023-25.**

Dr. Monika Jyotiyana
Project Guide, Department of Computer Applications,
Manipal University Jaipur

Dr. Shilpa Sharma
Head of the Department, Department of Computer Applications,
Manipal University Jaipur

Armus Digital
THE BACKBONE FOR SCALABLE BUSINESS GROWTH

+91 1169320135
info@armusdigital.com
Godda, Jharkhand, India

May 22, 2025

CERTIFICATE

This is to certify that the project entitled Resume Builder was carried out by Divyanshu Singhal (23FS20MCA00072) at Armus Digital, Godda, Jharkhand under my guidance during Feb 2025 to May 2025.

Yash Kumar,
Project Manager
Armus Digital, Godda, Jharkhand

ACKNOWLEDGMENTS

I would like to express my heartfelt gratitude to the following individuals whose support and guidance were instrumental in the successful completion of this project, Resume Builder.

First and foremost, I extend my sincere appreciation to the Director for providing the necessary resources and infrastructure essential for the execution of this project.

I am deeply indebted to **Dr. Shilpa Sharma**, Head, Department of Computer Applications, for her constant encouragement and support throughout this endeavour.

I extend my profound gratitude to my project supervisor **Dr. Amit Hirawat**, Assistant Professor, Department of Computer Applications, for his invaluable guidance, patience, and expertise, which significantly contributed to the development and execution of this project.

Special thanks are due to my department guide **Dr. Monika Jyotiyana**, Assistant Professor, Department of Computer Applications, for her insightful feedback and constructive criticism, which greatly enhanced the quality of the work presented in this project.

I would also like to acknowledge the laboratory in-charge for their assistance in providing access to the required facilities and equipment essential for conducting experiments and simulations.

Finally, I am grateful to all the faculty members whose expertise and feedback were indispensable in shaping the direction and outcomes of this project. Their unwavering support and encouragement have been pivotal in every step of this journey, and I am truly thankful for their invaluable contributions.

ABSTRACT

In today’s fast-paced and competitive job environment, a well-crafted resume is often the key to landing an interview. However, many job seekers face challenges when it comes to formatting, structuring content, and meeting modern hiring standards like Applicant Tracking System (ATS) compatibility. With increasing digitalization in recruitment processes, resumes that are not machine-readable often get filtered out before reaching human recruiters. Recognizing this gap, the main objective of this project was to design and develop an intelligent Resume Builder web application that empowers users to create clean, professional, and ATS-optimized resumes. The tool also aims to support users with guided inputs and AI-based suggestions, making the resume creation process less time-consuming and more effective.

To bring this idea to life, a modern full-stack web development approach was adopted. The frontend of the application was built using Next.js and TypeScript to ensure fast performance, server-side rendering, and code safety. For managing user input and form data, React Hook Form was used along with Zod for real-time validation, improving both reliability and user experience. The backend was powered by Node.js and Express.js, with MongoDB used as the database to store user data and resume entries in a dynamic format. The project also integrated Google’s Gemini 2 AI model, which provided contextual suggestions for resume sections like summaries and cover letters. The entire system was structured into modular components to keep the code maintainable and the development process organized.

Upon implementation, the prototype successfully met the major goals of the project. It delivered a smooth user experience with clear form steps, autosave functionality, and real-time live preview of the resume. Users were able to securely register and log in using Clerk authentication, fill in their personal and professional details, and receive AI-generated suggestions that improved the quality and relevance of their content. The PDF export functionality also worked effectively, generating polished resumes in a layout suitable for both recruiters and ATS algorithms. User testing showed that even those with minimal technical knowledge could easily create a high-quality resume using the tool, which validated the project's practical usefulness.

A range of advanced tools and technologies were employed throughout the project to ensure robustness and scalability. Next.js enabled fast frontend rendering, while Tailwind CSS simplified responsive UI design. Shadcn library provided a ready-made collection of modern, accessible UI components like buttons, modals, and forms, improving development speed. Clerk handled secure authentication flows without the need to build custom logic, and Gemini 2 AI added a layer of intelligence to the application by offering resume content suggestions. Together, these tools contributed to building a smart, modular, and responsive resume builder solution that addresses a real-world need in today’s competitive job market.

The prototype serves as a strong foundation for future enhancements and scalability. Additional features such as multiple resume templates, drag-and-drop section reordering, theme customization, and language support can be introduced to expand user flexibility. The integration of analytics to track resume performance and recruiter engagement could also be explored. These improvements would not only make the tool more versatile but also increase its potential adoption by educational institutions, placement cells, and individual job seekers. With a growing emphasis on digital hiring tools, this project stands well-positioned for real-world implementation and iterative development based on user feedback.

LIST OF FIGURES

Contents

Chapter 1
Introduction

1.1 Overview

A well-written and organized CV can have a big impact on a person's employment prospects in the fast-paced highly competitive job market of today. Since a resume is a potential employer's first impression it is crucial that candidates showcase their skills and knowledge in a way that is both effective and professional. However a lot of students and job searchers have trouble formatting keyword optimization and general presentation of resumes. As part of my MCA major project. We created a Resume Builder web application to solve this issue.
The idea of the Resume Builder came while noticing that most students in colleges either copy templates from the internet or use basic Word documents without knowing what works in today's recruitment process. Most recruiters use ATS systems to scan resumes before they even reach a human. Many resumes are rejected before even reaching a recruiter due to poor formatting or lack of ATS compatibility. To address this, we set out to develop a tool that not only assists users in creating high-quality resumes but also evaluates their compatibility with Applicant Tracking Systems (ATS). The primary focus of this tool is to ensure that users can build resumes optimized to pass through ATS filters, which are widely used by companies in today's hiring processes.
The project is developed as a web-based platform using modern technologies. Frontend is made in Next.js and TypeScript so it is fast responsive and works well on all devices. The backend is built using Express.js and Node.js, while all the data like user info, resume sections, templates, etc. are stored in MongoDB database. Together, these technologies provide a smooth experience for users and make the system scalable and easy to maintain. A unique feature in this tool is the use of AI model Gemini 2 by Google which helps in analyzing the resume content and gives suggestions for improvements. This AI looks for missing keywords, weak points or suggestions for better phrases which make the resume stronger. This kind of feature usually exists in paid tools, but we wanted to implement it in our own project to make it more helpful and modern.

The application of this project can be seen in various areas. Colleges can use this tool during placement drives or training sessions. Job seekers can use it to prepare their applications, and career counsellors can use it for guiding students. Even individuals applying abroad or switching careers can benefit from such a smart tool.
The tool can be used by students, professionals or any individual who wants to quickly build a resume with proper structure and design. The benefits of using this tool are - easy to use, modern templates, real-time feedback from AI, and ATS compatibility which increases the chances of resume selection.

Some of the main advantages of this tool include:

- Easy to use interface that works on laptop.
- Multiple modern and professional templates to choose from.
- Real-time suggestions from AI for improving content.
- ATS-checking to make sure resume gets noticed.
- Secure data handling and fast performance.

Overall, this tool is made to bridge the gap between normal resume makers and smart AI-enhanced platforms. It helps users save time, avoid common mistakes, and create resumes that help in job selection. The goal was to make something useful and practical, not just for the sake of the project, but also something that can really help users in their career journey [1].

1.2 Objectives

The main objectives of the project can be listed as:

- To create a web-based Resume Builder using full-stack technologies.
- To allow users to enter resume data through an easy-to-follow form interface.
- Generating resumes with modern, professional templates that are ATS-compliant.
- To implement AI-based resume evaluation using Google's Gemini 2 model.
- To store and manage user data securely using MongoDB.
- To provide download/export option for resumes in common formats (like PDF).

All these objectives together aim to make the resume-making process easy, fast, and more result-oriented for users. Instead of struggling with formats or outdated templates users can now focus on their content while the system takes care of the rest.

1.3 Organization of Report

The report is arranged in multiple chapters so that every part of the project can be explained properly and step by step. Each chapter talks about a different stage of development from the basic planning and research to the actual coding and testing. This way it becomes easier for the reader to understand how the project was built and what kind of technologies, tools and logic were used at every step.
The content is divided in a way that first gives an overview of the idea and gradually moves into the technical parts like design, implementation and testing. Finally, some observations and suggestions for future improvements are also discussed.

Below is the basic layout of how the report is structured:

- **Chapter 1** gives a general idea of the problem being solved, explains the motivation behind the project, including real-world relevance applications, and benefits. Defines the main objectives and purpose of the project in a clear way. Gives a chapter-wise overview of how the report is organized.
- **Chapter 2** covers the basic concepts and theories related to resume building and ATS systems. Describes all technologies used in the project along with the reason for their selection.
- **Chapter 3** goes into the analysis and system design part. Explains the step-by-step method followed for the project from planning to execution. Contains diagrams like system architecture and block designs to show data flow and structure.
- **Chapter 4** contains the main implementation part where actual coding is explained. Describes different modules like resume editor, AI suggestion system, etc. Shows the working prototype of the project with basic explanation of how it works.
- **Chapter 5** includes testing details, sample outputs, and analysis of project performance and errors.
- **Final Chapter** wraps up the project with a conclusion. Summarizes what was achieved and what challenges were faced during the project. Lists possible future improvements like adding more resume templates, mobile version, etc.

This chapter-wise structure helps keep the report organized and ensures that nothing important is missing. It also makes it easy for someone who is reading the report to refer to any specific part of the project without confusion.

Chapter 2

Background Material

In this chapter, we explored the main theories and tools that helped me build the resume builder project. we understood how ATS works and why resumes need to be optimized for it. we also studied how AI can help users write better content like summaries and cover letters using Google's Gemini 2 model. Technologies like Next.js, Node.js, MongoDB, and Clerk were used to make a full-stack web app. we focused on clean form handling, data validation, and secure login. Some basic security steps were followed to protect user data. The tech stack was selected for its popularity, flexibility, and ease of development.

2.1 Conceptual Overview

Before starting the actual development, we first tried to understand some core ideas around resumes and the job application process in today's world. One major area was the ATS, or Applicant Tracking System. Many companies nowadays don't even read resumes manually at first. Instead, they use software to automatically filter out resumes that don't match certain patterns or keywords. This means that even qualified candidates can be rejected just because their resume didn't follow the expected format or lacked specific terms.

This made me think that if the resume isn't optimized for ATS, then the person may not even get a chance for an interview. So, we decided that my resume builder should focus on ATS optimization. That involves using proper section headings, keyword-friendly job descriptions, and clean, simple formatting that machines can read. My tool generates resumes that follow these rules, which give job seekers a better chance of getting shortlisted.

Another important feature of the tool is AI-powered analysis focused specifically on ATS compatibility. we integrated Google's Gemini 2 model to analyze the uploaded resume and provide targeted suggestions. The AI reviews the content and structure to ensure it aligns with Applicant Tracking System requirements offering improvements related to formatting keyword usage and overall relevance. This helps users optimize their existing resumes for better chances of passing through ATS filters.

Along with resume logic and AI, we also had to understand how frontend and backend communicate in a full-stack web app. We studied REST APIs and how to send/receive data between the browser and the server. We used these concepts to build forms that save resume data in MongoDB and then render it in a preview format for export.

We also explored some design-related principles, like building reusable components in React (Next.js), separating logic cleanly into modules, and making the UI user-friendly for non-technical users. Things like auto-save, clear layout, and easy editing were all done keeping the end user in mind. Even small things like field validation and consistent design helped in building a smooth experience.

So overall, before starting to code, we spent time learning what really makes a resume successful and how tech can make the process easier. These conceptual understandings became the foundation for the whole project and helped me build something more practical and useful [2].

2.2 Technologies Involved

This resume builder project was developed using a modern full-stack web development approach. We combined frontend, backend, database, and AI tools to make the application smooth, interactive, and intelligent. Each technology was chosen based on what it could offer in terms of performance, development speed, and integration capabilities.

2.2.1 Next.js and TypeScript:

On the frontend side, we used Next.js with TypeScript. Next.js is a React-based framework that supports server-side rendering, which makes loading fast and improves SEO. We liked how it handles routing automatically and offers good performance by default. TypeScript helped a lot in catching bugs early since it gives type safety, especially when working with forms and APIs. Together, they made the frontend more reliable and easier to manage [3][4].

2.2.2 Node.js and Express.js:

For the backend, we went with Node.js and Express.js. Node.js allowed me to run JavaScript on the server, and Express made it simple to create REST APIs for communication between frontend and backend. It handles things like routes, request/response handling, and middleware in a clean way. The backend connects with the database and talks to the AI model for resume suggestions [5][6].

2.2.3 MongoDB:

All user data and resume content are stored in MongoDB, which is a NoSQL database. We chose MongoDB because of its flexible structure, which is useful when storing resume sections like education, experience, and skills that may vary in length and format. It's easy to update or retrieve nested data and that makes form processing much smoother[7].

2.2.4 Clerk:

To implement user authentication, we utilized Clerk, which offers pre-built components for login, signup, and session management. Instead of writing custom auth logic with JWTs and handling cookies manually, Clerk gave a faster way to add secure user authentication with less code and better security out of the box [8].

2.2.5 Google Gemini 2 AI Model:

One of the most important parts of this project is the AI integration. We used Google Gemini 2, a powerful AI model, for analyzing uploaded resumes and generating suggestions. For example, when a user uploads a resume, the AI extracts key details, fixes formatting, and even helps generate professional summaries, achievements, and cover letter lines. This made the tool smarter and more helpful for users who don’t know how to write these things properly [9].

2.2.6 React Hook Form and Zod:

To handle user input on forms, we used React Hook Form along with Zod for validation. This combo made it easier to manage form states, show validation messages instantly, and keep the user experience clean. It also reduced code repetition and errors related to forms [10][11].

2.2.7 Tailwind CSS and ShadCN:

For the design part, we used Tailwind CSS for fast, utility-first styling. It helped in making the UI consistent and responsive across screen sizes. Along with that, we used Shadcn, which offers a collection of pre-designed components like buttons, modals, and form elements. It saved a lot of time and helped keep the design modern without writing every component from scratch [12][13].

2.3 Security and Validation

Although this project was built as a part of my academic curriculum, we made a conscious effort to implement basic but essential security practices to ensure that user data is handled responsibly. One of the key security measures was the use of Clerk for managing user authentication and session handling. Clerk is a secure and modern authentication solution that takes care of login, logout, session expiration, and user verification, which helped me avoid writing custom auth logic that could be error-prone or vulnerable to attacks like session hijacking or token theft.All API routes are kept protected so that only authenticated users can access their resume data. Unauthorized users are automatically restricted from viewing or modifying information. Middleware functions are used on backend routes to check the authentication status before executing any sensitive operations like reading or updating database records.

On the frontend, React Hook Form and Zod were used together to perform real-time validations on user input. This helped ensure that only properly formatted data—such as valid emails, correct dates, and required fields—could be submitted. But validation wasn't just limited to the frontend. The backend APIs also include schema-level checks to prevent any incorrect or malicious data from being inserted into the MongoDB database. This dual-layered validation helped maintain data integrity and avoid any form of injection or corrupt records.

Also, no passwords or sensitive information like payment details are stored anywhere manually. Clerk handles the user accounts securely, including OAuth tokens, session management, and encrypted user data. Only minimal non-sensitive user profile information, like name and email (with consent), is stored in the database for resume generation. These practices not only added credibility to the project but also gave me hands-on experience with real-world security concepts used in professional-grade applications [14].

2.4 Why This Stack Was Chosen

The choice of tech stack played a crucial role in the success and overall learning experience of this project. we selected the MERN stack (MongoDB, Express.js, React/Next.js, and Node.js) with TypeScript, which is widely adopted in the industry for building modern, scalable web applications. One of the main reasons for choosing this stack was to align my learning with technologies that are in high demand among tech companies and startups today.Next.js was chosen for its powerful features like server-side rendering, routing, and built-in support for API routes. It also supports TypeScript natively, which helped in writing type-safe and error-resistant code. Combined with React, it provided a flexible and component-driven architecture that made the frontend more maintainable and interactive. Learning and using React Hook Form along with Zod further improved my skills in managing complex forms and ensuring real-time validation without overloading the codebase.

For the backend, Node.js and Express offered a lightweight yet powerful setup to build RESTful APIs quickly. Express middleware helped in structuring the backend logic cleanly, and integrating MongoDB with Mongoose made it easy to manage data models and queries. MongoDB was a natural choice due to its flexible schema design, which fit well with the dynamic nature of resume content.

To enhance the user interface, we used Tailwind CSS and Shadcn, which drastically reduced the time required for building and styling UI components. These tools provided a utility-first approach and access to pre-styled, accessible UI components, which made the frontend both attractive and consistent.

Authentication was managed through Clerk, which simplified user management and provided out-of-the-box secure login, signup, session management, and profile handling features. Finally, integrating Gemini 2 AI allowed me to experiment with real AI services and add meaningful value to the app by suggesting professional summaries and cover letters. Overall, this stack was not only developer-friendly but also taught me practical development patterns that we will carry into future projects.

Chapter 3
Methodology

3.1 Detailed methodology that will be adopted

To complete this project, we followed a modular and iterative approach to break the work into small manageable steps. The first thing we did was gather the overall idea of how a resume builder works, what kind of data it collects from the user, and how ATS systems filter resumes. We also checked a few existing resume tools online to understand what features are usually offered and what can be improved. After planning, we started setting up the frontend part using Next.js with TypeScript. We created separate components for every section like Personal Info, Education, Work Experience, Skills, Projects, etc. These components use React Hook Form so that we can manage the form state easily and apply validations in real time using Zod. It helped to prevent empty or incorrect data entries.

Once the basic form structure was complete, we implemented a live preview panel that displays the resume on the right side as users enter their details. This real-time feedback enhances the user experience by allowing users to instantly see how their resume will appear.

Next, we developed the backend using Node.js and Express.js. RESTful APIs were created to handle saving, editing, and retrieving resume data for preview and download. MongoDB was used as the database due to its flexible schema design, which is well-suited for cases where users may not fill in every section of the resume.

For authentication, we integrated Clerk, which streamlined user login and session management without the need to manually handle passwords. This approach reduced development complexity while ensuring secure handling of user data.

A key highlight of the project was the integration of AI. We utilized Google's Gemini 2 model to provide intelligent suggestions for resume ATS. The AI analyses uploaded resumes to identify formatting issues and suggest improvements. Users can click "ATS Checker" to receive AI suggested formats or suggestions based on their profile details, making it easier to create professional content [15].

Each step of development was done separately and tested properly before moving to the next, so that bugs and issues can be solved early.

3.2 Workflow of the Project

The overall workflow of the resume builder was designed to be simple but efficient. Here's how it generally works:

1. The user logs into the platform using Clerk authentication.
2. After login, the user is taken to the resume form page.
3. Each section of the form is filled by the user, and the data updates the live preview side-by-side.
4. Once the form is complete, the data is saved in MongoDB and a download button is shown to export the resume in PDF format.
5. The user can also update their data anytime or start a new resume from scratch.
6. If the user needs help, they can use "AI Suggest" to get automatic recommendations using ATS Checker.

   This workflow ensures the app is both beginner-friendly and functional for advanced users who need a clean, ATS-compliant resume quickly.

3.2.1 Workflow Diagram

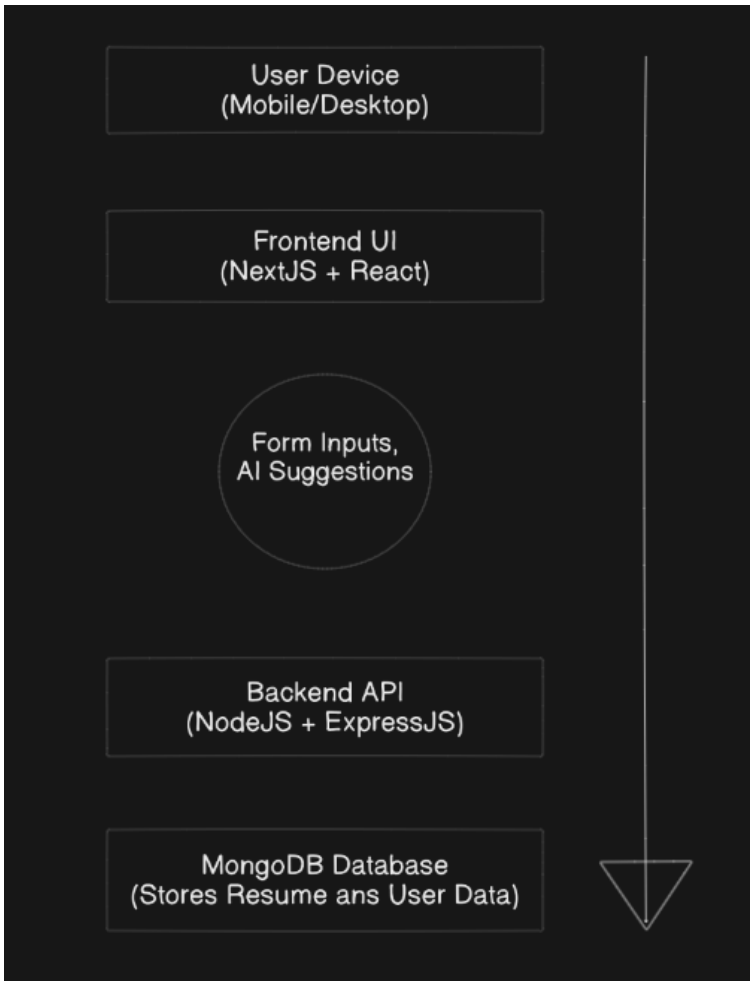

Figure 3.1 Workflow Diagram

3.3 Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation that illustrates the flow of data within a system. It shows how data is input, processed, stored, and output, helping to visualize the system's functional aspects and data movement between processes, external entities, and data stores. DFDs are useful for understanding and analyzing the system's information flow at various levels of detail [16].

3.3.1 Context Level DFD

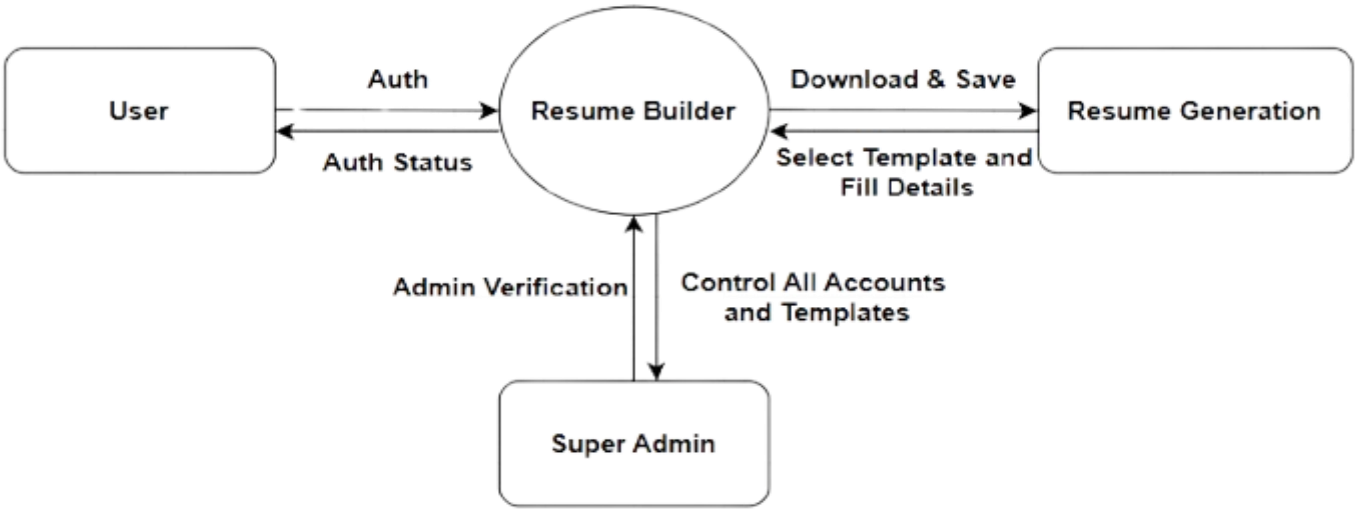

Figure 3.2 Context Level DFD

3.3.2 DFD – 0

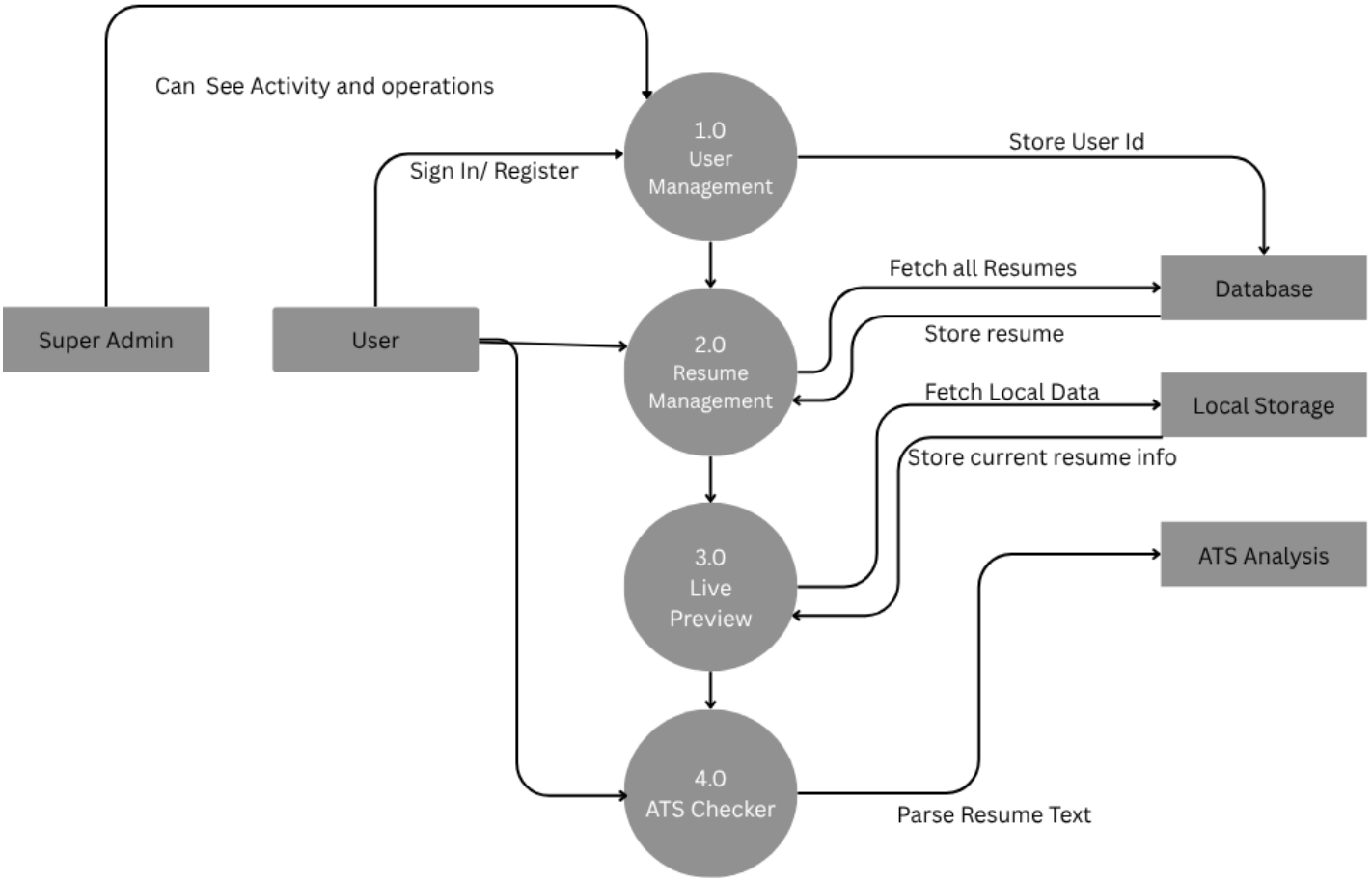

Figure 3.3 DFD-0

3.3.3 DFD – 1

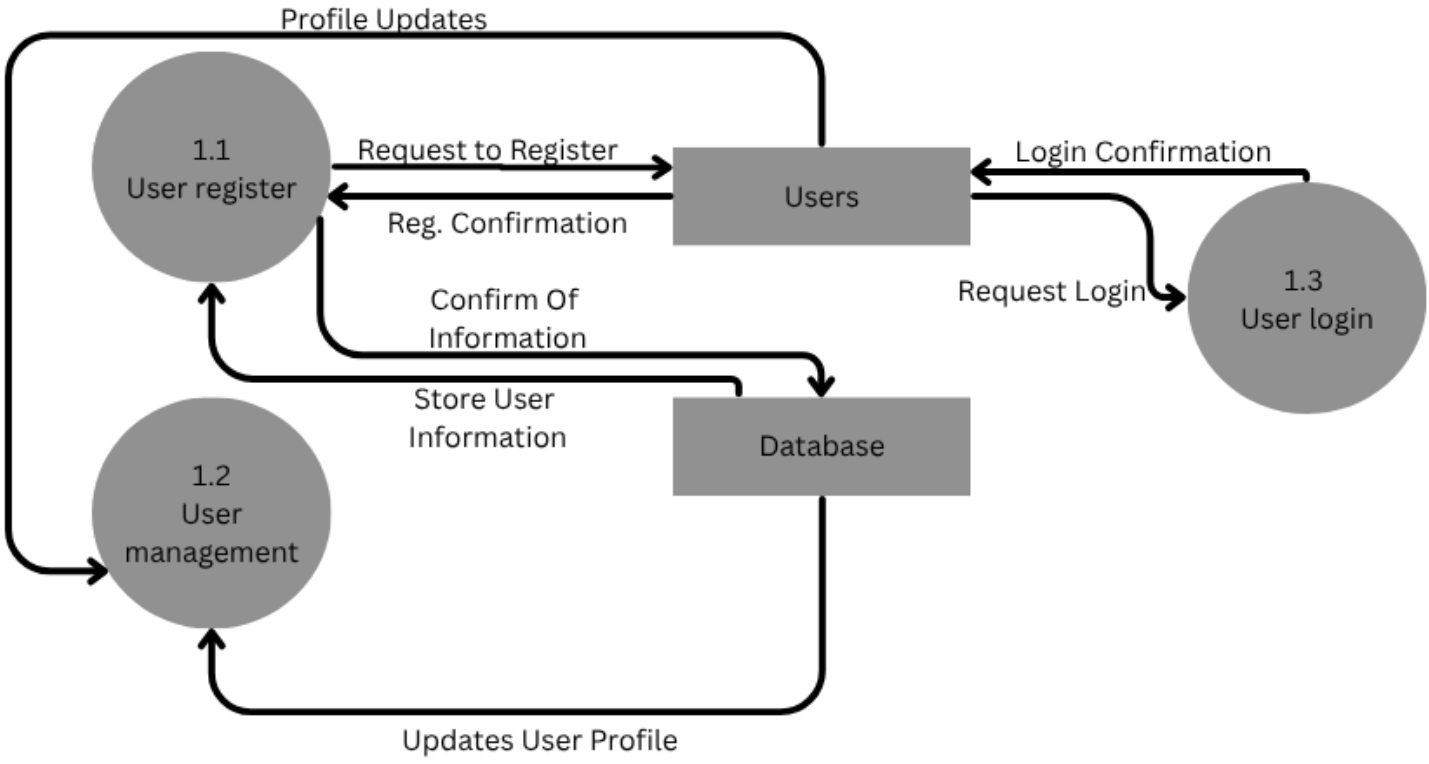

Figure 3.4 DFD-1 1.0

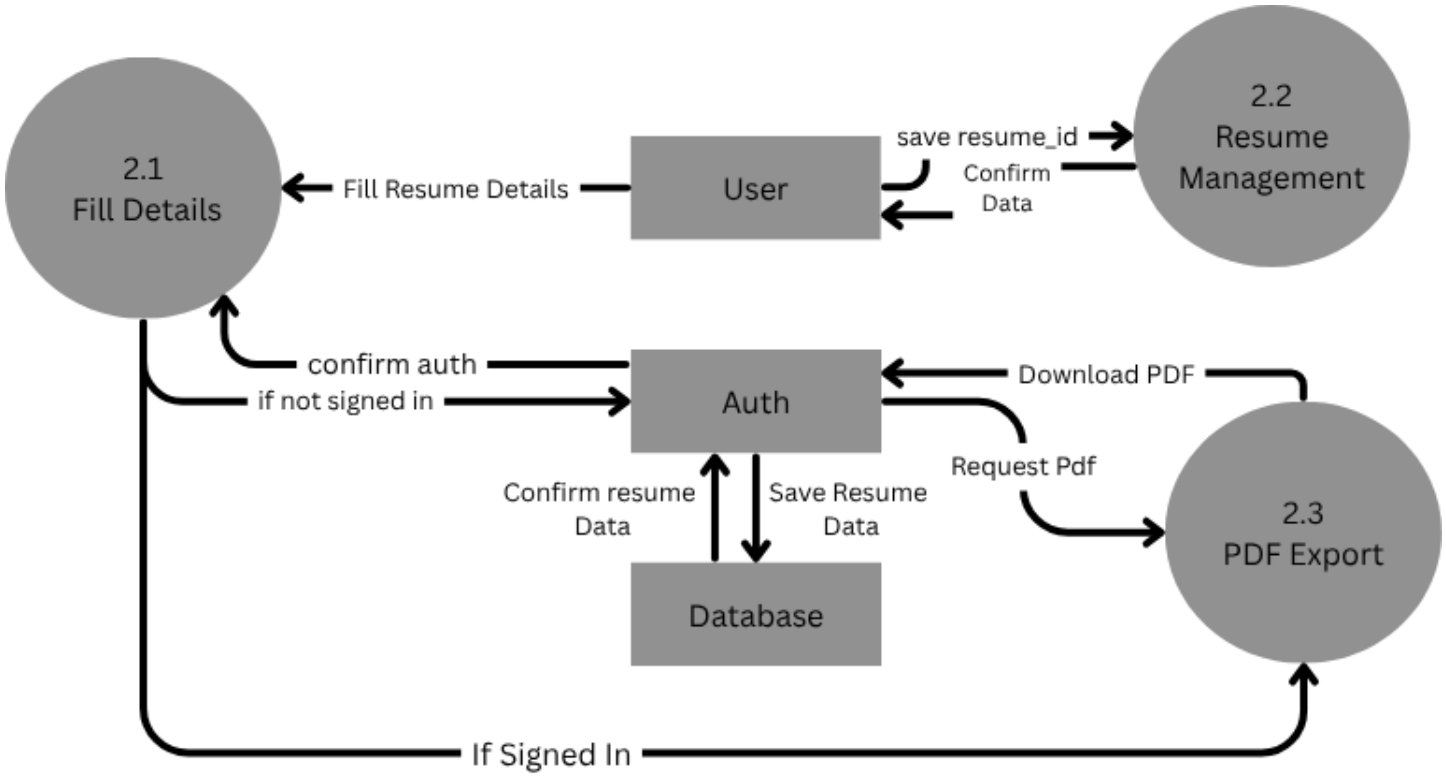

Figure 3.5 DFD-1 2.0

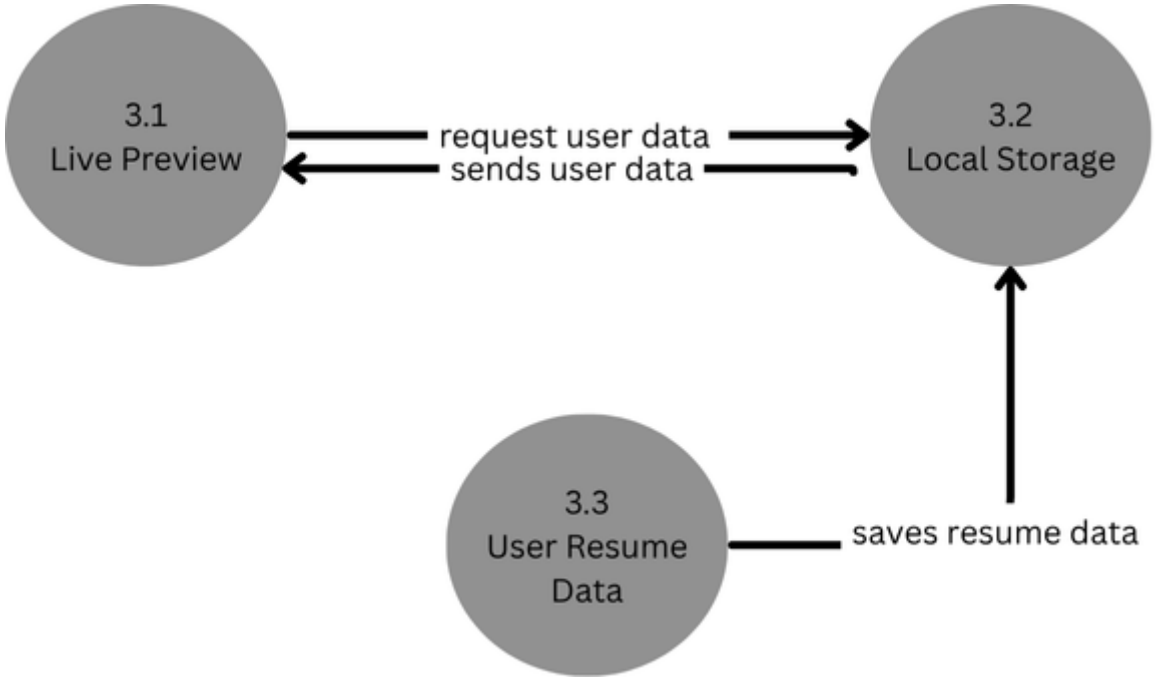

Figure 3.6 DFD-1 3.0

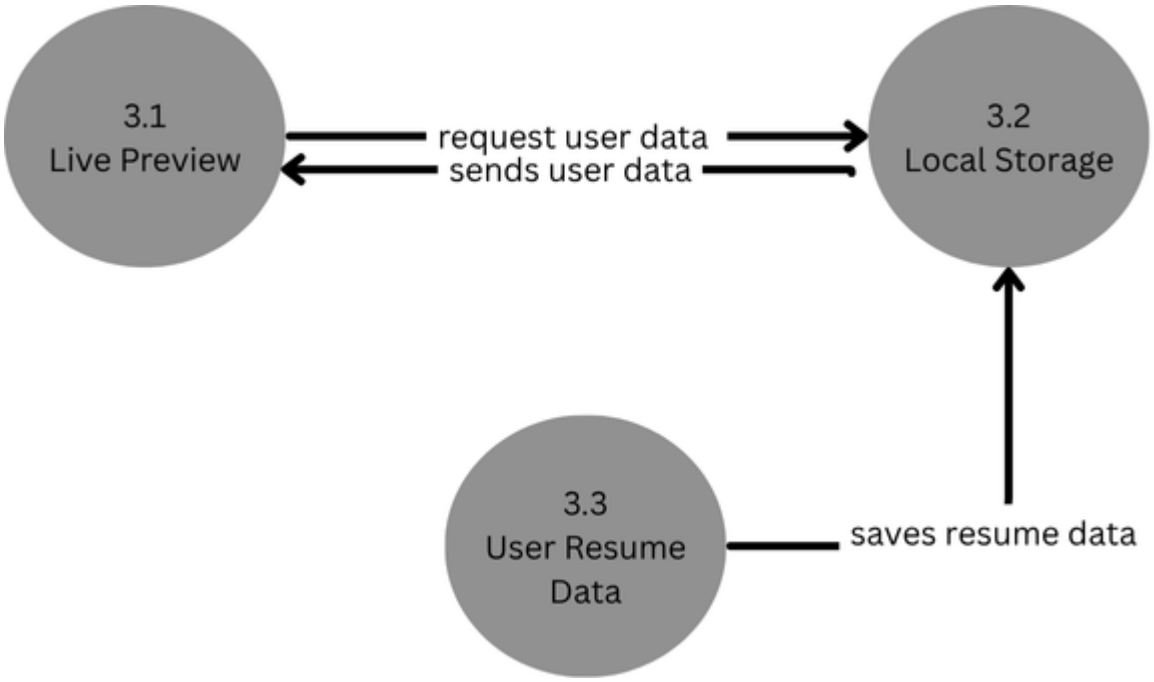

Figure 3.7 DFD-1 4.0

Chapter 4
IMPLEMENTATION

4.1 Modules

To build the Resume Builder system in an organized and maintainable manner, the entire project was divided into modular components. Each module has its own responsibility and interacts with others when needed. This modular approach improves code reusability, debugging, scalability, and clarity for future upgrades. Below is a detailed description of all the core modules:

4.1.1 Authentication Module:

This module is responsible for managing user login, sign-up, and session control. It uses Clerk, a third-party authentication provider that supports secure login with minimal setup. The module allows for email-based login, stores user session securely, and keeps the application protected from unauthorized access. It also manages token expiration, user profile fetching, and auto-redirects for unauthenticated users. This way, developers don't need to write custom authentication flows from scratch.

4.1.2 Form Module:

This is the heart of the system where users enter all their resume-related information. It includes sections like Personal Details, Education, Work Experience, Projects, Skills, Certifications, and Social Links. It is built using React Hook Form for state management and Zod for schema-based validation. Real-time validation gives immediate feedback if something is missing or incorrectly entered. The modular form structure makes it easy to update or add new sections in the future.

4.1.3 Live Preview Module:

As the user types or edits their details, this module reflects those changes in real-time in the resume layout. It mimics how the actual PDF will look and helps users visualize their resume as they build it. This live preview improves usability and saves time by removing the need for frequent manual previews. The layout is dynamically adjusted using React states and components, ensuring a smooth user experience across different screen sizes.

4.1.4 AI Suggestion Module:

To improve the quality of resume content, this module connects to Google Gemini 2 API for generating smart suggestions. The AI analyzes the existing user data and gives polished, formal text suggestions that are keyword-optimized and ATS-friendly. Users can either accept the generated content or modify it as per their need.

4.1.5 Resume Output Module:

Once the user has filled in all the required data, the system compiles the data into a professionally designed template. This module is responsible for rendering that layout and offering a "Download as PDF" feature. It converts the frontend view into a static, clean, and ATS-compatible resume file. The layout respects proper spacing, fonts, and order, giving it a professional look suitable for job applications. It also supports light theming or minor style customization options.

4.1.6 Database Module:

All data provided by users — including their resume fields and account metadata — is stored in MongoDB, a NoSQL database. This module includes API interactions that save, fetch, and update data securely. The use of MongoDB allows flexible and dynamic storage, especially useful since different users may have different projects or experiences. The backend handles all requests using Express.js routes and ensures the data is securely written and read.

4.1.7 Routes Module:

The Routes Module in the backend is responsible for linking client-side actions to server-side processing. All API routes such as POST /resume, GET /user, or PUT /resume/:id are defined here. It also applies middle wares such as authentication checks (via Clerk), error handling, and data validation before forwarding requests to the controller. This keeps the backend clean, structured, and scalable, especially when new routes need to be added later.

4.1.8 Controller Module:

This is where the actual business logic is implemented. Each controller function performs a specific action, such as saving the resume, retrieving user data, or sending requests to the AI API. It separates the data-handling logic from route declarations, which makes the code easier to maintain. For instance, createResumeController() will receive the data from the form, validate it, send it to MongoDB, and return a response, all within one controller.

4.1.9 Data Schema Module:

In this module, the structure of all MongoDB documents is defined using Mongoose models. This includes how each resume section (like Education or Experience) is stored, required fields, and default values. It ensures consistency in the database and avoids issues with missing or malformed data. The schema also supports nested fields and arrays, which is helpful for storing multiple job roles or academic entries per user.

Each of these modules was developed and tested individually before integrating them together. This modular architecture not only helped in parallel development during the project but also makes it easier to debug and extend the app in future versions.

4.2 Prototype

The prototype of the Resume Builder project is a fully functional web application that represents the core features and flow of the final system. The user journey starts with a login page, handled through Clerk authentication. Once a user signs in, they are redirected to a personalized dashboard where they can begin creating a new resume or edit a previously saved one. The user interface is designed to be clean, minimal, and distraction-free, allowing users to focus entirely on building their resume.

After starting a new resume, the user is taken to a multi-step form section. Each step includes specific parts of the resume such as personal information, education, experience, projects, and skills. These sections are implemented using React Hook Form and validated using Zod. The UI elements are styled with Tailwind CSS, and components like buttons, input fields, accordions, and dialogs are taken from the ShadCN UI library, ensuring a consistent and professional appearance. Each section allows users to add multiple entries, such as multiple degrees or work experiences, making the system flexible and dynamic.

One of the core features of this prototype is the Live Preview Panel. This panel, located on the right side of the screen, updates in real-time as the user enters their data. It shows a preview of how the final resume will appear, formatted into a professional template. This gives immediate visual feedback to users, helping them ensure everything looks correct. In addition, the prototype features "Use AI Suggestion" buttons powered by Google’s Gemini 2 API, which generate smart, formal suggestions for resume summaries and cover letters based on the user’s inputs.

All user data is autosaved to the database (MongoDB) as the user fills out the form, reducing the risk of data loss and allowing users to resume their work later. When the resume is completed, the user can click the "Download PDF" button, which triggers a process that renders the resume layout and exports it as a downloadable PDF file. This PDF is designed to be ATS-friendly, ensuring compatibility with modern applicant tracking systems.

Although this is an early-stage prototype, it already includes the most important system components — authentication, structured form handling, real-time preview, AI assistance, PDF generation, and a responsive user interface. It performs smoothly on both desktop and mobile devices, offering a strong base for further improvements. In future versions, enhancements like additional resume templates, drag-and-drop layout editing, and better mobile UX can be considered.

The PDF export feature has been carefully designed to ensure the resume looks professional and neat, matching industry standards. The layout uses standard fonts, spacing, and margin sizes that are known to work well with ATS software. So, resumes built with this tool don’t just look good to humans but also stay readable by automated systems. This was one of the major goals of the prototype. In many job applications, resumes get rejected just because they aren't formatted properly — which led us to implement a solution for it in the tool itself.

Overall, the prototype provides a complete, beginner-friendly experience for anyone who wants to build a resume that’s both attractive and functional. Even though there is still room for improvement and more features in future versions, the current release is functionally complete and delivers a consistent and user-friendly interface. The smart AI suggestions, live editing, and export-ready format all work together to make resume building simpler, smarter, and more efficient.

Chapter 5
Results And Analysis

5.1 Overview of results

After the complete development of the Resume Builder tool, the final system was tested across various parameters such as user experience, performance, AI functionality, and data accuracy. Testing aimed to ensure that the tool delivers the intended features smoothly and gives users a simple yet powerful way to build their resumes. In most cases, the system worked well and gave satisfying results, even during stress testing and real-time multi-user usage. The first thing that stood out during testing was the usability of the platform. Test users were able to navigate the platform easily and understood the layout without needing any guide or walkthrough. The forms for adding education, experience, projects, and skills were easy to fill, and real-time validation helped avoid wrong inputs. Even non-technical users were able to complete their resume without much difficulty, which was a major positive outcome.

One of the most critical parts of the system, the AI integration using Google Gemini 2 was also tested separately. Users uploaded rough resumes or just wrote small bullet points, and the AI provided improved summaries, formatted sections, and content suggestions. The responses from the Gemini model were generally accurate and made resumes more professional. There were a few moments when the suggestions were a bit generic or slow, especially if the input was too lengthy, but in most cases, the AI worked quite reliably.

The PDF export functionality was also checked in detail. It was able to generate resumes with clean layouts that looked polished and followed ATS-friendly formats. The exported files-maintained font consistency, section-wise spacing, and supported special characters without breaking the design. Testers confirmed that the downloaded resumes looked professional enough to be submitted in real job applications.

From the backend and data perspective the tool saved all the data in MongoDB correctly. Whenever a user added or edited data, it was reflected immediately on the frontend via API calls. The communication between frontend (Next.js) and backend (Express.js) worked smoothly without any major failure. There were some early bugs like missing fields and duplicate entries, but those were fixed quickly through debugging.

In conclusion, the overall outcome of the project was a working stable prototype of a modern resume builder. It included smart features like AI suggestions, real-time previews, form validation, resume export and user authentication. While a few minor issues came up during early testing (like slow responses and occasional design glitches) they were successfully resolved. The final result matched almost all the objectives set at the beginning of the project.

5.2 User Testing and Feedback

To properly evaluate the tool in real-life usage, user testing was conducted with a small group of target users. The primary focus was to observe how users interacted with the application and whether the core features were functional and helpful. The testing was informal and mostly done with students freshers and some friends from tech backgrounds. Both black box testing and exploratory testing methods were used.

Based on their input, several enhancements were planned, including refining the live preview responsiveness and improving the AI suggestion accuracy. This iterative testing approach helped ensure the tool aligns closely with user needs and delivers a practical, user-friendly experience before broader deployment [17].

Types of Testing Performed:

- **Black Box Testing**: Users were not given any internal details about the system. They used the application and gave feedback based on what they saw and experienced.
- **Exploratory Testing**: Users were allowed to explore freely. No fixed test cases were given, and their natural flow through the app was noted.
- **Usability Testing**: Observations were made on how easily users navigated through the resume builder, how they filled the form, and how they interacted with the AI suggestions.
- **Responsiveness Testing**: The app was tested on different devices, mainly laptops and smartphones, to check how well it adapted to screen sizes and touch interactions.

Testing Setup:

- **Devices Used**: Windows laptop, Macbook
- **Browsers**: Chrome, Firefox, Edge.
- **Users**: 8 participants – 5 MCA/BCA students, 2 developers, 1 HR intern.
- **Testing Time**: Each session lasted approx. 30–40 minutes.

5.3 Feedback & Observations:

Most of the users found the tool simple and intuitive. They were able to sign up quickly, enter their data, and view the live resume preview without much confusion. The layout was appreciated, especially the clear form sections like personal info education and projects.

The AI suggestion feature was well received. Many users said it helped them write better descriptions for internships and skills. Some even said it gave them ideas they hadn't thought of before.

A few issues were also reported:

- On mobile, the preview section was hard to read due to small fonts.
- The AI responses were sometimes too generic or repetitive if the user refreshed multiple times.
- One tester pointed out that the system didn't allow saving multiple resumes, which would be helpful if someone applied to different jobs.

In terms of learning, the testing helped reveal real-world usability gaps which were not visible during development. For example, one user tried to paste a resume from a Word document but found no easy import option. This shows potential future improvement.

5.4 Performance & Functional Testing

The system was tested for different performance metrics and behavior during real-world usage[18][19]. Below are a few points that were analyzed during functional and performance testing:

- **Responsiveness**: The web app is responsive and works on mobile, tablet, and desktop without layout breakage.
- **Speed**: Loading time for the app was around 1.2 to 1.5 seconds. AI responses took around 3–5 seconds depending on input length.
- **Validation**: All form inputs were validated properly using Zod, and incorrect formats like invalid emails or blank sections were flagged.
- **Data Accuracy**: The MongoDB database stored all information in proper schema format. Fetch and update requests were successful.
- **PDF Export**: Resume export worked fine and generated a downloadable PDF file with accurate formatting.

Manual testing was done rather than automated tests. Still, edge cases were checked, like large input text, empty fields, or saving resumes without login, and the app handled most of them without crashing.

5.5 Limitations and Challenges

Although the final output was functional, the project had a few limitations and difficulties during its development:

- **AI Integration Delays**: Connecting Gemini 2 API and handling the asynchronous nature of AI suggestions took time to figure out. Sometimes, the response took longer and needed retry logic.
- **Form Complexity**: Managing large forms with multiple steps and validations was tricky. Initially, many validation errors were not showing properly, and it took time to make the UX smoother.
- **PDF Styling Issues**: Getting the exact same resume layout in the exported PDF as shown in the live preview was difficult due to rendering differences.
- **Time Limitations**: Because this was a student project with a limited timeframe, advanced features like multi-template support or drag-and-drop layout customization couldn't be added.
- **No Admin Panel**: For now only the user-facing part is functional. Features like admin analytics or multiple resume versions are not added yet.

  Even with these challenges the core goal was successfully completed. The tool can create AI-enhanced, ATS-ready resumes and is easy to use for end users.

5.6 Results

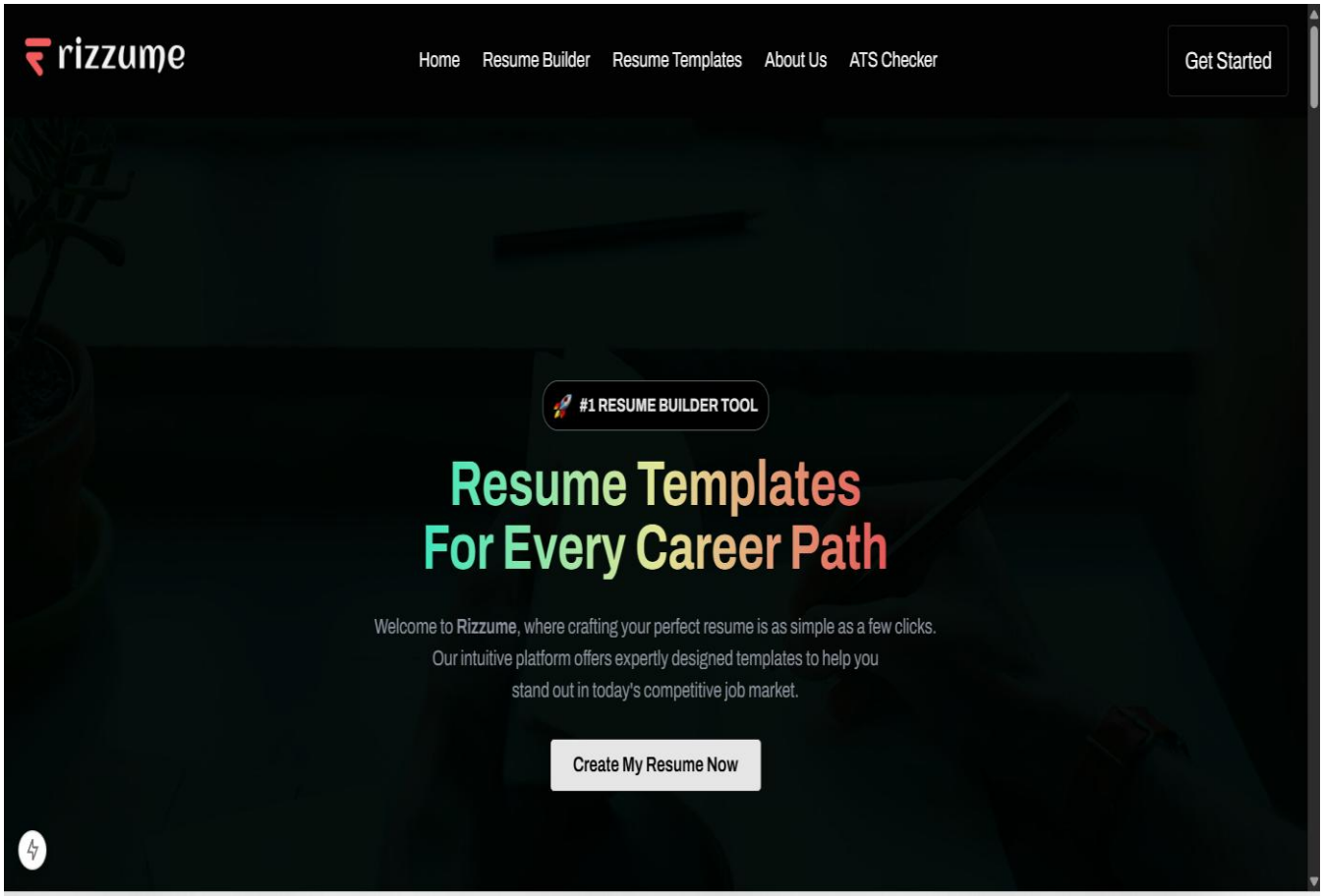

Figure 5.1 Home Page

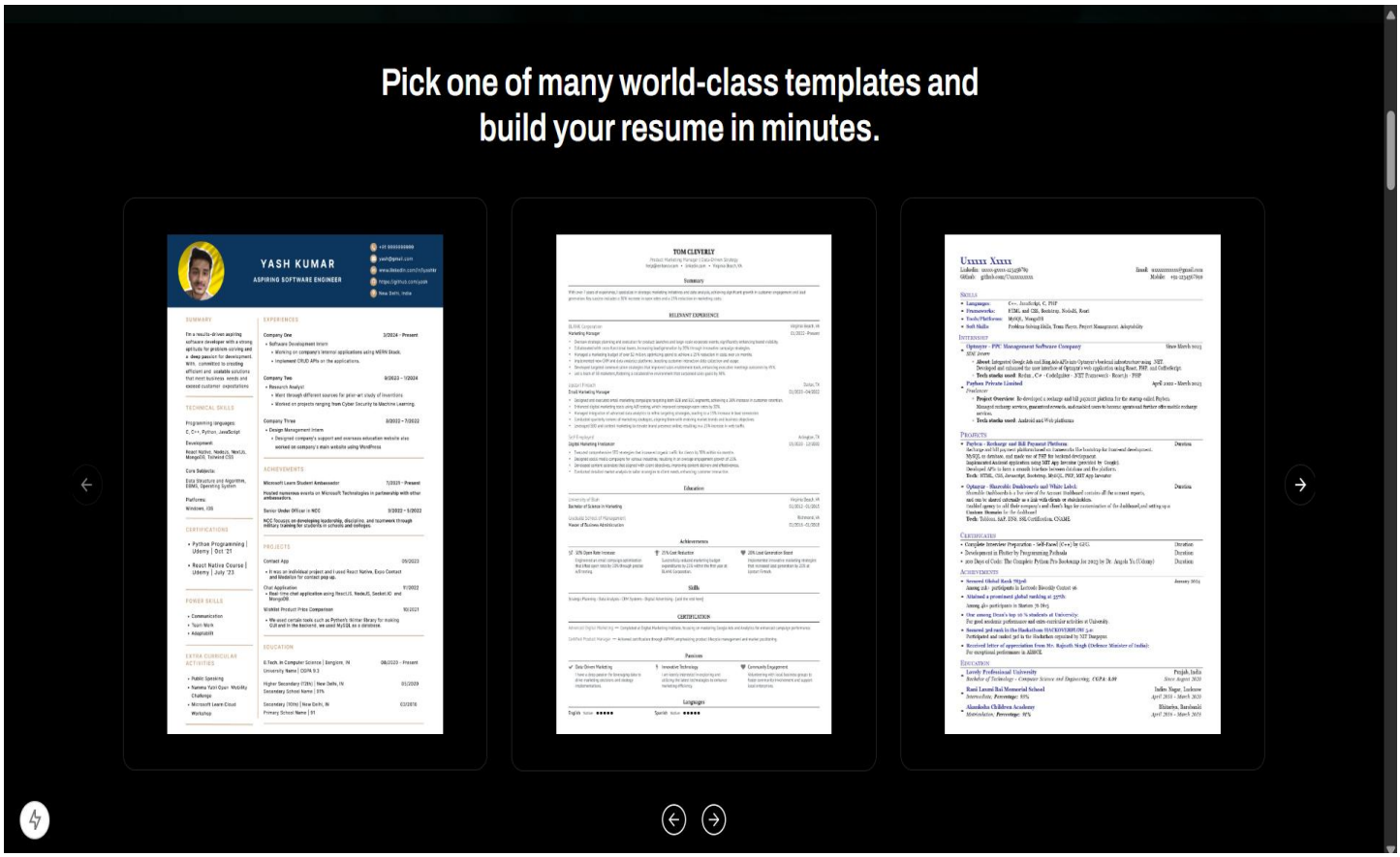

Figure 5.2 Template Section

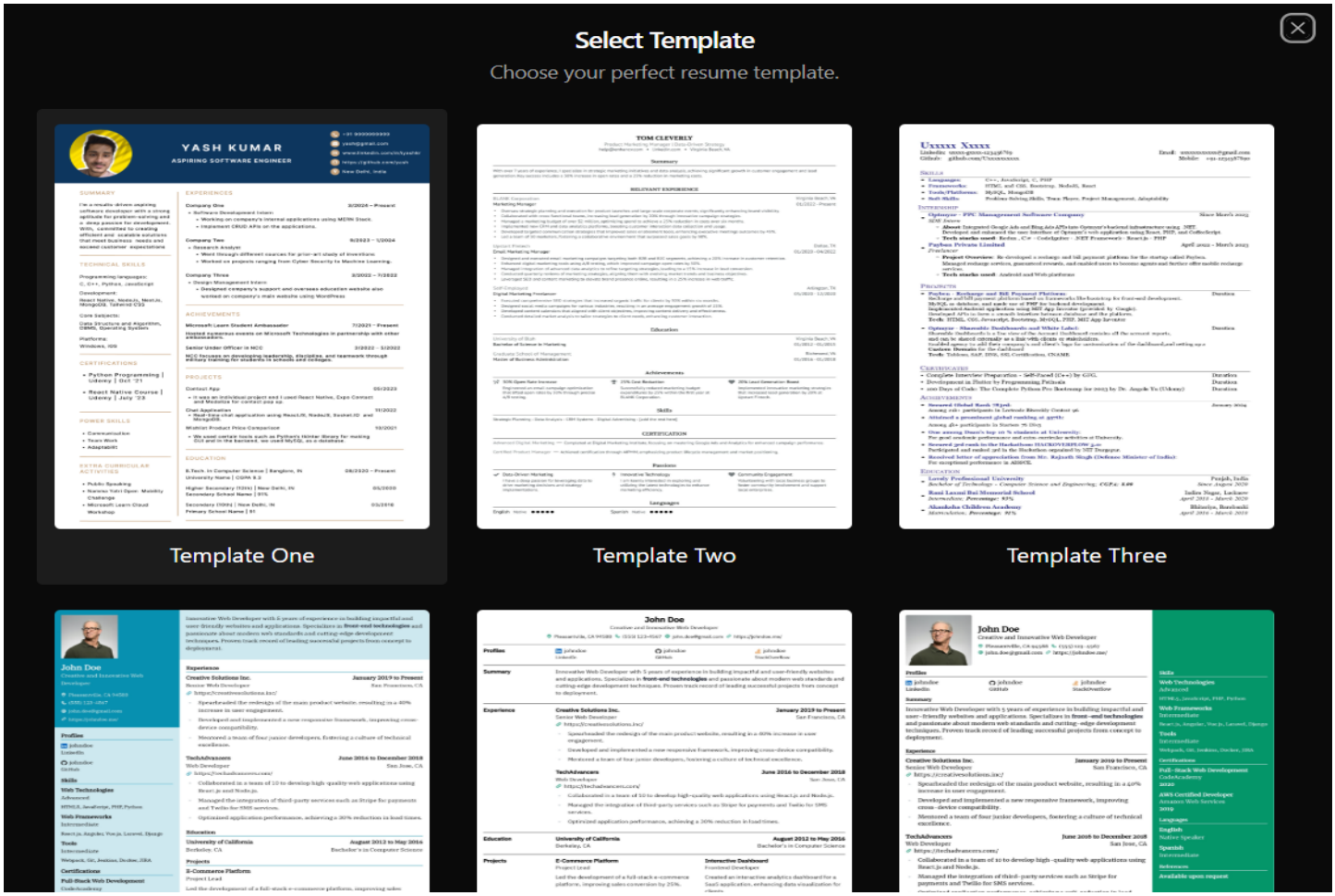

Figure 5.3 Select Template

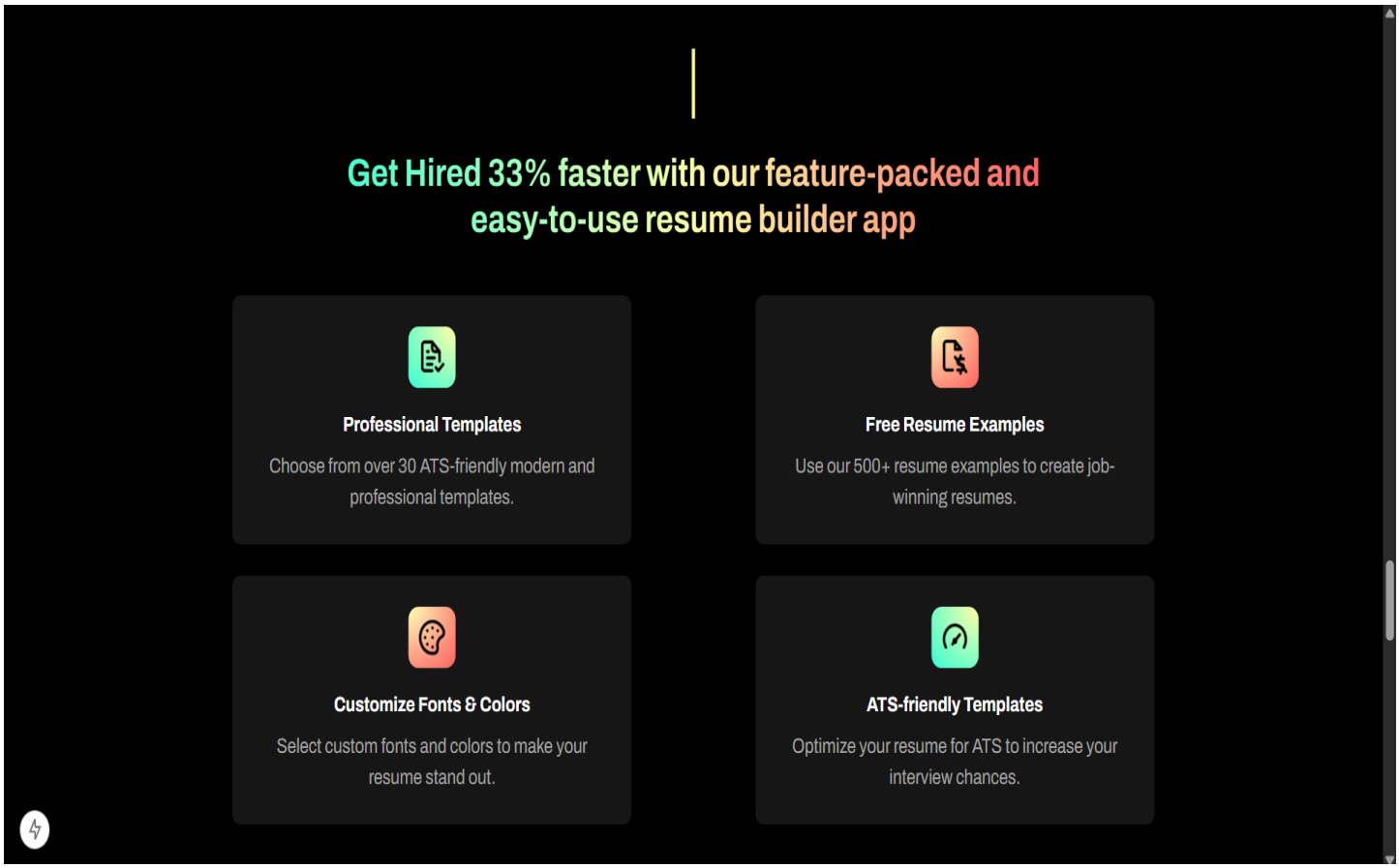

Figure 5.4 Testimonial Section

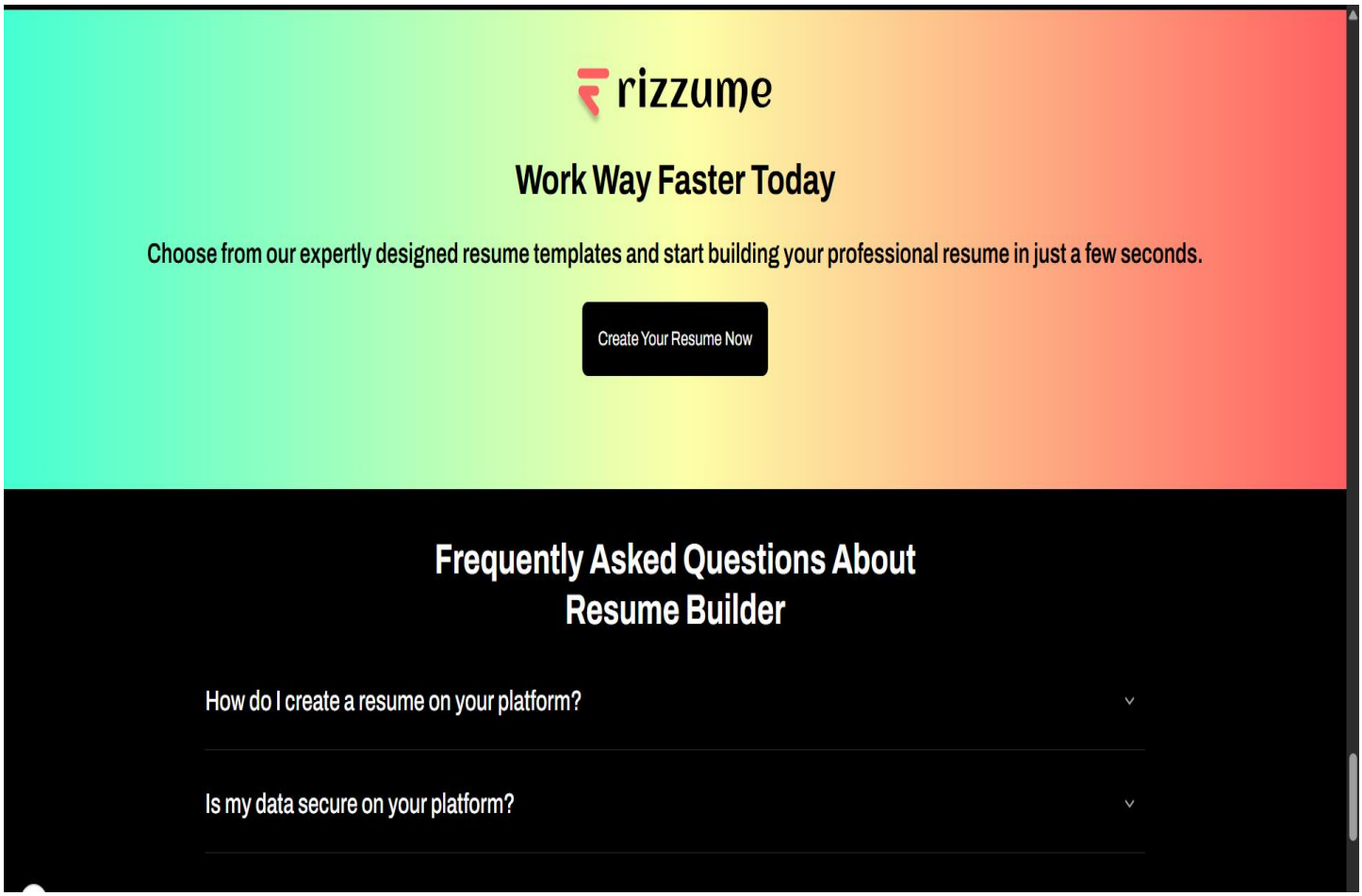

Figure 5.5 FAQ Section

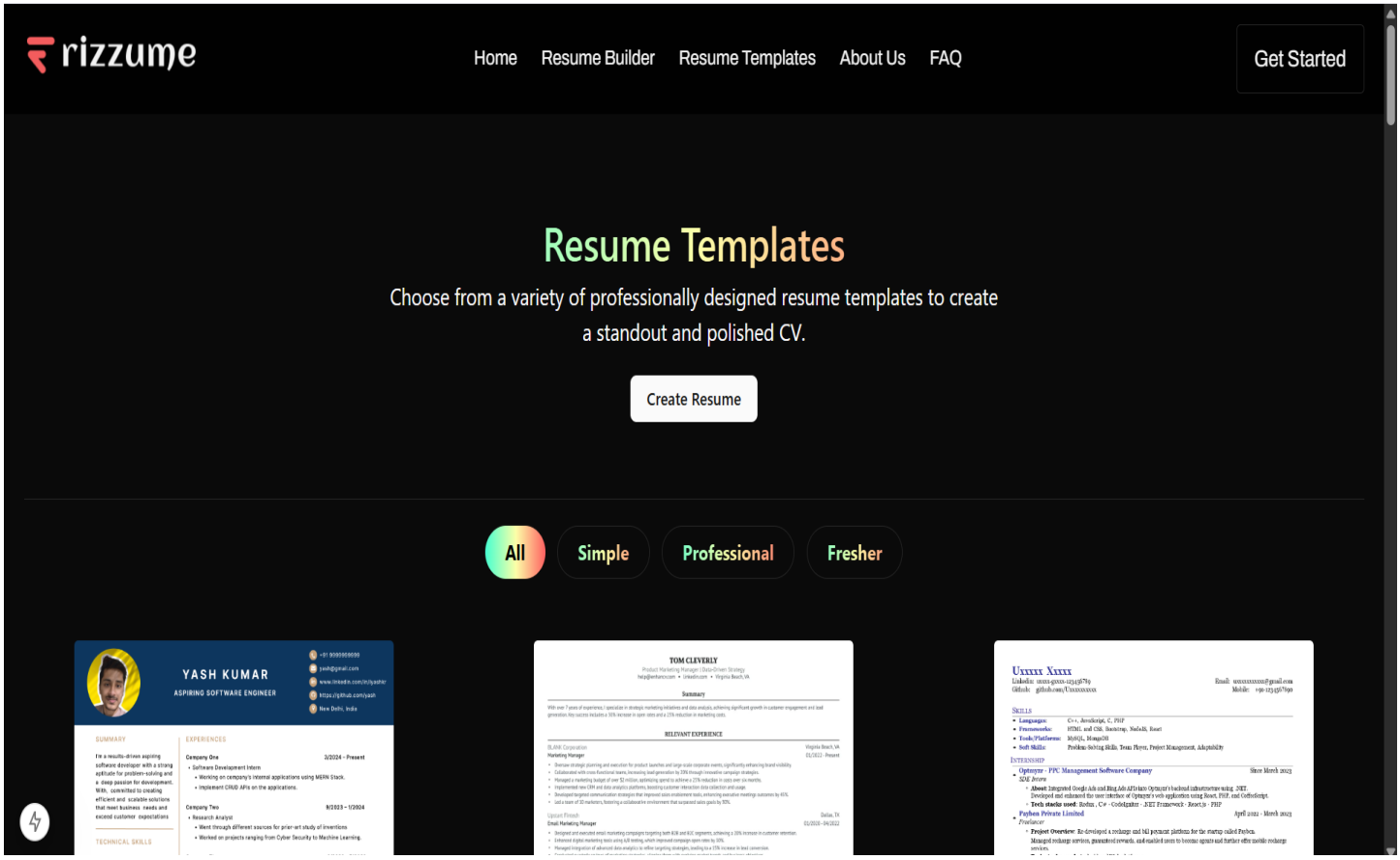

Figure 5.6 Template Page

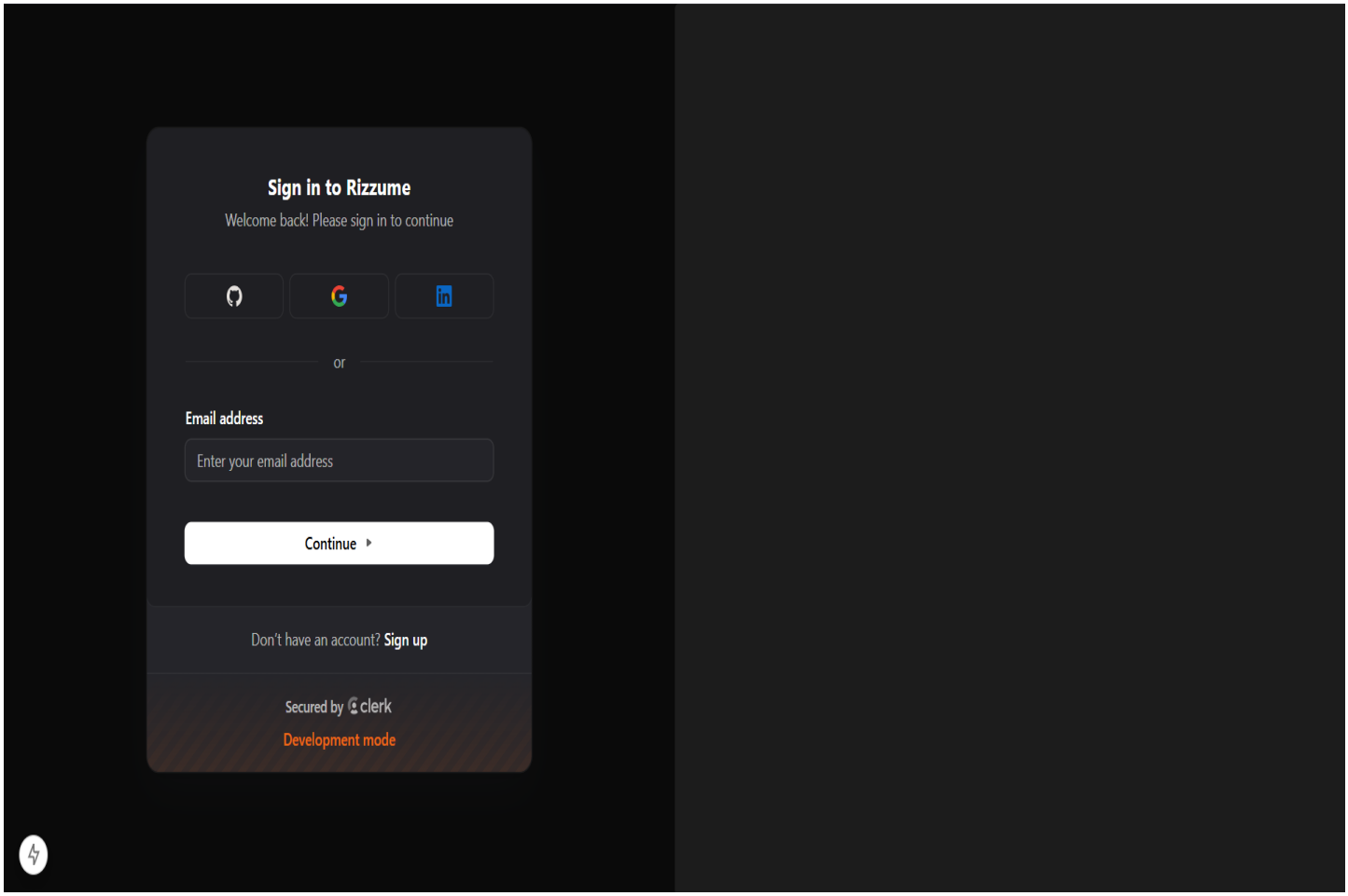

Figure 5.7 Sign In Page

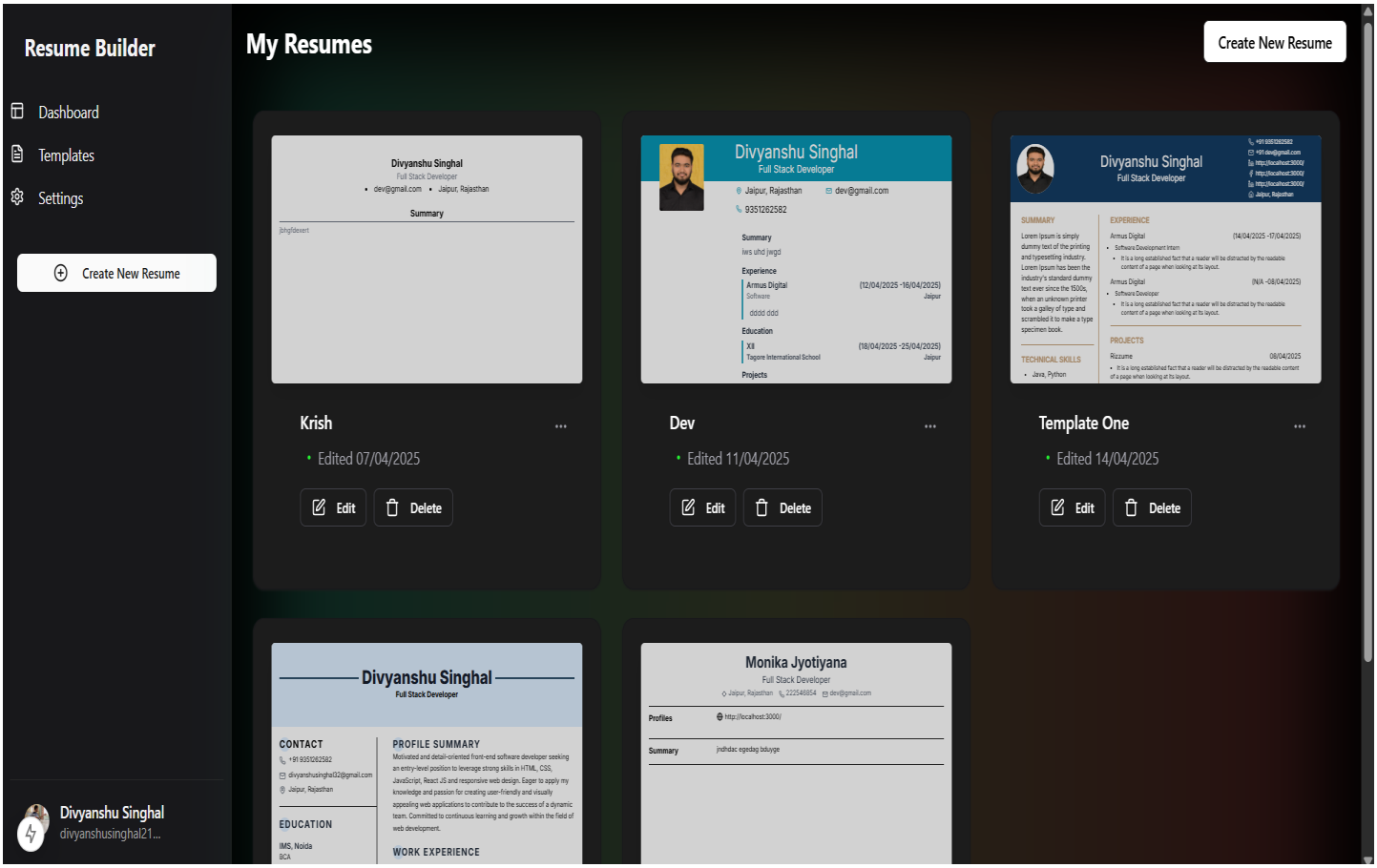

Figure 5.8 Dashboard Page

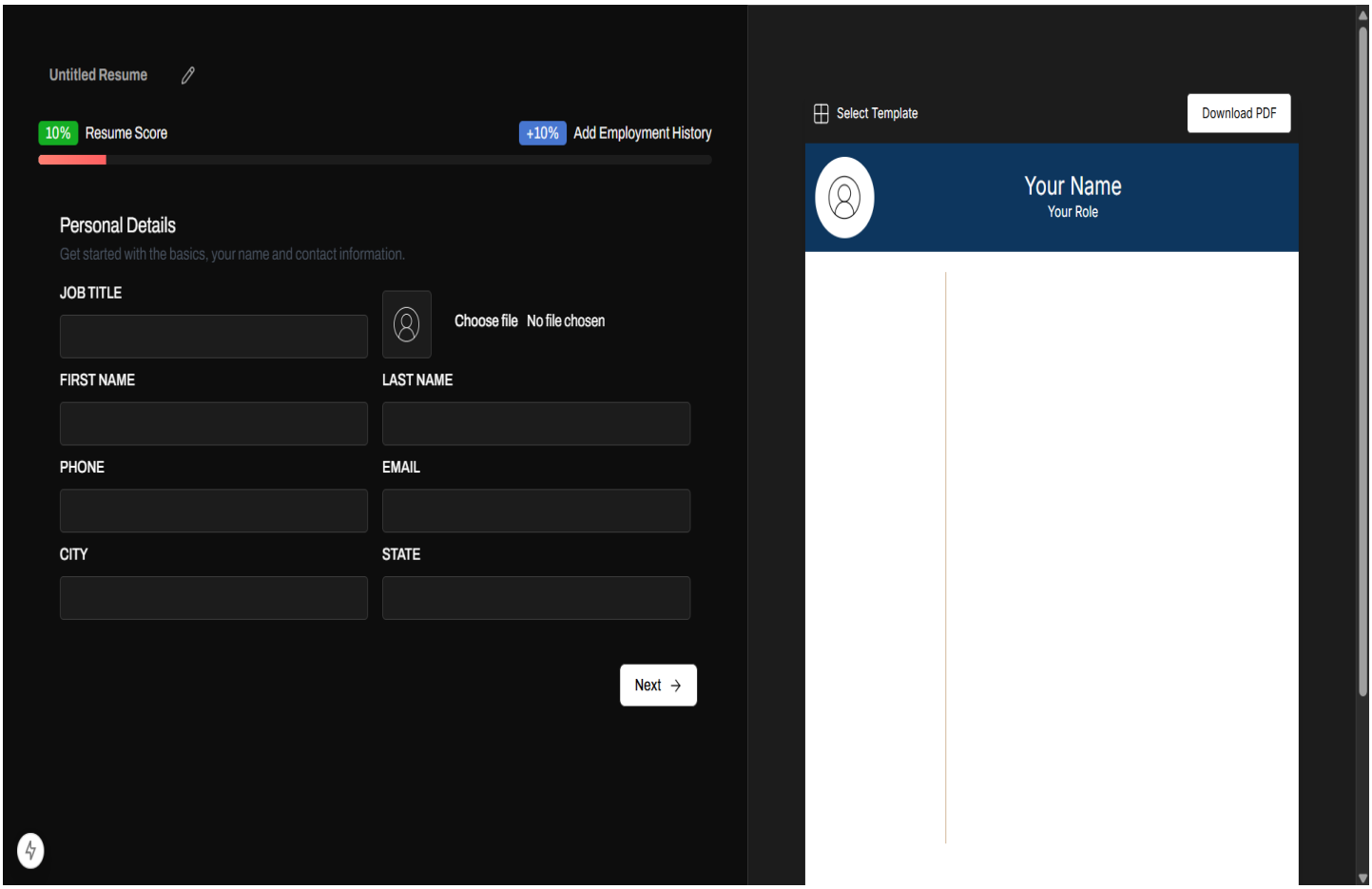

Figure 5.9 Resume Builder Page

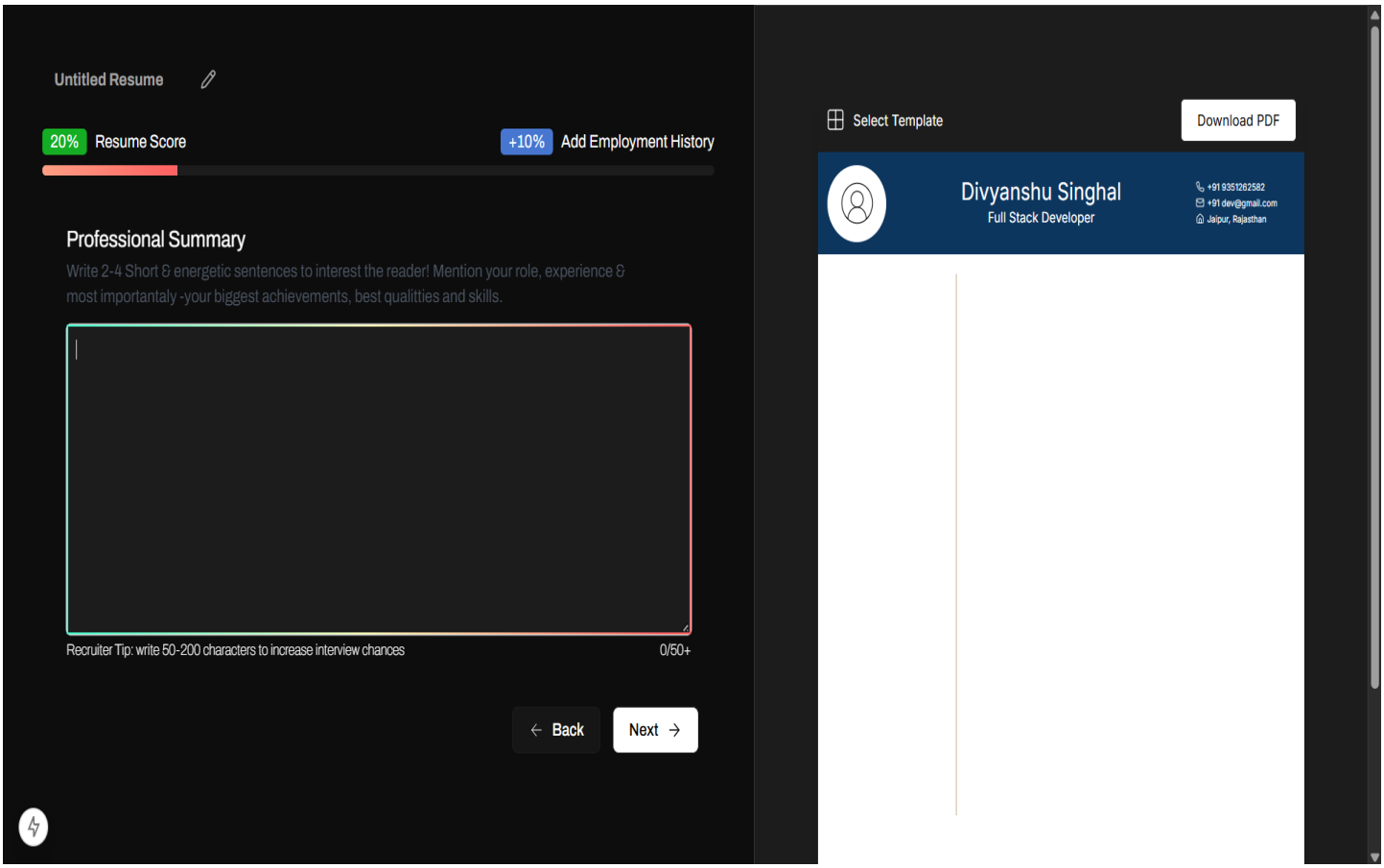

Figure 5.10 Professional Summary

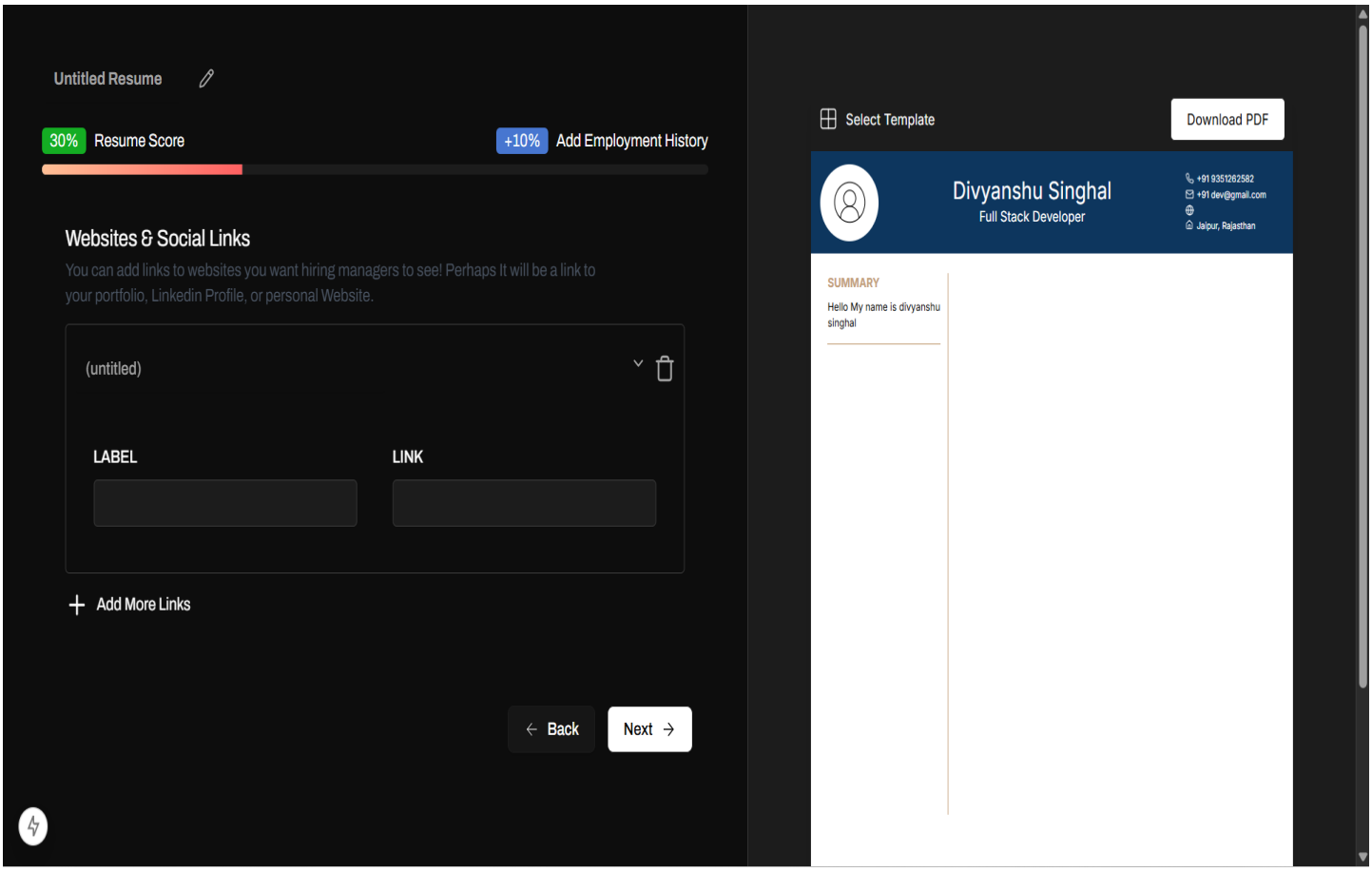

Figure 5.11 Websites and social links

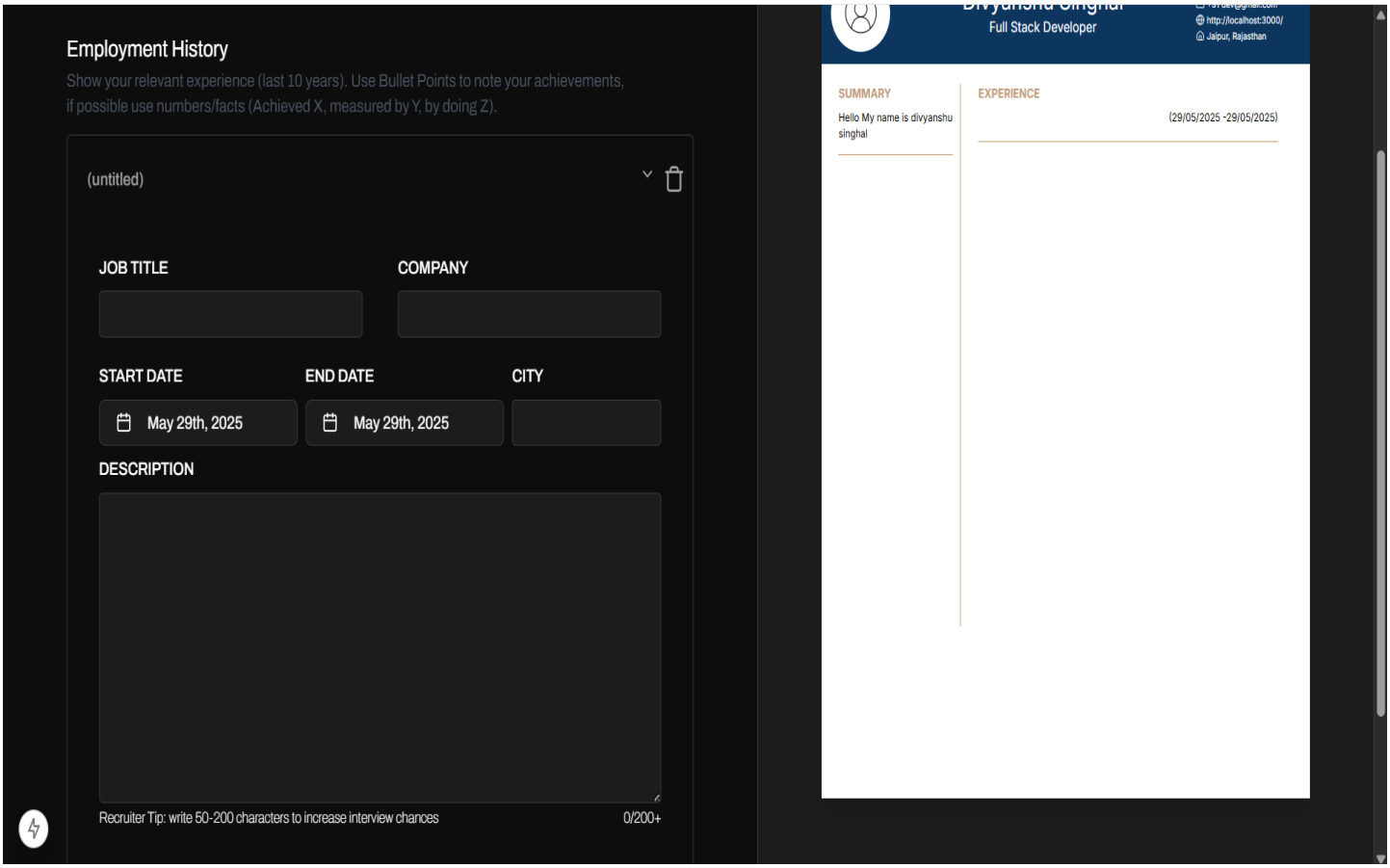

Figure 5.12 Employment History

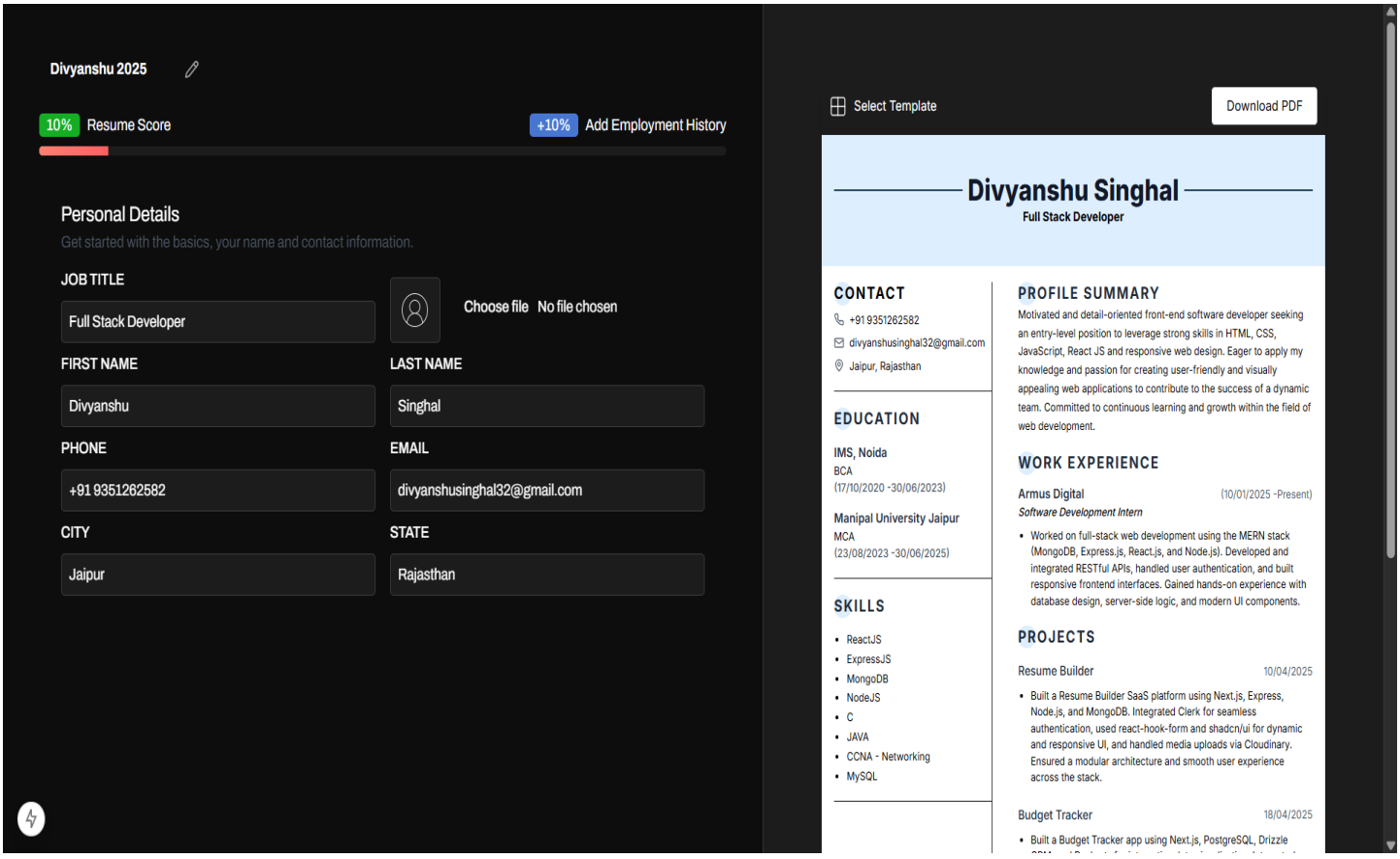

Figure 5.13 Resume Generation

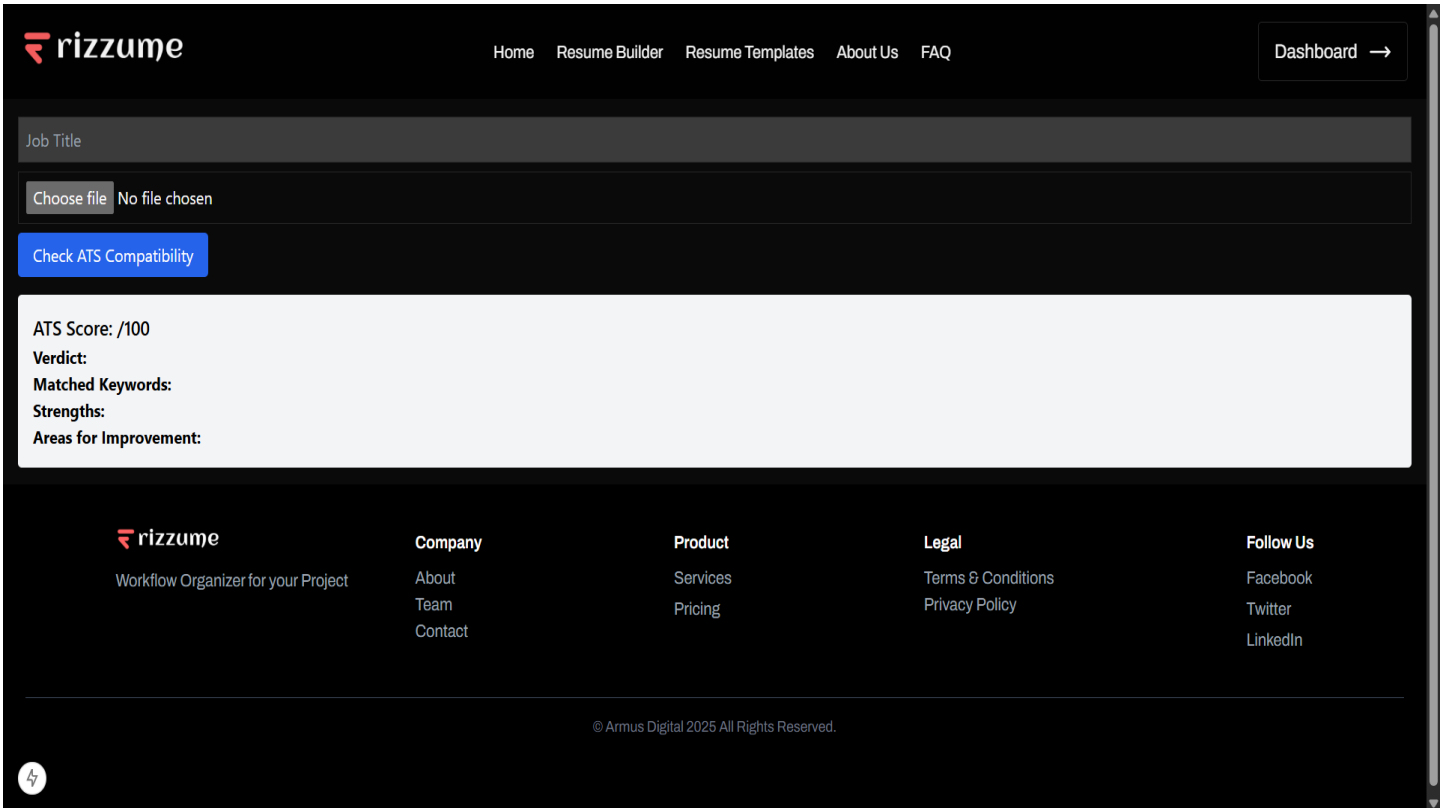

Figure 5.14 ATS Checker

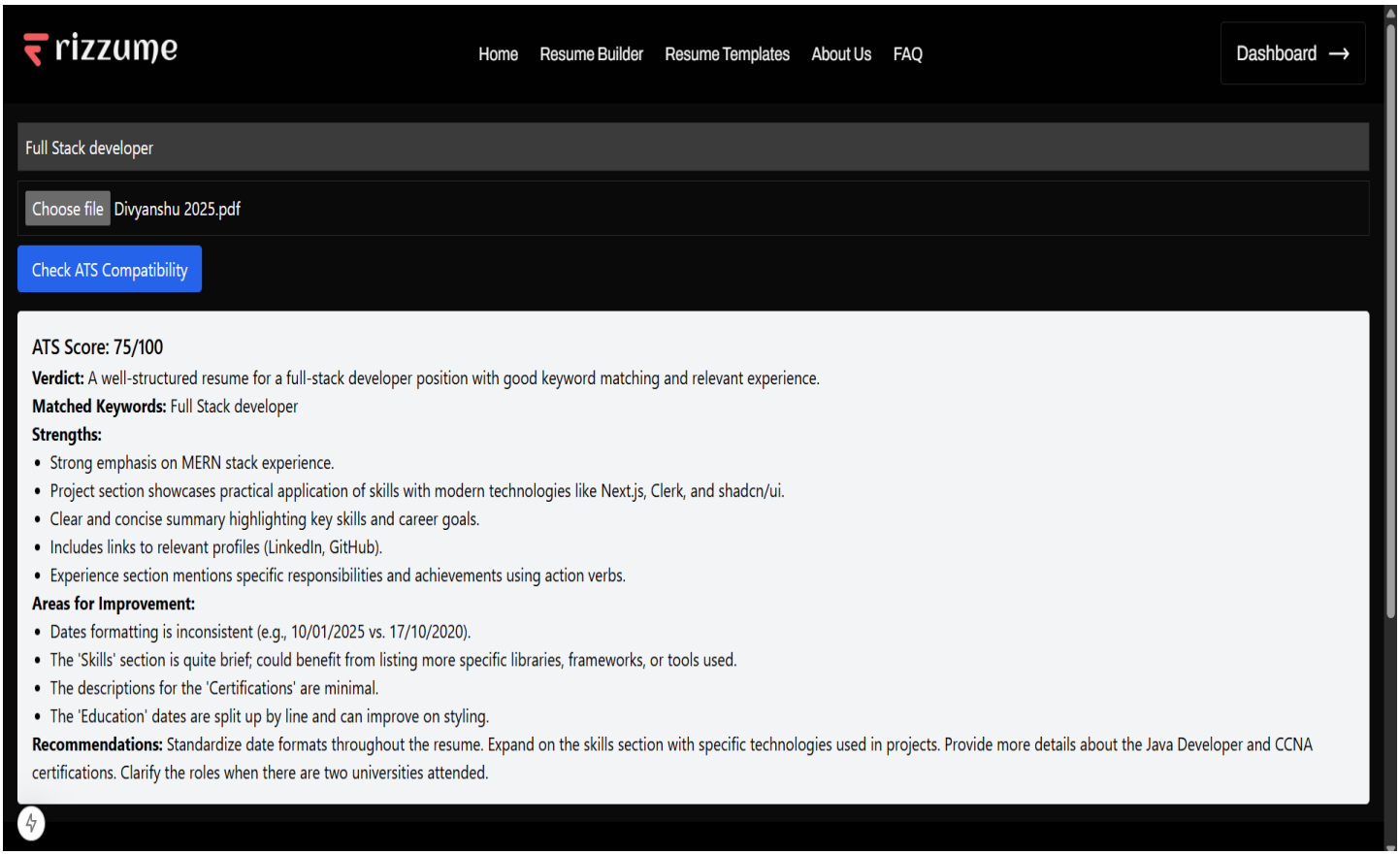

Figure 5.15 ATS Result

Chapter 6
Conclusions And Future Scope

6.1 Conclusions

The Resume Builder project was created to solve a real and common problem faced by students, job seekers, and freshers – the difficulty in creating a professional, clean, and ATS-compliant resume. Many users either use generic templates or manually edit Word files which often lead to formatting issues or lack of clarity. This project focuses on making resume creations simple, guided, and enhanced with AI so users don't feel lost when writing about their experiences and skills.

The tool has been built using a full-stack JavaScript environment, combining Next.js and TypeScript on the frontend with Node.js and Express.js on the backend, and MongoDB for database operations. The architecture was kept modular and scalable, and major functionalities like form-based input, live resume preview, PDF generation, and AI-enhanced content suggestions were successfully implemented. Gemini 2 AI was used to give real-time suggestions for sections like summary, experience descriptions, and cover letters, which helped improve the quality of the resume content significantly.

From a development point of view, the project allowed learning and applying various real-world software development practices. Routing and API structures were cleanly separated into modules like routes, controllers, and schema models. Error handling and form validation were taken seriously to make the platform more stable and reliable. The entire resume form is broken into steps, which improves user experience and allows them to focus on one section at a time. Features like auto-save and preview also contributed to the ease of use.

Testing the tool with real users gave useful insights. While most users appreciated the design and AI assistance, there were small problems noticed, like some layout issues on smaller screens and occasional slow AI responses. These issues have been documented and could be fixed in future versions. It was also observed that users preferred more customization options in templates, which was not included due to time limitations in the current version.

This project not only helped in applying knowledge gained during MCA but also improved problem-solving skills, team communication (where required), and project management. It involved planning the entire workflow – from database structure to frontend logic – and understanding how everything connects in a full-stack web application. Special attention was given to making the UI clean and accessible, keeping in mind the non-technical nature of the target audience.

Though some advanced features like user dashboard, multiple resume support, or white-labeled deployment were not added, the current version of the project still performs well in fulfilling its main goal – helping users build impactful and professional resumes with minimal effort. This can serve as a foundation for future improvements where additional features and scalability can be built on top.

In conclusion, the Resume Builder project achieved its objectives in terms of functionality, performance, and user-friendliness. It has strong practical relevance and can be further evolved into a full-fledged career tool in the future. The development process itself was a great learning journey, reinforcing core concepts and exposing new technologies. It stands as a successful academic project with real-world value.

6.2 Future Scope of Work

While the current version of the Resume Builder fulfills the core requirements and provides a working system with AI suggestions and ATS optimization, there are still many areas where it can be further improved or expanded in future updates. These improvements would not only enhance the user experience but also make the platform more scalable, flexible, and competitive in the long run.

6.2.1 Multiple Resume Templates

Right now, the system offers just one basic template for all users. But different jobs and industries require different styles of resumes. So, in the future, more pre-designed resume templates can be added, each with unique layouts and color schemes. Giving users the option to switch between templates with just one click would improve personalization and make the platform more useful for a wider audience.

6.2.2 Multiple Resume Versions for Each User

Many users apply to different companies with different roles. So instead of redoing the whole resume again and again, there can be a feature to create and manage multiple resume versions under a single user profile. This way, they can quickly duplicate a resume, make small changes for a new job, and save all versions for future use. It saves time and allows more focused job applications.

6.2.3 Admin Panel and Analytics Dashboard

To make the system more complete, especially for enterprise use, an admin panel can be introduced. This would allow backend admins or college admins (if used in institutes) to monitor user activity, see how many resumes are being created, which templates are most used, and identify trends in real-time. It can also include user management features, bug reporting tools, and content moderation if AI is being used widely.

6.2.4 Mobile App Version

In today's time, most users prefer doing small tasks directly from their phone. So a mobile version of the tool, either as a PWA (Progressive Web App) or as a native Android/iOS app, can make it easier for users to build and update resumes on the go. It doesn’t need to have all the features of the desktop app, but basic resume editing, AI suggestions, and exports should be supported on mobile.

6.2.5 Import from Word or PDF

Another powerful addition could be an import feature that allows users to upload their old Word or PDF resumes. The AI system can read and extract important content such as work experience, education, and skills from those files and automatically fill the form sections. This reduces the user’s effort and improves data accuracy while reusing older content.

6.2.6 Offline Mode

Many users may not always have a stable internet connection. To make the system more accessible, an offline mode can be developed where the user can fill out the forms and edit data without an internet connection. Once they are back online, the system can sync the data with the database. This is especially helpful for students or people in remote areas who don’t have reliable internet.

6.3 Learning Outcomes

Working on this Resume Builder project was a great learning experience that enhanced both technical and soft skills. One of the major outcomes was gaining a deeper understanding of full-stack web development using the MERN stack (MongoDB, Express.js, React/Next.js, Node.js) along with TypeScript. From setting up backend APIs and authentication to handling real-time form validation and database storage, the project gave hands-on exposure to building scalable web applications.

Another key takeaway was learning how to integrate AI models like Google's Gemini 2 into practical use case. Understanding how to send data to the AI API, handle the response, and use it to improve resume content taught a lot about working with external APIs and processing natural language-based data. It also opened a broader perspective on how AI can actually improve user experience in modern apps.

Apart from the technical side, this project also strengthened skills related to debugging, user testing, and performance optimization. Fixing layout issues, handling mobile responsiveness, and improving load times helped develop a better sense of UI/UX design. Managing autosave features, PDF generation, and ensuring a smooth data flow between frontend and backend contributed to writing clean and modular code [20].

Lastly, it also improved soft skills like time management, problem-solving, and communication. Documenting the project, presenting it to peers, and collecting feedback helped in looking at the application from a user's point of view. Overall, this project gave a real sense of how modern software projects are built and deployed, making it a valuable part of the learning journey during MCA.

6.4 Final Remarks

Overall, the Resume Builder project was a great learning journey. From idea planning to development and testing, every phase helped gain new insights into software engineering. Though the system is not perfect, it performs its main task well and leaves room for future innovation. The use of AI for resume assistance is something that can be scaled further with better models, UI improvements, and advanced features. The project has strong potential to grow into a more complete platform in the future.

REFERENCES

[1] Kungwani, B., Manglani, A., Dembal, N., Hirani, H., & Sawlani, L. (2020). Analytical Resume Builder-A web Application for creating a resume which gives a best impact in this competitive world. *Annals of the Romanian Society for Cell Biology*, *24*(2), 235-238.

[2] Kanjalkar, P., Patil, S., Pembarti, S., Sarpe, S., & Kanjalkar, J. (2024, February). Resume Building and Course Recommendation System. In *2024 IEEE International Conference for Women in Innovation, Technology & Entrepreneurship (ICWITE)* (pp. 706-709). IEEE.

[3] Srivastava, S., Shukla, H., Landge, N., Srivastava, A., & Jindal, D. (2024). A Comprehensive Review of Next. js Technology: Advancements, Features, and Applications. *Features, and Applications (May 16, 2024)*.

[4] Richards, G., Zappa Nardelli, F., & Vitek, J. (2015). Concrete types for TypeScript. In *29th European Conference on Object-Oriented Programming (ECOOP 2015)* (pp. 76-100). Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[5] Tilkov, S., & Vinoski, S. (2010). Node. js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, *14*(6), 80-83.

[6] Mardan, A. (2014). Starting with Express. js. In *Pro Express. js* (pp. 3-14). Berkeley, CA: Apress.

[7] Chauhan, A. (2019). A review on various aspects of MongoDB databases. *International Journal of Engineering Research & Technology (IJERT)*, *8*(05), 90-92.

[8] Clerk, A. A., Devoret, M. H., Girvin, S. M., Marquardt, F., & Schoelkopf, R. J. (2010). Introduction to quantum noise, measurement, and amplification. *Reviews of Modern Physics*, *82*(2), 1155-1208.

[9] Imran, M., & Almusharraf, N. (2024). Google Gemini as a next generation AI educational tool: a review of emerging educational technology. *Smart Learning Environments*, *11*(1), 22.

[10] Bugl, D. (2019). *Learn React Hooks: Build and refactor modern React. js applications using Hooks*. Packt Publishing Ltd.

[11] Zod, N., Mucci, A., Bahn, O., Provencal, R., & Shao, Y. (2022). Steel slag-bonded strand board as a carbon-negative building product. *Construction and Building Materials*, *340*, 127695.

[12] Klimm, M. C. (2021). *Design Systems for Micro Frontends-An Investigation into the Development of Framework-Agnostic Design Systems using Svelte and Tailwind CSS* (Doctoral dissertation, Hochschulbibliothek der Technischen Hochschule Köln).

[13] Rajala, O. (2024). Impact of React component libraries on developer experience-An empirical study on component libraries' styling approaches.

[14] Protheroe, G. J. (2004). *Searching for security in a new Europe: the diplomatic career of Sir George Russell Clerk*. Routledge.

[15] Thai, D. (2024). A Next. js-based application for crafting ATS-compatible resumes.

[16] Jilani, A. A., Usman, M., & Nadeem, A. (2011). Comparative study on DFD to UML diagrams transformations. *arXiv preprint arXiv:1102.4162*.

[17] Tan, W. S., Liu, D., & Bishu, R. (2009). Web evaluation: Heuristic evaluation vs. user testing. *International Journal of Industrial Ergonomics*, *39*(4), 621-627.

[18] Denaro, G., Polini, A., & Emmerich, W. (2004, January). Early performance testing of distributed software applications. In *Proceedings of the 4th International Workshop on Software and Performance* (pp. 94-103).

[19] Beizer, B. (1995). *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc..

[20] Niranjanamurthy, M., Yogish, H. K., Abhishek, K. L., & Amulya, M. P. (2020). Progression of Information Sharing, Managing and Storage Using Cloud Environment. *Journal of Computational and Theoretical Nanoscience*, *17*(9-10), 4525-4530.

ANNEXURES
(Frontend)

Home Page:

```
import Banner from "@/components/home/BannerSection";
import Details from "@/components/home/DetailSection";
import FAQs from "@/components/home/FAQSection";
import Hero from "@/components/home/HeroSection";
import Templates from "@/components/home/TemplateSection";
import Testimonials from "@/components/home/TestimonialSection";
import Footer from "@/components/includes/Footer";
import Header from "@/components/includes/Header";


export default function Home() {
  return (

    <div className="bg-[#1E1E1E] text-white overflow-hidden font-[archivo]">
    <Header />
    <Hero />
    <Templates />
    <Details />
    <Details />
    <Testimonials />
    <Banner />
    <FAQs />
    <Footer />
   </div>
  );
}
```

Global CSS:

```
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer base {

  :root {
    --background: 0 0% 100%;
    --foreground: 0 0% 3.9%;
    --card: 0 0% 100%;
    --card-foreground: 0 0% 3.9%;
    --popover: 0 0% 100%;
    --popover-foreground: 0 0% 3.9%;
    --primary: 0 0% 9%;
    --primary-foreground: 0 0% 98%;
```

```
  --secondary: 0 0% 96.1%;
  --secondary-foreground: 0 0% 9%;
  --muted: 0 0% 96.1%;
  --muted-foreground: 0 0% 45.1%;
  --accent: 0 0% 96.1%;
  --accent-foreground: 0 0% 9%;
  --destructive: 0 84.2% 60.2%;
  --destructive-foreground: 0 0% 98%;
  --border: 0 0% 89.8%;
  --input: 0 0% 89.8%;
  --ring: 0 0% 3.9%;
  --chart-1: 12 76% 61%;
  --chart-2: 173 58% 39%;
  --chart-3: 197 37% 24%;
  --chart-4: 43 74% 66%;
  --chart-5: 27 87% 67%;
  --radius: 0.5rem;
  --sidebar-background: 0 0% 98%;
  --sidebar-foreground: 240 5.3% 26.1%;
  --sidebar-primary: 240 5.9% 10%;
  --sidebar-primary-foreground: 0 0% 98%;
  --sidebar-accent: 240 4.8% 95.9%;
  --sidebar-accent-foreground: 240 5.9% 10%;
  --sidebar-border: 220 13% 91%;
  --sidebar-ring: 217.2 91.2% 59.8%;

}

.dark {
  --background: 0 0% 3.9%;
  --foreground: 0 0% 98%;
  --card: 0 0% 3.9%;
  --card-foreground: 0 0% 98%;
  --popover: 0 0% 3.9%;
  --popover-foreground: 0 0% 98%;
  --primary: 0 0% 98%;
  --primary-foreground: 0 0% 9%;
  --secondary: 0 0% 14.9%;
  --secondary-foreground: 0 0% 98%;
  --muted: 0 0% 14.9%;
  --muted-foreground: 0 0% 63.9%;
  --accent: 0 0% 14.9%;
  --accent-foreground: 0 0% 98%;
  --destructive: 0 62.8% 30.6%;
  --destructive-foreground: 0 0% 98%;
  --border: 0 0% 14.9%;
  --input: 0 0% 14.9%;
  --ring: 0 0% 83.1%;
  --chart-1: 220 70% 50%;
  --chart-2: 160 60% 45%;
```

```
    --chart-3: 30 80% 55%;
    --chart-4: 280 65% 60%;
    --chart-5: 340 75% 55%;
    --sidebar-background: 240 5.9% 10%;
    --sidebar-foreground: 240 4.8% 95.9%;
    --sidebar-primary: 224.3 76.3% 48%;
    --sidebar-primary-foreground: 0 0% 100%;
    --sidebar-accent: 240 3.7% 15.9%;
    --sidebar-accent-foreground: 240 4.8% 95.9%;
    --sidebar-border: 240 3.7% 15.9%;
    --sidebar-ring: 217.2 91.2% 59.8%;

  }
}

@layer base {
  * {
    @apply border-border;
  }
  body {
    @apply bg-background text-foreground;
  }
}

@media print {
  #resumePreviewContent {
    zoom: 1 !important;
    padding: 0;
  }
}

@page {
  size: A4;
}

@layer utilities {
  .gradient-border {
    border-image: linear-gradient(
      to right,
      #45ffd2 0%,
      #feffb9 48%,
      #ff6061 96%
    );
    border-image-slice: 1;
    border-width: 2px;
  }
}
```

```
input:-webkit-autofill,
input:-webkit-autofill:hover,
input:-webkit-autofill:focus,
input:-webkit-autofill:active {
  -webkit-box-shadow: 0 0 0 30px #1c1c1c inset !important;
  box-shadow: 0 0 0px 30px #1c1c1c inset !important;
  -webkit-text-fill-color: white !important;
}

/* Custom class for shadcn Input to remove focus styles */
.no-focus:focus {
  outline: none !important;
  border: none !important;
  box-shadow: none !important;
}

:root {
  --gradient-colors: linear-gradient(
    90deg,
    #45ffd2 0%,
    #fdffa8 51%,
    #ff6061 100%
  );
}

.gradient-text {
  background: var(--gradient-colors);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
}

.gradient-bg {
  background: var(--gradient-colors);
}

.gradient-color {
  color: var(--gradient-colors);
}
```

Resume Builder Page:

```
'use client';
import React, { useEffect, useRef, useState } from 'react'

import { ArrowLeft, ArrowRight, Loader2, PencilIcon } from 'lucide-react'
import { Progress } from "@/components/ui/progress"
import { type CarouselApi } from "@/components/ui/carousel"
import {
   Carousel,
   CarouselContent,
   CarouselItem,
} from "@/components/ui/carousel"
import { Button } from '@/components/ui/button'
import { Schema, formDataSchema } from "@/lib/schema/FormSchema"
import { defaultValues } from '@/lib/schema/FormDefaults'
import { useForm } from "react-hook-form"
import { zodResolver } from "@hookform/resolvers/zod"
import { z } from 'zod'
import PersonalDetails from '@/components/builder/forms/PersonalDetails'
import ResumePreview from '@/components/builder/preview/ResumePreview'
import { Form } from "@/components/ui/form"
import ProfessionalSummary from './forms/ProfessionalSummary'
import SocialLinks from './forms/SocialLinks'
import EmploymentHistory from './forms/EmploymentHistory'
import Skills from './forms/Skills'
import Achievements from './forms/Achievements'
import Projects from './forms/Projects'
import Education from './forms/Education'

import Certifications from './forms/Certifications'
import ExtraCurricularActivities from './forms/ExtraCurricularActivities'
import { useFormData } from '@/lib/context/FormDataContext'
import debounce from "debounce";
import useFormPersist from 'react-hook-form-persist'

import FormTitle from './forms/fields/FormTitle';
import { useUser } from '@clerk/nextjs';
import axios from "axios";
import { useParams, useRouter } from 'next/navigation';
import { useTemplate } from '@/lib/context/TemplateContext';

import { Dialog, DialogContent, DialogDescription, DialogFooter, DialogHeader, DialogTitle } from
'../ui/dialog';


function ResumeBuilder({ initialValues }: { initialValues?: Schema }) {
   const [api, setApi] = useState<CarouselApi | null>(null)
   const [step, setStep] = useState(1)
   const { formData, setFormData } = useFormData();
   const [progress, setProgress] = useState(10);
   const { selectedTemplate } = useTemplate();
   const [loader, setLoader] = useState(false)
   const [open, setOpen] = useState(false);

   const router = useRouter()
```

```
type Schema = z.infer<typeof formDataSchema>

const form = useForm<Schema>({
    mode: "onChange",
    defaultValues: initialValues || defaultValues,
    resolver: zodResolver(formDataSchema),
})

const params = useParams()

console.log(params)

// Fetch data from localStorage and initialize the form

const debouncedUpdate = useRef(debounce((data) => setFormData(data), 300)).current;

form.watch((data) => {
    debouncedUpdate(data)

})



useFormPersist("resumeData", {
    watch: form.watch,
    setValue: form.setValue,
    storage: typeof window !== "undefined" ? window.localStorage : undefined,


});




const onNext = async () => {
    const stepKeys: (keyof Schema)[] = [
        "personalDetails",
        "professionalSummary",
        "socialLinks",
        "employmentHistory",
        "skills",
        "achievements",
        "projects",
        "certifications",
        "education",
        "activities",
    ]

    if (stepKeys[step - 1]) {
        const isValid = await form.trigger(stepKeys[step - 1])
        if (isValid) {
            setStep((prev) => Math.min(prev + 1, stepKeys.length))
            api?.scrollTo(step)
            setProgress(progress + 10)
        }
    }
}
```

```
  const onPrev = () => {
    setStep((prev) => Math.max(prev - 1, 1))
    api?.scrollTo(step - 2)
    setProgress(progress - 10)
  }

  function onSubmit(values: Schema) {

    console.log(values)
  }

  const { user } = useUser()

  const handleConfirm = () => {
    setOpen(false);
    router.push("/sign-in"); // Navigate to sign-in page
  };

  const handleResumeData = async () => {
    if(!user)

    {
      setOpen(true)
      return
    }
    else
    {
      setLoader(true)
    }

    const storedData = localStorage.getItem("resumeData");
    if (!storedData) {
      console.error("No resume data found in localStorage");
      setLoader(false)
      return;
    }


    const resumeData = JSON.parse(storedData);
    const formData = new FormData();
    const fileInput = document.querySelector('input[type="file"]') as HTMLInputElement; // Select the file
input
    const file = fileInput?.files?.[0];

    if (file) {
      formData.append("profileImg", file);
      console.log(file) // Append the file
    }

    // Append the JSON data separately
    formData.append("resumeData", JSON.stringify(resumeData));
    formData.append("resumeName", selectedTemplate.name);
    if (user?.id) {

      formData.append("userId", user.id);
      console.log("user", user?.id)
```

```
    }

    try {
      if (params.id) {

        const response = await axios.put(
          `${process.env.NEXT_PUBLIC_API_URL}/api/v1/resume/updateResumeData/${params.id}`,
          formData,
          {
            headers: {
              "Content-Type": "multipart/form-data",
            },
          }
        );
        if (response) {
          console.log("Resume data updated successfully:", response.data);
          router.replace("/dashboard");
          localStorage.removeItem("resumeData");
        }

      } else {
        const response = await axios.post(
          `${process.env.NEXT_PUBLIC_API_URL}/api/v1/resume/saveResumeData`,
          formData,
          {
            headers: {
              "Content-Type": "multipart/form-data",
            },
          }
        );
        if (response) {
          console.log("Resume data saved successfully:", response.data);
          router.replace("/dashboard");
          localStorage.removeItem("resumeData");
        }
      }
    } catch (error) {
      console.error("Error saving/updating resume data:", error);
    } finally {
      setLoader(false)
    }
  };

  return (
    <div className='h-full  text-white font-[archivo]'>
      <div className='flex overflow-hidden'>
        <div className='w-[55%] px-10 py-10 flex flex-col items-start bg-primaryBg'>
          <div className='w-full'>
            <Form {...form}>
              <form onSubmit={form.handleSubmit(onSubmit)}>
                <div className='flex space-x-4 items-center'>
                  <FormTitle name='title' placeholder='Untitled Resume' />
                  <PencilIcon className='w-4' color='#a5a5a5' />
                </div>
                <div className='w-full text-sm mt-5'>
                  <div className='flex justify-between'>
                    <div className='flex items-center space-x-2'>
```

```
                        <div className='bg-[#14AD23] px-2 rounded'>{progress}%</div>
                        <div>Resume Score</div>

                    </div>
                    <div className='flex items-center space-x-2'>
                        <div className='bg-[#4776D0] px-2 rounded'>+10%</div>
                        <div>Add Employment History</div>
                    </div>

                </div>

                <div className='pt-2'>
                    <Progress value={Number(progress)} className='bg-[#1c1c1c] [&>div]:bg-
gradient-to-r [&>div]:from-[#45FFD2] [&>div]:via-[#FEFFB9] [&>div]:to-[#FF6061]' />
                </div>

                <div className='mt-7 px-4'>
                    <Carousel setApi={setApi}>
                        <CarouselContent>

                            <CarouselItem><PersonalDetails /></CarouselItem>
                            <CarouselItem><ProfessionalSummary /></CarouselItem>
                            <CarouselItem><SocialLinks /></CarouselItem>
                            <CarouselItem><EmploymentHistory /></CarouselItem>
                            <CarouselItem><Skills /></CarouselItem>
                            <CarouselItem><Achievements /></CarouselItem>
                            <CarouselItem><Projects /></CarouselItem>
                            <CarouselItem><Certifications /></CarouselItem>
                            <CarouselItem><Education /></CarouselItem>
                            <CarouselItem><ExtraCurricularActivities /></CarouselItem>

                        </CarouselContent>
                    </Carousel>

                    <div className="mt-7 flex justify-end gap-3">
                        {step > 1 && (
                            <Button
                                className="dark:bg-[#161616] text-white border-[#1c1c1c] border
hover:shadow hover:shadow-white"
                                type="button"
                                onClick={onPrev}
                            >
                                <ArrowLeft strokeWidth={1} />
                                Back
                            </Button>
                        )}
                        {step < 10 ? (
                            <Button
                                className="bg-white border-[#1c1c1c] border text-black"
                                type="button"
                                onClick={onNext}
                            >
                                Next
                                <ArrowRight strokeWidth={1.5} />
                            </Button>
                        ) :
                            (
```

```
        <Button
                                    className="bg-green-600 border-green-700 text-white hover:bg-green-
700"
                                    type="submit"
                                    onClick={handleResumeData}
                                    disabled={loader}
                                >
                                    {loader ? (
                                        <>
                                            <Loader2 className="mr-2 h-5 w-5 animate-spin" />
                                            {params?.id ? "Updating...." : "Submitting....."}
                                        </>
                                    ) : (
                                        params?.id ? "Update" : "Submit"
                                    )}

                                </Button>
                            )
                        }
                        <Dialog open={open} onOpenChange={setOpen}>
                            <DialogContent>
                                <DialogHeader>
                                    <DialogTitle>Sign In Required</DialogTitle>
                                    <DialogDescription>
                                        You need to sign in to download. Would you like to continue?
                                    </DialogDescription>
                                </DialogHeader>
                                <DialogFooter className="flex justify-end gap-2">
                                    <Button variant="outline" onClick={() => setOpen(false)}>
                                        Cancel
                                    </Button>
                                    <Button onClick={handleConfirm}>
                                        Go to Sign In
                                    </Button>
                                </DialogFooter>
                            </DialogContent>
                        </Dialog>
                    </div>
                </div>
            </div>
        </form>
    </Form>
</div>
</div>

{/* Right Side - Resume Preview */}
<div className='w-full p-16 bg-[#1c1c1c] border border-[#3c3c3c]'>
    <ResumePreview />
</div>
</div>
</div>
)
}

export default ResumeBuilder
```

Resume Preview Component:

```
"use client";
import React, { useRef, useState } from "react";
import useDimensions from "@/hooks/useDimension";
import { useFormData } from "@/lib/context/FormDataContext";
import { useReactToPrint } from "react-to-print";
import { Grid2x2 } from "lucide-react";
import { useTemplate } from "@/lib/context/TemplateContext";  // Import the context
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogFooter,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "@/components/ui/dialog";
import { useUser } from "@clerk/nextjs";
import { useRouter } from "next/navigation";
import { Button } from "@/components/ui/button";

function ResumePreview() {
  const containerRef = useRef<HTMLDivElement>(null);
  const contentRef = useRef<HTMLDivElement>(null);
  const { width } = useDimensions(containerRef);
  const { formData } = useFormData();
  const { selectedTemplate, setSelectedTemplate, templates } = useTemplate();
  const [open, setOpen] = useState(false);

  const reactToPrintFn = useReactToPrint({
    contentRef,
    documentTitle: formData.title || "Resume",
  });

  const [templateDialogOpen, setTemplateDialogOpen] = useState(false);

  const SelectedTemplateComponent = selectedTemplate.component;

  const { user } = useUser()
  const router = useRouter()
  const handleDownload = () => {
    if (!user) {
      setOpen(true); // Show dialog if no user
    } else {
      reactToPrintFn(); // Trigger print function if logged in
    }
  };

  const handleConfirm = () => {
    setOpen(false);
    const currentPath = "/resume"
    router.push(`/sign-in?redirect_url=${encodeURIComponent(currentPath)}`); // Navigate to sign-in page
  };

  return (
    <div className="w-full h-fit aspect-[210/297] bg-white shadow-lg text-black font-[inter]"
```

```
ref={containerRef}>
    <div className="flex justify-between text-xs bg-[#1c1c1c] text-white p-2 items-center">
      <div className="flex items-center space-x-2">
        <Dialog open={templateDialogOpen} onOpenChange={setTemplateDialogOpen}>
          <DialogTrigger>
            <div className="flex items-center gap-2">
              <Grid2x2 size={20} color="#ffffff" strokeWidth={1.25} />
              Select Template
            </div>
          </DialogTrigger>
          <DialogContent className="lg:min-w-[60%] max-h-full my-5 overflow-y-auto">
            <DialogHeader>
              <DialogTitle className="text-center">Select Template</DialogTitle>
              <DialogDescription className="text-center">
                Choose your perfect resume template.
              </DialogDescription>
            </DialogHeader>
            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3">
              {templates.map((template, index) => (
                <div key={index}  className={`rounded cursor-pointer hover:gradient-border p-3 m-1 text-center
${selectedTemplate.name === template.name ? "bg-innerCard" : "" }`}
                  onClick={() =>
                    {setSelectedTemplate(template);
                    setTemplateDialogOpen(false);}
                  }   >
                  <img src={template.image} alt={template.name} className="w-96 h-80 rounded mb-2" />
                  <span className="text-white">{template.name}</span>
                </div>
              ))}
            </div>
          </DialogContent>
        </Dialog>
      </div>
      <div className="border bg-white text-black p-2 px-4 rounded cursor-pointer hover:shadow-sm
hover:shadow-white"
        onClick={() => handleDownload()}> Download PDF </div>
      <Dialog open={open} onOpenChange={setOpen}>
        <DialogContent>
          <DialogHeader>
            <DialogTitle>Sign In Required</DialogTitle>
            <DialogDescription> You need to sign in to download. Would you like to continue?
</DialogDescription>
          </DialogHeader>
          <DialogFooter className="flex justify-end gap-2">
            <Button variant="outline" onClick={() => setOpen(false)}> Cancel</Button>
            <Button onClick={handleConfirm}>Go to Sign In </Button>
          </DialogFooter>
        </DialogContent>  </Dialog>
    </div>
    <div style={{ zoom: (1 / 794) * width }} className={`${!width && "invisible"} text-black`}
ref={contentRef} id="resumePreviewContent">
      <SelectedTemplateComponent data={formData} />
    </div> </div>
  );
}

export default ResumePreview;
```

(Backend)

Database Connect:

```
import dotenv from 'dotenv'
import connectDB from './db/index.js'
import { app } from './app.js'

dotenv.config({
   path: './.env'
})

connectDB()
.then(() => {
   app.listen(process.env.PORT || 6000, () => {
      console.log(`Server is running on port ${process.env.PORT}`)
   })

   app.on("Error: ", (error) => {
      console.log("Error: ", error)
      throw error
   })
})
.catch((error) => {
   console.error("Database Connection Failed !!!", error)
})
```

ExpressJS (App File Code) :

```
import express from 'express'
import cors from 'cors'
import cookieParser from 'cookie-parser'
import passport from './config/passport.js'


//import Routes

import userRouter from './routes/user.routes.js'
import resumeRouter from './routes/resume.routes.js'

const app = express()

app.use(cors({
   origin: process.env.CORS_ORIGIN,
   credentials:true
}))
app.use(express.json())
app.use(express.urlencoded({extended: true, limit:"16kb"}))
app.use(express.static("public"))
app.use(cookieParser())
app.use(passport.initialize());

//Router Declarations
app.use("/api/v1/users", userRouter)
app.use("/api/v1/resume", resumeRouter )

export {app}
```

Routes

User Routes:

```
import { Router } from 'express'
import { handleOAuthSuccess, loginUser, logoutUser, refreshAccessToken, registerUser } from
'../controllers/user.controller.js'
import { imageUpload } from '../middlewares/multer.middleware.js'
import { verifyJWT } from '../middlewares/auth.middleware.js'
import passport from "passport";

const router = Router()

router.route("/register").post(imageUpload.single("avatar"), registerUser)
router.route("/login").post(loginUser)

// private routes
router.route("/logout").post(verifyJWT, logoutUser)
router.route("/refresh-token").post(refreshAccessToken)

// Google Auth
router.get("/auth/google", passport.authenticate("google", { scope: ["profile", "email"] }));
router.get("/auth/google/callback", passport.authenticate("google", { session: false }), handleOAuthSuccess);

// GitHub Auth
router.get("/auth/github", passport.authenticate("github", { scope: ["user:email"] }));
router.get("/auth/github/callback", passport.authenticate("github", { session: false }), handleOAuthSuccess);

// LinkedIn Auth
router.get("/auth/linkedin", passport.authenticate("linkedin", { scope: ["r_emailaddress", "r_liteprofile"] }));
router.get("/auth/linkedin/callback", passport.authenticate("linkedin", { session: false }),
handleOAuthSuccess);

export default router
```

Resume Operation Routes:

```
import { Router } from "express"
import {
    getResumeData,
    saveResumeData,
    selectedResumeData,
    updateResumeData,
    deleteResume,
    atsChecker
} from "../controllers/resume.controller.js"
import { imageUpload, resumeUpload } from "../middlewares/multer.middleware.js";

const router = Router()
// router.route("/saveResumeData").post(saveResumeData)
router.route("/saveResumeData").post(imageUpload.single('profileImg'), saveResumeData);
router.route("/getResumeData").get(getResumeData)
router.route("/selectedResumeData/:id").get(selectedResumeData)
router.route("/updateResumeData/:id").put(imageUpload.single('profileImg'), updateResumeData)
router.route("/deleteResume/:id").delete(deleteResume)
router.route("/uploadResume").post(resumeUpload.single('resume_file'), atsChecker)

export default router
```

Controllers

User Controller:

```
import { User } from '../models/user.model.js'
import { ApiError } from '../utils/ApiError.js'
import { ApiResponse } from '../utils/ApiResponse.js'
import { asyncHandler } from '../utils/asyncHandler.js'
import { uploadOnCloudinary } from '../utils/Cloudinary.js'

const generateAccessAndRefreshTokens = async (userId) => {

    try {

        const user = await User.findById(userId)

        const refreshToken = await user.generateRefreshToken()
        const accessToken = await user.generateAccessToken()

        user.refreshToken = refreshToken;
        await user.save({ validateBeforeSave: false })

        return {
            refreshToken,
            accessToken
        }

    } catch (error) {
        throw new ApiError(500, "Internal Server Error")
    }
}

const registerUser = asyncHandler(async (req, res) => {

    const { fullName, email, password, contact } = req.body

    if (
        [fullName, email, password, contact].some((field) => field?.trim === "")
    ) {

        throw new ApiError(400, "All Fields are required!")
    }

    const existedUser = await User.findOne({ email })


    if (existedUser) {
        throw new ApiError(409, "User email already exists")
    }

    const avatarLocalPath = req.file?.path


    const avatar = await uploadOnCloudinary(avatarLocalPath)


    const user = await User.create({
```

```
        fullName,
        avatar: avatar?.url,
        email,
        password,
        contact
    })

    const createdUser = await User.findById(user._id).select(
        "-password -refreshToken"
    )

    if (!createdUser) {
        throw new ApiError(500, "User not created !")
    }

    return res.status(201).json(
        new ApiResponse(200, createdUser, "User registered Successfully!")
    )

})

const loginUser = asyncHandler(async (req, res) => {

    const { email, password } = req.body

    if (!email) {
        throw new ApiError(400, "Email is required")
    }

    const user = await User.findOne({ email })

    if (!user) {
        throw new ApiError(400, "User does not exist")
    }

    const isPasswordValid = await user.isPasswordCorrect(password)

    if (!isPasswordValid) {
        throw new ApiError(400, "Invalid User Credentials")
    }

    const { accessToken, refreshToken } = await generateAccessAndRefreshTokens(user._id)

    const loggedInUser = await User.findById(user._id).select("-password -refreshToken")

    const options = {
        httpOnly: true,
        secure: process.env.NODE_ENV === "production",

    }



    return res.status(200)
    .cookie("accessToken", accessToken, options)
    .cookie("refreshToken", refreshToken, options)
    .json(
```

```
        new ApiResponse(
            200,
            {
                user: loggedInUser, accessToken, refreshToken
            },
            "User logged in SuccessFully"
        )
    )

})

const logoutUser = asyncHandler( async(req,res) => {

    await User.findByIdAndUpdate(
        req.user._id,
        {
            $set:{
                refreshToken: undefined
            },

        },
        {
            new:true
        }
    )
    const options = {
        httpOnly: true,
        secure: process.env.NODE_ENV === "production",
    }
    return res
    .status(200)
    .clearCookie("accessToken", options)
    .clearCookie("refreshToken", options)
    .json(new ApiResponse(200,{},"User Logged out "))
})

const handleOAuthSuccess = async (req, res) => {
    try {
        const { user, tokens } = req.user;
        const options = {
            httpOnly: true,
            secure: process.env.NODE_ENV === "production",
        }
        // Set authentication cookies
        res.cookie("accessToken", tokens.accessToken, options);
        res.cookie("refreshToken", tokens.refreshToken, options);

        return res
        .status(200)
        .json(new ApiResponse(200,
            {
            message: "Authentication successful",
            provider: user.provider,
            user,
            accessToken: tokens.accessToken,
            refreshToken: tokens.refreshToken
        }, "User Logged In"));
```

```
    } catch (error) {

        throw new ApiError(500, error?.message || "Internal Server Error while log in with provider");
    }
};

const refreshAccessToken = asyncHandler(async (req, res) => {
    const incomingRefreshToken = req.cookies.refreshToken || req.body.refreshToken

    if (!incomingRefreshToken) {
        throw new ApiError(401, "unauthorized request")
    }

    try {
        const decodedToken = jwt.verify(
            incomingRefreshToken,
            process.env.REFRESH_TOKEN_SECRET
        )

        const user = await User.findById(decodedToken?._id)

        if (!user) {
            throw new ApiError(401, "Invalid refresh token")
        }

        if (incomingRefreshToken !== user?.refreshToken) {
            throw new ApiError(401, "Refresh token is expired or used")

        }
        const options = {
            httpOnly: true,
            secure: true
        }
        const {accessToken} = await user.generateAccessToken()

        return res
        .status(200)
        .cookie("accessToken", accessToken, options)
        .json(
            new ApiResponse(
                200,
                {accessToken},
                "Access token refreshed"
            )  )
    } catch (error) {
        throw new ApiError(401, error?.message || "Invalid refresh token")
    }
})
export {
    registerUser,
    loginUser,
    logoutUser,
    generateAccessAndRefreshTokens,
    handleOAuthSuccess,
    refreshAccessToken,
}
```

Multer Middleware (To upload File on server):

```
import multer from "multer";

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, './public/temp')
  },
  filename: function (req, file, cb) {
    cb(null, file.originalname)
  }})
const imageFileFilter = (req, file, cb) => {
  if (file.mimetype.startsWith("image/")) {
    cb(null, true);
  } else {
    cb(new Error("Invalid file type. Only images are allowed."), false);
  }
};
const pdfFileFilter = (req, file, cb) => {
  if (file.mimetype === "application/pdf") {
    cb(null, true); } else {
    cb(new Error("Invalid file type. Only PDFs are allowed."), false);
  }};
export const imageUpload = multer({
  storage,
  fileFilter: imageFileFilter
})
export const resumeUpload = multer({
  storage, fileFilter: pdfFileFilter
})
```

User Database Model:

```
import mongoose, { Schema } from "mongoose";
import jwt from 'jsonwebtoken'
import bcrypt from 'bcrypt'

const userSchema = new Schema(
    {
        avatar: {
            type: String,
            default: "https://www.gravatar.com/avatar/"
        },
        fullName: {
            type: String,
            required: [true, "Full Name is required"],
        },
        email: {
            type: String,
            unique: true,
            lowercase: true,
            required: [true, "Email is required"],
        },
     password: {
            type: String,
        },
        contact: {
            type: Number,
            unique: true,
```

```
        },
        provider: {
            type: String,
            enum: ["email", "google", "github", "linkedin"],
            required: true
        },
        providerId:
        {
            type: String
        },
        emailVerified: {
            type: Boolean,
            default: false
        },
        refreshToken: {
            type: String,
            required: true
        }
    },
    {
        timestamps: true  })

userSchema.pre("save", async function (next) {
    if (!this.isModified("password")) return next()
    this.password = await bcrypt.hash(this.password, 10)
    next()
})

userSchema.methods.isPasswordCorrect = async function (password) {
    return await bcrypt.compare(password, this.password)
}
```

Cloudinary (how to update files) :

```
import { v2 as cloudinary } from 'cloudinary';
import fs from 'fs'

cloudinary.config({
    cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
    api_key: process.env.CLOUDINARY_API_KEY,
    api_secret: process.env.CLOUDINARY_API_SECRET
});

const uploadOnCloudinary = async (localFilePath) => {

    try {

        if(!localFilePath) return null

        const response = await cloudinary.uploader.upload(localFilePath,{
                resource_type:"image"
            })

        fs.unlinkSync(localFilePath)
        return response

    } catch (error) {
```

```
        fs.unlinkSync(localFilePath)
        return null
    }
}
export const updateOnCloudinary = async (oldImageUrl, localFilePath) => {
    try {
        // Extract the public ID from the old image URL
        if (oldImageUrl) {
            const parts = oldImageUrl.split("/");
            const publicIdWithExtension = parts[parts.length - 1];
            const publicId = publicIdWithExtension.split(".")[0];  // Remove file extension
            await deleteFromCloudinary(publicId);
        }
        const newUpload = await uploadOnCloudinary(localFilePath);
        console.log("ProfileImg Uploaded Successfully!")
        fs.unlinkSync(localFilePath)
        return newUpload;
    } catch (error) {
        fs.unlinkSync(localFilePath)
        console.error("Cloudinary Update Error:", error);
        return null;
    }
};

export {uploadOnCloudinary}
```

GeminiAI (ATS Response Generator):

```
import { GoogleGenAI } from "@google/genai";

const ai = new GoogleGenAI({ apiKey: process.env.GEMINI_AI_KEY });


export async function atsResponse(jobTitle, resumeText) {

      const prompt = `
      I want you to act as an ATS (Applicant Tracking System) evaluator.
      The job title for this position is: "${jobTitle}"
      Please review the following resume content and provide feedback on its ATS compatibility.
      Return an ATS score (from 0 to 100) based on relevance, keyword usage, and formatting.
      Also, provide feedback for improvement, if necessary.

      Resume:
      "${resumeText}"
      And also give Response in API Response Format Always
    `;

  const response = await ai.models.generateContent({
    model: "gemini-2.0-flash",
    contents: `${prompt}`,
  });

 return response.text
}
```