

| Team Member Full Name | NetID |
|-----------------------|---------|
| Chris Capone | ccapone |
| Mark Rumsey | mrumsy |
| John Daniel Lee | jlee88 |
| | |

Chosen Technology Stack

☒ Python + Django

☐ Python + Tkinter

Persistent Storage Design

We are using SQLite database to persist our data. Our database includes 15 tables, as seen in Figure 1.1. Many of these are mainly related to the infrastructure of the website, but the tables we primarily interacted with are as follows:

- auth_group:
 - Defines the two groups of users, recruiter and candidate
- auth_user:
 - Contains data for the user from account creation page for both groups of users, including which group (candidate/recruiter) they belong to. The user object diagram is included in Figure 1.2.
- candidate_candidateprofile/recruiter_recruiterprofile:
 - Contains the inputted candidate/recruiter information from the profile creation page, for both candidates and recruiters respectively. New from the previous report are uninterested_ids and compatibility score, the former being a string containing a text-encoded array of uninterested post ids. The compatibility score is fairly self-explanatory - the algorithm will be described later. The candidate/recruiter object diagram is included in Figure 1.2.
- recruiter_jobpost/recruiter_offer
 - Essentially a sub-data structure that contains the information of job postings and recruiter offers. The job posts and offers point to the recruiter but are used by both candidates and recruiters. The object diagrams are in Figure 1.2.

| |
|------------------------------|
| Tables (15) |
| > django_migrations |
| > sqlite_sequence |
| > auth_group_permissions |
| > auth_user_groups |
| > auth_user_user_permissions |
| > django_admin_log |
| > django_content_type |
| > auth_permission |
| > auth_group |
| > auth_user |
| > django_session |
| > recruiter_recruiterprofile |
| > recruiter_jobpost |
| > recruiter_offer |
| > candidate_candidateprofile |

Figure 1.1 tables

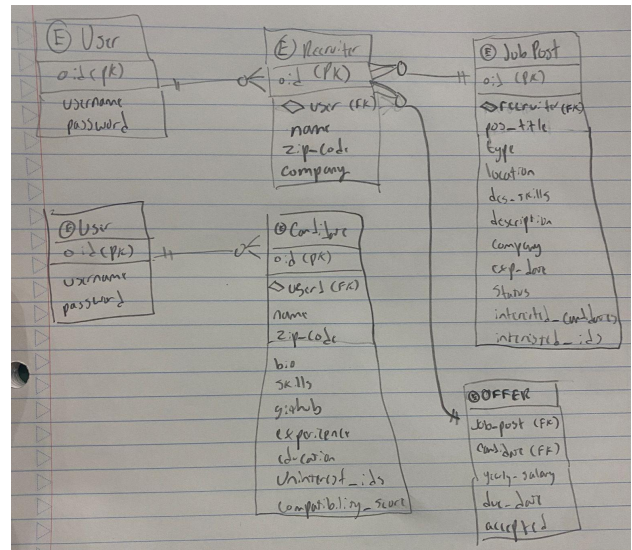


Figure 1.2 database schema

Demonstration of the Features

Feature 1.1

Figure 1.1.1 shows a screenshot of the signup page. Here, a candidate enters a username and password and selects that they are a candidate. When the sign-up button is clicked, the user is redirected to the login page. Figure 1.1.2 shows the profile creation page for a candidate. The first time a candidate logs in, they will be taken to this page to enter the required information.

The screenshot shows a web browser window titled "Candidate Signup" with the URL "127.0.0.1:8000/register/signup/". The form contains the following elements:

- Signup** header
- Username** field: candidate_test
- Password** field: masked with dots
- Re-enter Password** field: masked with dots
- Profile Type** section:
 - ☒ Candidate
 - ☐ Recruiter
- Buttons**: Sign Up (blue), Back to Login (grey)

Figure 1.1.1 screenshot for feature 1.1 showing the account creation for a candidate

Candidate Profile Creation

Name
Chris

Zip Code
12345

Bio
I'm Chris

Skills
Verilog HDL
Android Studio

Github
ccapone1

Experience
15

Education
Notre Dame

Create

Figure 1.1.2 screenshot for feature 1.1 showing the profile setup for a candidate

Feature 1.2

Figure 1.2.1 shows a screenshot for the signup page. This page is shared the same as the one for candidates, but a recruiter will check the recruiter box to indicate the account type. After clicking sign-up, they will be brought to the login page. Figure 1.2.2 shows the profile creation page for recruiters. This will be displayed to the recruiter the first time they log in with the account to fill in the required information.

Signup

Username
recruiter_test

Password

Re-enter Password

Profile Type
☐ Candidate ☒ Recruiter

Sign Up Back to Login

Figure 1.2.1 screenshot for feature 1.2 showing the account creation for a recruiter

Recruiter Profile Creation

127.0.0.1:8000/recruiter/profile_creation/

Setup Recruiter Profile

Name

Mark

Zip Code

12345

Company

Tindey

Create

Figure 1.2.2 screenshot for feature 1.2 showing the profile setup for a recruiter

Feature 1.3

Figure 1.3.1 shows a screenshot for the login page. If a user has already created an account, they can enter their credentials and log in to be taken to either the profile setup page or the homepage (depending on whether or not they have finished setting up their profile). There is also a signup button that will redirect the user to the signup page, where they can create either a candidate or recruiter account.

Login

Username

Username

Password

Password

Login Signup

Figure 1.3.1 screenshot for feature 1.3 showing the login page

Feature 1.4

Figure 1.4.1 shows a screenshot of where the logout button is located on the candidate homepages (identical on recruiter). When clicking logout, it uses Django's built-in logout method to remove the user's middleware and redirect the user back to the login screen.

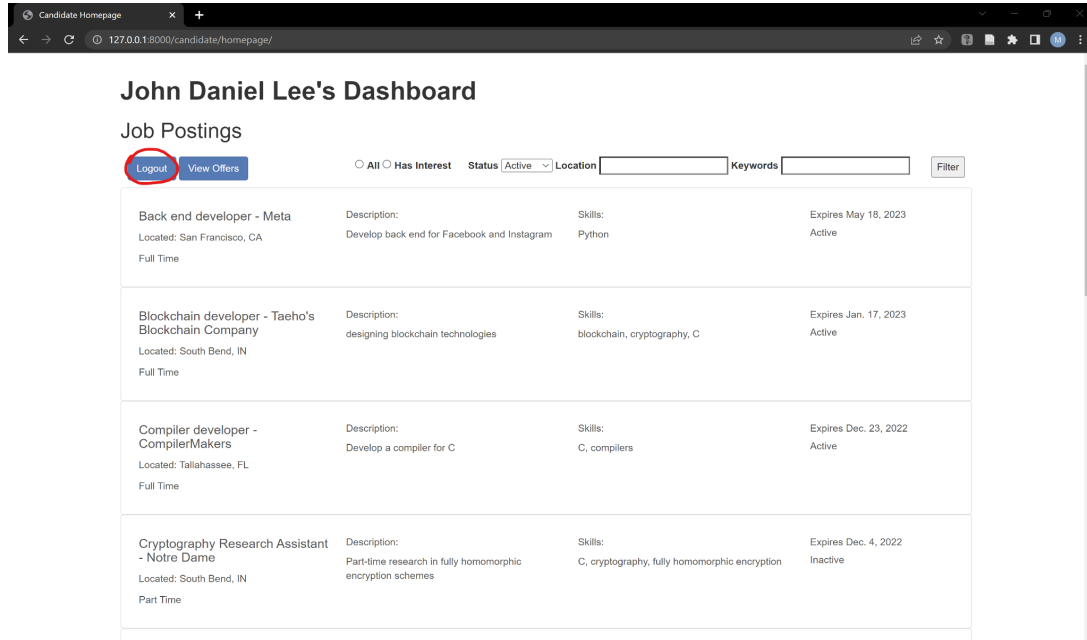


Figure 1.4.1 screenshot for feature 1.4 showing the logout button

Feature 2.1

Figures 2.1.1-4 show a screenshot of the recruiter homepage (each with different filters), which contains a list of all the jobs with necessary summary information given. In the top right of the list, there is a selector for the different filtering operations. For each, select the desired filter and click the filter button and the posts will be filtered as desired. When the recruiter clicks on one of their job posts, it will bring them to the job post detail page, shown in the figure, where they can update or delete the post and make offers to interested candidates.

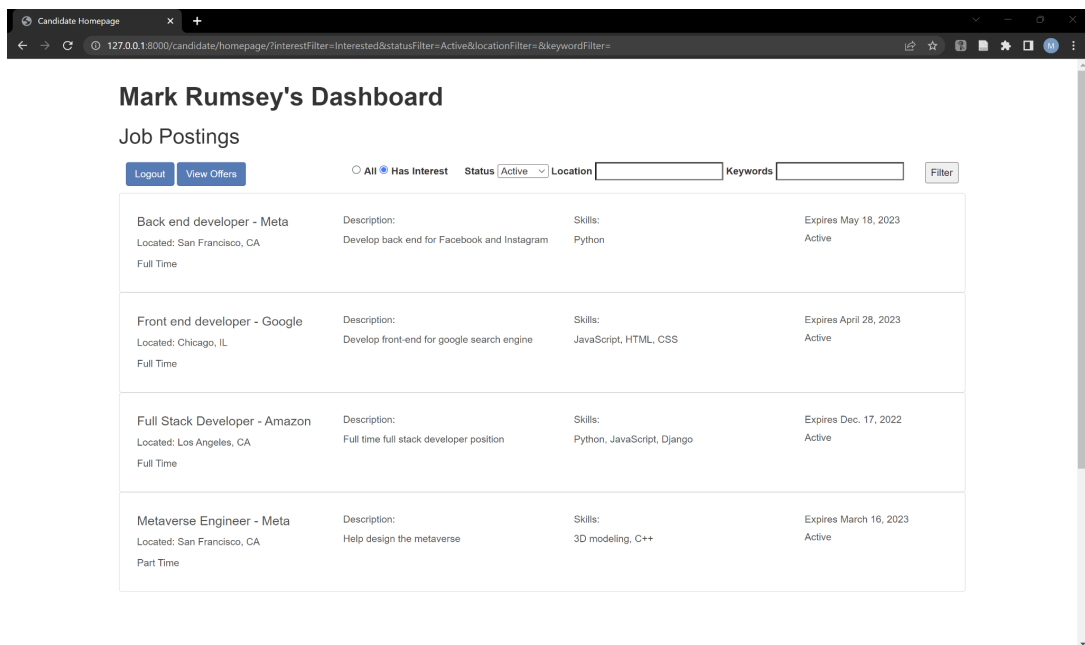


Figure 2.1.1 screenshot for feature 2.1 showing the interest filter

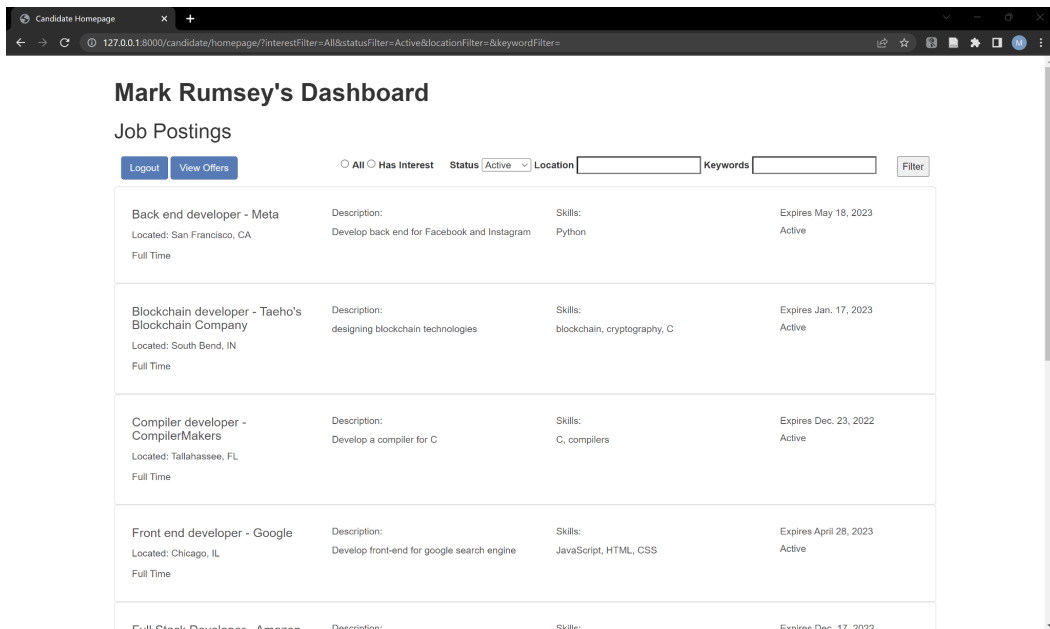


Figure 2.1.2 for feature 2.1 showing the active filter

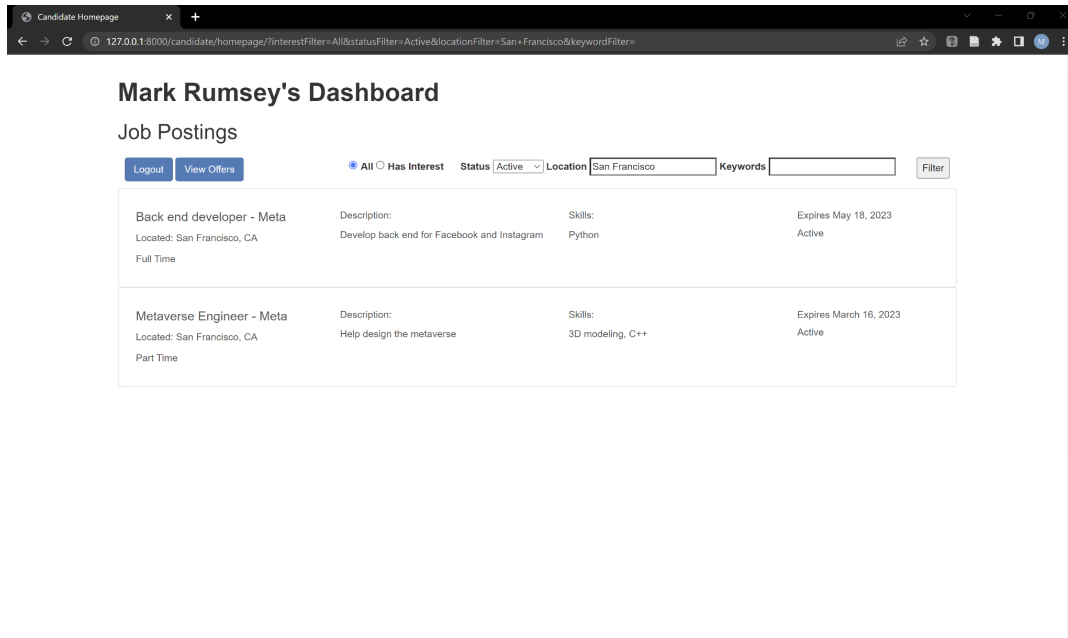


Figure 2.1.3 for feature 2.1 showing the location filter

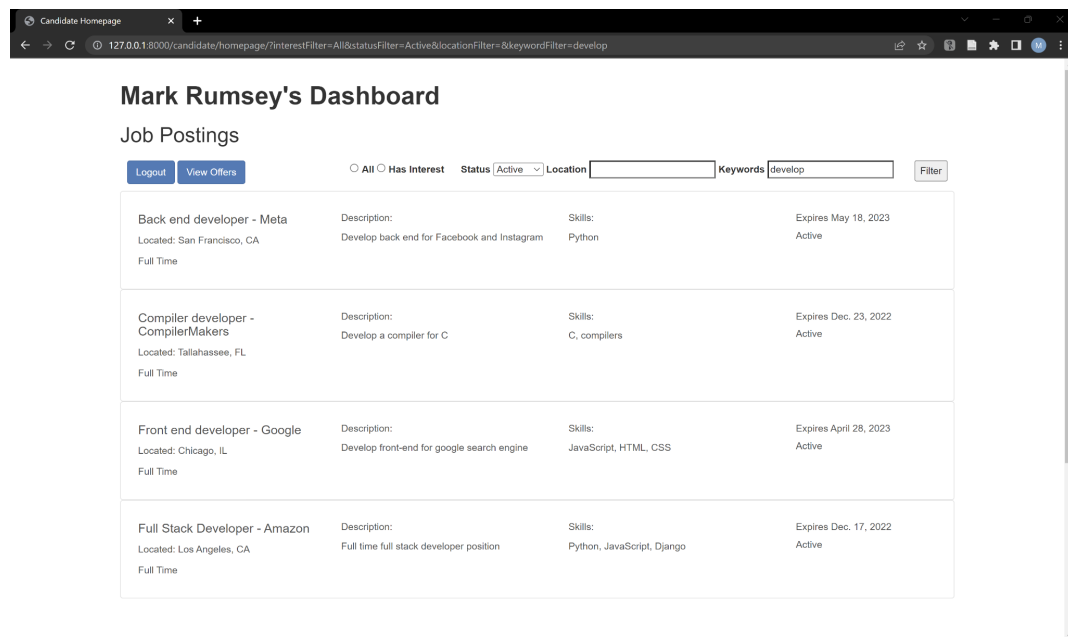
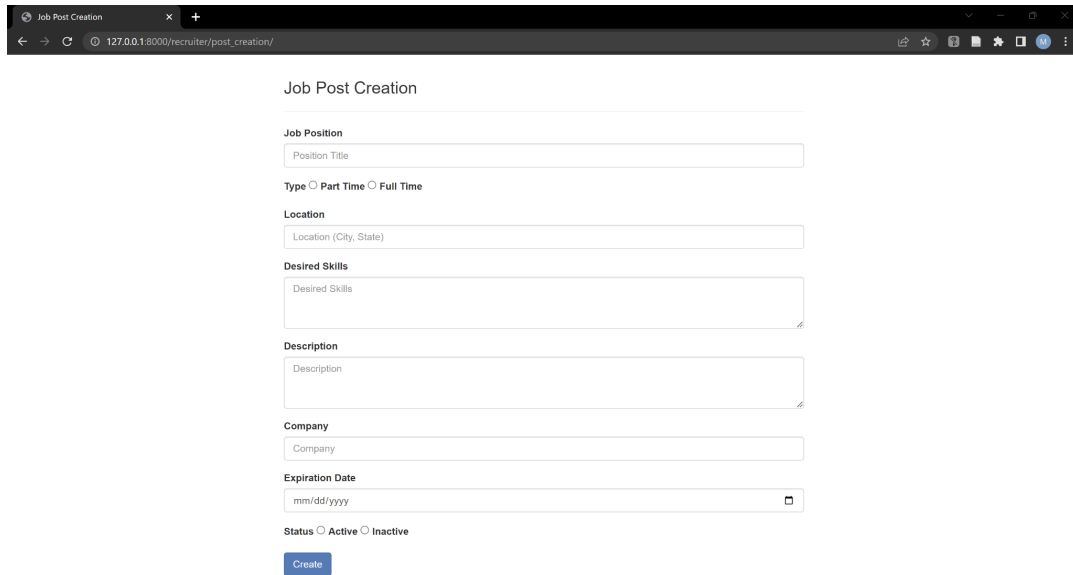


Figure 2.1.4 for feature 2.1 showing the keywords filter

Feature 2.2

Figure 2.2.1 shows a screenshot of the post creation page for recruiters. Here, the recruiter will enter in the job information such as title, desired skills, description, etc. When the recruiter clicks the create button, a new job post linked to that recruiter will be created in the database.



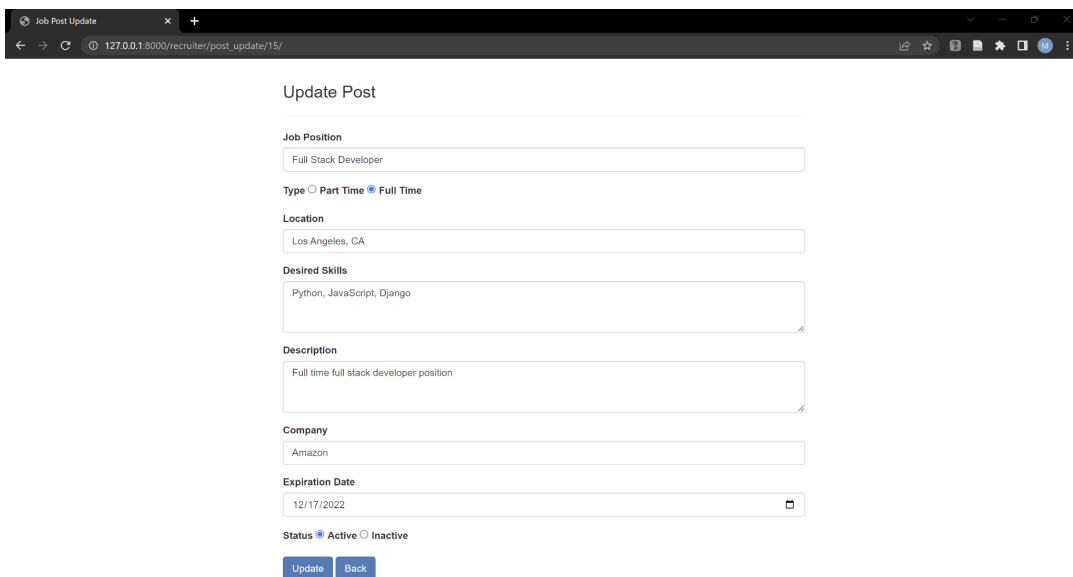
The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/recruiter/post_creation/". The page title is "Job Post Creation". The form contains the following fields and options:

- Job Position:** A text input field with the placeholder "Position Title".
- Type:** Radio buttons for "Part Time" and "Full Time".
- Location:** A text input field with the placeholder "Location (City, State)".
- Desired Skills:** A text input field with the placeholder "Desired Skills".
- Description:** A text input field with the placeholder "Description".
- Company:** A text input field with the placeholder "Company".
- Expiration Date:** A date input field with the placeholder "mm/dd/yyyy" and a calendar icon.
- Status:** Radio buttons for "Active" and "Inactive".
- Create:** A blue button at the bottom.

Figure 2.2.1 screenshot for feature 2.2 showing the post creation page

Feature 2.3

Figure 2.3.1 shows a screenshot of the post update page for recruiters. Here, the recruiter can change any information from an existing job post. When the recruiter clicks the update button, the new information will replace the old information in the database.



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/recruiter/post_update/15/". The page title is "Update Post". The form contains the following fields and options:

- Job Position:** A text input field with the value "Full Stack Developer".
- Type:** Radio buttons for "Part Time" and "Full Time", with "Full Time" selected.
- Location:** A text input field with the value "Los Angeles, CA".
- Desired Skills:** A text input field with the value "Python, JavaScript, Django".
- Description:** A text input field with the value "Full time full stack developer position".
- Company:** A text input field with the value "Amazon".
- Expiration Date:** A date input field with the value "12/17/2022" and a calendar icon.
- Status:** Radio buttons for "Active" and "Inactive", with "Active" selected.
- Update:** A blue button.
- Back:** A blue button.

Figure 2.3.1 screenshot for feature 2.3 showing the post update page

Feature 2.4

Figure 2.4.1 shows a screenshot of the post delete feature for recruiters. This is part of the post detail page. When the recruiter clicks the delete post button, the post is removed from the database, and they will be redirected back to their dashboard.

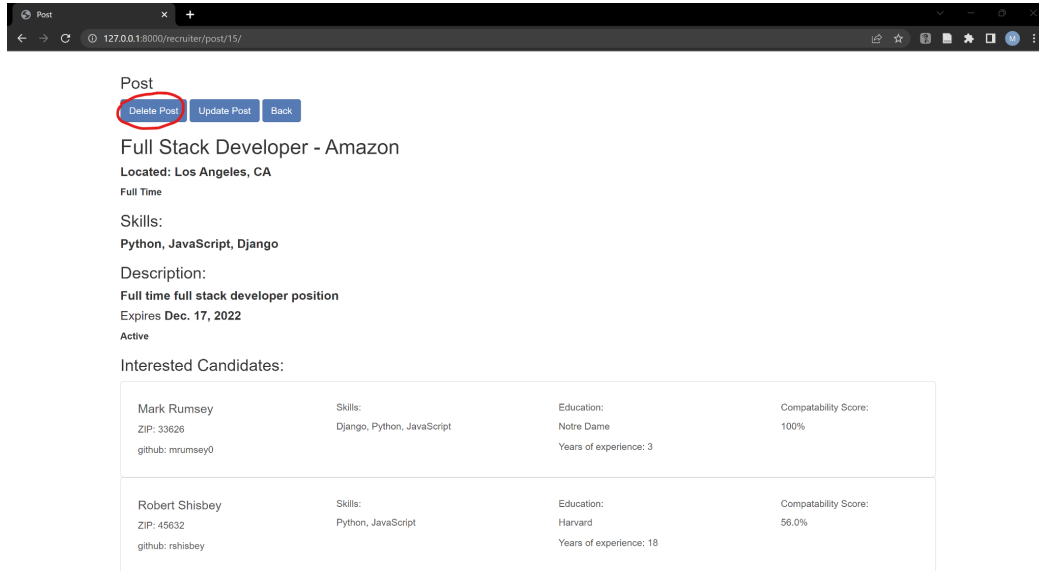


Figure 2.4.1 screenshot for feature 2.4 showing the delete feature

Feature 2.5

Figure 2.5.1 shows a screenshot of the offer page for recruiters. Here, the recruiter can enter the yearly salary of the job and the due date of the offer. When the recruiter clicks the send offer button, a new offer is linked to the job post and the candidate is added to the database.

Post

[Delete Post](#)[Update Post](#)[Back](#)

Compiler developer - CompilerMakers

Located: Tallahassee, FL

Full Time

Skills:

C, compilers, Java, R

Description:

Computer science position in which you will develop compilers for C.

Expires Dec. 23, 2022

Active

Interested Candidates:

| | | | |
|--|--|---|---------------------------------------|
| Daniel Fitzpatrick ZIP: 29089 github: dfitz | Skills: compiler design, C, Java | Education: Notre Dame Years of experience: 6 | Compatability Score: 61.5% |
| Jake Leischner ZIP: 55555 github: jleischn | Skills: R, C, C++ | Education: Boston College Years of experience: 1 | Compatability Score: 50.0% |
| Marco Tan ZIP: 77777 github: mtan | Skills: cryptography, C, gmp library | Education: Notre Dame Years of experience: 6 | Compatability Score: 42.25% |

Figure 2.5.1 screenshot for feature 2.5 showing the interested candidates

Post

[Back](#)

Eric Kim

Zip Code: 88888

Bio:

Theoretical Computer Science expert, researching the P vs NP problem

Skills:

theoretical computer science, python

Experience: 35 years

Education: University of Florida

Github: ekim5

Interested Candidates:

Yearly Salary

Due Date



[Send Offer](#)

Figure 2.5.2 screenshot for feature 2.5 showing the offer page

Feature 2.6

Figure 2.6.1 shows a screenshot of the implementation of the candidate algorithm. Generally speaking, the algorithm consists of the following: first, it scans the skills of the job posting, and checks to see if each skill is in the candidate's skills. The number of skills found is divided by the total number of desired skills, and that serves as one-third of the compatibility score. Next, the algorithm scans the candidate bio and checks to see what percent of the words from the candidate bio are in the job posting. That serves as the second third. Lastly, the final third is comprised of a score based on the years of experience and the type of job (full or part-time). More experience means a higher score, and a full-time job requires more experience than a part-time job. These three factors in total make up the compatibility score for a given candidate in a given position, which is displayed to the recruiter.

```

def expCheck(canExp, jobType):
    if jobType == "full":
        if canExp > 10:
            return 1
        elif canExp > 5:
            return .5
    else:
        if canExp >= 3:
            return 1
        else:
            return .5

def calculateCompatScore(candidate, job):
    compScore = 0

    canBio = candidate.bio.lower().split()
    canSkills = candidate.skills.lower().split()
    canExp = candidate.experience

    jobDescrip = job.description.lower().split()
    jobSkills = job.des_skills.lower().split()
    jobType = job.type

    compScore += .33*bioCompare(canBio, jobDescrip)
    compScore += .33*skillCompare(canSkills, jobSkills)
    compScore += .34*expCheck(canExp, jobType)

    compScore *= 100

    if compScore > 100:
        compScore = 100

    return round(compScore, 2)

def bioCompare(candidate, posting):
    print()
    numerator = 0
    denominator = 0

    # handle extra chars at end of words, like commas and periods
    for index, item in enumerate(posting):
        if not item[-1].isalpha():
            posting[index] = item[:-1]

    # see how similar candidate bio is to job descrip
    for word in candidate:
        if not word[-1].isalpha():
            word = word[:-1]
        if not len(word):
            continue
        if word in posting:
            numerator += 1
            denominator += 1

    result = numerator/denominator
    return 2*result

def skillCompare(candidate, posting):
    numerator = 0
    denominator = 0

    # handle extra chars at end of words, like commas and periods
    for index, item in enumerate(candidate):
        if not item[-1].isalpha():
            candidate[index] = item[:-1]

    # find number of candidate skills in required skills
    for skill in posting:
        if not skill[-1].isalpha():
            skill = skill[:-1]
        if not len(skill):
            continue
        if skill in candidate:
            numerator += 1
            denominator += 1

    result = numerator/denominator
    return result

```

Figure 2.6.1 screenshot for feature 2.6 showing the candidate algorithm

Feature 3.1

Figure 3.1.1 shows a screenshot of the candidate homepage. On this homepage, every job posting available to them is displayed in a list group, with an HTML template that they all follow. The user can then filter the homepage by postings they are and are not interested in, which posts are and are not active, and by name or location. The first two simply filter by the attributes of the postings, and the second two search the bio and location of the posting to see if it contains the given strings. After the filter button is pressed, only the postings that follow these requirements

are shown. The user can then click on these postings to be redirected to their specific post pages.

John Daniel Lee's Dashboard

Job Postings

Logout

View Offers

☐ All
☐ Has Interest

Status

Active

Location

Keywords

Filter

| | | | |
|--|---|---|---|
| <div>Back end developer - Meta</div> <div>Located: San Francisco, CA</div> <div>Full Time</div> | <div>Description:</div> <div>Develop back end for Facebook and Instagram</div> | <div>Skills:</div> <div>Python</div> | <div>Expires May 18, 2023</div> <div>Active</div> |
| <div>Blockchain developer - TaeHo's Blockchain Company</div> <div>Located: South Bend, IN</div> <div>Full Time</div> | <div>Description:</div> <div>designing blockchain technologies</div> | <div>Skills:</div> <div>blockchain, cryptography, C</div> | <div>Expires Jan. 17, 2023</div> <div>Active</div> |
| <div>Compiler developer - CompilerMakers</div> <div>Located: Tallahassee, FL</div> <div>Full Time</div> | <div>Description:</div> <div>Computer science position in which you will develop compilers for C.</div> | <div>Skills:</div> <div>C, compilers, Java, R</div> | <div>Expires Dec. 23, 2022</div> <div>Active</div> |
| <div>Cryptography Research Assistant - Notre Dame</div> <div>Located: South Bend, IN</div> <div>Part Time</div> | <div>Description:</div> <div>Part-time research in fully homomorphic encryption schemes</div> | <div>Skills:</div> <div>C, cryptography, fully homomorphic encryption</div> | <div>Expires Dec. 4, 2022</div> <div>Inactive</div> |
| <div>Front end developer -</div> | <div>Description:</div> | <div>Skills:</div> | <div>Expires April 28, 2023</div> |

Figure 3.1.1 screenshot for feature 3.1 showing the candidate homepage

Feature 3.2

After clicking on a job posting, the user is taken to the job posting's page, as seen in Figure 3.2.1. Here they can see all of the information about a posting, and mark it as interested or uninterested. If they mark it as interested, their id is stored in the `interested_ids` field of the post, and the `interested_candidates` counter is incremented. If they mark it as uninterested, that posting's id will be included in the user's `uninterested_ids` field. Because sqlite3 does not support lists as a field in a database, we utilized string encoding to store more than one value. Any time a user chooses one of interested or uninterested and then the opposite one after, it will revert the previous actions.

Post

[Back](#)

P/NP Theorist - P/NP Inc.

Located: Salt Lake City, UT

Full Time

Skills:

theoretical computer science, C

Description:

Theorizing about the P vs NP problem. Looking for knowledge about various NP-hard problems such as TSP, Knapsack, and HAMCYCLE

Expires Feb. 16, 2023

Active

Interested?



Figure 3.1.2 screenshot for feature 3.2 showing the job posting page

Feature 3.3/3.4

Figure 3.3.1 shows a screenshot of the viewing offers page. From this page, the user can view their offers in a similar visual format to Feature 3.1. They can accept and decline offers from this page, which is then added to the database, but they cannot accept expired offers - for every job posting, a check is made to see if the current date is past the expiration date. If so, the accept and decline buttons are no longer made available.

Mark Rumsey's Dashboard

Job Offers

| | | | |
|--|--|-----------------------------------|-------------------------|
| Logout Back | | | |
| Full Stack Developer at Amazon Located: Los Angeles, CA Full Time | Description: | Salary: | Accept |
| | Full time full stack developer position | \$90000/year | Decline |
| | | Expires Dec. 29, 2022 Active | |
| Front end developer at Google Located: Chicago, IL Full Time | Description: | Salary: | Accept |
| | Develop front-end for google search engine | \$75000/year | Decline |
| | | Expires Dec. 15, 2022 Active | |
| Metaverse Engineer at Meta Located: San Francisco, CA Part Time | Description: | Salary: | Accept |
| | Help design the metaverse | \$95000/year | Decline |
| | | Expires Dec. 15, 2022 Active | |
| Cryptography Research Assistant at Notre Dame Located: South Bend, IN Part Time | Description: | Salary: | expired |
| | Part-time research in fully homomorphic encryption schemes | \$100000/year | |
| | | Expires Oct. 12, 2022 Inactive | |

Figure 3.3.1 screenshot for features 3.3/3.4 showing the offers page and response buttons

Project's Learned Lessons

In this section, you will reflect on the group's activities and performance through the entire project, and capture your reflections, observations and thoughts. It should address the following items, but feel free to expand on these in light of your group's unique experiences.

- 1. What programming paradigm(s) have you chosen to use and why? If you were to start the project from scratch now, would you make different choices? Do you think the paradigm(s) chosen helped (or not) in developing the project?*
- 2. What were the most intellectually challenging aspects of the project?*
- 3. What aspects of your process or your group's organization had the largest positive effect on the project's outcome?*

As a group, we feel like we learned a lot throughout this project.

Throughout the course, we've discussed that programming paradigms are almost always used in tandem, and this was definitely the case in our project. Both imperative and declarative paradigms could be seen. On the imperative side, object-oriented programming played a big role, as each candidate, recruiter, job posting, and offer was essentially its own version of an object template. Passing around and manipulating these objects was at the heart of what the project sought to accomplish. On the declarative side, functional programming was used in cases such as the filtering done on the recruiter and candidate home pages - utilizing Python's filter function allowed us to eloquently manipulate large amounts of data in complex ways, all in just a few lines. There were certainly other examples of paradigms that we used, but in general, the message is that many paradigms worked together to get us to our final result.

One intellectually challenging aspect of the project was its scale - this included planning everything out, making sure all of the pieces fit together, keeping track of what is happening in each of the many files, and even just getting GitHub to cooperate. This is one of the largest projects any of us have worked on thus far at Notre Dame, so handling that size was definitely a big undertaking.

The aspect of our process that really stood out as working really well for us was simultaneous coding - even if we were working on different portions of the project, if we all came together and worked on it at once, we could ask each other questions and quickly move past blockers that may have otherwise taken hours for one person to figure out.