CareerMatch-AI

Sebastian Grammas, Will Gaca, Adityaa Suratkal, Sebastian Landeta, Adrian Corujo, Pablo Semidey, Ryo Yoshida

Introduction

CareerMatch AI is a work-in-progress intelligent job-matching platform that merges frontend interactivity, backend automation, and a deep learning-based recommender system to help users discover career opportunities tailored to their skills, interests, and experiences.

Completed Tasks This Semester

- Wrote webscraping script that can be scaled on the zensearch job board for any company
- Fully automated using github actions to update job postings data every 48 hours
- Created initial website where people will be able to upload data, create profiles and upload resumes
- Began setting up flask app to connect back end to front end code
- Created working front end to back end connection to collect data on resumes and scrape them
- Initial implementation of user registration and login with homomorphic encryption of user profiles using SQLite3
- Automated welcome emails upon signup
- Designed job data schema for consistent output across all scrapers
- Integrated a pdf preview on front end to view the file before submitting to our back end
- Designed and built a PyTorch-based Transformer encoder recommendation model with collaborative filtering decoder
- Collected and cleaned dataset of job descriptions and resumes for training
- Trained the recommender model over large datasets using multi-day compute power.
- Implemented functions for model preference feedback (e.g., accept, reject) to personalize job matching.
- Created modular structure for scaling backend logic and scraper extensions.
- Initiated Flask-based integration of user model preferences to serve dynamic recommendations.

Frontend (User Interface)

Responsive web interface built with HTML/CSS/JavaScript.- Key Pages: - index.html: Interactive landing page showcasing platform features. - about_us.html: Team introduction and project overview. - personal_info.html: Resume and personal data collection with real-time preview. - login.html and signup.html: Secure user authentication with validation.- Smooth, modern design with parallax sections, animated overlays, and intuitive navigation.- Implements fetch-based communication with the backend for user flows and data submission.

Backend (Infrastructure & Data Handling)

Flask-powered backend connecting user-facing frontend with internal logic.- Key Features: - Rudimentary but functional user registration and login. - Homomorphic encryption of user profiles using SQLite3. - Automated welcome emails upon signup. - Resume scraping and parsing via resume_scraper.py for structured data storage and matching.

Job Data Scraping Engine (Zensearch)

----

zensearch.js: JavaScript-based scraper using node-fetch to retrieve job postings from companies listed in zensearchData.csv.- Automatically scrapes every 48 hours and outputs structured CSV files per company.- Captures: - Title, Pay, Location - Description, Remote Status - Job Type (Full/Part/Internship) - Experience Level, Date Posted, Link- Scalable to international job markets.

Recommender System (RecsysFiles)

Developed in PyTorch using Transformer encoder models.- Architecture: - Dual encoders process user resumes and job descriptions separately. - A decoder outputs similarity scores for matching. - Inspired by Meta's approach to content recommendations.- Workflow: - Cleaned and paired large-scale job + resume datasets for training. - Collaborative filtering-based decoder enables fast inference. - Final model trained over 1–2 days of compute time.- Backend integration: - Real-time preference tracking via model-aware functions (accept, reject, etc.). - Dynamically updates user-job match scores to personalize suggestions.

Recommender System

- [ ] Integrate PyTorch recommendation model fully into backend Flask routes.
- [ ] Create API endpoints to serve personalized job recommendations to users.
- [ ] Add support for live model updates based on new user input and behavior (e.g., likes, skips).
- [ ] Deploy model to cloud server or containerized environment (e.g., Docker + AWS/GCP).
- [ ] Evaluate and fine-tune model performance using real user data and feedback loops.

Frontend Enhancements

- [ ] Build a personalized dashboard to display job matches and profile info.
- [ ] Implement a swipe-like interface (Tinder-style or card carousel) for job recommendations.
- [ ] Add UI for viewing job postings scraped from Zensearch.
- [ ] Provide visual feedback when resume parsing is complete.
- [ ] Improve error messaging and form handling for all user inputs.

Authentication & User Management

- [ ] Upgrade login/signup to use JWT tokens for session management.
- [ ] Add "Forgot Password" functionality.
- [ ] Build user profile editing capabilities (name, resume re-upload, preferences, etc.).
- [ ] Implement role-based access if needed (e.g., admin dashboard for job data curation).

Job Scraping and Zensearch Expansion

- [ ] Add browser-based scraping (e.g., Puppeteer) for JavaScript-heavy company job boards.
- [ ] Include company logos, job tags, and skill requirements in scraped data.
- [ ] Improve error handling in zensearch.js for failed or throttled requests.
- [ ] Enable on-demand scraping for user-requested companies.

Database & Storage

- [ ] Migrate from SQLite3 to PostgreSQL or another scalable RDBMS.
- [ ] Add ORM integration (e.g., SQLAlchemy) for more maintainable backend code.
- [ ] Set up object storage (e.g., S3 or local equivalent) for resume file storage.

Communication & Email

- [ ] Create a user settings page for managing email preferences.
- [ ] Set up templated email notifications for new matching job postings.
- [ ] Track email open/click rates (optional analytics layer).

Testing & DevOps

- [ ] Add automated tests for frontend/backend components.
- [ ] Implement CI/CD pipeline for deploying frontend + backend.
- [ ] Containerize the app using Docker and provide dev/staging environments.
- [ ] Monitor uptime and scraper performance using logging and alerting.

Analytics & Feedback

- [ ] Track user engagement metrics (job views, clicks, signups).
- [ ] Add feedback option to report bad recommendations or scraped data issues.
- [ ] Build internal admin panel for viewing usage stats and managing job sources.

RCOS
Rensselaer Center for Open Source