

# **B9IS109 Web Development for Information Systems Module**

**Lecturer: Dinesh Kanojiya**

## **PROJECT NAME: JOB APPLICATION TRACKER**

<b>STUDENT NAME</b>	<b>STUDENT ID</b>
<b>MAGNUS LOPES</b>	<b>20033903</b>
<b>YASH PALANDE</b>	<b>20043004</b>
<b>HARISH KHUMKAR</b>	<b>20038908</b>
<b>MANKAZ AHAMMED</b>	<b>20047409</b>

<b>1. Introduction.....</b>	<b>3</b>
<b>2. Research and Planning.....</b>	<b>3</b>
What We Researched.....	3
Planning the Tech.....	3
Who Did What.....	4
Our Timeline.....	4
What We Wanted to Achieve.....	4
<b>3. Choice of Framework and Technologies.....</b>	<b>5</b>
Frontend.....	5
Backend.....	5
AI Integration.....	5
Deployment.....	5
<b>4. UX Design.....</b>	<b>6</b>
Figure 1: Dashboard Screen.....	6
Figure 2: Job Applications Page.....	7
Figure 3: Resume Tailoring Interface.....	8
Figure 4: Tailored Resume Preview.....	9
Figure 5: Settings Page.....	9
Figure 6: Login Page.....	10
Figure 7: Registration Page.....	10
Figure 8: Ansible Playbook Execution on AWS EC2 Instance.....	11
<b>5. Web Services.....</b>	<b>11</b>
<b>6. Security Threats and Measures.....</b>	<b>12</b>
Authentication and Authorization with JWT.....	12
Environment Variables for Secrets.....	12
Helmet for Secure HTTP Headers.....	13
Rate Limiting to Prevent Brute Force Attacks.....	13
MongoDB Injection Prevention.....	13
Input Validation and Data Integrity.....	14
Cross-Site Scripting (XSS) Protection.....	14
Secure File Uploads.....	14
Deployment Considerations.....	15
Summary.....	15
<b>7. Web Application Features.....</b>	<b>16</b>
User Authentication.....	16
Dashboard Overview.....	16
Job Application Management.....	16
Resume Tailoring with AI.....	16

Settings Page.....	16
Web Services & External Integration.....	16
Security.....	17
<b>8. Conclusion.....</b>	<b>17</b>
<b>9. Future Scope.....</b>	<b>17</b>
<b>11. Links.....</b>	<b>17</b>
<b>12. References.....</b>	<b>18</b>

## 1. Introduction

Finding and applying for jobs can be stressful, especially when you're juggling multiple applications and trying to tailor your resume for each role. To make this process easier, we created the **Job Application Tracker & Resume Optimizer** — a web app that helps users keep track of their job applications and improve their resumes using AI.

With this app, users can register and log in, add and manage job applications, and upload their resumes. The resume can then be matched with a job description, and the app uses AI to suggest improvements and highlight important keywords.

Built using **React** for the front end and **Node.js/Express** for the backend, the app also uses **MongoDB** to store user and job data. We focused on making the app easy to use, secure, and helpful for anyone looking to apply for jobs.

## 2. Research and Planning

Before building the app, we took time to understand what we wanted to create, what problems it would solve, and how we would divide the work. Our aim was to build a helpful tool that lets people **track their job applications** and also **optimize their resumes** using AI.

### What We Researched

We started by looking at how job seekers manage their applications. Many people use spreadsheets or basic notes, which can be confusing. Others want to customize their resume for every job, but don't know how.

We also checked existing apps like Huntr or LinkedIn. While useful, they don't offer features like **AI resume feedback**, **exporting to Word**, or a **clear dashboard** with all application statuses. That's where our idea came in.

### Planning the Tech

We chose technologies that are modern and work well together:

Part of App	Tech Used	Why We Picked It
Frontend	React + Vite + Tailwind	Fast, clean, and easy to design
Backend	Node.js + Express	Great for building APIs quickly
Database	MongoDB Atlas	Stores our job data and is cloud-based
AI Integration	Together.ai (DeepSeek)	Helps us generate better resumes
Deployment	Docker + AWS EC2	To run the app on the internet securely
Security	Helmet, Rate Limit, Mongo Sanitize	Protects app from common security threats

## Who Did What

We split the work between 3 members based on what each of us liked and was good at:

- **Mankaz** – Built the **Login** and **Register** functionality, and handled initial backend auth setup.
- **Harish Khumkar** – Focused on the **Dashboard** and **Jobs page**, including modals, form validation, and job tracking features.
- **Magnus Lopes** – Worked on the **Tailor Resume** and **Settings page**, integrated the **AI API**, Markdown display, and file export functionality.
- **Yash Palande** – Handled **Deployment**, including setting up **Docker** and **Ansible on EC2**, and fixed issues related to environment variables and production readiness.

## Our Timeline

We broke the project into weekly goals:

Week	Milestone
Week 1	Finalized requirements, designs, and decided on tech stack
Week 2	Implemented core features (auth, dashboard, job tracking)
Week 3	Integrated resume tailoring, added security, deployed to EC2

## What We Wanted to Achieve

- A clean dashboard to manage job applications
- Ability to upload and optimize resumes using AI
- Easy way to view and download the tailored resume
- Protect the app from common attacks
- Run it on the cloud using Docker and AWS

## 3. Choice of Framework and Technologies

We chose tools and technologies that are modern, easy to use, and work well together to build our full-stack job application tracker.

### Frontend

- **React + Vite** – We used React to build the user interface and Vite to make the app load and update really fast.
- **Tailwind CSS** – This helped us design the app quickly using ready-made styling classes.
- **React Markdown** – We used this to neatly show the tailored resume that comes back from the AI.
- **React Router**: Managed page navigation without full reloads.
- **React Hook Form + Yup** – These helped us handle form inputs (like job details) with built-in validation and good user experience.
- **React Hot Toast** – This library shows friendly pop-up messages like “Job added” or “Error occurred”.

### Backend

- **Node.js + Express** – We built our server and API using these, handling things like login, resume tailoring, and saving job data.
- **MongoDB Atlas** – This is where we store user and job data securely in the cloud.
- **Helmet & Rate Limiting** – These protect our app by blocking suspicious requests and limiting how often someone can hit our server.
- **Mongo Sanitize** – This prevents attackers from sending harmful data through our forms.
- **JWT (JSON Web Tokens)**: Used for secure user authentication and protected routes.

### AI Integration

- We connected our backend to an AI service (DeepSeek via Together.ai).
- When users upload their resume and job description, the backend sends it to the AI, which then returns a tailored resume and keywords.
- We show this in a nice format and even let users download it as a Word document.

## Deployment

- **Docker** – We packaged both frontend and backend into containers so they run smoothly anywhere.
- **AWS EC2** – We hosted the app on a cloud server.
- **Ansible** – This tool helped us automate the setup of the server and deployment process.
- **GitHub** – Our code is stored here.

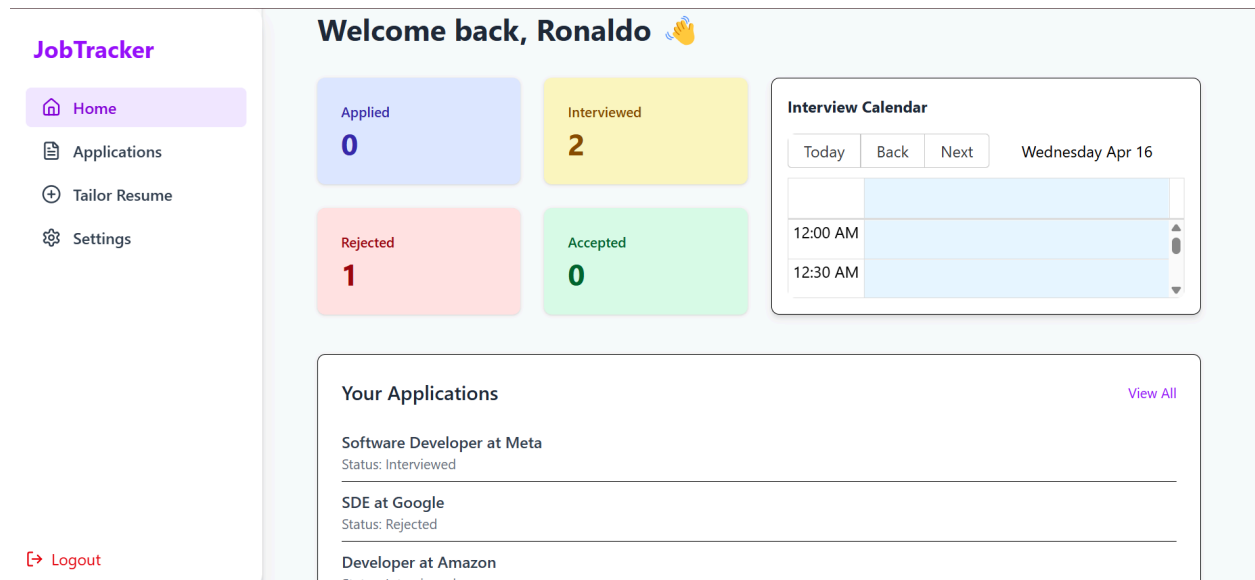
## 4. UX Design

Our application's user experience (UX) was designed with simplicity and clarity in mind to ensure users can easily track their job applications.

- **Clean Navigation:** A fixed sidebar gives users quick access to all main sections—Dashboard, Applications, Tailor Resume, and Settings—making navigation intuitive.
- **Responsive Design:** The interface adjusts seamlessly across desktop, tablet, and mobile devices for a consistent experience.
- **Forms & Feedback:** Forms are simple, validated, and provide instant feedback (like toast messages) on actions such as adding a job or exporting a resume.
- **Resume Preview:** Tailored resumes are shown in a styled markdown format, giving users a clear and professional preview before downloading.
- **Dark/Light Mode:** Users can switch between light and dark themes from the Settings page for better accessibility and comfort.

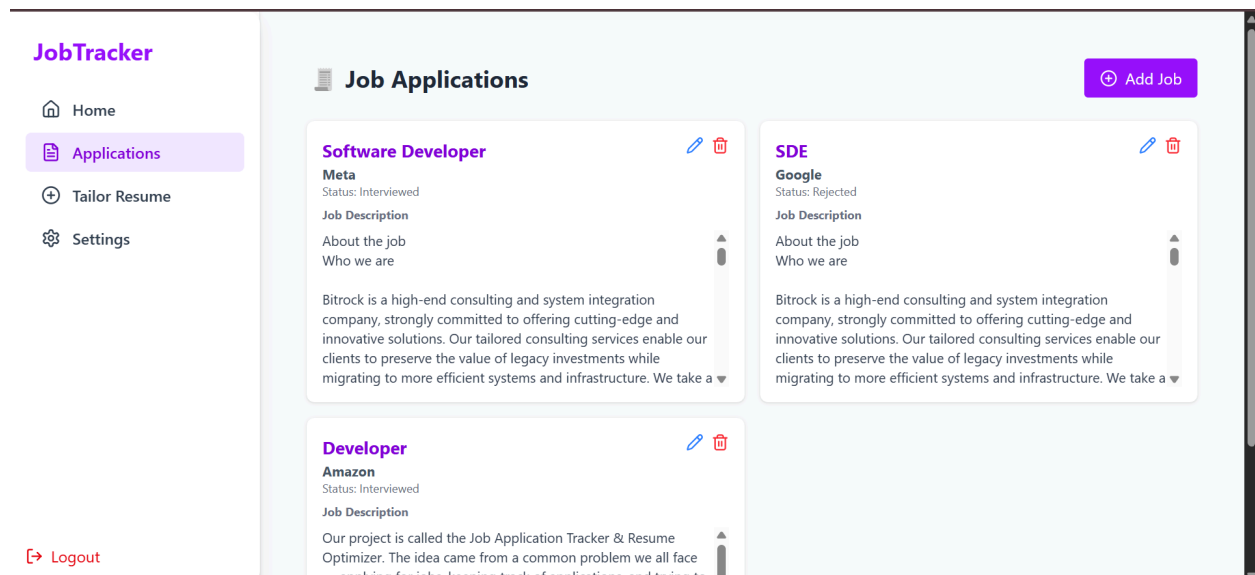
Our UX focus ensures that users can concentrate on managing their job search efficiently, without feeling overwhelmed by the interface.

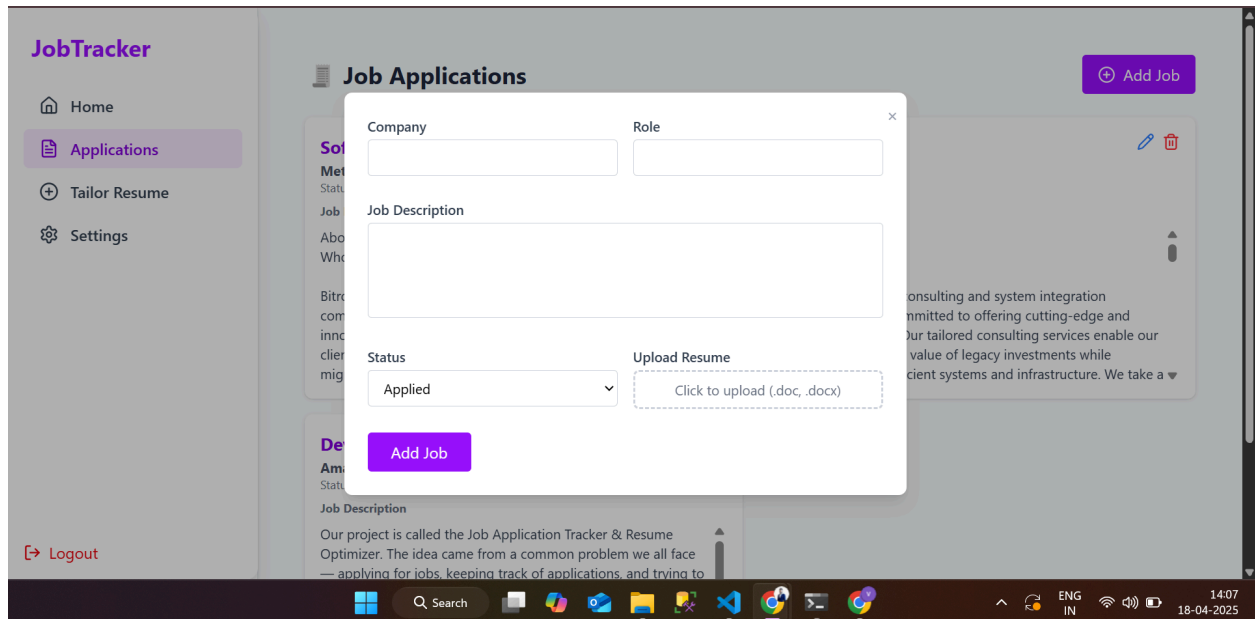
**Figure 1: Dashboard Screen**



The Dashboard provides users with an at-a-glance overview of their job application progress. It displays categorized stats (Applied, Interviewed, Rejected, Accepted) in colored cards for quick recognition. On the right, the Interview Calendar helps users keep track of upcoming interviews. Below, a list of recent applications with their statuses is shown, promoting quick access to updates and next actions. The navigation sidebar allows smooth transitions to other key pages such as Applications, Tailor Resume, and Settings.

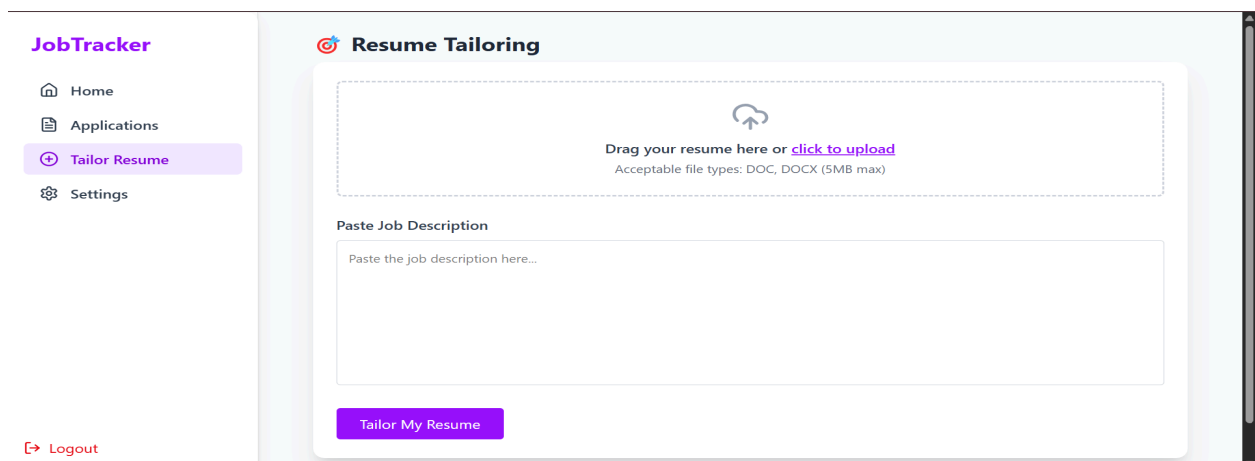
**Figure 2: Job Applications Page**





The Applications page allows users to view, edit, and delete previously added job applications. Each job card includes the role, company name, application status, and a job description snippet. The intuitive layout makes it easy to track progress for each application. Users can also add new applications using the “Add Job” button in the top-right, which opens a form to input job details and upload a resume. The UI emphasizes clarity, with color-coded elements and interactive icons for editing or deleting entries.

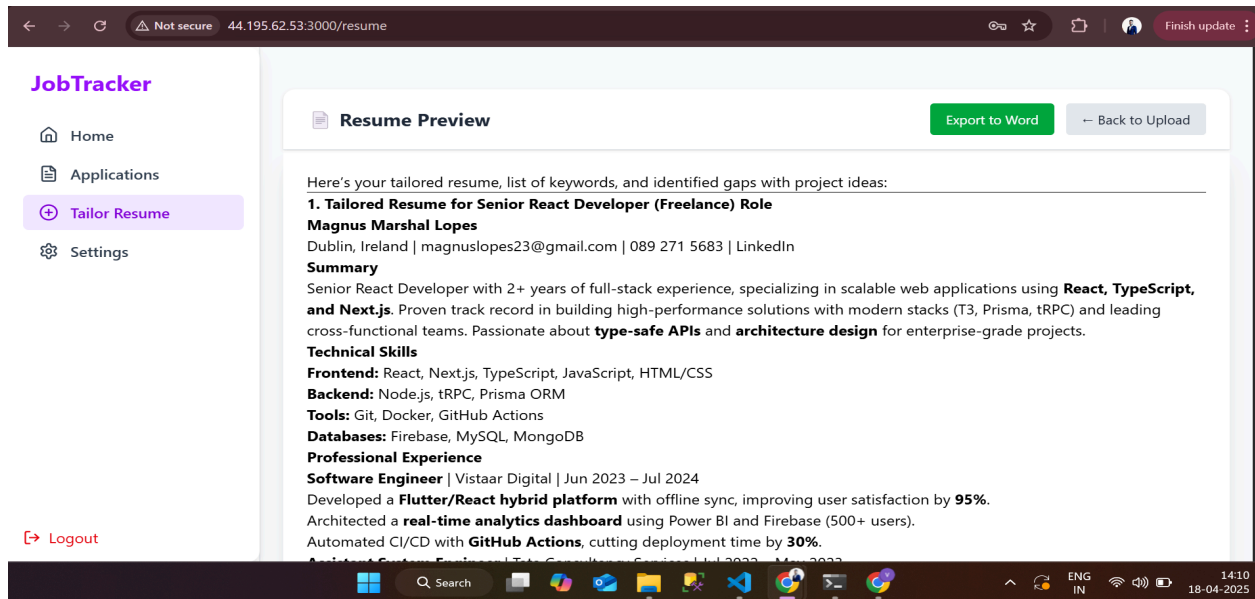
**Figure 3: Resume Tailoring Interface**



The tailor resume section allows users to upload their resume and paste a job description. On clicking “Tailor My Resume,” the app leverages a backend GPT API to personalize the resume according to the job description. It supports DOC and DOCX files up to 5MB.

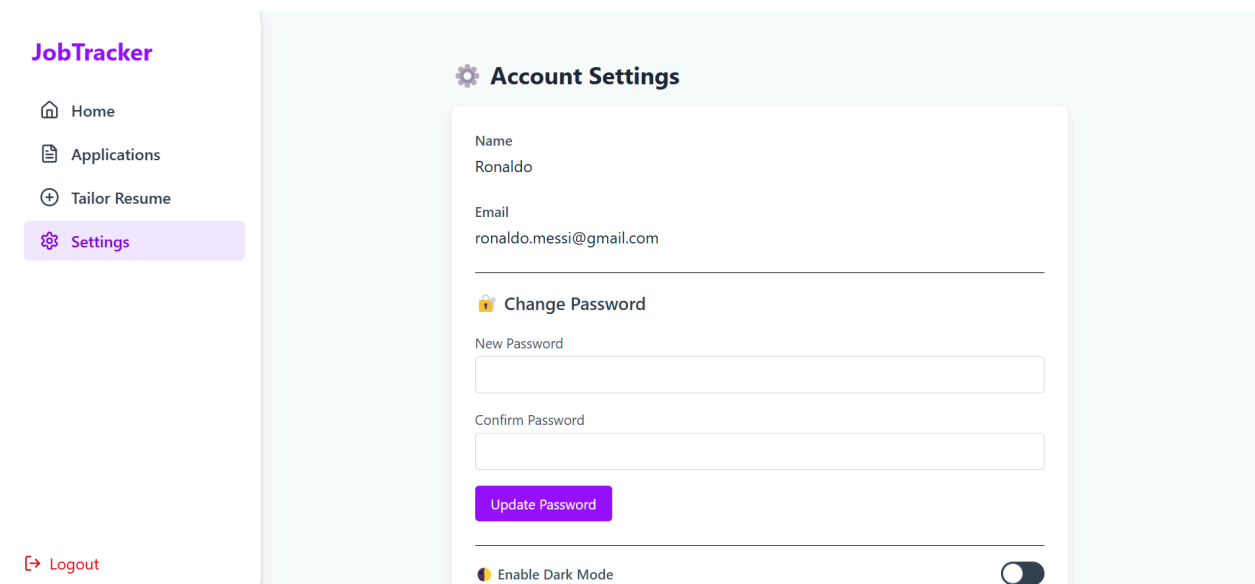


**Figure 4: Tailored Resume Preview**



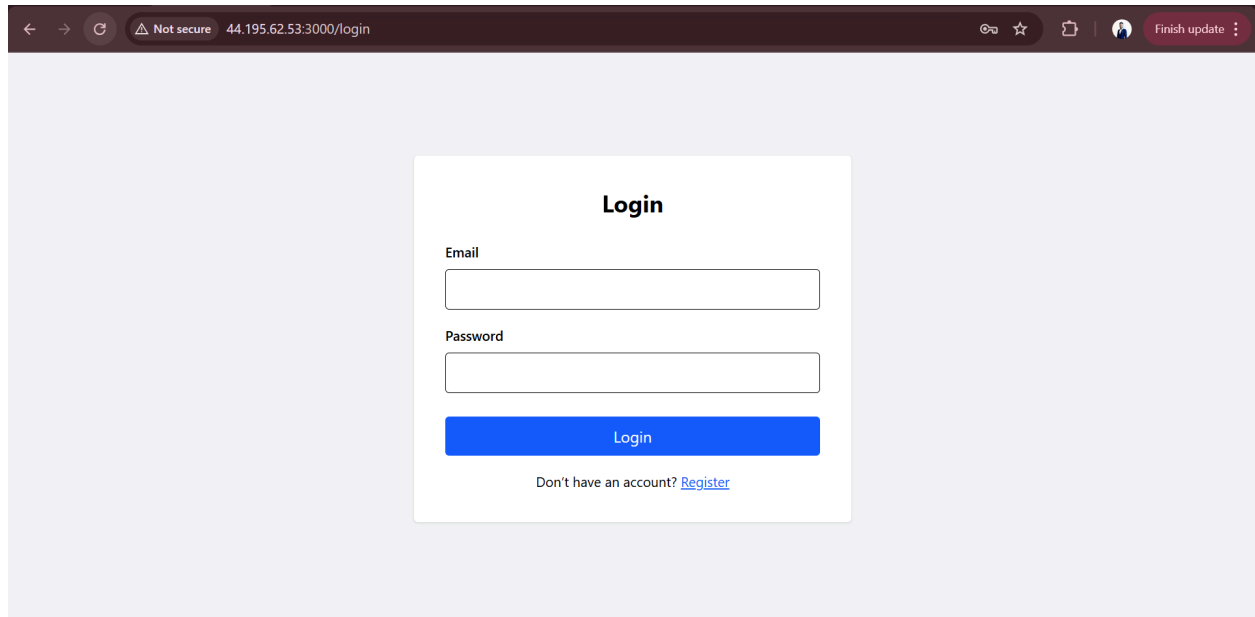
This screen displays the AI-tailored resume in a user-friendly markdown format. It includes details like summary, technical skills, and professional experience. Users can export the final resume to a Word document using the “Export to Word” button or return to upload another file.

**Figure 5: Settings Page**



The settings page enables users to manage their account preferences. It displays the user's name and email, provides functionality to change their password, and includes a toggle to switch between light and dark modes for better accessibility and user comfort.

**Figure 6: Login Page**

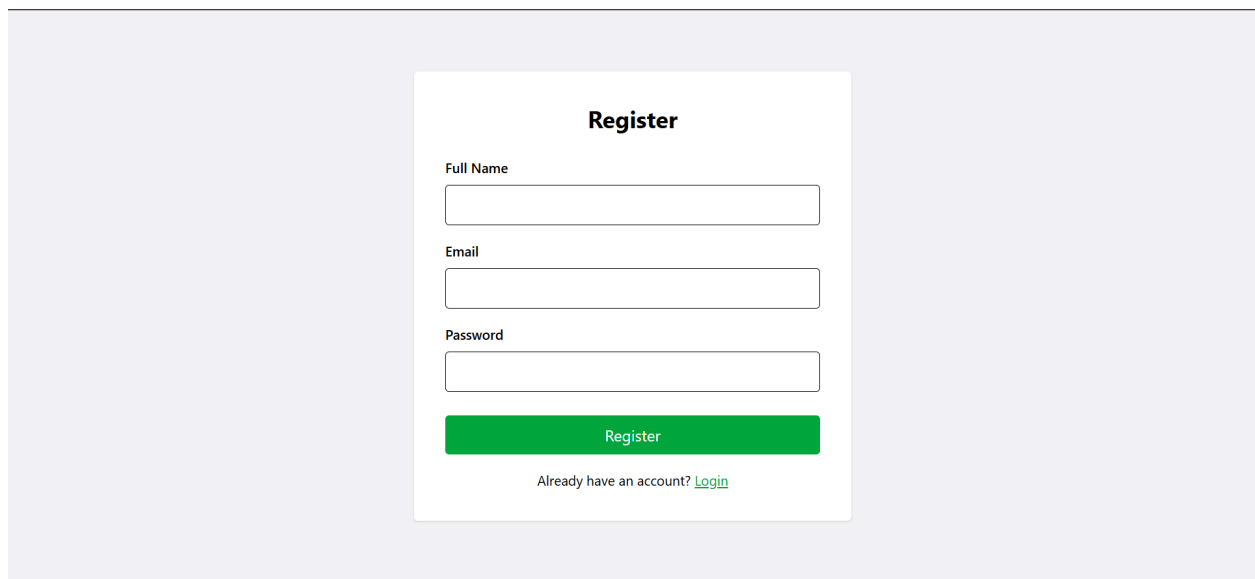


The screenshot shows a web browser window with a dark theme. The address bar displays '44.195.62.53:3000/login' and 'Not secure'. The main content area features a light gray background with a white login form centered. The form is titled 'Login' and contains two input fields: 'Email' and 'Password'. Below the password field is a blue button labeled 'Login'. At the bottom of the form, there is a link that says 'Don't have an account? [Register](#)'.

**Description:**

The login interface prompts users to authenticate using email and password. It ensures secure access to personalized dashboards and features.

**Figure 7: Registration Page**



The screenshot shows a web browser window with a dark theme. The main content area features a light gray background with a white registration form centered. The form is titled 'Register' and contains three input fields: 'Full Name', 'Email', and 'Password'. Below the password field is a green button labeled 'Register'. At the bottom of the form, there is a link that says 'Already have an account? [Login](#)'.

**Description:**

New users can create an account using the Register page. The form captures essential information and integrates validation to ensure user data accuracy and secure onboarding.

**Figure 8: Ansible Playbook Execution on AWS EC2 Instance**

```
ok: [44.195.62.53]
TASK [Clone the GitHub repository] *****
changed: [44.195.62.53]
TASK [Create .env file for backend] *****
ok: [44.195.62.53]
TASK [Run Docker Compose to build and start the app] *****
changed: [44.195.62.53]
TASK [Show container status] *****
changed: [44.195.62.53]
TASK [Print container status] *****
ok: [44.195.62.53] => {
  "container_output.stdout_lines": [
    "NAME", "IMAGE", "COMMAND", "SERVICE", "CREATED", "STATUS",
    "PORTS",
    "job-tracker-backend-1", "job-tracker-backend", "\"docker-entrypoint.s...\"", "backend", "8 seconds ago", "Up 1 seco
nd", "0.0.0.0:5000->5000/tcp", "::5000->5000/tcp",
    "job-tracker-frontend-1", "job-tracker-frontend", "\"docker-entrypoint.s...\"", "frontend", "8 seconds ago", "Up 5 seco
nds", "0.0.0.0:3000->3000/tcp", "::3000->3000/tcp"
  ]
}
PLAY RECAP *****
44.195.62.53 : ok=12 changed=3 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

### Description:

This screenshot shows the final output of the Ansible playbook run on an AWS EC2 instance. It confirms successful:

- Cloning of the GitHub repository
- Creation of backend environment variables
- Building and starting of Docker containers for both backend (**job-tracker-backend**) and frontend (**job-tracker-frontend**)
- Port binding: Backend (5000), Frontend (3000)

The play recap at the bottom (**ok=12, changed=3**) confirms that all tasks executed without any failures, proving the app is correctly hosted using Docker and Ansible on the cloud infrastructure.

## 5. Web Services

In our application, we primarily rely on a single external web service, the **Together.ai API**. This API is used in the **Tailor Resume** feature, where a user's uploaded resume and job description are sent to the API, and a tailored resume is generated in response.

Apart from Together.ai, the rest of our application logic is handled using internal services and open-source libraries in **React (frontend)** and **Node.js (backend)**. These include functionalities such as form validation, routing, markdown parsing, file handling, and secure authentication, all implemented through local code and packages rather than external web APIs.

This design keeps our app efficient and easy to maintain while leveraging the power of AI only where it's needed most.

## 6. Security Threats and Measures

In any modern web application, security plays a vital role in protecting sensitive user data, ensuring trust, and preventing exploitation of the system. Our Job Application Tracker is no exception. To make the application safe for users, we have implemented a variety of techniques and tools that address the most common vulnerabilities in web development. Below is a detailed overview of the threats we considered and how we secured the system against them.

### Authentication and Authorization with JWT

One of the first things we did to secure our app was to implement proper authentication and authorization using **JWT (JSON Web Tokens)**.

- When a user logs in or registers, they receive a secure token (JWT).
- This token is stored in the browser and sent with every request to the server.
- On the server, we validate the token and only allow access if the token is valid.
- This ensures that only authenticated users can access protected routes like the dashboard, job pages, or tailor resume features.

We also created a **middleware** in the backend called **protect**, which checks the token and makes sure the user is valid before allowing any sensitive operations.

### Environment Variables for Secrets

Sensitive information like:

- MongoDB URIs
- JWT Secret Keys

- Together AI API Key

should never be hardcoded into the codebase. Instead, we store these values in a **.env file**, which is listed in **.gitignore** so that it never gets pushed to GitHub.

This ensures that even if someone accesses the source code, they cannot see the critical secrets needed to misuse or hack the system.

## Helmet for Secure HTTP Headers

We used the **helmet** middleware to set various HTTP headers that improve security. This includes:

- Preventing clickjacking with **X-Frame-Options**
- Enforcing HTTPS with **Strict-Transport-Security**
- Hiding the server's technology stack with **X-Powered-By**
- Preventing XSS attacks with **X-XSS-Protection**

This protects the app from a broad range of attacks that rely on manipulating HTTP headers or browser behavior.

## Rate Limiting to Prevent Brute Force Attacks

We implemented **rate limiting** using the **express-rate-limit** package.

- Each IP address can make only a certain number of requests (e.g., 100) in a given time window (e.g., 15 minutes).
- If a user exceeds that limit, they are temporarily blocked.

This helps to prevent **brute force login attempts** and reduces server overload from abuse.

## MongoDB Injection Prevention

MongoDB is powerful but can be vulnerable if not handled carefully. For example, a malicious user might try to send requests like:

```
{  
  
  "email": { "$ne": null },
```

```
"password": "anything"
}
```

To stop such injections, we used the `express-mongo-sanitize` middleware. It automatically removes dangerous MongoDB operators like `$ne`, `$or`, `$gt`, etc., from the request body, params, or query strings.

This helps to **sanitize all incoming inputs** and prevent NoSQL injection attacks.

## Input Validation and Data Integrity

We used `yup` and `react-hook-form` on the frontend to validate forms before they are submitted. This ensures that:

- Required fields are not empty
- Emails are valid
- Resume files are of the right type and size

On the backend, we use checks to ensure the request body is valid and required fields are not missing before processing them.

This prevents malformed data and improves data consistency.

## Cross-Site Scripting (XSS) Protection

While React already escapes input by default, we took extra care when rendering markdown content in the Resume Preview page.

- We used the `react-markdown` package, which by default does not allow dangerous scripts.
- We did not dangerously use `innerHTML`, which is a common cause of XSS in JavaScript apps.

This prevents attackers from injecting JavaScript code via text fields or markdown.

## Secure File Uploads

We allow users to upload resumes in `.doc` or `.docx` format. To make this secure:

- We limit file types using `accept=".doc,.docx"`
- We restrict file size to under 5MB

- We store files in a non-public uploads folder and clean them up as needed

This avoids uploading malicious scripts or oversized files that can slow down or crash the server.

## Deployment Considerations

Although we used environment variables to secure the application, it's important to remember that:

- These variables should be injected securely in the Docker/EC2 environment
- The `.env` file should never be committed to version control
- Secrets in production should be managed using encrypted secrets or cloud services like AWS Secrets Manager

## Summary

Here's a quick checklist of the threats we protected against:

Threat	Protection Used
XSS (Cross-site scripting)	React default + <code>react-markdown</code>
CSRF (Cross-site request)	JWT and CORS protection
SQL/NoSQL Injection	<code>express-mongo-sanitize</code>
Brute Force Attacks	<code>express-rate-limit</code>
Exposed Secrets	<code>.env</code> + <code>.gitignore</code>
Insecure File Uploads	File validation, size/type check
HTTP Vulnerabilities	<code>helmet</code>

By implementing these security techniques, our application is robust, trustworthy, and safe for users. Security is not a one-time step but an ongoing responsibility, and this project follows good practices right from the start.

## 7. Web Application Features

Our Job Application Tracker is a full-featured web application designed to help users manage their job hunt efficiently. Below is a summary of the key features we implemented:

### User Authentication

- **Register and Login:** Secure sign-up and login functionality using JWT tokens.
- **Protected Routes:** Only authenticated users can access features like the dashboard, jobs page, and tailor resume tool.
- **Logout:** Users can easily log out, which clears the session token and redirects to the login page.

### Dashboard Overview

- Displays a summary of job applications by status: Applied, Interviewed, Rejected, and Accepted.
- Includes an **interactive interview calendar** that visually shows upcoming interview events.

### Job Application Management

- **Add, Edit, Delete Jobs:** Users can create job entries with company, role, status, job description, and resume upload.
- **Resume Upload:** Accepts `.doc` or `.docx` files and securely stores them.
- **Status Tracking:** Users can update the status of any job (e.g., Applied, Interviewed, Accepted).
- **Responsive UI:** Clean layout with modals for adding or editing jobs.

### Resume Tailoring with AI

- **Upload Resume + Paste JD:** Users upload their resume and paste a job description.
- **AI Integration (Together.ai):** The backend sends both to a language model API to receive a tailored resume with job-specific keywords.
- **Preview in Markdown:** The response is rendered using `react-markdown` with proper styling.
- **Export to Word:** The user can download the optimized resume as a `.docx` file using a utility function.

### Settings Page

- **Change Password:** Placeholder feature with form validation.
- **Theme Switcher:** Users can toggle between light and dark mode for better accessibility and user preference.

### Web Services & External Integration



- Consumes the **Together.ai API** for resume optimization.
- Uses **React Hook Form + Yup** for validation.
- Applies reusable toast notifications using **react-hot-toast**.

## Security

- Middleware to protect routes, sanitize inputs, prevent brute force attacks.
- Secure file uploads, rate limiting, and environment variables for sensitive keys.

These features were designed to balance usability, security, and performance. The user experience is responsive, modern, and easy to use, with powerful tools to simplify job tracking and improve resume customization.

## 8. Conclusion

The Job Application Tracker is a full-stack web application that helps users manage and tailor their job applications efficiently. Built with React, Node.js, and MongoDB, and integrated with AI through Together.ai, it delivers a secure, scalable, and user-friendly experience. The project showcases our skills in frontend/backend development, API integration, security, and deployment using Docker and AWS EC2.

## 9. Future Scope

- **Job Suggestions Integration:** Connect with public job listing APIs (like LinkedIn, Indeed) to suggest jobs based on user profile.
- **Email Notifications:** Automatically notify users about upcoming interviews or status updates via email.
- **AI-Based Recommendations:** Suggest improvements to resumes based on previous applications and interview results.
- **Recruiter/Admin Dashboard:** Enable recruiters to post jobs, view applications, and communicate with applicants.
- **Multi-language Support:** Add language options to make the app more accessible to global users.
- **Collaboration Features:** Let users share resumes or job applications with mentors or friends for feedback.
- **Mobile App Version:** Extend functionality to iOS and Android platforms using React Native or Flutter.
- **OAuth Integration:** Support login with Google, GitHub, and LinkedIn for better authentication experience.
- **Export Analytics:** Let users download their job application history and insights in CSV or PDF format.

## 11. Links

Github : <https://github.com/magnuslopes23/JobApplicationTracker>

Website : <http://44.195.62.53:3000/>

Video:

[https://drive.google.com/file/d/1J\\_0QcEHhz8lpbh7b2RVfR1o7QypAaGaQ/view?usp=sharing](https://drive.google.com/file/d/1J_0QcEHhz8lpbh7b2RVfR1o7QypAaGaQ/view?usp=sharing)

## 12. References

- [1] React Documentation. [Online]. Available: <https://react.dev>
- [2] Node.js Documentation. [Online]. Available: <https://nodejs.org/en/docs>
- [3] MongoDB Atlas Documentation. [Online]. Available: <https://www.mongodb.com/docs/atlas>
- [4] Express.js Guide. [Online]. Available: <https://expressjs.com>
- [5] Vite Documentation. [Online]. Available: <https://vitejs.dev>
- [6] Tailwind CSS Docs. [Online]. Available: <https://tailwindcss.com/docs>
- [7] React Hook Form. [Online]. Available: <https://react-hook-form.com>
- [8] Yup Schema Builder. [Online]. Available: <https://github.com/jquense/yup>
- [9] React Markdown. [Online]. Available: <https://github.com/remarkjs/react-markdown>
- [10] html-docx-js - Convert HTML to Word. [Online]. Available: <https://www.npmjs.com/package/html-docx-js>
- [11] Together.ai API Documentation. [Online]. Available: <https://docs.together.ai/docs>
- [12] OpenAI API Documentation. [Online]. Available: <https://platform.openai.com/docs>
- [13] Nginx Reverse Proxy Configuration. [Online]. Available: <https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy>
- [14] Docker Documentation. [Online]. Available: <https://docs.docker.com>
- [15] Amazon Web Services - EC2 Documentation. [Online]. Available: <https://docs.aws.amazon.com/ec2>
- [16] Ansible Playbooks Documentation. [Online]. Available: [https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html)
- [17] OWASP Cheat Sheet Series – Security Best Practices. [Online]. Available: <https://cheatsheetseries.owasp.org>