Programing Paradigms Project Final Report:

Team Member Full Name	NetID
Katie Grace	**kgrace2**
Sarah Cullinan	**scullin2**
Sofia Dillhoff	**sdillho2**
Celina Ryan	**cryan27**

Chosen Technology Stack

☐ Python + Django + Bootstrap

Persistent Storage Design

We are using SQLite database for persistent data storage. Our database, as seen in *Figure 1*, contains the following tables: `CanidateProfile`, `RecruiterProfile`, `Post`, and `Offer`. These tables are connected and participate in inheritance as a recruiter can make a post and an offer. A candidate can view an offer and all posts (from recruiters), as well as show interest or not-interest in posts, and accept or decline offers. Additionally, candidates can access posts through the offers they have received. For these reasons. all of these models are connected and associated with each other.

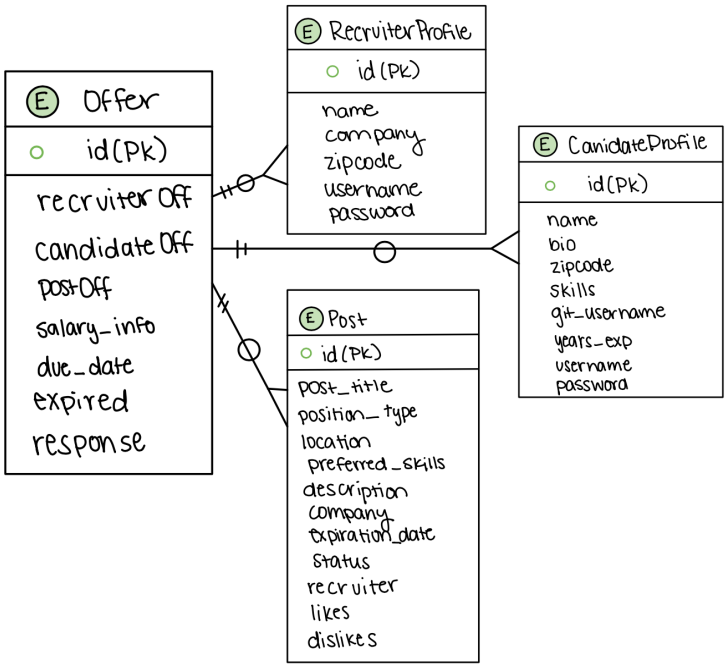

Figure 1 database schema

Feature 1: Sign Up/ Sign In

1.1: Create New Candidate Profile

Figure 3 shows a screenshot for creating a candidate profile. The user is directed to this page when they click on the "Sign up as a Candidate" link on the homepage of our TinDev website (*Figure 2*). The information requested from the candidate includes name, bio, skills, GitHub username, years of experience, username, and password. This allows the candidate to create an account with proficient information for recruiters actively looking for candidates.

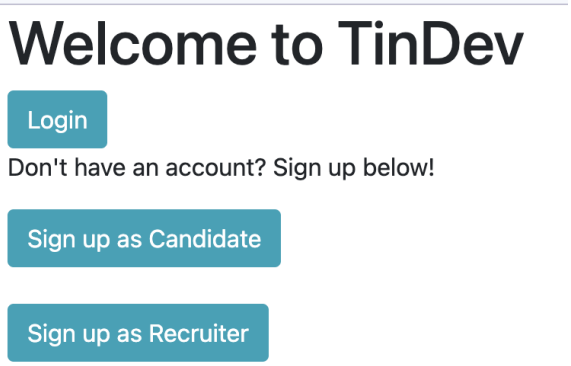

Figure 2 Screenshot for feature 1.1 showing the homepage of our TinDev website

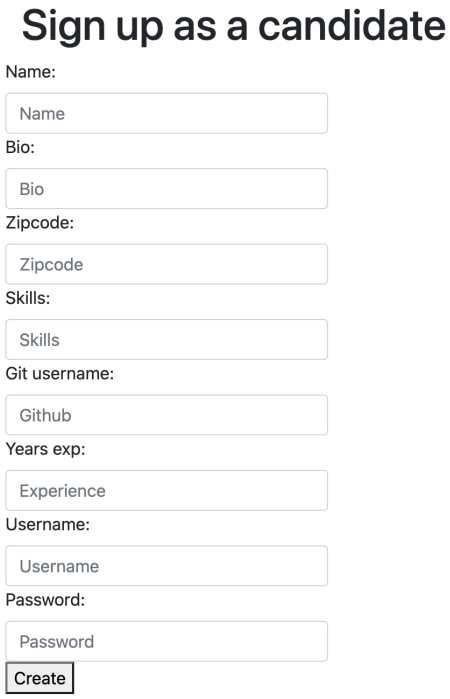

Figure 2 Screenshot for feature 1.1 showing the information a candidate inputs when creating a TinDev account

The website is designed so that no two users can have the same username. If so, the candidate will get the following error notification when trying to sign up shown in *figure 3* below.

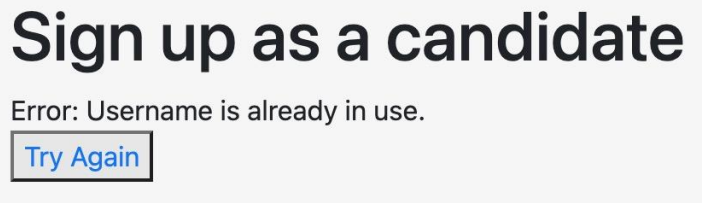

Figure 3 screenshot for feature 1.1 showing a signup error notification when the username is already in use

1.2: Create new Recruiter Profile

Figure 4 shows a screenshot for creating a recruiter profile. The user is directed to this page when they click on the "Sign up as a Recruiter" link on the homepage of our TinDev website. The information requested from the candidate includes name, company, zip code, username, and password. This allows the recruiter to create an account with proficient information that is valuable for candidates actively looking for jobs/internships etc.

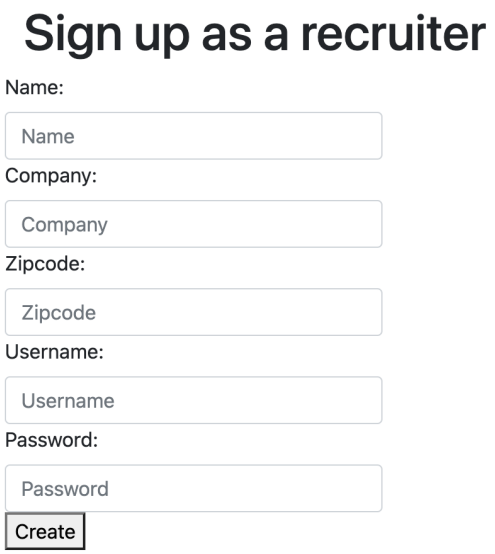

Figure 4 screenshot for feature 1.2 showing the information a recruiter inputs when creating a TinDev account

The website is designed so that no two users have the same username. If so, the recruiter will get the following error notification when trying to sign up, shown in *figure 5* below.

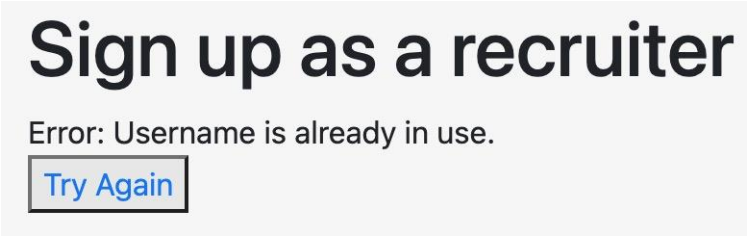

Figure 5: screenshot for feature 1.2 showing a signup error notification when the username is already in use

1.3: Log-in

Figure 6 shows a screenshot for our login page. This allows users who have already created an account to log in, view job postings, and interact with other users on our TinDev website.

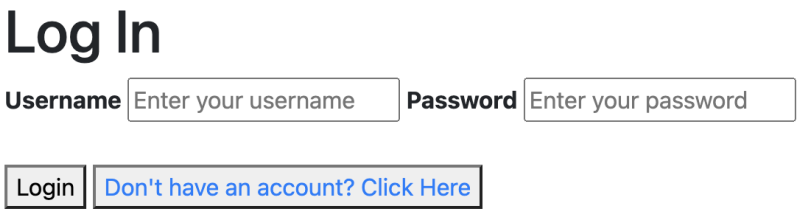

Figure 6 screenshot for feature 1.3 showing the login page

If a user attempts to log in with the incorrect username or password, they will receive the error notification seen in *Figure 7*.

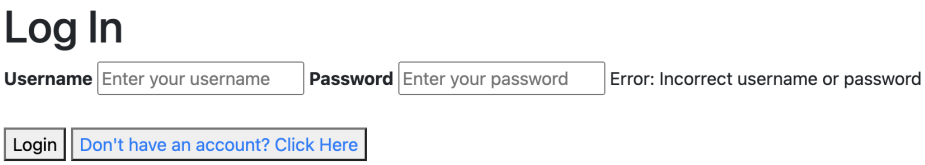

Figure 7 screenshot for feature 1.3 showing login error notification when they have entered an incorrect username/password

1.4: Log-out

Figure 8 shows screenshots of both the Recruiter and Candidate Dashboards (which are separate links). Users are taken to this webpage when they log in. Once at the dashboard page, users can click the "Logout" button, and they will be redirected to the login page, as seen in *figure 6*.

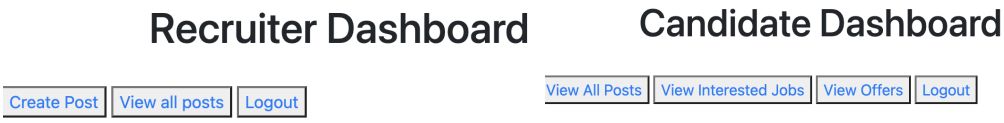

Figure 8 screenshots for feature 1.4 showing the logout buttons on the dashboards

Feature 2: Recruiter's Dashboard

2.1: View all posts

Figure 10 shows a screenshot of the Job Postings page, which the user is redirected to when they click the "View all posts" button on the recruiter dashboard home page as seen in *Figure 9* below. Note that a recruiter can only see the jobs that they have posted and not the jobs other recruiters have posted. For example, let's say that Grigorii works for the Cincinnati Reds. This means that he can only see the jobs that he posted and not what other recruiters have posted, as seen in *Figure 10.*

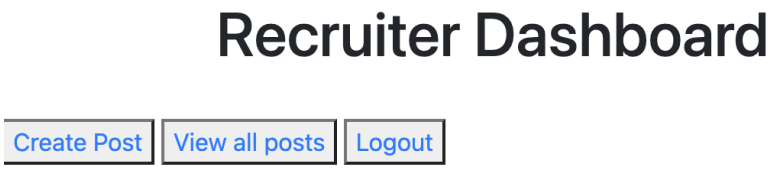

Figure 9 screenshot of the Recruiter Dashboard home page

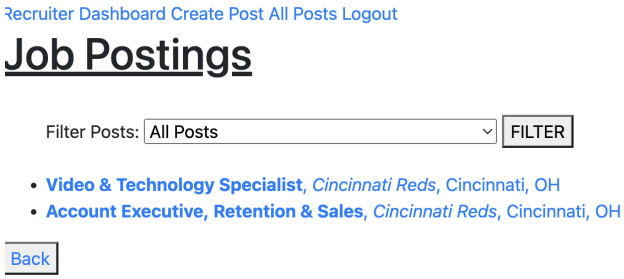

Figure 10 screenshot for feature 2.1 showing Job Postings page

If Grigorrii tries to "View all Posts" and he has not made any posts, he will be redirected to the page seen in *Figure 11.*

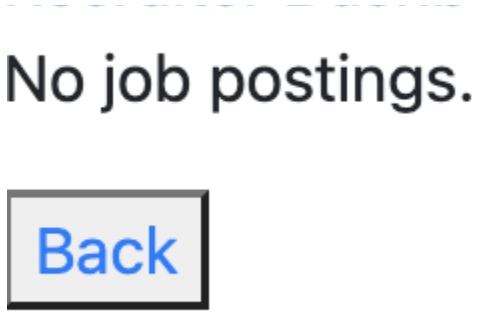

Figure 11 screenshot of the Recruiter Dashboard home page

Finally, if a recruiter has made posts, they can filter them with the dropdown menu shown in *Figure 12.*

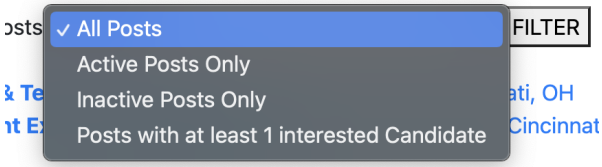

Figure 12 screenshot to filter the Job Postings in feature 2.1

In this example, if Grigorri selected to filter his posts to only see active posts, he would only see the post for the Account Executive (and not the video and technology specialist) since it is the only active post as seen in *Figure 13* below. If no posts fall under the filter Grigorri chose, then he would be redirected to the page in *Figure 11*.

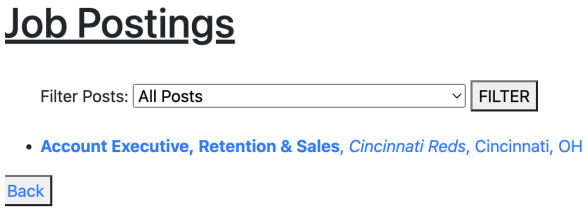

Figure 13 screenshot of filtered job posting for feature 2.1

2.2: Creating a post

From the recruiter dashboard, when a recruiter clicks on the "Create Post" button, they are led to this form, as seen in *Figure 14*. Recruiters then input the necessary information to create a post for candidates to view.

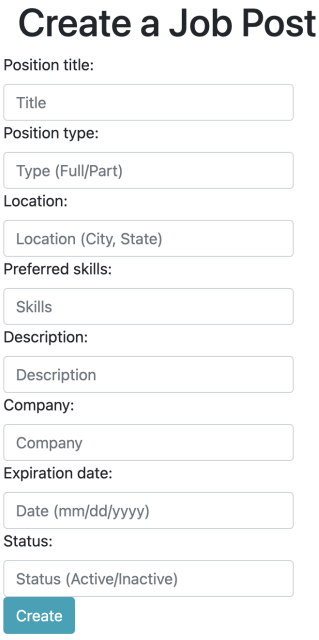

Figure 14 screenshot for feature 2.2 of the Create a Job Post form

If a recruiter attempts to submit a form and leaves a field blank, they receive the error notification seen in *figure 15*.

Figure 15 screenshots for feature 2.2 when a recruiter does not properly fill out the form

2.3 Updating a post

When a recruiter wants to update a post, they must click on the "View all Posts" button on the recruiter dashboard (*Figure 9)*. Then they must click on the specific post they want to update, and they will be redirected to the page shown below in *Figure 16,* which is the detail view for a specific post. At the bottom of this page is an "Edit Post" button. The recruiter will finally be redirected to the webpage shown in *Figure 17,* where the recruiter can update and make changes to the job postings they have created through their already completed form.

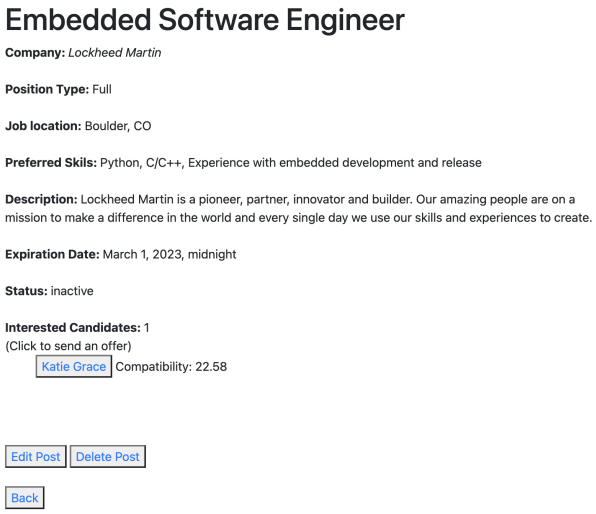

Figure 16 screenshot for feature 2.3

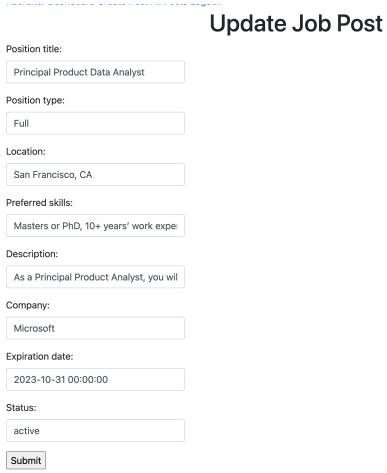

Figure 17 screenshot for feature 2.3 to update a post

2.4 Deleting a post

To delete a post, the recruiter will follow a similar process that they did to update one. Once the recruiter gets to the Job Postings page, they must click on the job posting they would like to delete and click the "Delete Post" button, as seen in *Figure 16*. Once the recruiter does so, the specified post will be deleted from the database and no longer be seen on the job postings page (see *Figure 18)*.

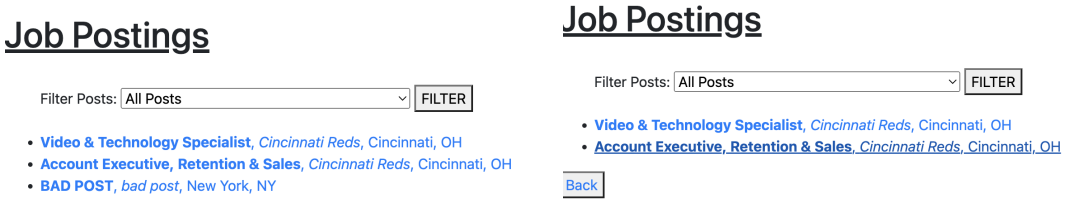

Figure 18 screenshot for feature 2.4 before (left) and after (right) deleting "BAD POST"

2.5 Make an offer to interested candidates

To make an offer to interested candidates, the recruiter will click on the job posting of the job they want to offer, then click on the person's name who they want to send the offer to (see *Figure 16)*. Once they do so, they are redirected to the page shown in *Figure 19*, they can then input the offer information and click create to send the candidate an offer.

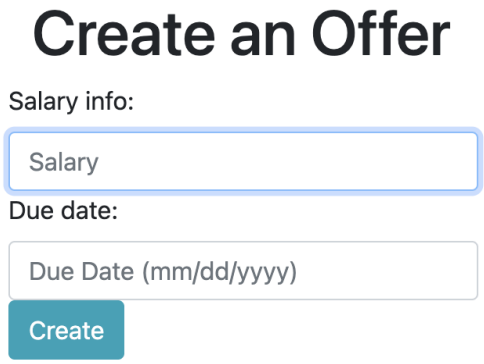

Figure 19 screenshot for feature 2.5 when a recruiter enters in offer details

2.6 View compatibility with interested candidates

When a recruiter clicks on one of their job postings and gets a detailed view at the bottom of the page. When the recruiter sees the interested candidates (for that specific job posting), they will also see the compatibility score next to the user's name, as seen in *figure 20.* This can help a recruiter better evaluate their decision on giving a candidate an offer.

Figure 20 screenshot for feature 2.6

Feature 3: Candidates' Dashboard

3.1: Viewing Job Postings

When a candidate logs in, they are directed to the Candidate Dashboard (*Figure 21),* to view the job postings, the candidate must click on the "View All Posts" button where they will be directed to the job postings page as seen in *Figure 22.* Unlike a recruiter, a candidate can see the job posting created by everyone.

Figure 21 screenshot for feature 3.1, the candidate dashboard

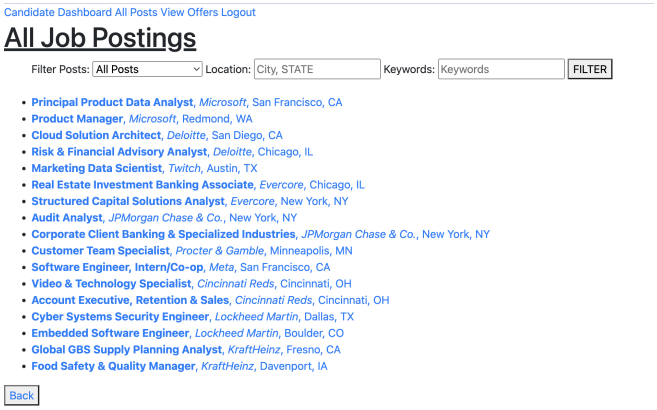

Figure 22 screenshot for feature 3.1 of the Job Postings page for candidates

If the candidate wants to, they can filter these posts with the drop-down menu (*Figure 23)* to see active and inactive job postings. Additionally, they can enter keywords and a location to refine their job search. The user can choose to filter one, multiple, or all categories.

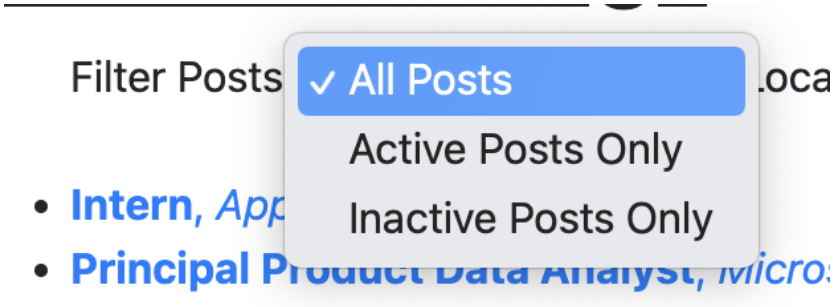

Figure 23 screenshot for feature 3.1 of the filter dropdown menu for candidates

3.2: Demonstrating Interests/ not interested in a job

When a candidate wants to demonstrate an interest in a job, they will first go to the index view of the job postings page (*Figure 22)*. From here, the candidate can then click on a specific job and view the posting with more detail (detail view), as seen in *Figure 24*, and at the bottom of the post, there are "interested" and "not interested" buttons. When the candidate clicks on either one, they will be redirected to the job posting page.

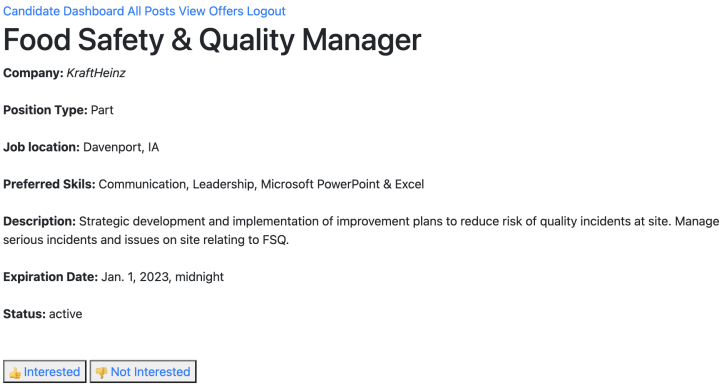

Figure 24 screenshot for feature 3.2

From the candidate dashboard, a candidate can click on the "View Interested Jobs" button to see the job postings they have liked/ demonstrated interest in, as seen in *Figure 25* below.

Figure 25 screenshot for feature 3.2

Additionally, on the recruiter side, when a recruiter clicks on one of their job postings, they can see if any candidates are interested in the posting, and if so, they will see both the total number and the names (see *Figure 26 and Figure 27)*.

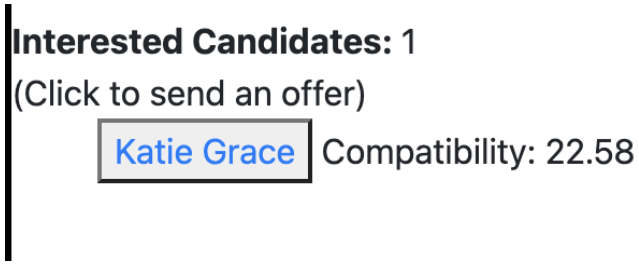

Figure 26 screenshot for feature 3.2 illustrating what a recruiter sees when there are interested candidates

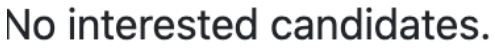

Figure 27 screenshot for feature 3.2 illustrating what a recruiter sees when there are NO interested candidates

3.3: Viewing offers

On the candidate dashboard, candidates can click on the “View Offers” button to see if they have any offers in index view as seen in *Figure 28.* They can click on the index view to get a detailed view of the offer, including salary information and expiration date (*Figure 29).* If a candidate does not have any offers, they will see a “no offers available” message *(Figure 30).*

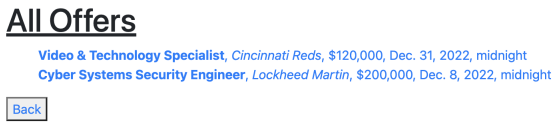

Figure 28 screenshot for feature 3.3 for a candidate viewing their offer

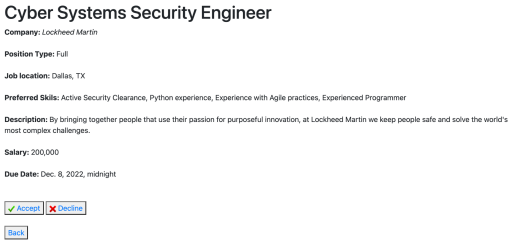

Figure 29 screenshot for feature 3.3

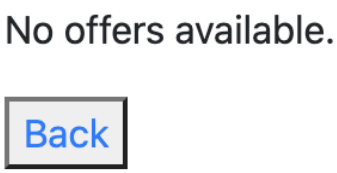

Figure 30 screenshot for feature 3.3 when a candidate has no offers

3.4 Accepting/Declining offers

After a candidate clicks on the “View Offers” button on the candidate dashboard, they can click on specific offers to get a more detailed view of the offer (salary, expiration date, etc.). If the offer has not yet expired, they can either click on the “Accept” or “Reject” buttons to make a decision on a job to take (*Figure 31).*If the offer has expired, the date on which it expired is clearly indicated.

Figure 31 screenshot for feature 3.4 of a Candidate's options to accept/decline an offer

Finally, if a candidate has an offer that has expired, they can no longer accept/decline the offer, but they can still view the offer as seen in *figure 32* below.

Figure 32 screenshot for feature 3.4 when a candidate's offer has expired

Project's Learned Lessons

We used Django, a high-level Python web framework, for this project. The paradigms we used include imperative programming, object-oriented programming, and event-driven programming.

Python is an imperative language with a light and uncluttered feel to it. Implementing imperative programming allowed us to optimize our code, making it easier to understand and debug than implementing declarative programming.

The event-driven paradigm allowed us to change our program based on user actions. We chose this so we could get user input and so users could have an easy time interacting with our TinDev web application. Event-driven programming allows users to enter their personal information, create posts/offers, view postings/others' profiles, filter results, react to the postings/offers, etc. This was done through HTML DOM events.

Finally, python uses class-based OOP. Object-Oriented programming allowed us to use class inheritance which was helpful when connecting objects and models.

The paradigms we chose helped develop our project as we utilized DOM and user interactions with the webpage, optimized our code, and created inheritance to connect the different aspects of our project.

One of the significant factors of this project that made it so intellectually challenging was our steep learning curve. Beginning this project was challenging. This project was more complex than the examples we had seen in class. Also, there were few examples online to refer to; if there were examples, they were hard to follow and relate to our project. Therefore beginning this project and fully understanding how Django works were challenging (so difficult we had to re-do phase 1), but we eventually got the hang of things and were able to speed up our work.

Another difficult portion of our project was that our frontend and backend variables could not connect. As we were going through candidate and recruiter profiles, there were certain objects we could not access on the front end. This is done for safety reasons (as we do not want random accounts editing others' accounts, posts, and interactions). However, it made things frustrating as we could not access and modify objects as easily.

Additionally, comprehending the models and what connections to make was intellectually challenging. For example, an offer has a recruiter, candidate, and post all associated with it. Figuring out how to make these connections and code them was even more difficult, but with the help of the TAs and Professor Santos, we understood and made progress in our code.

Our group's organization and focus contributed to our success and learning. We were very organized in planning our meeting times and created a shared google calendar to coordinate these daily times. For some of the project features, we worked together on them rather than splitting up the work. This made us more efficient as working on these alone would be difficult. It was helpful to get multiple eyes on the code and different opinions/ideas on adding the different features to our project and getting them fully functioning.

Additionally, attending office hours, asking questions before/after class, and reaching out to TAs (especially Grigorii) and Professor Santos via CampusWire helped tremendously. We often needed help understanding the bugs in our code, but we were able to get help. Also, this

was our first time collaborating on a project via a GitHub Repo. GitHub was super easy to use, and it was helpful to see the history of our code changes and share this with Professor Santos and the TAs for them to look at and interact with our code. Overall, our organization and consistent meeting times were critical to our group's success. This collaboration allowed us to learn from each other and test our ideas.