CS 490 Proposal: Talent Search Portal for Quantitative Recruiters

Goal:

To learn about modern web technologies, apply academic CS principles, and practice writing maintainable code by building a modern web application. Specifically, I will build a website that allows recruiters at quantitative trading firms to view detailed data about talented, technical university students. This data includes the students’ scores on quantitative exams that assess their skills in quantitative problem solving, coding, data science, and mental math. (Note: only the technical platform, and not the collection of student data, is part of this senior project.)

Background:

Right now, LinkedIn is one of the best online resource for recruiters at quantitative trading firms to get information about potential candidates (particularly software engineers and quantitative traders). But LinkedIn is not perfect — a student’s LinkedIn profile is a noisy signal in predicting their technical abilities. Company recruiting would be much more efficient and less costly for firms if recruiters and hiring managers could directly tell from a student’s profile their technical levels on the skills that directly pertain to the job at hand.

To this end, I will build a website that only displays information about students who have passed a certain technical bar for hiring. For the ~4% of students that pass the technical bar, the website would include detailed information on those students’ technical strengths and weaknesses, as
assessed by the company administering the application. In this project, I will build the technological components of such a platform.

**Feature Specs:**

The core features of the portal are:

- Ability to search the database by name, school, and technical exam results
- A dashboard for recruiters to view candidates they have already invited to interviews.
- Multiple recruiters from the same firm should be able to create separate accounts.
- SSO authentication that companies can integrate with their existing auth systems
- A profile page for each student, listing detailed information about their experience and technical abilities (based on the exam)
- Recruiters should be able to reach out to students they want to invite for an onsite interview
- Students should be able to log in to the platform and update their information

**Technical stack and challenges:**

**Frontend:**

I will build an easy-to-use, intuitive GUI using React and Typescript. State management with MobX. Apollo GraphQL as a GraphQL client.

**Backend:**

I will write the server using Python Flask. I will containerize the application with Docker. I will also use the Auth0 service for authentication.

**Database:**
I will use PostgresSQL, and I will make queries using the SQLAlchemy object-relational mapper. I will make sure to add appropriate indices to the database to improve query efficiency and to do database normalization. In general, I’ll follow the design principle of moving as much logic as possible into the database (tests, functions), so that multiple applications can query the database while maintaining data integrity.

Unit tests:

As I write the codebase, I will make sure each new feature or behavior of the application is tested with appropriate unit tests. I will use Pytest for testing the server and Jest for the frontend. On the database side, I’ll use pgTap for tests and sqitch for migrations.

The purpose of these tests will be

1. To ensure that new features behave the way I expect them to
2. To prevent new changes from breaking the behavior of existing code

Hosting infrastructure:

The frontend assets and backend application will be served using Google Cloud Platform, each run in Docker containers. I will have separate development, staging, and production environments. I will also use a continuous integration solution (perhaps CircleCI) that runs the test suite on all pull requests before they are merged

Timeline:

- 9/27: Finish design of screens and story boards (drawn on paper)
• 10/7: Set up the environment — virtual environments, React application, state management, PostgresSQL database. Set up BitBucket. Set up continuous integration to run unit tests. Set up hosting, provisioning servers, and DNS configuration on GCP.

• 10/14: Decide on database models. Set up Flask server. Add backend unit tests. Add GraphQL schema and queries

• 10/21: Move backend to Docker.

• 10/28: Finish 1/3 of frontend. (Profile page, dashboard page, search page).

• 11/4: Finish next 1/3 of frontend. Should display all information from the backend.

• 11/11: Finish remainder of frontend

• 11/28: Implement search features — look into ElasticSearch.

• 11/25: Test the application with beta users, make improvements.

• 12/2: Draft the written report. Continue testing the application, improving frontend, fixing bugs.

• 12/9: Final draft of written report.