JOB FORGE

Eduardo Benjamin
Jaden Castle
Jackson Giemza

INFO 4700 Capstone
University of Colorado Boulder
May 1st, 2025

Abstract

JobForge is a web-based platform designed to streamline and simplify the job search process by combining resume enhancement, real-time job matching, and application tracking into one user-friendly tool. The project was developed with both technical functionality and user experience in mind, integrating features such as resume upload and parsing, AI-powered job scoring, personalized resume suggestions, and a centralized dashboard for tracking saved jobs. Users can search for jobs via the Indeed API, receive a resume score based on how well their resume aligns with a job description, and view personalized suggestions to improve their chances of passing applicant tracking systems (ATS). The front-end interface was carefully designed for clarity and ease of use, drawing inspiration from platforms like Grammarly to ensure a clean, intuitive layout. Core backend functionality was supported by a SQLite database that securely handled user data, resumes, and job listings. By merging natural language processing techniques with real-time feedback, JobForge empowers job seekers to tailor their resumes efficiently and stay organized throughout their application journey.

Introduction

The modern job search has evolved into a time-consuming, high-stakes process that overwhelms millions of candidates worldwide. While platforms like LinkedIn, Indeed, and company-specific portals have made it easier to access job listings, they've also introduced new hurdles. Today's job seekers must juggle multiple platforms, tailor resumes for each position, and contend with impersonal applicant tracking systems (ATS) that often eliminate qualified applicants before a human ever sees their resume.

This fragmentation has exposed a major gap in the job search ecosystem: the lack of an integrated tool that not only helps candidates discover jobs but also guides them through

optimizing their applications in one place. A 2022 Jobvite survey revealed that 74% of job seekers find the application process frustrating due to repetitive resume edits, lack of employer feedback, and confusion over ATS rejections. Meanwhile, over 90% of large employers rely on ATS filters that reject resumes for reasons as simple as formatting or missing keywords. The result is a job market that is more automated than ever, but not more accessible.

And the stakes are only getting higher. With an average of 250 applications submitted per job listing with only 4 to 6 candidates advancing to interviews, qualified applicants are frequently overlooked because their resumes don't align perfectly with the job description. Without better tools, many job seekers are left guessing how to tailor their resumes, track their applications, and improve their odds.

This challenge presents a clear opportunity. Our project, JobForge, is a web-based platform designed to unify the job-seeking experience by combining resume tailoring, real-time job search, and application tracking in a single system. Users can upload resumes in various formats, search jobs via the Indeed API, and receive AI-driven feedback on how to improve alignment between their resume and job descriptions. With features like resume parsing, match scoring, keyword insights, and a dashboard for tracking saved jobs, JobForge addresses the major pain points of today's hiring process.

As job hunting becomes more digitized and competitive, the need for smarter, user-focused tools has never been more urgent. JobForge empowers applicants to apply strategically, not just frequently, giving them a better shot at breaking through the noise and landing interviews.

Positionality

As undergraduate information science students, we approached this project from the perspective of students actively navigating the job market ourselves. Our shared experiences as

early-career job seekers have directly shaped our interest in the resume optimization space. We are not industry hiring professionals, but rather applicants who have firsthand experience with the challenges of standing out in the increasingly automated and tiring job market.

Our proximity to the subject gives us both insight and bias, as we are designing a tool for ourselves and people like us. Our research and development is informed by our own frustrations with lack of tools to automate the job search process and the impersonal hiring technologies that expect applicants to put in ten times the work as the recruiter. These experiences have influenced our desire to design more transparent, accessible, and effective resume optimization solutions. While we bring technical knowledge from our coursework in machine learning, data science, and user experience and design, we also recognize that our understanding is shaped by our position as students in a U.S. based academic setting. We acknowledge that job seeking experiences may differ significantly across industries, educational backgrounds, and cultural contexts, and our tool reflects the biases of our own perspectives as U.S. undergrad job seeks in the tech industry.

Background

In the evolving world of digital hiring, many platforms have emerged to help job seekers build resumes to pass Application Tracking Systems (ATS). These tools aim to bridge the gap between job descriptions and resumes through keyword alignment, formatting, and clarity. However, existing platforms often fall short in several critical areas.

Services like Resumeworded and Zety provide tools that claim to increase a resume's chance to pass through ATS filters. While useful, these systems often rely on static scoring metrics that do not fully consider the context or semantics of the job description. These platforms provide a lack of dynamic suggestions that adapt based on the nuances of both the job description and the applicant's resume. Instead, feedback is often generic, such as "add more

hard skills" or "use action verbs, " rather than offering actionable, tailored improvements or skills an applicant can acquire.

Objectives

Our overall goal is to take JobForge from concept, through development, and into a fully functional platform that streamlines the job search process. To achieve this, we will focus on building a robust system that helps job seekers stay organized and ahead of their peers.

One of JobForge's core features is resume optimization, designed to help job seekers pass applicant tracking systems (ATS) and get noticed by recruiters. Many companies use ATS software to filter resumes, often rejecting poorly formatted ones before they reach a recruiter. We plan on integrating an ATS filtering algorithm to test resumes against industry standards, providing LLM-powered feedback on structure, formatting, and content. Users receive actionable suggestions, such as improving keyword density, adjusting headings, and reformatting text to enhance ATS compatibility.

Beyond ATS optimization, JobForge will evaluate how well a resume aligns with specific job descriptions. Using natural language processing (NLP) techniques like `nltk` and Rapid Automatic Keyword Extraction (RAKE), it will extract key terms from both the resume and job listing to generate a match score. Users receive insights on missing skills, experiences, and keywords, along with tailored recommendations for projects, courses, and skills to improve their job prospects. This ensures resumes are optimized for each application, increasing the chances of passing automated filters and securing interviews.

We plan on creating a job search dashboard to simplify the job-hunting process by providing users with real-time job postings tailored to their preferred industry, role, and location. By integrating with job aggregation APIs, our platform will continuously update users with the latest opportunities, ensuring they never miss a relevant posting.

Methodology

Beginning our project we knew that we set out a large goal and if we wanted to have a complete project it we would be on a tight schedule. Therefore we knew we had to have a thorough understanding of every aspect of what we wanted to create and a timeline of how we would get there. This meant that our first two weeks were devoted strictly to planning our project, covering every potential feature we would include and what features we might scrap. This allowed us to create a realistic timeline that still had a bit of wiggle room for disruptions or other unforeseen events. This plan also allowed us to create roles for ourselves based on our strengths and weaknesses, these roles not only allowed us to work in areas we'd like to pursue career wise, but also meant we were able to stick to our timeline.

We had three main phases of development, the first phase we set out to begin our web application to gauge what it might look like, as well as obtain a resource for real up to date job postings. To do this we split up with Jaden working on the frontend, Eddie working on creating the integrated backend and Jackson researching an API for jobs.

As far as creating the backend, I knew that I wanted to use Flask going into it so I started by creating the sqlalchemy database that would hold the users. We created a csv with seed data so my initial goal was to get that seed data into the database and from the database displayed to a Flask endpoint. Creating the database started with creating a model for the users which would dictate the allowed columns the users table had. I then created static functions to retrieve all the users and a specific user based on their ID from the database. Creating Flask endpoints was as simple as calling these functions. I also used `CORS` and `uvicorn` to set up a specific port to display the backend and allow for only the frontend to interact with it.

The development process began with the creation of the Users page, which helped establish the foundational layout and aesthetic direction of the app. This initial stage was

essential for shaping the user experience and setting the visual tone for future components. A key part of the approach involved drawing inspiration from platforms with clean and intuitive design—particularly Grammarly. Its sidebar and navigation structure served as a major influence, guiding decisions around layout, spacing, and interface clarity. By focusing early on usability and visual consistency, the team was able to build a flexible front-end framework that could scale alongside the app's growing functionality. The Users page acted as a starting point not just for technical implementation, but also for imagining what JobForge could become both visually and interactively.

To gather job posting data at scale, Jackson evaluated several job platforms including LinkedIn, Glassdoor, and Indeed. The goal was to find a reliable source of aggregated job listings that would allow us quick access to current job listings and their descriptions. After testing access limitations and usage policies, we determined that Indeed offered the most practical option for our needs. Unlike LinkedIn, which has strict anti-scraping measures and limited public API access, Indeed had more lenient restrictions and consistently returned structured HTML content suitable for parsing. By leveraging this data, we ensured that our platform was grounded in current, real-world job postings.

After we all came together we then took some time to integrate all the components we created to each other. With the frontend and backend it was as simple as plugging in the ports each component would be talking. Incorporating the Indeed API would then become something we focused on in the next phase. With this marking week 3-4 we had the users completely functioning which didn't have much to do with JobForge's goals but gave us lots of confidence moving forward.

We were able to create the last few components with relative ease so the next phase of our project would include way more. We all were able to gauge that this project would be achievable and mostly just relied on our ability to code a project this large. From here we set out to complete the majority of our project including, the ability to search jobs, upload resumes, save jobs, get a resume score, and view saved jobs. We would also meet every week to incorporate the components we created separately ensuring that we didn't allow those problems to stack up.

Regarding the backend incorporating most of these abilities was the same workflow as before: create database, create functions to manipulate/retrieve data from the database, use those functions in Flask endpoints. I did have to do some research regarding how to do some things like retrieve and serve a resume and the different ways of doing so, also the job search was a bit challenging. The search relied on formatting the location and job title in a way that Indeed would return the proper jobs. This mainly meant turning states into their abbreviations and querying the job titles both split and together. This solved the issue of retrieving jobs but another issue arose when trying to retrieve those jobs after storing them in the database. For example if you search ‘software engineer’ , Indeed might return jobs like ‘System Technician’ which is for all intents and purposes is a ‘software engineer’ job. However with my SQL queries even after making the matching as loose as possible and trying all combinations, you'd never retrieve this job. The solution would eventually be to create an instance of jobs after they are retrieved and then do the database retrieval and combine the two instances discarding duplicates. This meant that all jobs retrieved from Indeed were shown while still having access to older jobs as this Indeed search would be limited to the current day. Other than this meeting every week and getting our progress together was very helpful as it allowed us to catch any bugs as they happened.

Each feature was carefully designed with a focus on simplicity, clarity, and responsiveness to ensure a smooth and efficient experience for job seekers. The job search interface was built to be clean and intuitive, allowing users to easily search for roles based on job title and location. Search results were displayed in a card-based layout, with each job featuring essential details and a clear option to save the listing for later. The design emphasized readability and quick navigation, helping users browse multiple postings without feeling overwhelmed.

For the resume upload feature, the front end supported multiple file formats (PDF, DOCX, TXT) and included simple prompts and feedback to guide users through the process. Once uploaded, the resume was parsed in the backend and prepared for analysis, with the front end clearly indicating confirmation of success.

The Dashboard provided users with a centralized space to revisit and manage their previously saved job opportunities. Presented in a clean, scrollable list format, each entry displayed key information such as the job title, company name, and the personalized resume score.

To enable live job search functionality, we integrated an open-source web scraper tailored to Indeed's structure. This allowed the application to retrieve real-time job postings without requiring access to a proprietary API. This ensured that users could access up-to-date job opportunities from Indeed.

For resume upload, we built backend functionality that could extract raw text from various resume formats, particularly PDFs. This process included handling formatting inconsistencies and stripping non-textual elements to ensure clean, usable input for downstream processing. These parsed resumes were then stored in a vectorized format for efficient access and scoring.

Using the sentence-transformers library, we embed both the uploaded resume and the selected job description into high-dimensional vector representations. We then calculated a cosine similarity score between the two embeddings to evaluate how well the resume aligned with the job. This raw similarity score was mapped to a generalized 1–10 scale to make it more digestible and user-friendly. The resulting score was presented alongside actionable feedback, guiding users toward improving their resume's alignment with each job posting.

At the end of this phase our project was pretty much finished with the frontend and backend fully functional and communicating, as well as utilizing the software we created. We did however still have the suggestions to create which was a large highlight of our project but would require the most work on all fronts (except backend).

The last section of our project would be incorporating the suggestions of both general resume improvements and suggestions based on jobs added to your dashboard. In the beginning we wanted to allow users to directly edit their resume on our application but we quickly realized that was out of the realm of this project. We instead shifted to just displaying these suggestions allowing the users to make the improvements in the software they created on their resume.

The backend was business as usual, this time I just had to add some columns to the resume database as storing them here would mean that the time intensive process of retrieving them would only have to happen once. During this phase I also took a lot of time to clean up the time intensity of our project as it was getting to the point where it would affect users. I found the best way to do this was to vectorize the users resume upon retrieval and get general suggestions at this time. This meant that the only thing that would take any real amount of time was this upload. The vectorized user resume was used all throughout the software and it was currently happening every time which takes a lot of time as it uses pyvision.

Through the dashboard, users had direct access to tailored suggestions through the resume editor tab, which opened a detailed view combining the resume viewer, general resume improvement tips, and job-specific recommendations. This cohesive design made it easy for users to assess how well their resume aligned with each role and encouraged them to make targeted updates to improve their chances of standing out. The intuitive layout ensured that users could navigate their job applications efficiently and take meaningful steps toward optimization.

We integrated the OpenAI API and engineered prompts that combined resume best practices with contextual resume content. The prompts were informed by formatting and content guidelines covered in the Portfolio and Professional Development course, ensuring that the suggestions were grounded in real-world standards for resume writing.

For general suggestions, the system analyzed the uploaded resume and identified areas for improvement such as clarity, formatting, skill phrasing, and section completeness. For job-specific suggestions, the uploaded resume was compared to the job description, and targeted advice was generated to highlight missing skills, rewording opportunities, and ways to better tailor the content. This dual-layered feedback system helped users understand not just how to improve their resume broadly, but also how to adapt it to specific roles—making their applications more relevant and competitive.

Considerations and Limitations

We created a pretty thorough plan from the beginning so there weren't too many roadblocks we ran into. We did however have a few limitations in the ability to directly implement suggestions and the job search still not being as polished as a site like Indeed. These mainly came up due to time constraints but honestly at this time we aren't aware of any solution to these problems if we were to try to incorporate them.

We also did not spend too much time getting feedback or doing research to support our ideas. We mainly focused on creating the application then seeing how people felt about it afterwards. This was likely due to the fact that the general sentiment toward applying to jobs right now is negative. Things like ATS have been complained about at least in circles that we are in so we felt the need was there for an application like this.

Team Contributions

Jaden - Frontend

Eddie - Backend & Assistance

Jackson - Systems

Deliverables

We created two deliverables to bring to the showcase, a poster displaying supporting evidence of why users need tools to assist with the job search and our web application. We were able to fully deploy the web application at the time of the showcase. This allowed users to utilize the app from any device and utilize it as they would under any circumstances.

The website contains the following components:

- Frontend
    - Job dashboard: collection of potential jobs that match your qualifications, including things such as a score showing your fit. Tracking of existing applications.
    - Resume Enhancer: UI showing suggested changes and the ability to reformat resumes to better formats.
        - Real Time Scoring Feedback
    - Resume Upload
    - User Logins
- Backend
    - **Sqlite Database**
        - **Users**: This parent database stores the information of the users including username, password. City, and desired job title for use in our application. Storing this information also allows users to utilize JobForge seamlessly across devices.

        - **Resumes**: Child database within users, holds resume information and storage location, also holds vectorized resume text.
        - **SavedJobs:** This holds information regarding saved jobs of the users. This includes the id, score and time of saving.
    - **Jobs:** This stores the jobs searched by users, this database is cleaned daily to both ensure up to date jobs and prevent an overload of jobs.
- **Endpoints**
    - **Users**
        - get_users(): returns all users
        - get_users(user_id): returns specific user based on their id
        - register_user(): takes all necessary information and registers user
        - get_saved_jobs(user_id): takes user id and returns their saved jobs
        - remove_saved_job(job_id, user_id): takes user id and job id and removes specified job.
        - save_job(user_id, job_id): takes user id and job id and saves specified job.
    - **Resumes**
        - upload_resume(): takes a resume and stores it
        - resume_score(user_id, job_id): takes user id and job id and returns users score for that job.
        - view_resume(user_id): takes user id and returns their resume
        - general_suggestions(user_id): takes users id and returns some general suggestions to improve their resume.
        - job_specific_suggestions(job_id, user_id): takes the job id and users id and returns job specific recommendations to improve their resume.
    - **Jobs**
        - `get_jobs()`: returns all jobs
        - `job_search(job_title, location)`: takes title and location of jobs and returns matching postings.
        - `get_job_by_id(job_id)`: returns a specific job based on id.
- **Systems**
    - **Job Score**
        - *'Score'*: This class contains the core function to generate the similarity score.
            - 'clean_resume()': extracts experience and skills sections and cleans the text to remove stop words and special characters.
            - 'compute_similarity()': Takes cleaned resume and job description text and embeds them using the

'all-MiniLM-L6-v2' sentence transformer, and uses the BERT semantic textual similarity algorithm to retrieve a percent similarity score.
    - 'similarity_to_score()': Takes the percentage similarity score and converts it into a 1 to 10 scale to be human readable.
    - 'get_score()': Given user ID and job ID pulls resume and job description from the database and computes and cleans the score and returns json.
- **Resume Suggestions**
    - 'general_suggestions()': Given a resume the OpenAI API is prompted with resume evaluation principles and the task of returning actionable suggestions in json format'
    - 'job_based_suggestions()': Given a resume and job description OpenAI API does the same but for job specific suggestions.
- **Resume Scraping**
    - Using pytesseract to extract text from resume pdfs, with pdf2image as a backup which uses computer vision to extract text. The text is rigorously cleaned to preserve bullet points and section formatting.
- **Job Scraping**
    - Using an open source web scraper JobSpy to pull the most recent Indeed data.

The user flow begins with registration or login. After authentication, the user lands on a home page where they are prompted to upload their resume. Once uploaded, the system parses and stores the resume while preparing it for scoring later. The user can then search for jobs by title and location. Relevant postings are displayed in a scrollable list, and each job includes the option to view details or save it to the dashboard. Upon selecting a job, the platform immediately evaluates the resume and returns a match score using natural language processing techniques. From there, the user can go see both general and job-specific suggestions to improve their resume and increase their score.

Conclusions and Future Work

Over the course of this project, we developed a fully functional web application that tackles key pain points in the modern job search process. Through JobForge, we learned how to integrate natural language processing techniques with practical frontend and backend design, enabling users to upload resumes, receive tailored feedback, search and save job listings, and track their application progress, all in one place. Our work demonstrates that resume optimization can be made more transparent and user-friendly, particularly when job seekers are empowered with tools that provide real-time insights and actionable suggestions.

Given the scope of our timeline, we focused on building a stable and intuitive experience. However, there is significant room for future development. One major improvement would be enabling in-app resume editing, allowing users to modify and re-upload resumes directly based on our suggestions without leaving the platform. Another area for expansion is improving the intelligence of our scoring and recommendation systems, diving deep past similarity and into the weeds of formatting, grammar, and content scoring. We would also like to expand our job sourcing beyond Indeed, ideally incorporating more APIs or partnerships with job boards to diversify listings and improve the relevance of job matches.

Additionally, incorporating user research is a priority for future iterations. While our current implementation is shaped by our own experiences and assumptions as job seekers, broader usability testing and engagement with users from different industries and backgrounds would help us refine the product and ensure it serves a more diverse audience. Questions remain around how users interpret match scores, how they prioritize feedback, and whether real-time suggestions significantly impact job application outcomes, questions we hope to explore in future versions of JobForge.