Campus Jobs

Final Report

CS 4624: Multimedia, Hypertext, and Information Access

Instructor: Dr. Edward A. Fox

Client: Trey Mayo, Director of Graduate Programs, Department of Computer Science

Date: Dec. 8, 2021

Authors: Adam Chen, Emily Ashburn, Nuvya Baliyan, Matthew Hunderup

Table of Contents

List of Figures

List of Tables

1 Executive Summary

The Campus Jobs project has one clear goal: to connect graduate students to work opportunities. Many graduate students enrolled in Computer Science struggle to find work and are forced to knock on the doors of professors in hopes of finding research opportunities. Upon successful completion of our project, we hope to allow these students easy access to jobs in Blacksburg. Our client, Trey Mayo, is the Director of Graduate Programs at Virginia Tech. He proposed that we create a website similar to LinkedIn, but with an important facet being it specializes in opportunities for graduate students in Blacksburg [7]. The structure and design of this website was left open for us to explore. We elected to host a website on a designated Virginia Tech Virtual Machine on the cs.vt.edu domain. Additionally, we hosted a database in MySQL on the same machine in order to store student and job information. We designed the website to allow for student and faculty login, as well as storing general data about the student and information about the student's candidacy for the job. We implemented a matching algorithm to allow for students and professors to find the most appropriate matches.

In this report, we give a detailed explanation of the work that we completed for our client including the design, approach and plans for future work. We also provide a detailed developer's manual and user's manual for others to easily understand the final product.

2 Introduction

According to the Graduate School Data and Reports of Virginia Tech, there are over 600 graduate students in the Department of Computer Science, 70% of whom are international students [11]. International students are typically enrolled with a student visa, meaning that they are not able to find a source of income outside of the learning institution they are enrolled in, which provides a struggle for students trying to find funding. While some students are offered a GRA or GTA position through CS, there are many others who are not so fortunate. As such, it is important to provide these students with a way to find job opportunities that are with Virginia Tech. We partnered with our client, Trey Mayo, to provide a service that students can use to find employment opportunities and employers. The service also allows Virginia Tech staff to display openings they have available and jobs they need assistance with.

3 Research

In order to gauge users' needs and expectations from our service, several graduate students and professors were interviewed. This research not only helped provide new ideas for the User Interface and Design, but also validated ideas that we already had regarding the service. These interviews included questions regarding the UI experience (UX) as well as features in the service that a potential employer or job seeker would want to see.

For the employer perspective, the following questions were asked:

1. Have you ever posted a job opportunity anywhere?
2. What are some of the most important pieces of information you would want to know about an applicant? Classes taken, language skills, research participation, etc.?
3. What kind of information would you think is important to provide about the job opportunity?
4. Would you like some kind of pre-screening system for applicants' skills (i.e., only applicants with a 3.0 or higher may apply, applicants must have experience with Python, etc.)?
5. Would you be fine with students performing a self-assessment of their skills or would you prefer less subjective measurements?

From these interviews we learned a few things about the employers' expectations. Employers wanted to see a candidate's citizenship status, Ph.D. vs. Masters status, degree program or concentration, interests, relevant coursework, and previous experience. Moreover, they wanted us to provide a scale to specify experience for each skill that the candidate includes on their application. With regards to the UI, the employers want a simple design that allows them to track applications efficiently.

For the candidate/student perspective, the following questions were asked:

1. Would you want personal information about the professor, i.e., past experience, credentials, languages of interest?
2. What information would you like to know about potential job opportunities?
3. Would you like to see the salary information for a job displayed?
4. What are some important features that you would like to see in the service?
5. What is something you have seen in another application service that you liked/did not like?

From these interviews we learned about features and information on the service that were important to students. For example, the students would want this service to be simple, and would want information required from them not to be redundant. If a position has been filled, it should be removed from the website. Moreover, they would want to see the salary information for each position and the required or preferred qualifications expected from each candidate. Lastly, it was very important to the students that the matching service be accurate and efficient so that the best jobs were recommended to them for application.

We also asked students to demonstrate what their application process was like on another service, LinkedIn (except for one student who did not use LinkedIn, so we asked him to show using Google search). We tried to be as hands-off as possible in this demonstration, only asking why the student performed some actions and asking what could be improved upon after the process. Not only did this provide us with a good understanding of the types of information or features that a job-seeking student is looking for, but also showed us the UI flow for various actions within the application. For example, each student took a different route to navigate through the website to perform the same action.

In addition to the professors, semantics and features of the service were discussed with our client Trey Mayo. While discussing expectations regarding the service with him, the following questions were asked:

1. Is Virginia Tech the only employer? This would be important to consider for students who require special visas.
2. Is there a specific UI/platform that you would like for the matching service?
3. Should there be restrictions on applicants such as those with a VT email only?
4. Should we support accounts/account management, or should it be a ‘guest’ service?
5. What could the matching service look like?
6. What would you want the user and employer experience to look like?
7. Would it be a good idea to have a counter for how many people applied to a job?
8. What info should we require from an employer about a job opening? Should the salary be included?
9. What information should we track for each potential employee? Do you have a complete list of attributes to track for both?
10. What kind of checks should we place for employers before they can publish a job?
11. Are the jobs restricted to CS students? Will our service only be shared with CS students? Or is that something our service should handle?
12. What information about a candidate would you like for us to track?

4 Requirements

1. A database to manage data from students and employers.
2. A matching service that matches candidates with jobs that match their skillset.
3. A simple user interface that supports:
    a. Entering and updating data
    b. Queries and matching requests
    c. Management and other reports
    d. Account creation and management

5 Design

Steps to implement database and connection with our web site:

1. Creation of CampusJobs website's database and table.

    Figure 5.1. Columns for the table t_user.

id	username	password	real_name	email	phone_number	city	skills	experience	resume

Figure 5.1 showcases the various columns in our 'campus_jobs' database. This database table includes a row of data for each 't_user' in the application.

2. Start to create JavaBean class and variables that accord with our database.

    Figure 5.2. Java User Class.

```
public class User {
    private Integer id;
    private String username;
    private String password;
    private String type;
    private String firstName;
    private String lastName;

    public User(Integer id, String username, String password, String type, String firstName, String lastName) {
        this.id = id;
        this.usern
        this.passw
        this.type = type;
        this.firstName = firstName;
        this.lastName = lastName;
    }
```

The ID, username, password, type, firstName, and lastName variables are in accord with the attributes in our database.

3. Start to implement JDBCUtils class and use Junit to test.

   Figure 5.3. JDBCUtils Connection code.

```java
/**
 * establish a connection with the data source.
 * @return if return null, it means connection failed.
 */
public static Connection getConnection() {
    Connection conn = null;

    try {
        conn = dataSource.getConnection();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return conn;
}
```

   Figure 5.3 shows part of our code in the JDBCUtils class to establish a connection with the data source.

   We use this part of code to create a MySQL object to establish the MySQL connection.

4. Start to implement the DAO (Data Access Object) class for database interaction.

   Figure 5.4. Database Connection Interface.

```java
/**
 * search and check the users info in the database
 * @param username username
 * @return return null if the user doesn't exist in the database.
 */
public User queryUserByUsername(String username);

/**
 * keep user's information
 * @param user
 * @return
 */
public int saveUser(User user);

/**
 * search users' information based on their username and password.
 * @param username
 * @param password
 * @return
 */
public User queryUserByUsernameAndPassword(String username, String password);
```

Figure 5.4 shows the interface of our DAO class for handling the different requests and sending responses back to the users.

5. Start to write the Service class to handle multiple requests from users, such as user login, user registration, checking account existence, and so forth.

Figure 5.5. Service Class Interface.

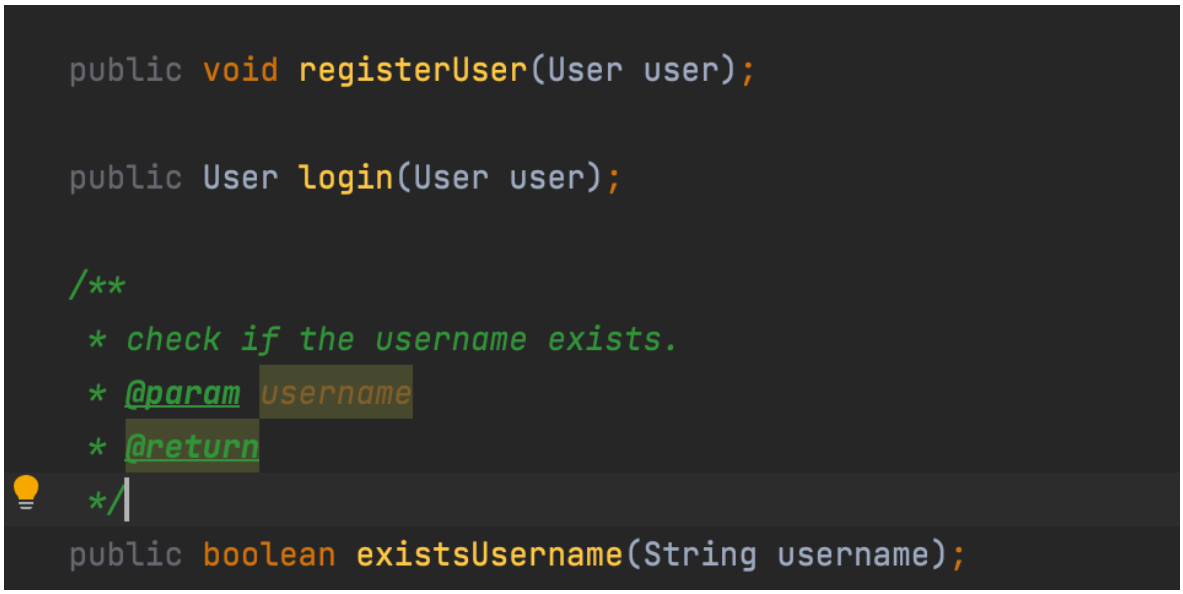


Figure 5.5 shows part of our service class interface example code to process the requests from the users.

6. Start to implement a web servlet to handle the post and get requests from the users.

To query the database, we utilized the queryUserByUsername() function. Figure 5.6 shows the web servlet implementation and the various functions that are used in this implementation. The use of userDao.queryUserByUsername(username) has been made to query and find the exact user JavaBean object. After that we are using the HttpSession object to store the information and using HttpObject to send the data back to the web page to fulfill the interaction between the database and the front-end web pages.

Figure 5.6. Web Servlet Implementation.

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String id = request.getParameter( s: "user_profile");
    HttpSession session = request.getSession();
    String username = (String) session.getAttribute( s: "username");
    User user = userDao.queryUserByUsername(username);
    if (user.getReal_name() != null){
        session.setAttribute( s: "real_name", user.getReal_name());
    }
    if (user.getEmail() != null){
        session.setAttribute( s: "email", user.getEmail());
    }
    if (user.getPhone_number() != null) {
        session.setAttribute( s: "phone", user.getPhone_number());
    }

    if (user.getCity() != null) {
        session.setAttribute( s: "city", user.getCity());
    }

    if (user.getSkills() != null) {
        session.setAttribute( s: "skills", user.getSkills());
    }

    if (user.getExperience() != null) {
        session.setAttribute( s: "experience", user.getExperience());
    }

    request.getRequestDispatcher( s: "pages/user/user_profile.jsp").forward(request, response);
}
```

5.1 Frontend

For the static website, we use the following languages:

- HTML
- CSS
- JavaScript
- JQuery
- Ajax

For the dynamic part, we are using the Java Server Pages (JSP) to transfer the data from the website to the server [6]. For the implementation, both JSP files and HTML files are stored in the web directory. We are using Ajax to communicate with the database before clicking the submit button to send data to the server. The purpose of doing that is to check the validity of data.

5.2 Database design with MySQL

For our project, we decided to use Java to implement the web application. Therefore, we are using JDBC (Java database connectivity) for our data access [5]. MySQL is a secure and stable database that is also compatible with JDBC drivers. Therefore, we decided to use MySQL as our database. The MySQL driver for JDBC in our project that we are using is in the mysql-connector-java-8.0.26.jar file.

Creating database connections constantly is costly and inefficient. Therefore, we need to construct a database connection pool to create some database connections and store them in the pool, so it will be much more convenient for multiple data queries. The database connection pool we are using is Druid, which provides powerful monitoring functionalities on the database connection pool. To successfully run Druid in our code, we need to download the druid-1.1.9.jar file and put it under our lib directory [8]. In order to incorporate Druid, we created a Druid configuration file and jdbc.properties file as shown in Figure 5.7.

Figure 5.7. Configuration File.

As we can see in the configuration file, the username is the username of our database. The password is the password of our database. The URL entry (“url”) gives the relative directory plus the name of our table in our MySQL database. The maxActive property is the maximum database connections we are able to create in the pool.

For our database, we have 2 types of users, and each type of user has the following properties in the database:

Students:

- id
- username
- password
- real_name
- email
- phone_number

- city
- skills
- experience
- resume

Employer:

- id
- job_name
- city
- job_description
- skills
- email
- phone

5.3 Matching Service

For the matching service, we encountered a lot of different design drafts. Initially we thought we would have students fill out a form, and jobs would be recommended to the student based on their input, but we found that this idea would be very hard to implement due to the diversity of job offerings. We also wanted to reduce redundancy since many of the graduate students interviewed said this was their biggest complaint when it came to other job application services. We also got feedback from a professor implying that they wanted a quick way to make sure a candidate was qualified for the position so they didn't have to sift through numerous resumes. So, we decided to have the employer post a job opportunity with relevant tags attached. Then, when the student is looking for jobs, they will be able to filter through the openings by selecting tags from a list and can apply to jobs they're interested in. Upon account creation, students will create a profile with relevant information like how many years of experience they have, what languages they enjoy, and whether they are a masters or Ph.D. student.

6 Implementation

Table 6.1. Table of Tasks/Subtasks selected for implementation

Service ID	Service Name	Input file name(s)	Input file Ids	Output file name	Output file ID	Libraries; Functions; Environments	API endpoint (if applicable)
1	Client login	login.jsp	9	client.jsp	5	Tomcat server	HTTP requests

2	Student login	login.jsp	10	student.jsp	6	Tomcat server	HTTP requests
3	Client's job requirement	clien_profile .jsp	11	result.jsp	7	Tomcat server	HTTP requests
4	Student job matching	student_profile .jsp	12	result.jsp	8	Tomcat server	HTTP requests
5	Student Profile Creation	Student_attributes.jsp, resume.pdf	13, 14	student_profile.jsp	12	Tomcat server	HTTP requests
6	Apply Profile to job	student_profile.jsp	12	result.jsp	15	Tomcat server	HTTP requests

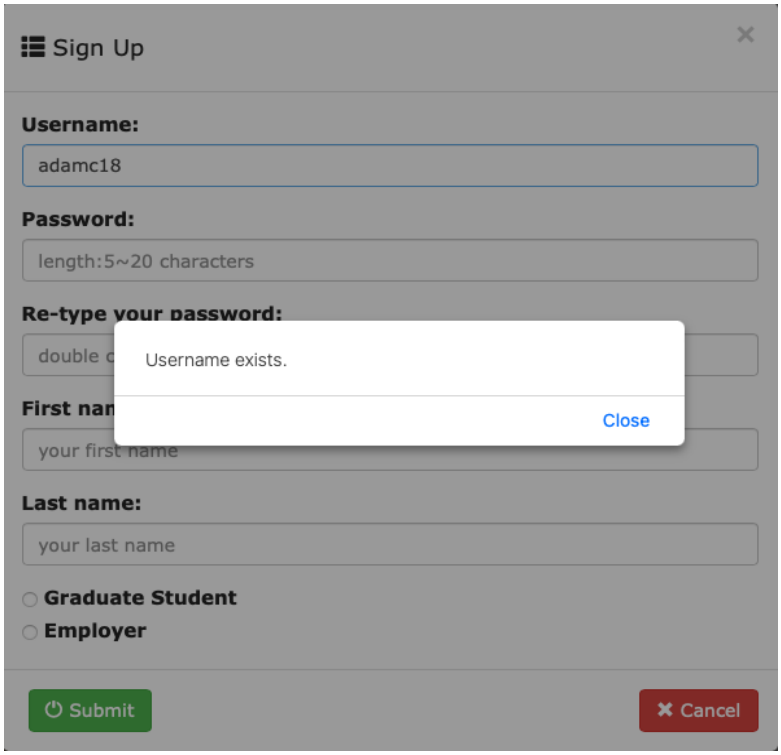
6.1 Login Page

The first page created was the login page. This page will be the first page on the website, so an account is required in order to view other pages like the find a job page. This sort of restricted access was requested by the subjects interviewed so student profiles and job applications can remain private. Currently, we have only focused on the functionality of this page. In the final implementation we plan on making all pages dynamically-sized using JS to support mobile-viewing. Figure 6.1 shows the current login page.

When a user does not have an account, they will use the sign-up page as shows in Figure 6.2. While our goal was to implement CAS authentication, we did not have the time or ability to implement it currently. Our current system simply verifies users by identifying if their email address is a valid Virginia Tech email address upon signing up..

Figure 6.1. A screen dump of the current login page draft

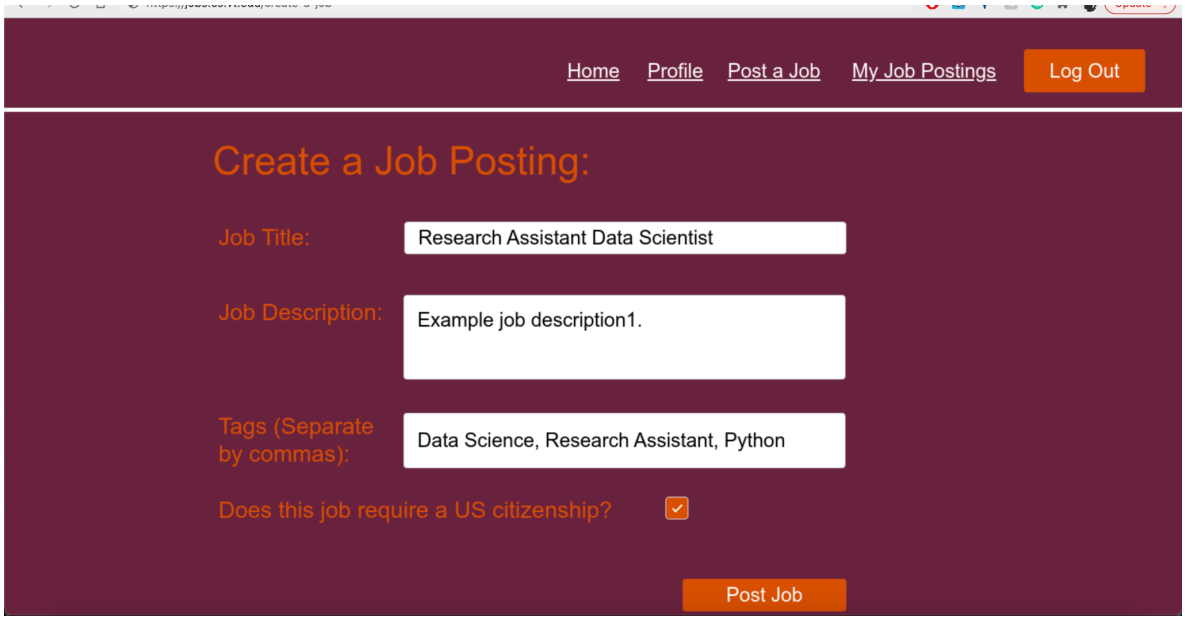
Figure 6.2. A screen dump of the current sign-up form



6.2 Matching Service

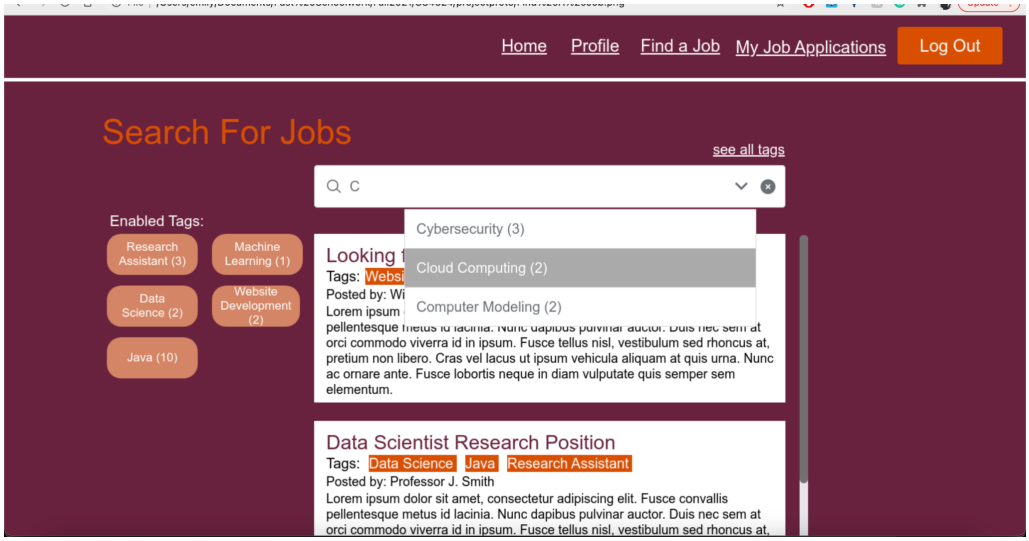
At the heart of our program is the matching service. When an employer wants to post an available position, they can navigate to the “Post A Job” page (which is only open to employer-type accounts). Here they would fill out a form detailing the position, tagging it with relevant tags. While tagging, we plan to implement auto-complete, in order to reduce redundancy in tags. This way, employers are first prompted to use an existing tag rather than to create a new one. However, if an existing tag is not found, then they would create a new tag.

Figure 6.3. A screen dump of the current draft of the “Post A Job” page



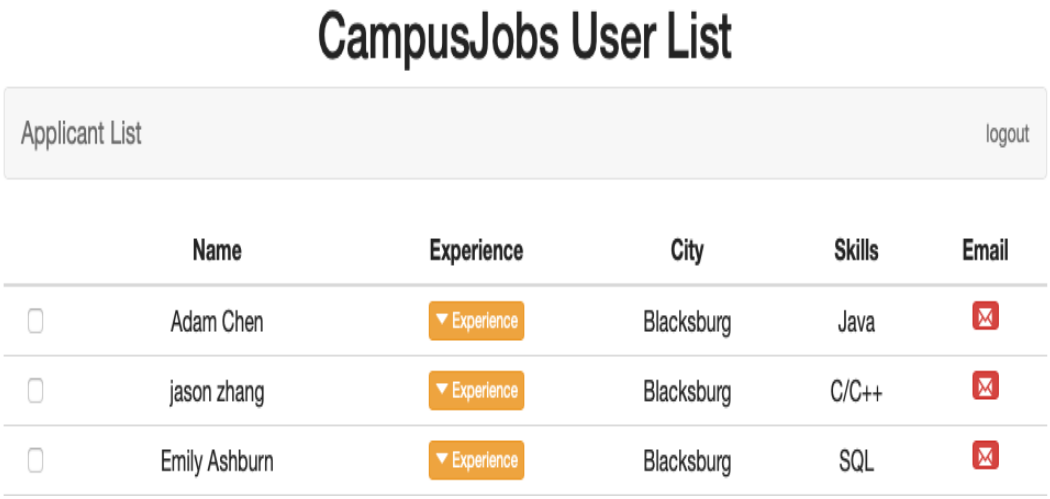
After a job has been created, student users can find the position on the “Find a job” page. This page lists all job positions with a search bar so students can search tags for the positions they’re interested in. Students can also click “see all tags” if they don’t want to search and instead want to choose from a list of tags. When a tag has been enabled, it performs the OR query on job positions, so that if a job contains an enabled tag, it will be shown.

Figure 6.4. A screen dump of the find-a-job page showing the auto-complete system.



6.3 Employer view

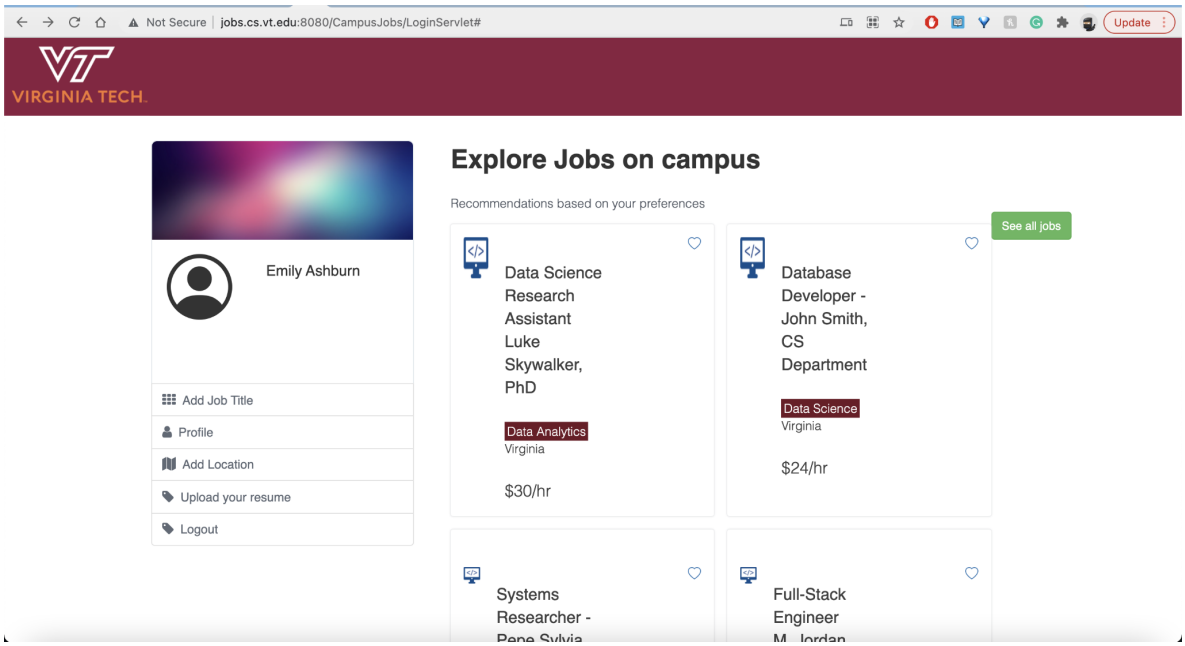
Figure 6.5. List of potential applicants shown in employer view.



6.4 Student view

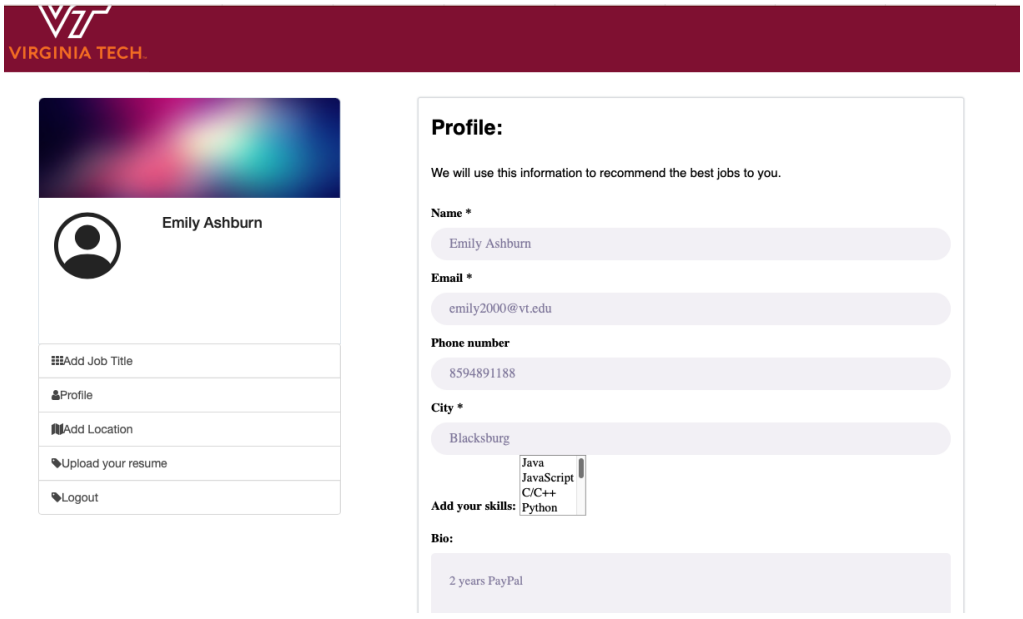
As shown in Figure 6.6, after students successfully login to the page, students can view and edit their profile. Also, they can view the recommended jobs on the page. Furthermore, they can click on the “see all jobs” button to view all the jobs.

Figure 6.6. A screen dump of the student’s homepage



When students click on the profile button, they can edit their profile. Those data students enter will be sent to the database. Also, after the students save their profile page, they can see their latest information the next time they enter the profile page.

Figure 6.7. User profile page.



7 Testing

In order to ensure that our service is accessible and usable, testing it with real potential users was crucial. Initially, we decided to wait until our service was fully functional before we started testing it. However, upon discussing this approach with Dr. Fox, we realized that it would be a better approach to test each aspect of the website as it was being implemented. This incremental testing approach ensured that our website, database, and matching system were working together appropriately, and made it much simpler to debug issues. We tested database entry, website pages, and more as the code was written which allowed us to identify early issues.

We have successfully completed our first round of beta testing, which was carried out by each member in our team. This initial testing introduced us to an array of bugs, most of which we were able to fix. For example, an SFTP bug prevented users from signing up for a new account. Figure 7.1 includes a screenshot of the error message resulting from this bug. However, there were a few bugs that we were unable to debug. For example, while on the user profile, an error occurs while trying to download an uploaded resume to the profile. Figure 7.2 shows the error message produced by this feature. Unfortunately, even after much debugging, we were unable to fix this issue, and hope that this is something that can be fixed in the future.

Figure 7.1: SFTP bug from user profile sign-up

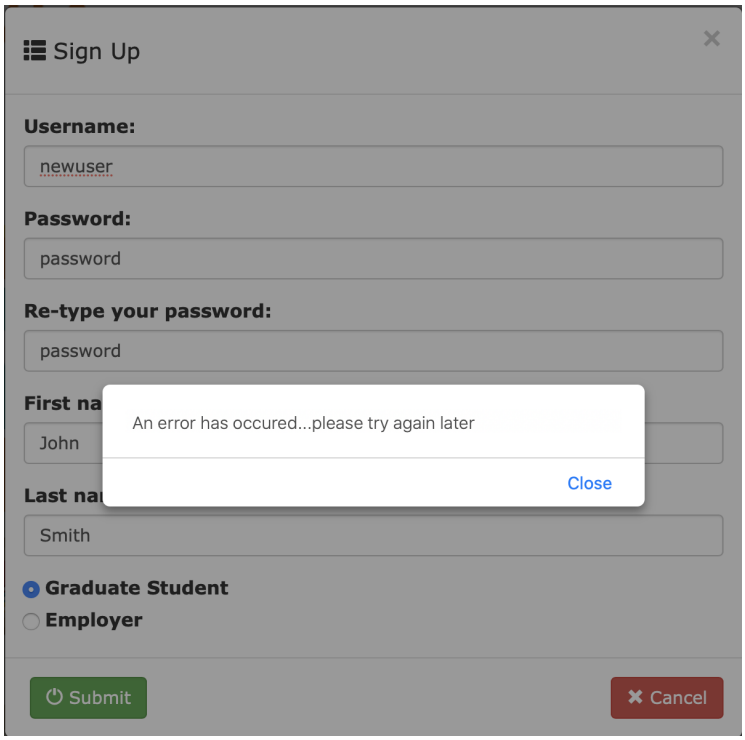
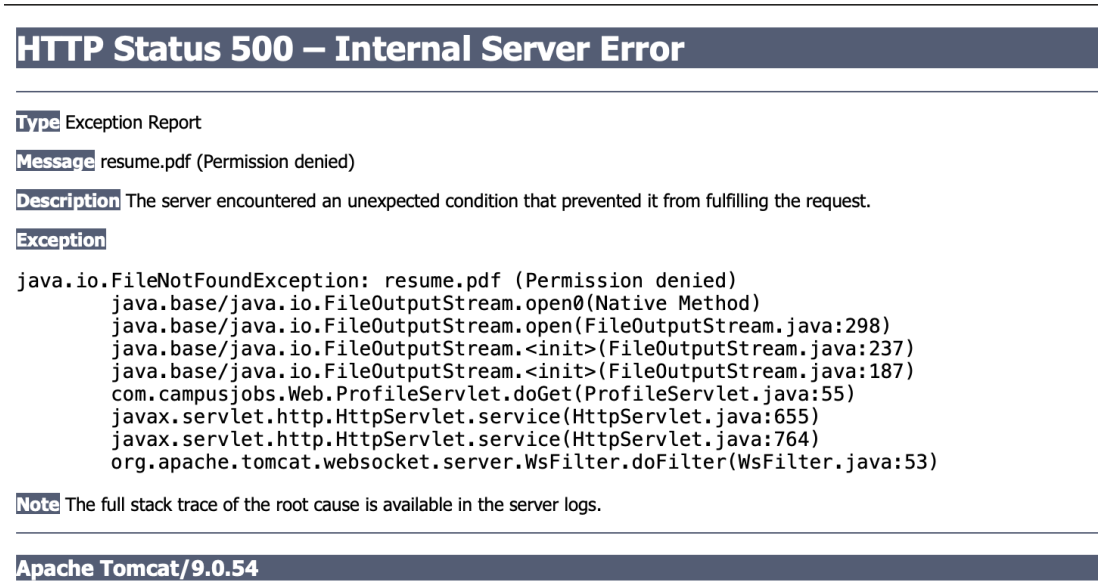


Figure 7.2: Permission denied error from resume download



This beta testing has a few main goals with regards to our project. Primarily, we needed to ensure that our service is intuitive and easy to use. In addition to testing our service ourselves, we hope to gather graduate students and professors to complete some steps of our service, such as account creation and job application/posting. In order to guide these participants through testing, we planned to incorporate “think aloud” testing, where the users would speak out loud their thought process and the steps that they are going to take. This approach would give us a better understanding of the user’s thought process and the possible UI flows in the application. With this data, we can make necessary adjustments to make our site easier to use and flow better.

8 User Manual

In order for a user to access the website, they simply need to copy the campusjobs.cs.vt.edu link into a browser of their choosing. They will be met with the welcome screen shown in Figure 6.1. If they do not already have an account, they will click the “Sign up” button as shown in Figure 8.1 to be redirected to the sign up page shown in Figure 6.2 where they can enter their information as well as select which role they are signing up for. The roles include student and employer. If the user already has an account, they can sign in as normal using their email/username and password in the fields shown in Figure 8.2 and clicking the “Sign in” button shown in Figure 8.1.

Figure 8.1. The sign up and sign in buttons.

Figure 8.2. Login information location.

Once logged in to the service, the user will be met with a home page depending on whether they are a student or an employer shown in Figures 6.3 and 6.4. Both userbases have access to their profile page shown in Figure 8.1, which they can navigate to using the top navigation bar shown in Figures 8.3 and 8.4. This page will display their submitted personal information. Students will have access to a job search page shown in Figure 6.4 that can be used to search for job postings. They will also have access to a page to track completed job applications. Employers will have access to a page for posting job opportunities, as well as a page that will show all posted jobs and associated applications. Users can log out of the application at any time by using the button in the top right corner.

Figure 8.3. Navigation bar in student view.

Figure 8.4. Navigation bar in employer view.

9 Developers Manual

Figure 9.1. UML of project class structure

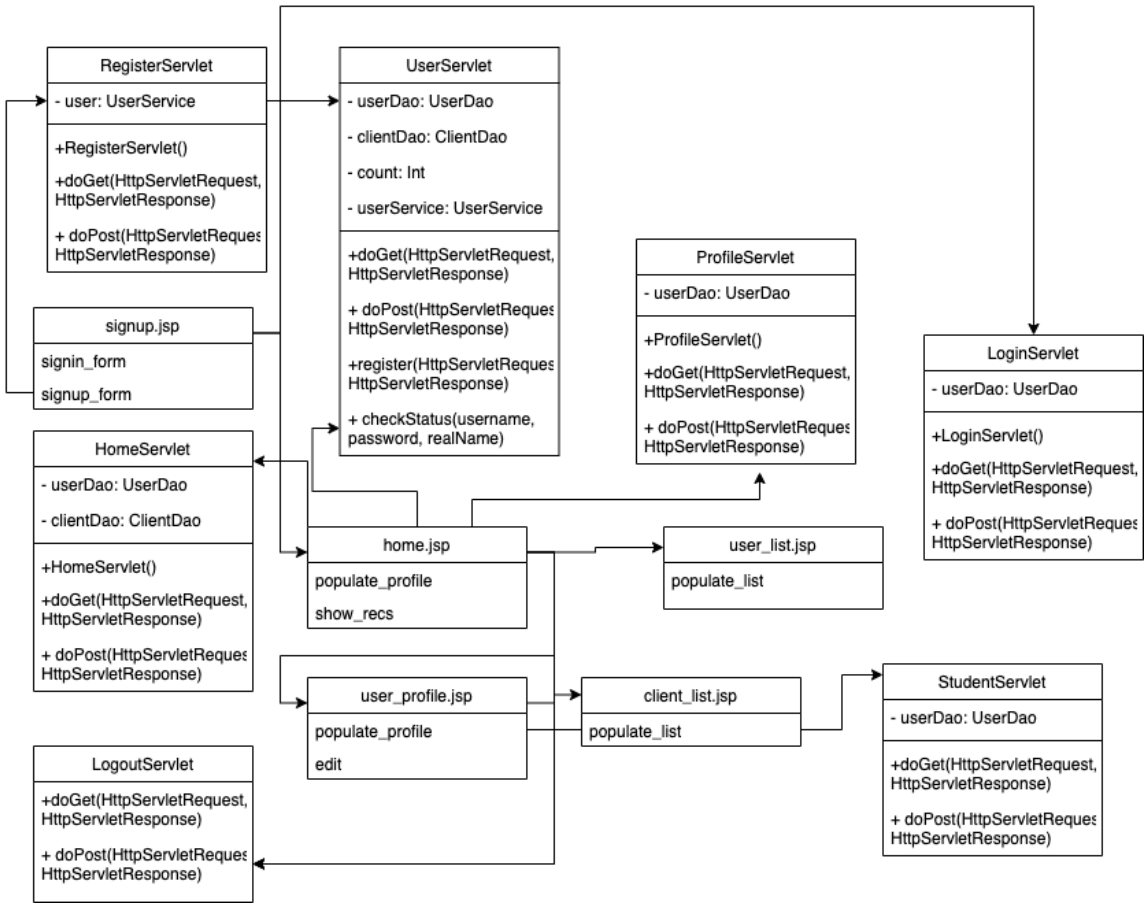


Figure 9.1 shows the UML diagram for the class structure of the project including class dependencies, functions and fields utilized. Any future implementations of the project will be structured around this class design.

To plan the user interface and flow for our service, we devised various medium-fidelity wireframes. These wireframes were created based on requirements outlined by our client, professors and graduate students interviewed prior to the start of the project implementation. Figures 9.2 and 9.3 showcase wireframes from the student and employer perspectives.

Figure 9.2 Medium-fidelity student wireframe

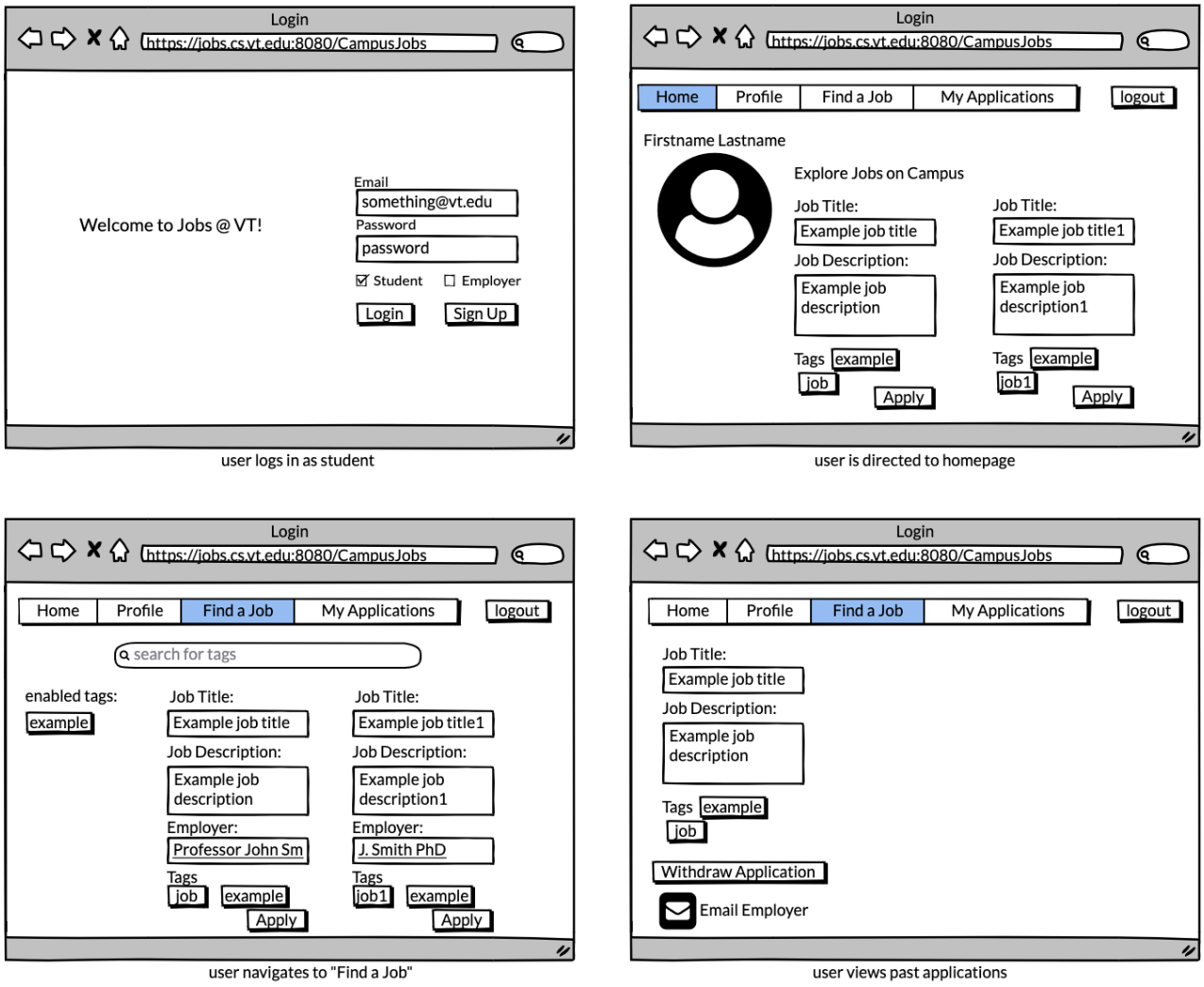
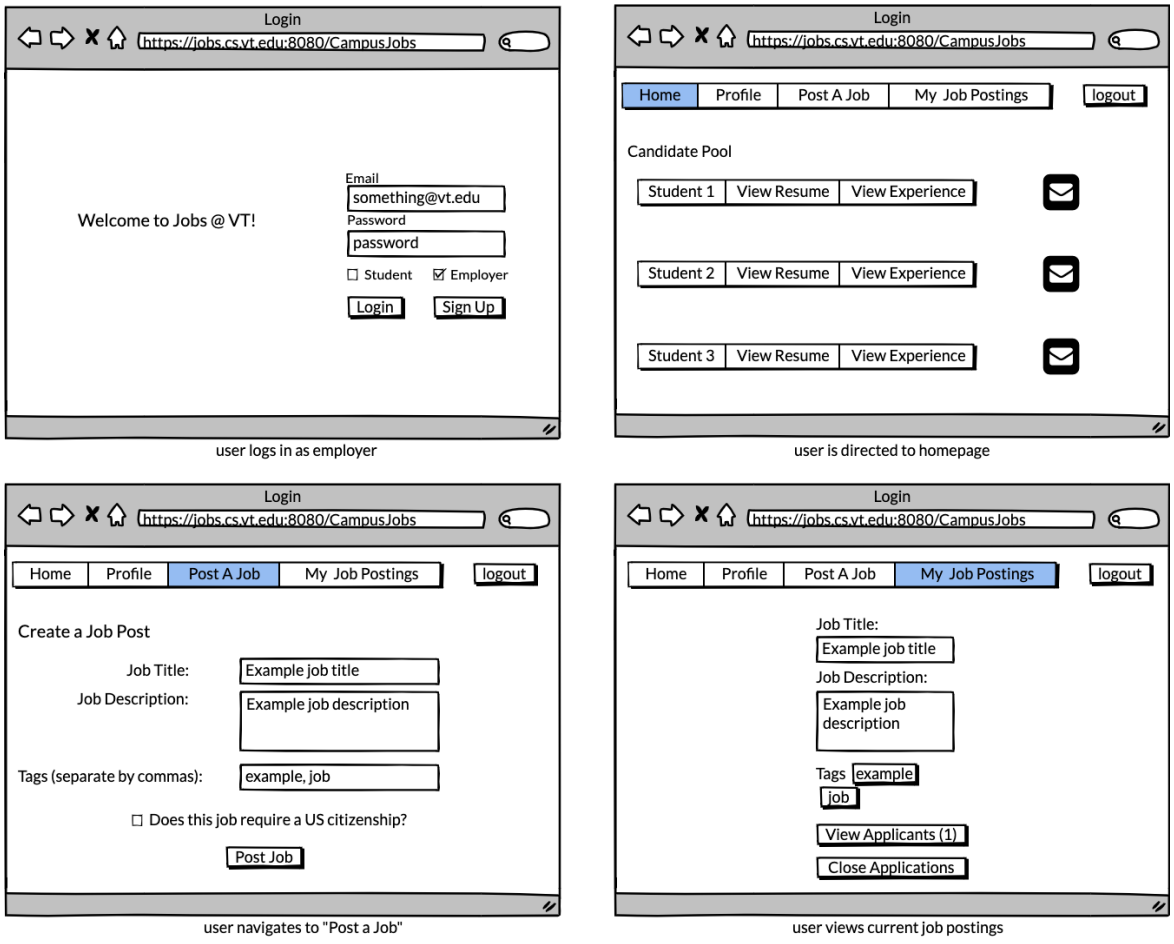


Figure 9.3 Medium-fidelity employer wireframe



Below are different actions and tips for working with our website and with the Tomcat server.

- **the path of the Tomcat folder in the VM:**

  /usr/share/tomcat

- **to restart the tomcat server:**

  service tomcat restart

- **to make everything reload and restart the server:**

  sudo systemctl daemon-reload

  sudo systemctl start tomcat

sudo systemctl enable tomcat

- **to check the status of tomcat:**
    - Tomcat will often behave unexpectedly. This is the first place to look if the website is not functioning properly

systemctl status tomcat

- **our website link:**
    - Website link is formatted as http://<vm-name>:<port-number>/<.war file>

http://jobs.cs.vt.edu:8080/CampusJobs

- **move the war file to the webapps:**
    - This allows Tomcat to deploy whatever pages are in your .war file to a functioning web application.

sudo mv CampusJobs.war /usr/share/tomcat/webapps

In order to demo the web application before deploying it to the VM, we were able to deploy it locally first through IntelliJ. IntelliJ Ultimate, which offers students a free subscription, provides support for deploying .war files on a local Tomcat server [3]. However, future developers should know that there is a strong possibility that the web application will behave differently when deployed versus on a local machine, so we strongly recommend that developers continuously test their application on the VM when making large changes to code.

10 Lessons Learned

10.1 Timeline/Schedule

Table 10.1. A table showing the team's schedule for the semester

Date	Milestone	Date	Milestone
9/10	● Complete OKRs and finalize milestones for the project ● Get approval from client on our milestones and team roles	9/17	● Complete questionnaires for potential employers and employees ● Have a basic outline for our website ● Acquire a VM for the website

10/1	• Complete interviews with potential employers • Complete interviews with graduate students • Construct a schema for the database -- does not need to be complete	10/15	• Have a concrete plan for a matching algorithm (ElasticSearch vs. database) • Create a browsing functionality for the website • Begin implementing the employer perspective for the website (25% completed) • Implement authentication for employer login
11/5	• Begin implementing the employee perspective for the website (25% completed) ○ Implement authentication for employee login • Complete employer perspective for the website (100%) • Complete implementation for the matching algorithm • Acquire candidates for beta testing • Begin quality assurance	11/19	• Complete employee perspective implementation for the website (100%) • Begin beta testing
11/26	• Perform maintenance and fix any bugs discovered during beta testing • Have a completed product	12/3	• Finish final presentation • Continue performing bug fixes

10.2 Problems

One of the primary issues our team ran into was designing a system for our client. None of the team members had any experience with building a website from the ground up like this. Another obstacle we encountered was a lack of knowledge when it came to what our client wanted and what our target audience wants. Although our team is composed of students, the graduate student experience is very different from the undergraduate experience, so we still had a gap in knowledge when it came to graduate school specifics. Finally, our largest problem has been in keeping ourselves on top of the development of a project this size over the course of a semester when everybody has such hectic schedules.

10.3 Solutions

We were able to tackle the first problem, lack of experience with website development, by using the

internet and educating ourselves on the process. Using documentation for various resources such as Tomcat, MySQL, Java, and HTML/CSS allowed us to easily apply each tool to create a complete product [1][2][4][9][11]. Although we have had setbacks with failed initial designs, we eventually came to a design that the team was happy with. We also were connected with a team of graduate students from ODU who helped us learn from their experience with an ElasticSearch website by giving us access to their website repo and explaining subjects clearly. Although we eventually decided against using elastic search, the discussion was very beneficial in brainstorming our design. In order to make up for the lack of knowledge about the graduate student experience, we decided to interview graduate students and professors who are looking for graduate student assistance to see what their ideal experience would be when it came to using our provided service. We also have maintained bi-weekly meetings with our client, Trey Mayo, to provide him updates to ensure our visions are still aligned. Finally, our team has used Ally and daily check-ins to make sure that everyone is on track. This way, everyone can work on their own time, but each member is still held accountable for their portion of the project.

10.4 Graduate school familiarity

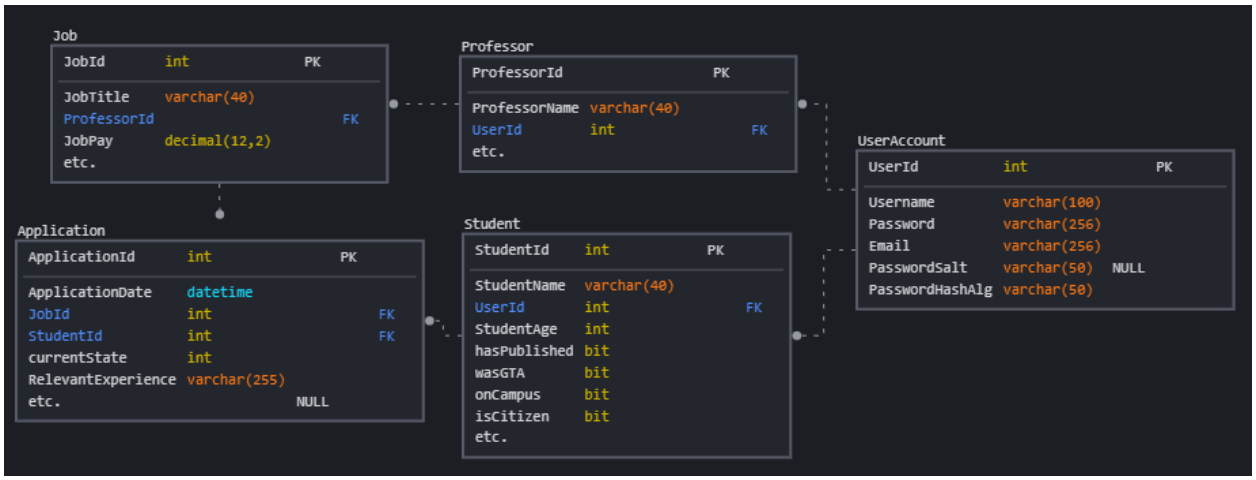
Before this semester, none of the members of the group had much familiarity or understanding of employment and projects within graduate school, especially at Virginia Tech. Through the various conversations that we had with graduate students and our client Mr. Mayo, we learned the ins and outs of graduate school and how vital and common it is for graduate students to be employed. Moreover, even though graduate students are students just like undergraduates, both groups have different college experiences. Throughout the design and implementation process, understanding these differences and similarities allowed us to better cater to our target audience.

11 Future work

11.1 Update database

At the moment, we currently have implemented a very rudimentary database design with some information for our users. Our database can be improved upon to allow us to store all necessary information, as well as connect any associated tables. Our current database mostly just contains information on students and professors. The proposed database design is as follows:

Figure 11.1 Schema for updated database layout.



As can be seen in the screenshot, the database would consist of five tables: UserAccount, Professor, Student, Application, and Job. UserAccount will contain login information for a user as well as a hashed password to keep information secure. Professor and Student tables will keep general information about each as well as a reference to their account information stored in UserAccount. The Job table would keep job specific information as well as a reference to the professor that is posting it. Finally, the Application table would track all applications, as well as an integer representation of their current state: active, accepted, rejected, or withdrawn. This design would allow developers to track all necessary information to enhance the user experience.

11.2 Enhance website features

Our web pages are still in the early stages, and have multiple areas for improvement that we can iron out. Maintaining a standardized design for each page so that it is both appealing to the eye and easy to follow is very important. We will also need to add all necessary links and information, such as contact information and redirection to any resources the users may need.

11.3 Expand matching algorithm

Once the database implementation is updated and our website has all the necessary supporting features, the implementation of the matching algorithm can be improved upon. This would not only require some function to use our data to match between jobs and students, but also finalized web pages to display the matches for both students and professors alike. The current matching algorithm implemented is very rudimentary as we did not make enough progress to implement a complete algorithm.

11.4 Patch security issues

What is probably the biggest weakness in our website is security. Currently, we have no measures in place to prevent DOS attacks nor do we hash user information (besides user password), which may leave users

vulnerable in the case of a data leak. While Tomcat provides basic security measures, Tomcat's documentation mentions that some versions hold memory leaks, openings for DOS attacks, and the existence of opportunities for request smuggling [10]. We recommend that future developers keep the Tomcat server up to date and to keep updated on Tomcat documentation.

11.5 Complete Beta Testing

To ensure that our site is secure and that user data is not accessible, we plan to test common cybersecurity flaws with fabricated data. This will help to make sure that users can trust our site with their identifiable information. After the addition of the Virginia Tech CAS login, we also plan to test that this service works appropriately with our application.

12 Acknowledgements

Dr. Edward Fox, VT Professor of CS 4624 Capstone

contact: fox@vt.edu

Trey Mayo, Director of CS Graduate Programs

contact: treymayo@vt.edu

Brian Mayer, VT Research Associate

contact: bmayer@vt.edu

Winston Shields, ODU Graduate Student

contact: wshie002@odu.edu

Graduate student interviewees

13 References

[1] Mozilla. *CSS: Cascading style sheets*. MDN. 2021. Retrieved December 9, 2021, from https://developer.mozilla.org/en-US/docs/Web/CSS.

[2] Mozilla. *HTML: Hypertext markup language*. 2021. Retrieved December 9, 2021, from https://developer.mozilla.org/en-US/docs/Web/HTML.

[3] JetBrains. *IntelliJ IDEA: The capable & ergonomic Java IDE by JetBrains*. JetBrains. 2021. Retrieved December 12, 2021, from https://www.jetbrains.com/idea/.

[4] Oracle. *Java documentation - get started*. Oracle Help Center. (2021, April 23). Retrieved December 9, 2021, from https://docs.oracle.com/en/java/.

[5] Java Database Connectivity. Java JDBC API. 2021. Retrieved December 9, 2021, from https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/.

[6] Netscape. *JavaScript*. MDN. 2021. Retrieved December 9, 2021, from https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[7] LinkedIn. *LinkedIn job search: Find us jobs, internships, jobs near me*. 2021. Retrieved December 9, 2021, from https://www.linkedin.com/jobs.

[8] Apache Software Foundation. *Maven Repository*: Com.alibaba " druid " 1.1.9. 2021. Retrieved from https://mvnrepository.com/artifact/com.alibaba/druid/1.1.9

[9] MySQL. *MySQL documentation*. MySQL. 2021. Retrieved December 9, 2021, from https://dev.mysql.com/doc/.

[10] Apache. *Apache Tomcat*®. Apache Tomcat® - Welcome! 2021. Retrieved December 12, 2021, from https://tomcat.apache.org/.

[11] Virginia Tech. (2021, November 11). Graduate Student Enrollment. Graduate School Reports. 2021. Retrieved November 11, 2021, from https://graduateschool.vt.edu/numbers/gradenroll.html.