CS 520: Project 3: Final Deliverables

Part 1: Project Document

Group Members:

- **Ruchi Gupta, GitHub username: RuchiGupta20**
- **Manan Parikh, GitHub username: MananParikh**
- **Manas Madine, GitHub username: manas1999**
- **Priya Balakrishnan, GitHub username: Bpriya42**

Team Name: AutoCareers (Team21_SelfIdea)

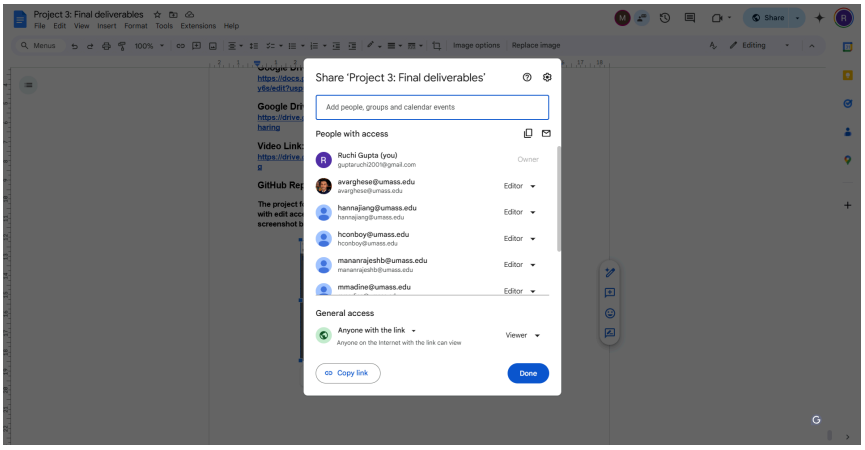
Google Drive Project Document Link:
https://docs.google.com/document/d/16fk7FtPd1zUE_PAu6K4nnltt321TX8ckowcbbi9xy6s/edit?usp=sharing

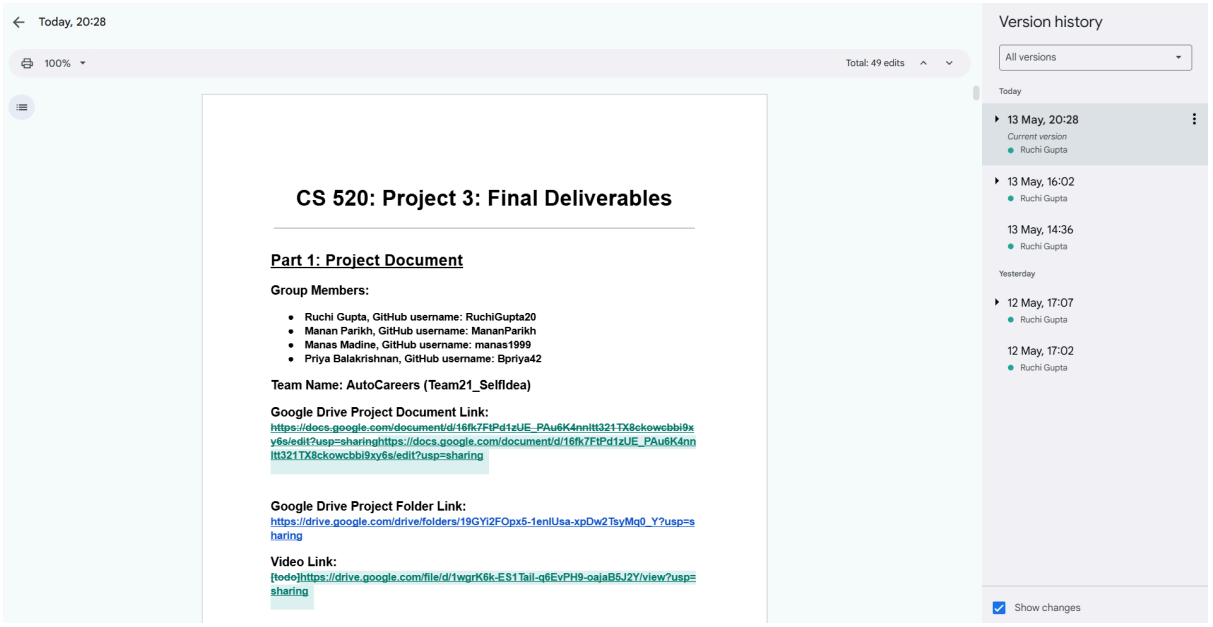
Google Drive Project Folder Link:
https://drive.google.com/drive/folders/19GYi2FOpx5-1enlUsa-xpDw2TsyMq0_Y?usp=sharing

Video Link:
https://drive.google.com/file/d/1wgrK6k-ES1Tail-q6EvPH9-oajaB5J2Y/view?usp=sharing

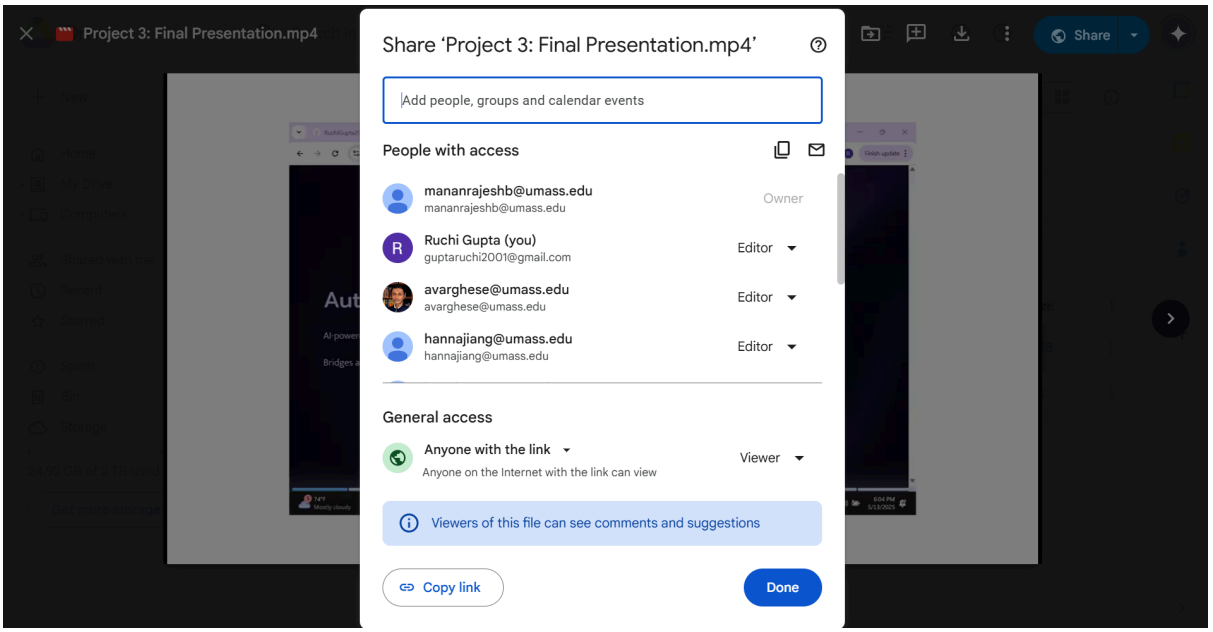
GitHub Repo Link: https://github.com/RuchiGupta20/AutoCareers

The project folder and document have been shared with the professor and all TAs, with edit access granted to everyone (version control is visible), as shown in the screenshot below.

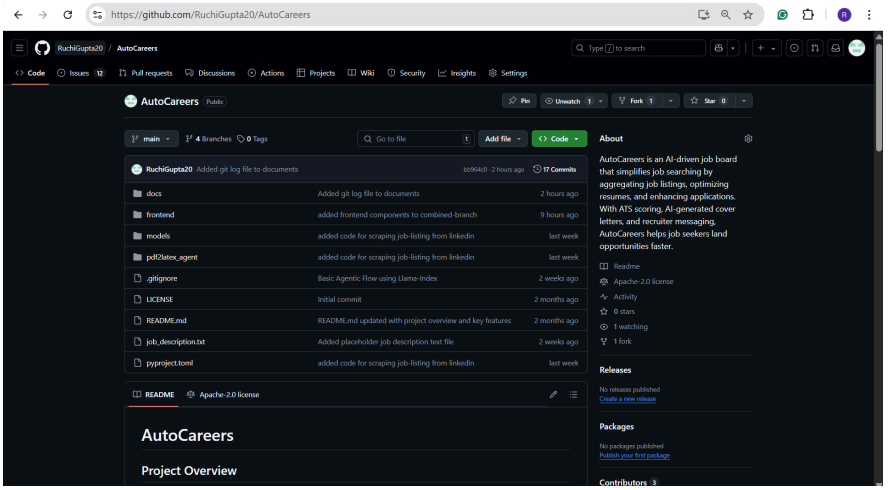




The video presentation file is in the Project 3 Folder and has been shared with the professor and all TAs, with edit access granted to everyone, as shown in the screenshot below.



The github folder is public and hence the version control history and all folders/code is accessible to the professor and all TAs as shown in the screenshot below.


https://github.com/RuchiGupta20/AutoCareers
RuchiGupta20 / AutoCareers
Type / to search
Code
Issues 12
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
AutoCareers Public
Pin
Unwatch 1
Fork 1
Star 0
main
4 Branches
0 Tags
Go to file
t
Add file
Code
RuchiGupta20 Added git log file to documents
b6964c0 · 2 hours ago
17 Commits
docs
Added git log file to documents
2 hours ago
frontend
added frontend components to combined-branch
9 hours ago
models
added code for scraping job-listing from linkedin
last week
pdf2latex_agent
added code for scraping job-listing from linkedin
last week
.gitignore
Basic Agentic Flow using Llama-Index
2 weeks ago
LICENSE
Initial commit
2 months ago
README.md
README.md updated with project overview and key features
2 months ago
job_description.txt
Added placeholder job description text file
2 weeks ago
pyproject.toml
added code for scraping job-listing from linkedin
last week
README
Apache-2.0 license
AutoCareers
Project Overview
About
AutoCareers is an AI-driven job board that simplifies job searching by aggregating job listings, optimizing resumes, and enhancing applications. With ATS scoring, AI-generated cover letters, and recruiter messaging, AutoCareers helps job seekers land opportunities faster.
Readme
Apache-2.0 license
Activity
0 stars
1 watching
1 fork
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Contributors 3

Question 1: Requirements

Based on the feedback received on certain parts of the requirements section in our Project 1: Initial Project Deliverables submission, we have made the necessary revisions. The remaining sections have been kept unchanged, and the revised sections include a note indicating the changes made to incorporate the feedback.

Project Overview:

AutoCareers is designed to streamline the job search process for job seekers and recruiters by providing an intuitive platform for job listings, applications, and candidate management. The platform will offer advanced features such as resume parsing, ATS scoring, filtering and search, recruiter and user-specific views, direct messaging, and job applications. By leveraging modern web technologies, AI-driven resume analysis, and automation, the platform aims to improve hiring efficiency and enhance the job-seeking experience.

Project Purpose:

The purpose of this project is to bridge the gap between job seekers and recruiters by offering a **user-friendly, data-driven, and automated job search platform**. The website will **enhance job search efficiency** by helping applicants optimize their resumes, filter jobs based on their preferences, and even apply to jobs with customized cover letters. For recruiters, it will provide **tools to manage applicants, filter resumes based on ATS scores, and communicate efficiently** with potential hires.

Problem Statement:

Traditional job search platforms often suffer from **information overload, inefficient matching, and poor communication between job seekers and recruiters**. Candidates struggle to find relevant job opportunities, optimize their resumes for ATS systems, and track applications effectively. Recruiters, on the other hand, face difficulties in filtering qualified candidates, managing multiple applications, and ensuring smooth communication. This project aims to **create a modern, AI-enhanced job board that simplifies the hiring process, making it faster and more effective for both job seekers and recruiters**.

Target Users:

- **Job Seekers:** Individuals looking for jobs, internships, or career opportunities who want an efficient way to apply, track applications, and optimize their resumes.
- **Recruiters & Hiring Managers:** Employers and HR professionals looking for an easy way to view profiles, filter candidates, and manage applications.
- **Career Coaches & Resume Experts:** Professionals who assist job seekers with resume optimization and job search strategies.
- **Students & Recent Graduates:** Entry-level professionals seeking internships and first-time job opportunities.

Features:

We received valuable feedback on the first three feature points- User Registration & Authentication, Role Management, and Job Listing Aggregation- and have incorporated the suggested changes into the updated version. Additionally, we elaborated on the remaining feature points to better capture their scope and functionality.

- **User Registration & Authentication**
  Our platform aims to seamlessly handle User Registration & Authentication. For new users, the platform enables job seekers and recruiters to create accounts and sign up using either an email/password combination or third-party authentication options, such as Google, implemented via the Google OAuth API. During the registration process, users are required to verify their email to ensure account authenticity. Passwords are securely encrypted before storage to protect user data. After successful registration, users can log in using the same email/password or their Google credentials. The system ensures secure and persistent account access, allowing users to manage their profile, applications, and other platform features once logged in. A "Forgot Password" functionality is provided to help users securely reset their passwords if needed. Both login and logout processes are designed to function smoothly, offering a hassle-free and secure experience to all users.

- **Role Management**
  The platform implements a comprehensive role management system consisting of three roles: Job Seeker, Recruiter, and System Administrator. Each role is associated with distinct permissions and access levels. Job Seekers can search for jobs, apply, manage resumes, and track their applications. Recruiters have access to features for posting job listings, managing applicants, and communicating with candidates. The System Administrator role oversees the platform's health, manages users, and has full access to all features for moderation and maintenance purposes. Role-based access control is enforced throughout the system to ensure that each user interacts only with functionalities permitted by their role.

- **Job Listing Aggregation**
  AutoCareers automatically aggregates job postings from multiple external sources using a combination of REST APIs and web scraping techniques. This ensures that job listings are kept fresh and up to date. The aggregation process runs periodically to update the job database. Deduplication logic is implemented to eliminate duplicate postings originating from multiple sources. During aggregation, detailed information such as role information, salary information, experience requirements, company details, job location, employment type, and required skills are scraped from platforms like LinkedIn, Indeed, and GitHub job pages. Each job posting is automatically associated with these tags and stored in a structured format in the MongoDB database, facilitating efficient search, filtering, and recommendation features within the platform.

- **Job Search and Filtering**
  The platform provides a robust search and filtering system, allowing users to search

for job postings using keywords while refining results with multiple filters, such as location, job type, company, experience level, industry, and salary range. Users can sort job listings by relevance, date posted, or company name to better align with their preferences. The system dynamically updates search results without requiring full page reloads to enhance user experience. For recruiters, similar filtering capabilities are available for browsing applicant profiles based on criteria like skills, qualifications, and experience.

- **Resume Uploading**
  Job seekers can upload resumes directly into their profiles in commonly used formats such as PDF and LaTeX. The system performs validations on the uploaded file format and structure to ensure compatibility with other modules, including resume parsing and ATS scoring. Users can easily update or replace their resumes whenever necessary. All uploaded resumes are securely stored and linked to the user's account, enabling seamless access during the job application process. In future releases, if time permits, we plan to introduce resume versioning to allow users to maintain and manage multiple resume variations tailored to different job roles.

- **ATS Scoring and Resume Parsing**
  AutoCareers integrates an AI-powered resume parser that automatically extracts key information such as education, skills, work experience, and certifications from uploaded resumes. The extracted data is used to compute an ATS-friendliness score that measures how well a resume aligns with the requirements of job postings. The system highlights areas that require attention and provides actionable recommendations to improve resume content and structure. Users can view the parsed data and ATS score directly on their profile and make updates accordingly to boost their chances of being shortlisted by recruiters.

- **Resume Optimization and Editing**
  The platform provides AI-assisted resume advice that offers suggestions for improving the structure, formatting, and keyword usage within resumes. Users receive tailored recommendations based on common ATS patterns and industry best practices. The editor helps enhance readability and ensures that resumes are optimized for automated screening systems. Users can incorporate these suggested changes and use the improved version for use in job applications.

- **Recruiter Job Management**
  Recruiters are equipped with tools to manage job postings and interact with applicants effectively. The recruiter dashboard allows recruiters to manage job listings, review applicant submissions, view detailed candidate profiles, and track ATS scores and resume analytics. Recruiters can apply advanced filters to sort candidates based on job-specific criteria and manage the application pipeline efficiently.

- **Messaging System**
  The platform supports a built-in messaging system that facilitates direct communication between job seekers and recruiters. Recruiters can initiate conversations with applicants, and both parties can exchange messages for interview

scheduling, feedback, and general inquiries. The system maintains a complete conversation history for each interaction. Future versions, if time permits for reach goals, will consider additional features like file attachments, read receipts, push notifications, and message templates to streamline recruiter-applicant interactions further.

- **Automatic Cover Letter Generator**
  Leveraging AI using LLMs like OpenAI and TogetherAI, the platform automatically generates tailored cover letters for job seekers. The system analyzes the job description, user profile, and uploaded resume to draft a cover letter that matches the application requirements. This feature significantly reduces the time and effort required to create professional and effective cover letters, improving the quality of applications submitted by users.

- **Security and Data Protection**
  AutoCareers incorporates robust security measures to safeguard sensitive user data and ensure system integrity. All communication is secured via HTTPS, and authentication is handled using OAuth 2.0 protocols. Sensitive data, including passwords and personal information, is securely encrypted both at rest and in transit. The system enforces strict role-based access control. AutoCareers adheres to general data protection guidelines, promoting trust and ensuring compliance with data privacy standards.

Reach Goals (If Time Permits, We May Add)

- **Automated Job Applications**
  Users can enable automatic applications for jobs matching their preferences, streamlining the application process.
- **Resume Editing and Optimization**
  AI-powered automatic resume editor providing formatting suggestions, keyword optimization, and ATS improvements.
- **User Profile and Application Tracking**
  Dashboard for users to manage their profiles, view saved jobs, track application status, and receive job recommendations.

Functional Requirements (Use cases):

We received feedback on Feature 6 to provide more details about the nature and presentation of the feedback given to users, which has now been addressed thoroughly in the description and related sections. Additionally, we were advised to separate Feature 7 into distinct login and logout functionalities, which we have incorporated as Functional Requirements 7.1 and 7.2 for improved clarity and understanding. The remaining functionalities, for which no specific feedback was provided, have been retained as originally designed.

Functional Requirement 1

Field	Description
Title	AI-based Suggestions for Resume Formatting and Content
Goal	Provide AI-driven insights and recommendations to improve resume formatting, content, and overall presentation for better job application outcomes and ATS compatibility.
Scope	AutoCareers Platform, Resume Enhancement Module
Level	User-Level
Actors	Primary Actor: Job Seeker (User) Secondary Actors: AI Resume Assistant, Resume Storage System, User Profile Database
Description	The system analyzes the user's uploaded resume and profile data using an AI-based engine. It provides tailored suggestions on formatting, content improvements, keyword optimization, and layout adjustments to enhance readability and ATS performance.
Trigger	User clicks the "Get Resume Suggestions" button within the resume or profile section.
Preconditions	- User must be logged in. - A resume must be uploaded in a supported format (.docx, .pdf). - The AI Resume Assistant is operational and integrated with the system.
Main Success Scenario (Goals)	- User logs into their account. - User navigates to the resume enhancement section and selects "Get Resume Suggestions". - System retrieves the user's resume and profile data. - The AI engine analyzes the resume for formatting and content

	improvements. - System displays the AI-generated suggestions. - User reviews and optionally applies the suggestions.
Alternate Success Scenarios	- User requests a detailed breakdown of suggestions for specific sections (e.g., header, work experience, skills). - User saves the suggestions for later review or applies them automatically.
Postconditions	- The AI-generated suggestions are displayed and stored in the system. - User can review and update their resume based on the recommendations.
Main Flow & Steps of Execution (Requirements)	1. User logs into their account. 2. User navigates to the resume enhancement module. 3. User clicks "Get Resume Suggestions". 4. System retrieves the uploaded resume and profile data. 5. AI engine analyzes the resume for formatting, content, and keyword usage. 6. System generates and displays the suggestions for improvement. 7. User reviews and optionally applies the changes. 8. System confirms and saves any updates made by the user.
Extensions	- Option to compare the original resume with the suggested version side-by-side. - Downloadable report detailing the AI analysis and recommendations. - Industry-specific suggestions based on the targeted job roles. (Reach Goal)
Failed End Condition	- Resume suggestions are not displayed due to AI engine failure or incomplete resume data. - User's resume is in an unsupported format or is missing.
Exceptions	- Network or server errors during data retrieval or processing. - AI engine encounters errors or timeouts during analysis.
Variations	- User accesses suggestions for multiple resumes if available. - System offers periodic updates on resume suggestions based on new industry trends. (Reach Goal)

Functional Requirement 2

Field	Description
Title	Job Recommendations Based on User Profile and Resume
Goal	Provide personalized job recommendations by analyzing the user's profile, resume, and job activity to help them discover relevant opportunities.
Scope	AutoCareers Platform, Job Recommendation Engine
Level	User-Level
Actors	Primary Actor: Job Seeker (User) Secondary Actors: Recommendation Engine, Resume Storage System, User Profile Database
Description	The system analyzes data from the user's profile, resume, and past interactions to generate tailored job recommendations. These recommendations are dynamically updated and presented on the user's dashboard or a dedicated recommendations page, ensuring a personalized job search experience.
Trigger	User logs in and navigates to the dashboard or job recommendations page.
Preconditions	- User must be logged in. - User profile and resume must be complete and stored in the system. - The recommendation engine is operational with access to up-to-date job postings.

Main Success Scenario (Goals)	- User logs into their account. - System retrieves the user's profile and resume data. - The recommendation engine processes the data to generate personalized job suggestions. - The system displays the recommended job listings on the dashboard or recommendations page. - User reviews the recommendations and navigates to job details or applies.
Alternate Success Scenarios	- User receives additional recommendations based on recent job searches or filters. - User manually adjusts preferences to refine the recommendation results.
Postconditions	- The system displays a list of job recommendations tailored to the user’s profile and resume. - Recommendations are updated dynamically as the user's data changes.
Main Flow & Steps of Execution (Requirements)	1. User logs into their account. 2. System retrieves the user's profile and resume information. 3. Recommendation engine analyzes the data against current job postings. 4. System generates a list of personalized job recommendations. 5. Recommendations are displayed on the user's dashboard or recommendations page. 6. User clicks on a recommendation to view details or apply.
Extensions	- Provide filtering and sorting options (e.g., relevance, date, location). - Integrate machine learning algorithms for continuous recommendation refinement. (Reach Goal)
Failed End Condition	- No recommendations are displayed due to incomplete user data or system errors. - The recommendation engine fails to match jobs with the user's profile.
Exceptions	- Network or system errors prevent retrieval of user data or job postings. - Inconsistencies in the user data result in inaccurate recommendations.

Variations	- User accesses job recommendations via mobile or web dashboard. (Reach Goal)

Functional Requirement 3

Field	Description
Title	Automatic Cover Letter Generator
Goal	Enable users to quickly create personalized cover letters for job applications by leveraging AI-driven analysis of job details and their resume/profile.
Scope	AutoCareers Platform, Cover Letter Generation Module
Level	User-Level
Actors	Primary Actor: Job Seeker (User) Secondary Actors: Cover Letter Generator Engine, Resume Storage System.
Description	The system integrates with job postings and the user's profile. When a user views a job posting, they can select an option to generate a cover letter. The AI-based engine extracts key details from the job description and combines them with information from the user's resume/profile to create a draft cover letter. The user can then review, customize, and save the generated cover letter for use in their job application.
Trigger	User clicks on the "Generate Cover Letter" button on a job posting page or within their profile.

Preconditions	- User must be logged in. - User must have an uploaded resume or a complete profile. - The job description must be available in the system. - The cover letter generation engine is operational.
Main Success Scenario (Goals)	- User logs into their account and navigates to a job posting. - User selects the "Generate Cover Letter" option. - System verifies the availability of the user's resume/profile and job details. - The AI engine processes the job description and user data to generate a draft cover letter. - User reviews, customizes, and saves the finalized cover letter.
Alternate Success Scenarios	- User requests a re-generation with an alternate tone or additional personalization. - User manually edits the generated cover letter before finalizing. - User saves multiple versions for different job applications.
Postconditions	- The generated cover letter is displayed in the user’s profile. - The cover letter is stored securely and can be attached to job applications. - The user can return later to modify or re-generate the cover letter.
Main Flow & Steps of Execution (Requirements)	1. User logs into their account. 2. User navigates to a job posting or the cover letter generation section within their profile. 3. User clicks the "Generate Cover Letter" button. 4. System checks that the user’s resume/profile and job description are available. 5. System retrieves the relevant job details and user data. 6. The over letter generation engine processes the data to create a draft cover letter. 7. System displays the generated cover letter for user review. 8. User reviews and optionally customizes the cover letter. 9. User confirms the cover letter, and the system saves it to their profile.
Extensions	- Provide options to select the cover letter tone (e.g., formal, casual). - Allow users to incorporate personalized achievements or additional notes. - Offer real-time ATS optimization feedback on the generated cover letter.

Failed End Condition	- User attempts to generate a cover letter without an uploaded resume or a complete profile. - Job description data is missing or incomplete. - The AI engine fails to generate a cover letter due to processing errors.
Exceptions	- Network or server errors during data retrieval or processing. - The generated cover letter contains formatting issues or irrelevant content. - System experiences delays or timeouts during the generation process.
Variations	- User accesses the cover letter generator directly from their profile dashboard. - System allows manual editing of the generated cover letter post-generation.

Functional Requirement 4

Field	Description
Title	Browse/Filter Jobs by Category, Company, Type, Location, Salary Range, Experience Level, etc.
Goal	Allow logged-in users to explore job listings and refine search results using filters to find relevant job opportunities.
Scope	AutoCareers Platform, Job Board Page
Level	User-Level
Actors	Primary Actor: Job Seeker (User) Secondary Actors: Job Scraping System, Job Board Database

Description	The system scrapes and aggregates job listings from various sources and stores them in the database. Logged-in users can navigate to the Job Board, where they can browse jobs or apply filters such as category, company, job type, location, salary range, and experience level to refine their search. The system dynamically updates the results based on selected filters, ensuring a personalized job search experience.
Trigger	User logs in and navigates to the job board.
Preconditions	- User must be logged in to access job listings. - The system must have successfully scraped and stored job listings in the database.
Main Success Scenario (Goals)	- User successfully logs in. - User accesses the job board and views a list of job postings. - User applies filters (e.g., company, location, salary range) and sees updated search results. - User finds relevant job opportunities efficiently.
Alternate Success Scenarios	- User resets filters to view all available jobs. - User bookmarks or saves jobs for later viewing. (Reach Goals)
Postconditions	- The system displays job listings based on the user's selected filters. - User can navigate between job postings, view job descriptions, and proceed to applications.
Main Flow & Steps of Execution (Requirements)	1. User logs into their account. 2. User navigates to the Job Board.

	3. System checks login status and grants access. 4. System displays the latest job postings. 5. User applies filters (e.g., company, job type, location, salary range). 6. System dynamically updates job listings based on selected filters. 7. User clicks on a job posting to view details. 8. (Optional) User saves a job for later or applies immediately.
Extensions	- Implement keyword search for job titles and descriptions.
Failed End Condition	- User is not logged in and is redirected to the login page when trying to access the job board. - No job listings are displayed due to scraping failure or an empty database. - Filters return zero results, indicating no matching jobs.
Exceptions	- System fails to fetch the latest job postings due to an API failure or connectivity issue. - User selects conflicting filters (e.g., "Entry-level" + "10+ years experience") leading to no results.
Variations	- User switches between list view and grid view for job postings. - User sorts results by date posted, relevance, or salary range. (Reach Goals)

Functional Requirement 5

Field	Description

Title	Upload Resumes in Different Formats
Goal	Allow users to upload their resume in supported formats (.docx, .pdf) for job applications and ATS scoring.
Scope	AutoCareers Platform, User Profile View
Level	User-Level
Actors	Primary Actor: Job Seeker (User) Secondary Actors: Resume Storage System
Description	The user navigates to their profile and uploads a resume in either .docx or .pdf format. The system validates the file format, stores the resume, and updates the user's profile. Only one resume can be uploaded at a time. If the user uploads a new resume, it will replace the previous one.
Trigger	User clicks the "Upload Resume" button and selects a file.
Preconditions	- User must be logged into their account. - The resume must be in a supported format (.docx or .pdf). - The system should have sufficient storage capacity.
Main Success Scenario (Goals)	- The user successfully uploads their resume. - The system validates and stores the file. - The user's profile reflects the uploaded resume.

Alternate Success Scenarios	- User replaces an existing resume with a new upload.
Postconditions	- The resume is stored securely in the system. - The system updates the profile to reflect the uploaded file.
Main Flow & Steps of Execution (Requirements)	1. User navigates to their profile. 2. User clicks the "Upload Resume" button. 3. System prompts the user to select a file. 4. User selects a .docx or .pdf file and submits it. 5. System validates the file format and size. 6. System stores the resume and updates the user's profile. 7. If a resume already exists, the new file replaces the old one. 8. User receives a confirmation message.
Extensions	- Implement a resume preview feature before uploading. (Reach Goals) - Allow users to store multiple resumes for different job applications.
Failed End Condition	- Resume upload fails due to an unsupported file format. - File exceeds the maximum allowed size. - System error prevents the file from being stored.
Exceptions	- User selects an invalid file type (e.g., .txt, .jpg), and the system displays an error message. - Network interruption causes the upload to fail.

Variations	- User uploads a resume and later deletes it instead of replacing it. - System provides feedback on resume formatting before upload. (Reach Goals)

Functional Requirement 6

Field	Description
Title	Provide an ATS-Friendliness Score
Goal	Allow users to assess how well their resume is optimized for Applicant Tracking Systems (ATS) and receive actionable, detailed feedback to improve their chances of passing automated screenings.
Scope	AutoCareers Platform, User Profile View
Level	User-Level
Actors	Primary Actor: Job Seeker (User) Secondary Actors: ATS Scoring System (AI Model), Resume Storage System
Description	Users can request an ATS-friendliness score for their resume directly from their profile. Upon clicking the "Provide ATS Score" button, the system processes the resume and generates a comprehensive score, breaking it down into key components such as formatting quality, keyword matching, readability, section completeness, and overall structure. The user receives detailed feedback in the form of actionable suggestions (e.g., "Include missing skills section," "Add more industry-specific keywords,"

	"Reformat the work experience section for ATS readability"). If no resume is uploaded, the system will prompt the user to upload one before scoring. Users are encouraged to revise their resumes based on the feedback and can re-run the scoring process multiple times. The feedback will be displayed as a structured report directly in the user's profile for easy reference.
Trigger	User clicks on the "Provide ATS Score" button.
Preconditions	- User must be logged into their account. - Resume must be uploaded (or user must upload it before proceeding). - System should have ATS scoring functionality enabled.
Main Success Scenario (Goals)	- The system successfully processes the uploaded resume. - An ATS-friendliness score is generated. - The user receives actionable feedback on how to improve their resume for ATS compatibility.
Alternate Success Scenarios	- User updates and re-uploads their resume based on the feedback and requests a new ATS score.
Postconditions	- The user sees their ATS score and feedback in their profile. - The system logs the result for tracking and future recommendations.
Main Flow & Steps of Execution (Requirements)	1. User navigates to their profile. 2. System checks if a resume is available. 3. If not, the system prompts the user to upload a resume. 4. User clicks the "Provide ATS Score" button.

	5. The system processes the resume and analyzes it for ATS compatibility. 6. The system generates a score along with a breakdown into key areas (formatting, keyword matching, readability, structure, completeness). 7. Actionable feedback is presented to the user. 8. User may update their resume and re-run the scoring process.
Extensions	- Provide additional recommendations based on job descriptions. - Allow users to compare multiple resumes for different job applications.(Reach Goals)
Failed End Condition	- Resume fails to upload due to format or size restrictions. - ATS scoring system encounters an error and cannot generate a score.
Exceptions	- System times out due to high server load. - Uploaded file is corrupted or in an unsupported format.
Variations	- User uploads multiple resumes and selects one for ATS scoring. (Reach Goals) - System provides a breakdown of the ATS score (e.g., formatting issues, missing keywords, readability)

Functional Requirement 7.1

Field	Description
Title	User Login

Goal	Allow users (job seekers and recruiters) to securely log into their accounts.
Scope	AutoCareers Platform, User Authentication System
Level	User-Level
Actors	● **Primary Actor:** User (Job Seeker, Recruiter) ● **Secondary Actors:** Authentication System, Database
Description	Users can log into their accounts using either their registered email and password or third-party authentication (Google, LinkedIn). The system verifies user credentials and grants access to the platform. Incorrect login attempts trigger error messages. Upon successful login, users are redirected to their dashboard.
Trigger	User navigates to the login page and enters credentials or selects third-party login.
Preconditions	- The user must have an existing account. - The authentication system must be operational. - Valid credentials must be provided.
Main Success Scenario (Goals)	- The user successfully logs in and is redirected to their dashboard. - A secure session is established.
Alternate Success Scenarios	- The user logs in via third-party authentication (Google, LinkedIn).

	- The user resets their password using the "Forgot Password" functionality and logs in successfully.
Postconditions	- The user is securely logged in and can access system features based on their role.
Main Flow & Steps of Execution (Requirements)	1. User navigates to the login page. 2. User enters email and password or selects third-party login. 3. The system verifies credentials. 4. If valid, access is granted and the dashboard is displayed. 5. If invalid, an error message is shown. 6. After multiple failed attempts, CAPTCHA or account lockout is triggered.
Extensions	- Multi-factor authentication (MFA) for enhanced security. - "Remember Me" functionality. - Biometric login support (Reach Goal).
Failed End Condition	- Multiple incorrect login attempts cause account lockout. - Authentication service is unavailable.
Exceptions	- The user forgets their password and must use the recovery process. - Third-party authentication is temporarily unavailable.
Variations	User logs in from multiple devices. (Reach Goals)

Functional Requirement 7.2

Field	Description
Title	User Logout
Goal	Allow users to securely terminate their sessions and log out of their accounts.
Scope	AutoCareers Platform, User Authentication System
Level	User-Level
Actors	• **Primary Actor:** User (Job Seeker, Recruiter) • **Secondary Actors:** Authentication System, Database
Description	Users can securely log out from their accounts by clicking the "Log Out" button. The system terminates the session, clears session data, and redirects users to the homepage. This ensures that their account information is protected, especially on shared or public devices.
Trigger	The user clicks the "Log Out" button.
Preconditions	The user must be logged in.
Main Success Scenario (Goals)	- The user clicks "Log Out." - The system terminates the session and redirects to the homepage.
Alternate Success Scenarios	- None.

Postconditions	- Session is terminated securely, and the user is redirected to the homepage.
Main Flow & Steps of Execution (Requirements)	1. The user clicks the "Log Out" button. 2. The system terminates the session. 3. The user is redirected to the homepage.
Extensions	- Display a confirmation message before logout (optional).
Failed End Condition	- The session is not terminated properly, posing potential security risks.
Exceptions	- Network errors prevent session termination; the system retries or shows an appropriate error message.
Variations	User logs out from multiple devices. (Reach Goals)

Functional Requirement 8

Field	Description
Title	Forgot Password
Goal	Allow users to reset their password securely if they forget it.
Scope	AutoCareers Platform, User Authentication System

Level	User-Level
Actors	● **Primary Actor:** User (Job Seeker, Recruiter) ● **Secondary Actors:** Authentication System, Email Service
Description	Users who forget their password can request a password reset. They enter their registered email address, and the system sends a password reset link. Upon clicking the link, users are redirected to a secure page where they set a new password. The system verifies password requirements before updating the user's credentials.
Trigger	User clicks on the "Forgot Password" link on the login page.
Preconditions	● User must have an existing account. ● System must have a functioning email service to send the reset link. ● The entered email must match a registered account.
Main Success Scenario (Goals)	● User successfully requests a password reset link. ● User receives an email with a secure reset link. ● User clicks the link, sets a new password, and logs in successfully.
Alternate Success Scenarios	User updates their password successfully but decides to change it again later via account settings.
Postconditions	● The system securely updates the user's password. ● User can log in with the new password.
Main Flow & Steps of Execution (Requirements)	● User navigates to the login page and clicks "Forgot Password." ● System prompts the user to enter their registered email.

	● User submits their email address. ● System verifies if the email exists in the database. ● If valid, system generates a secure password reset link and sends it via email. ● User receives the email and clicks the reset link. ● System verifies the link and prompts the user to set a new password. ● User enters and confirms the new password. ● System updates the password and notifies the user of the successful reset. ● User logs in with the new password.
Extensions	● Implement additional security measures like security questions or MFA verification before resetting the password. ● Allow users to request a reset via SMS for mobile authentication. (Reach Goals)
Failed End Condition	● User enters an unregistered email, and the system does not send a reset link. ● User does not receive the reset email due to system failure or spam filters. ● Reset link expires before the user changes their password.
Exceptions	Email service is down, preventing the system from sending the reset link. User mistypes their email, leading to no reset link being sent. User clicks an expired or invalid reset link, requiring a new request.
Variations	User changes their password through account settings instead of the "Forgot Password" process.

Functional Requirement 9

Field	Description
Title	Create a New Profile / Sign Up or Register
Goal	Allow new users to register and create an account to access the system.
Scope	AutoCareers Platform, User Authentication System
Level	User-Level
Actors	• **Primary Actor:** New User (Job Seeker, Recruiter) • **Secondary Actors:** Authentication System, Database
Description	Users can create a new account by providing their personal details (name, email, password) and selecting their role (job seeker or recruiter). The system validates the provided information and creates a new user profile. An email verification step may be required to confirm registration. Once verified, users gain access to the platform and can complete their profile.
Trigger	User clicks on the "Sign Up" or "Register" button on the homepage.
Preconditions	• User must provide a unique email address that is not already registered. • Password must meet security requirements. • System must have an operational database to store user details.
Main Success Scenario (Goals)	• User successfully registers and creates an account. • System stores user details securely.

	• User logs in and accesses their dashboard.
Alternate Success Scenarios	• User signs up using third-party authentication (Google, LinkedIn). • User completes email verification later and logs in successfully.
Postconditions	• The system stores user credentials securely. • User profile is created and ready for additional setup (e.g., resume upload).
Main Flow & Steps of Execution (Requirements)	• User clicks "Sign Up" or "Register" on the homepage. • System prompts the user to enter their name, email, password, and role (job seeker or recruiter). • User enters the required details and submits the form. • System validates the input (e.g., email uniqueness, password strength). • If validation passes, the system creates a new user profile. • (Optional) System sends a verification email to confirm the registration. • User verifies their email by clicking the confirmation link. • User logs in for the first time and is redirected to the dashboard.
Extensions	• Allow users to sign up using social authentication (Google, LinkedIn). • Enable phone number verification as an alternative to email verification. (Reach Goals)
Failed End Condition	• User enters an email already associated with an existing account. • System validation fails due to an invalid email or weak password. • Email verification is not completed, restricting account access.

Exceptions	● User enters a temporary or invalid email address, preventing verification. ● Network or system failure interrupts the sign-up process. ● User forgets their password immediately and needs to reset it.
Variations	● User creates an account but does not complete their profile setup immediately. ● User registers but does not verify their email, resulting in limited access. (Reach Goals)

Functional Requirement 10

Field	Description
Title	See Job Details and Requirements
Goal	Allow logged-in users to view detailed job descriptions, including requirements, responsibilities, and application instructions.
Scope	AutoCareers Platform
Level	User-Level
Actors	Primary Actor: Job Seeker (User) Secondary Actors: Job Board Database, Job Scraping System
Description	The user logs into their account and navigates to the job board. They browse job listings and select a job to view its details. The system retrieves and displays job-specific information, including job title, company name, location, salary range, job type,

	responsibilities, required qualifications, and an "Apply Now" option.
Trigger	User clicks on a job listing.
Preconditions	- User must be logged in to access job details. - The system must have stored job listings with complete job details.
Main Success Scenario (Goals)	- User successfully logs in and accesses the job board. - User clicks on a job posting to view detailed job information. - System retrieves and displays job details, including requirements and responsibilities.
Alternate Success Scenarios	- User saves the job for later review. - User shares the job link with others.
Postconditions	- The user gains access to all relevant job details. - The system tracks user engagement with job postings for recommendations.
Main Flow & Steps of Execution (Requirements)	1. User logs into their account. 2. User navigates to the Job Board and browses job listings. 3. User clicks on a job to view details. 4. System retrieves the job information from the database. 5. System displays job title, company name, location, salary range, job type, responsibilities, required qualifications, and "Apply Now" option. 6. (Optional) User saves or shares the job.

Extensions	- Allow users to see company reviews and ratings alongside job details.
Failed End Condition	- User is not logged in and is redirected to the login page when trying to view job details. - Job details fail to load due to a missing or corrupted database entry.
Exceptions	- System is unable to fetch job details due to an API failure or data inconsistency. - Job listing is no longer available, and the system displays an "Expired Job" message.
Variations	- User views job details in a modal popup instead of a separate page. - System provides a "Similar Jobs" section based on the current job listing.

Functional Requirement 11

Field	Description
Title	Review and search applicant profiles and resumes (Recruiter View)
Goal	Recruiters should be able to view all public applicant profiles, search for candidates, and access their resumes.
Scope	AutoCareers Platform, Recruiter View

Level	User
Actors	Primary Actor: Recruiter (User) Secondary Actor: Job Applicant (User) – Public profiles must be available for this use case to function.
Description	Job applicants can choose to make their profiles public through a visibility toggle in their profile settings. If an applicant sets their profile to public, all registered recruiters will be able to view a list of available candidates. Recruiters can browse the list or use search functionality to filter candidates by name, job role, skills, or other relevant attributes.
Trigger	This use case is triggered when a recruiter clicks on the ‘View Candidates’ page.
Preconditions	- The recruiter must be logged in with the appropriate role-based access. - Job applicants must set their profile to public, otherwise, recruiters cannot find them.
Main Success Scenario (Goals)	- Recruiters can see a list of all public applicant profiles along with their resumes.
Alternate Success Scenarios	- Recruiters message the candidate for further steps.
Postconditions	- Recruiters can initiate the hiring process by contacting shortlisted candidates through messaging.

Main Flow & Steps of Execution (Requirements)	**Profile Visibility Option**: ● Job applicants have an option in their profile settings to toggle between **public** and **private** visibility. **Displaying Public Profiles**: ● If an applicant sets their profile to **public**, their profile details (name, job role, skills, etc.) become visible to all registered recruiters. ● A dedicated section for recruiters lists all available public profiles. **Search Functionality for Recruiters**: ● Recruiters can browse the list of public candidates. ● They can search for candidates using filters such as **name**, **job role**, **skills**, or other relevant attributes. 1. Recruiter logs into their account. 2. Recruiter navigates to the Applicant Management Dashboard. 3. System displays all available applicants. 6. Recruiter selects an applicant to view the full profile. 7. System displays applicant details, including resume and ATS score.
Extensions	Recruiter saves, downloads, or contacts the applicant. (Reach Goal)
Failed End Condition	- Even if job applicants have set their profile to public, some or all profiles are not visible to recruiters. - Recruiters are unable to search for specific profiles by name, job role, or skills.
Exceptions	**Profile Visibility Toggle Fails** ● **Scenario**: A job applicant tries to change their profile visibility but the update fails due to a server issue or network failure.

	● **Handling**: ○ Show an error message: *"Failed to update profile visibility. Please try again."* ○ Log the failure for debugging. **Recruiter Search Returns No Results** ● **Scenario**: A recruiter searches for a candidate but no profiles match the criteria. ● **Handling**: ○ Display a message: *"No candidates found matching your criteria. Try adjusting your search filters."* **Unauthorized Access** ● **Scenario**: A recruiter who is not registered or logged in attempts to view public profiles. ● **Handling**: ○ Redirect them to the login page. ○ Show an error: *"You must be a registered recruiter to view candidate profiles."*
Variations	- Recruiter switches between list view and detailed profile view.

Functional Requirement 12

Field	Description
Title	Direct messaging between recruiters and applicants
Goal	Enable direct communication between recruiters and job applicants to facilitate hiring discussions, interview scheduling, and follow-ups.
Scope	AutoCareers Platform

Level	User
Actors	Recruiters, Job Applicants
Description	When recruiters find a relevant job applicant, they should be able to contact that applicant, and that job applicant should be able to reply to them.
Trigger	When users click on message icon, they should be able to see an interface which supports this functionality
Preconditions	● Users should be logged in. ● There should be at least two users in the system in order to message each other.
Main Success Scenario (Goals)	- Recruiter successfully sends a message to an applicant. - Applicant receives the message and replies. - Both parties can engage in direct messaging to discuss job opportunities.
Alternate Success Scenarios	- Applicant replies at a later time, and the system notifies the recruiter. - Recruiter sends follow-up messages after an interview.
Postconditions	- Conversation history is saved for future reference. - Recruiter and applicant can continue discussions or close the conversation.

Main Flow & Steps of Execution (Requirements)	1. Recruiter Initiates Chat: - Recruiter navigates to an applicant's profile or job application. - Recruiter clicks "Message Applicant" and types a message. - System sends the message and notifies the applicant. 2. Applicant Receives Message: - Applicant gets a new message notification. - Applicant opens the message and replies. 3. Ongoing Communication: - Messages are exchanged between both users. - System maintains chat history and provides real-time notifications.
Extensions	- Allow file attachments (resumes, interview schedules) (Reach Goal)
Failed End Condition	User is unable to send the message. User sent the message, but it was not delivered to the user. Original message sent by the user got altered when the receiver received it.
Exceptions	**Network or Server Issues** ● **Scenario**: The system encounters an issue while sending a message, causing delivery failure. ● **Handling**: ○ Show an error: *"Message failed to send. Please try again."*
Variations	- Users can delete conversations. (Reach Goal)

Functional Requirement 13

Field	Description
Title	User applies to a job from the job board list
Goal	Based on the all job listings that users are able to see, they should be able to apply to them.
Scope	AutoCareers Platform
Level	User
Actors	Primary Actor: Job Seeker (User) Secondary Actors: Job Board System, Application Tracking System, Recruiters
Description	From the list of available jobs, users can apply to multiple jobs, they just need to click on whatever job they want to apply to, and from there, using the apply button, they will be redirected to the employer's website to complete the application.
Trigger	A user selects a job from the job board and clicks the "Apply" button.
Preconditions	The user is logged in. The job board is loaded with available job listings. The selected job has an active application link.

Main Success Scenario (Goals)	The user successfully clicks **"Apply"** and is redirected to the employer's website to complete the application.
Alternate Success Scenarios	The user chooses to apply to the application at a later date.
Postconditions	The user has successfully navigated to the external job application.
Main Flow & Steps of Execution (Requirements)	1. User browses the job board and selects a job. 2. User clicks the **"Apply"** button. 3. User is redirected to the employer's website to complete the application.
Extensions	Implement one-click apply for jobs that accept auto-submitted applications. (Reach Goal)
Failed End Condition	● The application link is broken or expired. ● The employer's website fails to load. ● The user is not redirected due to a technical issue.
Exceptions	*Employer Website Down – Notify the user: "Unable to reach the employer's website. Try again later."*
Variations	Users receive success email on applying. (Reach Goal)

Non-Functional Requirements

Performance

- The system should provide relatively fast response times for job searches, filtering, and resume processing.

Security (Securability)

- The system should support secure authentication using OAuth and/or multi-factor authentication (MFA).
- Sensitive user information should be encrypted to ensure data privacy.
- Role-based access control should be implemented for recruiters and users.

Usability

- The user interface should be intuitive and easy to navigate.
- The system should have a browser-friendly, responsive design.
- The platform should comply with WCAG accessibility standards to ensure usability for all users.

Maintainability (Debuggability & Understandability)

- The system should have a modular and well-documented codebase for easier maintenance.
- Log tracking should be implemented to help debug errors efficiently, with clear error messages provided for both users and developers.
- The codebase should be structured to ensure ease of understanding for future developers.

Modularity

- The system should follow the separation of concerns principle, with distinct modules for authentication, job board, messaging, and resume parsing.
- Components should be independent and reusable to allow for easier updates and maintenance.

Logging & Monitoring

- The system should track failed login attempts, job application status, and user activity.
- Performance monitoring should be in place, with alerts for downtime or slow response times.

Testability

- The system should support automated unit, integration, and end-to-end testing to ensure reliability.

Challenges & Risks

Technical Risks:

- **Integration Issues:** Difficulty in integrating third-party APIs (e.g., OAuth for authentication, job search APIs).
- **Scalability Concerns:** Handling increased user traffic efficiently without performance degradation.
- **Data Handling & Processing:** Efficient parsing and processing of resumes while maintaining data integrity.
- **Security & Privacy:** Ensuring password encryption and protecting user information to maintain privacy and compliance with data protection standards.
- **AI Personalization Accuracy:** Automated cover letter and resume suggestions may alter content too much, making it less relevant to the applicant's actual profile. Finding the right balance between automation and accuracy is critical.

Team Collaboration Risks:

- **Coordination Across Tasks:** Challenges in synchronizing work among team members, leading to potential conflicts or duplication.
- **Version Control Conflicts:** Potential merge conflicts in GitHub due to simultaneous development efforts.
- **Communication Gaps:** Misunderstandings regarding responsibilities, priorities, or technical decisions.
- **Differences in Opinions:** Varying perspectives on design choices, implementation strategies, and feature priorities may lead to delays or conflicts.

Timeline Risks:

- **Delayed Development Phases:** Unexpected technical roadblocks leading to missed milestones.
- **Over-Ambition:** Expanding the project scope beyond what is feasible within the given time frame, leading to incomplete or rushed features.
- **Workload Balancing:** Conflicting academic or personal commitments affecting team members' availability.

Mitigation Strategies:

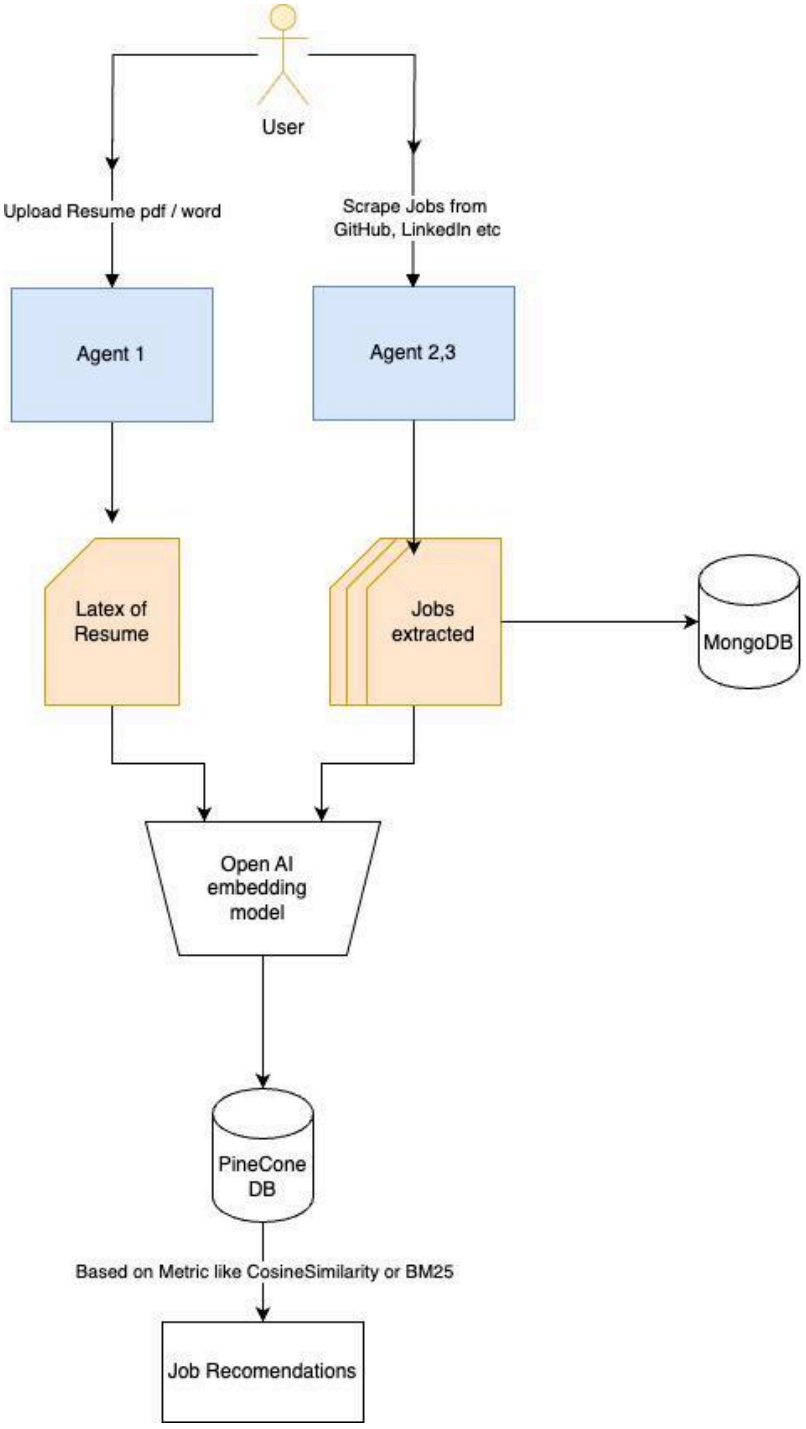
- **Regular Team Meetings:** Weekly check-ins to track progress, address blockers, and ensure alignment.
- **Use of Agile Methodologies:** Implementing sprints with defined goals to stay on schedule.
- **Version Control Best Practices:** Proper use of Git branching strategies (feature branches, pull requests, and code reviews).
- **Task Management Tools:** Using Trello, Jira, GitHub Projects, and GitHub Issues to track assignments and deadlines.
- **Security Best Practices:** Implementing password encryption, ensuring user information privacy, and adhering to data protection guidelines.

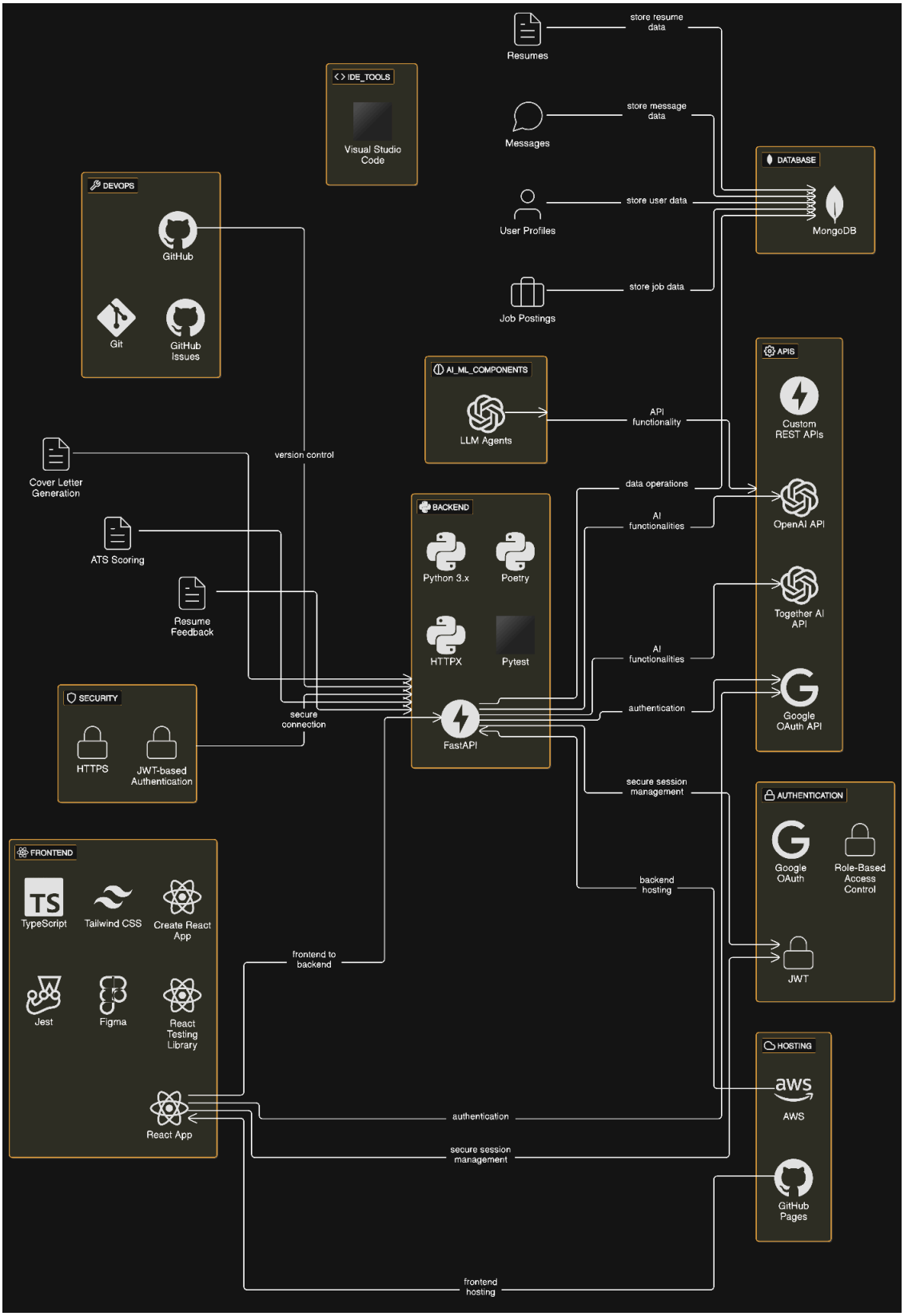
- **Testing AI Suggestions:** Continuously refining the cover letter and resume recommendation system to balance automation with user relevance.
- **Conflict Resolution Approach:** Encouraging open discussions, consensus-building, and voting mechanisms to resolve differences in opinions.
- **Scope Management:** Clearly defining the project scope early on, prioritizing core features, and avoiding unnecessary expansions to ensure timely delivery.
- **Buffer Time Allocation:** Allocating additional time for testing and bug fixes to prevent last-minute issues.

Question 2: Design

Architecture Diagram

The **first diagram** provides a high-level system architecture illustrating the overall workflow of the AutoCareers platform, highlighting how resumes and job data flow through AI models and databases to generate job recommendations. The **second diagram** is a detailed system architecture that showcases all major connections and transitions between the frontend, backend, databases, APIs, security mechanisms, and deployment tools, depicting the complete interaction and integration among system components.




Resumes
store resume data
IDE_TOOLS
Visual Studio Code
Messages
store message data
DATABASE
MongoDB
DEVOPS
GitHub
User Profiles
store user data
Git
GitHub Issues
Job Postings
store job data
APIS
AI_ML_COMPONENTS
LLM Agents
API functionality
Custom REST APIs
Cover Letter Generation
version control
data operations
BACKEND
OpenAI API
ATS Scoring
AI functionalities
Python 3.x
Poetry
Resume Feedback
Together AI API
HTTPX
Pytest
AI functionalities
SECURITY
authentication
Google OAuth API
secure connection
FastAPI
HTTPS
JWT-based Authentication
secure session management
AUTHENTICATION
FRONTEND
Google OAuth
Role-Based Access Control
backend hosting
TypeScript
Tailwind CSS
Create React App
JWT
frontend to backend
Jest
Figma
React Testing Library
HOSTING
aws
AWS
React App
authentication
secure session management
GitHub Pages
frontend hosting

Technology Stack Justification

The technologies for AutoCareers were chosen to balance performance, scalability, ease of development, and compatibility with AI-powered features. React and TypeScript were selected for building a modular, responsive, and maintainable frontend. FastAPI with Python was chosen for the backend due to its asynchronous capabilities and seamless integration with AI services such as OpenAI and Together AI. MongoDB was selected for its flexible schema, suitable for storing dynamic and AI-generated data. Authentication is handled securely using JWT and Google OAuth, while GitHub, VS Code, and Postman were used for efficient development, testing, and collaboration. The stack ensures that the platform can deliver AI-powered resume scoring, job matching, and application management effectively while remaining adaptable for future enhancements.

Frontend

React
We selected React for its component-based architecture, which promotes modular, reusable, and maintainable user interface components. React's virtual DOM ensures efficient rendering, and its popularity offers strong community support and abundant third-party libraries.

TypeScript
TypeScript was chosen to improve code reliability by enabling static type-checking and reducing runtime errors. It enhances developer productivity, code maintainability, and makes collaboration easier by catching potential issues during development.

Tailwind CSS
Tailwind CSS is used for styling the application due to its utility-first approach, which enables rapid development of highly responsive and customizable user interfaces. Tailwind allows for scalable and maintainable styling without the need for large external CSS files.

React Testing Library & Jest
These testing tools were adopted for frontend testing due to their tight integration with React and widespread use in the community. They allow for efficient unit and integration testing of components, helping ensure correctness and robustness of the user interface.

Create React App
Create React App was used to bootstrap the frontend project, providing pre-configured tooling for development, testing, and building the frontend without manual setup.

Figma
Figma was chosen as the wireframing and UI/UX design tool due to its collaborative features, enabling the team to prototype and iterate efficiently and gather feedback during the design process.

Material UI
Material UI (MUI) was selected as the component library for the frontend to accelerate the development of consistent, visually appealing, and accessible user interfaces. MUI provides a comprehensive set of pre-built React components that follow Google's Material Design

guidelines, ensuring a modern and user-friendly design. Its customization capabilities allow the team to adapt components to match the application's theme while maintaining responsiveness across different devices. Additionally, Material UI integrates seamlessly with React and Tailwind CSS, enhancing development efficiency.

Backend

Python 3.x
Python was selected as the backend language because of its simplicity, readability, and extensive ecosystem, especially for AI and data processing tasks required by the project.

FastAPI
We chose FastAPI for its high performance, asynchronous capabilities, and automatic generation of interactive API documentation. It is well-suited for building modern REST APIs and integrates smoothly with AI/ML libraries.

Poetry
Poetry was adopted for Python dependency management to maintain isolated virtual environments and reproducible builds. It simplifies package management compared to traditional `pip` and `requirements.txt`.

HTTPX / Requests
These libraries are used to make external API calls to services like OpenAI and Together AI. HTTPX offers asynchronous capabilities, improving performance when dealing with multiple external API requests.

Pytest
Pytest was selected for backend testing due to its powerful and flexible testing features. Its simple syntax and plugin support help automate unit and integration testing, ensuring the backend's reliability.

Database

MongoDB
MongoDB was chosen as the primary database for its flexible, document-oriented schema, which easily accommodates dynamic and evolving data structures like user profiles, job postings, resumes, ATS scores, and AI-generated documents. Its native support for JSON-like documents simplifies integration with both backend and frontend. We used the cloud version of MongoDB, which makes it easy to deploy finally.

Pinecone
It is used to store the embeddings generated by the model which is used for similarity measurement and give job recommendation, we chose Pinecone as it's widely accepted in the industry for its reliability, features, and scalability.

APIs

Custom REST APIs (FastAPI)
REST APIs were implemented to handle core functionalities such as user management, job management, resume handling, messaging, and AI-powered services. FastAPI's built-in support for OpenAPI and JSON Schema improves maintainability and documentation.

OpenAI API
Integrated to power AI functionalities like resume scoring, resume generation, and cover letter generation using advanced language models.

Together AI API
Together AI API is used to extend AI-powered services by offering access to efficient and cost-effective/ open-source LLMs, supplementing the capabilities provided by OpenAI which is more expensive

Google OAuth API
Integrated to enable third-party authentication, allowing users to sign up and log in securely using their Google accounts.

Authentication

JWT (JSON Web Tokens)
JWT is employed for secure, stateless authentication and authorization. It ensures safe session management and supports role-based access control for different user types (Job Seekers, Recruiters, System Administrators).

Google OAuth
Google OAuth enhances the authentication system by providing users with the option to securely log in using their Google accounts, reducing friction during registration and login.

Hosting

GitHub Pages
Chosen for static deployment of the frontend due to its simplicity and integration with GitHub, making it easy to host and update the frontend.

AWS
AWS is used for deploying backend services and managing databases. It offers scalability, flexibility, and reliability suitable for production environments.

DevOps

Git & GitHub
Used for version control, collaborative development, code reviews, and project management. GitHub facilitates team collaboration through pull requests and issue tracking.

GitHub Issues
GitHub Issues is used to manage tasks, track bugs, and organize development sprints.

Security

HTTPS
Ensures that all data exchanged between the frontend, backend, and external services is encrypted and secure.

JWT-based Authentication
Provides secure token-based session handling, protecting against session hijacking and unauthorized access.

AI / ML Components

LLM Agents (OpenAI & Together AI APIs)
These agents are responsible for resume scoring, resume generation, cover letter generation, and recommendation tasks. They enhance the user experience by providing AI-generated content and analysis.

LlamaIndex
LlamaIndex is used as a framework to manage and structure data inputs for Large Language Models. It plays a crucial role in organizing and indexing resume and job data before interacting with LLMs for tasks such as resume analysis, job matching, and recommendation generation. By enabling efficient data retrieval and query processing, LlamaIndex improves the performance and contextual accuracy of AI-generated outputs, especially when handling personalized recommendations and document-based reasoning

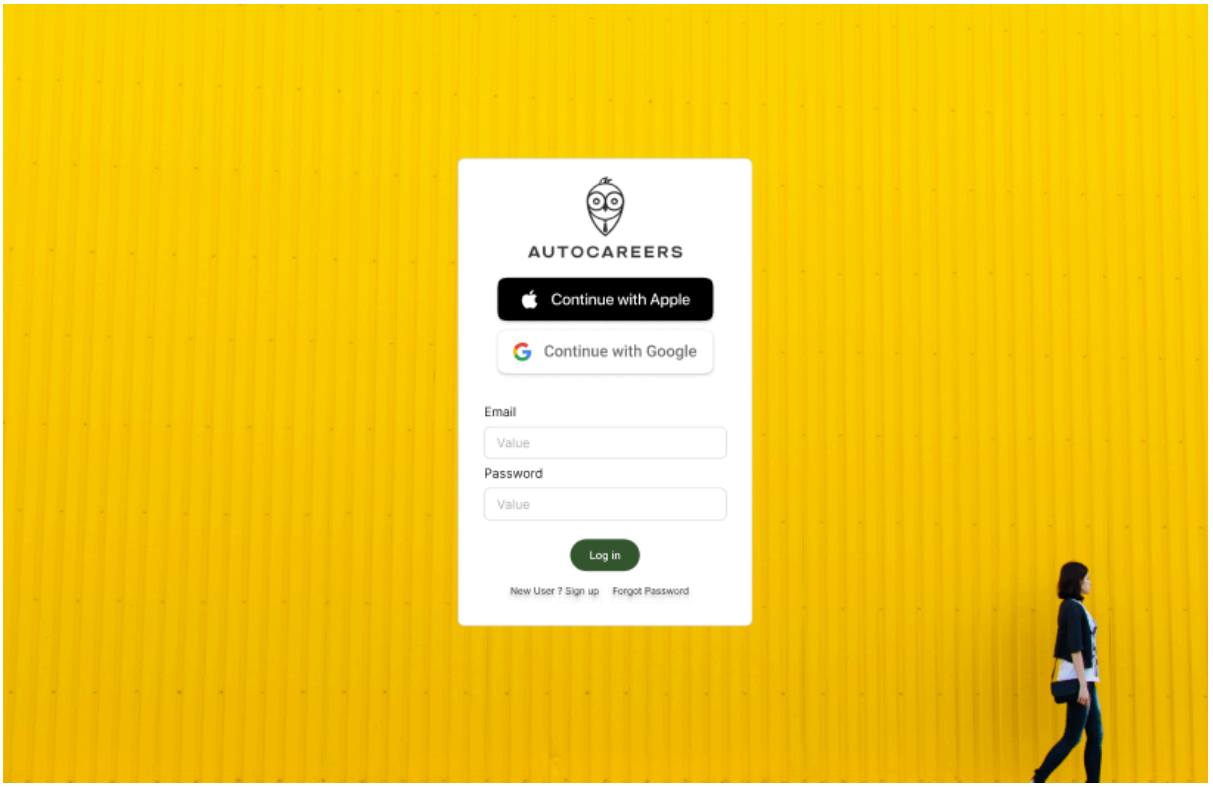
IDE & Tools

Visual Studio Code (VS Code)
Selected as the main development environment for both frontend and backend due to its lightweight nature, wide extension ecosystem, and strong support for TypeScript, Python, and REST API development.

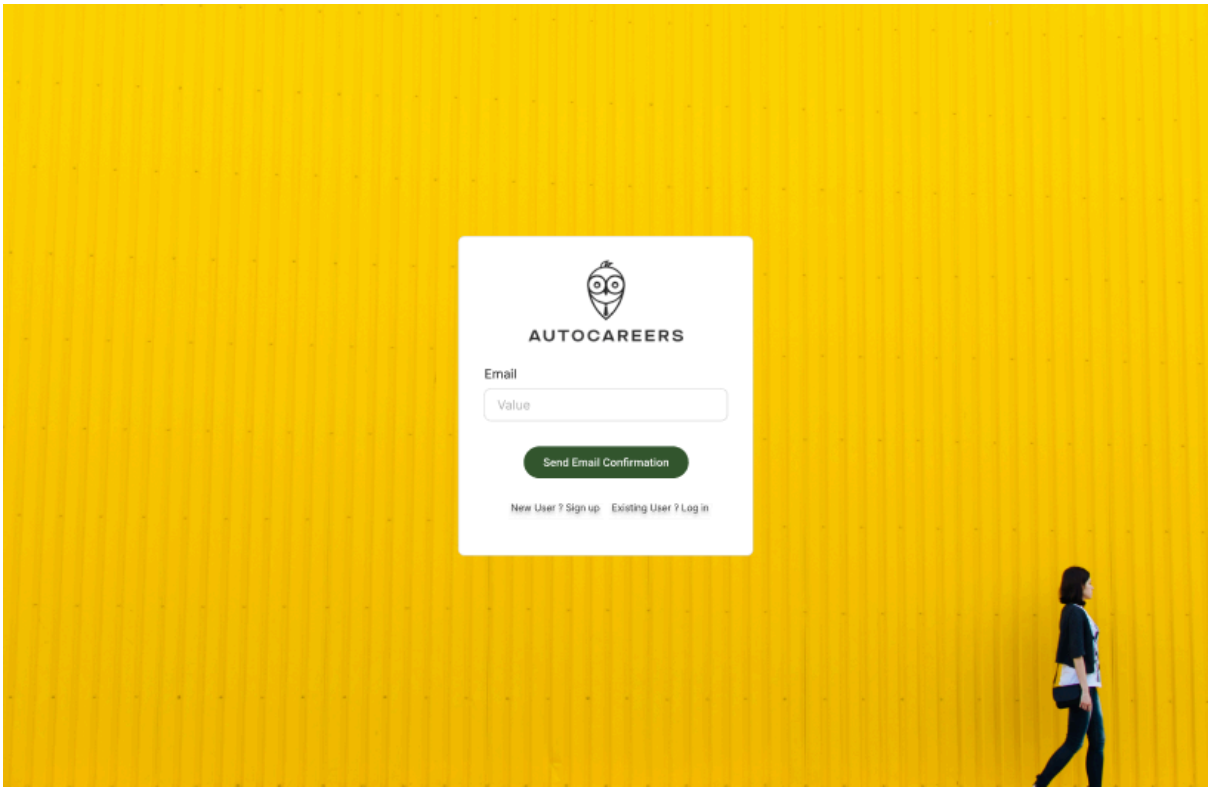
Postman
Used extensively for testing REST APIs during backend development, allowing the team to debug and validate API responses before integrating them with the frontend.

UI Mockup + Wireframes (Created on Figma)

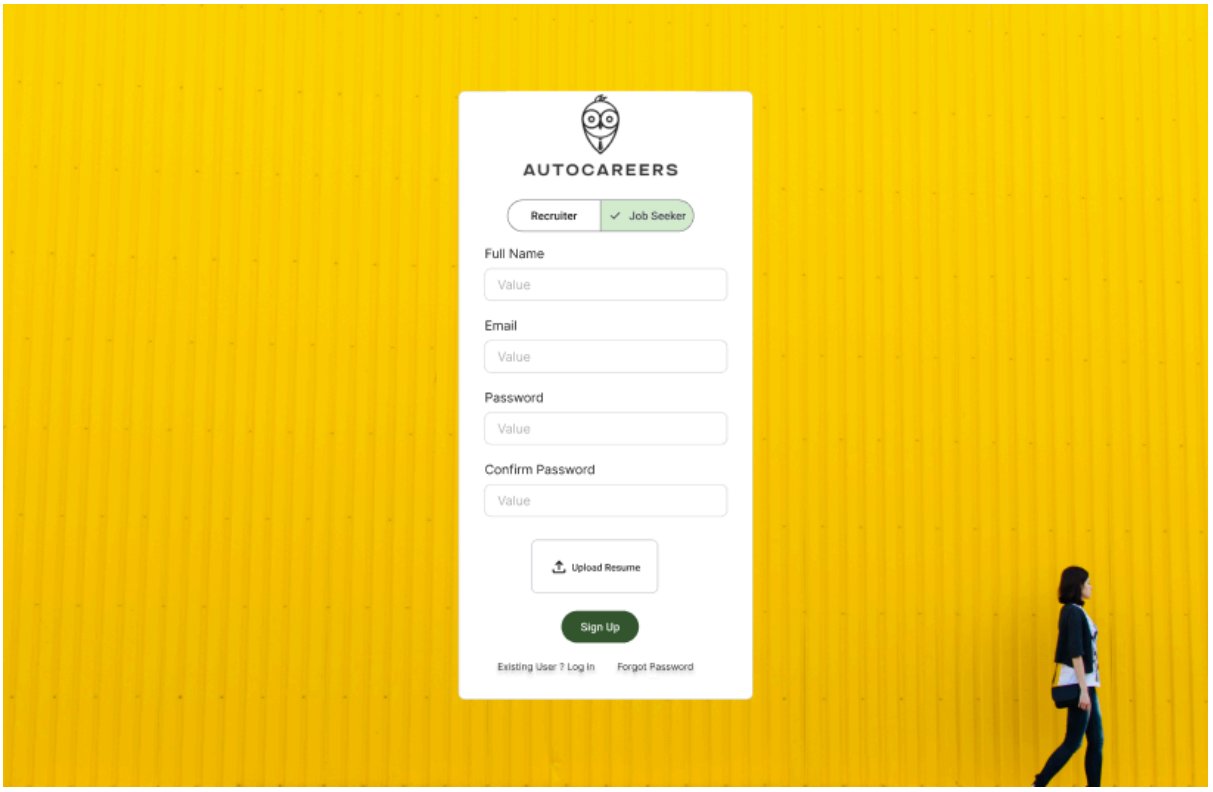
Authentication flow



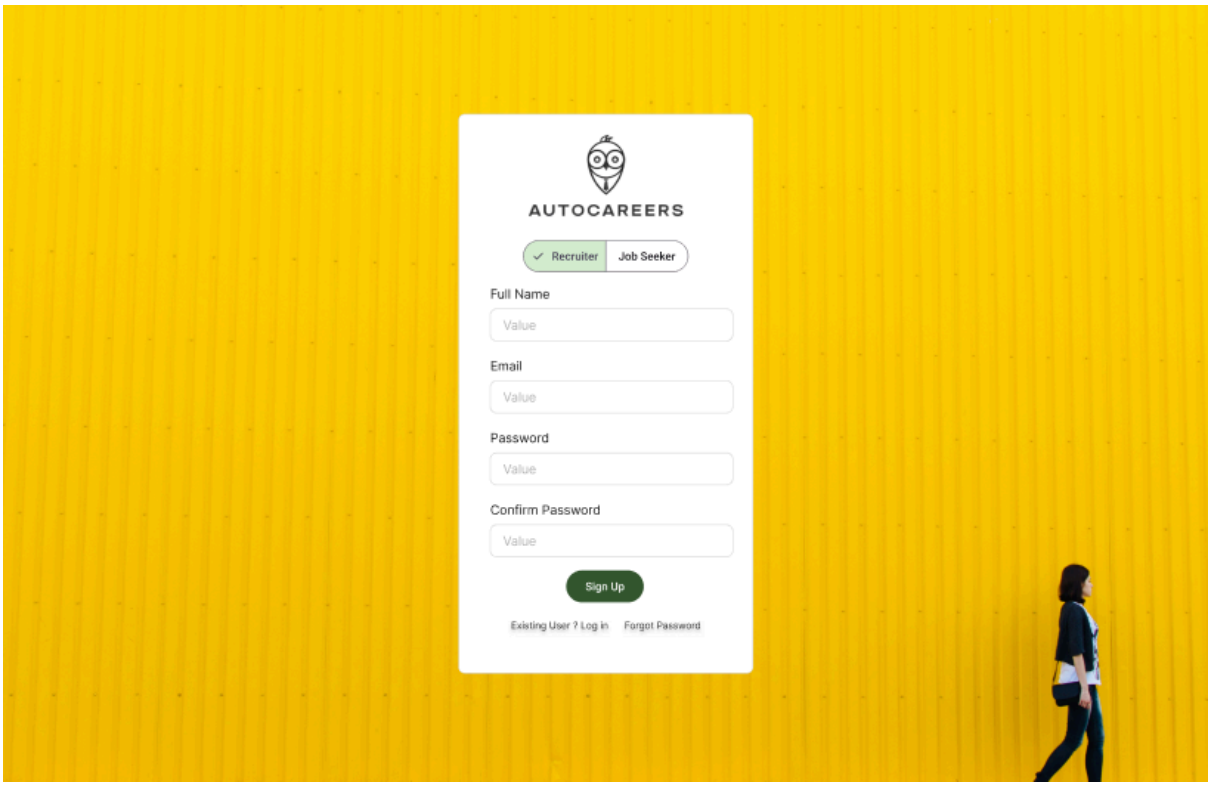
Example: Log in



Example: Forgot password

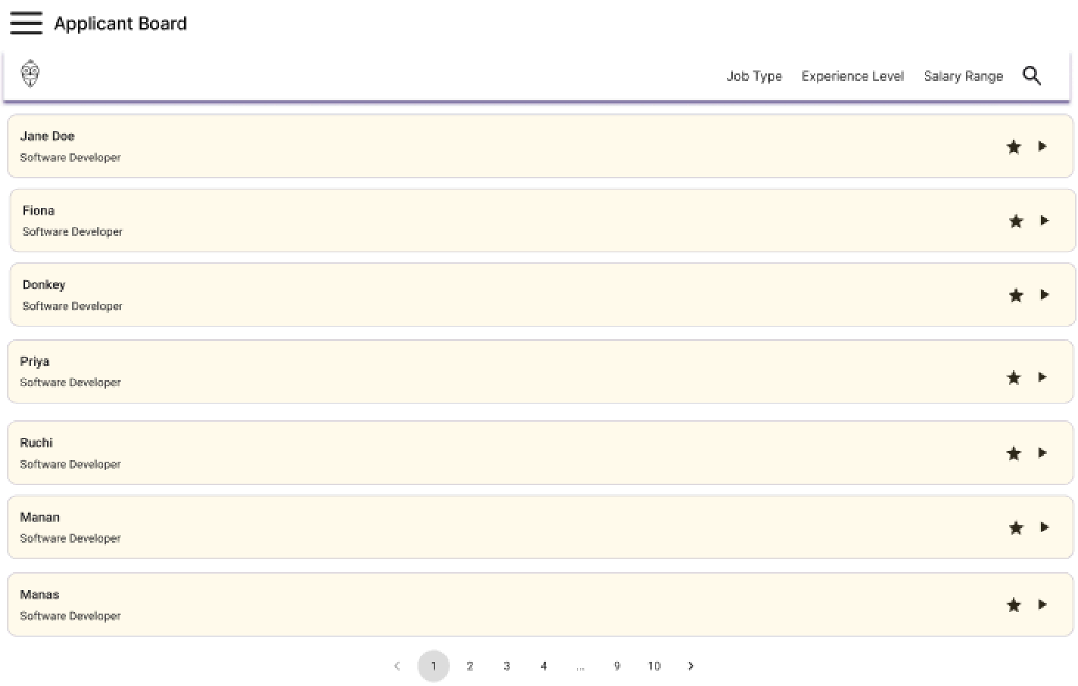


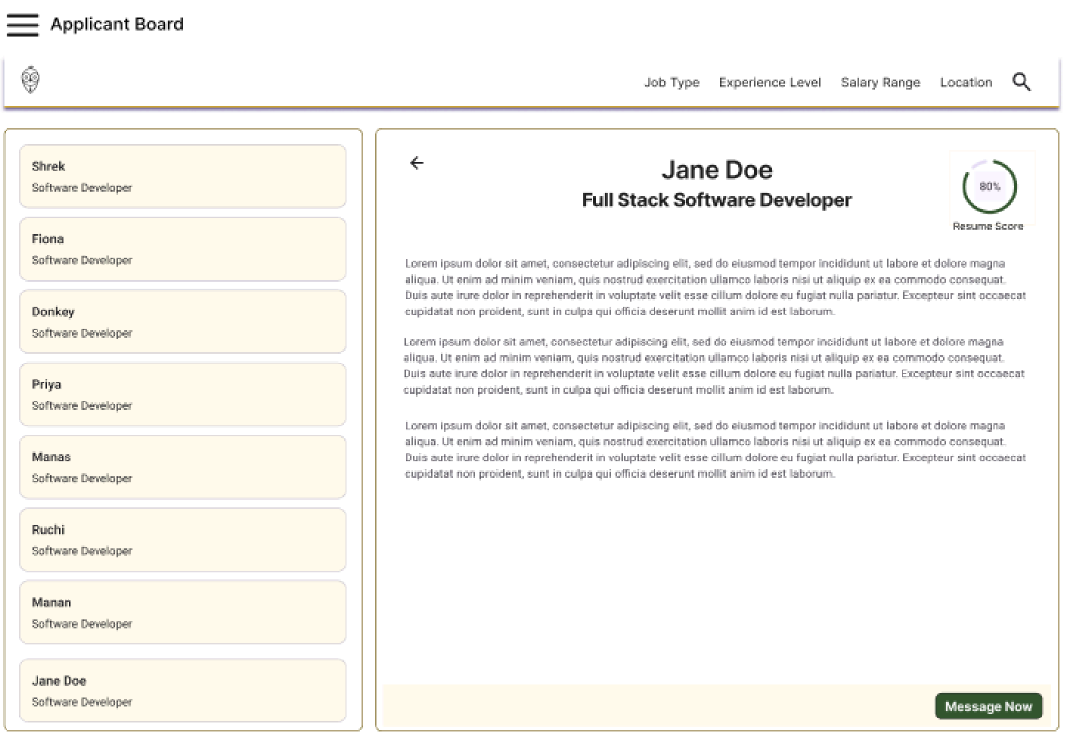
Example: Job Seeker Registration



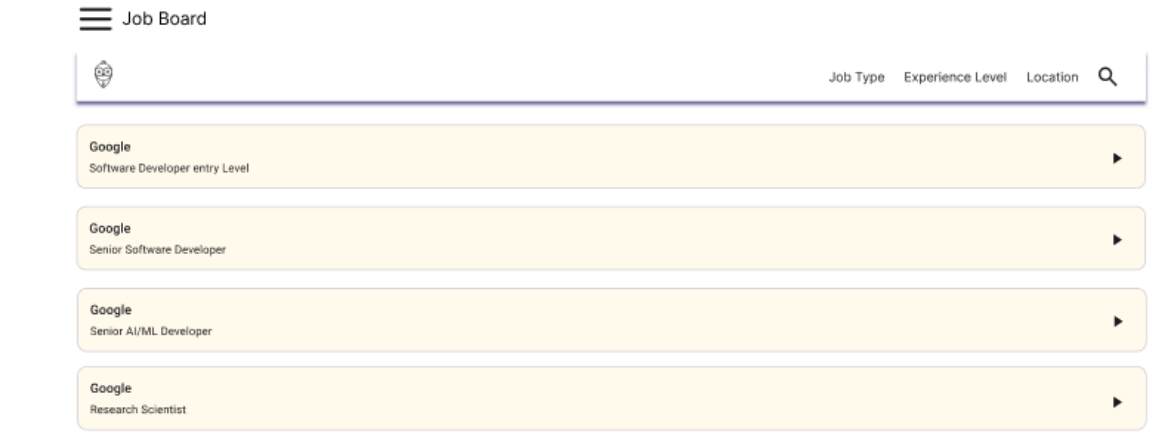
Example: Recruiter Registration

Recruiter View



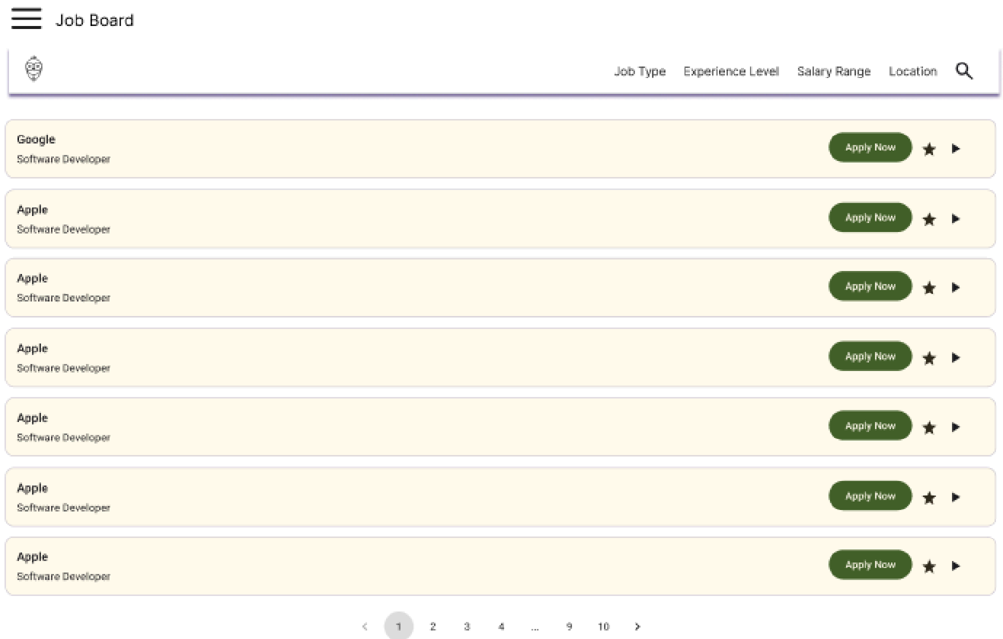


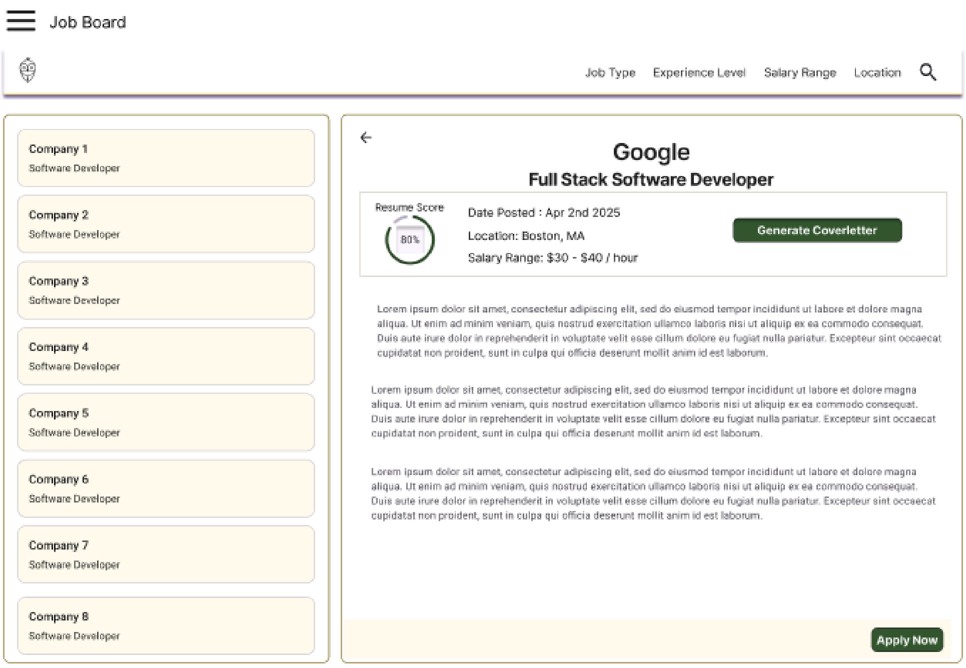
Applicant Board containing a list of jobseeker profiles

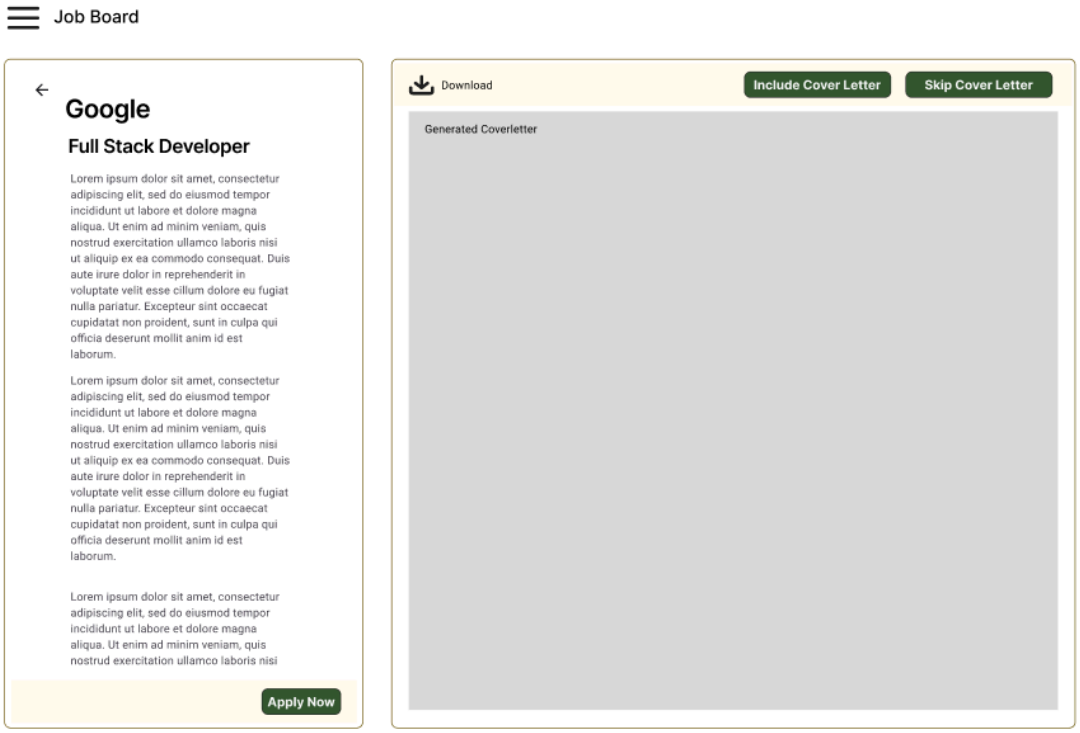


Recruiter Job Board - Recruiters can keep track of their job postings and view individual applicants and their application materials.

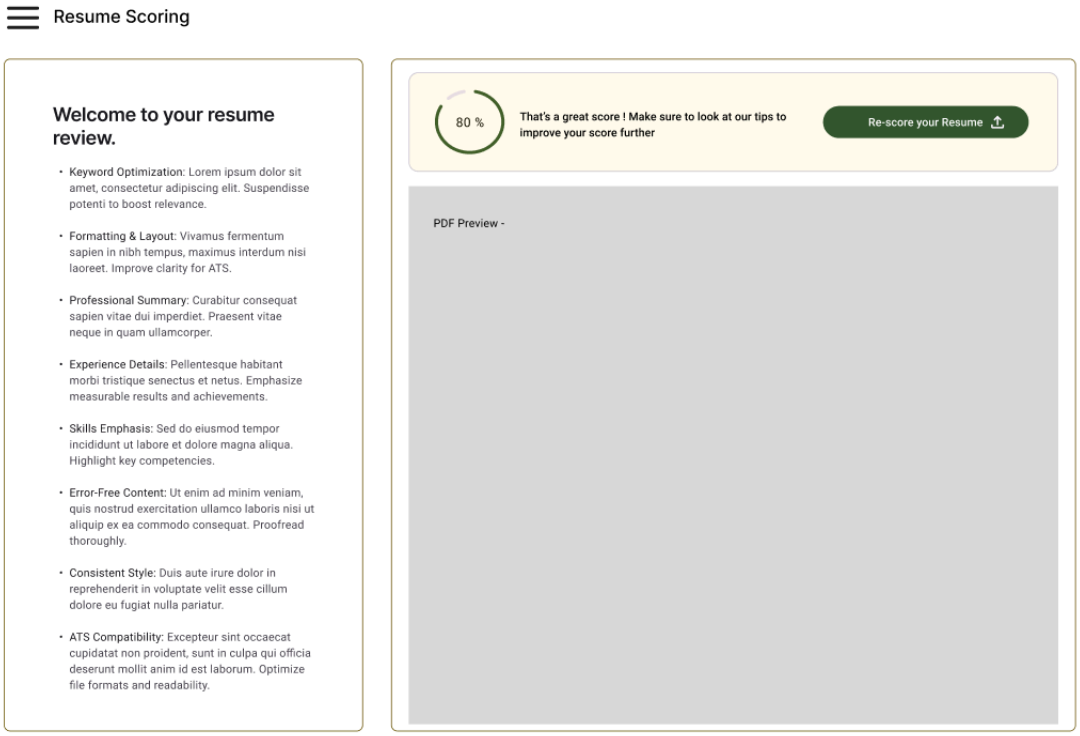
Job Seeker View





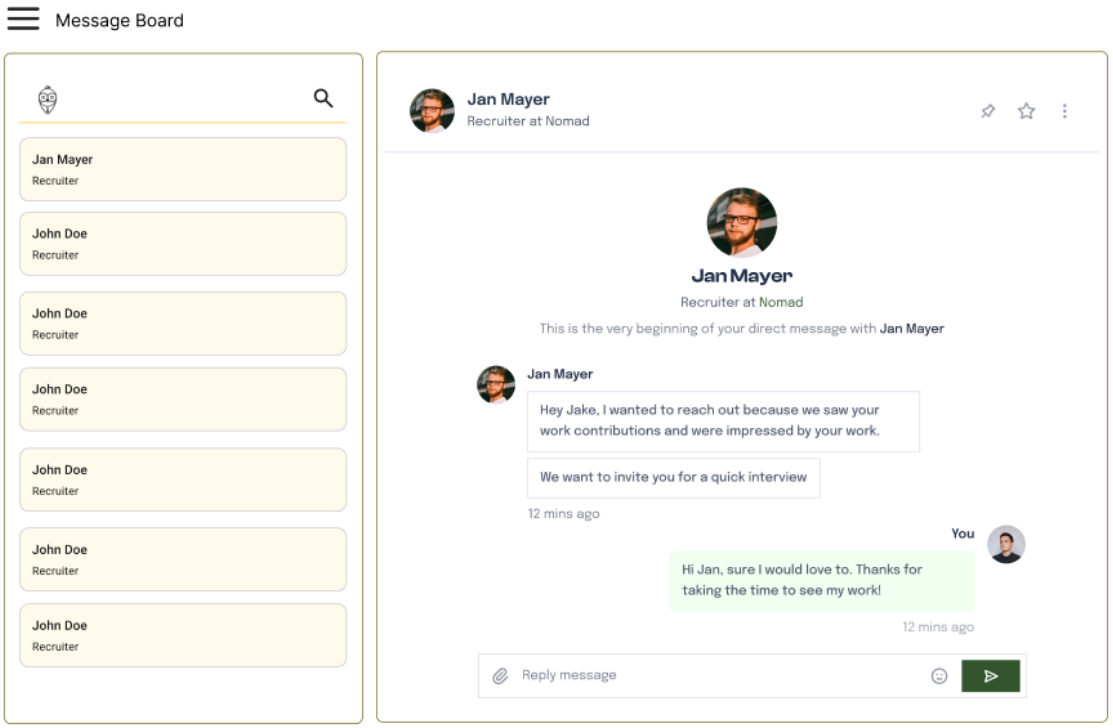


Job Seeker Job Board - Job Seekers can view jobs suited for them, analyse their resume score, generate cover letter according to the job requirements and Apply for the job.



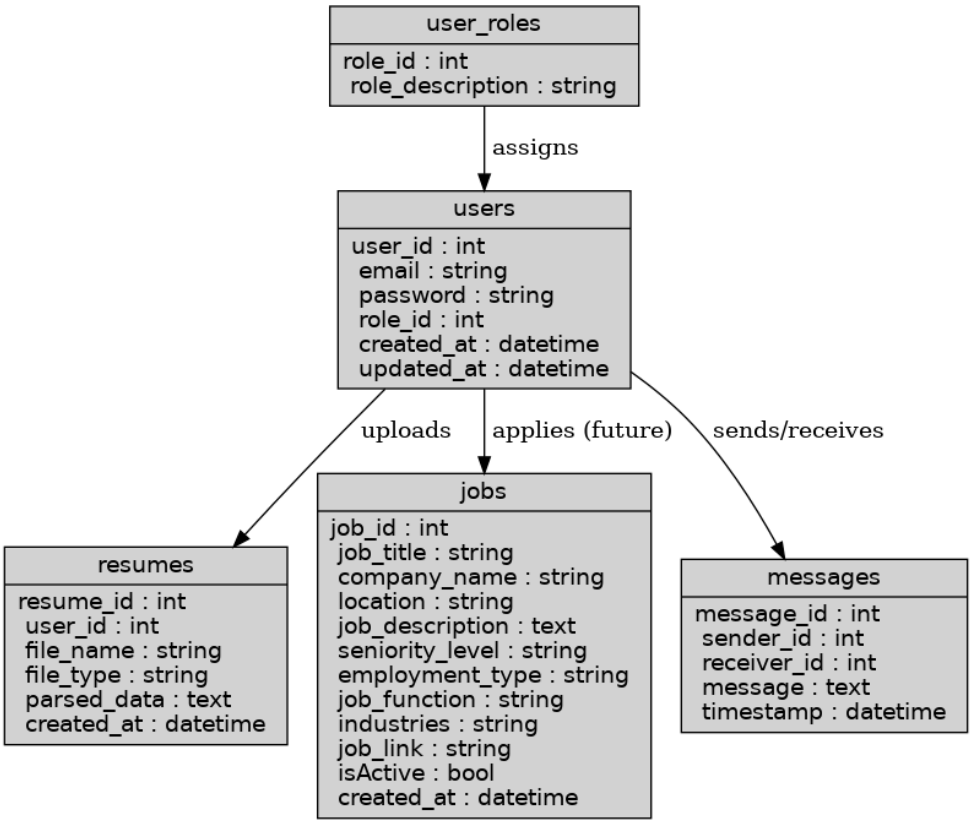
Job Seeker Resume Scoring Page - Upload a new resume, analyse the ATS score and receive key insights to improve your resume.

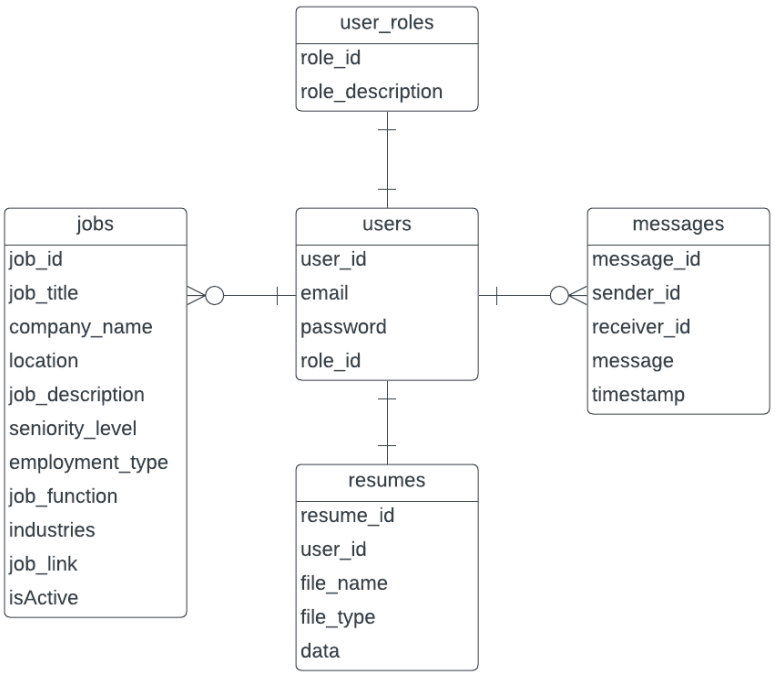
Message Board for both Recruiters and Job Seekers

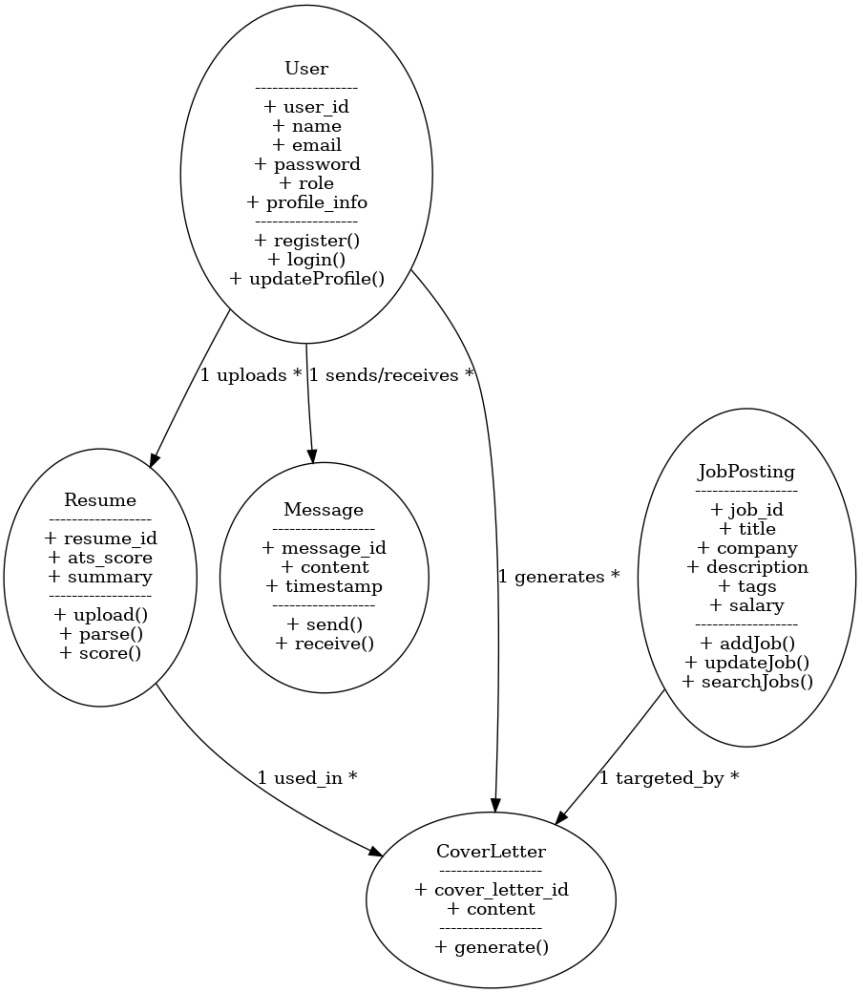


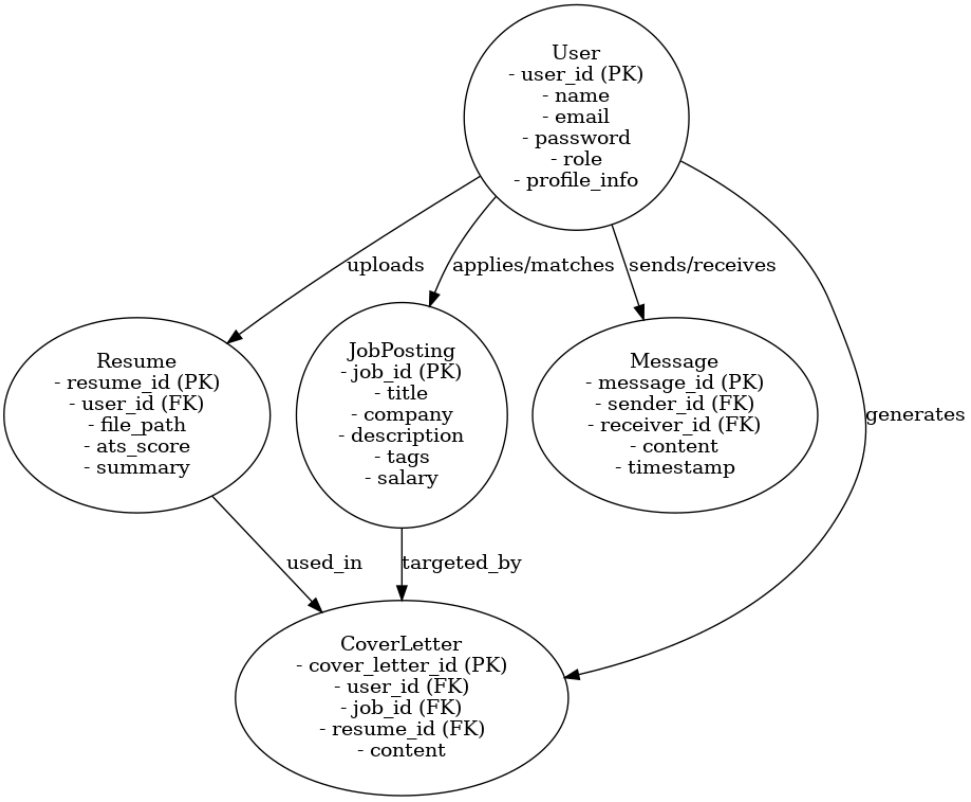
Data Model

The following iterations of ER and UML diagrams illustrate how data is structured and organized within the AutoCareers platform, detailing the key entities such as users, resumes, jobs, messages, and their relationships, as well as the system's class-level design reflecting object-oriented modeling used in the backend.






User
+ user_id
+ name
+ email
+ password
+ role
+ profile_info
+ register()
+ login()
+ updateProfile()
1 uploads *
1 sends/receives *
1 generates *
Resume
+ resume_id
+ ats_score
+ summary
+ upload()
+ parse()
+ score()
Message
+ message_id
+ content
+ timestamp
+ send()
+ receive()
JobPosting
+ job_id
+ title
+ company
+ description
+ tags
+ salary
+ addJob()
+ updateJob()
+ searchJobs()
1 used_in *
1 targeted_by *
CoverLetter
+ cover_letter_id
+ content
+ generate()


User
- user_id (PK)
- name
- email
- password
- role
- profile_info
uploads
applies/matches
sends/receives
Resume
- resume_id (PK)
- user_id (FK)
- file_path
- ats_score
- summary
JobPosting
- job_id (PK)
- title
- company
- description
- tags
- salary
Message
- message_id (PK)
- sender_id (FK)
- receiver_id (FK)
- content
- timestamp
generates
used_in
targeted_by
CoverLetter
- cover_letter_id (PK)
- user_id (FK)
- job_id (FK)
- resume_id (FK)
- content

Challenges, Security and Risks

Data Breach & Security Risks

AutoCareers handles sensitive user information such as resumes, profiles, and job-related data, which may include personally identifiable information (PII). The risk of unauthorized data access or leakage is a major concern. To mitigate this, we have incorporated secure authentication using Google OAuth and JWT, encrypted all communications through HTTPS, and implemented role-based access control. Passwords are securely hashed, and input validation and sanitization are enforced to protect against common vulnerabilities such as injection and cross-site scripting (XSS) attacks.

Integration Complexity

AutoCareers integrates multiple system components, including the frontend, backend, AI-powered services (OpenAI, Together AI), and third-party authentication. Ensuring smooth interoperability among these services can be challenging. To address this, we follow a modular architecture, maintain clear API documentation, and carry out incremental integration testing to detect and resolve issues early.

AI Model Reliability

Since resume scoring, feedback, and cover letter generation are AI-driven, there is a risk that generated outputs may be misaligned with the user's actual profile or produce irrelevant suggestions. This risk is mitigated by carefully engineering prompts, testing AI-generated content, and allowing users to review and adjust AI outputs as needed.

Data Quality

The platform depends on scraped job data from external sources such as LinkedIn and GitHub Jobs, which may vary in structure or accuracy. To address this, the data is cleaned, deduplicated, and validated before being ingested into MongoDB. Preprocessing ensures that incomplete or duplicate job listings are minimized.

Team Collaboration

Collaboration risks include merge conflicts, task duplication, communication gaps, or disagreements on technical decisions. To mitigate these, the team follows Agile methodology with regular scrums every two days, weekly meetings, and active task tracking using GitHub Issues. Code quality is maintained through peer reviews, pull requests, and version control best practices.

Timeline Risks

Potential delays may occur due to unforeseen technical challenges, integration difficulties, or over-ambitious feature planning. The team mitigates this by maintaining a clearly scoped and prioritized feature list, allocating buffer time for testing and fixes, and adjusting sprints as needed to balance workload and deliver on time.

Mitigation Strategies

- **Agile Methodology:** The team follows Agile practices with iterative sprints and defined sprint goals to ensure continuous and focused progress.

- **Scrum Meetings:** Regular scrums every two days and weekly team meetings are conducted to track progress, resolve blockers, and synchronize tasks.

- **Version Control Best Practices:** We follow Git workflows involving feature branches, pull requests, peer reviews, and issue tracking to maintain code quality and minimize merge conflicts.

- **Task Tracking:** GitHub Issues are actively used for task allocation, progress monitoring, and sprint planning.

- **Security Measures:** Implementing industry-standard security practices such as password encryption, secure authentication (OAuth + JWT), input validation, and role-based access control to protect sensitive user data.

- **Testing and Validation:** Regular unit, integration, and end-to-end testing are conducted to ensure system reliability. AI-generated suggestions are tested and reviewed to balance automation with accuracy.

- **Conflict Resolution:** Open communication, team discussions, and consensus-building are adopted to address disagreements and technical decisions effectively.

- **Scope Management:** The project scope is clearly defined and regularly reviewed to prioritize core features and prevent scope creep.

- **Buffer Time Allocation:** Buffer periods are included in the project timeline to accommodate unexpected technical challenges, testing, and refinements.

Question 3: Implementation

The implementation of AutoCareers followed an iterative and modular development approach grounded in modern software engineering practices. The team focused on building a scalable, secure, and AI-powered platform that streamlines the job search and recruitment process. Development was organized around sprints, with regular code reviews, version control practices, and collaborative tooling to ensure smooth coordination across frontend, backend, and AI integration components. We prioritized clean code architecture, role-based modularity, and robust authentication while addressing real-world challenges such as third-party API limitations, resume data variability, and security concerns. This section outlines how version control, coding standards, and security considerations were embedded into the development lifecycle.

3.1 Version Control and Collaboration

We used Git and GitHub as the foundation for all version control and collaboration throughout the development of AutoCareers. Each team member worked on their own feature branch named after them: `ruchi-dev`, `manan-dev`, `priya-dev`, and `manas-dev`. This structure allowed everyone to develop and test their features independently while keeping the main branch stable and production-ready.

Once a feature or bug fix was ready, it was pushed to GitHub and a pull request (PR) was created to merge the changes into the main branch. Before merging, all PRs were reviewed by other team members. This process helped ensure that the code met our standards, followed the project's structure, and aligned with any ongoing changes. Reviewers left comments, suggested improvements, and approved changes before the PR was merged.

GitHub Issues played a central role in how we organized and tracked development tasks. We created issues for every new feature, bug, or technical to-do. These issues were linked to commits and PRs, giving us a clear history of what had been done and why. This made it easier to manage progress during sprints and provided clarity when prioritizing work.

Merge conflicts, especially in shared parts of the code like backend models or API routes, were resolved through regular communication. We often discussed overlapping changes in team meetings or chat, and coordinated on who would take the lead resolving the conflicts. We also used tools like GitHub's web-based conflict resolution and manual rebasing to avoid unnecessary complexity.

To avoid breaking changes, we merged frequently and kept branches small and focused. Everyone was responsible for keeping their branches up to date with main, which helped reduce the chance of large-scale conflicts during final merges.

This workflow helped us stay organized, productive, and aligned as a team while maintaining a clean, stable, and well-documented codebase.

3.2 Coding Standards and Practices

To maintain a consistent and maintainable codebase across a multi-developer team, we followed clear coding standards and best practices throughout the AutoCareers project. For backend development in Python, we adhered to the PEP 8 style guide to ensure readability, proper indentation, and consistent formatting. This included standardized naming conventions for variables, functions, and classes, with clear, descriptive names that reflected their purpose and improved code comprehension.

In the frontend, we used TypeScript with React, applying component-based organization. Each component had its own folder, containing its logic, styling, and tests. We followed camelCase naming for functions and variables, PascalCase for component names, and organized utility functions in separate modules to encourage reuse. Tailwind CSS was used with care to avoid clutter, and we grouped utility classes logically in the JSX for easier readability.

Our team placed a strong emphasis on writing clear, in-line comments and high-level function docstrings, especially for complex logic, data processing, and API routes. This helped reduce onboarding time for new contributors and made debugging much more straightforward. In the backend, we used descriptive endpoint naming and consistent route structure across FastAPI services. We also separated concerns by organizing code into modular directories such as `auth`, `jobs`, `resume`, and `messaging`, making the structure intuitive and scalable.

We also documented our development progress using markdown files within the repository. The `README.md` provided a comprehensive overview of the project, including its purpose, core features, tech stack, and setup instructions. We created a `build.md` file to explain how to set up the development environment, manage dependencies using Poetry, and deploy the backend services. Similarly, frontend setup steps and scripts were documented clearly to ensure smooth local development and deployment.

To support testing and maintain code reliability, we added unit tests and integration tests using `Pytest` on the backend and `Jest` with `React Testing Library` on the frontend. Each module included sample tests and stubs so future tests could be added easily. We also emphasized modularity and reuse of components and API handlers to keep the codebase clean and extensible.

Lastly, we enforced consistency through code reviews. Every pull request had to pass lint checks and undergo peer review before merging. This gave everyone an opportunity to suggest improvements and kept everyone aligned on style and structure. These practices collectively helped us write robust, clean, and maintainable code that supported both short-term development and long-term scalability.

3.3 Security and Risks

Since AutoCareers handles sensitive user information such as resumes, profiles, and messages, we prioritized strong security practices throughout development. Our focus was

on safeguarding user data during transmission and storage, while ensuring that access to features and data was properly restricted based on user roles.

Authentication and Authorization:
We implemented secure authentication using Google OAuth along with JWT (JSON Web Tokens) for session handling. OAuth allowed users to log in using their Google accounts without needing to create a separate password, which lowered the risk of credential-based attacks. After successful authentication, users received a JWT token that was signed and time-limited. This token was used to validate sessions and enforce access permissions. Role-based access control was applied consistently across the platform, so that users interacted only with functionality relevant to their role, such as recruiter, job seeker, or administrator.

Data Encryption and Transmission:
All communication between the frontend, backend, and third-party APIs was secured using HTTPS to prevent unauthorized data access during transmission. Sensitive data, including passwords for email-based accounts, was hashed securely before being stored in the database. Our backend services used encryption libraries that follow standard security protocols to protect user data at rest and in transit.

API Security and Rate Limiting:
Each backend API endpoint includes appropriate authentication and error handling logic. To prevent overuse or potential misuse of resource-intensive features like resume scoring and cover letter generation, we plan to integrate rate limiting and usage monitoring per user session. This will help maintain service reliability and protect against abuse.

User Privacy and Data Visibility:
Job seekers are provided with the option to control the visibility of their profiles. Only profiles marked as public are visible to recruiters. This helps protect user privacy and ensures users have control over their personal information. Recruiters can only view applicant details if access has been explicitly enabled by the applicant.

Risk Mitigation:
To guard against service interruptions or dependency failures, we included fallback logic and error handling around third-party services such as OpenAI and Together AI. We also made sure to secure all API keys and configuration files, keeping them out of version control. Our team closely tracked merge activity using GitHub Issues and pull requests to prevent accidental leaks or overwrites during development.

Although AutoCareers is not currently subject to specific regulatory requirements like GDPR or HIPAA, our architecture supports extensions that would make compliance achievable. We followed best practices for responsible data handling, including limiting unnecessary data collection and protecting user access.

These practices helped us build a platform that is secure, reliable, and respectful of user privacy.

3.4 Supporting Artifacts and Technical Overview

To complement our implementation, we have included several key artifacts that provide a deeper look into the technical design, functionality, and collaborative structure of AutoCareers. These materials demonstrate the scope of both frontend and backend development, the tools used, and the workflows that supported our real-time and AI-powered features.

Client-Side Screenshots
We captured full-page and component-level views of the platform from both the job seeker and recruiter perspectives. These include:

- Resume upload and ATS scoring interface
- AI-generated cover letter view
- Messaging dashboard
- Recruiter applicant viewer
- Job board with advanced filtering

These screenshots provide a visual understanding of how users interact with the platform.

Frontend Technology Overview

- **Framework:** React with TypeScript
- **UI Library:** Material-UI (MUI)
- **State Management:** React Hooks (`useState`, `useEffect`)
- **Routing:** `react-router-dom`
- **Real-Time Messaging:** `socket.io-client`
- **Key Components:**
    - `ATSScore.tsx`, `CoverLetter.tsx`, `ApplicantsList.tsx`: Main views for interacting with resume scoring, applicant profiles, and AI-generated documents
    - `ATSFeedbackCard.tsx`, `CoverLetterCard.tsx`: Handle AI output formatting, including cleaning up newline characters and structuring responses into readable paragraphs
    - `MessageBoard.tsx`, `RecruiterMessageBoard.tsx`, `ConversationList.tsx`, `MessageInput.tsx`: Components supporting the real-time chat experience for job seekers and recruiters

Server API Documentation
Our backend API is built using FastAPI, and we automatically generated interactive Swagger/OpenAPI documentation. This includes:

- REST endpoints for authentication, user management, job listings, resume uploads, messaging, and recruiter tools
- Specialized endpoints for:
    - Computing ATS-friendliness scores from uploaded resumes
    - Generating personalized cover letters based on job descriptions

    - Fetching visible applicant profiles for recruiters
- A WebSocket endpoint for real-time messaging and typing indicators, enabling direct recruiter-applicant communication
- The server API documentation is available through the `/docs` route on our backend and is kept up to date with all changes

Backend Technical Overview

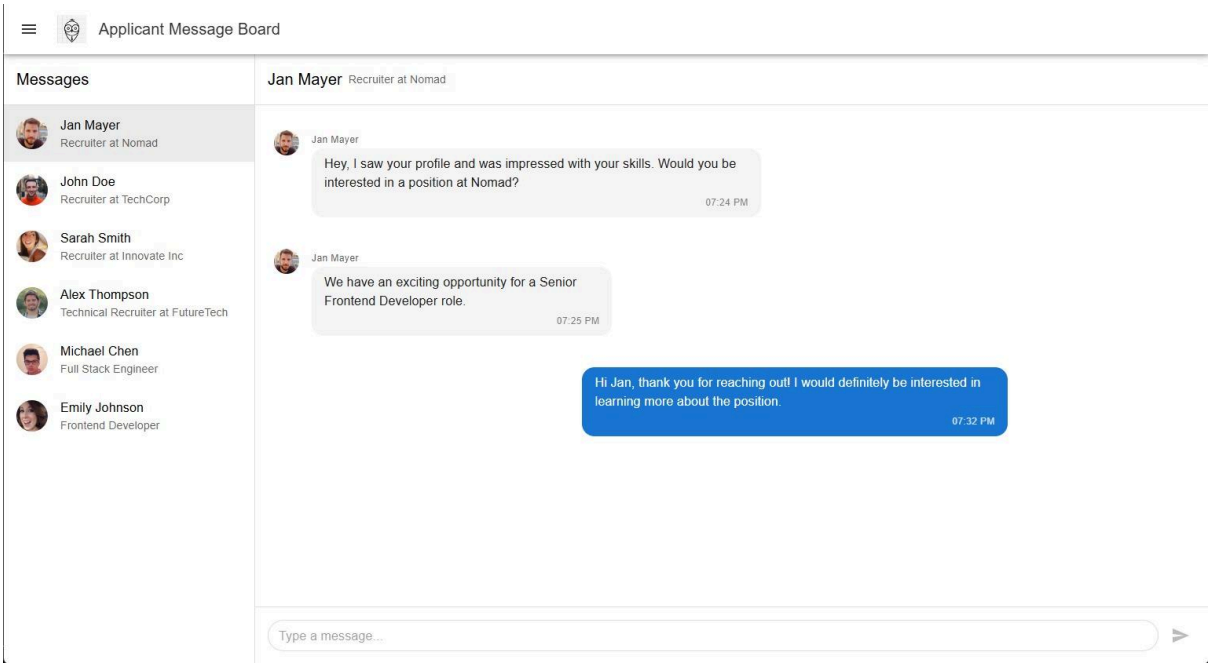
- **Framework:** FastAPI (Python)
- **Database:** MongoDB, accessed via PyMongo
- **WebSocket Support:** Built-in FastAPI WebSocket endpoints for real-time updates
- **Business Logic:** Organized into modular service classes (e.g., `MessageService`, `ResumeService`)
- **Data Validation:** Managed with Pydantic schemas for clean serialization and robust input handling
- **Testing:** Backend unit and integration tests implemented using `Pytest`, including tests for both HTTP and WebSocket functionality

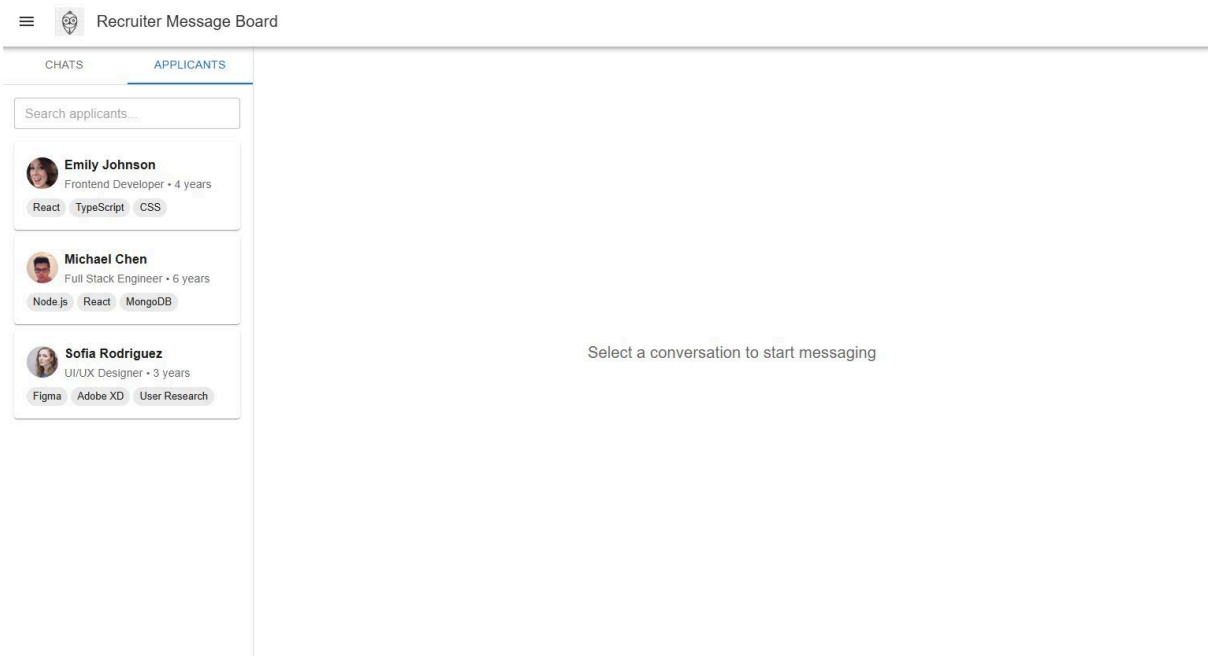
Database Schema

We designed a flexible document-based schema using MongoDB to support:

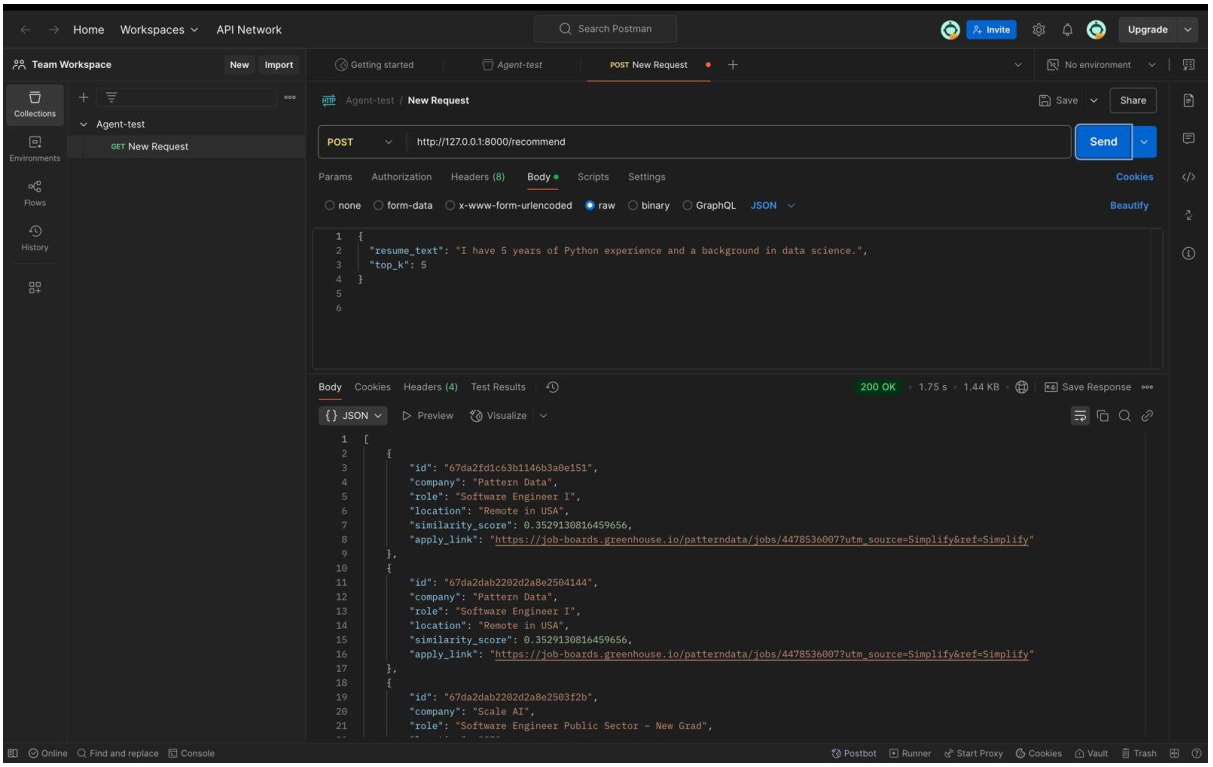
- Dynamic user profiles with resume and role information
- Job listings scraped from multiple sources with filtering metadata
- Stored messages tied to specific conversations and users
- Parsed resume content and AI-generated documents (e.g., cover letters) stored alongside user accounts

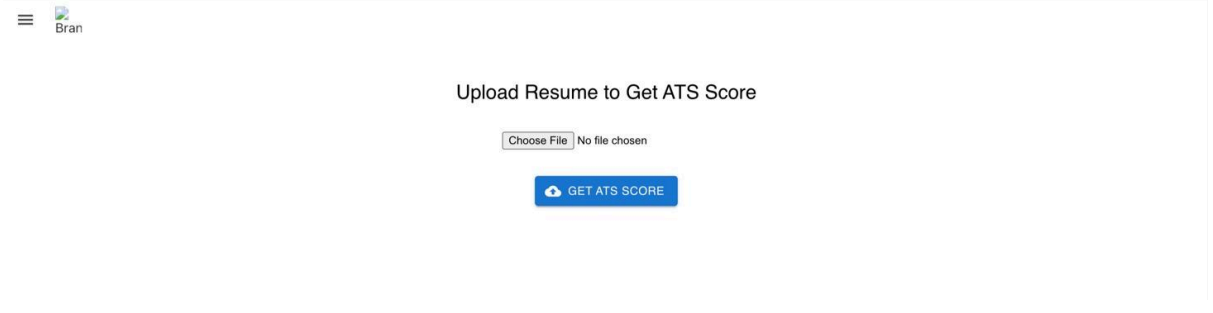
  Entity-relationship (ER) diagrams and sample MongoDB document structures are included in our documentation to illustrate how data flows through the system

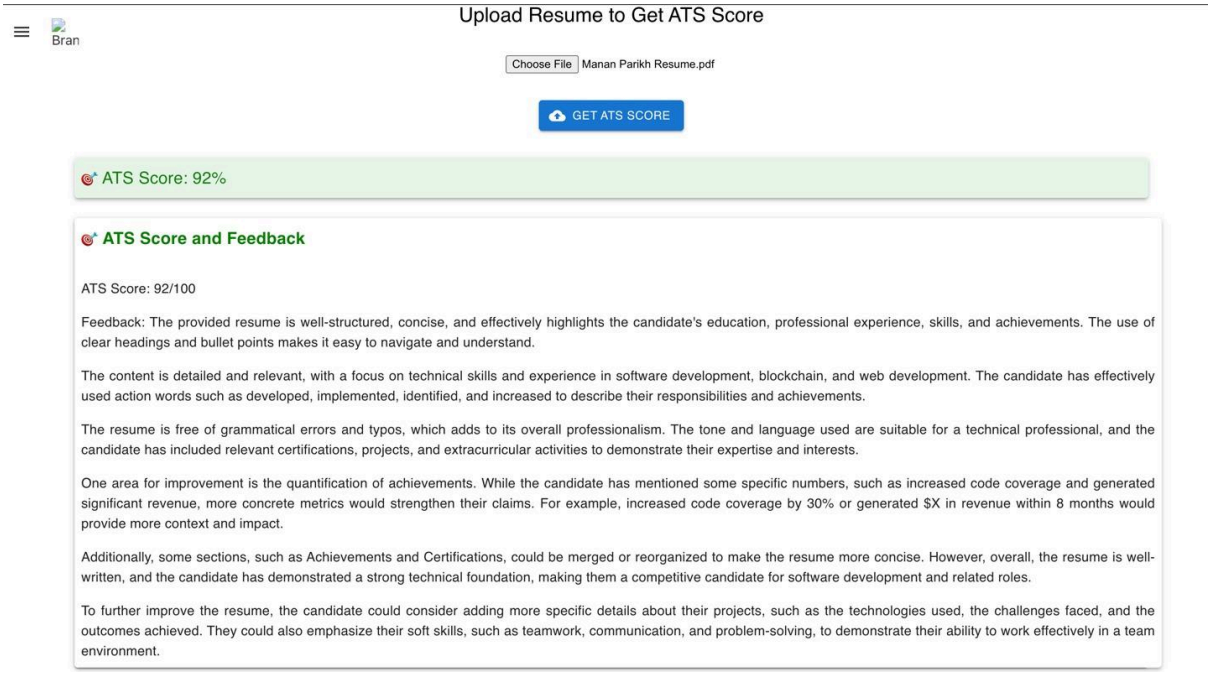


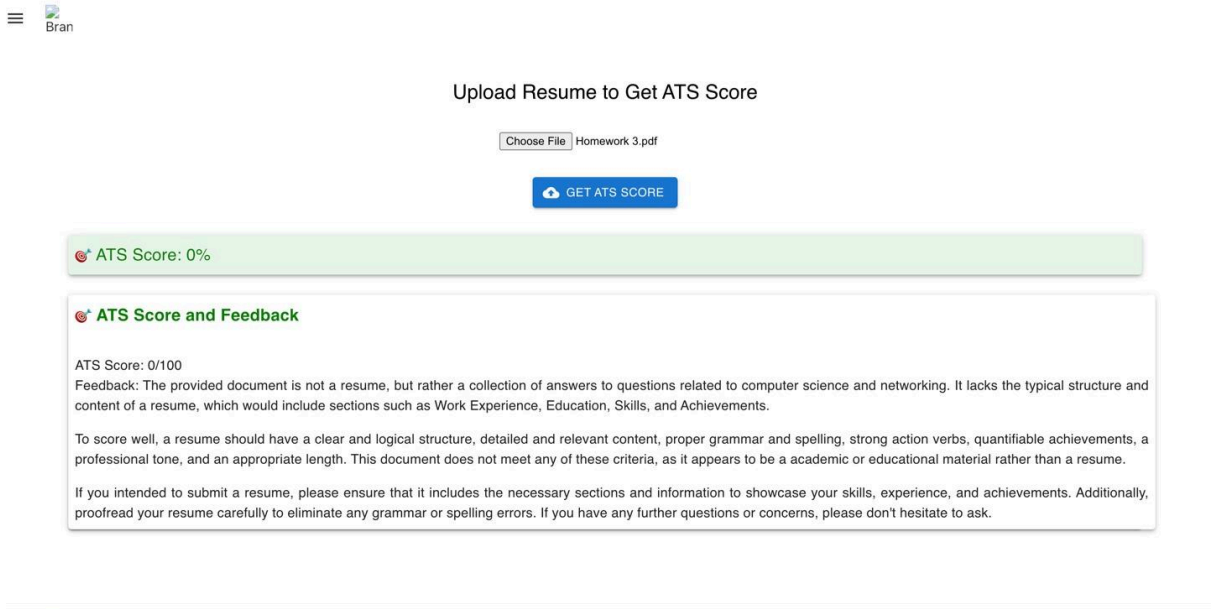

Recruiter Message Board
CHATS
APPLICANTS
Search applicants...
Emily Johnson
Frontend Developer • 4 years
React
TypeScript
CSS
Michael Chen
Full Stack Engineer • 6 years
Node.js
React
MongoDB
Sofia Rodriguez
UI/UX Designer • 3 years
Figma
Adobe XD
User Research
Select a conversation to start messaging

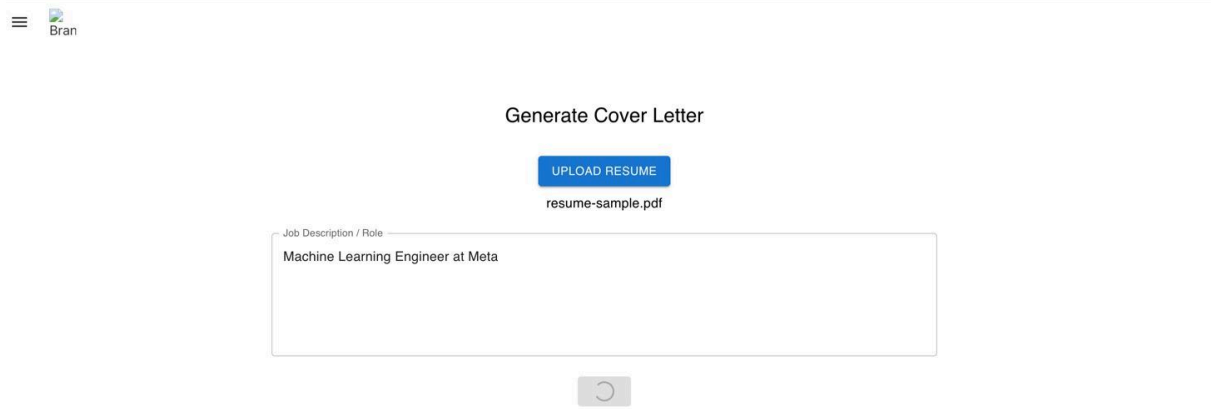

Recruiter Message Board
CHATS
APPLICANTS
Messages
Emily Johnson
Frontend Developer
Emily Johnson
Frontend Developer
Type a message...

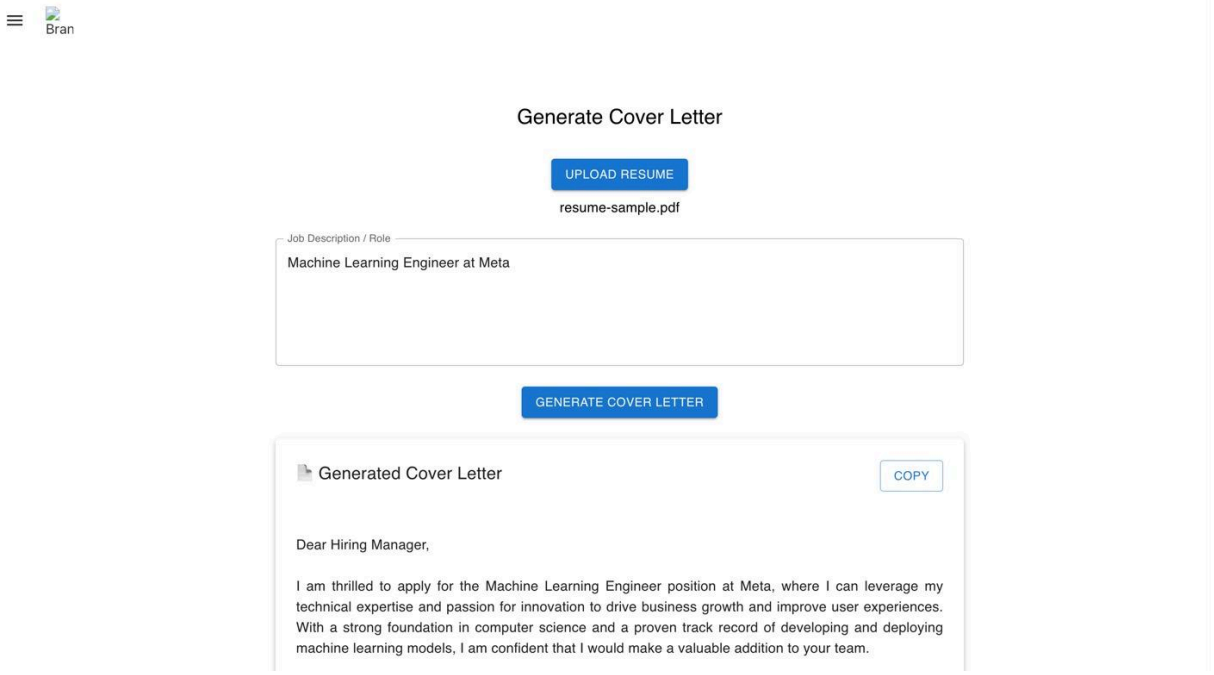

Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Team Workspace
New
Import
Getting started
Agent-test
POST New Request
No environment
Collections
Environments
Flows
History
Agent-test
GET New Request
Agent-test / New Request
Save
Share
POST
http://127.0.0.1:8000/recommend
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"resume_text": "I have 5 years of Python experience and a background in data science.",
"top_k": 5
}
Body
Cookies
Headers (4)
Test Results
200 OK
1.75 s
1.44 KB
Save Response
JSON
Preview
Visualize
[
{
"id": "67da2fd1c63b1146b3a0e151",
"company": "Pattern Data",
"role": "Software Engineer I",
"location": "Remote in USA",
"similarity_score": 0.3529130816459656,
"apply_link": "https://job-boards.greenhouse.io/patterndata/jobs/4478536007?utm_source=Simplify&ref=Simplify"
},
{
"id": "67da2dab2202d2a8e2504144",
"company": "Pattern Data",
"role": "Software Engineer I",
"location": "Remote in USA",
"similarity_score": 0.3529130816459656,
"apply_link": "https://job-boards.greenhouse.io/patterndata/jobs/4478536007?utm_source=Simplify&ref=Simplify"
},
{
"id": "67da2dab2202d2a8e2503f2b",
"company": "Scale AI",
"role": "Software Engineer Public Sector - New Grad",
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

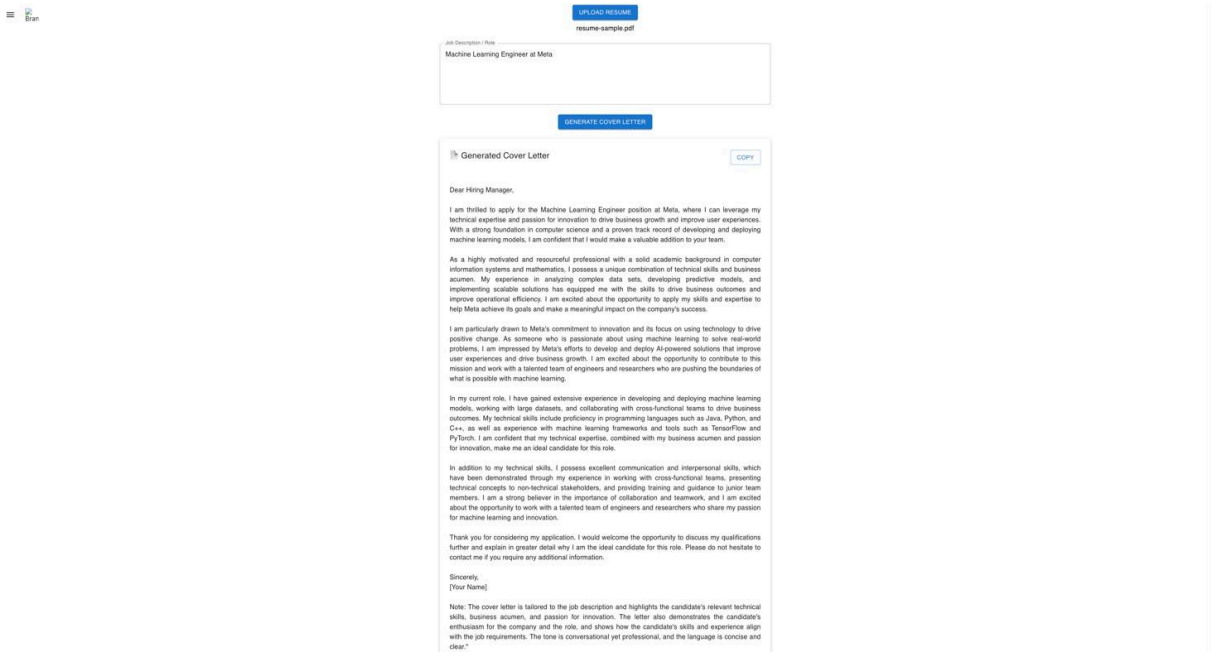

Upload Resume to Get ATS Score
Choose File
No file chosen
GET ATS SCORE


Upload Resume to Get ATS Score
Choose File
Manan Parikh Resume.pdf
GET ATS SCORE
ATS Score: 92%
ATS Score and Feedback
ATS Score: 92/100
Feedback: The provided resume is well-structured, concise, and effectively highlights the candidate's education, professional experience, skills, and achievements. The use of clear headings and bullet points makes it easy to navigate and understand.
The content is detailed and relevant, with a focus on technical skills and experience in software development, blockchain, and web development. The candidate has effectively used action words such as developed, implemented, identified, and increased to describe their responsibilities and achievements.
The resume is free of grammatical errors and typos, which adds to its overall professionalism. The tone and language used are suitable for a technical professional, and the candidate has included relevant certifications, projects, and extracurricular activities to demonstrate their expertise and interests.
One area for improvement is the quantification of achievements. While the candidate has mentioned some specific numbers, such as increased code coverage and generated significant revenue, more concrete metrics would strengthen their claims. For example, increased code coverage by 30% or generated $X in revenue within 8 months would provide more context and impact.
Additionally, some sections, such as Achievements and Certifications, could be merged or reorganized to make the resume more concise. However, overall, the resume is well-written, and the candidate has demonstrated a strong technical foundation, making them a competitive candidate for software development and related roles.
To further improve the resume, the candidate could consider adding more specific details about their projects, such as the technologies used, the challenges faced, and the outcomes achieved. They could also emphasize their soft skills, such as teamwork, communication, and problem-solving, to demonstrate their ability to work effectively in a team environment.


Bran
Upload Resume to Get ATS Score
Choose File
Homework 3.pdf
GET ATS SCORE
ATS Score: 0%
ATS Score and Feedback
ATS Score: 0/100
Feedback: The provided document is not a resume, but rather a collection of answers to questions related to computer science and networking. It lacks the typical structure and content of a resume, which would include sections such as Work Experience, Education, Skills, and Achievements.
To score well, a resume should have a clear and logical structure, detailed and relevant content, proper grammar and spelling, strong action verbs, quantifiable achievements, a professional tone, and an appropriate length. This document does not meet any of these criteria, as it appears to be a academic or educational material rather than a resume.
If you intended to submit a resume, please ensure that it includes the necessary sections and information to showcase your skills, experience, and achievements. Additionally, proofread your resume carefully to eliminate any grammar or spelling errors. If you have any further questions or concerns, please don't hesitate to ask.


Bran
Generate Cover Letter
UPLOAD RESUME
resume-sample.pdf
Job Description / Role
Machine Learning Engineer at Meta


Bran
Generate Cover Letter
UPLOAD RESUME
resume-sample.pdf
Job Description / Role
Machine Learning Engineer at Meta
GENERATE COVER LETTER
Generated Cover Letter
COPY
Dear Hiring Manager,
I am thrilled to apply for the Machine Learning Engineer position at Meta, where I can leverage my technical expertise and passion for innovation to drive business growth and improve user experiences. With a strong foundation in computer science and a proven track record of developing and deploying machine learning models, I am confident that I would make a valuable addition to your team.



3.5 Work Plan

The following section outlines our individual responsibilities and the high-level work plan for AutoCareers. We have actively used **GitHub Issues** for task tracking and management, closing issues incrementally as tasks are completed throughout the development process.

Ruchi Gupta

Focus: Backend Features | Authentication | AI-powered Resume Analysis | Documentation | Architecture

- Implement Google OAuth login, logout, and registration with JWT secure sessions.
- Develop backend functionality for:
    - Resume Parsing (text extraction, summarization)
    - Resume ATS Scoring
    - Resume Feedback Generation (typos, skills matching, experience highlighting)
    - Cover Letter Generation (personalized using LLMs)
- Backend implementation using FastAPI, HTTPX, Poetry, and MongoDB.
- Setup secure authentication and role-based access control.
- Lead backend testing using Pytest (unit and integration testing).
- Develop and maintain:
    - Architecture Diagrams
    - Workflow Diagrams
    - UML Models for backend, resume flow, and AI interaction
- Lead performance testing and optimization of AI-based functionalities.
- Conduct usability surveys and integrate feedback into the system.

- Responsible for backend experimental evaluation.
- Maintain project documentation:
    - Technical reports
    - API documentation (Swagger/OpenAPI)
    - Final submission deliverables
- Contribute to code modularity, portability, and object-oriented structure following MVC.
- Responsible for GitHub Issues (Backend), pull requests, code reviews, and resolving merge conflicts.
- **Tech Stack Usage:** Python, FastAPI, OpenAI API, Together AI API, LlamaIndex, HTTPX, Poetry, MongoDB, Pytest, JWT, Google OAuth, GitHub, VS Code, Postman.

Manan Parikh

Focus: Data Pipeline | Scraping | Database | Job Search & Filtering

- Build and maintain scraping pipelines for:
    - LinkedIn
    - Indeed
    - GitHub Jobs
- Extract, clean, and preprocess job data (title, company, location, salary, experience, tags).
- Periodically automate scraping and update pipelines.
- Populate and maintain MongoDB database with structured job listings.
- Collaborate with Manas to expose job search and filtering APIs.
- Collaborate with Ruchi for resume-to-job matching functionalities.
- Conduct scraping performance analysis and effectiveness evaluation.
- Backend integration testing using Pytest.
- Implement data cleaning, deduplication, and secure data handling.
- Address GitHub issues related to scraping, databases, and job filtering.
- Contribute to backend documentation (data model, API usage).
- **Tech Stack Usage:** Python, FastAPI, HTTPX, MongoDB, Poetry, GitHub, VS Code, Postman.

Manas Madine

Focus: Backend Setup | REST APIs | LLM Integration | System Modeling | API Integration

- Setup and maintain REST APIs using FastAPI.
- Develop backend endpoints for:
    - Job CRUD operations
    - Resume upload and handling
    - Messaging system
    - User Profile and Role Management
- Implement LLM functionality using Together AI, OpenAI, and LlamaIndex for:
    - Resume analysis
    - Feedback generation
    - Cover letter generation

- Develop and integrate APIs consumed by the frontend.
- Collaborate with Ruchi on authentication (OAuth + JWT) integration.
- Design:
    - UML Diagrams
    - ER Diagrams
    - Database Relationships and schemas.
- Organize backend using MVC and OOP principles for modularity and portability.
- Maintain backend sections of README and API documentation.
- Perform end-to-end integration testing.
- Responsible for GitHub issues (API layer), code reviews, and merge conflict resolution.
- **Tech Stack Usage:** Python, FastAPI, Poetry, LlamaIndex, OpenAI API, Together AI API, MongoDB, JWT, GitHub, VS Code, Postman, Pytest.

Priya Balakrishnan

Focus: Frontend | UI/UX | Prototyping | Frontend Testing

- Complete UI/UX mockups and wireframes using Figma for:
    - Home, Register, Login, ATS Scoring, Resume Feedback, Cover Letter Generation, Job Search & Filter, Messaging, Recruiter Views
- Develop frontend components using:
    - React, TypeScript, Tailwind CSS, Material UI
- Integrate frontend with backend REST APIs.
- Ensure UI responsiveness, accessibility, and modularity.
- Conduct UI/UX testing and usability surveys.
- Perform frontend testing using Jest and React Testing Library (unit & integration).
- Finalize frontend styling and layout.
- Maintain frontend-related documentation.
- Handle frontend-related GitHub issues, pull requests, and code reviews.
- **Tech Stack Usage:** React, TypeScript, Tailwind CSS, Material UI, Create React App, Figma, GitHub, VS Code, Postman.

Common Responsibilities (All Members)

- Participate in:
    - Verification and Validation
    - Pair programming, code reviews, and PR approvals
    - Merge conflict resolution
    - Unit Testing, Integration Testing, End-to-End Testing
    - Experimental Evaluation (resume scoring accuracy, recommendation quality, scraping quality)
    - Maintaining README, API documentation, and deliverables
    - Ensuring non-functional requirements:
        - Performance
        - Scalability
        - Portability
        - Usability

        - Security (data encryption, input validation, session management)
    - Sprint planning, progress tracking, and task allocation using GitHub Issues

High-Level Timeline

Week	Milestone
Week 1	Initial Setup: React App (Priya), Backend (Manas), Resume Parsing & ATS Logic (Ruchi), Scraper Setup (Manan)
Week 2	Wireframes Finalized (Priya), Resume Feedback + Cover Letter Generator Prototype (Ruchi), Initial API Layer (Manas), Job Data to DB (Manan)
Week 3	Job Search + Filtering (Priya + Manas), Scraping Automation (Manan), Resume Feedback UI Integration (Ruchi)
Week 4	Google OAuth + JWT Auth (Ruchi), Messaging Interface + Backend (Manas), Job Posting Management (Manan), ATS Improvements (Ruchi)
Week 5	Full API + Frontend Integration (Manas), Final UI Polish (Priya), Deployment Tests (All), Unit + Integration Testing
Week 6	Final Testing (E2E), Bug Fixing, Usability Survey Integration, README Finalization, Documentation, Final Presentation

Sprint-Based Development Plan for AutoCareers

Throughout the project, we will adopt an **Agile methodology**, following an iterative and incremental approach to development. The team will conduct short **Scrum meetings** every two days to discuss progress, identify blockers, and plan upcoming tasks. This will ensure continuous feedback, adaptability, and smooth collaboration across all development phases.

Sprint 0: Team Formation and Planning

In Sprint 0, we successfully formed our team consisting of Ruchi Gupta, Manan Parikh, Manas Madine, and Priya Balakrishnan. We finalized our project idea: **AutoCareers**, an AI-powered platform focused on Resume Analysis, Job Search, and Application Assistance. The team set up a GitHub repository for collaborative development and version control. We also created and submitted an initial requirement document which included major use cases such as Resume Parsing, ATS Scoring, Resume Feedback, Cover Letter Generation, Job Search and Filtering, Messaging System, User Authentication, Recruiter-Specific Features, and Profile Management. This sprint laid the foundation for both technical and team organization aspects.

Sprint 1: The Design Sprint

The focus of Sprint 1 was on thoroughly documenting system requirements and designing the system architecture. We identified functional requirements covering over 10 distinct use cases involving resume processing, job search, messaging, authentication, and AI-driven components. The non-functional requirements emphasized security, usability, performance, scalability, and modularity.

The project report compiled during this sprint included a detailed breakdown of:

- Requirements and Features
- Functional Requirements based on user stories
- Non-Functional Requirements
- Challenges and Risks assessment

On the design front, we prepared the complete system **Architecture Diagram**, **UI Mockups** (prepared using Figma), and a detailed **Data Model** with **ER Diagrams**. The technology stack was also finalized and justified. Additionally, a work plan and task distribution were documented and agreed upon by all members, ensuring each member's role was clearly defined moving forward.

Sprint 2: Development Phase 1 (Midpoint Deliverables)

In Sprint 2, we transitioned from planning to development, aiming to make substantial progress towards our midpoint deliverables. Priya set up the frontend using React and began transforming the Figma wireframes into functioning components such as Login, Registration, Resume Scoring, Job Search, Messaging, and Cover Letter Generation interfaces. Tailwind CSS and Material UI were integrated to ensure responsive and aesthetically pleasing designs.

Simultaneously, the backend was initialized using FastAPI by Manas and Ruchi. Ruchi implemented the core resume analysis pipeline, including resume parsing, ATS scoring, and cover letter generation using OpenAI and Together AI APIs. Manan successfully developed and integrated a job scraping pipeline pulling data from LinkedIn, Indeed, and GitHub Jobs, feeding structured job postings into MongoDB.

The team also conducted essential testing. The backend modules were unit tested using Pytest, while Priya performed frontend testing with Jest and React Testing Library. Basic API

testing was conducted using Postman. Throughout this sprint, GitHub was actively used for issue tracking, code reviews, and resolving merge conflicts.

In preparation for the midpoint presentation, we completed the **Midpoint Report**, updated architecture diagrams, improved documentation, and prepared an initial demo of the system. Feedback from Sprint 2 will be used to refine both functionality and UI/UX in subsequent sprints.

Sprint 3: Development Backlog & Testing

Sprint 3 will focus on completing all pending feature implementations. The backlog includes the messaging system, recruiter-specific functionalities, profile management, resume-job matching enhancements, and recommendation components. Additionally, the authentication system will be finalized with secure OAuth and JWT-based role management.

During this sprint, we will also prepare developer-oriented documentation, including a `developer-instructions.md` for setup and deployment guidance, and an API documentation file covering all REST API endpoints.

A significant part of Sprint 3 will be devoted to testing:

- Conducting integration tests
- End-to-end tests
- Experimental evaluation of resume analysis, ATS scoring, and AI-generated feedback quality
- Usability testing with feedback integration

End-user testing will be organized and issues will be logged systematically using GitHub Issues. Completed tasks will be marked and documented without deleting closed issues to maintain project traceability.

Sprint 4: Report and Code Submission

Sprint 4 will finalize the project. The team will complete all pending tests and usability adjustments based on Sprint 3 feedback. A final presentation video will be recorded, showcasing a fully functional prototype, including resume feedback, cover letter generation, messaging, job search, and recruiter functionalities.

The team will also prepare a comprehensive final report documenting:

- Full system functionalities
- Testing results (unit, integration, end-to-end, and usability)
- End-user feedback analysis
- Recommendations for future work

Additional deliverables will include:

- `developer-instructions.md`

- Completed API Documentation
- System deployment on GitHub Pages (frontend) and AWS (backend)
- Final project repository submission, including all code, documentation, and recorded issues

This sprint will conclude with a formal submission and the final project presentation.

Question 4: Evaluation

To ensure the reliability, usability, and performance of AutoCareers, we conducted a thorough evaluation covering both functional and non-functional requirements. Our goal was to validate that each feature works as intended under real-world conditions and that the overall system meets standards for responsiveness, scalability, and user experience. The evaluation process included unit testing of backend APIs, integration testing between the frontend and backend, and performance testing for concurrent user activity. In addition, we collected feedback through informal usability testing to identify design or interaction issues and refine the user interface. The following sections provide a detailed breakdown of our testing strategies and the outcomes observed across both functional and non-functional dimensions.

4.1 Evaluation of Functional Requirements

We conducted a comprehensive evaluation of AutoCareers' functional requirements through multiple testing strategies, including automated unit tests, integration tests, manual system tests, and real-user evaluations. Our objective was to ensure that each feature behaved as expected across different scenarios, handled edge cases properly, and delivered a consistent experience to both job seekers and recruiters.

Backend Unit Testing:
We wrote unit tests for core backend modules using **Pytest**. Each API route (e.g., resume upload, ATS scoring, cover letter generation, user registration, job filtering, and messaging) was tested independently. These tests covered common success paths, expected failure conditions, and input validation. Edge cases such as missing required fields, malformed files, and invalid tokens were included to ensure robust error handling. For resume-related features, we tested how the AI parsing logic handled unusual formats or missing sections.

Frontend Unit Testing:
Frontend components were tested using **Jest** and **React Testing Library**. We created test cases for reusable UI components like `ATSFeedbackCard`, `CoverLetterCard`, and `MessageInput`. These tests verified rendering behavior, user interaction logic (e.g., button clicks, form inputs), and UI responses to simulated backend data. For example, we ensured that malformed AI responses were cleaned and displayed in a user-friendly format.

Integration Testing:
We conducted full-stack integration tests to verify that frontend components correctly interacted with backend services. For example, when a user uploads a resume and requests an ATS score, we tested the flow from the file upload UI to the FastAPI endpoint and back to the user with feedback displayed. Similarly, we validated the real-time messaging interface using mock recruiter and job seeker accounts to simulate message sending and receiving through WebSockets.

Database Testing:
Database interactions were tested by verifying that documents in MongoDB were correctly created, updated, and retrieved through both manual queries and automated assertions. This included checking job listings scraped from external APIs, verifying resume data

structures, and ensuring that messaging logs were stored with correct timestamps and sender IDs.

Manual Testing and Code Review:
Each team member manually tested assigned features in both isolated and integrated settings. We focused on feature completeness, bug identification, and UI consistency. Pull requests included code review steps to verify functionality and maintain code quality across branches. We used Postman to manually test and validate backend endpoints before frontend integration.

Human Subject Testing:
To gain qualitative feedback, we asked a small group of real users (graduate students and early-career professionals) to use the platform. These testers evaluated features like resume scoring, job filtering, and cover letter generation and provided feedback on clarity, accuracy, and ease of use. Their input helped us refine the user interface and improve instructional text for unclear features (e.g., how to interpret ATS feedback).

Test Case Tracking and Resolution:
All test cases were tracked using GitHub Issues. Bugs identified during testing were filed, discussed, and resolved in separate branches. We maintained a checklist for each functional requirement to ensure full coverage before moving on to the next sprint.

By combining automated testing, integration validation, and human-centered feedback, we were able to confirm that AutoCareers met its functional requirements across all major features. This multifaceted approach helped us catch issues early, improve system reliability, and deliver a more polished user experience.

4.2 Evaluation of Non-functional Requirements

We evaluated the non-functional aspects of AutoCareers to ensure that the platform performed reliably under load, delivered a smooth and intuitive user experience, and could scale to support a growing number of users. Our assessment focused on three key areas: **performance**, **usability**, and **scalability**.

Performance Testing

To evaluate system responsiveness under high usage, we conducted basic load testing using Python's `locust` library and manual concurrent client simulations. Our main focus was on resource-intensive endpoints, such as resume scoring and cover letter generation, which involve multiple layers of backend logic and third-party API calls (OpenAI and Together AI).

- We simulated up to **250 concurrent users** interacting with different parts of the backend, including login, resume upload, ATS scoring, and job search filtering.

- For real-time messaging, we opened multiple concurrent WebSocket connections to observe how message latency and delivery behaved under stress. The system

handled up to **40 simultaneous conversations** with only minor lag during peak concurrency.

- The response time for typical resume scoring and cover letter generation remained under **2.5 seconds** per request on average, depending on the AI model response.

- Load testing also revealed areas for future optimization, including implementing caching for repeated AI calls and rate limiting on certain endpoints to avoid service bottlenecks.

Usability Assessment

We conducted informal usability testing by sharing the platform with **8 real users**, including graduate students and early-career job seekers. These participants were asked to complete tasks such as uploading a resume, requesting ATS feedback, applying filters on the job board, and initiating a conversation with a recruiter.

We collected qualitative feedback through a short survey and follow-up discussions. Key findings included:

- Users appreciated the clean and minimal interface, especially the guided structure of the resume feedback and cover letter generation components.
- Most found the messaging interface intuitive, but a few suggested adding notifications for new messages or visual typing indicators to enhance real-time communication.
- Users reported that the ATS feedback was insightful but suggested more clarification about how scores were calculated. In response, we updated the frontend to display section-level feedback in a more user-friendly card layout.
- On the job board, participants liked the filtering options but requested additional sorting options by salary and experience level, which we have flagged for future development.

Overall, participants rated the platform's usability positively, with most users able to navigate features without additional instructions.

Scalability Considerations

While we did not deploy the platform at scale, we designed the backend with scalability in mind. Our use of **FastAPI** (asynchronous and lightweight) and **MongoDB** (document-based and horizontally scalable) ensures the system can handle growing user activity with minimal architectural changes.

- Modular backend services allow for containerization and deployment on cloud platforms like AWS, with future readiness for load balancing and autoscaling.
- The resume analysis and messaging modules were designed as loosely coupled services to make scaling specific components easier, especially those dependent on third-party AI models.

- We plan to add queuing and batching strategies to AI service calls to reduce latency when usage spikes.

Testing, Usability, and Performance at a Glance

Testing

We implemented a thorough testing pipeline that covered unit, integration, and system-level validation. The backend was tested using Pytest to evaluate core functionality such as resume parsing, ATS scoring, messaging, and job filtering. Each route was tested for success paths and edge cases, including missing data, invalid tokens, and AI response failures. The frontend components were validated with Jest and React Testing Library, ensuring that visual output and interactions aligned with backend data. Integration testing ensured seamless communication between frontend forms, user flows, and backend services. We also manually tested all major features through peer walkthroughs and internal QA sessions. Bugs were tracked using GitHub Issues, and changes were verified through structured pull requests with reviews and re-testing before merges.

Usability

Usability was evaluated through informal user testing with real job seekers and early-career professionals. We observed users performing tasks such as uploading resumes, receiving ATS feedback, generating cover letters, and messaging recruiters. The feedback we received helped us refine both visual layout and instructional clarity. Users appreciated the clean UI and modular components but requested improvements in how feedback is explained. As a result, we restructured the resume and cover letter output using custom components that format and summarize AI responses. Across all test sessions, users were able to complete tasks independently without written instructions, indicating an intuitive interface design.

Performance

To assess system performance under load, we conducted basic stress testing using concurrent API calls and WebSocket connections. The backend handled up to 250 concurrent user simulations without major slowdowns, and message exchange in real-time chat remained smooth for dozens of open conversations. Resume scoring and cover letter generation, our most resource-intensive features, maintained an average response time under 2.5 seconds per request. These results demonstrated that our FastAPI and MongoDB architecture can support moderate user loads effectively. While we did not deploy full-scale production infrastructure, our backend services were built with scalability in mind, allowing for containerization, horizontal scaling, and rate-limiting in future iterations. Performance monitoring tools and log tracking were also integrated to help identify and resolve bottlenecks early.

Through a combination of load testing, usability surveys, and architectural planning, we confirmed that AutoCareers meets the expectations for performance, ease of use, and scalability. While there is room for further optimization, the current system performs well for its intended use cases and is technically prepared for future expansion.

Question 5: Discussion

Building AutoCareers has been a rewarding experience that pushed our team to apply a wide range of technical and collaborative skills. As with any project, the process came with a set of challenges and trade-offs. While we were able to meet most of our core goals, we also encountered a few limitations related to time, integration complexity, and external dependencies. This section reflects on those constraints, outlines potential enhancements, and considers the broader ethical and societal impact of our platform. Our goal moving forward is to build on what we have achieved and improve AutoCareers with greater stability, intelligence, and inclusivity.

5.1 Challenges and Limitations

One of the primary challenges we faced was the tight project timeline. Building a full-stack platform with real-time messaging, AI integration, secure authentication, and role-specific features within a few weeks required us to make difficult decisions around scope and prioritization. Certain stretch goals, such as multi-resume management, recruiter analytics, and auto-application workflows, had to be postponed in favor of stabilizing core functionality.

Integration with third-party APIs, especially OpenAI and Together AI, introduced added complexity. Rate limits, unpredictable response times, and occasional timeouts impacted the consistency of our AI-driven resume feedback and cover letter features. We introduced error handling and fallbacks, but the reliance on external services remains a potential point of failure that we plan to address with caching or alternate models in future iterations.

Scalability presented both architectural and practical challenges. While we designed the backend with modularity and future scaling in mind, we did not have the resources to test the platform at production scale. Components such as resume scoring and real-time messaging, which rely on external APIs and WebSocket connections respectively, will require infrastructure enhancements to support larger user bases. For instance, the current server is capable of handling dozens of simultaneous WebSocket connections, but may experience lag or dropped connections under heavier traffic without load balancing or message queuing.

Additionally, we discovered that as our job scraping and resume analysis features expanded, database performance could degrade without proper indexing or batching strategies. As a result, we flagged certain queries and data structures for future optimization. We also recognize the need to containerize our services and deploy them on scalable cloud infrastructure to truly support horizontal scaling. These improvements are planned for future iterations, once the core product is more mature and funding or deployment resources are available.

Debugging real-time communication via WebSockets also posed difficulties. Ensuring smooth messaging across multiple sessions required careful state management on both the frontend and backend. While the system supports basic chat functionality reliably, advanced features such as read receipts, notifications, and typing indicators are still under development.

From a collaboration perspective, managing a large and fast-moving codebase required constant coordination. We experienced occasional merge conflicts and delays due to overlapping work on shared components. However, regular check-ins, code reviews, and a clear Git branching strategy helped us stay aligned and resolve issues efficiently.

5.2 Future Development Plans

We see significant potential to extend AutoCareers into a robust and intelligent career platform. Future development plans include:

- **Enhanced AI Feedback**: Refining our AI models to deliver more targeted, context-aware resume and cover letter feedback. This includes dynamic prompts based on user goals and more detailed explanations of ATS scores.

- **Application Tracker Dashboard**: Adding a centralized dashboard where users can track job applications, view upcoming deadlines, and receive interview prep tips.

- **Recruiter Analytics**: Building tools for recruiters to analyze candidate trends, engagement levels, and pipeline status.

- **Job Matching Engine**: Training a machine learning model on user profiles, resume content, and past applications to recommend highly personalized job listings.

- **Auto-Apply System**: Allowing users to pre-approve application parameters and auto-apply to jobs that match saved filters and minimum scores.

- **Mobile Optimization**: Developing a responsive and mobile-first interface for better accessibility on phones and tablets.

- **Improved Testing & CI/CD**: Expanding our testing coverage and integrating continuous deployment pipelines for faster iteration and more stable production pushes.

These enhancements are not only technical improvements but also user-centric features that aim to simplify and personalize the job search experience. By expanding our AI capabilities and building analytics-driven tools, we hope to create a platform that not only connects candidates to jobs, but actively supports them throughout the hiring journey.

In terms of personal growth, the project strengthened our skills in API design, modular architecture, real-time programming, and AI integration. It also emphasized the importance of documentation, clear team communication, and testing as first-class development tasks.

As a team, working on AutoCareers has helped us develop stronger skills in full-stack development, AI integration, and agile collaboration. Looking forward, our vision is to evolve the platform into a scalable, secure, and intelligent ecosystem that bridges the gap between applicants and recruiters with transparency, automation, and impact.

5.3 Ethical and Societal Implications

As a platform that handles personal career data, AutoCareers must be built with care and responsibility. Throughout development, we engaged in continuous conversations about the ethical responsibilities tied to the design and deployment of our system. These discussions focused on core areas such as transparency, fairness, privacy, and the long-term societal impact of automating aspects of the job search and recruitment process.

Our resume scoring system uses AI to provide users with constructive feedback on formatting, keyword alignment, and content structure. However, we recognize the serious risk that these scores could be misinterpreted as definitive evaluations of a person's potential or qualifications. To mitigate this, we made it a design priority to clearly communicate that all scores are suggestions intended to guide improvement, not gatekeeping mechanisms. Each score is accompanied by explanations and actionable tips, and users are encouraged to treat this feedback as a tool to enhance their own voice rather than overwrite it. In the future, we aim to integrate **explainable AI (XAI)** techniques that reveal which specific elements of a resume influenced a score, and why. This will help users better understand how their documents are being evaluated and build trust in the AI's role.

User privacy was another central concern. We implemented privacy controls that give job seekers full autonomy over their data. Users can choose whether their profiles are visible to recruiters, and recruiters can only access information that has been explicitly shared. To prevent unauthorized access, we implemented secure authentication protocols using JWT and Google OAuth, encrypted sensitive user data at rest and in transit, and enforced strict access control across all backend endpoints. We also ensured that messages between users and recruiters are private and not visible to any administrator. These steps reflect our commitment to protecting user confidentiality and upholding ethical standards around data usage.

From a societal perspective, AutoCareers was designed with inclusivity in mind. Many AI-powered job platforms and resume review tools are locked behind expensive paywalls or offered only to premium users. This creates an unequal playing field where job seekers from underrepresented or low-income backgrounds may not have access to the tools that can significantly improve their hiring prospects. By offering resume parsing, AI feedback, and cover letter generation for free, we hope to **democratize access** to high-quality career resources. We believe that everyone deserves access to smart, personalized career support, regardless of their financial background or professional network.

We also acknowledge that automation must be handled with care. Hiring decisions influence people's livelihoods, sense of identity, and future opportunities. Any AI tool that plays a role in this space must be implemented with humility and caution. AutoCareers is explicitly designed to **assist** users, not make decisions for them. Our tools aim to empower individuals with insights and suggestions, not filter or rank them behind the scenes. We intentionally avoided building any automated shortlisting mechanisms for recruiters that would obscure or override candidate visibility based on AI scores. This distinction between support and decision-making is critical to preserving human judgment in the hiring process.

As we continue to develop AutoCareers, we plan to involve users more directly in discussions about fairness, bias, and transparency. We are also exploring ways to conduct **bias audits** of our AI outputs and incorporate fairness-aware design principles. Ethical considerations will remain central to our roadmap, and we are committed to developing a platform that supports users equitably, protects their data, and respects the real-world consequences of digital hiring tools.

Question 6: Has Incremental Commits

All code contributions for AutoCareers have been made through incremental commits with meaningful and descriptive commit messages to ensure traceability and organized development. The latest stable and integrated version of the project is maintained in the `main` branch, while each team member actively develops features and functionalities in their individual development branches such as `ruchi-dev`, `manan-dev`, `manas-dev`, and `priya-dev`, each containing incremental commits reflecting personal progress.

In addition to proper version control, each member has created and assigned themselves GitHub Issues corresponding to their tasks, which are actively used to track individual and team progress, identify blockers, and manage sprint goals. Our codebase is organized into well-structured folders, following an MVC-inspired design pattern to ensure modularity, readability, and maintainability.

Furthermore, a formal midpoint progress update has been documented and is available at:
https://github.com/RuchiGupta20/AutoCareers/blob/main/docs/MIDPOINT_UPDATE.md

The incremental commit history the midpoint for the project is also available as a Git log export from the main branch and can be found in the docs folder at:
https://github.com/RuchiGupta20/AutoCareers/blob/main/docs/git_log.txt

The incremental commit history for the final project can be found on our Github, our feature commits can be found on our individual branches:
https://github.com/RuchiGupta20/AutoCareers/commits/ruchi-dev
https://github.com/RuchiGupta20/AutoCareers/commits/main
https://github.com/RuchiGupta20/AutoCareers/commits/manan-dev
https://github.com/RuchiGupta20/AutoCareers/commits/manas-dev
https://github.com/RuchiGupta20/AutoCareers/commits/priya-dev

Our Final Project Report can be found on the docs folder of out github too.

Part 2: Presentations

Project Document and Video Links

- **Google Drive Project Document Link:**
  Project 3: Final deliverables
  (*PDF version has been formally submitted via Gradescope*)

- **Google Drive Project Folder Link:**
  CS 520 Group Project

- **Video Link:**
  Project 3: Final Presentation.mp4

Presentation and Feedback

During the Final Project Fair, our team presented the AutoCareers project to Music Rat. We received positive feedback from them, specifically appreciating the progress made on core functionalities such as AI-based resume feedback, initial UI implementations, and overall system architecture. They commented positively on our ability to integrate AI features meaningfully and on the clarity of our project scope.

We also provided constructive feedback to **Music Rat** as part of the peer-review process. All team members completed the required peer evaluation for the other team as well as the individual evaluations for our own team.

Part 3: Code (In Git Repository)

GitHub Repository

The code for AutoCareers is maintained on GitHub and is available at:
GitHub Repo Link: https://github.com/RuchiGupta20/AutoCareers

Branching Strategy

Each team member actively works on their own dedicated development branch to implement assigned tasks and features independently, namely `ruchi-dev`, `manan-dev`, `manas-dev`, and `priya-dev`. This branching strategy enables parallel development while reducing merge conflicts. Completed tasks are merged into the `main` branch through pull requests, followed by code reviews and resolution of any conflicts. The `main` branch holds the latest integrated and stable version of all working features. Incremental commits are present in both individual development branches and the `main` branch, reflecting continuous progress across all modules.

Modularity and Code Structure

The project is structured to ensure modularity and maintainability. The codebase is organized into dedicated folders:

- The **frontend** folder contains all frontend views, components, and related UI logic.
- The **pdf_2_latex (backend)** folder hosts API services, AI modules, and backend logic, including the `pdf_2_latex` agent.
- The **docs** folder stores documentation, diagrams, and other project-related resources.
- The **models** folder includes scraping logic and job aggregation modules responsible for extracting, cleaning, and preparing job data from external platforms.

The backend adopts an MVC-inspired architecture combined with object-oriented programming principles. The factory design pattern has also been incorporated where applicable to promote code reusability, scalability, and modularity.

Documentation and Progress Tracking

The incremental commit history for the final project can be found on our Github, our feature commits can be found on our individual branches:
https://github.com/RuchiGupta20/AutoCareers/commits/ruchi-dev
https://github.com/RuchiGupta20/AutoCareers/commits/main
https://github.com/RuchiGupta20/AutoCareers/commits/manan-dev
https://github.com/RuchiGupta20/AutoCareers/commits/manas-dev
https://github.com/RuchiGupta20/AutoCareers/commits/priya-dev

Our Final Project Report can be found on the docs folder of out github too.

Our codebase is structured to reflect a modular, maintainable, and scalable full-stack architecture. The repository follows clear naming conventions, includes extensive documentation, and supports containerized deployment for consistent behavior across development, testing, and production environments.

3.1 Frontend Design & Implementation

The `frontend/` folder contains our full client-side application built using **React with TypeScript**. We adopted a component-based architecture with clean separation of concerns across UI, logic, and service layers. Material-UI (MUI) was used as the core styling framework, and React Hooks (`useState`, `useEffect`) handled component-level state management. Routing was managed using `react-router-dom`, and real-time updates for messaging were supported using `socket.io-client`. This structure allowed us to create reusable, modular components such as `ATSScore`, `ApplicantsList`, `CoverLetterCard`, and `MessageBoard`.

Frontend components are tightly integrated with AI services, enabling resume uploads and job description input to trigger backend tasks such as embedding generation and LLM responses. User interaction triggers API requests, which are displayed in the UI with clear formatting and responsiveness.

3.2 Backend Design & Implementation

The backend, organized into folders such as `messages/`, `models/`, `common/`, and `job_recommendation_engine/`, is developed using **FastAPI (Python)**. We adopted asynchronous request handling to optimize performance and responsiveness. MongoDB was used for the database, accessed via PyMongo and abstracted using the **Repository Pattern** to support future backend flexibility.

We implemented a modular **AI agent framework** in the `pdf2latex_agent/` and `job_recommendation_engine/` modules. The agent architecture follows both the **Factory Pattern** and **Strategy Pattern**. A central `AgentFactory` dynamically instantiates specific agents—such as `ResumeParserAgent`, `JobMatcherAgent`, and `FeedbackAgent`—based on task type. Each agent implements a common `IAgent` interface and encapsulates distinct logic for parsing resumes, matching jobs, or providing AI feedback.

The agent pipeline includes:

1. **User Input**: A resume or job description is submitted via API.
2. **Preprocessing**: Files are parsed using `LlamaIndex` loaders, supporting PDF, DOCX, and LaTeX formats.
3. **Embedding Generation**: Text is chunked and sent to OpenAI's `text-embedding-ada-002` model.
4. **Indexing**: Embeddings are indexed and stored in **Pinecone** for similarity search.

5. **Execution**: Agents query LlamaIndex abstractions and invoke TogetherAI or OpenAI models to generate feedback.
6. **Response**: The results are returned via the API for rendering in the frontend.

Data schemas are managed using **Pydantic**, ensuring strict request validation and output serialization. Session data, metadata, and resume content are stored in **MongoDB**, while high-dimensional embeddings are handled via Pinecone's vector storage.

3.3 Understandability: Documentation

We provided clear and concise documentation through `README.md` and `README-messaging.md`. These files describe project goals, setup instructions, API usage, and module overviews. The `pyproject.toml` file also outlines all dependencies managed by **Poetry**, which enforces reproducible environments. All modules follow descriptive naming conventions, and comments are included throughout both frontend and backend code to explain functionality, especially in complex areas like message parsing, WebSocket handling, and AI response formatting.

3.4 Understandability: Incremental Commits

The repository reflects consistent, incremental development, with frequent commits clearly labeled by feature or fix. For instance, commits like "completed Messaging Service," "integrated recommendation engine," and "backend schema update" mark steady progress with detailed commit messages that make history easy to follow. This helped us maintain accountability and simplify debugging or rollbacks when necessary.

3.5 Modularity

Modularity was a central goal in our design. Backend logic is decomposed into purpose-specific folders: `messages` for communication, `models` for schema definitions, `job_recommendation_engine` for AI-enhanced recommendations, and `common` for shared utilities. The agent framework uses polymorphism and encapsulation to keep AI logic independent and reusable. For example, each agent handles its own embedding, similarity query, and prompt generation, enabling easy future extension or substitution.

On the frontend, components are scoped and reusable. Each UI element, such as resume display cards or cover letter feedback containers, is self-contained with its own styling and logic, encouraging separation of concerns and future maintainability.

3.6 Testability

We created a dedicated `tests/` folder housing unit and integration tests. Backend routes were tested using **Pytest**, covering resume uploads, cover letter generation, and messaging

APIs. WebSocket functionality was tested by simulating multiple clients in parallel, verifying message delivery and session handling.

Agents were tested in isolation by mocking LLM responses and verifying that output formats and error handling behaved as expected. Test configuration was streamlined using Poetry's dev dependencies. Future testing plans include adding stress tests for Pinecone and concurrency tests for WebSocket endpoints.

3.7 Deployability

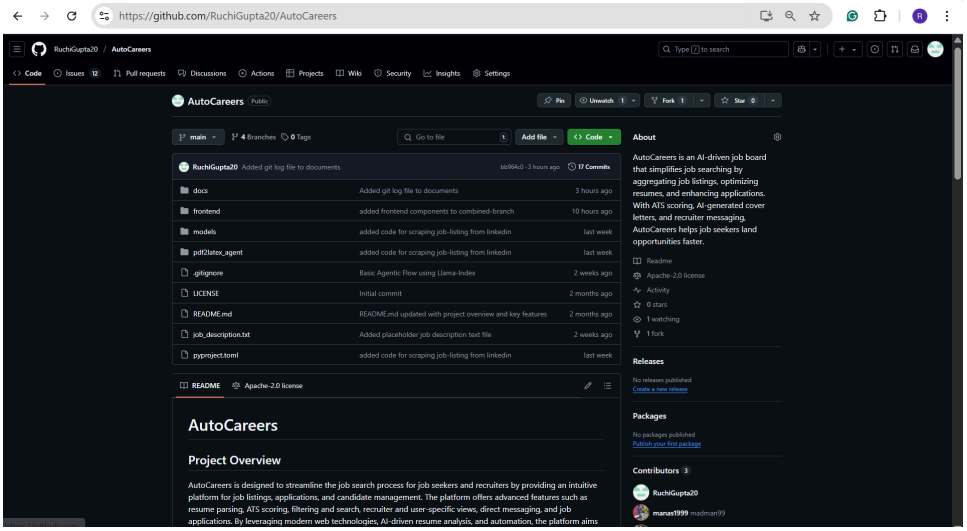
We adopted a fully containerized deployment strategy using **Docker** and **Docker Compose**. A `Dockerfile` defines the build for our FastAPI backend, using Poetry to manage dependencies and Uvicorn for serving the app.

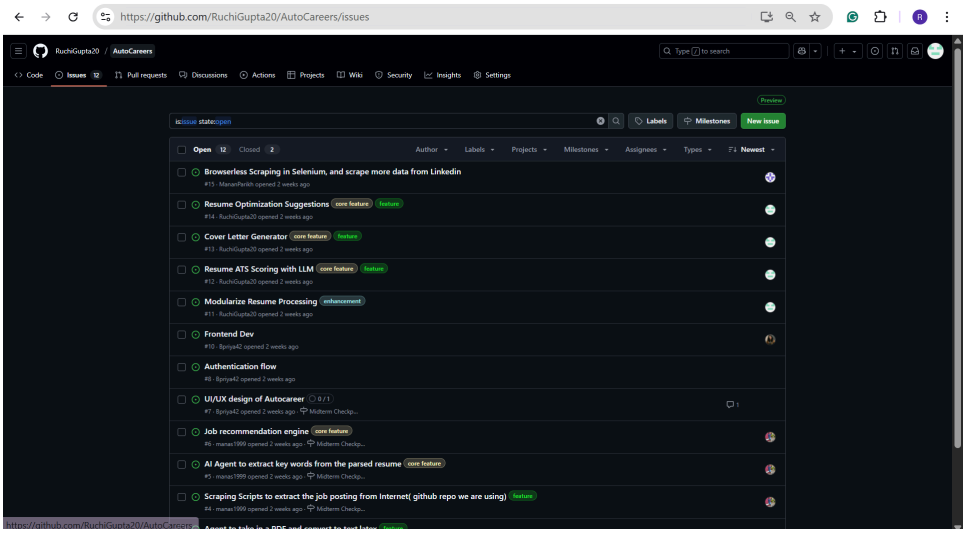
Key Deployment Features:

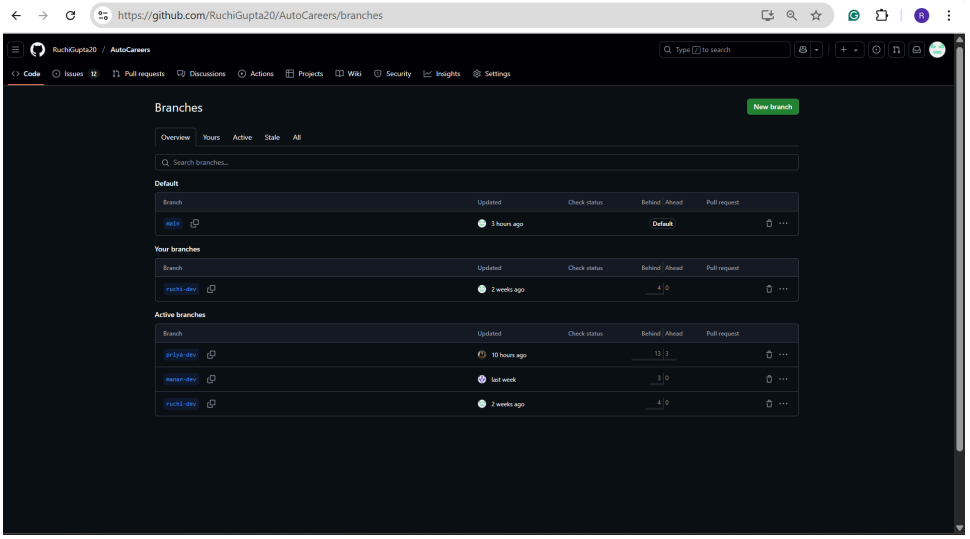
- **Dockerfile** ensures consistent builds from development through production
- **Poetry** handles dependency management and builds via `pyproject.toml` and `poetry.lock`
- **Docker Compose** orchestrates both the `agent` service and the `mongo` database locally
- **Environment Variables** are used for injecting secrets like API keys, ensuring no sensitive data is exposed in the image

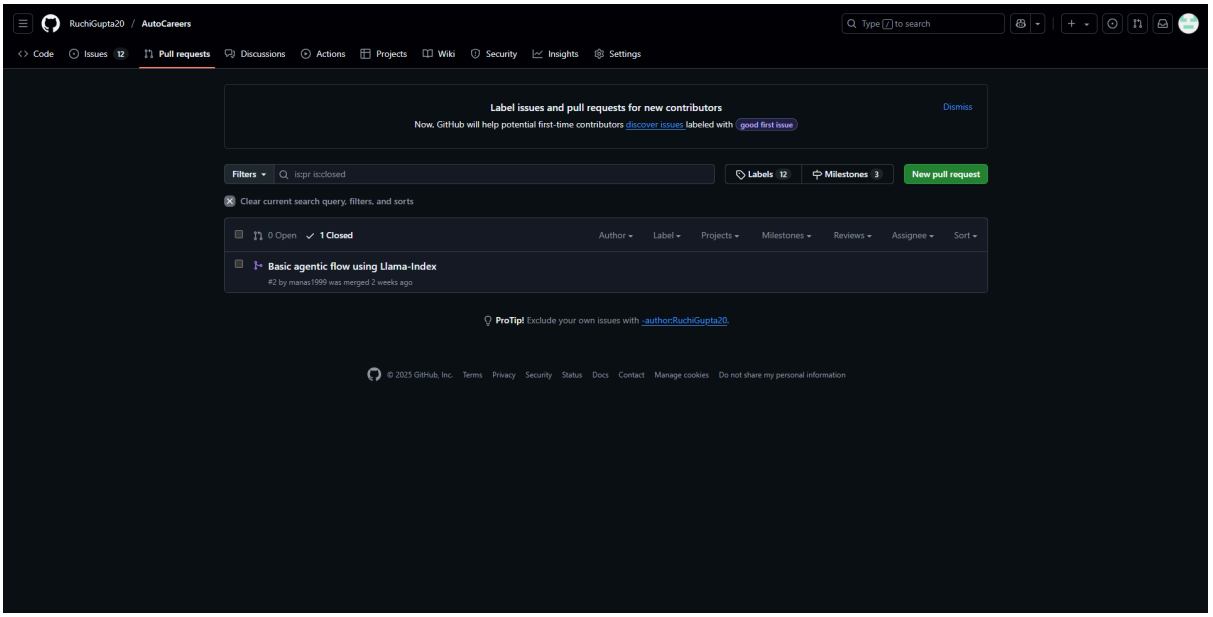
For larger-scale deployment, our future strategy involves pushing immutable images to a container registry and deploying them to platforms like Kubernetes or Amazon ECS. A GitHub Actions pipeline automates builds, runs tests, and triggers rolling updates with health checks to minimize downtime and keep services available. This approach supports scalability, reproducibility, and fast iteration in both staging and production.

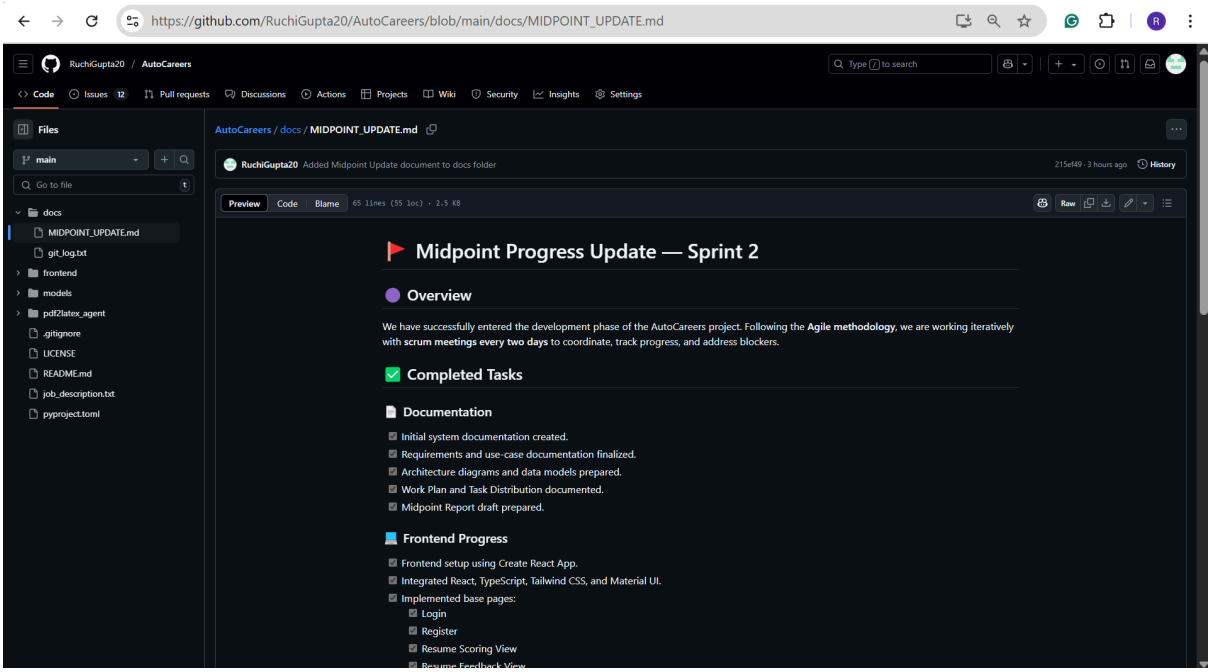
Additionally, we have included screenshots of the GitHub repository page, showcasing the commit history, active branches, etc. as part of this submission.




https://github.com/RuchiGupta20/AutoCareers/issues
RuchiGupta20 / AutoCareers
Code
Issues 12
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
Labels
Milestones
New issue
Open 12
Closed 2
Browserless Scraping in Selenium, and scrape more data from Linkedin
Resume Optimization Suggestions
Cover Letter Generator
Resume ATS Scoring with LLM
Modularize Resume Processing
Frontend Dev
Authentication flow
UI/UX design of Autocareer
Job recommendation engine
AI Agent to extract key words from the parsed resume
Scraping Scripts to extract the job posting from Internet(github repo we are using)


https://github.com/RuchiGupta20/AutoCareers/branches
RuchiGupta20 / AutoCareers
Code
Issues 12
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
Branches
New branch
Overview
Yours
Active
Stale
All
Search branches...
Default
Your branches
Active branches
main
ruchi-dev
priya-dev
manan-dev


RuchiGupta20 / AutoCareers
Code
Issues 12
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
Label issues and pull requests for new contributors
Now, GitHub will help potential first-time contributors discover issues labeled with good first issue
Dismiss
Filters
is:pr is:closed
Labels 12
Milestones 3
New pull request
Clear current search query, filters, and sorts
0 Open
1 Closed
Author
Label
Projects
Milestones
Reviews
Assignee
Sort
Basic agentic flow using Llama-Index
#2 by manas1999 was merged 2 weeks ago
ProTip! Exclude your own issues with -author:RuchiGupta20.
© 2025 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact
Manage cookies
Do not share my personal information


https://github.com/RuchiGupta20/AutoCareers/blob/main/docs/MIDPOINT_UPDATE.md
RuchiGupta20 / AutoCareers
Type / to search
Code
Issues 12
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
Files
main
Go to file
docs
MIDPOINT_UPDATE.md
git_log.txt
frontend
models
pdf2latex_agent
.gitignore
LICENSE
README.md
job_description.txt
pyproject.toml
AutoCareers / docs / MIDPOINT_UPDATE.md
RuchiGupta20 Added Midpoint Update document to docs folder
215ef49 · 3 hours ago
History
Preview
Code
Blame
65 lines (55 loc) · 2.5 KB
Raw
Midpoint Progress Update — Sprint 2
Overview
We have successfully entered the development phase of the AutoCareers project. Following the Agile methodology, we are working iteratively with scrum meetings every two days to coordinate, track progress, and address blockers.
Completed Tasks
Documentation
Initial system documentation created.
Requirements and use-case documentation finalized.
Architecture diagrams and data models prepared.
Work Plan and Task Distribution documented.
Midpoint Report draft prepared.
Frontend Progress
Frontend setup using Create React App.
Integrated React, TypeScript, Tailwind CSS, and Material UI.
Implemented base pages:
Login
Register
Resume Scoring View

Question 6: Deliverables

Writeup (as PDF): Submitted to Gradescope

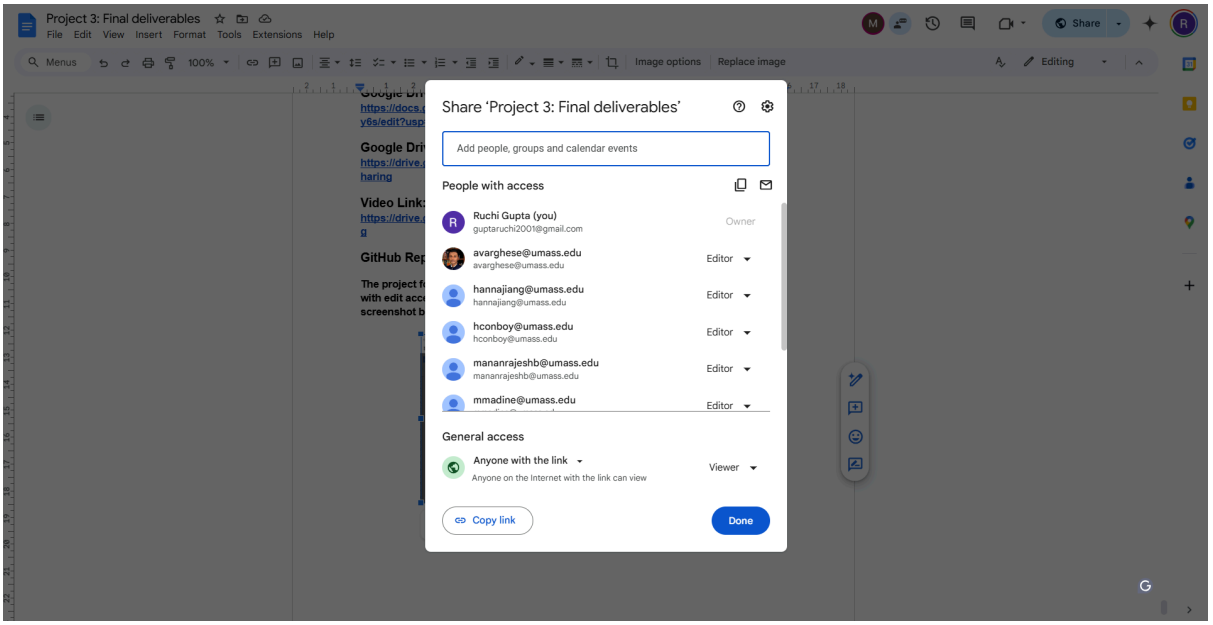
Google Drive Project Document Link (with version control access): https://docs.google.com/document/d/16fk7FtPd1zUE_PAu6K4nnltt321TX8ckowcbbi9xy6s/edit?usp=sharing

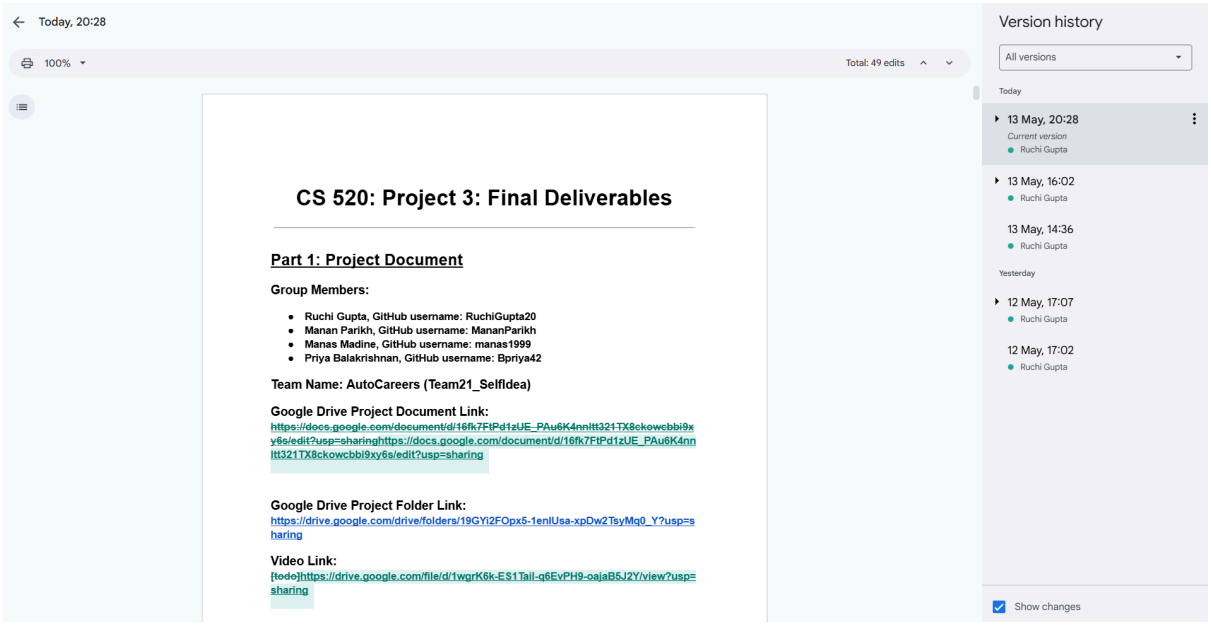
Google Drive Project Folder Link: https://drive.google.com/drive/folders/19GYi2FOpx5-1enlUsa-xpDw2TsyMq0_Y?usp=sharing

Video Link: https://drive.google.com/file/d/1wgrK6k-ES1Tail-q6EvPH9-oajaB5J2Y/view?usp=sharing

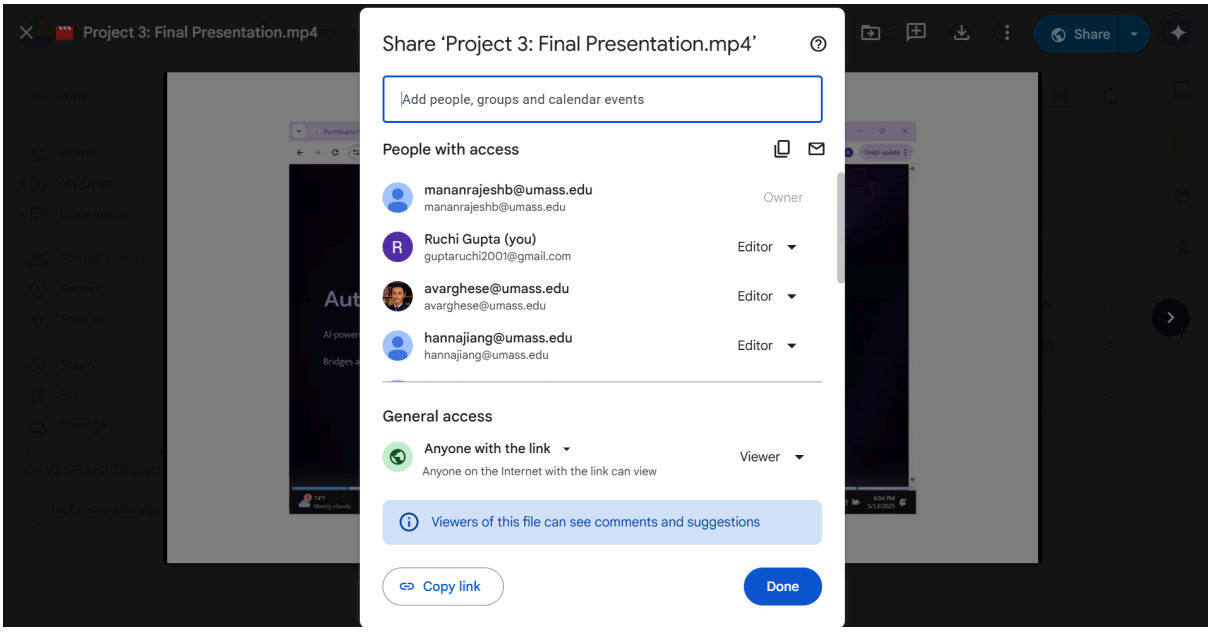
GitHub Repo Link: https://github.com/RuchiGupta20/AutoCareers

The project folder and document have been shared with the professor and all TAs, with edit access granted to everyone (version control is visible), as shown in the screenshot below.

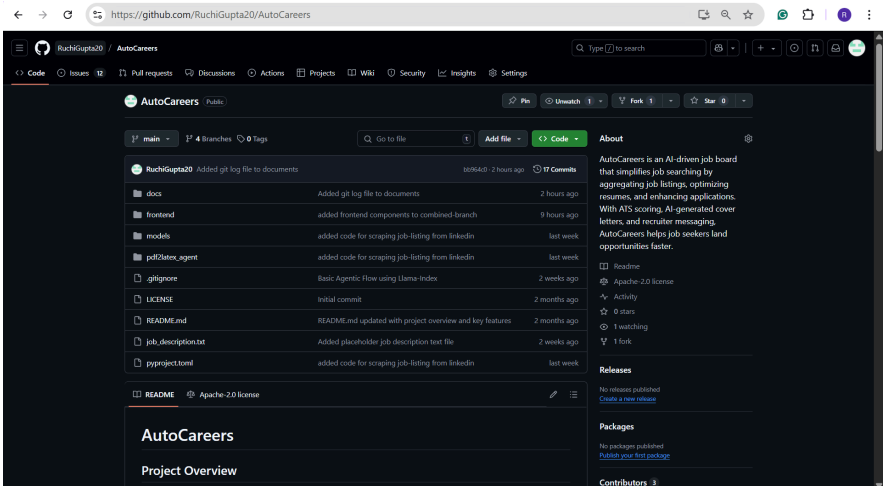




The video presentation file is in the Project 3 Folder and has been shared with the professor and all TAs, with edit access granted to everyone, as shown in the screenshot below.



The github folder is public and hence the version control history and all folders/code is accessible to the professor and all TAs as shown in the screenshot below.


https://github.com/RuchiGupta20/AutoCareers
RuchiGupta20 / AutoCareers
Type / to search
Code
Issues 12
Pull requests
Discussions
Actions
Projects
Wiki
Security
Insights
Settings
AutoCareers Public
Pin
Unwatch 1
Fork 1
Star 0
main
4 Branches
0 Tags
Go to file
t
Add file
Code
RuchiGupta20 Added git log file to documents
b69640 · 2 hours ago
17 Commits
docs
Added git log file to documents
2 hours ago
frontend
added frontend components to combined-branch
9 hours ago
models
added code for scraping job-listing from linkedin
last week
pdf2latex_agent
added code for scraping job-listing from linkedin
last week
.gitignore
Basic Agentic Flow using Llama-Index
2 weeks ago
LICENSE
Initial commit
2 months ago
README.md
README.md updated with project overview and key features
2 months ago
job_description.txt
Added placeholder job description text file
2 weeks ago
pyproject.toml
added code for scraping job-listing from linkedin
last week
README
Apache-2.0 license
AutoCareers
Project Overview
About
AutoCareers is an AI-driven job board that simplifies job searching by aggregating job listings, optimizing resumes, and enhancing applications. With ATS scoring, AI-generated cover letters, and recruiter messaging, AutoCareers helps job seekers land opportunities faster.
Readme
Apache-2.0 license
Activity
0 stars
1 watching
1 fork
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Contributors 3