UR Connect

Connect to Opportunity

DylanYamaguchi-Kuan
200365821 - yamagudy@uregina.ca

Logan Hesse
200435931 - lah272@uregina.ca

Zeeshan Nasir
200400229 - znu546@uregina.ca

Kaleb Saggu
200433745 -ksj214@uregina.ca

Ilias Echaouni Benabdallah
200395900- ieo347@uregina

Contents

List of Figures

1: Abstract

UR Connect: Bridging Students and Employers at the University of Regina

The University of Regina faces a critical challenge in facilitating connections between its students and local employers, hindering both parties from fully leveraging opportunities within the community. To address this issue, UR Connect is introduced as a comprehensive solution to facilitate interactions between students and employers within the University. UR Connect offers a user-friendly platform designed to help students to showcase their skills and achievements to local companies by enhancing their chances of securing fitting opportunities. The platform caters to students at various stages of their academic and professional journey. For students still studying, UR Connect offers a space to discover internship opportunities and professional growth, while for graduate students preparing to enter the professional world, it becomes a comprehensive resource offering a job board and personalized connections. Understanding the unique needs of students at different academic stages, UR Connect provides tailored opportunities. Advanced search functionality enables users to find specific profiles and job listings based on various criteria, enhancing the overall user experience. Additionally, can navigate a diverse pool of student talent with ease. By setting up detailed company profiles and posting targeted job listings, employers can attract students who align with their organizational culture and skill requirements.

2: Introduction and Problem Statement

In today's competitive job market, students often struggle to showcase their skills and secure fitting opportunities, while employers face challenges in identifying and connecting with talented individuals. Recognizing these obstacles, we have worked on developing UR Connect, a user-friendly platform designed to streamline interactions between students and employers within the university community. This solution aims to bridge the gap between local students seeking opportunities for professional growth and local employers in need of skilled individuals, ultimately fostering a more seamless and effective recruitment process. The creation of UR Connect stems from a pressing need within the University of Regina community. Students encounter difficulties in effectively presenting their achievements while studying, hindering their ability to secure meaningful opportunities during their academic journey and upon graduation. Simultaneously, local employers struggle to identify and connect with talented individuals with the skills and attributes they seek. This disconnect results in missed opportunities for both students and employers, highlighting the necessity for a platform like UR Connect to facilitate personalized interactions and streamline the recruitment process within the University of Regina ecosystem. UR Connect also aims to provide current students seeking internships with better opportunities by focusing solely on internship positions. This helps students to efficiently explore and apply to roles that align with their academic pursuits and career aspirations, all within the supportive environment of the University of Regina’s network.

3: Application Benefits

- **Enhanced Opportunities for Student**
  - Comprehensive profile creation allows students to showcase their education, skills, and work/volunteer experience.
  - Access to a customizable job board with filters based on industry, location, and job type increases the likelihood of finding fitting opportunities.
- **Streamlined Recruitment Process for Employers**
  - Comprehensive employer profiles enable companies to showcase their values, culture, and job opportunities, attracting candidates who align with their organizational ethos.
  - Access to a rich pool of local talent within the University of Regina community simplifies and expedites the recruitment process.
  - Customizable job board allows employers to post detailed job openings and engage with potential candidates efficiently.
- **Internship Focus for Current Students**
  - Centralized Internship Listings: Provides a one-stop shop for internship opportunities, reducing the need to visit multiple websites.
  - Academic Integration: Offers internships that align with students' academic schedules and career goals.

- **Strengthened University-Employer Connections**:
    - UR Connect serves as a centralized platform for fostering connections between students and local employers, strengthening ties within the University of Regina community.
    - Enhanced visibility for local companies through comprehensive employer profiles and job postings fosters greater engagement and collaboration with the university community.
    - Facilitates the exchange of opportunities, knowledge, and resources between students and employers, enriching the overall educational and professional experience for all stakeholders.

4: Requirements Elicitation & Specification

4.1 Functional Requirements

1. **<u>Functional User Requirements for Students</u>:**
   - **User Registration and Authentication**
     - Students should be able to register for an account on UR Connect using their University of Regina credentials or personal email.
     - The system should verify student identities and authenticate them securely during the login process.
   - **Profile Creation and Management**
     - Students should be able to create profiles highlighting their education, skills, work/volunteer experience, achievements, and career objectives.
     - They should have the ability to upload documents or multimedia content to enhance their profiles, such as resumes, and portfolios.
     - Students should be able to edit and update their profiles as needed, including adding or removing information.
   - **Job Search and Application**
     - The platform should feature a customizable job board where students can browse and search for job openings based on industry, location, job type, and other filters.
     - Students should be able to apply to job listings directly through the platform, with options to attach resumes or other relevant documents.

2. **<u>Functional User Requirements for Employers:</u>**

- **User Registration and Authentication**
  - Employers should be able to register for an account on UR Connect using their company email or other authorized credentials.
  - The system should verify employer identities and authenticate them securely during the login process.
- **Company Profile Creation and Management**
  - Employers should have the ability to create comprehensive profiles showcasing their company values, culture, job opportunities, and hiring preferences.
  - They should be able to upload company logos, descriptions, and other relevant information to enhance their profiles.
  - Employers should be able to edit and update their profiles as needed, including adding or removing job listings.
- **Job Posting and Management**
  - The platform should allow employers to post job listings with detailed descriptions, including job requirements, responsibilities, and application instructions.
  - Employers should have the ability to manage their job listings, including editing, deleting, or renewing listings as needed.

4.2 Use Case Diagrams

- **Student User**:

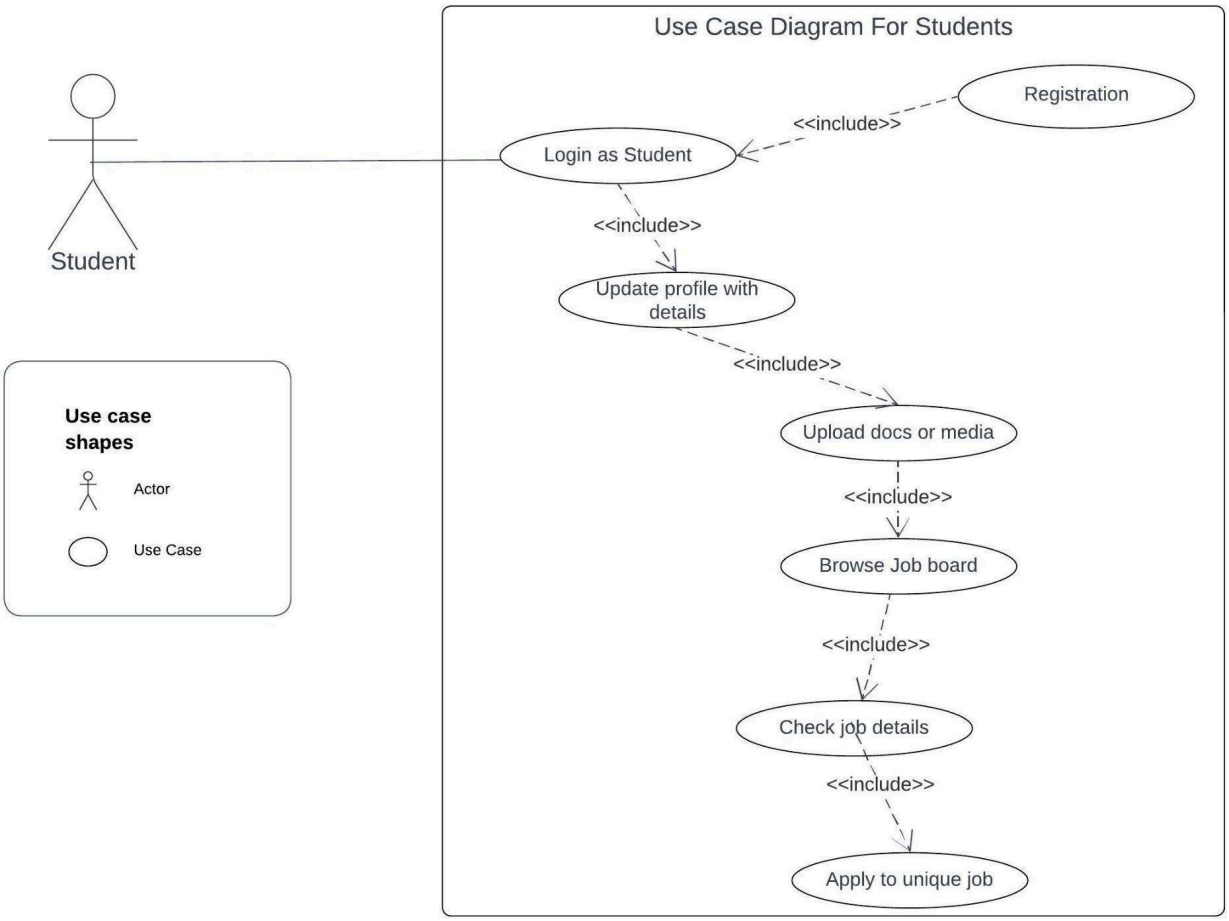

Figure 1: Use Case Diagram for Students

- **Employer**:

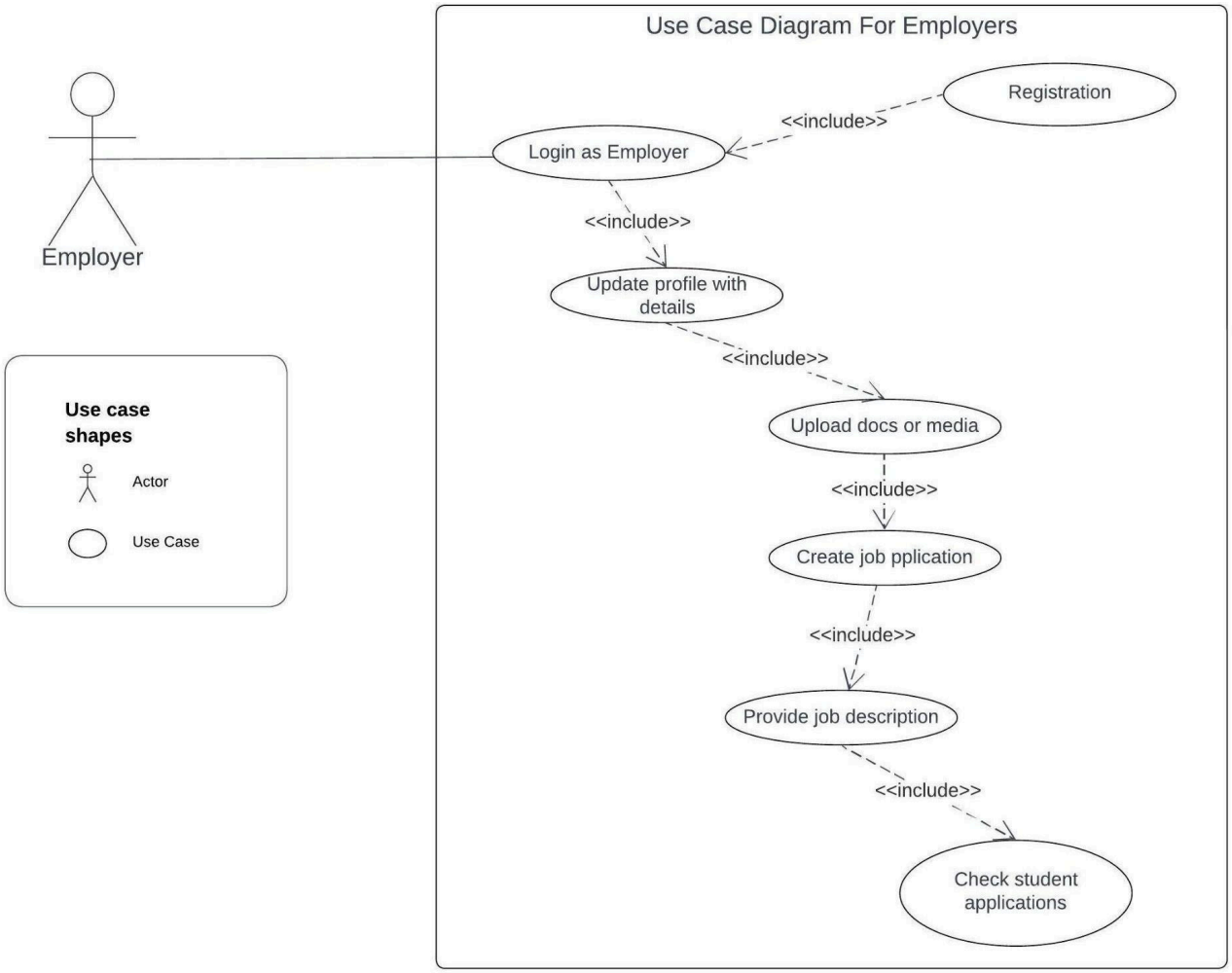

Figure 2: Use Case Diagram for Employers

4.3 Activity Diagrams

- **Student User:**

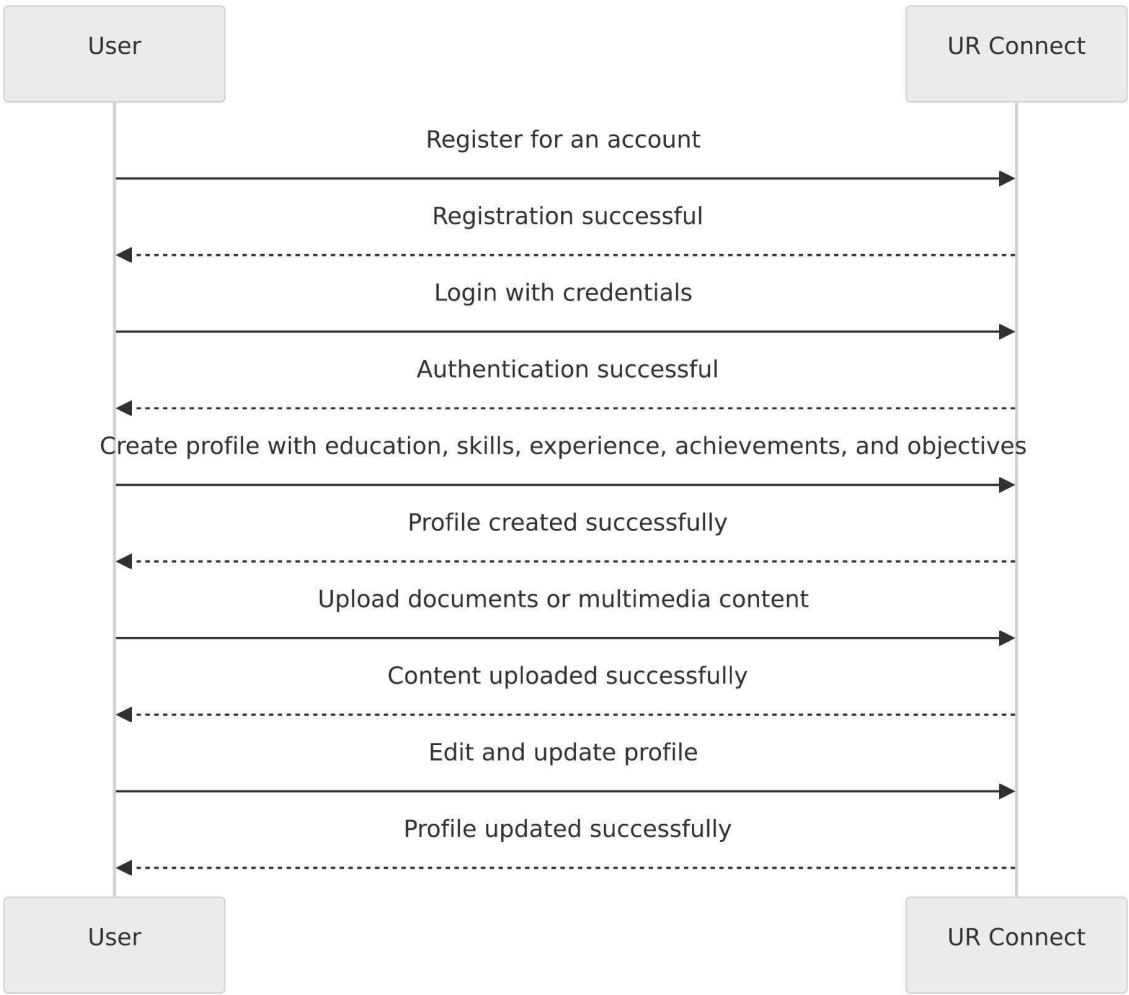

Figure 3: Activity Diagram 1

- **Employer**:

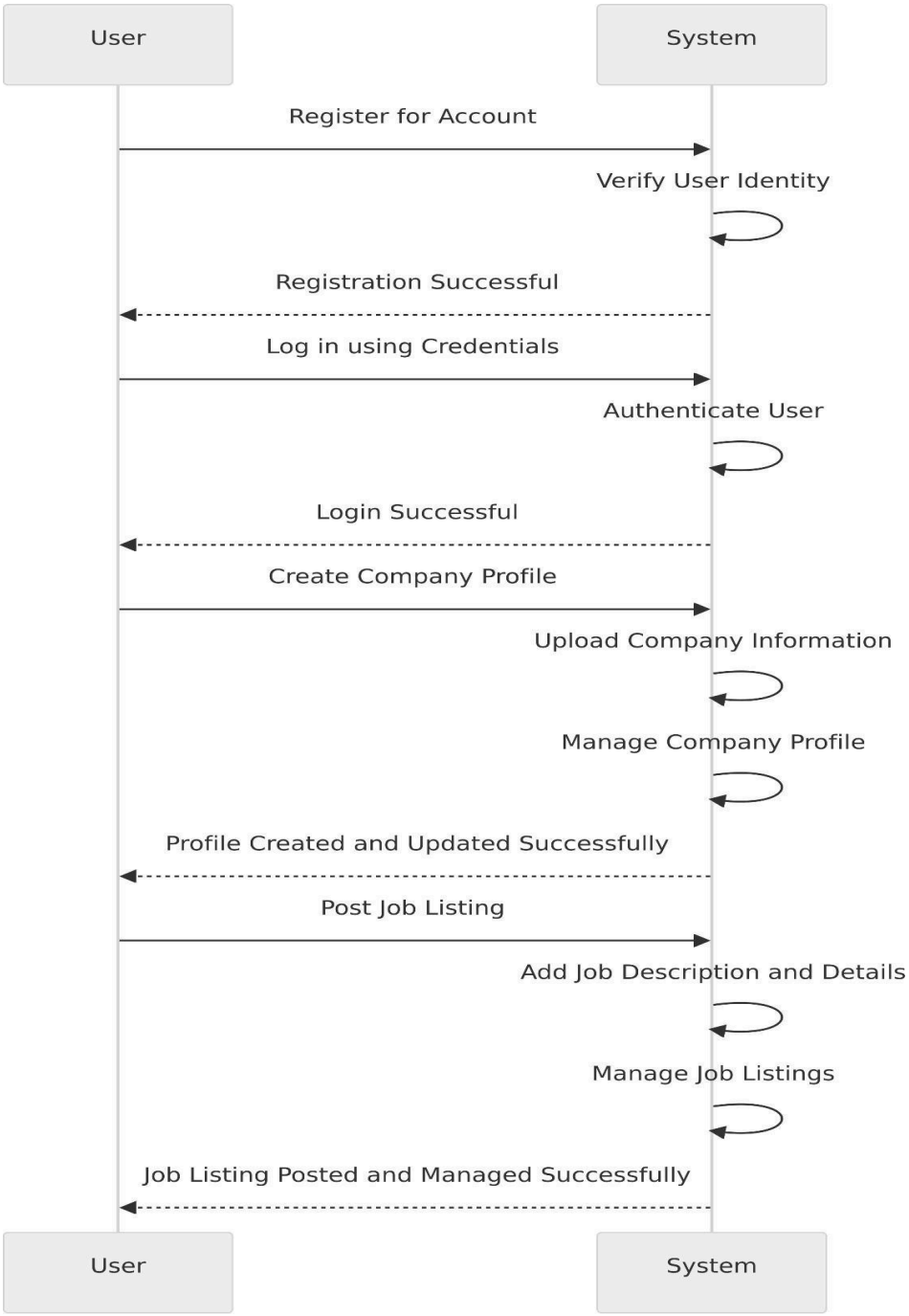

Figure 4: Activity Diagram 2

4.4 Software Qualities

1. <u>Correctness</u>

    a. **Student**

        i. **Validation during Registration**: When students register for an account, the system validates their information, such as email address, and student ID to ensure accuracy and prevent errors.

        ii. **Profile Completion**: The system prompts students to complete essential fields in their profiles, such as education, skills, and experience, ensuring that their profiles are comprehensive and accurate.

        iii. **Feedback and Correction**: Users can edit inaccuracies or inconsistencies in their profiles.

    b. **Employer**

        i. **Validation during Account Creation**: During the registration process, the system validates employer information, such as company email address and contact details, to ensure accuracy and prevent errors in account setup.

        ii. **Job Listing Accuracy**: Employers are prompted to enter essential details when posting job listings, such as job titles, descriptions, requirements, and application instructions, ensuring that job postings are comprehensive and accurate.

        iii. **Feedback and Correction**: Employers can review and edit their job listings for inaccuracies or updates after posting them, ensuring that the information presented to students remains accurate and up-to-date.

2. <u>Time efficiency</u>

   a. **Student**

      i. **Quick Registration Process**: Implement a streamlined registration process with minimal steps and validation requirements to allow students to create accounts quickly and efficiently.

      ii. **Time-Saving Filters**: Offer search filter options for job listings, enabling students to quickly narrow down their options based on criteria such as location, industry, and job type.

      iii. **Real-Time Notification**: Provide real-time notifications for completed applications allowing students to stay informed.

   b. **Employer**

      i. **Efficient Account Creation**: Offer a streamlined account creation process for employers, with minimal form fields and validation requirements, allowing them to create accounts quickly and start posting job listings immediately.

      ii. **Quick Job Posting**: Implement a quick job posting method that allows employers to speed up the process of creating a job listing with minimal input required.

      iii. **Application Access**: Employers are provided with a clear and organized list of job applications for each job listing. This list displays essential details such as applicant names, contact information, qualifications, and application status. Employers can easily scan through the list and access additional details without leaving the interface. This streamlined approach facilitates efficient candidate evaluation and selection, optimizing the hiring process.

3. <u>**Robustness**</u>

   a. **Student**

      i. **Data Privacy:** Ensure that student data, including personal information, academic records, and employment history, is protected by privacy regulations and best practices.

      ii. **System Security:** Protect student accounts and sensitive information from security threats such as unauthorized access. Implement password hashing, and security protocols to prevent unauthorized access and maintain system security.

      iii. **Integrity:** Ensure the integrity of interactions within the UR Connect platform, such as applications, job submissions, and profile updates.

      iv. **Compatibility:** Ensure compatibility with a wide range of devices, browsers, and operating systems to accommodate diverse student preferences and technological environments.

   b. **Employer**

      i. **Data Integrity:** Ensure that employer data, including company profiles, job listings, and candidate information, is securely stored and protected against unauthorized access, modification, or loss.

      ii. **Error Handling:** Implement comprehensive error handling mechanisms to gracefully handle unexpected errors, exceptions, and edge cases. Provide informative error messages to assist employers.

      iii. **Scalability:** Ensure that the UR Connect platform is scalable and capable of handling increasing numbers of employers, job listings, and candidate applications over time.

5: Top-level & low-level software design

5.1 MVC Architecture & Benefits

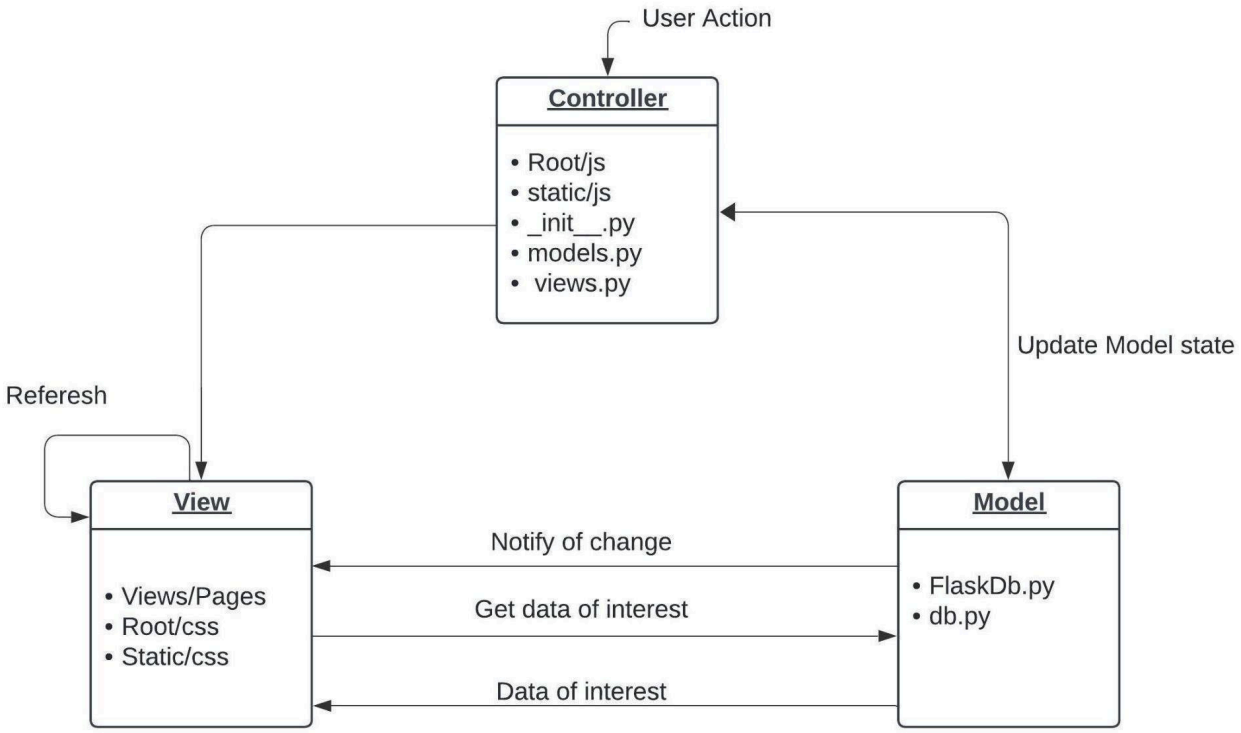

Figure 5: MVC Architecture

The UR Connect application was built using Flash stack. The Model-View-Controller (MVC) architecture has been implemented using the Flash stack as follows:

- **Model**:

  In UR Connect, the model component serves as the backbone for data management and retrieval. Utilizing Flask-SQLAlchemy, a Python ORM (Object-Relational Mapping) library for Flask, the model ensures efficient handling and manipulation of data entities crucial for the application's functionality. Files such as models.py and FlaskDb.py under the app directory encapsulate the data models, defining the structure and relationships among entities like users, job listings, and resumes. Additionally, db.py facilitates database operations, ensuring data integrity and adherence to the application's business logic. Validation and schema enforcement are implemented in models.py to maintain consistency and reliability within the database.

- **View**:

  the view component presents a dynamic and intuitive user interface, enhancing the user experience. Leveraging HTML templates, the view layer offers visually appealing and interactive pages accessible to users. Files within the templates directory, such as createEmployer.html, home.html, and login.html, define the various views accessible to users, ranging from profile creation to job listings browsing. CSS files under static/css fine-tune the presentation, ensuring consistency and aesthetic appeal across the application. JavaScript files under static/js facilitate client-side interactions, handling form submissions, and enhancing user interactivity for a seamless experience.

- **Controller**:

  The controller component in UR Connect orchestrates the flow of data and requests between the client-side and the server-side, enforcing business logic and managing endpoints. Utilizing Flask, a lightweight web application framework for Python,

the controller layer handles incoming HTTP requests and dispatches them to the appropriate handlers. Files such as views.py under the app directory define the request handlers, processing data from the model layer and rendering appropriate responses. The controller also manages routing through Flask's routing mechanism, ensuring requests are directed to the correct endpoints for processing. Additionally, JavaScript files like postJob.js handle client-side interactions, complementing the server-side logic and enhancing user experience through asynchronous requests.

UR Connect adopts the MVC architecture, aligning with the principles of modular design and separation of concerns. By segregating the application into distinct model, view, and controller components, UR Connect achieves scalability, maintainability, and extensibility, facilitating interactions between students and employers within the ecosystem.

The following are the benefits of MVC Architecture for UR Connect:

- **Separation of Concerns**: MVC architecture facilitates clear separation of concerns between different components of the application, enhancing maintainability and scalability. The Flask stack's directory structure aligns with this separation, making it easier to manage and update individual components without affecting others. For instance, modifying the database schema does not impact the view or controller code.
- **Scalability**: UR Connect can easily scale to handle a growing user base and increased traffic by adding more instances of the Flask application. Flask's lightweight nature and support for WSGI servers allow for efficient scaling without compromising performance. The modular nature of MVC architecture enables seamless integration of additional components to handle increased workload and user interactions.

- **Reusability**: by segregating the application into model, view, and controller components, UR Connect promotes code reusability across different parts of the application. Common functionalities, such as user authentication or data validation, can be encapsulated within reusable modules, reducing development time and effort. Flask's support for modular development and Flask extensions further enhances code reusability, allowing developers to leverage existing solutions and libraries.
- **Flexibility and Extensibility**: the Flask stack offers flexibility and extensibility, allowing UR Connect to adapt to evolving requirements and integrate new features seamlessly. Flask's modular design and support for third-party extensions enable easy integration of additional functionality, such as authentication mechanisms or RESTful APIs. MVC architecture's decoupled components enable independent development and testing of new features, minimizing disruption to existing functionality.
- **Maintainability and Testing**: the MVC architecture promotes maintainability by providing a structured framework for organizing code and enforcing separation of concerns. Flask's syntax makes codebase maintenance straightforward, enabling developers to quickly understand and modify existing code. The modular nature of MVC architecture facilitates comprehensive testing of individual components, ensuring the robustness and reliability of the UR Connect application.

5.2 Design Patterns

1. **Observer Design Pattern:**

   In the UR Connect application, we employ the Observer design pattern to enable multiple components to receive updates when changes occur in specific entities, allowing for responsiveness within the system. This design pattern plays a crucial role in various aspects of the application, such as user interactions, job postings, and profile updates. Here's how the Observer design pattern is implemented in UR Connect:

   a. **Subject**: In UR Connect, the subjects represent the objects being observed, which include entities such as Students, Employers, and Jobs. For example, the Students, Employers, and Jobs classes define the subjects being observed in the application. These classes correspond to database tables and encapsulate data related to students, employers, and job listings, respectively.

   b. **Observers:** Observers in UR Connect are components or entities interested in changes to the state of subjects. Examples of observers may include user interfaces, database operations, or other application components that react to changes in student profiles, job listings, or employer details.

   c. **Observer interface**: the interaction between subjects and observers is implicit in the routes and logic defined in the views.py file. Changes to subjects trigger corresponding actions or updates in observers through routes and database operations.

d. **Implementation Details**: The routes defined in views.py handle interactions between clients and the server, facilitating the observation of changes in subjects and triggering appropriate responses. For example, when a student registers (createStudent()), the corresponding observer updates the database with the new student's information. Similarly, when an employer logs in (emp_login()), the observer handles authentication and updates session data accordingly.

The class diagram for the Observer Design Pattern is as follows:

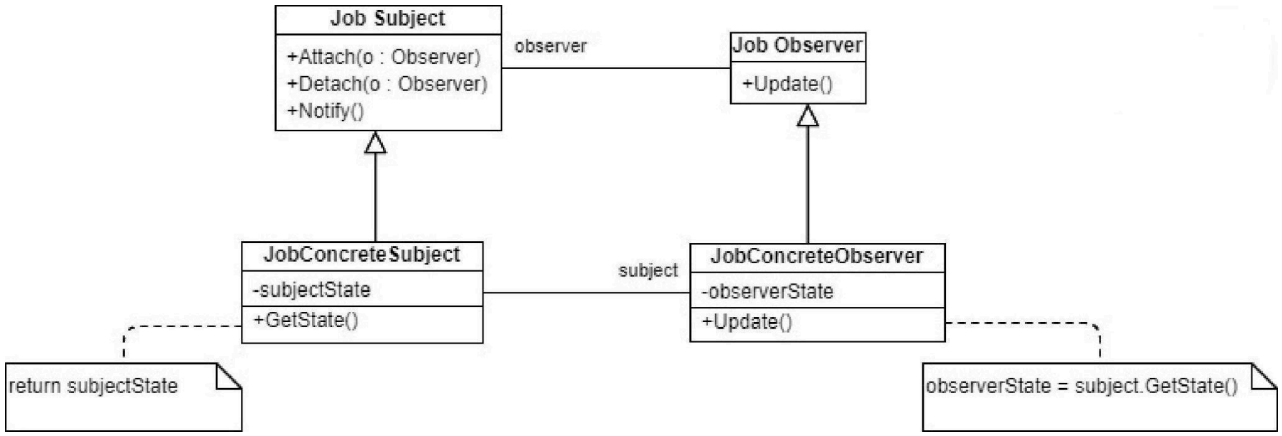

Figure 6: Class Diagram for Observer Pattern

Following is the list of attributes and prototypes along with the data types:

- Attributes:
    - observer: array
    - ContextProvider: observe
    - observerId: string
    - updatedJob: string
- Prototype Methods:
    - addObserver(observerId)
    - removeObserver(observerId)
    - notifyObserver()
    - notifyObserver(updatedJob, observerId)
    - getContext()

    - observerId: string
    - updatedJob: string
- Prototype Methods:
    - addObserver(observerId)
    - removeObserver(observerId)
    - notifyObserver()
    - notifyObserver(updatedJob, observerId)
    - getContext()

Following is the algorithm for the observer pattern's important method in Python:

```
//      method      to      add      observers schema. class
Schema:
    def init(self):
        self.observers = []

    def add_observer(self, observer_id):
        self.observers.append(observer_id)
# Example usage:
schema = Schema()
schema.add_observer(observer_id)
```

```
//      method      to        remove      observers schema . class
Schema:
    def init(self):
        self.observers = []


    def remove_observer(self, observer_id):
        if observer_id in self.observers:
            self.observers.remove(observer_id)


# Example usage:
schema = Schema()
schema.remove_observer(observer_id)

// method to notify all observers
  class Schema:
      def init(self):
          self.observers = []

      def notify_observers(self, updated_job):
          for observer_id in self.observers:
              observer = JobObserver() # create a new observer object
              observer.notify(updated_job, observer_id)


  class JobObserver:
      def notify(self, updated_job, observer_id):
          # Implement notify method logic here
          pass # Placeholder for implementation


  # Example usage:
  schema = Schema()job)
```

2. **Module Pattern and Event-driven Architecture:**

   The UR Connect web page does not use the Factory Design Pattern. Instead, our project uses the Module Pattern and Event-driven Architecture.

   - **Module Pattern**: Each JavaScript file in the project encapsulates its functionality within a module. For example, functions and variables are defined within a module scope using an Immediately Invoked Function Expression (IIFE) to avoid polluting the global namespace. This pattern helps in organizing code into logical units and provides encapsulation.
     - **Usability**: The Module Pattern is suitable for encapsulating individual functionalities or features of the job portal system. Each module can handle specific tasks such as form validation, user authentication, job listing management, etc.
     - **Benefits**:
       - **Encapsulation**: Each module encapsulates its logic, reducing the risk of naming conflicts and promoting code organization.
       - **Reusability**: Modules can be reused across different parts of the application, improving code efficiency.
       - **Maintainability**: Changes or updates to a module can be isolated without affecting other parts of the application.
       - **Example**: Modules like createEmployer, createStudent, login, etc., encapsulate form validation, authentication, and other related functionalities.

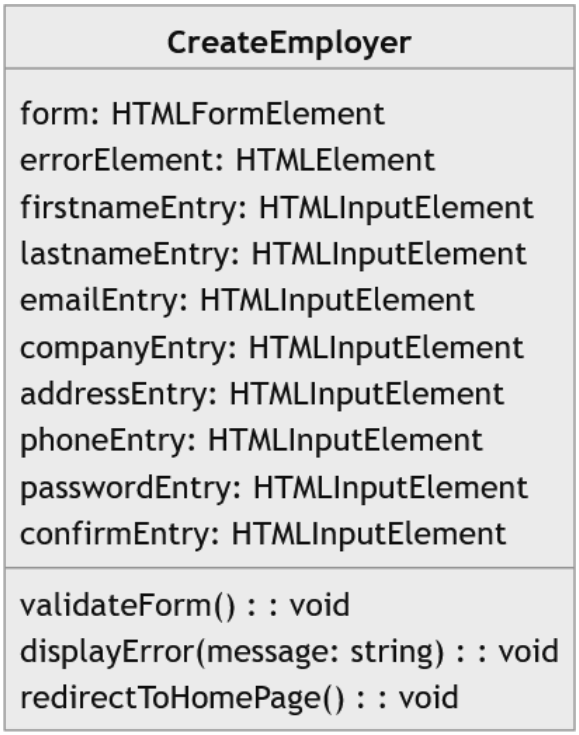

Figure 7: Class Diagram for Module Pattern for CreateEmployer Class

- **Attributes**:
  - form: HTMLElement - Represents the HTML form element with the id 'infoForm'.
  - errorElement: HTMLElement - Represents the HTML element with the id 'error'.
  - firstnameEntry: HTMLElement - Represents the HTML input element with the id 'firstnameEntry'.
  - lastnameEntry: HTMLElement - Represents the HTML input element with the id 'lastnameEntry'.
  - emailEntry: HTMLElement - Represents the HTML input element with the id 'emailEntry'.
  - companyEntry: HTMLElement - Represents the HTML input element with the id 'companyEntry'.

- addressEntry: HTMLElement - Represents the HTML input element with the id 'addressEntry'.
- phoneEntry: HTMLElement - Represents the HTML input element with the id 'phoneEntry'.
- passwordEntry: HTMLElement - Represents the HTML input element with the id 'passwordEntry'.
- confirmEntry: HTMLElement - Represents the HTML input element with the id 'confirmEntry'.
- Prototypes of Methods:
- addEventListener(event, callback): Adds an event listener to the form element.
- validateFirstName(): Validates the first name input field.
- validateLastName(): Validates the last name input field.
- validateEmail(): Validates the email input field.
- validateCompany(): Validates the company name input field.
- validateAddress(): Validates the company address input field.
- validatePhone(): Validates the phone number input field.
- validatePassword(): Validates the password input field.
- validateConfirmPassword(): Validates the confirm password input field.
- handleSubmit(e): Handles form submission event, performs validation and prevents submission if there are errors.
- redirectUser(): Redirects the user to the home page if there are no errors in the form submission.

- **Event-driven Architecture**: The project relies heavily on event listeners to handle user interactions and trigger specific actions accordingly. For instance, form submissions clicks on job postings, and login attempts are all handled using event listeners. This architecture decouples components, making the code more modular and easier to maintain.
    - **Usability**: The Event-driven Architecture is suitable for handling user interactions and asynchronous actions within the job portal system. Events such as form submissions, job postings clicked, etc., trigger actions or updates in response. This pattern promotes responsiveness and interactivity in the user interface.
    - Benefits:
        - **Asynchronicity**: Events and event handlers allow non-blocking execution of tasks, improving the responsiveness of the application.
        - **Scalability**: New features or components can be added without tightly coupling them to existing code, promoting scalability.
        - **Flexibility**: Components can react to events independently, providing flexibility in how the application responds to user actions.
        - **Example**: Files, like joblistingemployer.js and joblistingstudent.js handle events such as job postings, clicked to fetch data and update the UI dynamically.

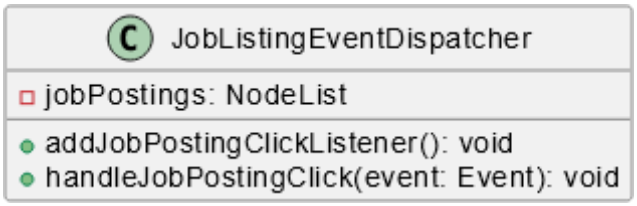

Figure 8: Event-driven Architecture for JobListingEventDispatcher class

- **Attributes**:
  - observers: Array - An array to store observer objects.
  - jobData: Object - Represents data related to a job listing.
- **Prototype Methods**:
  - attach(observer): Attaches an observer to the event dispatcher.
  - detach(observer): Detaches an observer from the event dispatcher.
  - notify(): Notifies all attached observers about changes in the job data.
  - getJobData(): Returns the current job data.
  - setJobData(data): Sets the job data to the provided

value. These attributes and prototype methods define the functionality of

the

JobListingEventDispatcher class. The observers array stores observer objects, while the jobData object holds information about job listings. The methods allow attaching and detaching observers, notifying observers of changes, and getting or setting the job data.

5.3 Class Diagram

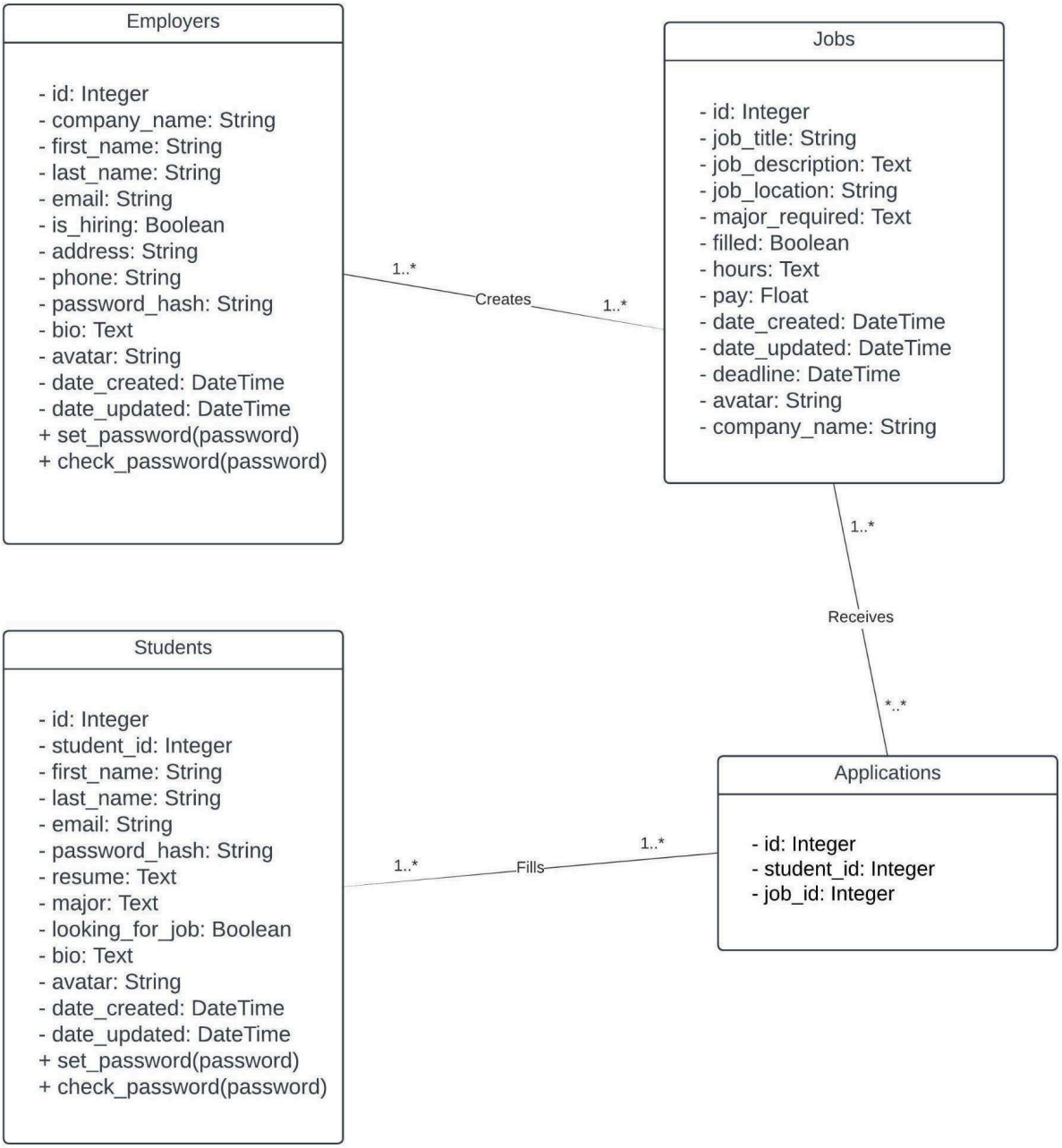

Figure 9: UR Connect Class Diagram

6: Software Construction

6.1 Struture of Code

- <u>Directory Structure:</u>

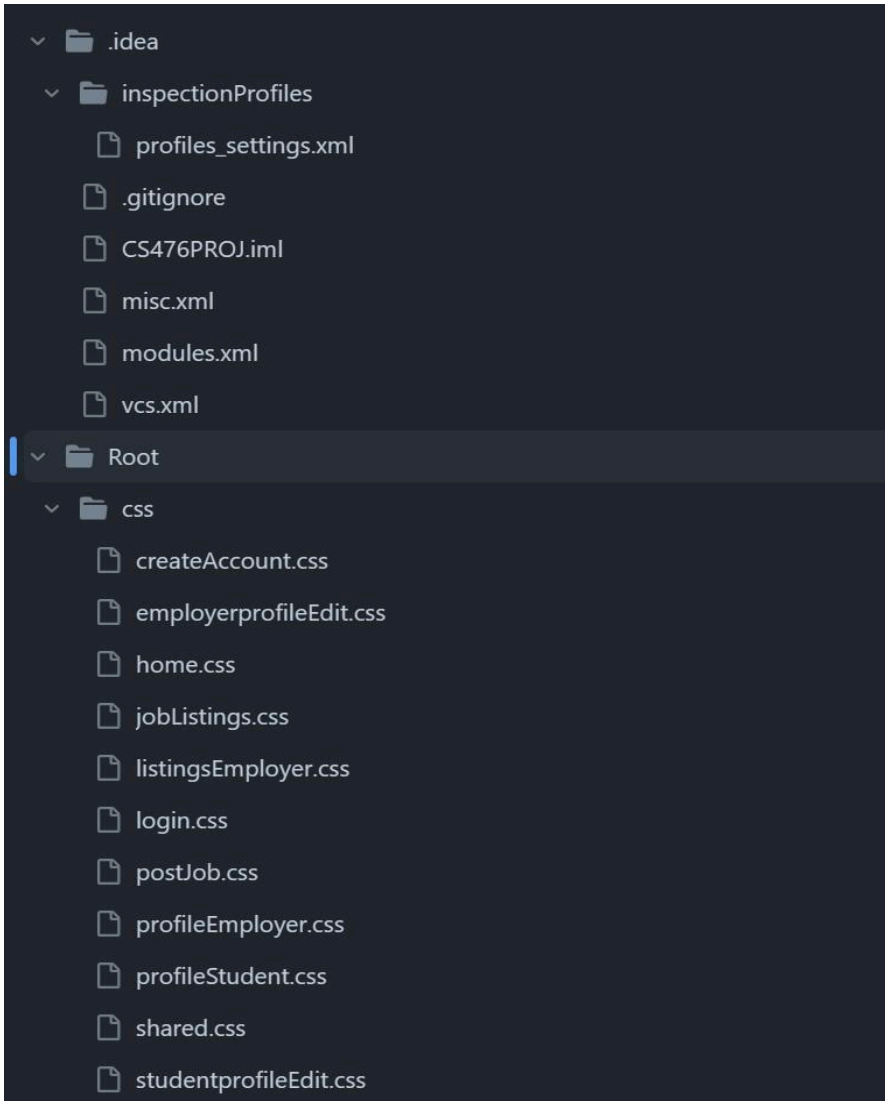

Figure 10: Structure Details 1

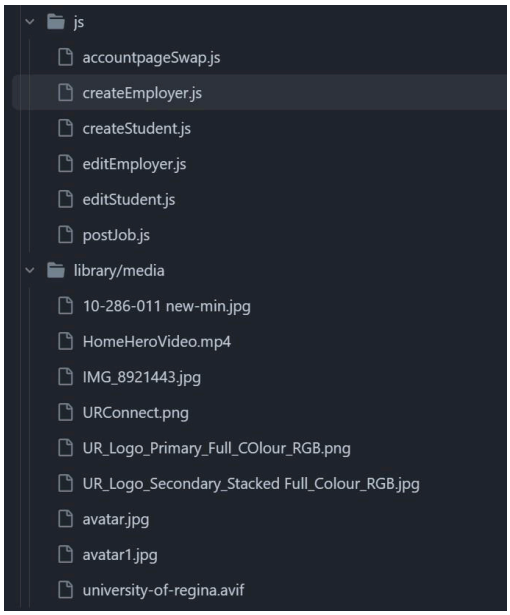

Figure 11: Structure Details 2

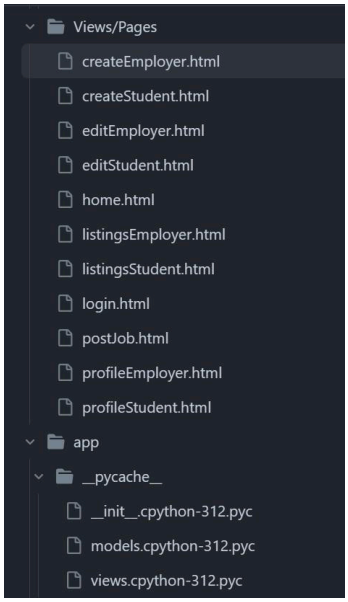

Figure 12: Structure Details 3

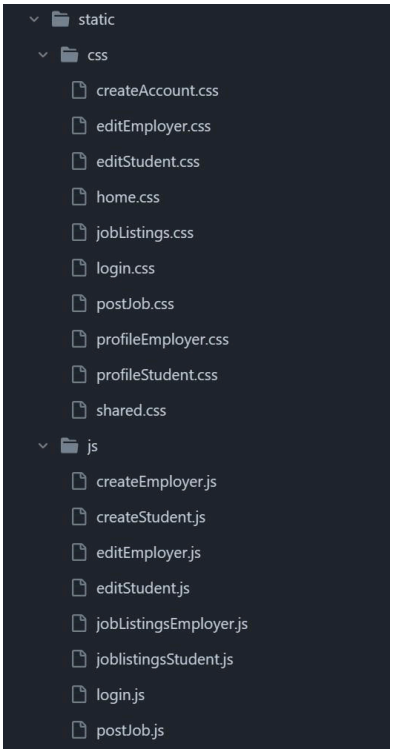

Figure 13: Structure Details 4

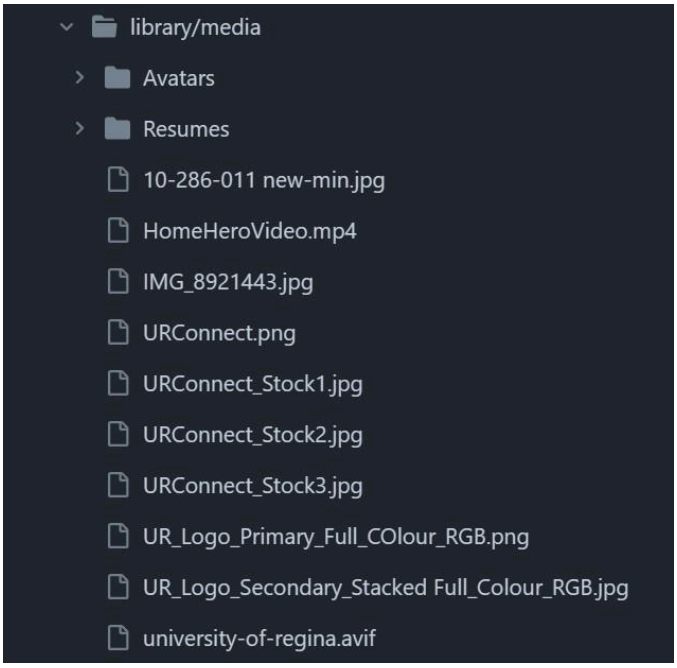

Figure 14: Structure Details 5

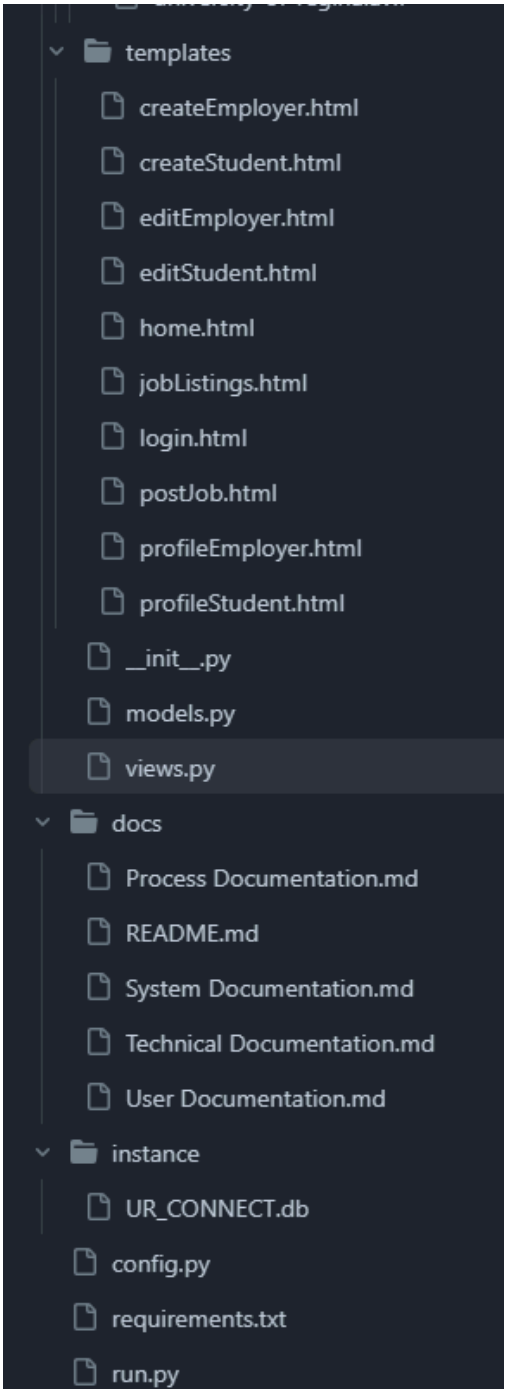

Figure 15: Structure Details 6

- **<u>Discord Server:</u>**

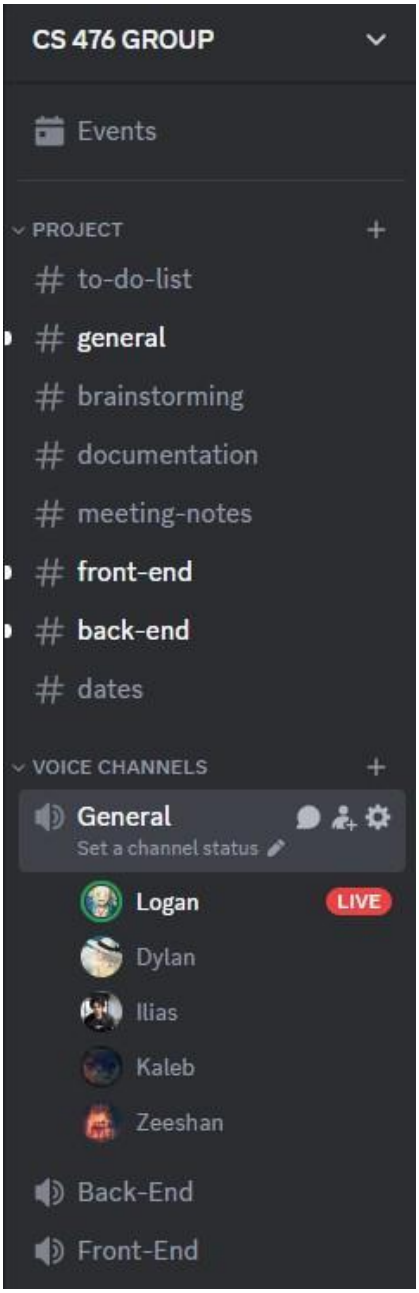

Figure 16: Discord server for UR Connect

6.2 Deployment Diagram

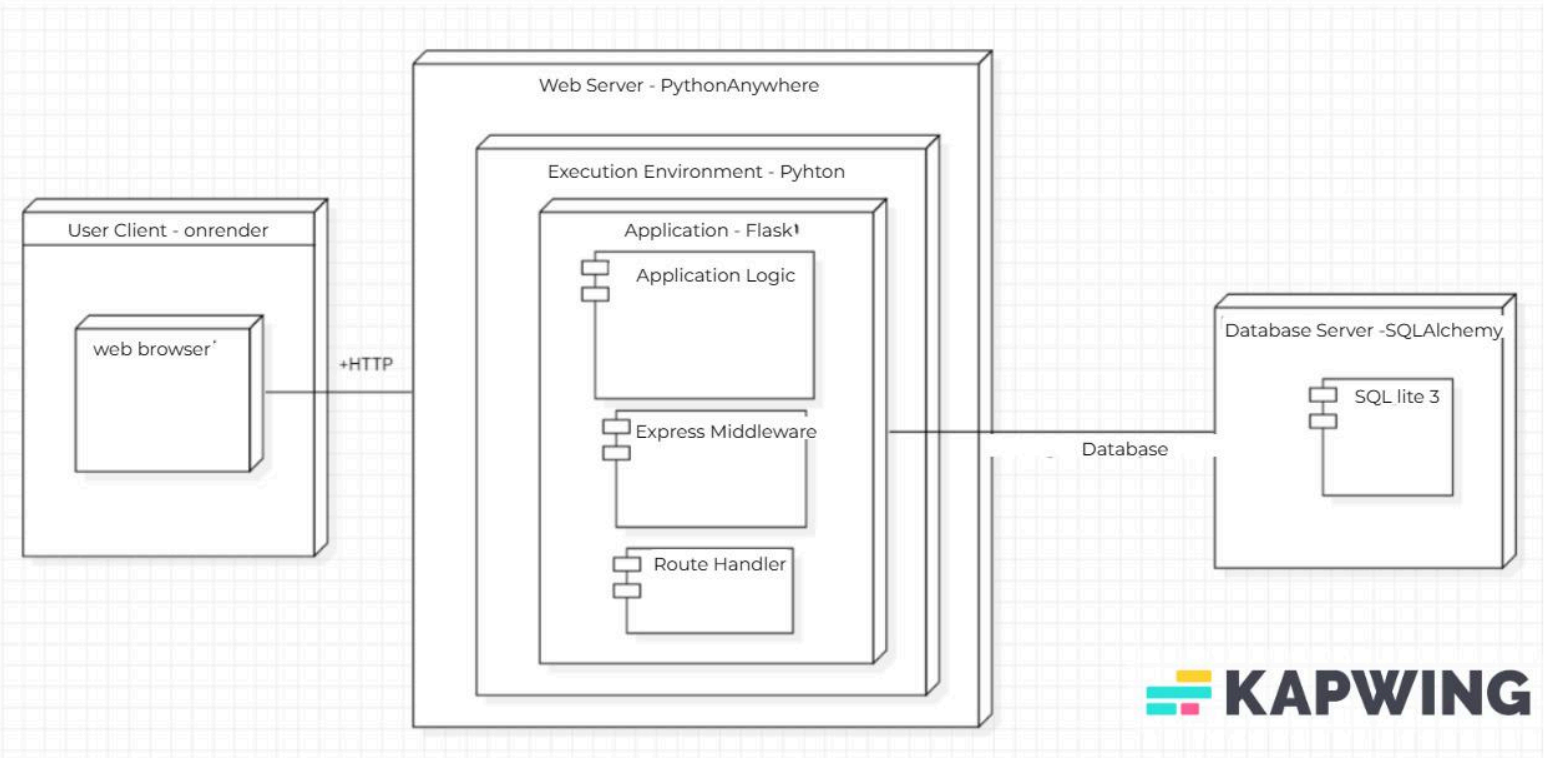

Figure 17: Deployment Diagram

- The website is deployed with the help of PythonAnywhere

6.3 System Data

- Structure of Database:

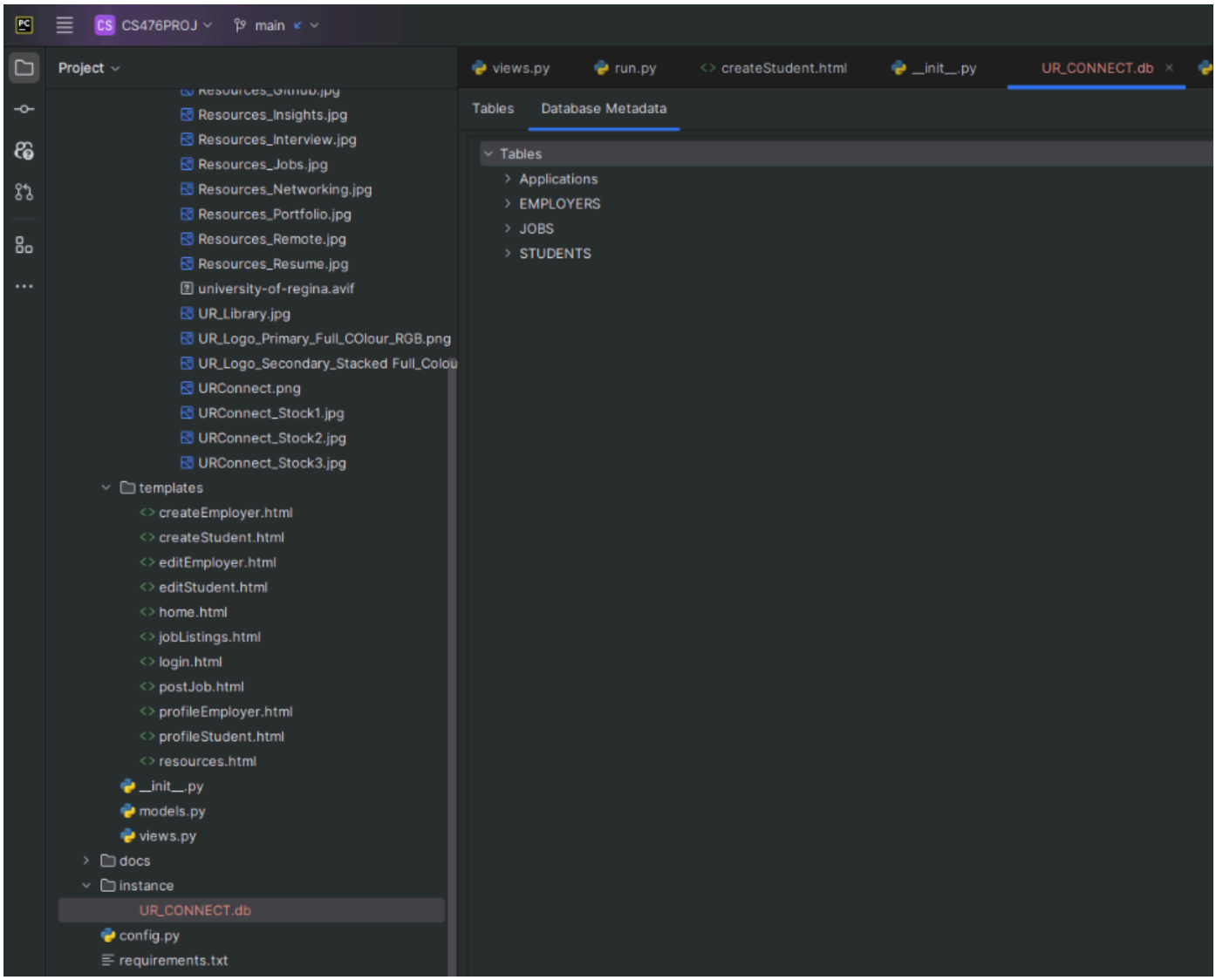

Figure 18: Structure of Database

- Applications:

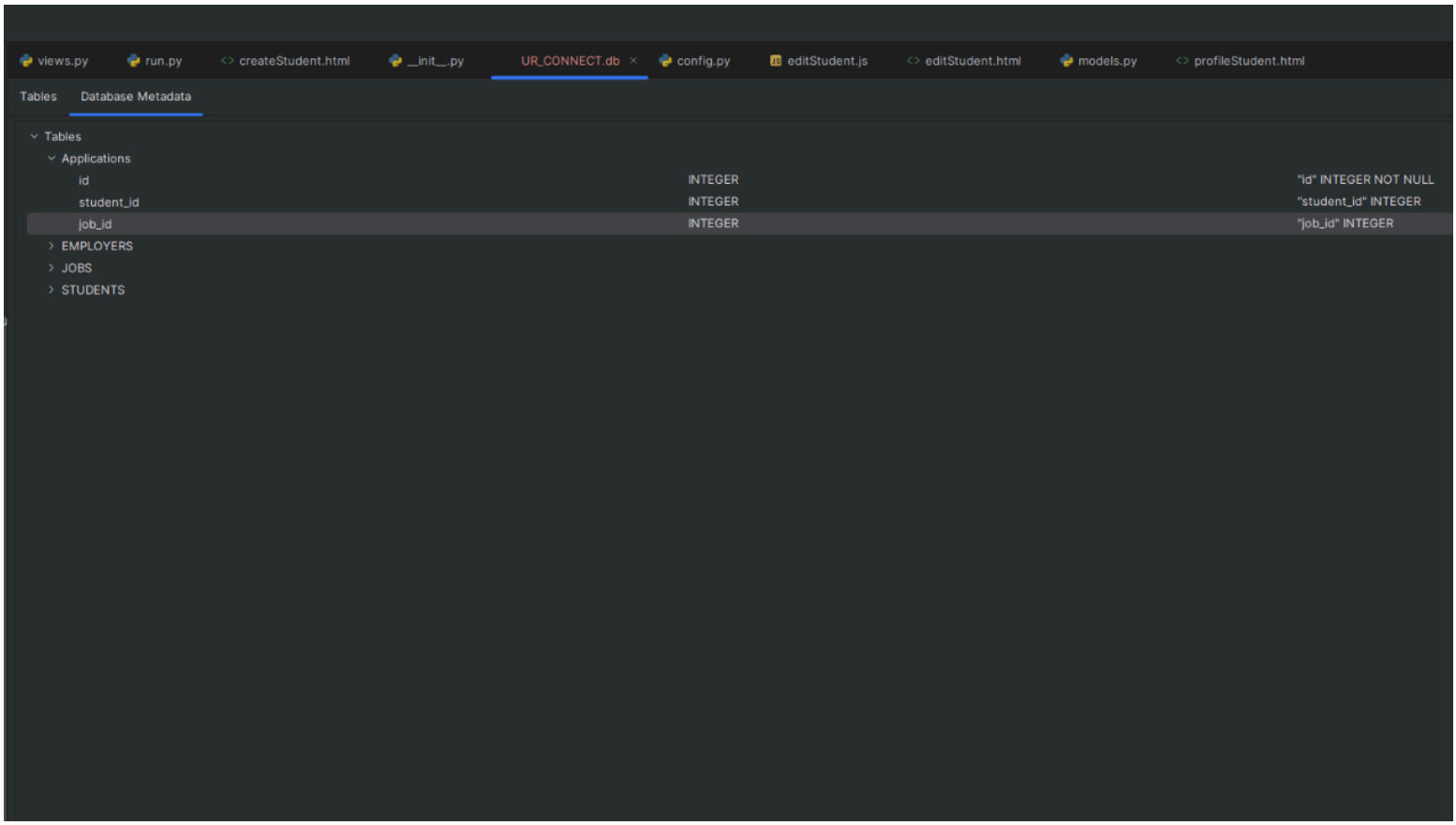

Figure 19: User Model Data

- Employers:

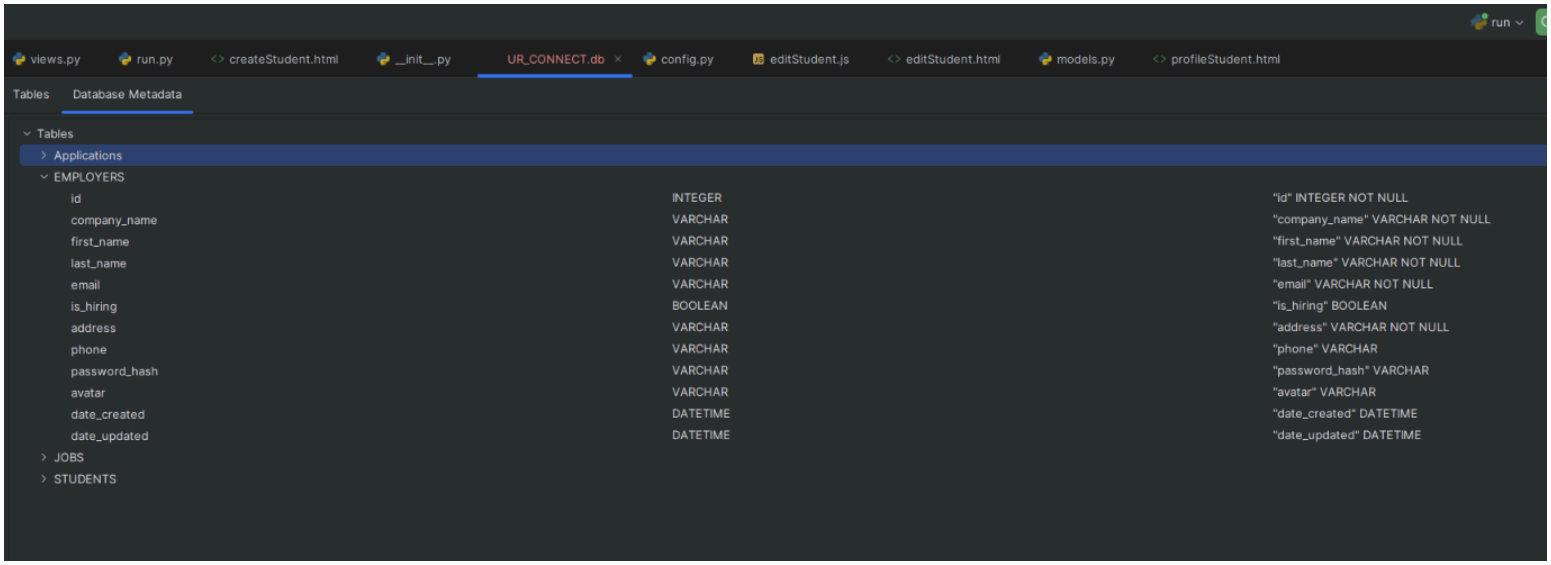

Figure 20: Task Model Data

- Jobs:

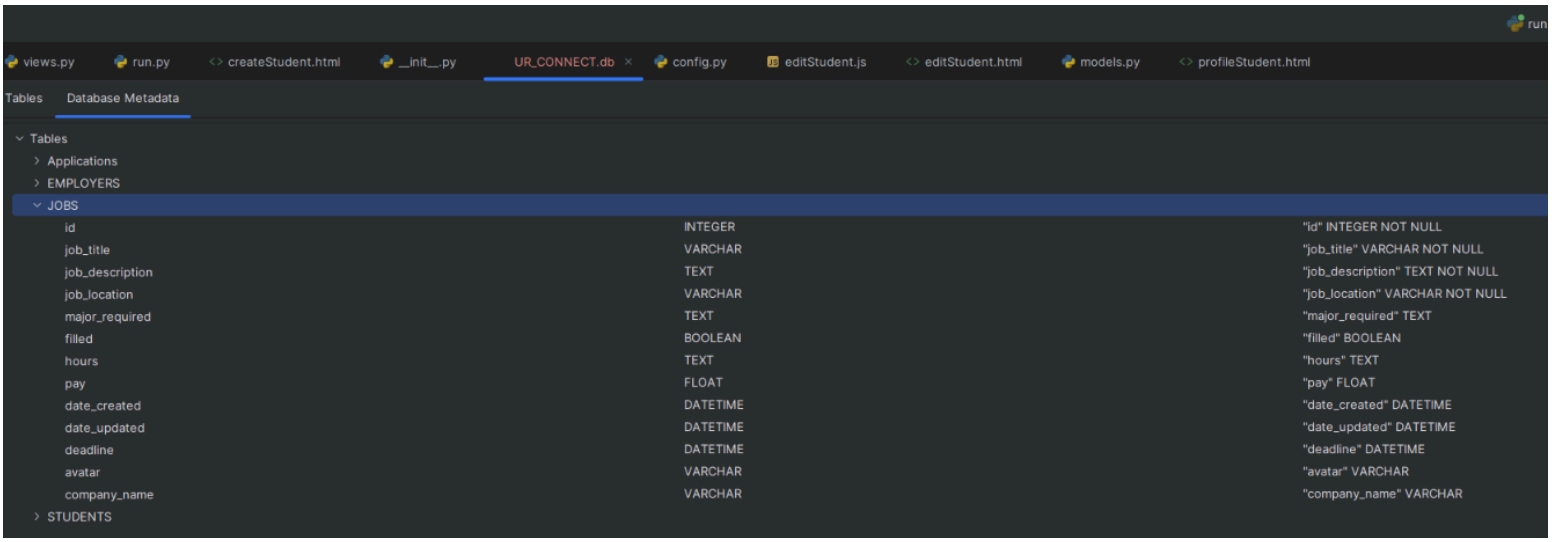

Figure 21: Membership Model Data

- Students

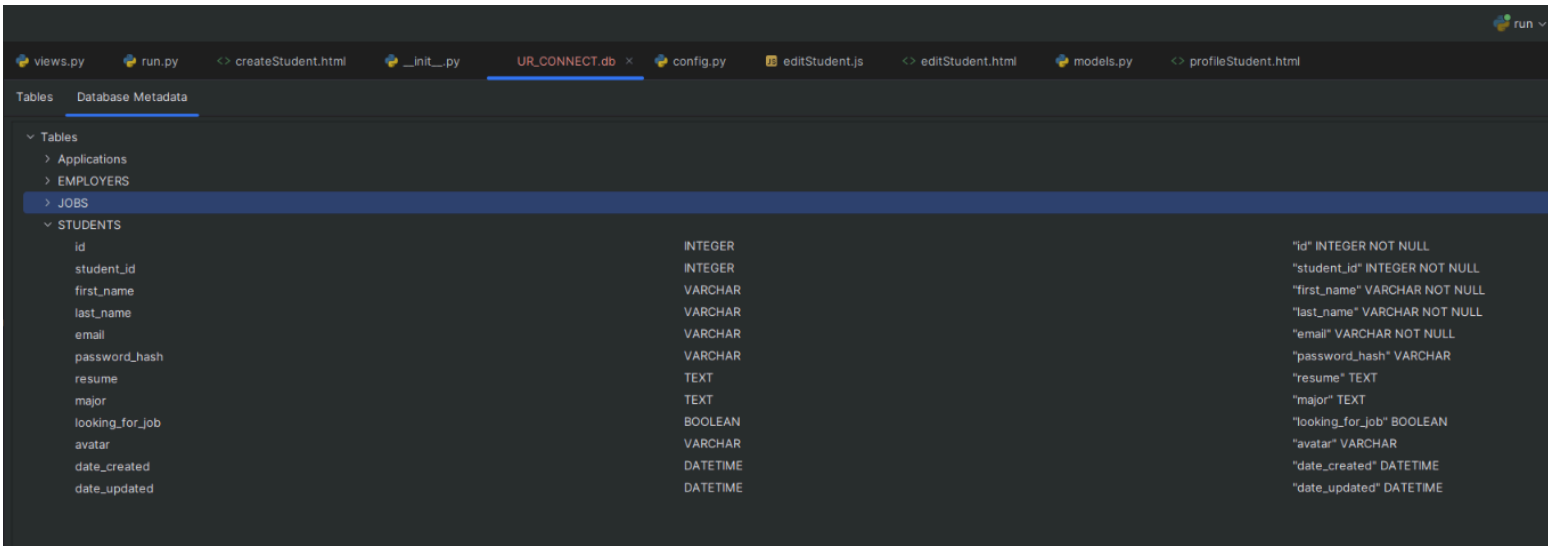

Figure 22: Membership Model Data

6.4 Github Links

- Repository: https://github.com/DylanKenji/CS476PROJ.git
- Link to Website: Home | UR Connect (urconnect-slickdawg.pythonanywhere.com)

7: Technical Documentation

7.1 Programming Languages:

- HTML
- CSS
- JavaScript
- Python

7.2 Software Tools & Environments:

- Pycharm - PyCharm is a Python Integrated Development Environment (IDE) developed by JetBrains. It provides comprehensive tools for Python programming, including code editing, debugging, and project management.
- VScode - VS Code is a lightweight, open-source code editor developed by Microsoft. It's widely used for various programming languages, including Python. It offers a customizable interface and a vast ecosystem of extensions.
- PythonAnywhere - PythonAnywhere is a cloud-based platform designed specifically for Python development and hosting. It provides an online Python environment with an editor, interpreter, and web hosting capabilities, making it convenient for Python project development and deployment.

8: Acceptance Testing

8.1 Correctness Testing

- **Employer**:
  - **Sign-Up Form**:

    ∗ Input: Half data input in Employer Form

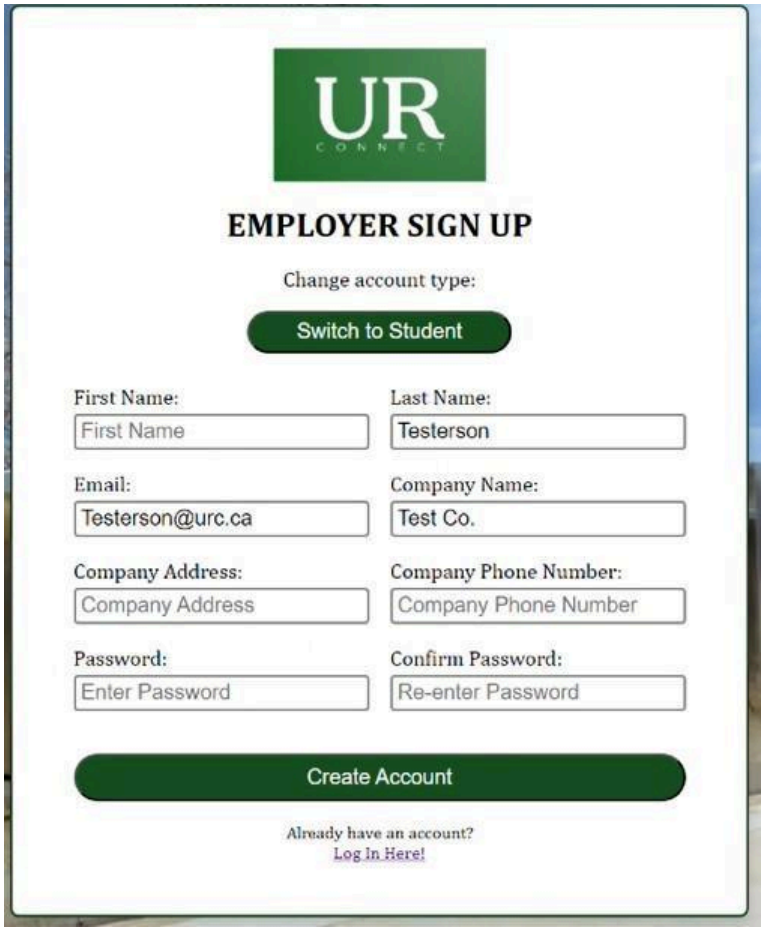

Figure 23: Correctness Testing Employer Test 1 Input

* Output: Errors shown in the field with no data

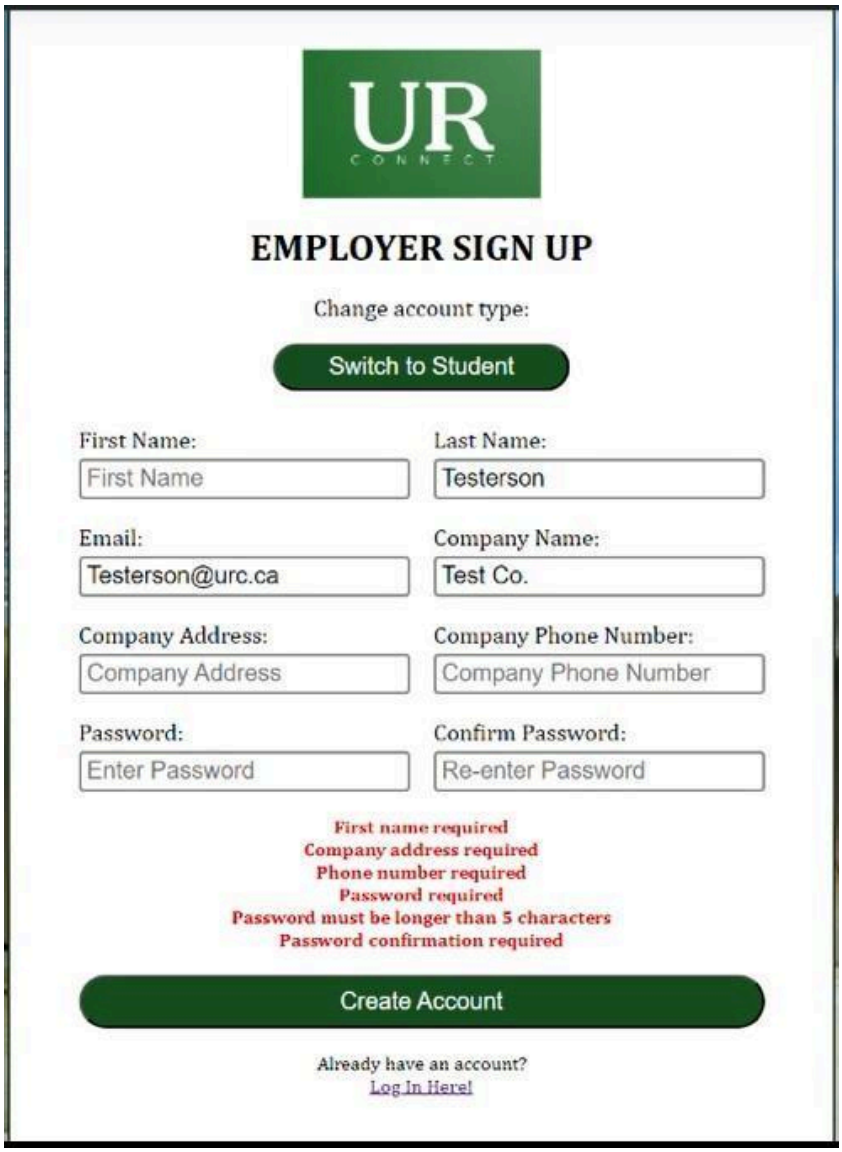

Figure 24: Correctness Testing Employer Test 1 Output

– **Job Posting**:

∗ Input: Job Posting page with incomplete details

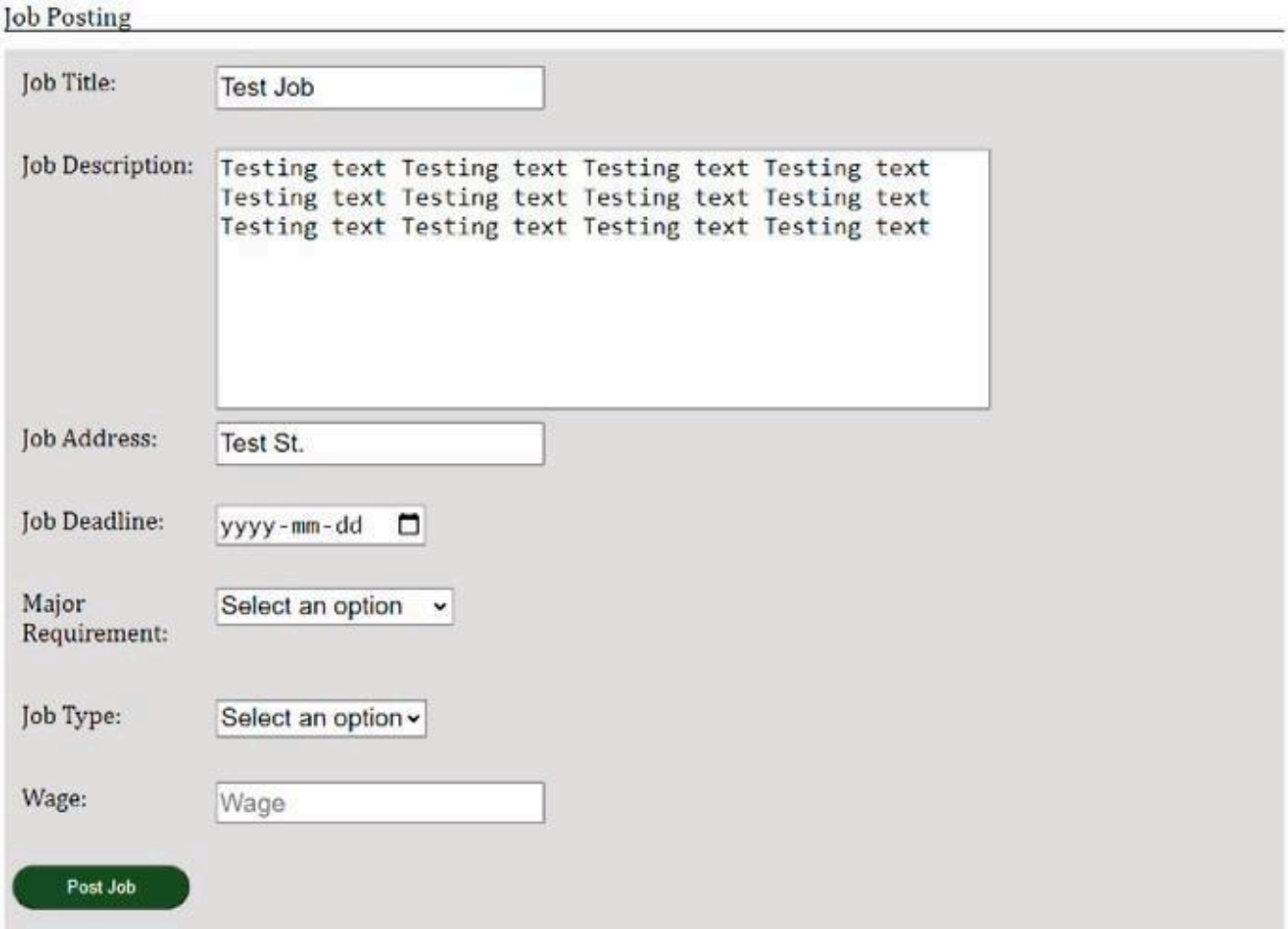

Figure 25: Correctness Testing Employer Test 2 Input

* Output: Errors informing employers to fill out the missing details

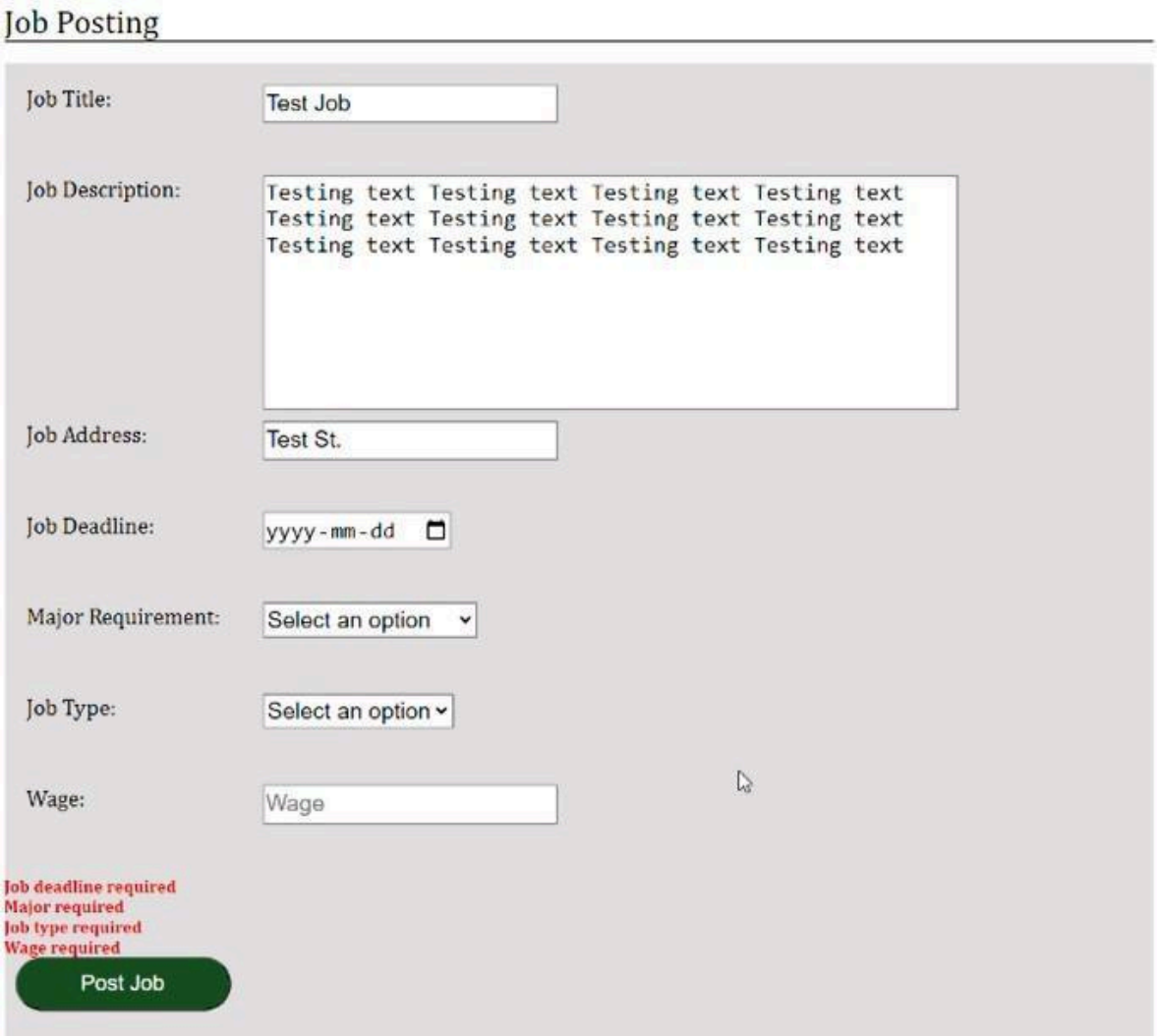

Figure 26: Correctness Testing Employer Test 2 Output

- **Student**:
  - **View resume page**:
    - Input: Student tries to access the view resume page

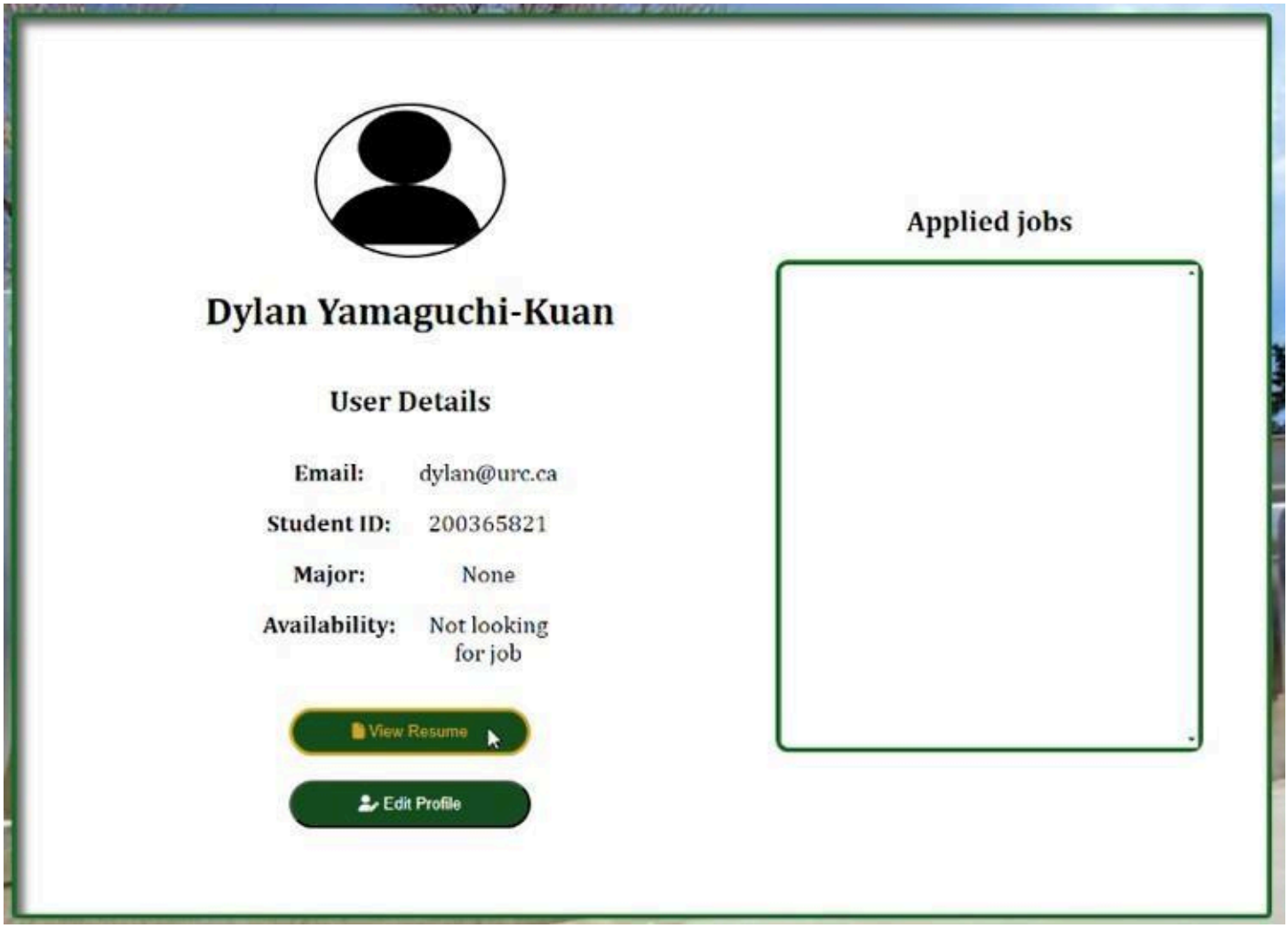

Figure 27: Correctness Testing Student Test 1 Input

* Output: Error message letting the student know to upload a resume first

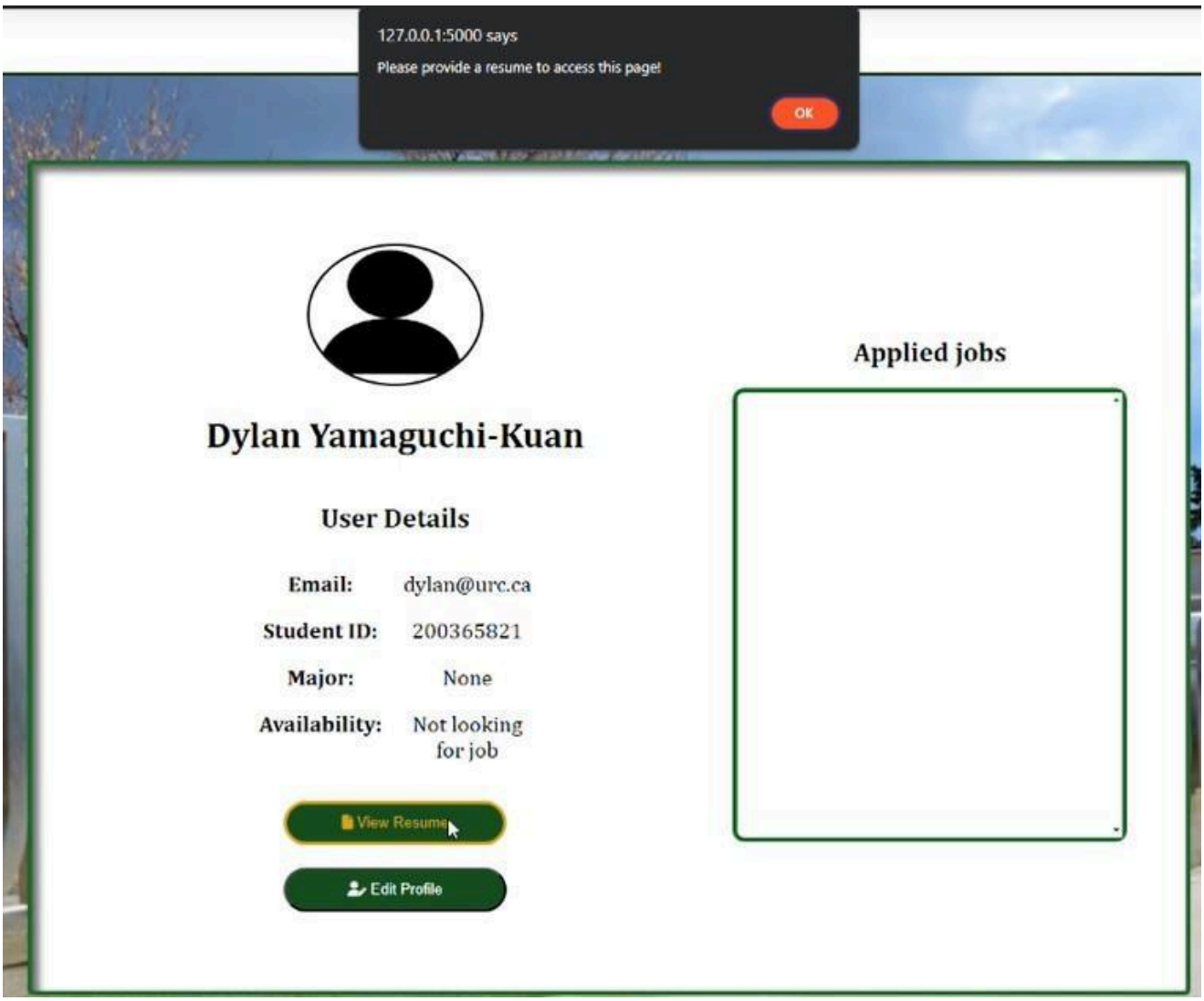

Figure 28: Correctness Testing User 2 Test 1 Output

– **Availability Checkmark**:

∗ Pre - Input: Student Profile page showing “not Looking for job”

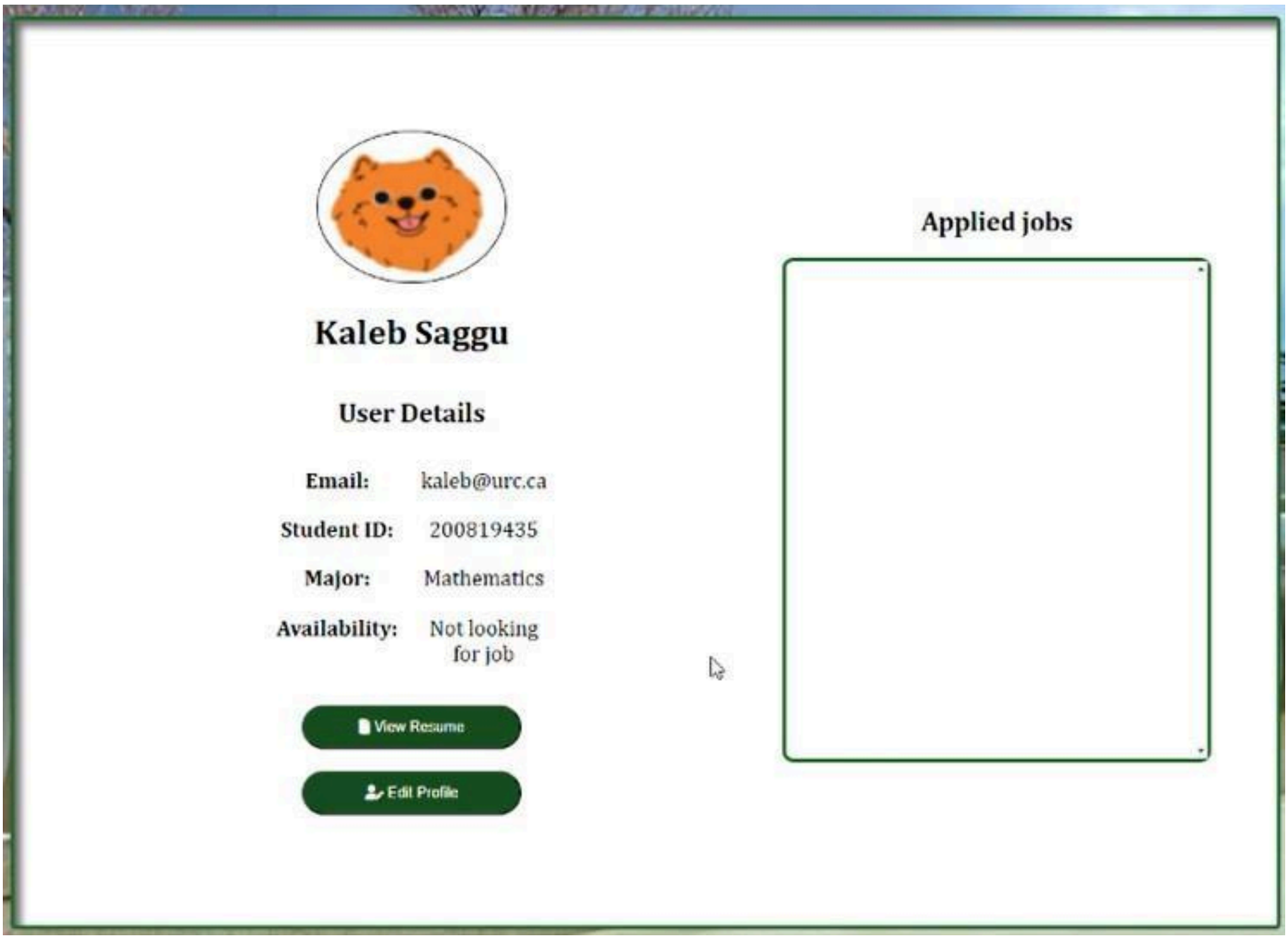

Figure 29: Correctness Testing Student Test 2 Pre - Input

* Input: Checking the availability box

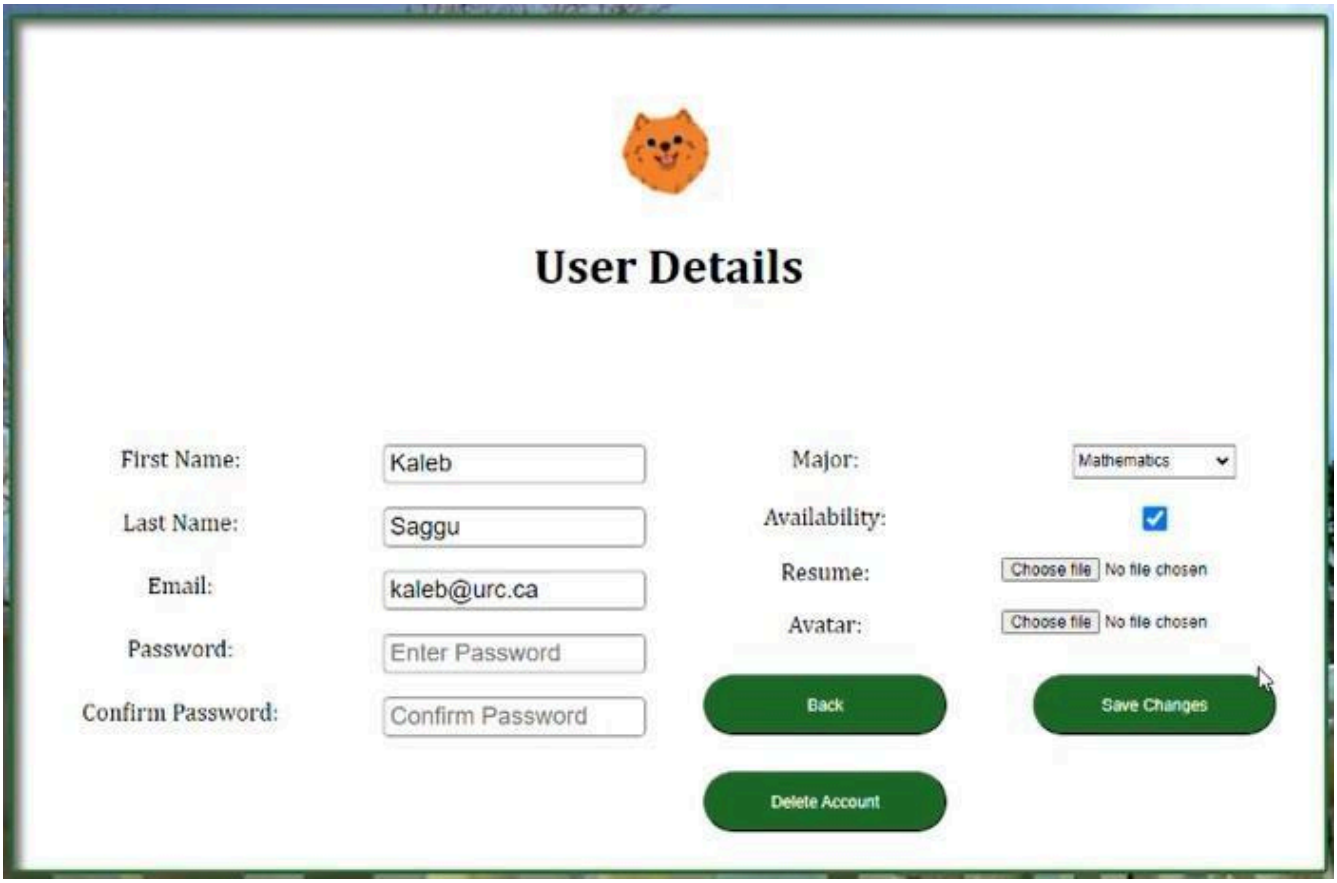

Figure 30: Correctness Testing Student Test 2 Input

* Output: Student profile updated to reflect the change

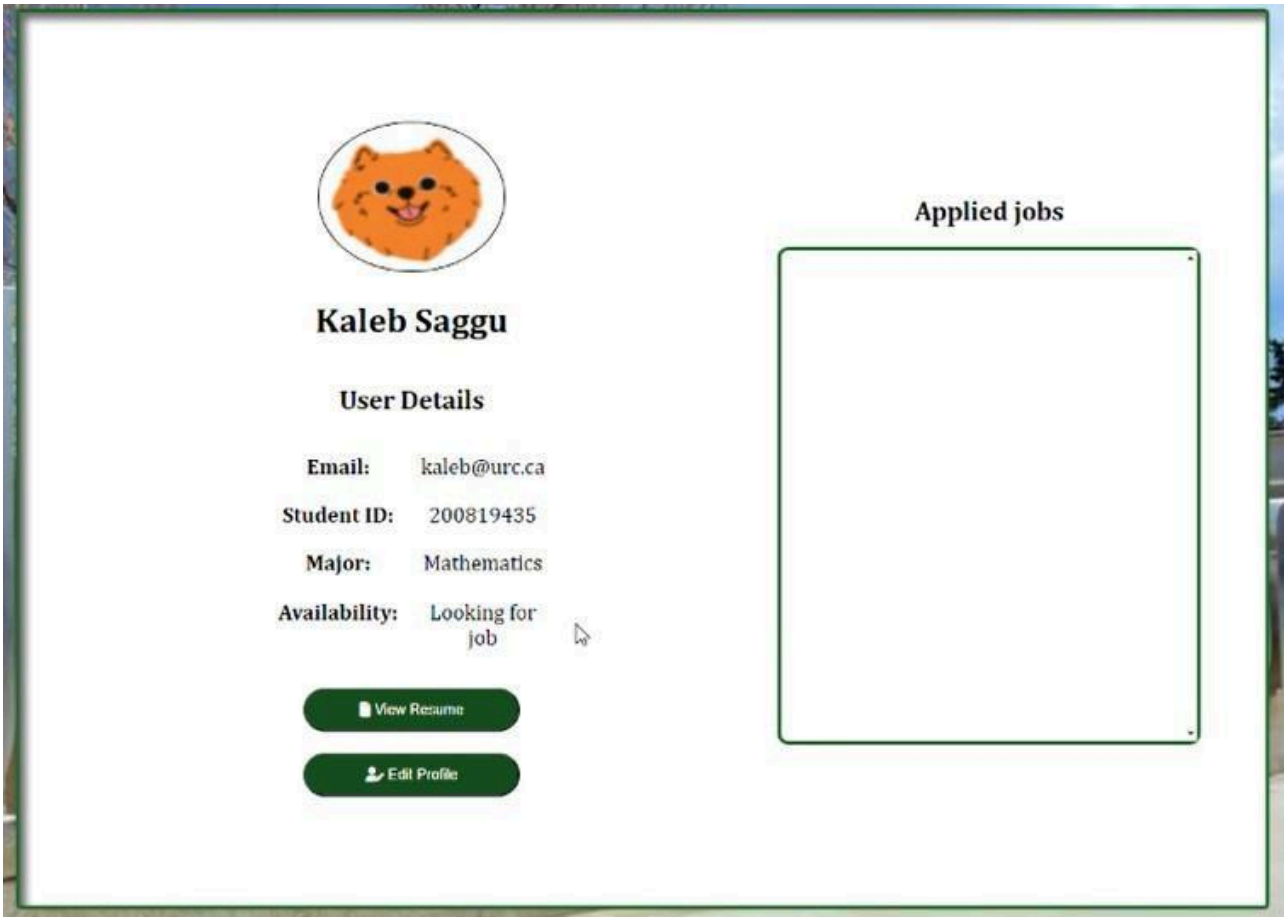

Figure 31: Correctness Testing Student Test 2 Output

8.2 Robustness Testing

- **Employer**:
  - Large Job Description Test:
    - ∗ Input: A job description that exceedes the characters limit

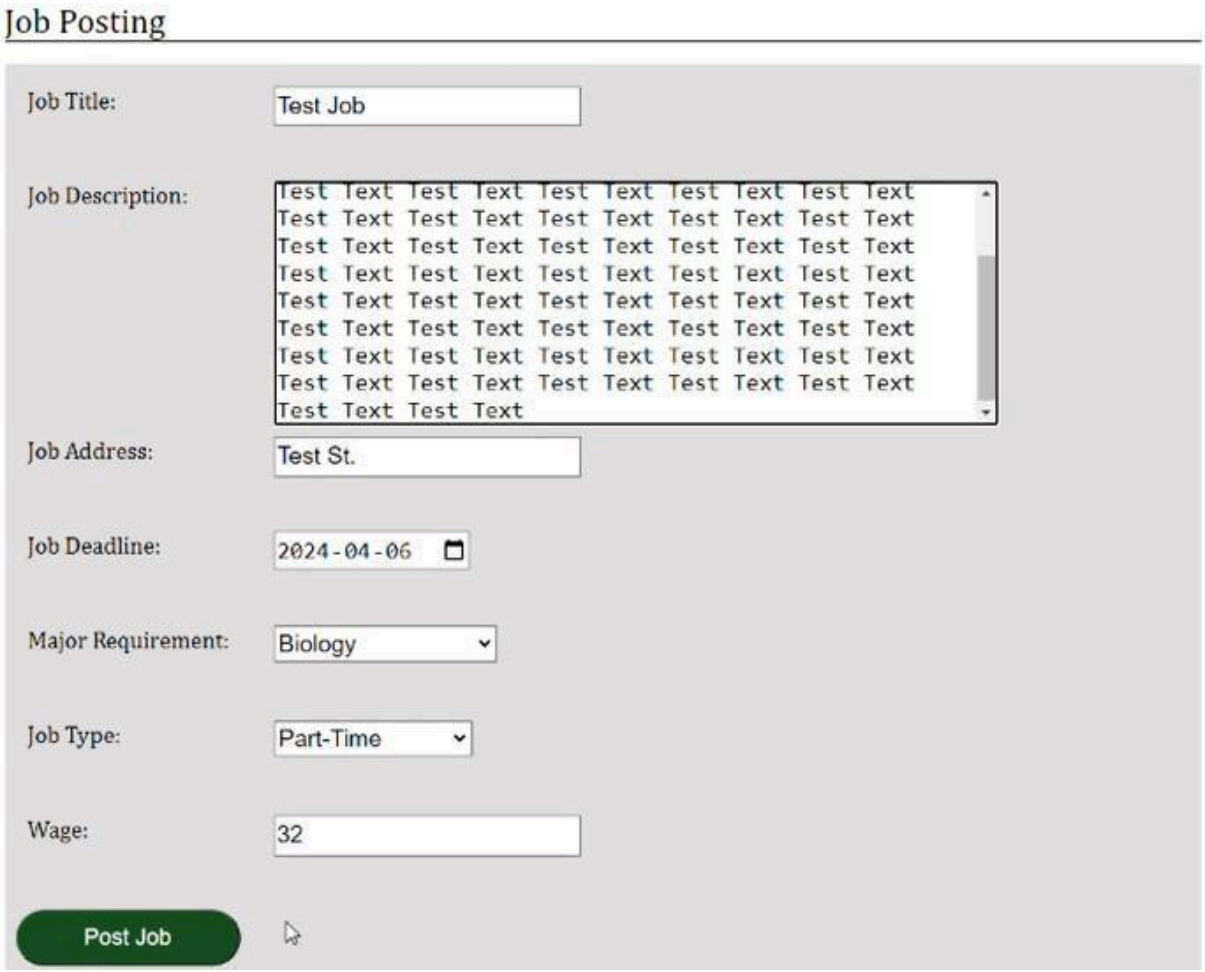

Figure 32: Robustness Testing Employer Test 1 Input

* Output: Text notification showing error on the same page

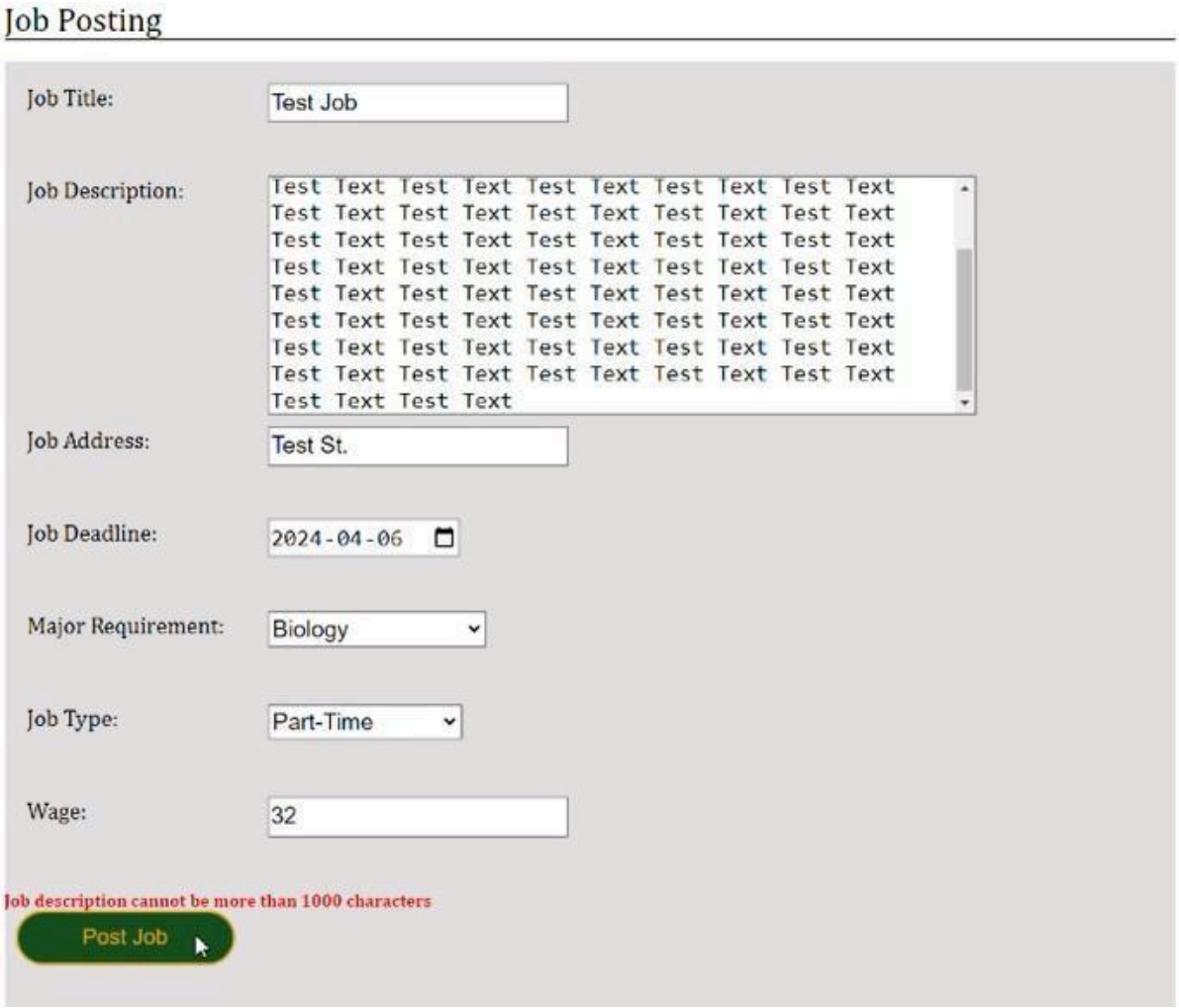

Figure 33: Robustness Testing Employer Test 1 Output

– Invalid Password input on Sign up form:

∗ Input: password that does not meet security standards

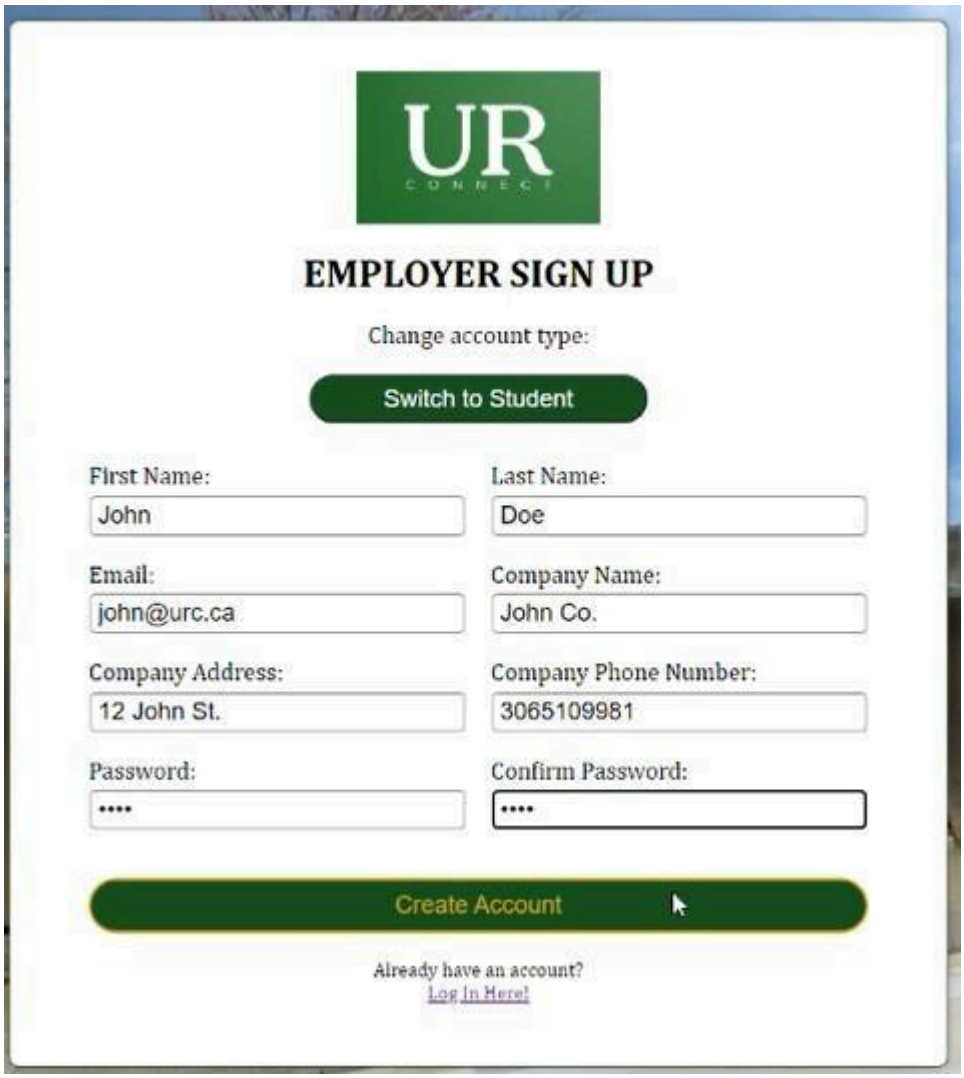

Figure 34: Robustness Testing Employer Test 2 Input

* Output: Errors shown indinacting missing requirements for password

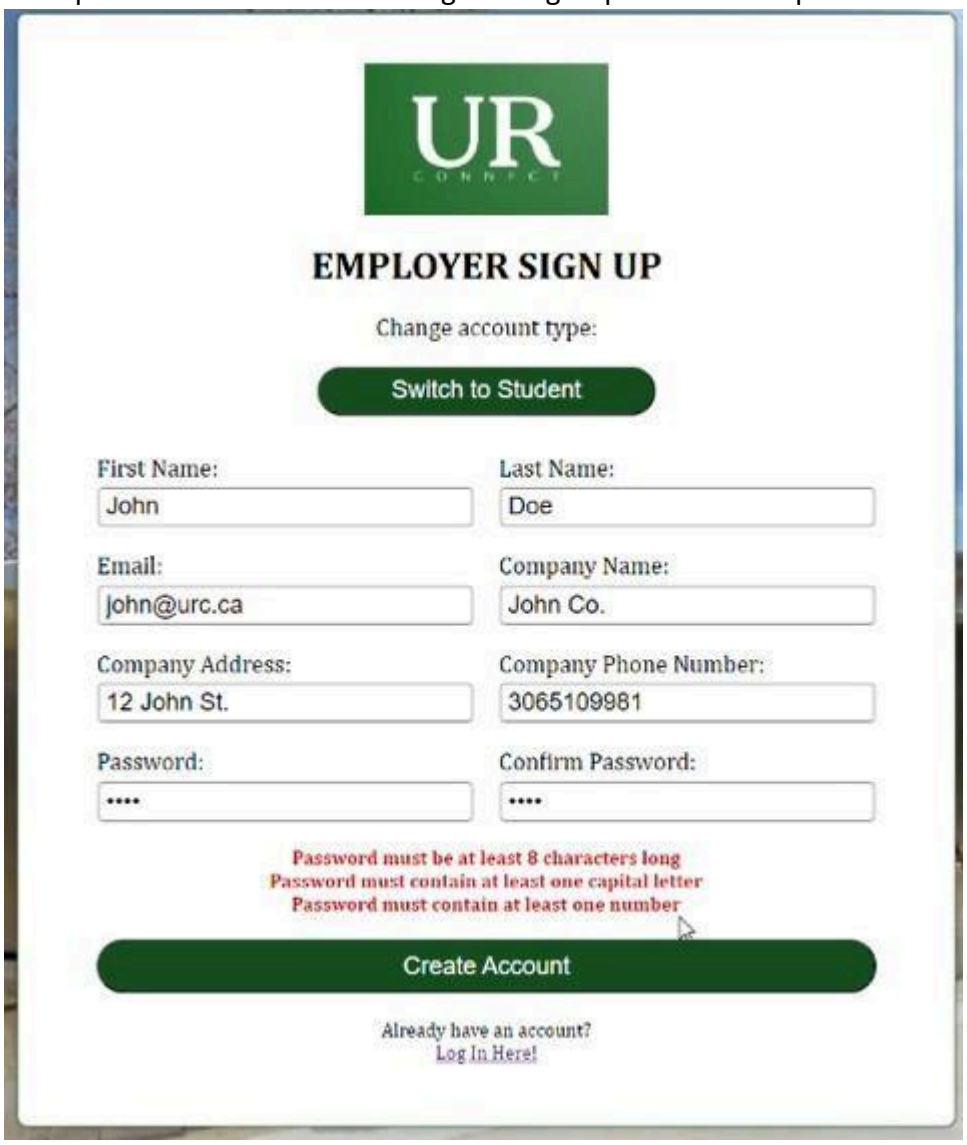

Figure 35: Robustness Testing Employer Test 2 Output

- Stuident User:
  - Sign up form:

    * Input: Incorrect email type

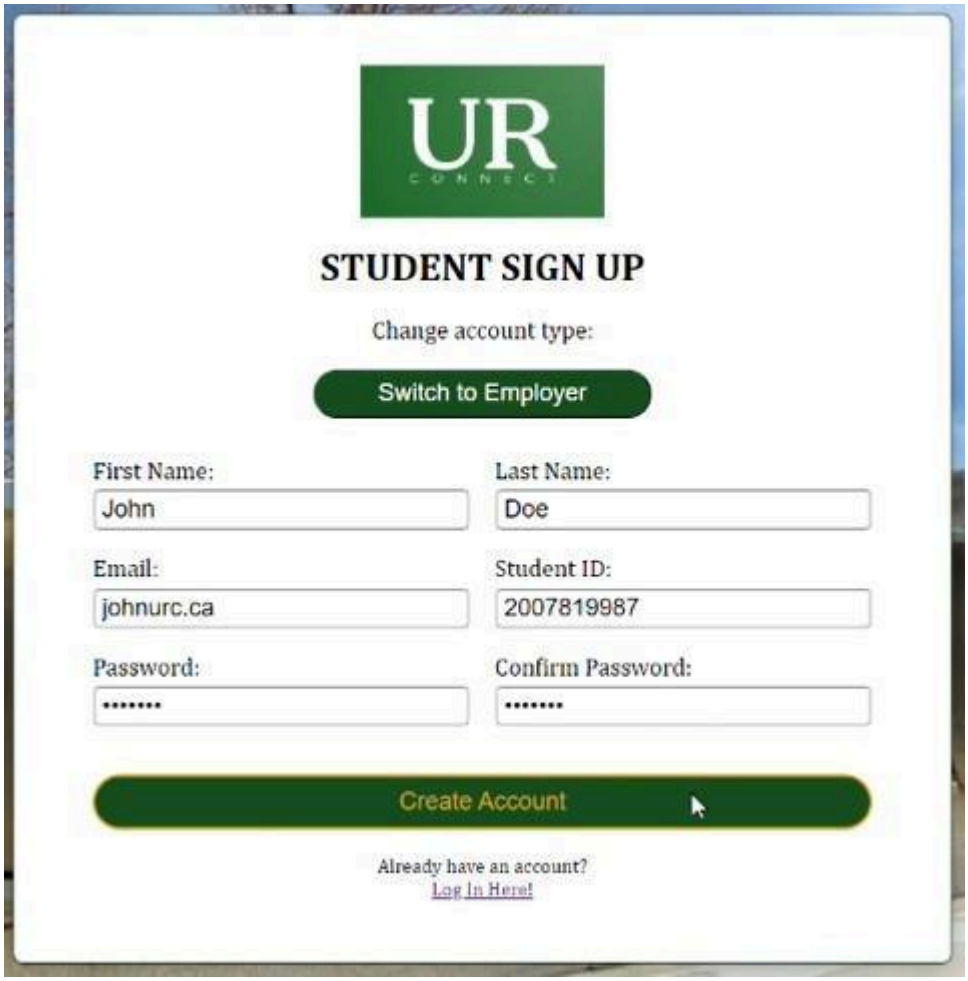

Figure 36: Robustness Testing Student Test 1 Input

* Output: Student cannot signup without an appropriate email

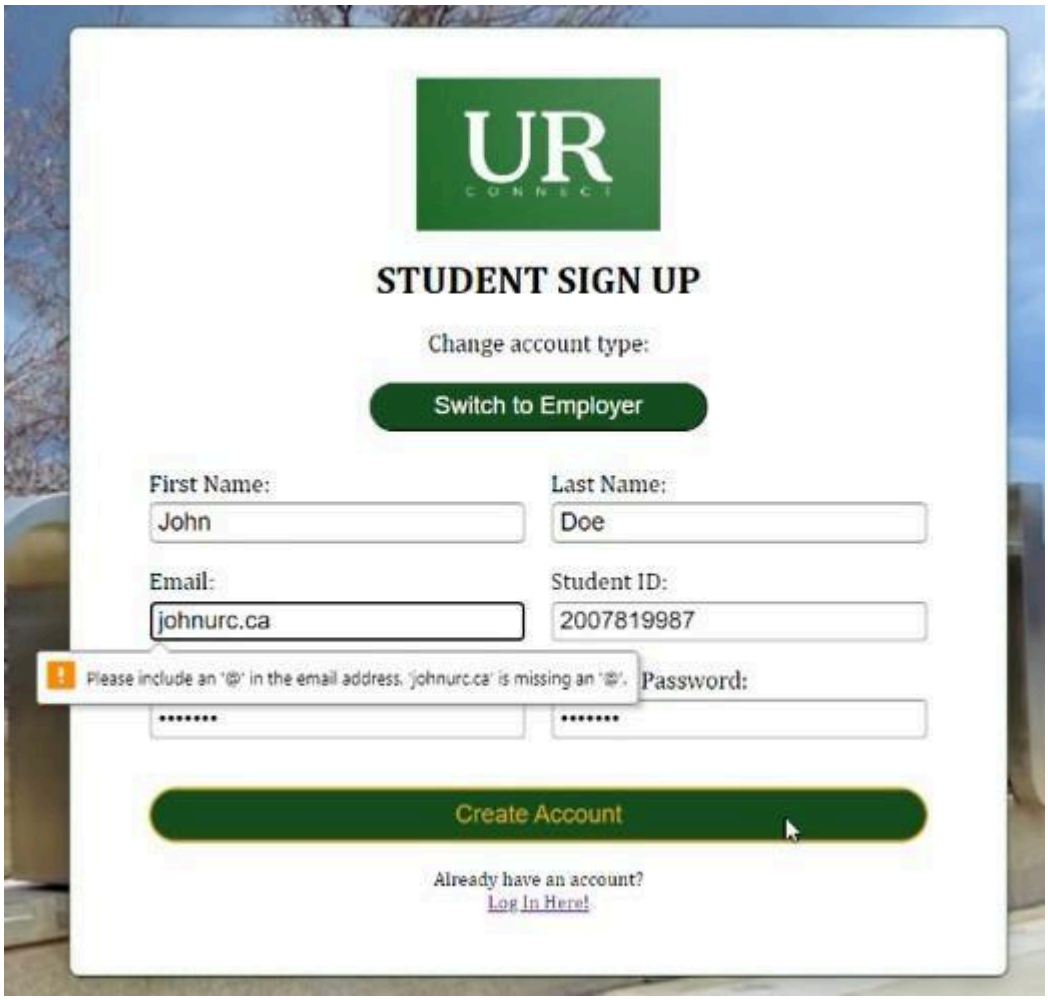

Figure 37: Robustness Testing User 2 Test 1 Output

– Upload does not accepted inappropriate file type for resume:

∗ Input: Student has a folder with multiple files available

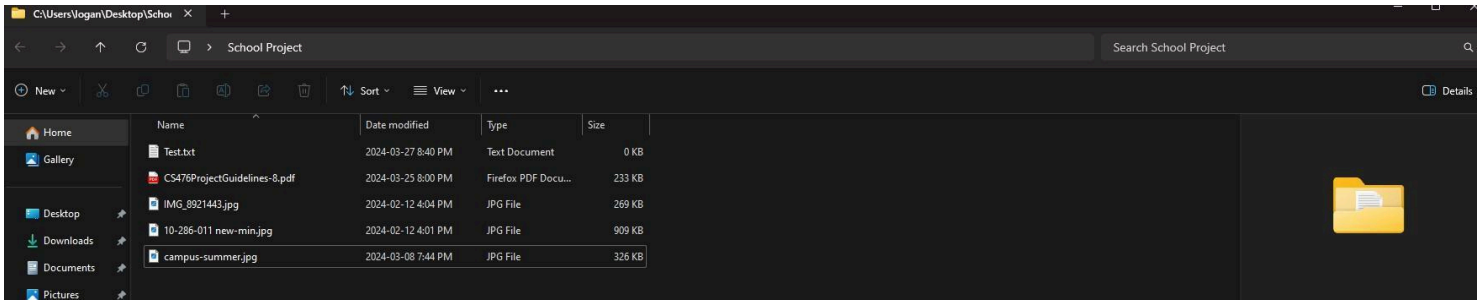

Figure 38: Robustness Testing User 2 Test 2 Input

* Output: the student can only choose pdf file types

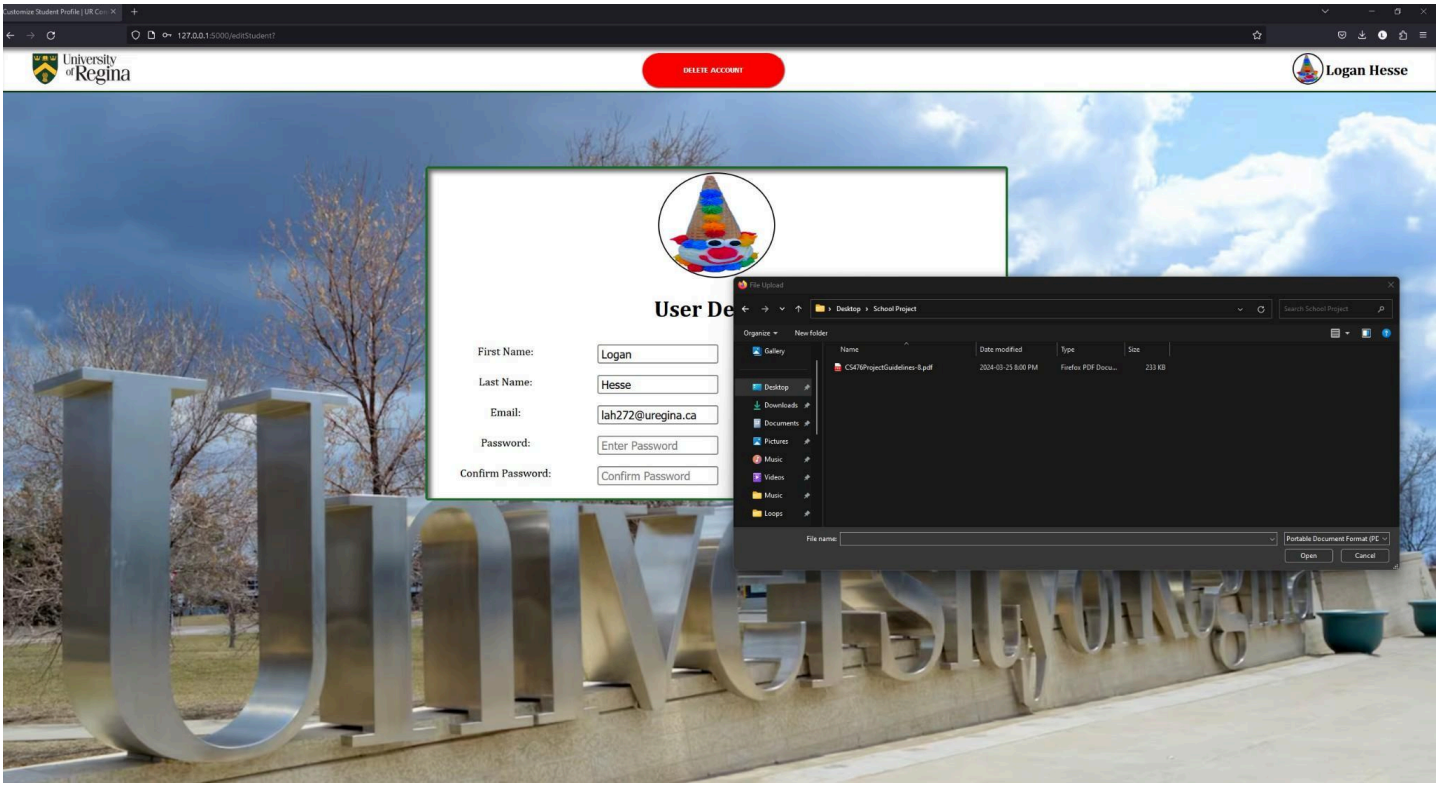

Figure 39: Robustness Testing User 2 Test 2 Output

8.3 Time-Efficiency Testing

- Student User:
  - Sign up page:

    ∗ Input: Sign up Page filled with all the data

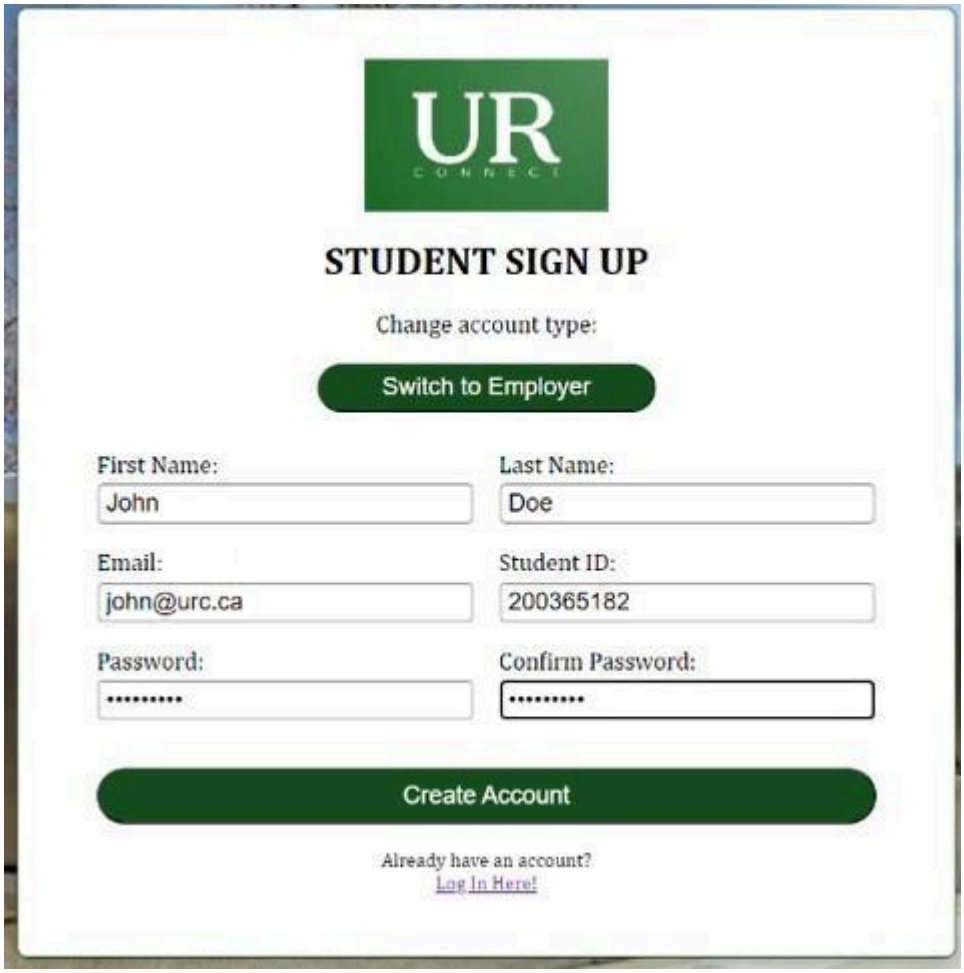

Figure 40: Time Efficiency Testing User 1 Test 1 Input

* Output: Time taken to validate the sign up shown in the timing tab on the web browser

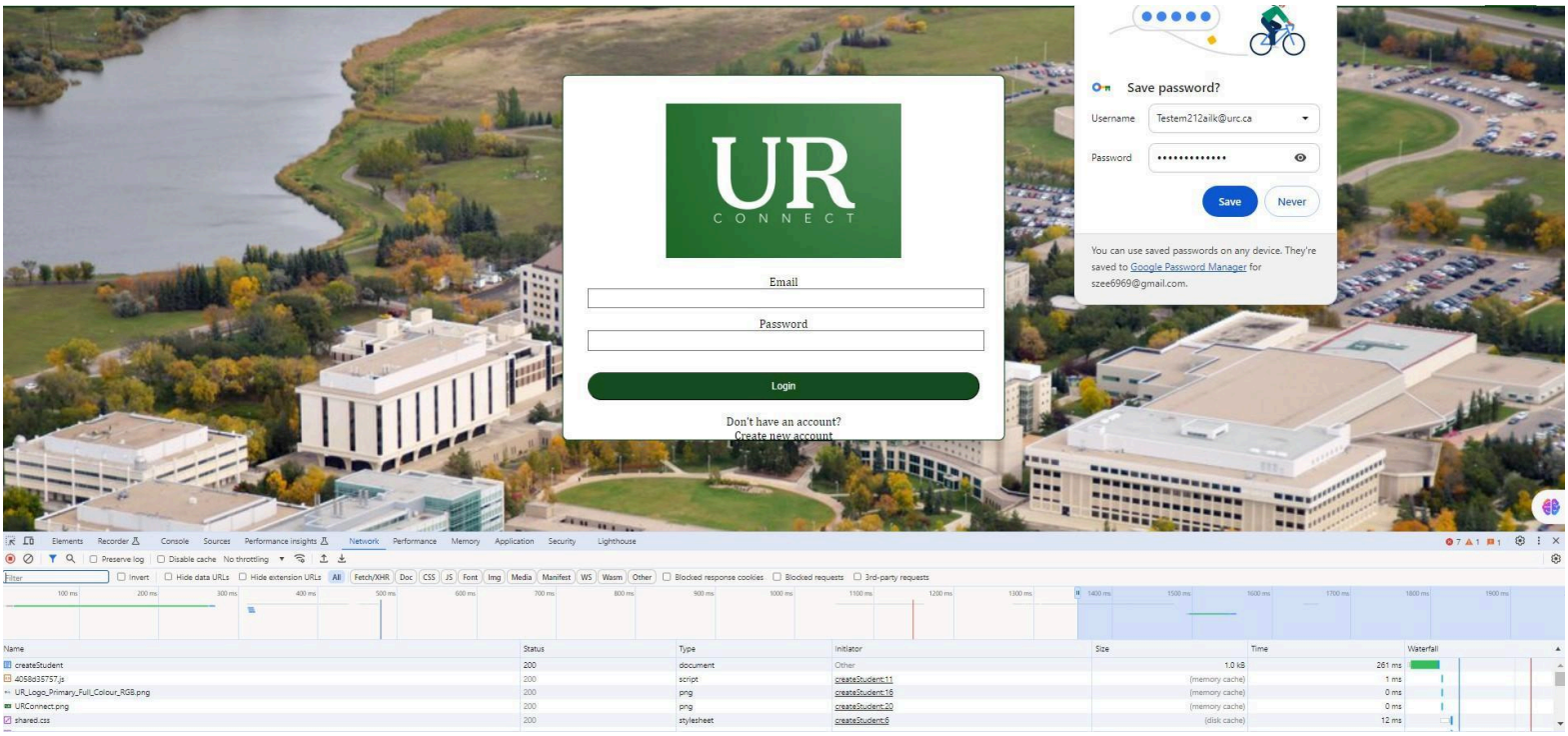

Figure 41: Time Efficiency Testing Student Test 1 Output

– Apply for a job :

∗ Input: Employer creating and submitting a new Job.

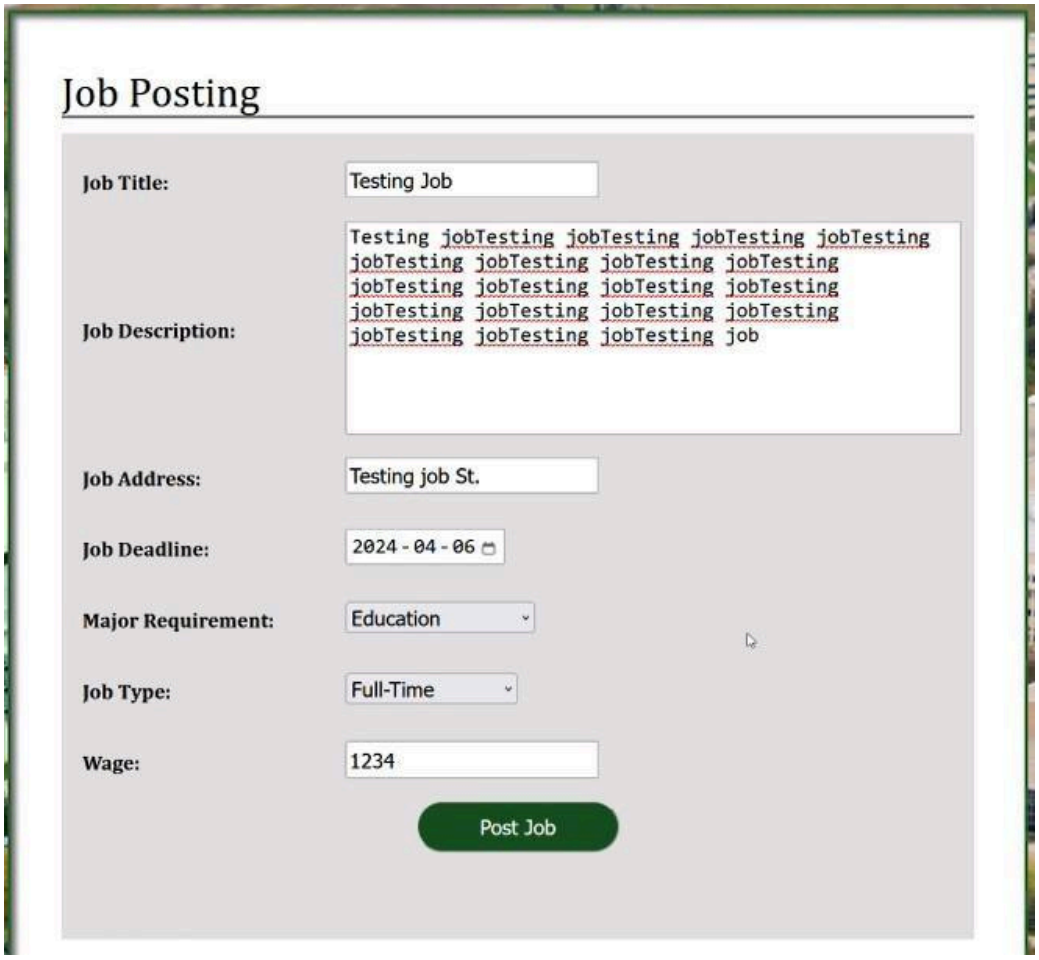

Figure 42: Time Efficiency Testing User 1 Test 2 Input

* Output: Time taken to load the confirmation of application submission

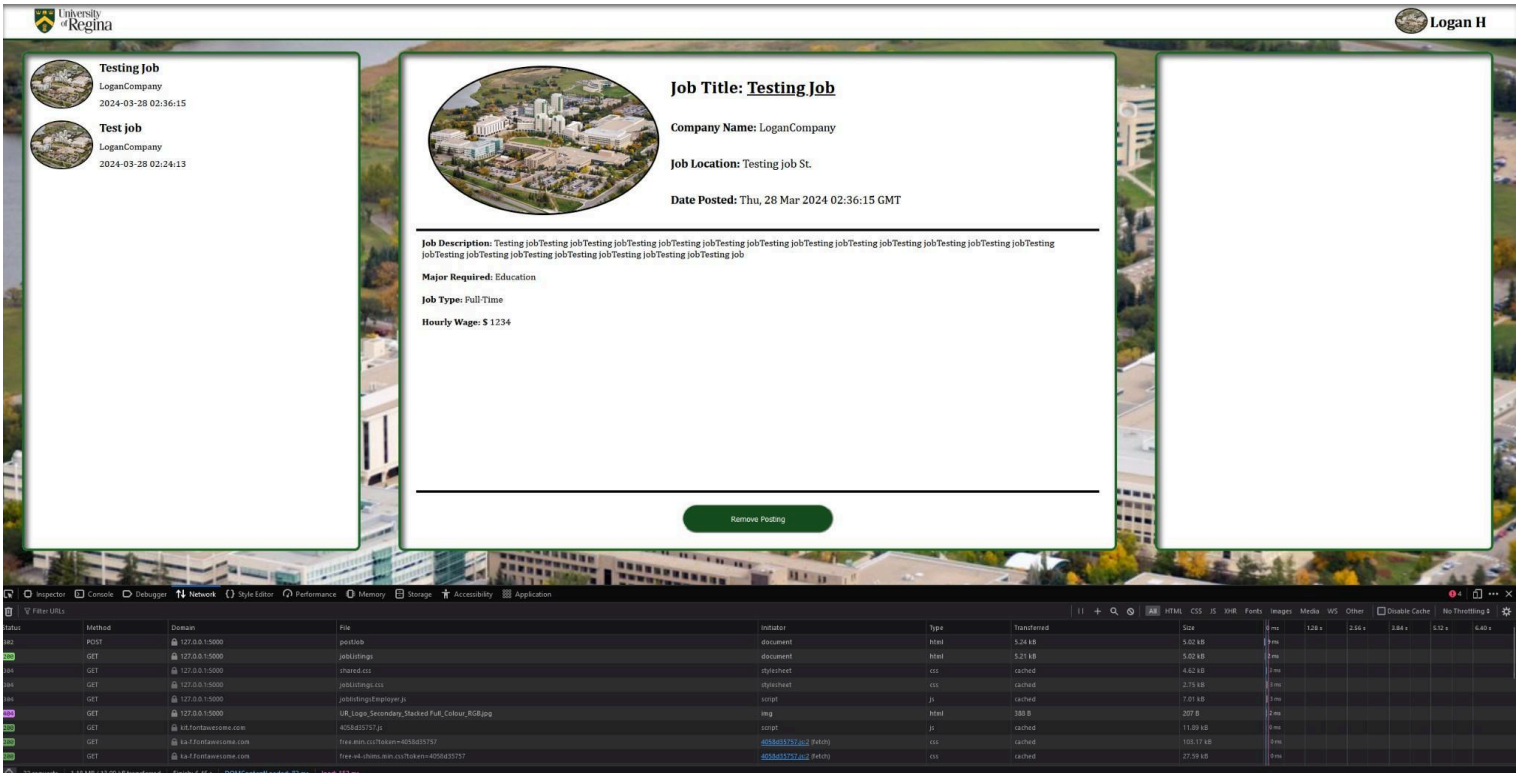

Figure 43: Time Efficiency Testing User 1 Test 2 Outp

9: Future Work

Here are some features that can be added in UR Connect in the future:

1. **Networking Events Integration:** Incorporate a feature that allows students and employers to discover and RSVP to networking events, job fairs, or workshops hosted by the University or local organizations.
2. **Mentorship Program:** Implement a mentorship platform where experienced professionals can mentor students in their field of study or career interests, providing guidance and advice.
3. **Integration with Learning Management Systems (LMS):** Integrate with the University's LMS or other educational platforms to automatically import academic achievements, course projects, and certifications into students' profiles.
4. **Company Reviews and Ratings:** Allow students to rate and review their internship or job experiences with different employers, providing valuable insights for other students and improving transparency in the hiring process.

References

[1] CSS Tutorial, www.w3schools.com/Css/, Mar. 2024.

[2] HTML Tutorial, www.w3schools.com/html/, Mar. 2024.

[3] Nishadha. "UML Diagram Types: Learn about All 14 Types of UML Diagrams." Creately Blog, 28 Sept. 2022, creately.com/blog/diagrams/uml-diagram-types-examples/, Mar. 2024.