VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

REPORT

CAPSTONE PROJECT

DEVELOPING AN ASSISTING SYSTEM FOR STUDENTS TO FIND INTERNSHIPS

MAJOR: COMPUTER SCIENCE

---o0o---

THESIS COMMITTEE: 10-OISP-Software Engineer

SUPERVISOR: PhD. TRƯƠNG TUẤN ANH

REVIEWER: MSc. MAI ĐỨC TRUNG

Student 1: MAI HỮU NGHĨA (2052612)

Student 2: TRẦN TRÍ ĐẠT (2052443)

HO CHI MINH CITY, MAY 2024

Comment

.....
.....
.....
.....
.....
.....
.....

Signature

Trương Tuấn Anh

Commitment

The concluding statement affirms once again that the entire project and its contents are the result of our team's concerted efforts, creativity, and dedication. We have worked diligently to ensure the uniqueness and creativity in every aspect of the project.

We solemnly declare that there has been no copying, replication, or utilization of information from any source other than our project team. Every idea, content, and source code within the website has been independently developed and remains uninfluenced by any external sources.

We commit to adhering to all ethical and professional principles throughout the execution of this project, ensuring integrity and transparency in all aspects. We take pride in the outcomes of this project and believe it will positively contribute to supporting internships and education within the student and business community.

Acknowledgement

To successfully conclude our capstone project, we extend our heartfelt appreciation to Mr. Truong Tuan Anh, a distinguished lecturer at the Faculty of Computer Science and Engineering. Throughout our project research, Mr. Anh demonstrated unwavering care, thoughtfulness, patience, and wholehearted dedication in guiding and supporting our team. His mentorship significantly deepened our understanding of the subject matter while adeptly identifying and addressing the challenges and deficiencies we encountered. Mr. Anh's invaluable feedback, suggestions, and insights have played a pivotal role in our substantial improvements.

Our team also wishes to express gratitude to the entire faculty of the Faculty of Computer Science and Engineering, as well as all the esteemed educators at the University of Science and Technology. Your commitment to imparting knowledge during our tenure at the university has equipped us with a wealth of expertise and insights that have been instrumental in our project's success.

Acknowledging our project's inherent limitations stemming from our limited experience and knowledge, we eagerly anticipate and welcome your constructive feedback, assessments, and guidance.

Our profound appreciation extends to all those who have contributed to this project's realization. This accomplishment is the culmination of not only our relentless efforts but also the invaluable support from our mentors and peers. This bridge we have constructed shall usher us into a more stable and promising future, and we extend our well wishes to all for abundant success in their life endeavors.

Summary

Chapter 1: Introduction

Introduction to the project, what is the project motivation, project scope and project goals. This also includes some definitions that will be used throughout the report.

Chapter 2: Analyze Current Systems

Analyze some current system in the market, in this report will analyst CSE internship website, University of Economics HCMC (UEH) Department of Student Affairs (DSA) Career site and LinkedIn - Social networking site for business community.

Chapter 3: Requirement Elicitation

This all the stakeholder in which will be revolved in this project, list the functional and non-functional requirements.

Chapter 4: UML Diagrams for System

Contains the Use-case diagrams, Use-case scenarios, Sequence diagrams and Activity diagrams of the system.

Chapter 5: System, Database and User Interface

Overall design of the whole system, database design and User interface design

Chapter 6: Technologies

All the technologies will be used in the frontend, backend, database and cloud service (Azure Blob Storage, Azure Caching for Redis, …)

Chapter 7: Implementation

Contains all the implementation of the user interface, list of the API, Google OAuth and Azure deployment

Chapter 8: Testing

Unit testing for backend, manual testing for frontend and workload testing for the whole system

Chapter 9: Deployment

Deployment of the project to Azure cloud service

Chapter 10: Conclusion

Workload for each member, the weekly process, overall conclusion for the whole project and the future plan and improvement.

Chapter 11: Bibliography

The reference for the report.

Table of Contents

List of Table

List of Figure

1. Introduction

1.1. Motivation

The economic recession has led to a decline in the demand for internships, as companies cut back on their workforce. However, internships are still a valuable opportunity for students to gain practical experience and learn about the workforce. This study proposes the development of a system to connect students, faculty, and businesses for internships. The system would automate procedures and improve communication between the three parties, making it easier for students to find internships.

The proposed system could be beneficial for students, faculty and businesses. For students, the system would make it easier to find internships and learn about different career paths. For businesses, the system would help them to find qualified interns and improve their recruitment processes. And finally for faculty, the system would help the faculty manage their students more easily and improve the internship acknowledgment process.

1.2. Project goal

The primary goal of this project is to address the challenges posed by the economic recession, which has resulted in a reduction in the availability of internships. Our aim is to develop a comprehensive system that facilitates seamless connections between students, faculty, and businesses in the context of internships. This system will leverage automation to streamline administrative processes and enhance communication channels among these key stakeholders. By achieving this goal, we intend to make the process of finding and participating in internships more accessible for students, while simultaneously providing significant benefits to businesses in terms of intern recruitment and improved hiring processes. Moreover, the project seeks to empower faculty members in effectively managing their students' internship experiences and optimizing the internship acknowledgment process. Ultimately, our project aims to bridge the gap

between the supply and demand for internships, offering a more efficient and enriching experience for all parties involved.

1.3. Project scope

1.3.1. Project Objectives:

Develop a comprehensive internship management system to address the challenges resulting from the economic recession, enhancing the availability of internship opportunities for students.

1.3.2. Deliverables:

The primary deliverable is a fully functional internship management system that connects students, faculty, and businesses. This includes a user-friendly web application, a database for storing relevant information, and associated documentation.

1.3.3. Features and Functionality:

The system will feature:

- Student portal for browsing and applying for internships.
- Faculty portal for managing student internships and approvals.
- Business portal for posting internship opportunities and reviewing applicant profiles.
- Automated procedures for streamlining recruitment, approval, and acknowledgment processes.
- Improved communication channels among students, faculty, and businesses.

1.3.4. Exclusions:

The system will not directly handle the recruitment processes beyond internship facilitation. It will not be responsible for student grades or academic assessment.

1.3.5. Constraints:

The project will operate within the constraints of the allocated budget, time frame, and available technology resources. Integration with existing university systems will be considered.

1.3.6. Assumptions:

It is assumed that participating businesses will actively engage with the system, posting accurate and timely internship opportunities. Faculty members are assumed to effectively utilize the system for student management.

1.3.7. Stakeholders:

Key stakeholders include students, faculty members, and businesses participating in the internship program. Additionally, project developers and administrators will play a crucial role.

1.3.8. Acceptance Criteria:

The project will be considered successful when the system is operational, facilitating seamless connections between students, faculty, and businesses, and when users report improved efficiency in finding, managing, and participating in internships.

1.3.9. Project Boundaries:

The system will focus on facilitating internship connections and managing related processes. It will not extend to academic assessments, employment beyond internships, or broader university management systems.

1.3.10. Dependencies:

The project is dependent on the active participation of students, faculty, and businesses. Integration with university databases or systems may be necessary for accurate student information.

1.3.11. Risks:

Risks include potential resistance to change, technical challenges in system integration, and fluctuating participation levels from businesses.

1.3.12. Project Timeline:

A tentative timeline for project completion and deployment is outlined, with milestones for system development, testing, and implementation.

Meeting	Time	Agenda
1	Sep 2023 - Week 1	Kick-off, project definition, and review references
2	Sep 2023 - Week 3	Project description with UML diagrams
3	Oct 2023 - Week 1	System, data, and UI designs
4	Oct 2023 - Week 3	Technology research and preparation for implementation
5	Nov 2023 - Week 1	Project implementation (focus on specific modules)
6	Dec 2023 - Week 3	Project Phase 1 summary and discussion
7	Feb 2024 - Week 1	Project Implementation
8	Feb 2024 - Week 3	
9	Mar 2024 - Week 1	
10	Mar 2024 - Week 3	Project Testing
11	Apr 2024 - Week 1	
12	Apr 2024 - Week 3	Project Deployment
13	May 2024 - Week 1	
14	May 2024 - Week 3	Project summary and discussion

Table 1.1: Project Timeline

- Meetings are scheduled every two weeks to allow for sufficient work progress between sessions.
- The project timeline spans from September 2023 to May 2024.
- The project implementation phase begins in November 2023, with a focus on specific modules to ensure a manageable and iterative approach. Then all of the features are being implemented in 6 weeks, from February 2024 to March 2024 - week 2.
- The final meeting in May 2024 includes a summary of the Project and a discussion to set the future improvement of the project.

1.3.13. Regulatory Compliance:

The system will comply with relevant data protection and privacy regulations to ensure the secure handling of sensitive user information.

1.3.14. Quality Standards:

The system will adhere to industry-standard coding practices, usability standards, and security best practices.

1.3.15. Training and Support:

Training materials and support mechanisms will be provided for students, faculty, and businesses to ensure effective utilization of the system.

1.4. Some definitions

- **Internship Program:** An organized educational or professional initiative that provides students or individuals with practical work experience in a specific field or industry. These programs are typically designed to help participants bridge the gap between academic learning and real-world applications.
- **Job:** A specific position of employment in an organization or company, where an individual is hired to perform a set of tasks or responsibilities in exchange for compensation, typically wages or a salary.
- **Faculty Staff:** The academic professionals employed by educational institutions, including universities and colleges, responsible for teaching, research, and various administrative duties related to the institution's educational programs.
- **Faculty Admin** refers to the management and coordination of various administrative tasks and responsibilities within an academic institution, particularly those that pertain to the faculty members. This encompasses a wide range of duties, including but not limited to faculty recruitment and onboarding, scheduling of classes, curriculum development, academic policies and regulations, faculty evaluations, and overseeing faculty support services.
- **Company Staff/HR:** The employees within a company or organization, including the Human Resources (HR) department, responsible for various functions, such as recruitment, employee management, training, and overseeing the welfare of the workforce.
- **Students:** Individuals enrolled in educational institutions, such as schools, colleges, or universities, who are pursuing academic studies and working towards the completion of their degrees or certifications.
- **Internship Student:** A student who participates in an internship program to gain practical, hands-on experience in a specific industry or field. Internship students work under the guidance of professionals to apply theoretical knowledge in

real-world situations. In this project, the internship student will have some more aspect such as

- **Foundation Test:** A standardized examination or assessment that evaluates a student's knowledge and skills in a particular subject or field. These tests are often used for academic or professional purposes and can serve as a foundation for further learning or career development. In this project, this test will be compulsory for the internship students before the real internship begins.

2. Analyze Current Systems

2.1. Current internship process

2.1.1. Recruitment process of students

1. Students register the internship course with the training department. After successfully registering, students receive the account for login to the CSE Internship website.
2. Students browse for internships from jobs posted on the CSE Internship website or self-find jobs from companies (will be on the website later).
3. Students personally contact companies for the recruiting process.
4. Students register the internship position on the website.
5. Students see the recruiting results on the website and in a separate file provided by faculty staff.
6. After finishing the internship course, students make an internship report. Results will be posted on a file provided by faculty staff.

2.1.2. Recruitment process of companies

1. Companies contact faculty to register an account for posting jobs.
2. Companies write internship programs and send them to faculty for verification.
3. Companies open positions for students to find and register for internships.
4. Companies complete the recruitment process with students.
5. Companies verify students' registration for the positions.
6. Companies send recruiting results to faculty.
7. After the internship course, companies send internship results to faculty. The result will be posted on a file.

2.1.3. Management process of faculty staff

1. Create accounts for students who successfully registered for an internship course with the training department.
2. Receive, verify companies information, and create accounts.

3. Verify internship programs from companies and post them to the website.
4. Receive and update recruiting results to a file.
5. Receive and update internship results to a file.
6. Receive and update internship reports from students.

2.1.4. Inefficiencies in the current internship process

- Student:
    - Lack of internship opportunities, due to recruitment information is limited and scattered.
    - Tracking recruitment results, internship results are not easy due to separated documents.
    - Internship report submission process must be done manually, which can lead to missing and hard to track results.
- Company:
    - Recruiting process is complex and requires many documents to be fulfilled.
    - Submitting internship results is prone to errors due to lack of contact and consistency with admitted students.
    - Many paper documents are needed to handle and easy to get lost during the submission process.
- Faculty staff:
    - Have to manage a lot of information, such as student information, business information, and upload them to the website or files manually.
    - Receiving and managing too many paper documents and uploading them manually, slows down the recruiting process and is prone to errors.

2.2. Internship website of the Faculty of Computer Science and Engineering

2.2.1. About CSE Internship website

The Internship website of the Faculty of Computer Science and Engineering (CSE) is currently the platform for CSE students to find internship opportunities and view internship recruiting results. This website is also the platform for companies to upload internship jobs in order to find the suitable candidates. The website is managed by CSE faculty staff.

2.2.2. Website features

- Company representatives can create accounts and post internship jobs (after being verified by faculty staff).
- Logo of companies who posted jobs are displayed in the home page of the website for students to browse and apply.
- Companies can choose among registered students for the position to proceed or reject.
- Students can login with a provided account to register for internship positions that are posted by companies.
- Students can manage their information displayed for companies human resources to view.
- Faculty staff can modify jobs posted on the website by companies and manage students' information.

2.2.3. Advantages of the website

- The system effectively serves as a reliable reference for students by storing and displaying companies' information.
- The information on the website is closely monitored and rigorously managed to ensure that it remains accurate and reliable.

- The website displays the number of students a company is looking to recruit and how many they have already hired, helping students understand the recruitment status of various companies.

2.2.4. Disadvantages of the website

- Many tasks need a staff to complete, such as verifying company and student information or managing the number of students for a certain position.
- The website works as a place to manage internship students and jobs that students applied to rather than a place for students to find internship opportunities and for companies to post jobs.
- Recruitment results are handled separately in a folder containing files that are sent from companies to staff and uploaded manually.
- Internship recruitment and the working process mostly occurs outside of the website so the management is not efficient but costly and time consuming.
- Not providing a means of communication, every communication is done outside the website.
- The website only displays and supports search for company names, hence students cannot search and filter internship jobs that are suitable for them.
- The previous semester's internship positions were deleted after the internship period ended, so students do not have a reference source for current and future semester internship positions.

2.3. University of Economics HCMC (UEH) Department of Student Affairs (DSA) Career site

2.3.1. About DSA and DSA Career site

The Department of Student Affairs (DSA) at UEH is a vital hub for students, offering comprehensive support and information services. DSA's multifaceted role includes facilitating student engagement, providing academic and career guidance, managing scholarship programs, and ensuring the well-being of all students, making it an indispensable resource within the university community.

Career site is the subpage of DSA Website, where job postings from various companies are posted. With a continuous accumulation of job opportunities since 2020, the number of jobs posted on the page has now reached over 1000. In the scope of this project, we only focus on analyzing the Career site.

2.3.2. Website features

- Main page of the website including a list of jobs posting from different companies, sorted with the posting time.
- For each job, the job detail is displayed in separate pages.

2.3.3. Advantages of the website

- Jobs are displayed with a logo, header, and a brief description for easy skimming.
- For each job, the information is displayed in separate pages with synchronized sections such as Job Title, Number of Openings, and Candidate Requirements, optimizing the display and searchability.
- Job positions accumulate over the years, serving as a valuable reference for students regarding current and future market demands.

2.3.4. Disadvantages of the website

- With the large number of jobs posted on the website, transparency and quality of companies may not be fully controlled or need a huge amount of effort to manage.
- Number of employees applying for a job is not displayed makes it difficult for users to track the current recruitment status of a job.
- Companies posting jobs individually for each position and for each year can make it challenging to track recruitment updates for a specific company.

2.4. LinkedIn - Social networking site for business community

2.4.1. About LinkedIn

LinkedIn is a professional networking platform that connects individuals and businesses worldwide. It provides a digital space for users to build their online resumes, connect with colleagues, industry peers, and potential employers, and share industry insights. LinkedIn is a valuable resource for job seekers, recruiters, and professionals looking to foster meaningful business connections and stay updated on industry trends. It has become an essential tool for career development and networking in the digital age.

2.4.2. Website features

- Users can create and customize their professional profiles, including work experience, skills, education, and contact information.
- Connect with colleagues, peers, and industry professionals to expand professional networks.
- Search for job listings, apply for positions, and receive job recommendations based on profile.
- Businesses can create and manage company pages to showcase their brand, post updates, and engage with followers.
- Provided a news feed to stay updated with relevant industry news, posts from connections, and company updates.
- Monitor the status of job applications and manage job search.

2.4.3. Advantages of website

- A powerful platform for connecting users with businesses.
- Users have the freedom to customize their personal profiles to make a strong impression on potential employers.
- Create numerous opportunities for users to find suitable jobs and for companies to find the best candidates.

- Efficiently manage the recruitment application process as well as candidate profiles.

2.4.4. Disadvantages of website

- Being a social media platform, controlling users and content is a significant concern.
- Lack of transparency and clear verification of job postings can lead to various issues for users.
- The publication of personal information that can be viewed by many people raises concerns about personal data privacy.
- A large management team is necessary to ensure the website remains stable and functions effectively.
- An excessive amount of job information can lead to information overload and make it time-consuming to find a suitable job.

3. Requirement Elicitation

3.1. Relevant stakeholders

- **Faculty Staff:** The academic professionals employed by educational institutions, including universities and colleges, responsible for teaching, research, and various administrative duties related to the institution's educational programs.
- **Faculty Admin** refers to the management and coordination of various administrative tasks and responsibilities within an academic institution, particularly those that pertain to the faculty members. This encompasses a wide range of duties, including but not limited to faculty recruitment and onboarding, scheduling of classes, curriculum development, academic policies and regulations, faculty evaluations, and overseeing faculty support services.
- **Company Staff/HR:** The employees within a company or organization, including the Human Resources (HR) department, responsible for various functions, such as recruitment, employee management, training, and overseeing the welfare of the workforce.
- **Students:** Individuals enrolled in educational institutions, such as schools, colleges, or universities, who are pursuing academic studies and working towards the completion of their degrees or certifications.

3.2. Expected jobs

- Faculty staffs
  1. Have an overview of the students details (name, studentId, email, etc)
  2. Have an overview of the companies details that registered to the system (name, taxId, email, address, etc)
  3. Have an overview of the jobs registered in the system.
  4. Can search for the jobs by criteria.
  5. Can approve or reject the students account's legitimate status.
  6. Can approve or reject the companies which register their account to the system.

7. Can access and modify the information of the student or the company in the system.
8. Can search and see the details of other staff members.
9. Be able to send a notification to both students and companies.
10. Can approve or reject the request for internship of the students.
11. Can receive, approve or reject the internship result of the students from the companies.

- Faculty admins
  1. Have an overview of the students details (name, studentId, email, etc)
  2. Have an overview of the companies details that registered to the system (name, taxId, email, address, etc)
  3. Have an overview of the jobs registered in the system.
  4. Can search for the jobs by criteria.
  5. Can approve or reject the students account's legitimate status.
  6. Can approve or reject the companies which register their account to the system.
  7. Can access and modify the information of the student or the company in the system.
  8. Can search and see the details of other staff members.
  9. Be able to send a notification to both students and companies.
  10. Can approve or reject the request for internship of the students.
  11. Can receive, approve or reject the internship result of the students from the companies.
  12. Can manage (add, edit and remove) staff accounts.
- University students
  1. Can search for the jobs by criteria.
  2. Can search the company based on name, address, industry field, etc.
  3. Can update and modify their own information.
  4. Can request for the approval for the internship period from the staff.

5. Can apply for the internship position from the company.
6. Have the overview tracking of the internship approval process.
7. Have the overview tracking of the application for the position from the company.

- Company human resources

1. Can register a new account on the website.
2. Can post and public the job opportunities into the website.
3. Can search and receive the potential candidates.
4. Can update and modify their own jobs position information.
5. Can update and modify their own company account information.
6. Have an overview for each jobs status (the number of approval, pending, applied candidates, etc)
7. Can download the information PDF file.

3.3. Functional requirements

- Faculty staffs
  1. View the information of students and companies.
  2. Login by Google.
  3. Adjust and modify the information of students and companies.
  4. View the statistics of jobs in the system.
  5. View the detailed information of each job in the system.
  6. Get the information of each company, student and job.
  7. Send notifications to students and companies.
  8. Adjust and modify the process of students when applying for internships.
- Faculty admins
  1. View the information of students and companies.
  2. Login by Google.
  3. Adjust and modify the information of students and companies.
  4. View the statistics of jobs in the system.
  5. View the detailed information of each job in the system.
  6. Get the information of each company, student and job.
  7. Send notifications to students and companies.
  8. Adjust and modify the process of students when applying for internships.
  9. Create and adjust the staff accounts.
- University students
  1. Get the information about each job.
  2. Login by Google.
  3. Get the information of each company.
  4. Get the overview of the application process.
  5. Register an internship approval process to staff.
- Company human resources
  1. Create a new account on the system.

2. Create a new job position on the system.
3. Get the overview information of potential students.
4. Get the overview of each job.
5. Export the information of the job into a PDF file

3.4. Non-functional requirements

3.4.1. Usability requirement

- Companies and students can easily use basic functions immediately.
- The faculty staff can use it after 10 minutes of training.
- Interface in English with the opportunities to support multiple languages.

3.4.2. Performance requirement

- The system must respond in less than 30 seconds. After 30 seconds, the system will notify the user.
- Can handle real-time data.
- Allow at least 50 users access at the same time without crashing.
- The user's information has to be updated in 2 minutes when the user updates it.

3.4.3. Space requirement

- The system can handle 100 concurrent visits without any effect on the system efficiency.
- The system should be able to handle data from at least 200 users at the moment and 1000 users in five years.

3.4.4. Availability requirement

- Working time of the system all days of the week, from 6am to 10pm.
- The system should not be down more than 5 minutes continuously, maximum 30 minutes a day.

3.4.5. Environmental requirement

- The system can run on an internet browser on multi-device.

3.4.6. Security requirement

- Password required when login or changing the information
- Encryption with BCRYPT.
- Ensure not leaking the user information.
- Authentication based on JWT (Java Web Token).

3.4.7. Operational requirements

- System data is backed up every month to prevent data loss.

4. UML Diagrams for System

4.1. Use-case diagrams & Scenario

4.1.1. Use-case diagram

In this system, 3 primary stakeholders can be identified: student, faculty staff, and company staff. The system provides some use cases that help each stakeholder achieve their goals in managing and finding jobs. As a student, he or she can log into the system by using their school Google account, and then perform different actions such as browse for jobs and apply for those jobs, as well as manage applied jobs, or contact with companies. Similar to companies, company staff can create an account and start working with the system to complete their needs such as post a job, find potential employees, manage internship processes. In order to help students and company staff to achieve their goals, faculty staff is needed to complete some tasks such as managing users, verifying information or creating accounts. These tasks must be done easily by using some use cases provided by the system.

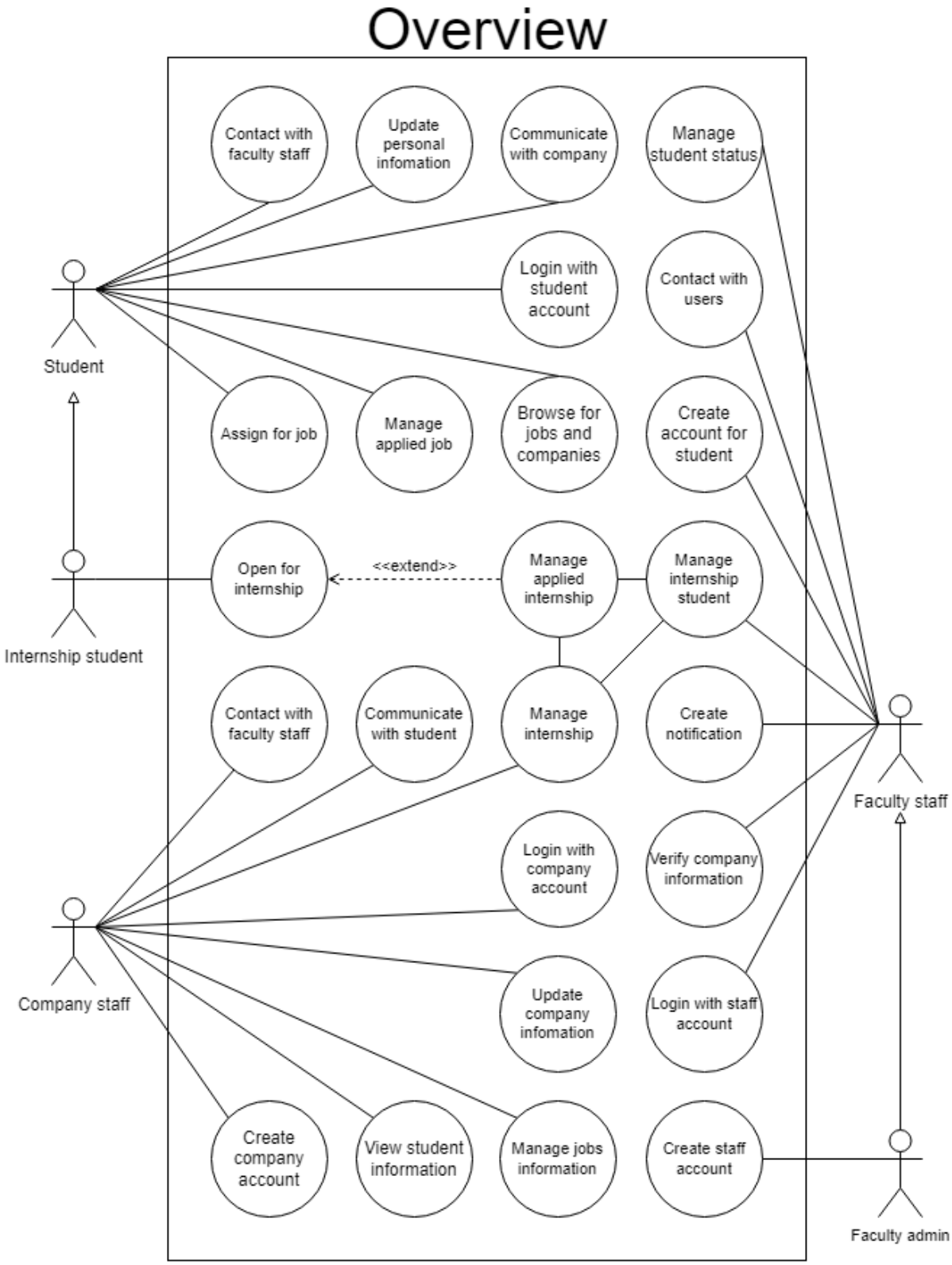


Figure 4.1: Overview use-case diagram

4.1.2. Use-case description

Login with student account

Use Case Name	Login with student account
Actor	Student
Description	This use case allow student to login to the system with provided student account
Pre-condition	None
Post-condition	Logged into the system
Normal flow	1. Student click the button "Login" displayed on the screen 2. Student choose "Login with Google" 3. Display a screen for student to login with Google account 4. Verify account 5. Navigate to the next screen
Alternative flow	None
Exception flow	At step 4, if the logged in account is not a student account, raise an error. Use case end.

Table 4.1: Use-case Scenario: Login with student account

Update personal Information

Use Case Name	Update personal Information
Actor	Student
Description	This use case allow student to update personal information
Pre-condition	Student have logged in, viewing personal information
Post-condition	Successfully updated personal information
Normal flow	1. Student click "Update information" button 2. Student modify personal information 3. Click "save" button 4. System display a confirmation box 5. Student click "Yes" 6. Navigate back to the information page with updated information
Alternative flow	5a. Student click "No" 6a. Continue to modify information
Exception flow	Exception 1: At step 3, if students enter information with incorrect format, raise error, turn back to step 2. Exception 2: At any step, if a student closes the tab, display alert "Leave the page? Your change has not been saved yet."

Table 4.2: Use-case Scenario: Update personal Information

Apply for job

Use Case Name	Apply for job
Actor	Student
Description	This use case allow student to apply for a viewing job
Pre-condition	Student have logged in, viewing a job
Post-condition	Successfully applied for the job
Normal flow	1. Student click "Apply for this job" when viewing the job 2. Display a pop-up with pre-filled information 3. Student review information to submit 4. Student change submitting information if want to 5. Student click "Submit" button 6. System display confirmation box 7. Student click "Yes" 8. Navigate back to the job screen
Alternative flow	6a. Student click "No" 7a. Continue to review submitting information
Exception flow	Exception 1: At step 4, if students enter information with incorrect format, raise error, turn back to step 2. Exception 2: At any step, if a student closes the tab, display alert "Leave the page? Your application will not be saved."

Table 4.3: Use-case Scenario: Apply for job

Manage applied job

Use Case Name	Manage applied job
Actor	Student
Description	This use case allow student to view and modify applied jobs
Pre-condition	Student have logged in
Post-condition	View and modify applied jobs
Normal flow	1. Student click "View my applications" 2. Display list of applied jobs information 3. Student choose a job to view detailed information of an applied job 4. Display a pop-up window with detailed information of the applied job 5. Student click "Cancel application" button 6. Display a confirmation screen 7. Student click "Yes" 8. Delete the application and navigate back to applied jobs information screen
Alternative flow	7a. Student click "No" 8a. Back to step 4
Exception flow	At any step, if a student closes the tab, end the use case.

Table 4.4: Use-case Scenario: Manage applied job

Contact with faculty staff

Use Case Name	Contact with faculty staff
Actor	Student
Description	This use case allow student to read and send message to faculty staff
Pre-condition	Student have logged in
Post-condition	Successfully send message to faculty staff
Normal flow	1. Student click "Support" button on the toolbar 2. Navigate to the "Support" screen 3. Student click "Contact the administrator" 4. Display a form to input message 5. Student type the message 6. Student click "send" button 7. Navigate back to "Support" screen
Alternative flow	None
Exception flow	Exception 1: At step 6, if the message is empty, "send" button is disabled Exception 2: At step 5, if the student closes the window, raise the alert "Your message has not been sent."

Table 4.5: Use-case Scenario: Contact with faculty staff

Communicate with company

Use Case Name	Communicate with company
Actor	Student
Description	This use case allow student to contact with company staff
Pre-condition	Student have logged in
Post-condition	Successfully view and send message to company staff
Normal flow	1. Student click message icon on the toolbar 2. Navigate to "Message" screen 3. Student view all conversation with companies 4. Student choose a conversation 5. Display all message in the chosen conversation 6. Student type new message 7. Student click "send" or hit "enter" 8. Send message to the company staff
Alternative flow	None
Exception flow	Exception 1: At step 7, if the message is empty, sending is not available Exception 2: At step 6, if student close the window, raise alert "Your message have not been sent"

Table 4.6: Use-case Scenario: Communicate with company

Open for internship

Use Case Name	Open for internship
Actor	Internship Student
Description	This use case allow student to open profile for companies to find the student information
Pre-condition	Student have logged in, currently in the profile page Student is registered for internship course
Post-condition	Successfully open profile for company to find the student information
Normal flow	1. Student click "Open for internship" button 2. Display a pop-up window that tells about risk of open profile 3. Student click "I understand" 4. Successfully open profile for companies to find the student information
Alternative flow	3a. Student click "Cancel" 4a. Close the window
Exception flow	At any step, if a student closes the tab, end the use case.

Table 4.7: Use-case Scenario: Open for internship

Login with company account

Use Case Name	Login with company account
Actor	Company staff
Description	This use case allow company staff to login to the system with registered and verified account
Pre-condition	None
Post-condition	Logged into the system
Normal flow	1. Company staff click the button "Login" displayed on the screen 2. Company staff enter email and password 4. Verify login information 5. Navigate to the next screen
Alternative flow	None
Exception flow	Exception 1: At step 4, if email and password not registered, raise error "account not registered" Exception 2: At step 4, if password not correct, raise error "incorrect password"

Table 4.8: Use-case Scenario: Login with company account

Communicate with student

Use Case Name	Communicate with student
Actor	Company staff
Description	This use case allow student to contact with company staff
Pre-condition	Company staff have logged in
Post-condition	Successfully view and send message to student
Normal flow	1. Company staff click message icon on the toolbar 2. Navigate to "Message" screen 3. Company staff view all conversation with students 4. Company staff choose a conversation 5. Display all message in the chosen conversation 6. Company staff type new message 7. Company staff click "send" or hit "enter" 8. Send message to the student
Alternative flow	None
Exception flow	Exception 1: At step 7, if the message is empty, sending is not available Exception 2: At step 6, if company staff close the window, raise alert "Your message have not been sent"

Table 4.9: Use-case Scenario: Communicate with student

View student information

Use Case Name	View student information
Actor	Company staff
Description	This use case allow company staff to view information of the chosen student
Pre-condition	Company staff have logged in
Post-condition	View student information
Normal flow	1. Company staff click "View student information" 2. Display the chosen information
Alternative flow	None
Exception flow	None

Table 4.10: Use-case Scenario: View student information

Update company information

Use Case Name	Update company information
Actor	Company staff
Description	This use case allow company staff to update company information
Pre-condition	Company staff have logged in, viewing company information
Post-condition	Company staff updated company information
Normal flow	1. Company staff click "Update information" button 2. Company staff modify company information 3. Click "save" button 4. System display a confirmation box 5. Company staff click "Yes" 6. Navigate back to the information page with updated information
Alternative flow	5a. Company staff click "No" 6a. Continue to modify information
Exception flow	Exception 1: At step 3, if company staff enter information with incorrect format, raise error, turn back to step 2. Exception 2: At any step, if Company staff close the tab, display alert "Leave the page? Your change has not been saved yet."

Table 4.11: Use-case Scenario: Update company information

Manage jobs information

Use Case Name	Manage jobs information
Actor	Company staff
Description	This use case allow company staff to view and modify jobs
Pre-condition	Company staff have logged in
Post-condition	View and modify jobs
Normal flow	1. company staff click "View all jobs" 2. Display list of created jobs information 3. Company staff choose a job to view detailed information 4. Display a pop-up window with detailed information of the created job 5. Company staff click "Delete job" button 6. Display a confirmation screen 7. Company staff click "Yes" 8. Delete the job and navigate back to jobs information screen
Alternative flow	Alternative flow 1: ● 7a. Company staff click "No" ● 8a. Back to step 4 Alternative flow 2: ● 7b. Company staff update information of the viewing job ● 8b. Company staff click "save" ● 9b. Display a confirmation box ● 10b. Company staff click "Yes" ● 11b. Back to step 4
Exception flow	Exception 1: At step 7b, if company staff close the window, display an alert "Leave the page? Your change have not been saved yet" Exception 2: At any steps, if company staff close the window, end the use case.

Table 4.12: Use-case Scenario: Manage jobs information

Contact with faculty staff

Use Case Name	Contact with faculty staff
Actor	Company staff
Description	This use case allow company staff to read and send message to faculty staff
Pre-condition	Company staff have logged in
Post-condition	Successfully send message to faculty staff
Normal flow	1. Company staff click "Support" button on the toolbar 2. Navigate to the "Support" screen 3. Company staff click "Contact the administrator" 4. Display a form to input message 5. Company staff type the message 6. Company staff click "send" button 7. Navigate back to "Support" screen
Alternative flow	None
Exception flow	Exception 1: At step 6, if the message is empty, "send" button is disabled Exception 2: At step 5, if the company staff close the window, raise the alert "Your message has not been sent."

Table 4.13: Use-case Scenario: Contact with faculty staff

Create company account

Use Case Name	Create company account
Actor	Company staff
Description	This use case allow company staff to create account for the company
Pre-condition	None
Post-condition	Successfully created an account for company
Normal flow	1. Company staff click "Sign up" button 2. Navigate to "Sign up" screen 3. Company staff enter required information 4. Company click "Submit" button 5. Display screen "Wait for your account to be verified"
Alternative flow	None
Exception flow	At step 4, if entered information is not valid, raise error, continue to step 3

Table 4.14: Use-case Scenario: Create company account

Login with staff account

Use Case Name	Login with staff account
Actor	Faculty staff
Description	This use case allow faculty staff to login to the system with provided staff account
Pre-condition	None
Post-condition	Logged into the system
Normal flow	1. Faculty staff click the button "Login" displayed on the screen 2. Faculty staff choose "Login with Google" 3. Display a screen for staff to login with Google account 4. Verify account 5. Navigate to the next screen
Alternative flow	None
Exception flow	At step 4, if the logged in account is not a faculty staff account, raise an error. Use case end.

Table 4.15: Use-case Scenario: Login with staff account

Mange student status

Use Case Name	Mange student status
Actor	Faculty staff
Description	This use case allow faculty staff to modify internship status of students
Pre-condition	Logged in with faculty staff account
Post-condition	Successfully updated student internship status
Normal flow	1. Faculty staff navigate to "Internship student" page 2. Display the internship student list 3. Faculty staff click the button "Update with file" 4. Open a box for faculty staff to choose a file 5. Display the input data 6. Faculty staff click "Update" button 7. System update the information 8. Navigate back to the internship student list
Alternative flow	3a. Faculty staff click "Update" button next to the student information 4a. Display an update form 5a. Faculty staff modify the information 6a. Faculty staff click "save" 7a. System update the information 8a. Navigate back to the internship student list
Exception flow	At step 5, if the provided file is not correct, raise error, back to step 2

Table 4.16: Use-case Scenario: Mange student status

Contact with users

Use Case Name	Contact with users
Actor	Faculty staff
Description	This use case allows faculty staff to read and send messages to other users.
Pre-condition	Faculty staff have logged in
Post-condition	Successfully send messages to desired users.
Normal flow	1. Staff click "Message" button on the toolbar 2. The message dialog pop-up. 3. Staff click to the desired message to view or response 4. Display a form to input message 5. Staff type the message 6. Staff click "send" button 7. Navigate back to the "Message" dialog.
Alternative flow	3a. Faculty staff click on a new message. 4a. Display a search input to find the desired user. 5a. Faculty staff click on that user. 6a. Display a form to input message 7a. Staff type the message 8a. Staff click "send" button 9a. Navigate back to the "Message" dialog.
Exception flow	Exception 1: At step 6 or step 8a, if the message is empty, "send" button is disabled Exception 2: At step 5, if the faculty staff close the window, raise alert "Your message has not been sent."

Table 4.17: Use-case Scenario: Contact with users

Verify company information

Use Case Name	Verify company information
Actor	Faculty staff
Description	This use case allow faculty staff to read and verify company verification request
Pre-condition	Faculty staff have logged in
Post-condition	Successfully accept or decline the verification request from company
Normal flow	1. Staff click "Verify" tab on the side bar 2. Display page "Verify" with a pending list 3. Staff look for the pending verifications 4. Staff click "View information" button at the desired company's request 5. Display a pop-up window that display request form of the company 6. Staff click "check" icon 7. Successfully verify information of the company, navigate back to the pending list
Alternative flow	6a. Staff click "cross" icon 7a. Successfully declined company's request, navigate back to the pending list
Exception flow	None

Table 4.18: Use-case Scenario: Verify company information

View notification

Use Case Name	View notification
Actor	Faculty staff
Description	This use case allows faculty staff to view notification.
Pre-condition	Faculty staff have logged in
Post-condition	View notification.
Normal flow	1. Faculty staff click the notification icon on the toolbar. 2. Display the notification.
Alternative flow	None
Exception flow	None

Table 4.19: Use-case Scenario: View notification

Create notification

Use Case Name	Create notification
Actor	Faculty staff
Description	This use case allows faculty staff to create notification.
Pre-condition	Faculty staff have logged in
Post-condition	Notification create successfully
Normal flow	1. Faculty staff click the notification icon on the toolbar. 2. Display the notification. 3. Faculty staff click in "new" icon 4. Display a form to input the notification. 5. Faculty staff click "Confirm". 6. Close the form and navigate to the notification page.
Alternative flow	5a. Faculty staff click "Cancel" 6. Close the form and navigate to the notification page.
Exception flow	Exception 1: At step 4, if the message is empty, "Confirm" button is disabled Exception 2: At step 5, if the faculty staff close the window, raise alert "Your notification has not been created."

Table 4.20: Use-case Scenario: Create notification

Create account for student

Use Case Name	Create account for student
Actor	Faculty staff
Description	This use case allow faculty staff to create account for students
Pre-condition	Faculty staff have logged in
Post-condition	Successfully created account for students
Normal flow	1. Faculty staff navigate to "Student" page 2. Display the student list 3. Faculty staff click the button "Create with file" 4. Open a box for faculty staff to choose a file 5. Display the input data 6. Faculty staff click "Create" button 7. System create accounts 8. Navigate back to the student list
Alternative flow	3a. Faculty staff click "Create" button 4a. Display a create form 5a. Faculty staff fill in the information 6a. Faculty staff click "Create" 7a. System create the account 8a. Navigate back to the student list
Exception flow	At step 5, if the provided file is not correct, raise error, back to step 2

Table 4.21: Use-case Scenario: Create account for student

Create staff account

Use Case Name	Create staff account
Actor	Faculty admin
Description	This use case allow faculty admin to create account for faculty staff
Pre-condition	Faculty admin has logged in
Post-condition	Successfully created an account for faculty admin
Normal flow	1. Faculty staff navigate to "Staff" page 2. Display the staff list 3. Faculty staff click "Create" button 4. Display a create form 5. Faculty staff fill in the information 6. Faculty staff click "Create" 7. System create the account 8. Navigate back to the staff list
Alternative flow	None
Exception flow	At step 5, if the faculty staff closes the window, raise the alert "The account has not been created."

Table 4.22: Use-case Scenario: Create staff account

4.2. Sequence diagrams

4.2.1. Staff and student login

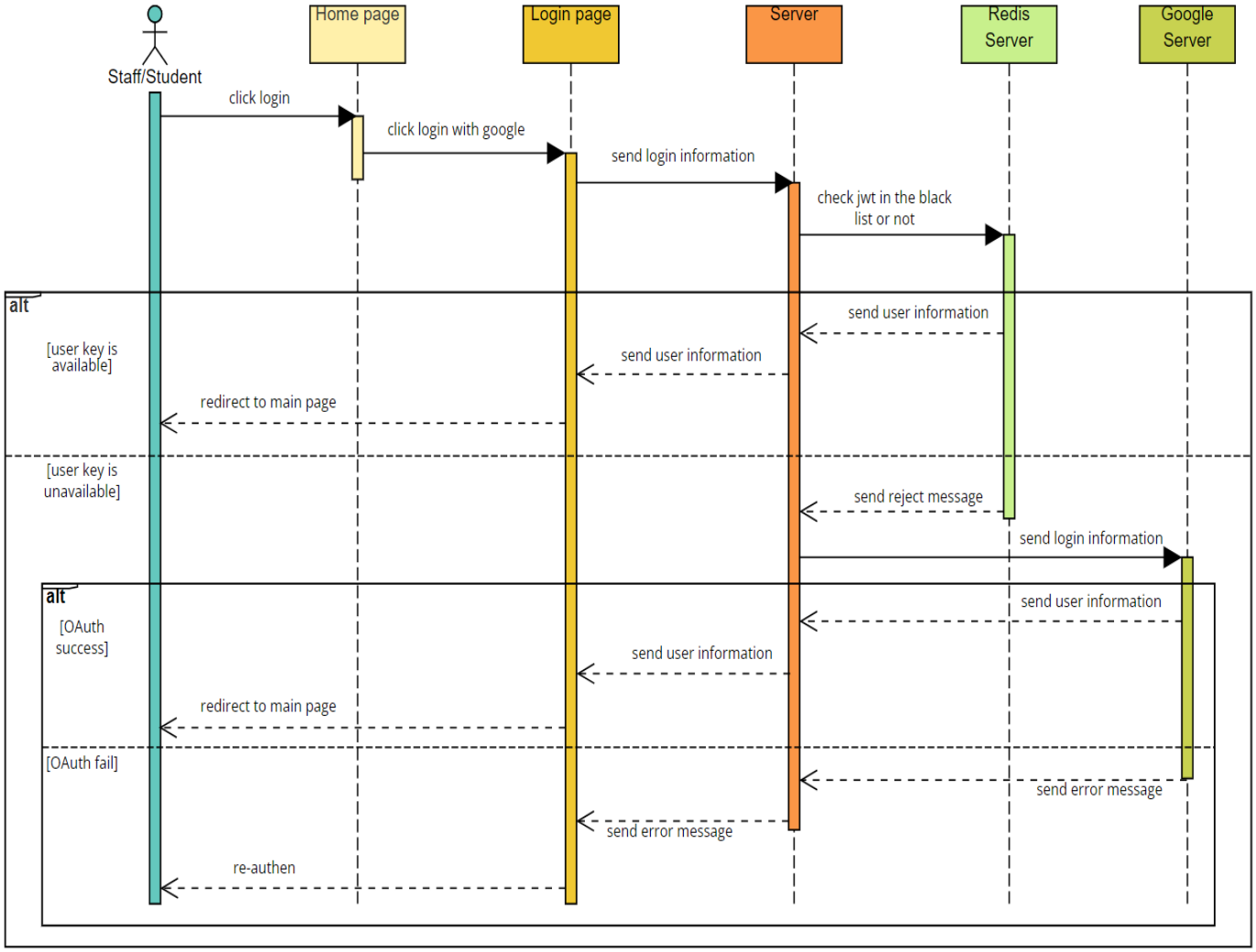


Figure 4.2: Staff and student login sequence diagram

4.2.2. Company login

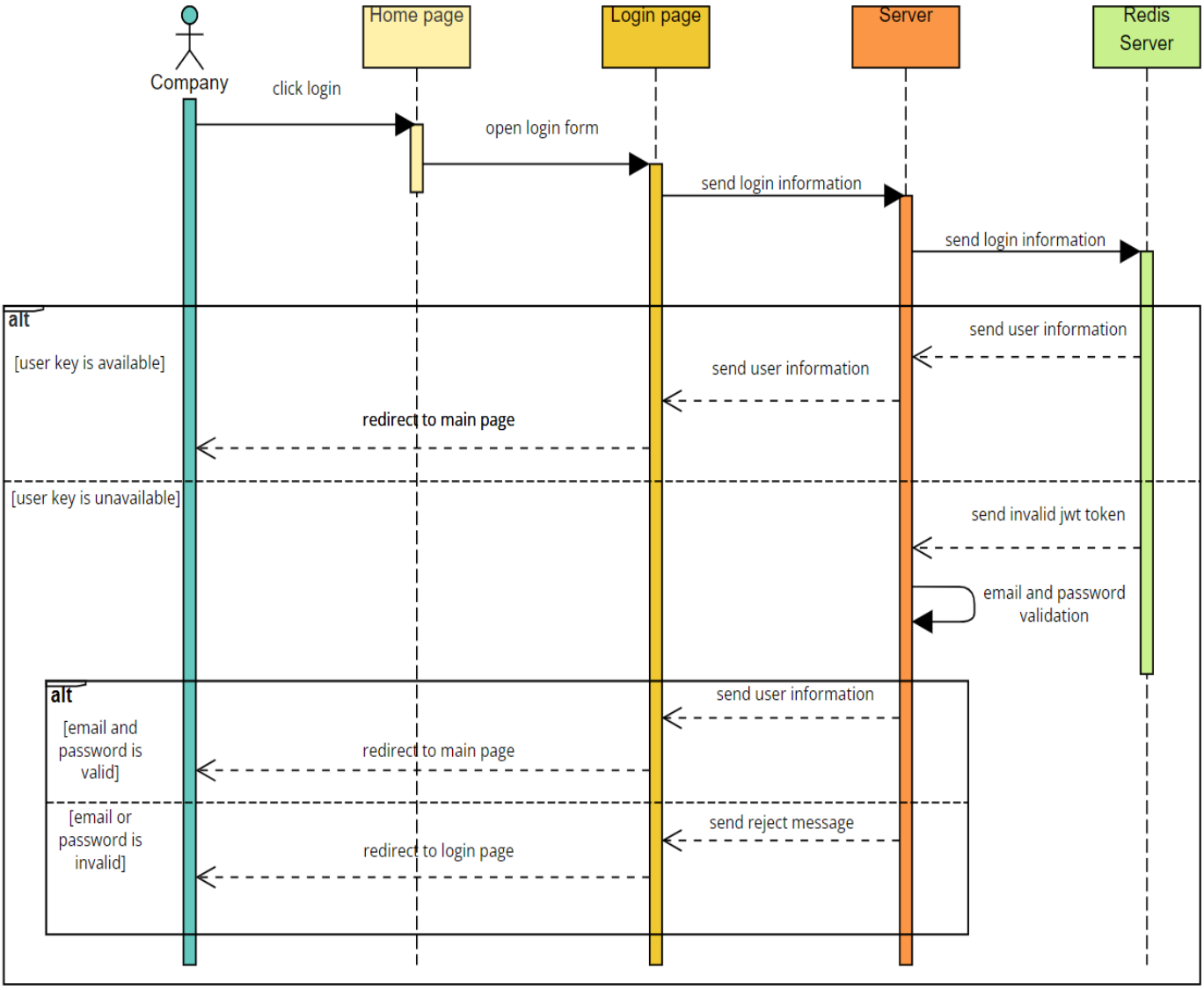


Figure 4.3: Company login sequence diagram

4.2.3. Company's account register process

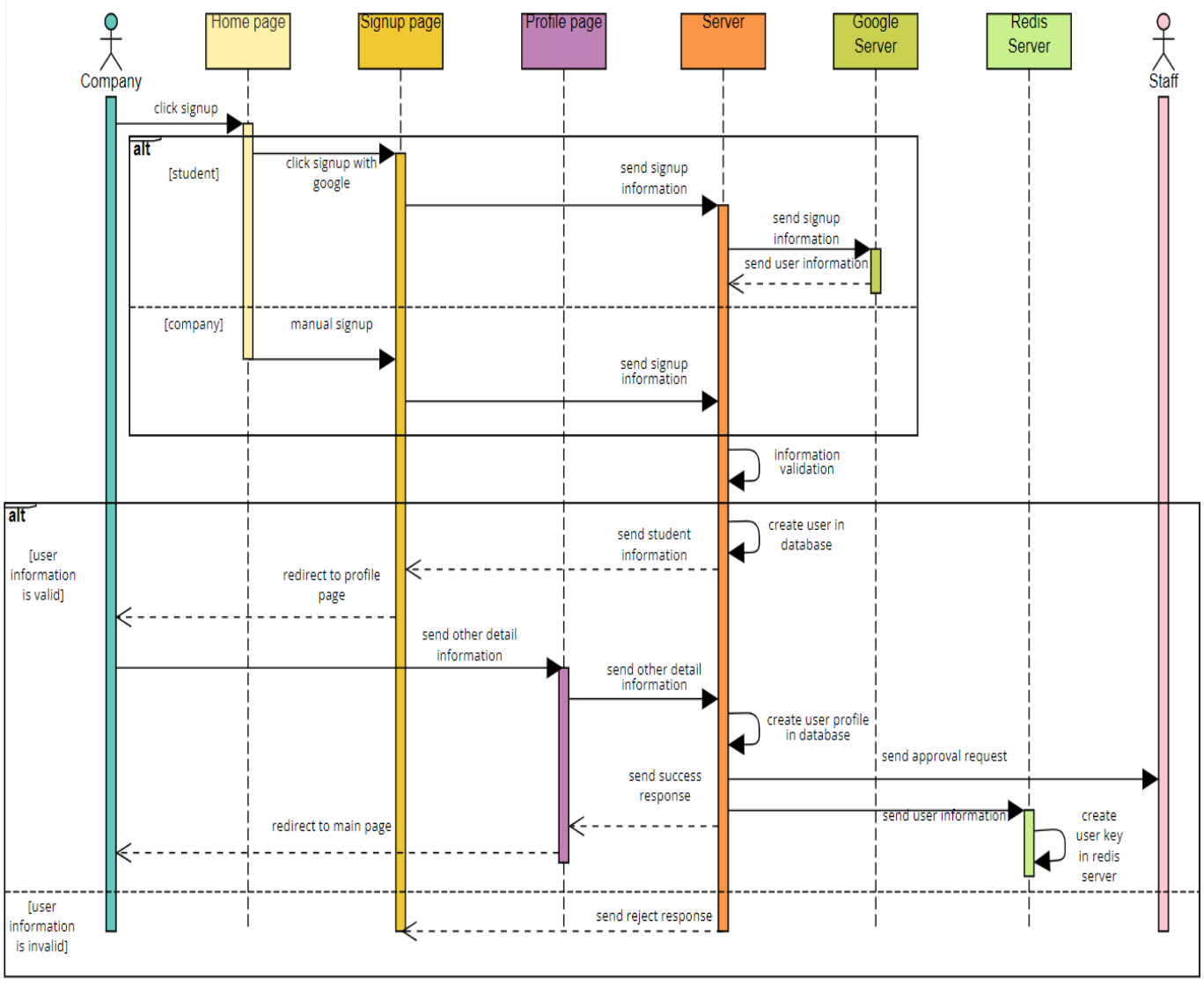


Figure 4.4: Company signup process sequence diagram

4.2.4. Create new job/internship opportunities.

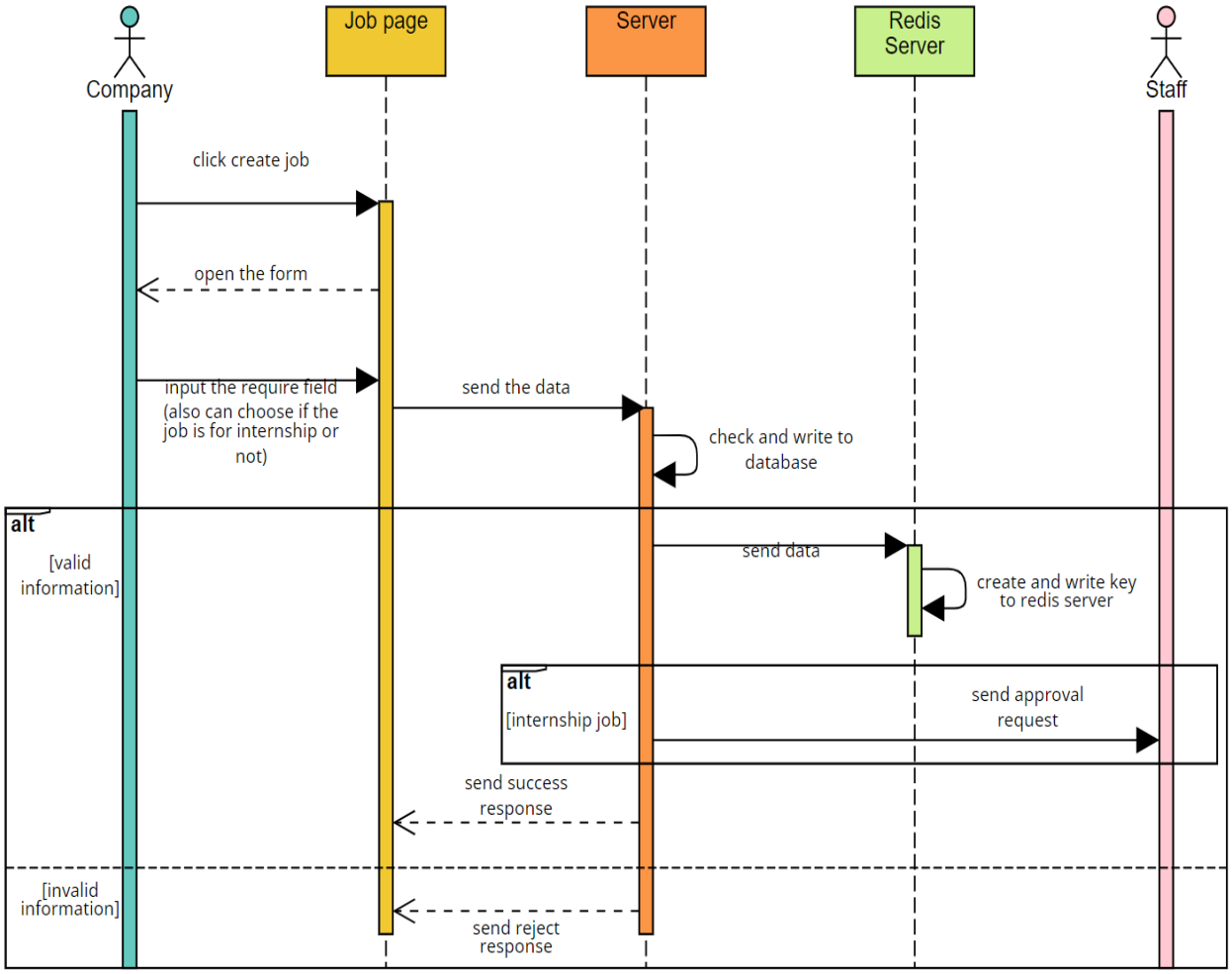


Figure 4.5: Job and internship create sequence diagram

4.2.5. Staff management for account and approval process

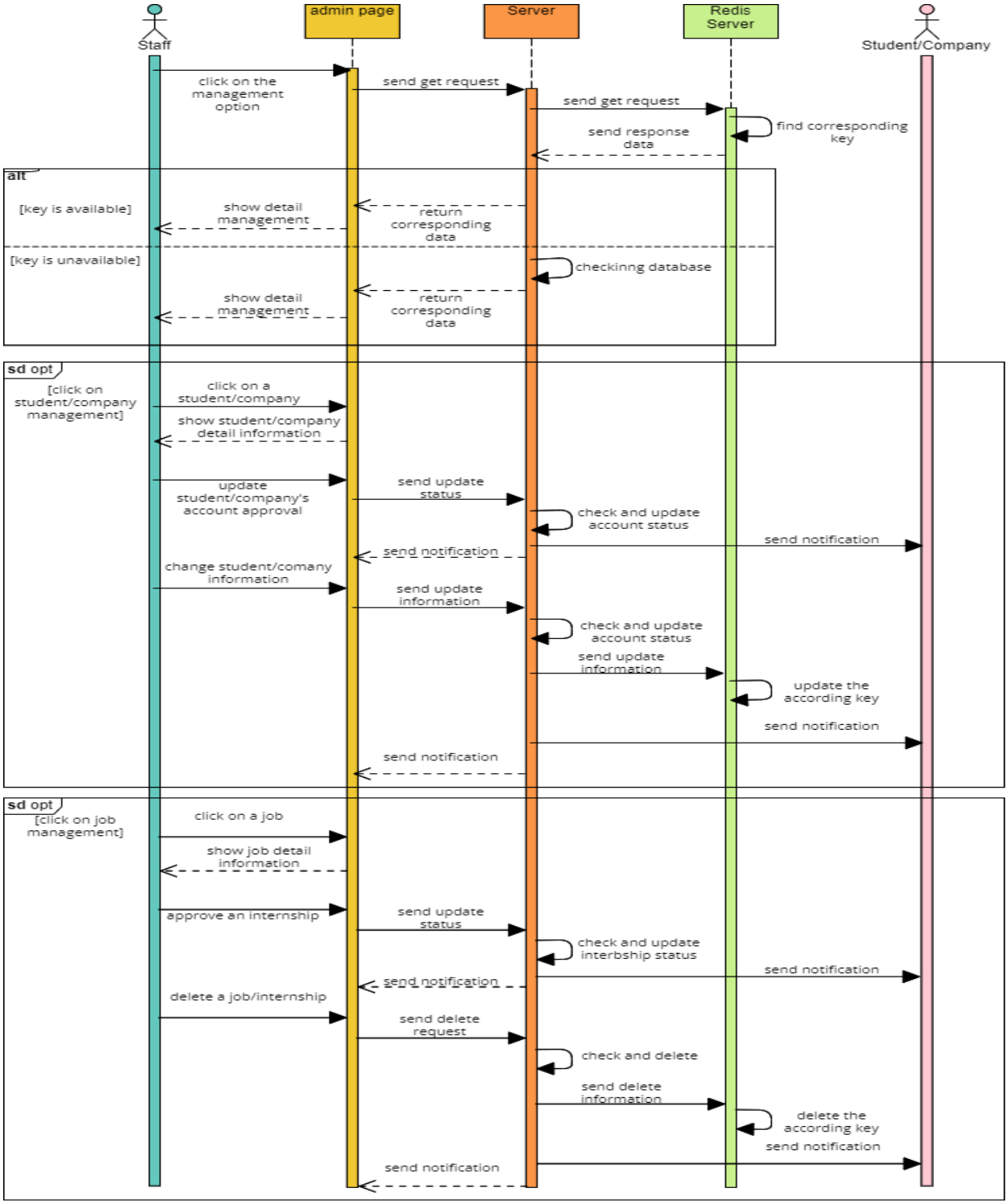


Figure 4.6: Staff management for account and approval process sequence diagram

4.2.6. Student account management by dataset uploading.

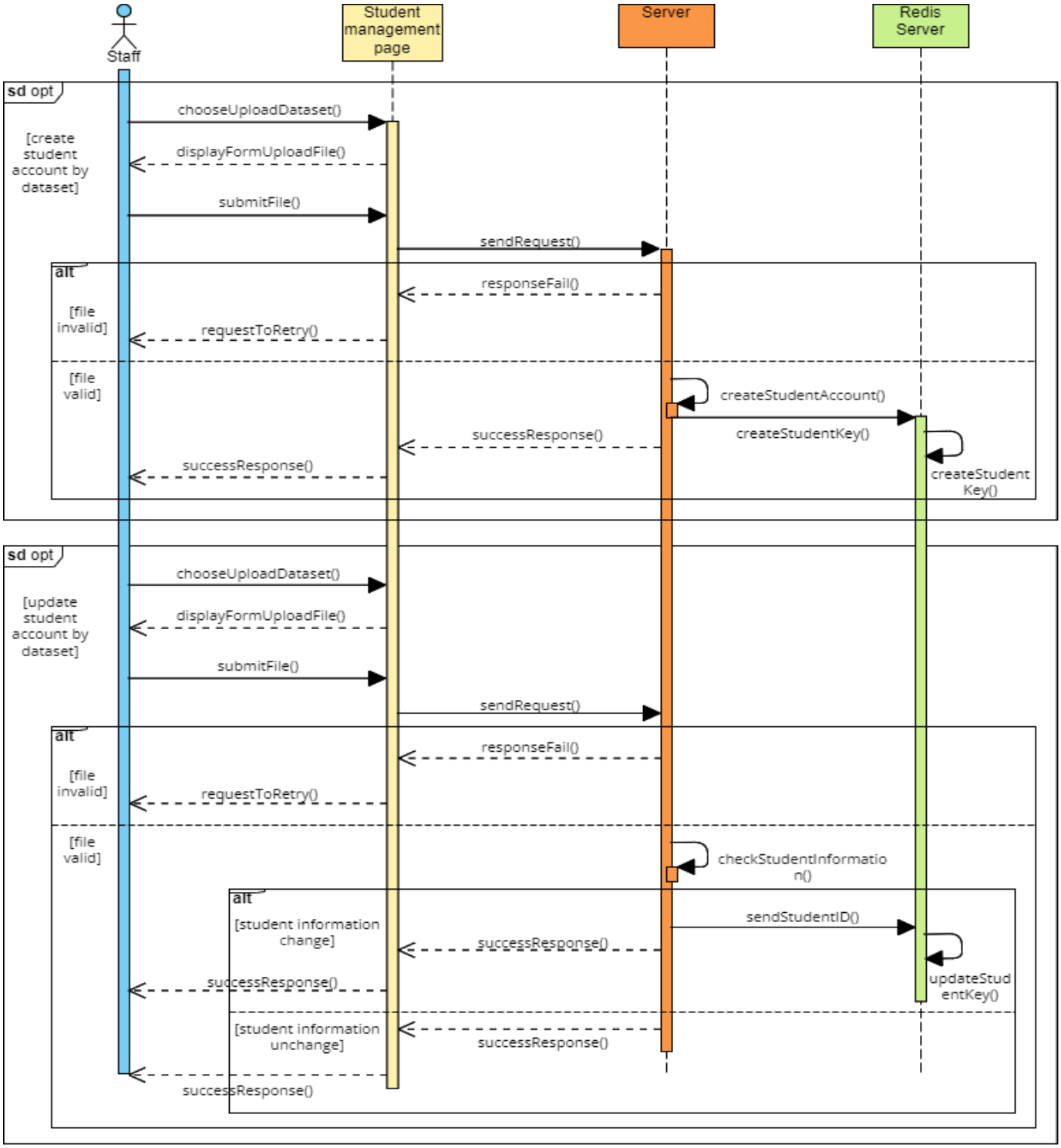


Figure 4.7: Student account management by dataset uploading sequence diagram

4.3. Activity diagrams

4.3.1. Student Login

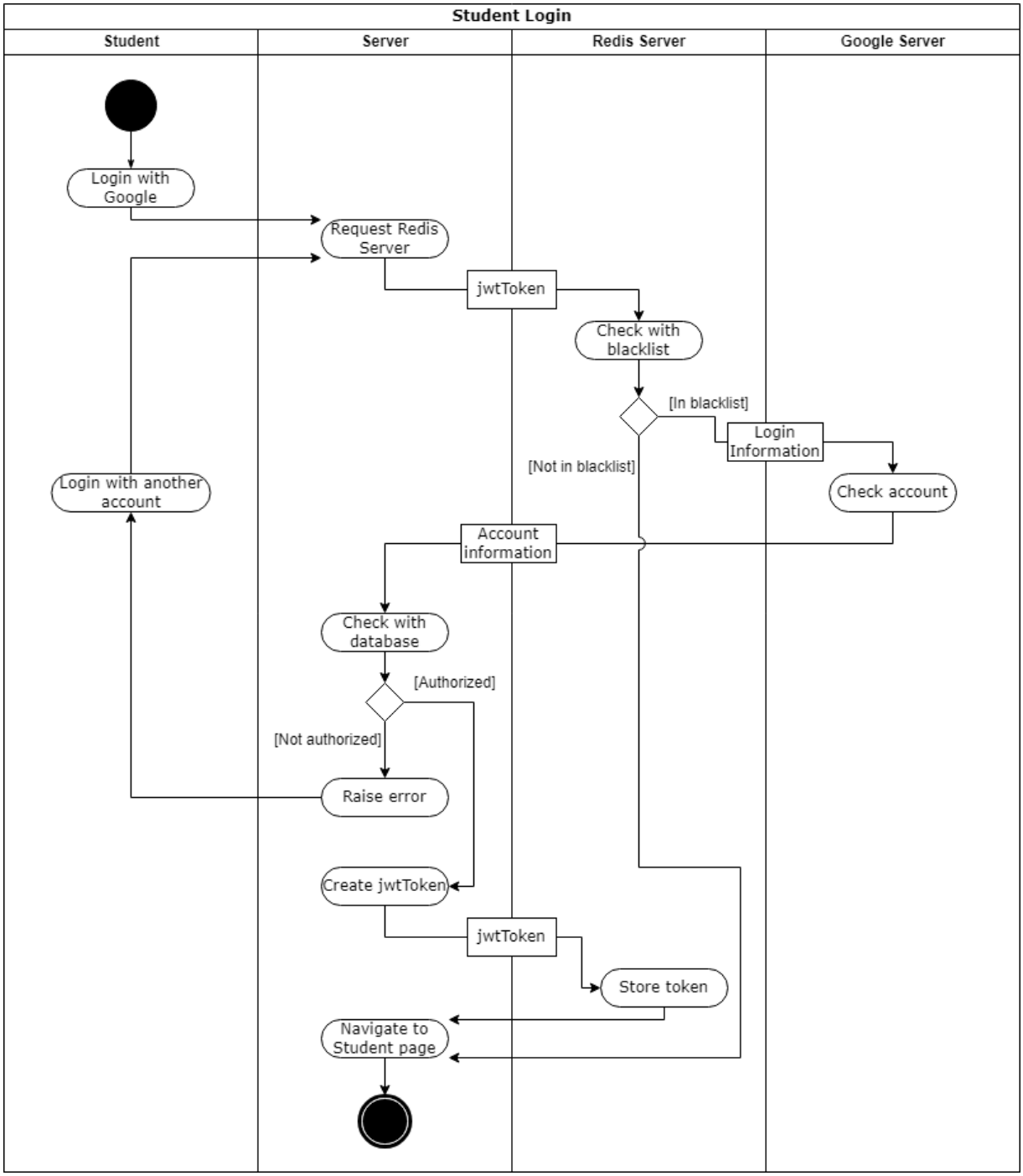


Figure 4.8: Activity diagram for Student Login process

When a student clicks Login, the server will check with the Redis server if the jwtToken is valid or not, if valid, the login is successful and navigate to the Student home page. If the jwtToken is not valid (in blacklist), the server requests a Google server to retrieve information. Server then checks this information, if it is an authorized student account, then creates a new jwtToken, stores it in the Redis server, and navigates to the Student home page. Else, stay on the Login page, request Student to use another account.

4.3.2. Staff Login

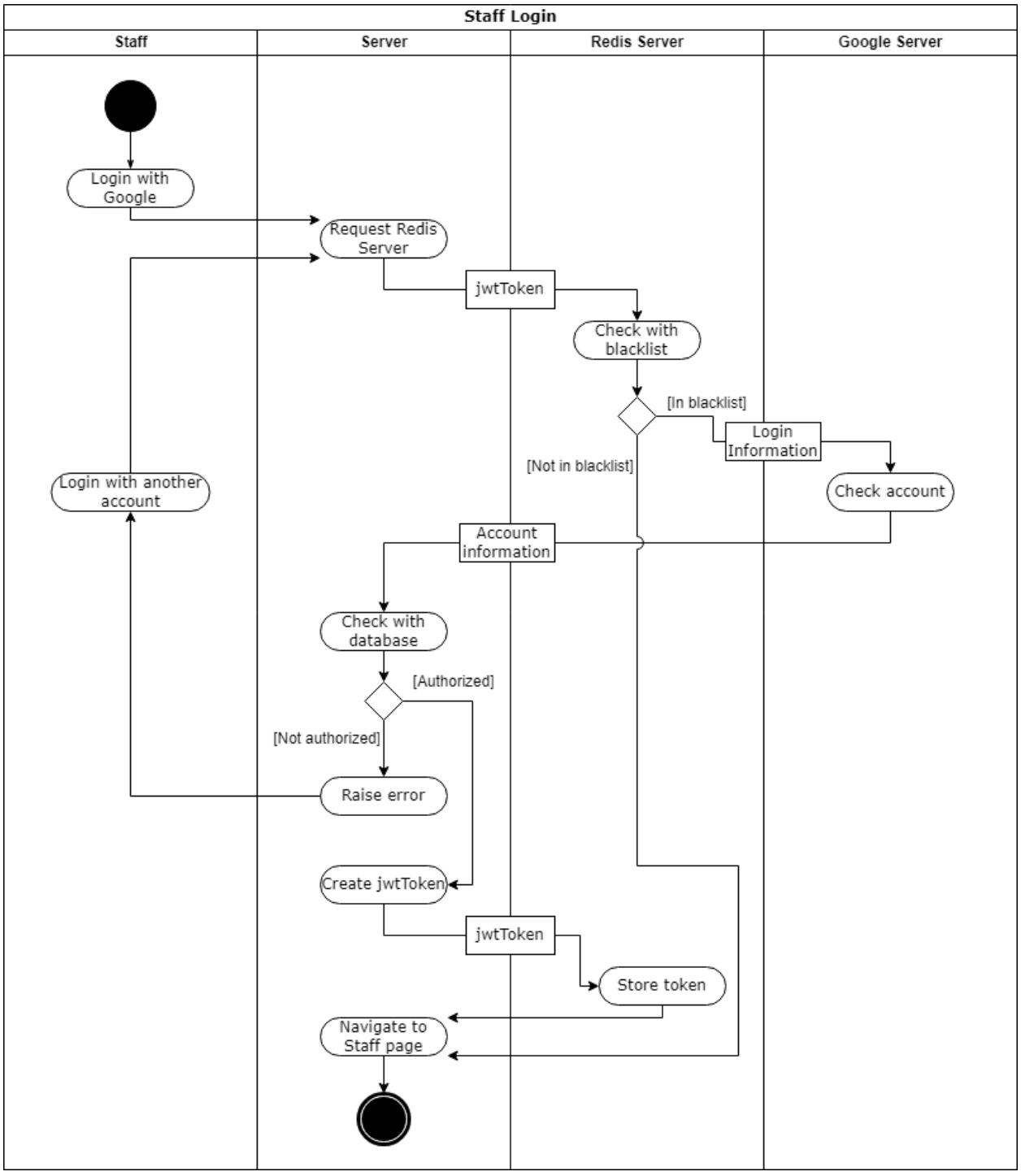


Figure 4.9: Activity diagram for Staff Login process

When a staff clicks Login, the server will check with the Redis server if the jwtToken is valid or not, if valid, the login is successful and navigate to the Staff home page. If the jwtToken is not valid (in blacklist), the server requests a Google server to retrieve information. Server then checks this information, if it is an authorized staff account, then creates a new jwtToken, stores it in the Redis server, and navigates to the Staff home page. Else, stay on the Login page, request the staff to use another account.

4.3.3. Company Login

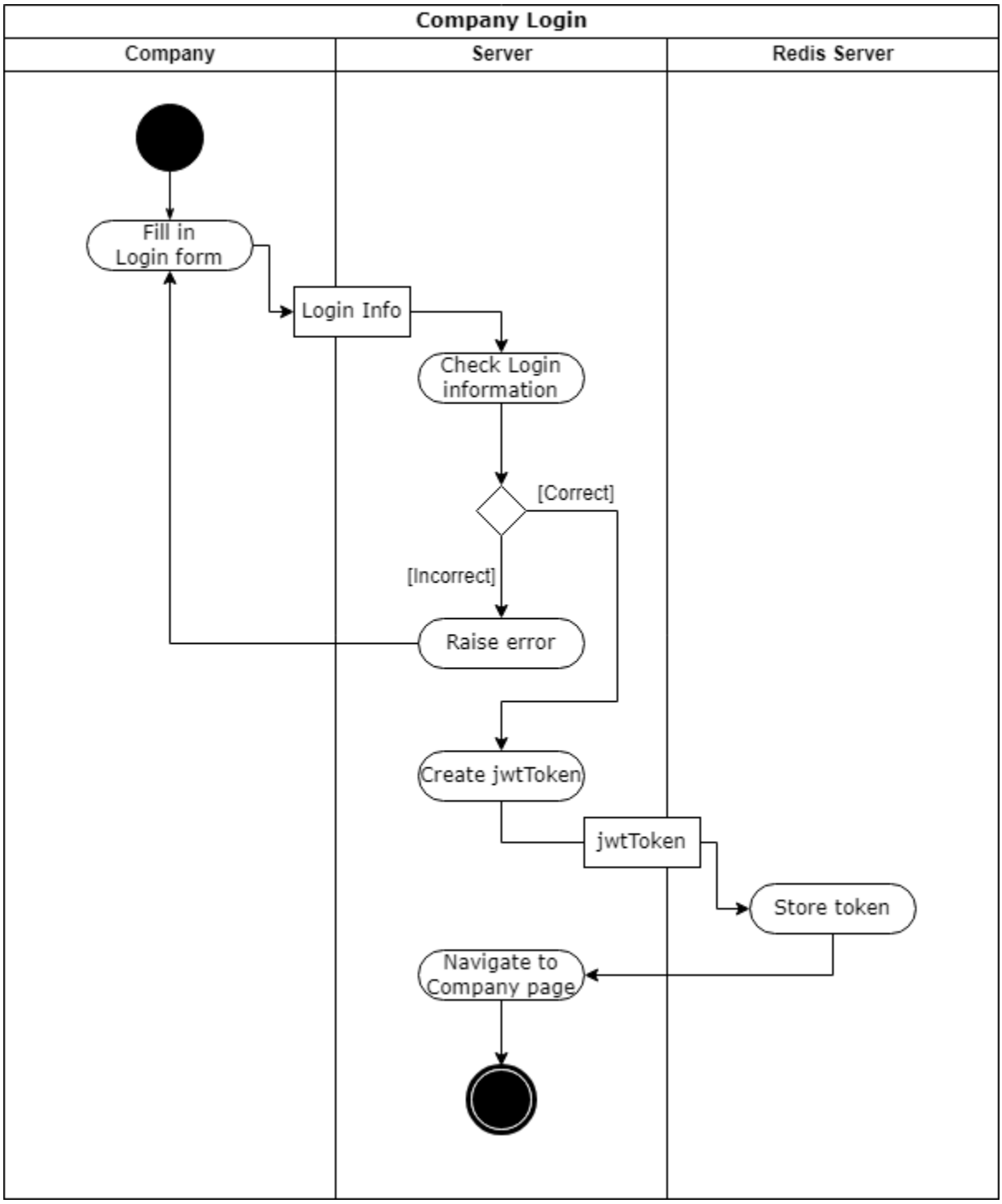


Figure 4.10: Activity diagram for Company Login process

When Company account user fills in Login form and clicks "Login" button, Login information is sent to server, this information then be checked with the database, if the information is correct, server create new jwtToken and store it in the Redis server, and navigate to Company home page. If information is not correct, display error notification and request the company user to fill the login information again.

4.3.4. Student Request Account

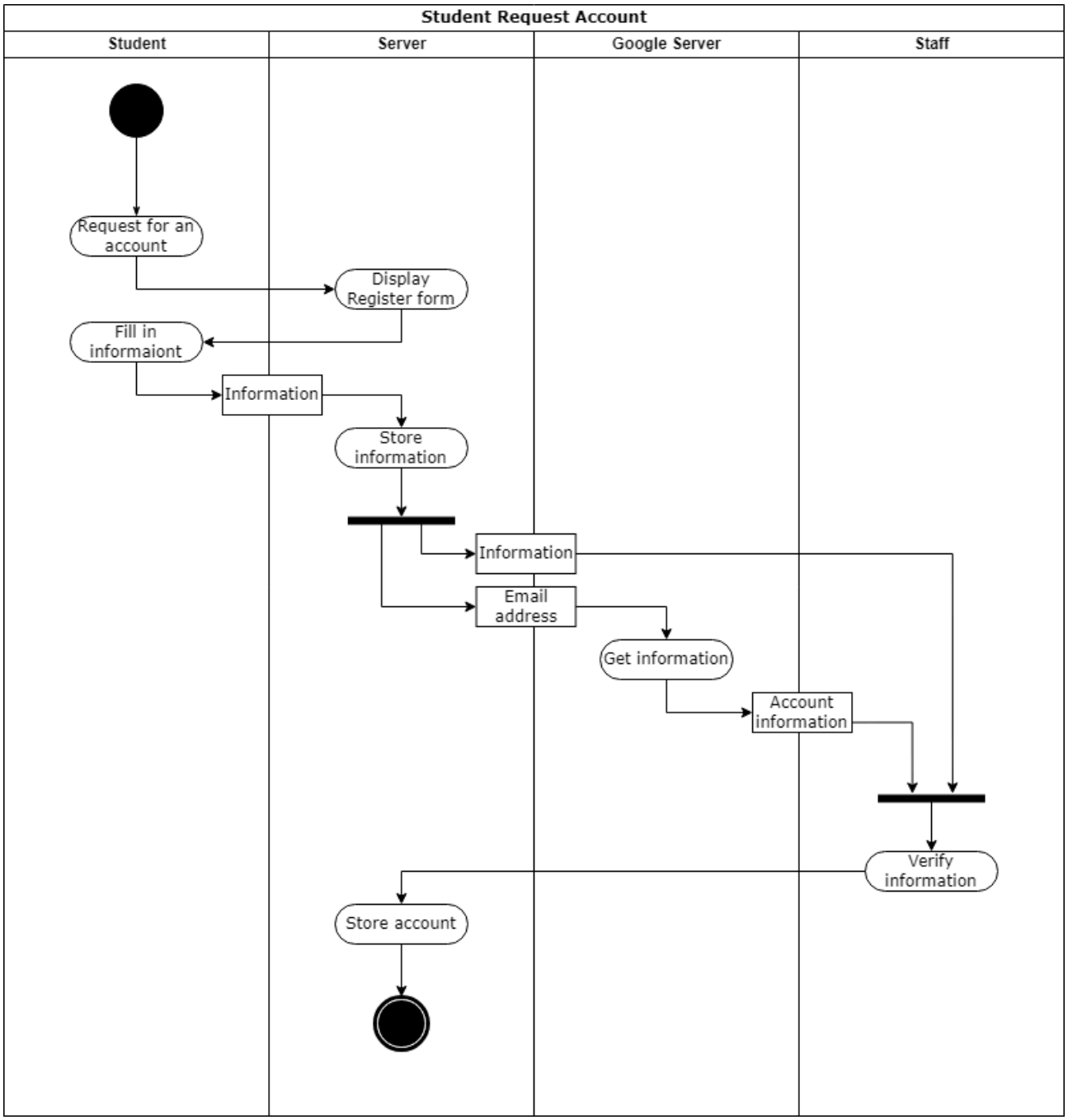


Figure 4.11: Activity diagram for Student Request Account process

When the Student clicks the "Request for an account" button, a form is displayed for students to fill in their information, this information is then sent and stored in the server. The email address is sent to Google server to retrieve account information. This information along with Student entered information is sent to Staff for verification. If the account is verified, store it in the database.

4.3.5. Company Sign-up

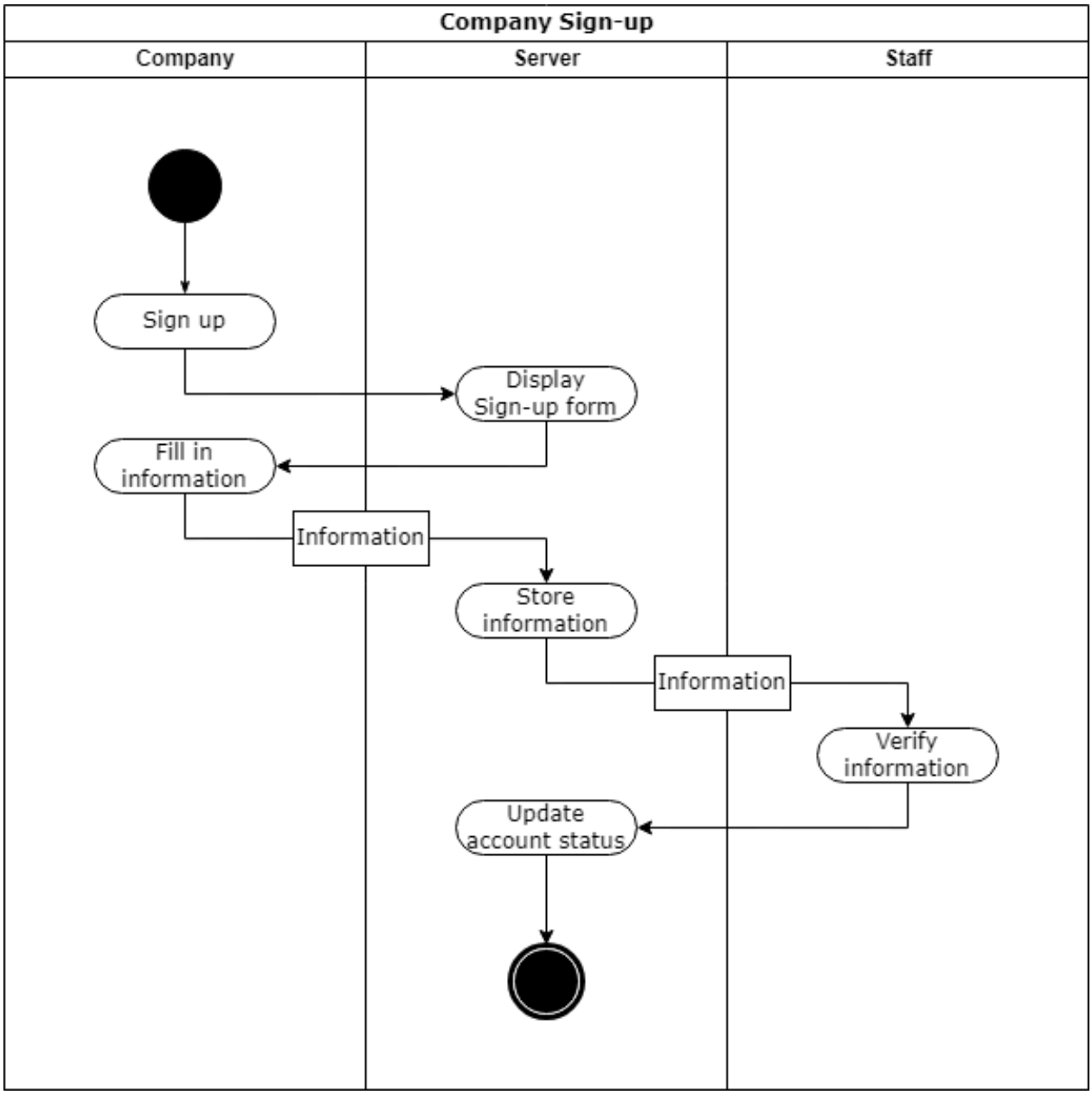


Figure 4.12: Activity diagram for Company Sign-up process

When the Company account user clicks the "Sign-up" button, a form is displayed to fill in the company's information. The information is then stored in the database, and sent to Staff for verification. After verification, the status of the account is updated.

4.3.6. Create job

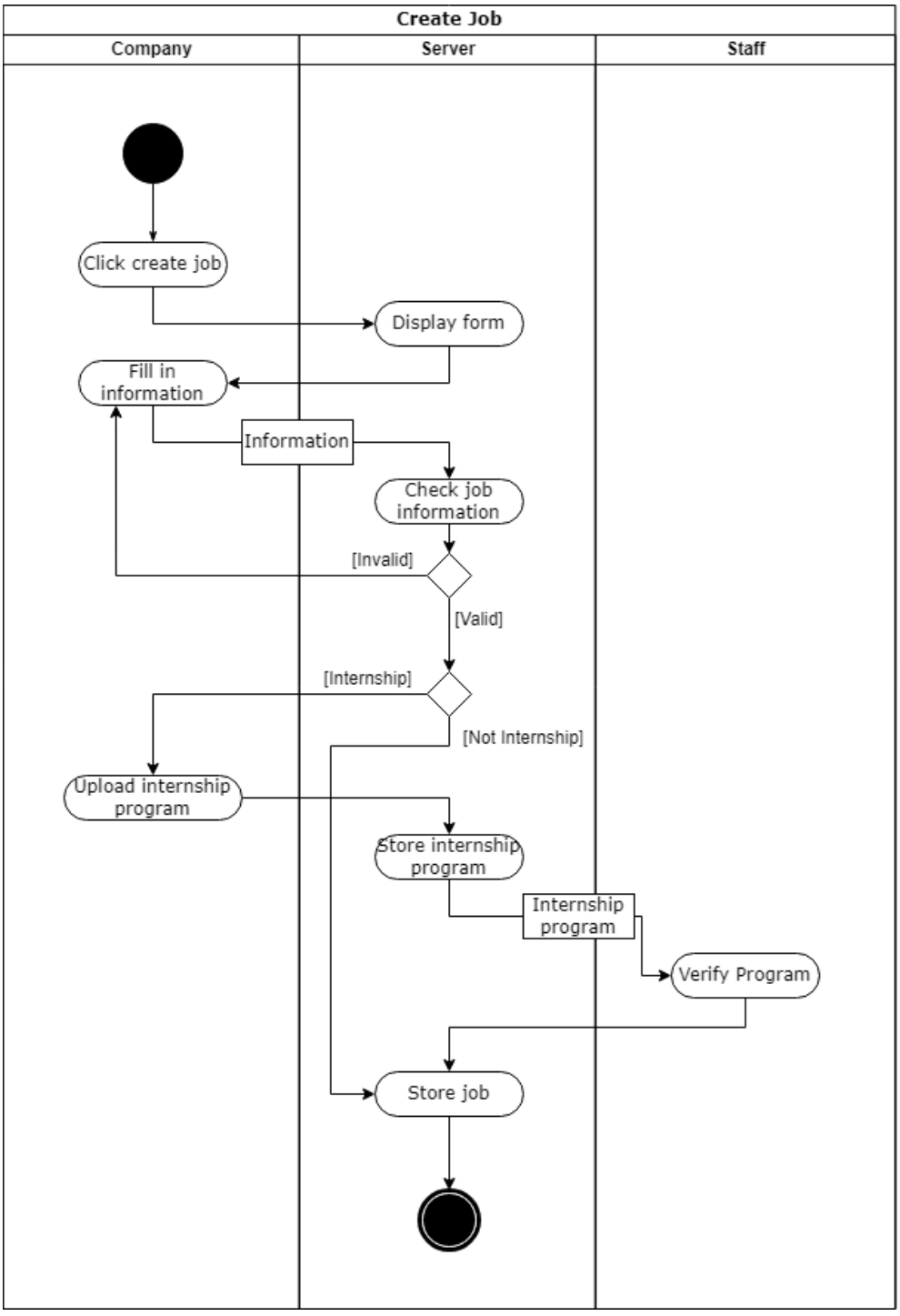


Figure 4.13: Activity diagram for Company Create Job process

When the Company account user clicks the "Create new job" button, a form is displayed to fill in the company's information. The information is then checked in the Server, if the information is invalid, the user needs to re-enter until the information is valid. If the job is an internship job, an internship program is required from the Company, the program is then verified by Staff. When all is done, the job is stored in the database. If the job is not an internship job, it is immediately stored in the database once all information is valid.

4.3.7. Apply for Intern Program

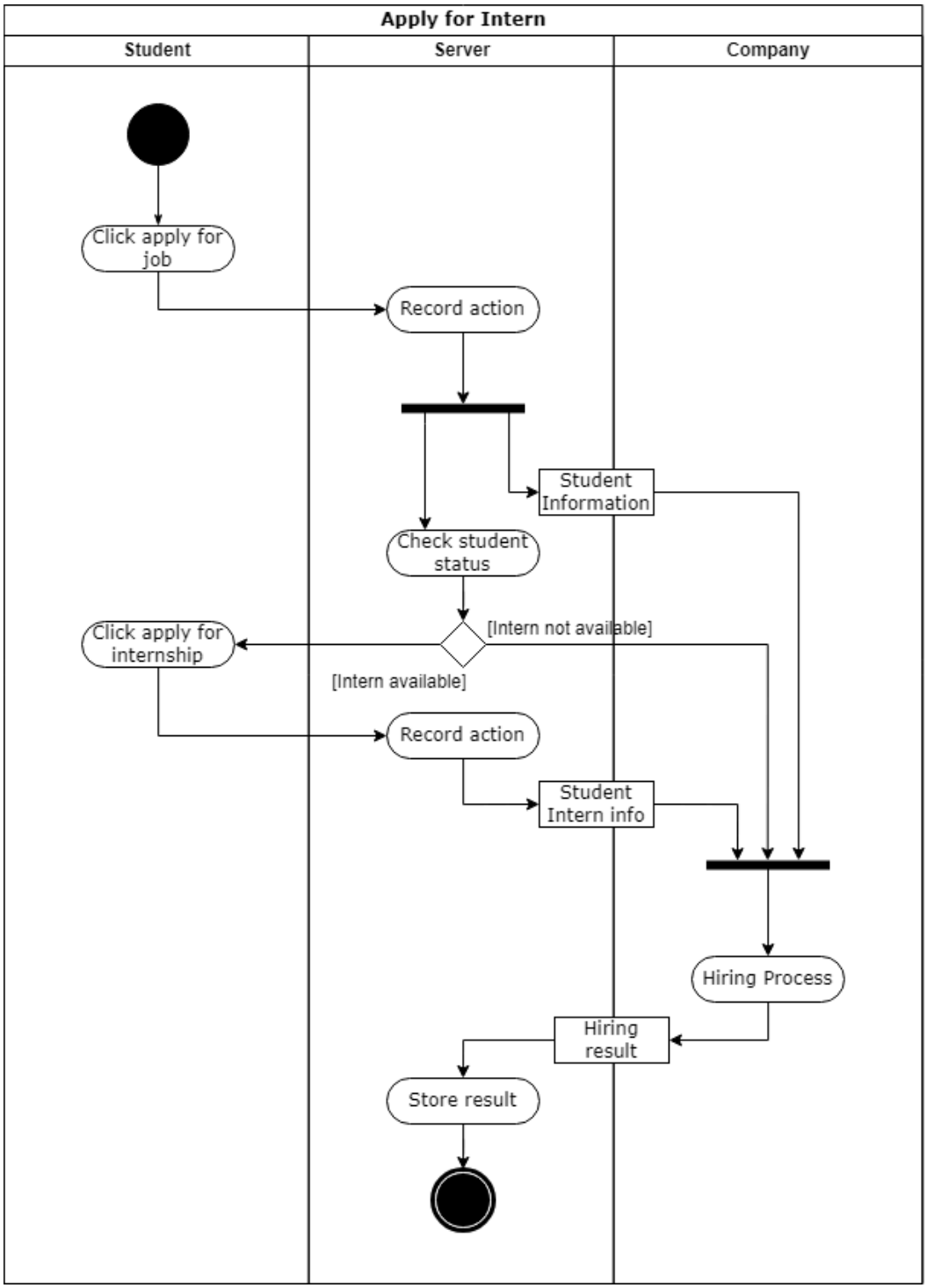


Figure 4.14: Activity diagram for Student Apply Internship Job process

When the Student clicks the "Apply for Job" button, the server then records the Student information and checks the status of that student. If the Student is available for an internship, a dialog is displayed to ask the student to apply for internship. All student information is then sent to the Company to review and start the recruitment. Once the recruit process is done, the result is sent to the server to be stored.

5. System, Database and User Interface

5.1. System design

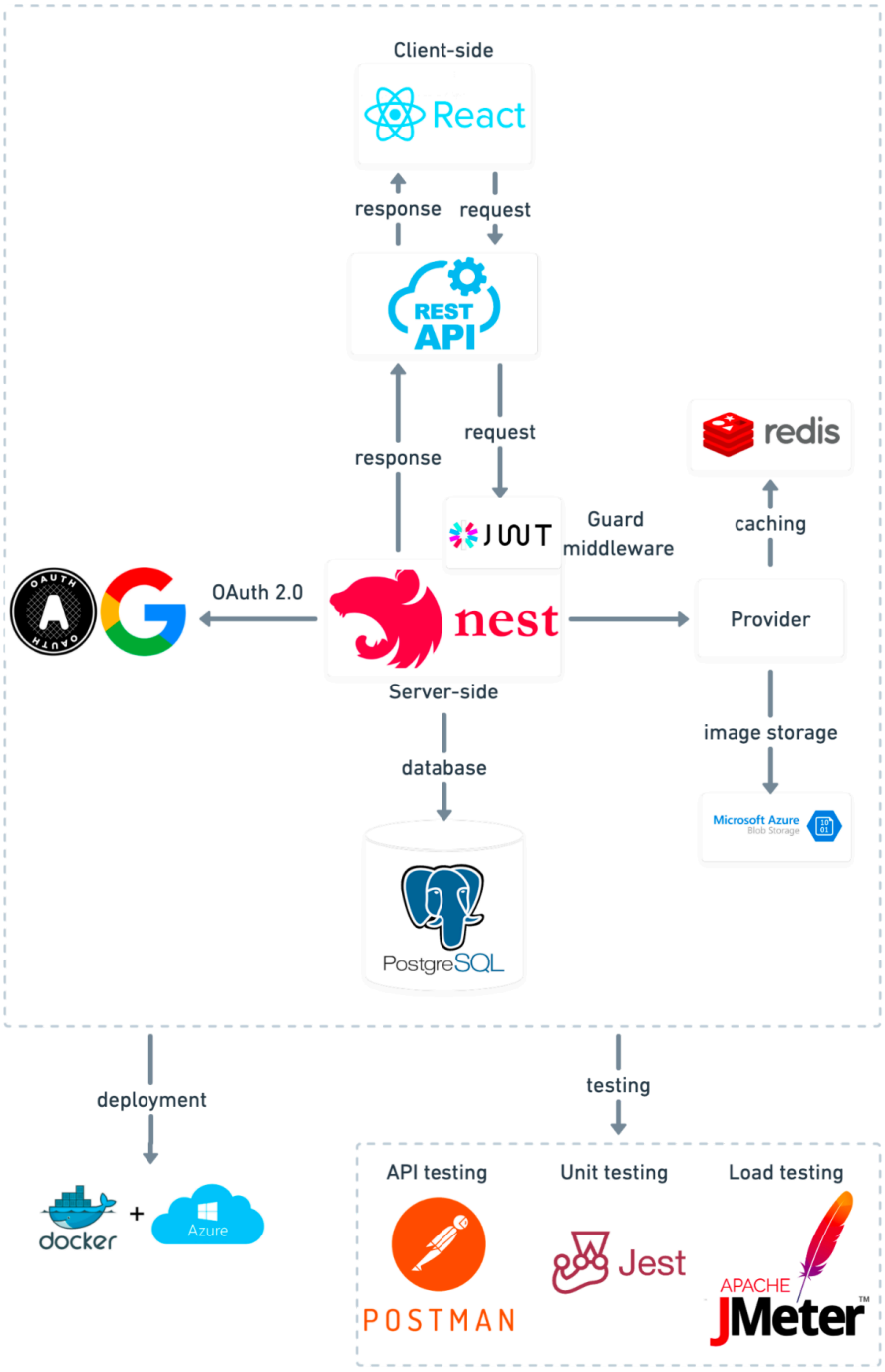


Figure 5.1: System design diagram

System Overview

The system utilizes a two-tier architecture, comprising:

- Client-side Tier: Responsible for user interface and interaction, built using the React JavaScript framework.
- Server-side Tier: Handles request processing and response generation, implemented using the Nest.js JavaScript framework.

Client-side Tier

- Components:
    - User Interface: Implements the visual elements and functionality that users interact with, leveraging React components and libraries.
    - Data Access Layer: Handles communication with the server-side tier through the REST API, leveraging HTTP requests and JSON data format.
- Responsibilities:
    - Receive user input and translate it into appropriate API requests.
    - Display responses from the server-side tier in a user-friendly format.
    - Manage user interface state and provide a responsive and interactive experience.

Server-side Tier

- Components:
    - REST API: Provides an interface for client-side applications to interact with the system, using Nest.js controllers and middleware.
    - Business Logic: Processes requests received from the API, performs necessary operations, and generates responses.
    - Data Access Layer: Handles interaction with the PostgreSQL database, utilizing database drivers and queries.

- Storage with Microsoft Blob Storage: To handle file storage efficiently, the server-side tier integrates Microsoft Blob Storage. This allows seamless and scalable storage of files, ensuring reliability and accessibility while benefiting from the robust features offered by Microsoft's cloud storage solution.
- Caching with Redis [1]: In optimizing performance, the server-side tier incorporates Redis for caching. By storing frequently accessed data in-memory, Redis significantly reduces response times, enhancing the overall system speed [2]. This caching mechanism ensures rapid access to frequently requested information, reducing the load on the database and improving user experience [2].
- Authentication with Google OAuth: Ensuring secure and seamless user authentication, the server-side tier integrates Google OAuth. This industry-standard authentication protocol provides a secure and user-friendly login experience. Leveraging Google OAuth adds an additional layer of security while simplifying the authentication process for users through their Google credentials.

- Responsibilities:
  - Receive and validate API requests from the client-side.
  - Execute business logic based on the received request.
  - Access and manipulate data in the PostgreSQL database.
  - Generate and return appropriate responses to the client-side.
  - Maintain system state and ensure data consistency.

Testing Tier

- Components:
  - Unit Tests: Verify the functionality of individual units of code (functions, classes).

    - API Tests: Validate the functionality and behavior of the REST API.
    - Load Tests: Assess system performance under varying loads.
- Responsibilities:
    - Ensure all system functionalities meet the defined requirements.
    - Identify and address potential errors and bugs.
    - Evaluate system performance and identify potential bottlenecks.
    - Facilitate continuous integration and delivery.

Component Interactions

The following flow describes the interaction between system components:

- User Interaction: The user interacts with the client-side application's user interface.
- Request Generation: The client-side application translates user action into a corresponding API request.
- API Communication: The client-side application sends the API request to the server-side tier.
- Request Processing: The server-side tier receives the request and validates its format and content.
- Business Logic Execution: The server-side tier executes the appropriate business logic based on the request.
- Database Interaction: The server-side tier interacts with the PostgreSQL database to access and manipulate data.
- Response Generation: The server-side tier generates a response based on the executed business logic.
- Response Delivery: The server-side tier sends the response back to the client-side application.
- Response Display: The client-side application receives the response and updates the user interface accordingly.

Deployment and Scalability

- The system is deployed to Microsoft Azure, a cloud platform offering scalability and reliability.
- Containerization technologies, like Docker, can facilitate deployment and scaling across multiple servers.

Security Considerations

- Secure authentication and authorization mechanisms are implemented to control user access and protect sensitive data.
- Data encryption is employed to safeguard information at rest and in transit.
- Secure coding practices and regular security audits are essential to maintain a robust security posture.

Conclusion

The system design presented demonstrates a well-structured and scalable architecture, utilizing popular and well-supported technologies. Deployment to Microsoft Azure ensures a reliable and flexible platform for future growth. By implementing rigorous testing procedures and security measures, the system's performance and integrity can be maintained effectively. This detailed report provides a comprehensive overview of the system design, paving the way for successful implementation and operation.

5.2. Database design

5.2.1. Database requirement

The database for this system requires tables to store information about students, companies, jobs, internship details, faculty, and staff. For students, there should be fields for email, name, student ID, ClassCode, phone numbers, avatar, GPA, verification status, resume links, creation date, and update date. Intern students should have additional fields for status, associated report, and internship progress, each with their respective creation and update dates.

Company information should include ID, name, description, phone numbers, address, avatar, website, TaxId, verification status, workfields, and creation/update timestamps. Jobs created by companies should have fields for ID, title, level, salary, work type, quantity, description, image, expiration date, and creation/update timestamps. Internship jobs would include process, result, and associated student results. Application details for students and internship students applying for jobs should store status, creation date, and update date.

The faculty table should have ID, name, and avatar, with creation and update dates. Staff information should include ID, StaffId, name, email, avatar, phone number, faculty affiliation, and creation/update timestamps. Staff_admin status should also be indicated.

The database design should ensure data integrity, security, and efficient retrieval for seamless system functionality.

5.2.2. Entity-Relationship Diagram

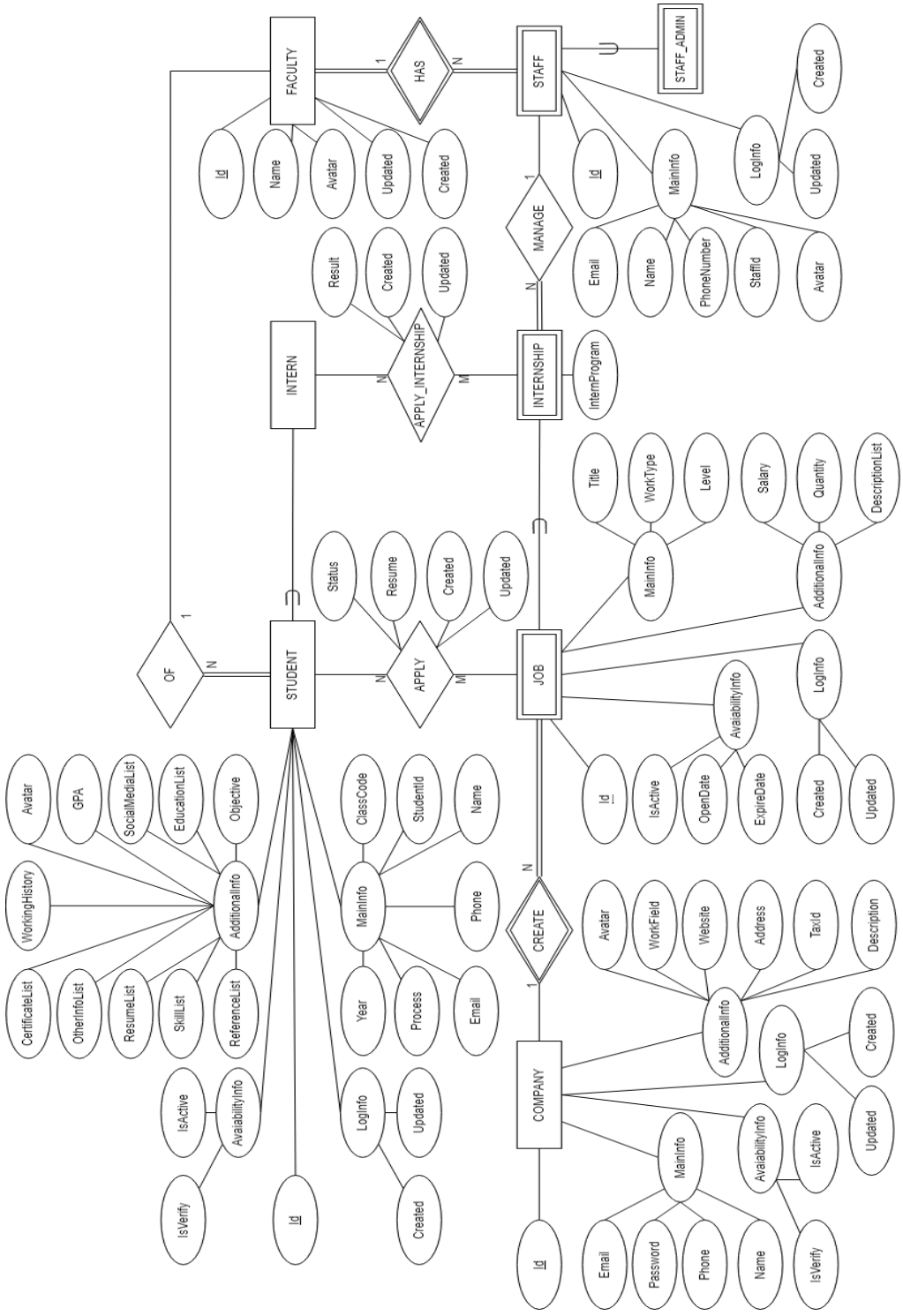


Figure 5.2: ERD for the database of the system

5.2.3. Database schemas

Figure 5.3: Diagram for the database of the system

- **Faculty:** this will contain the information of the faculty as below:
  - id: uuid (NOT NULL UNIQUE)
  - name: string (NOT NULL)
  - avatar: string (NOT NULL) (store the url return from the Azure Blob storage)
- **Staff:** this will contain the information of staff faculty:
  - id: uuid (NOT NULL UNIQUE)
  - name: string (NOT NULL)
  - email: string (NOT NULL)
  - phoneNumber: string
  - role: string (NOT NULL)
  - isAdmin: boolean (NOT NULL)
  - staffID: integer
  - facultyID: uuid (NOT NULL, FOREIGN KEY)
  - avatar: string (NOT NULL) (store the url return from the Azure Blob storage)
- **Student:** this will contain the information of a student:
  - id: uuid (NOT NULL UNIQUE)
  - name: string (NOT NULL)
  - email: string (NOT NULL)
  - phoneNumber: string
  - role: string (NOT NULL)
  - isVerify: boolean (NOT NULL)
  - studentID: integer
  - gpa: integer
  - classCode: string
  - resume: text[] (this will store an array of url to the student's resume)
  - skill: text[] (this will store an array of skillset of the student)

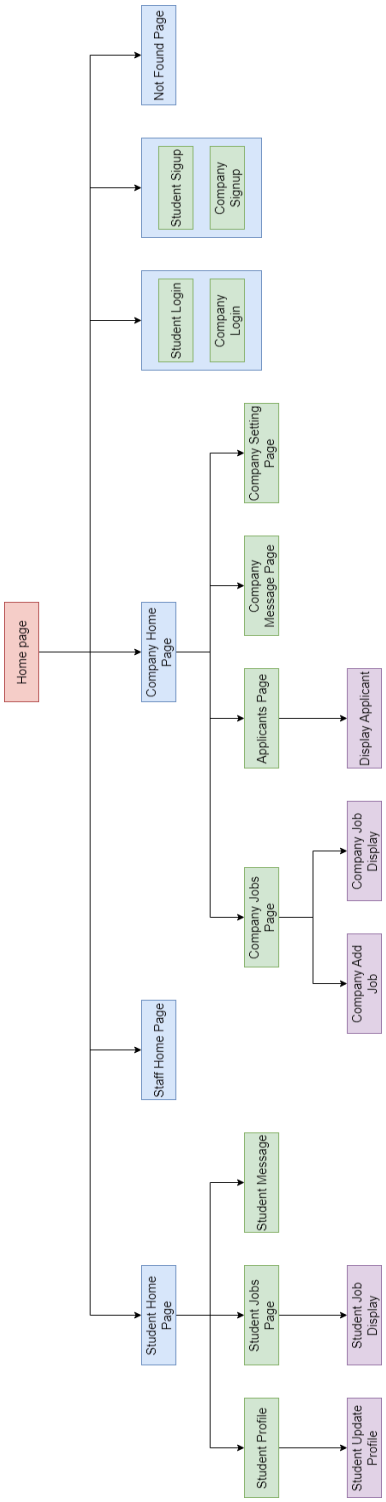
- facultyID: uuid (NOT NULL, FOREIGN KEY)
- avatar: string (NOT NULL) (store the url return from the Azure Blob storage)

- **Company:** this will contain the information of a company:
  - id: uuid (NOT NULL UNIQUE)
  - name: string (NOT NULL)
  - password: string (NOT NULL)
  - email: string (NOT NULL)
  - phoneNumber: string
  - role: string (NOT NULL)
  - workField: string
  - address: string
  - website: string
  - description: string
  - taxId: integer
  - isVerify: boolean (NOT NULL)
  - avatar: string (NOT NULL) (store the url return from the Azure Blob storage)
- **Job:** this will contain the information of a job which is create by a company staff:
  - id: uuid (NOT NULL UNIQUE)
  - title: string (NOT NULL)
  - image: text (NOT NULL)
  - salary: integer (NOT NULL)
  - level: string (NOT NULL)
  - workType: string (NOT NULL)
  - quantity: integer
  - description: json
  - companyID: uuid (NOT NULL, FOREIGN KEY)

- expiredDate: Date

- **Job_applicant:** this will contain the information of candidate who applying for this job
  - id: uuid (NOT NULL UNIQUE)
  - status: string (NOT NULL)
  - studentID: uuid (NOT NULL, FOREIGN KEY)
  - jobID: uuid (NOT NULL, FOREIGN KEY)
- **Internship:** this will contain the information of candidate who applying for this job in which has the level Internship
  - id: uuid (NOT NULL UNIQUE)
  - process: string (NOT NULL)
  - result: integer
  - jobID: uuid (NOT NULL, FOREIGN KEY)

5.3. Website Hierarchical Model



Home Page
↳Student Home Page
↳Student Profile
↳Student Jobs Page
↳Student Job Display
↳Student Companies Page
↳Student Company Display
↳Staff Page
↳Company Home Page
↳Company Jobs Page
↳Company Add Job
↳Company Job Display
↳Applicants Page
↳Display Applicant
↳Company Setting Page
↳Internships Page
↳Display Internship
↳Student Login
↳Company Login
↳Student Sign-up
↳Company Sign-up
↳Not Found Page

Figure 5.4: Website Hierarchical Model

5.4. User interface - wireframe design

5.4.1. Home page

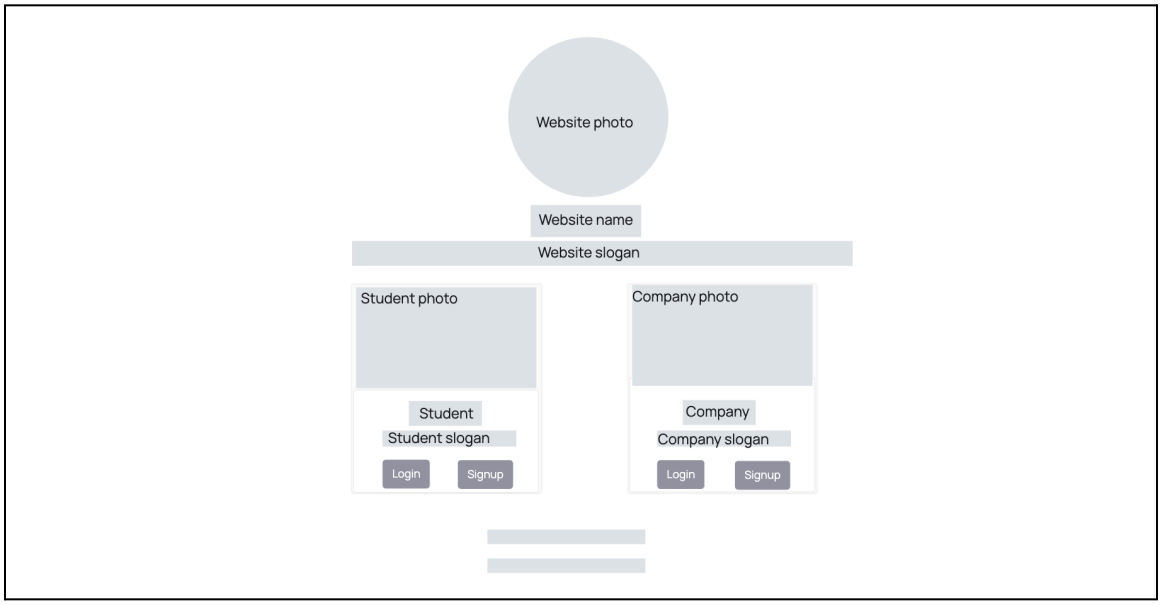


Figure 5.5: Home page

The home page of the website includes 2 cards for Student and Company to Login or Signup.

5.4.2. Student Login page

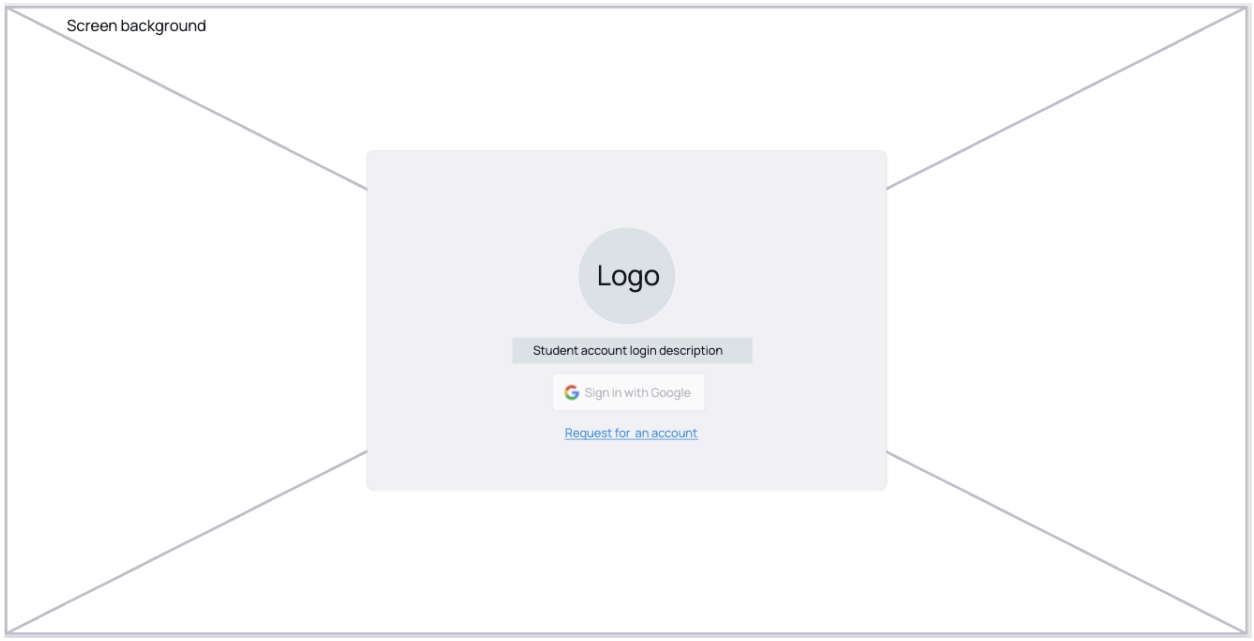


Figure 5.6: Student Login page

In the Student login page, students must login to the website using the school email address that is provided before. Else, students must request for an account.

5.4.3. Student request account page

Figure 5.7: Student request account page

To request an account, students must provide Name, Student Id, School Email address and phone number for staff faculty to verify and register an account.

5.4.4. Company login page

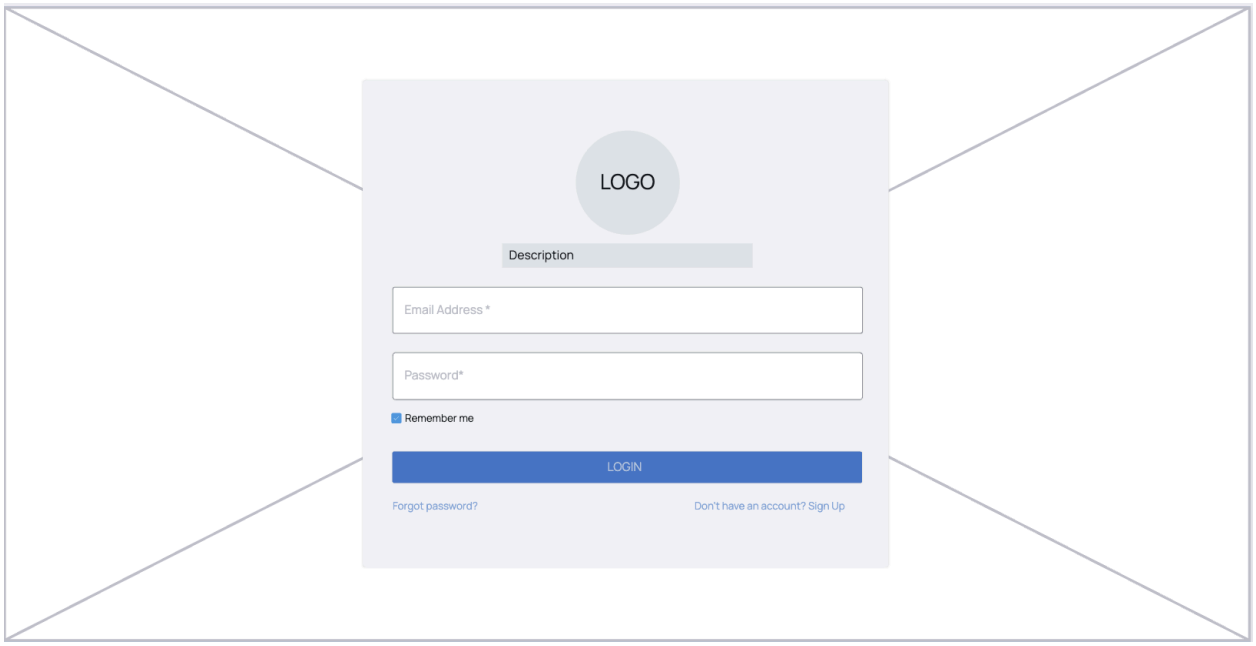


Figure 5.8: Company login page

For company staff, to login using a registered account, email address and password must be provided. Else, the company must register for a new account.

5.4.5. Company signup page

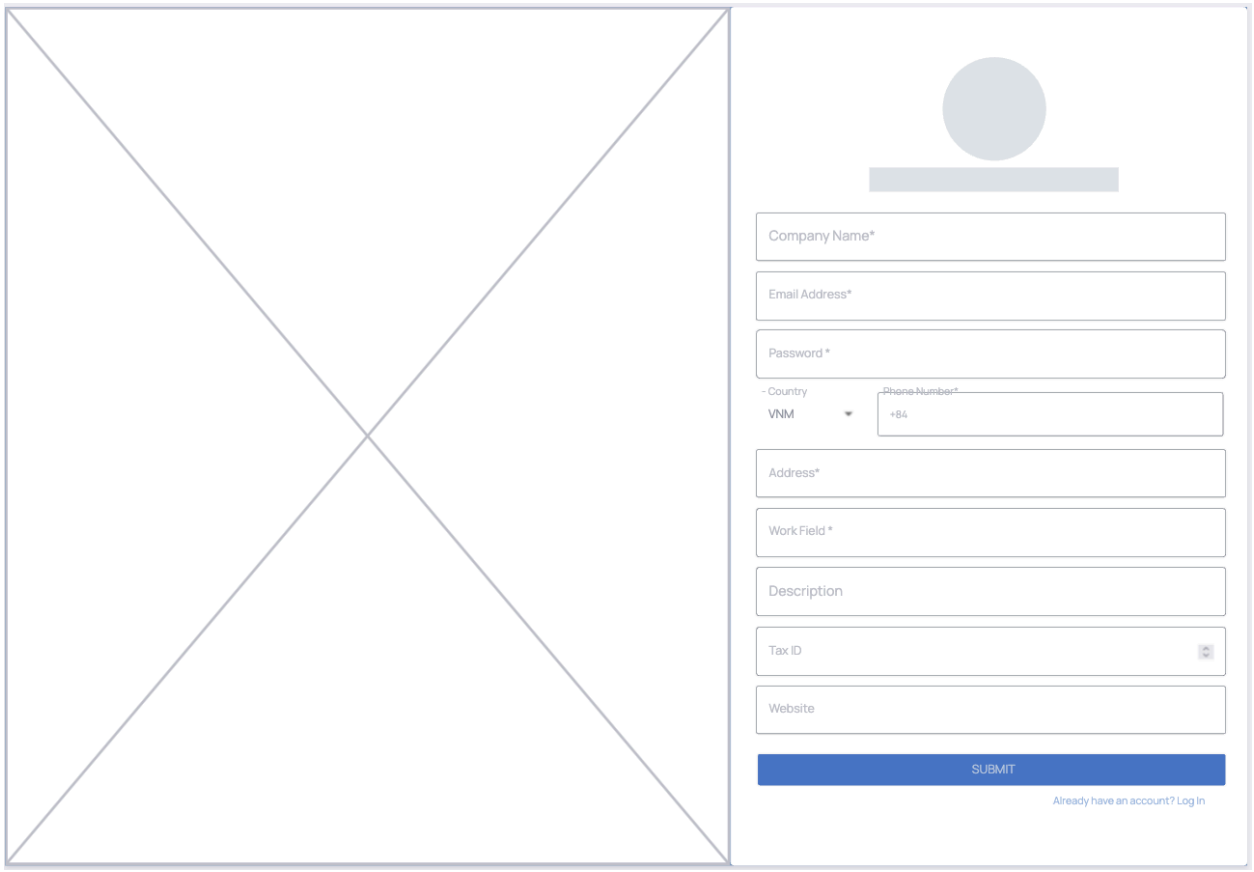


Figure 5.9: Company signup page

For account registration, companies must fill in some important fields such as Name, email address, phone number, tax ID, etc. This information will be read by faculty staff to verify the account.

5.4.6. Student home page

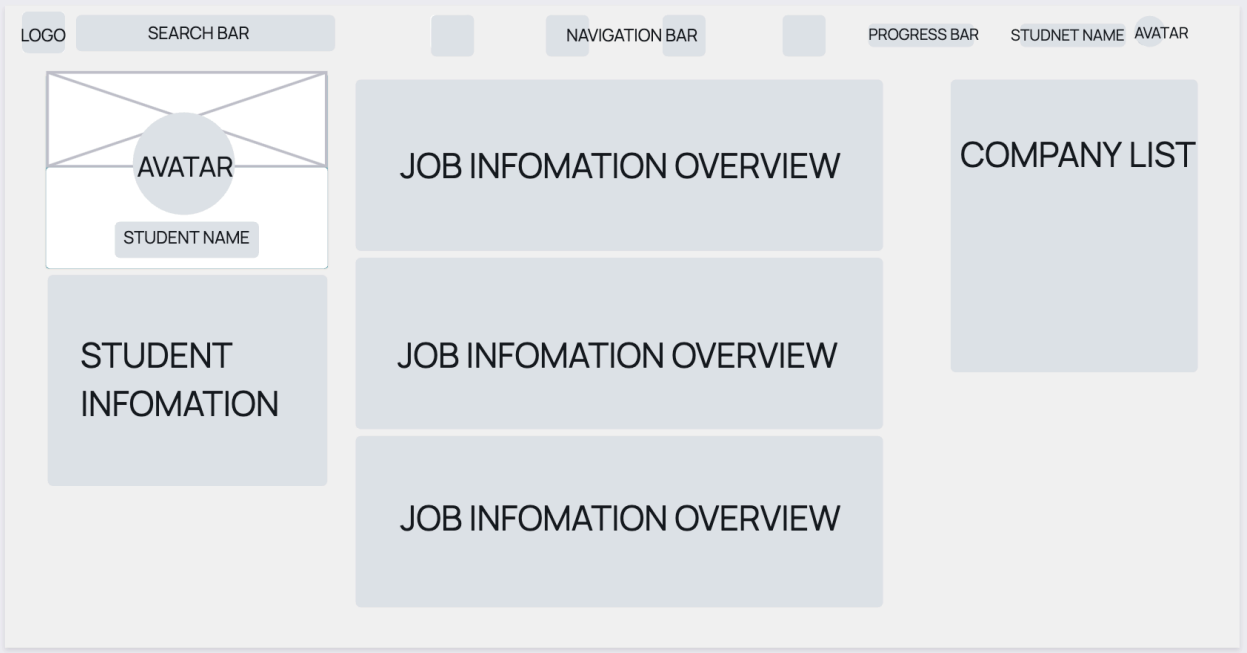


Figure 5.10: Student home page

After a student has logged in, the home page will be displayed. Home page includes 4 parts:

- A navigation bar at the top of the screen. It will have some navigation buttons to navigate to the Jobs page, Message page, Notifications page. Also this bar displays the status and email address of the student.
- Left sidebar. This displays some important information about the student such as name, faculty, major, class, GPA.
- Right sidebar. This bar displays the name and contains links to connected companies.
- Center. Displays all jobs' information posted by companies. These jobs are sorted with the most related first.

5.4.7. Student update information page

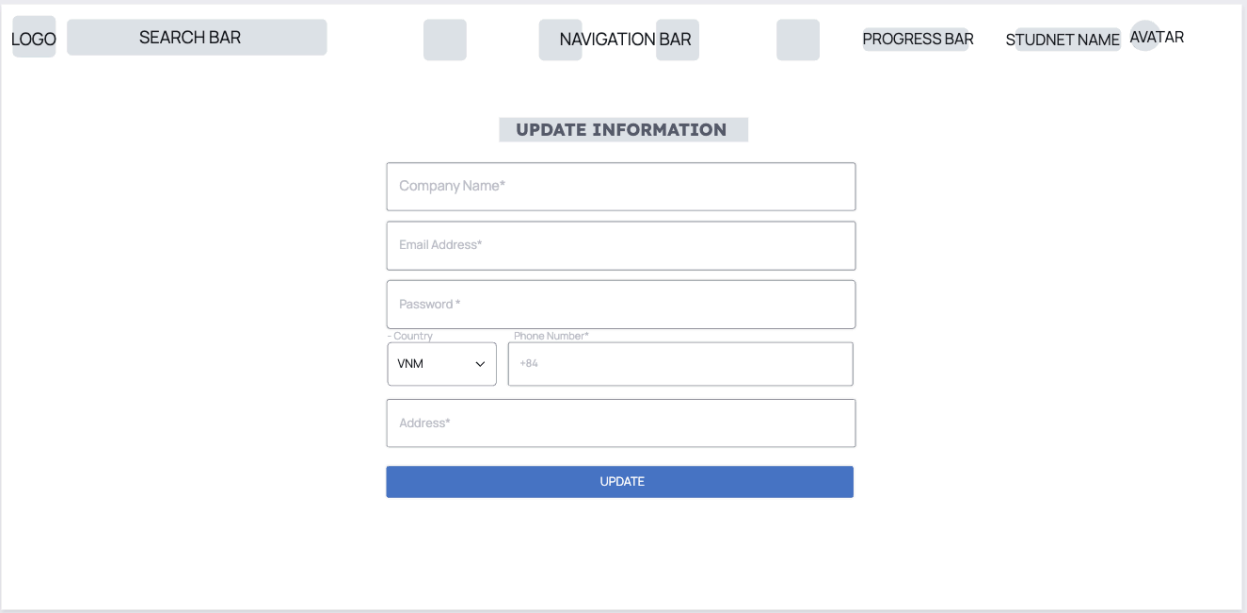


Figure 5.11: Student update information page

Students can update their information on this page. The correct password must be provided to update information.

5.4.8. Company home page

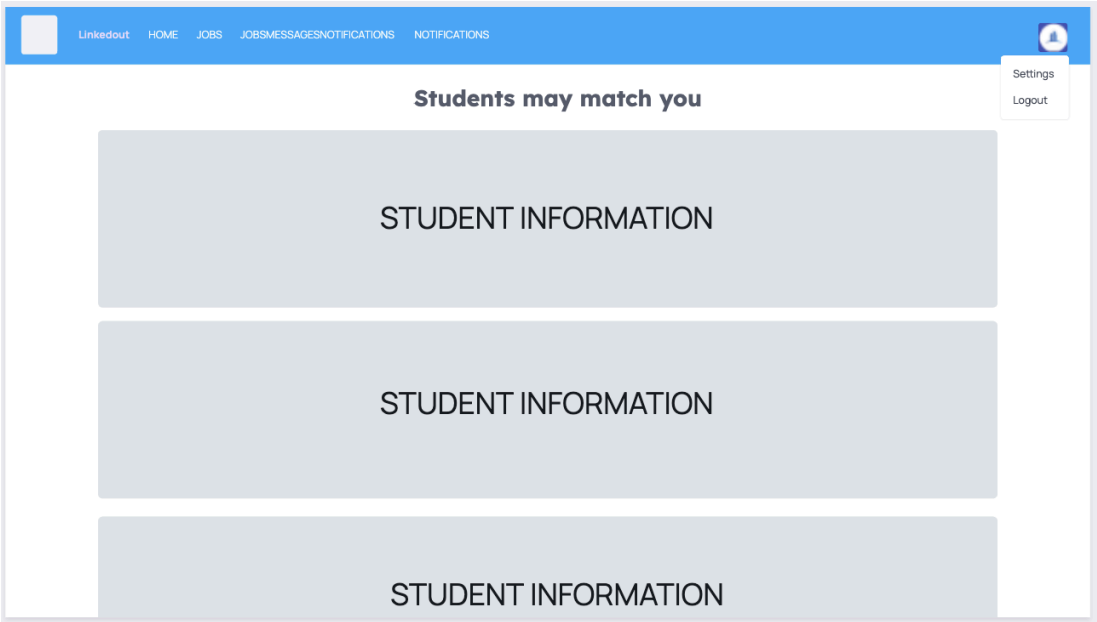


Figure 5.12: Company home page

For the company, the home page displayed some information of students that could match their posted jobs.

5.4.9. Company update information dialog

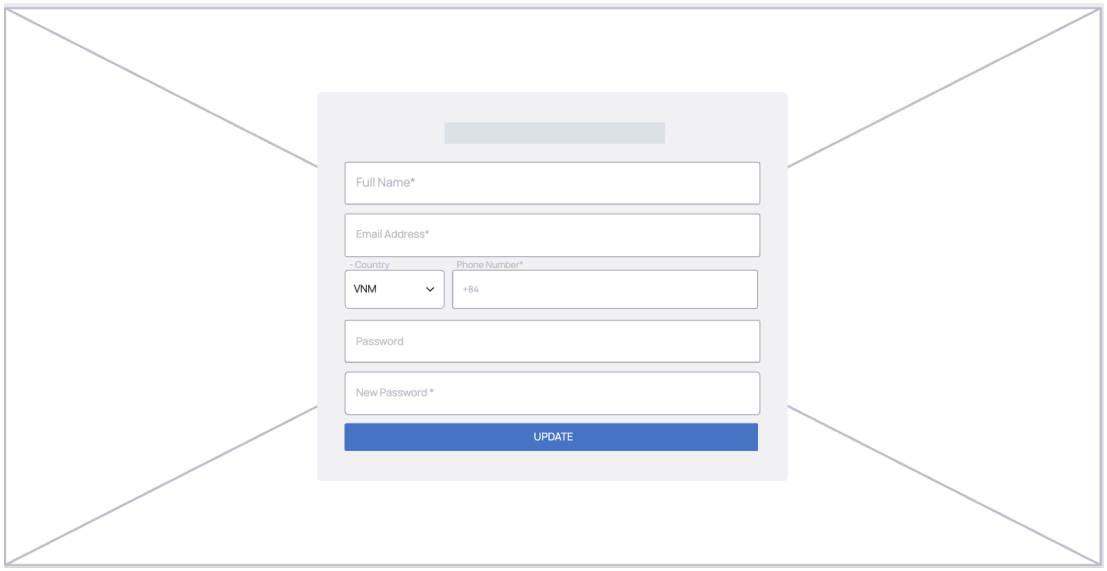


Figure 5.13: Company update information dialog

This popup allows companies to see and update their information.

5.4.10. Company Jobs management page

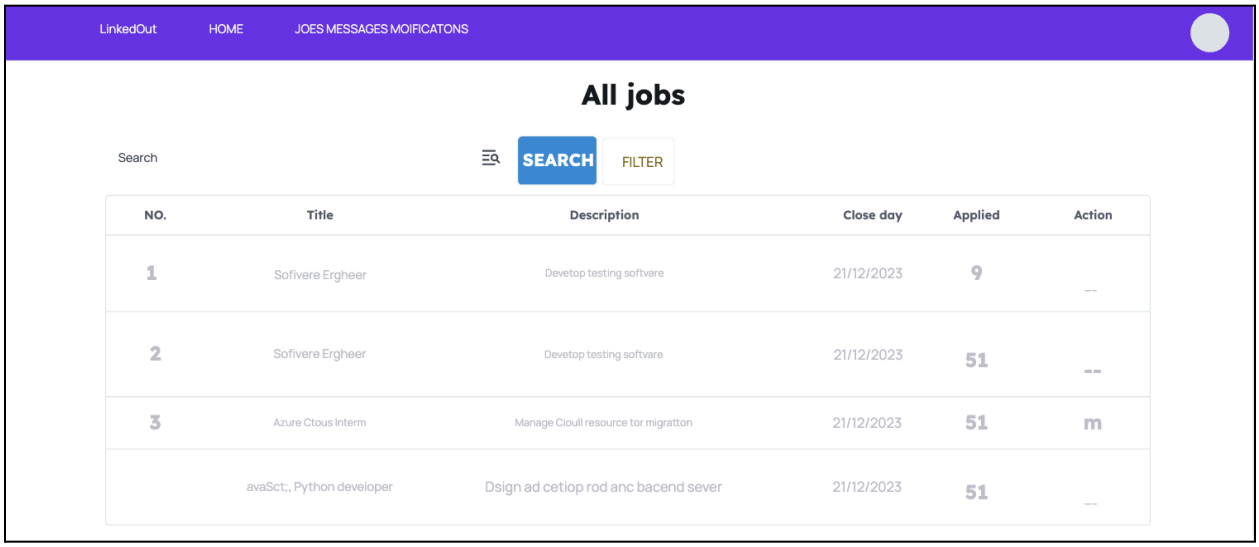


Figure 5.14: Company Jobs management page

This page shows all jobs the company posted.

5.4.11. Staff home page

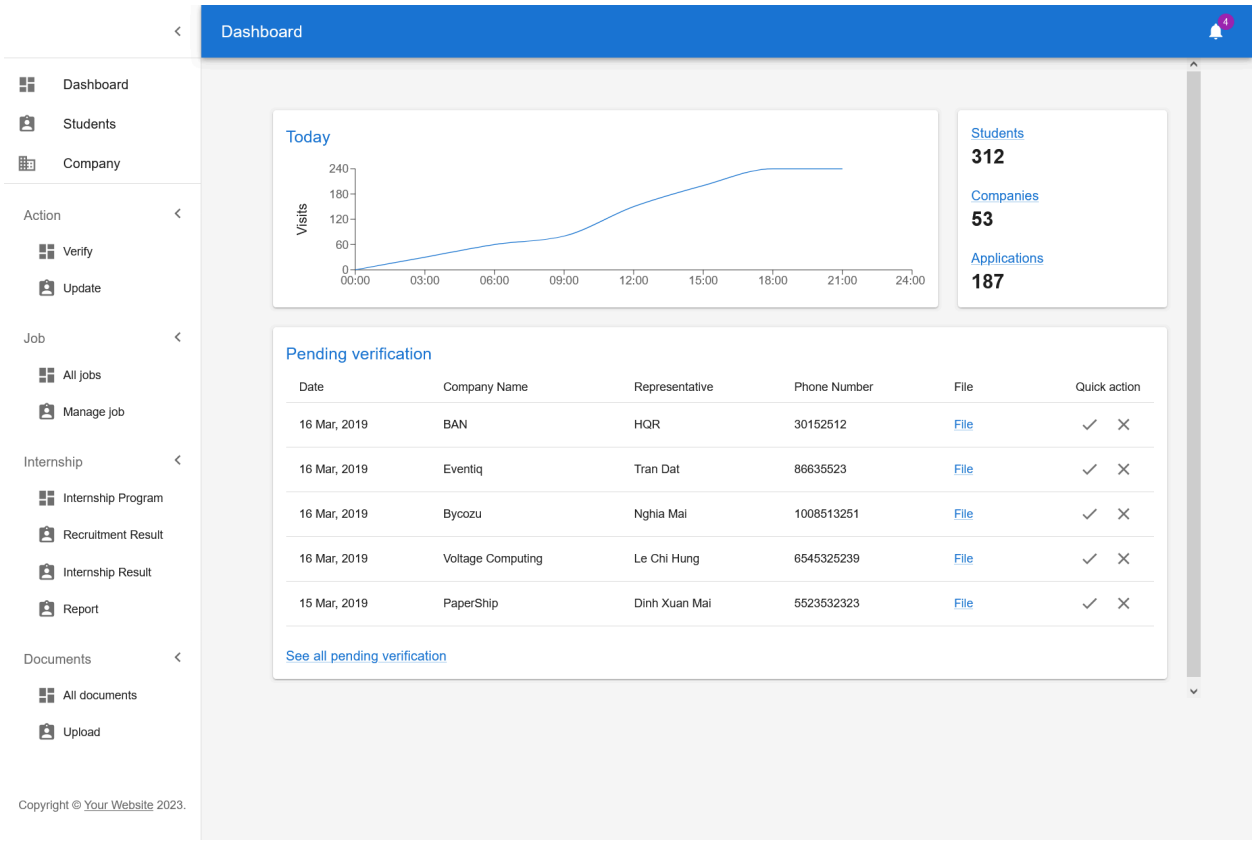


Figure 5.15: Staff home page

The home page for faculty staff will display some parts for staff quick actions such as view number of students, companies, applications, and a quick verify table.

5.4.12. Staff - Student management page

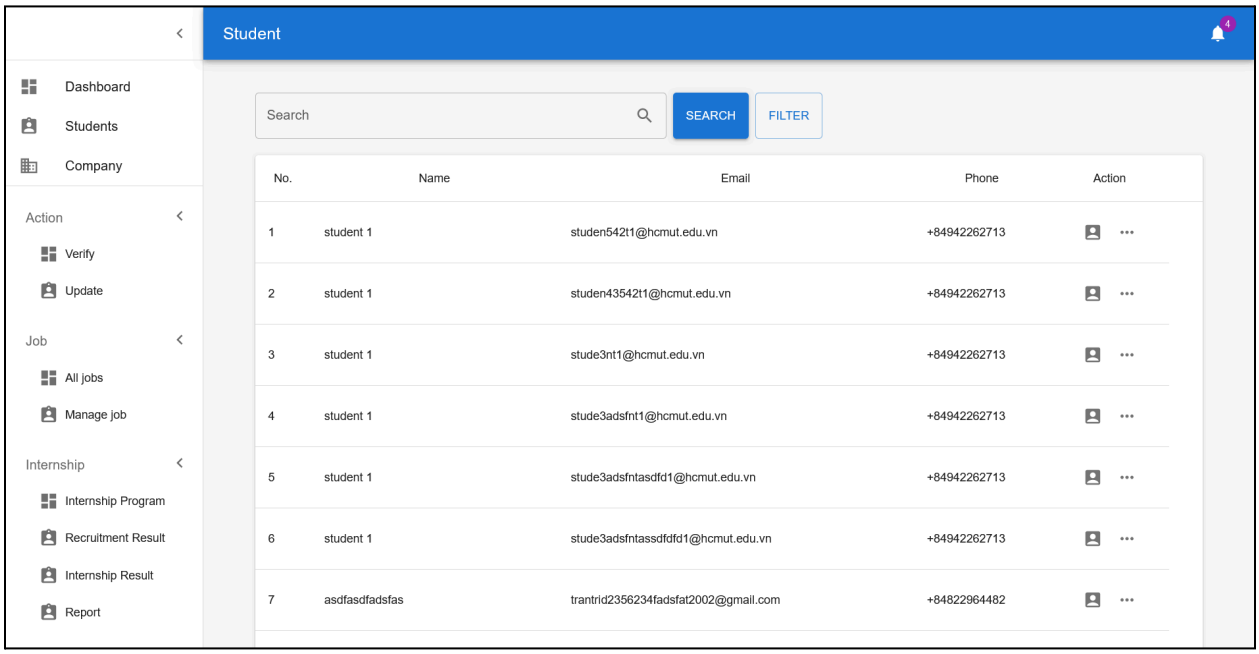


Figure 5.16: Staff - Student management page

In the student page, staff will see a list of all students' information and there are some buttons for more actions such as view details, delete, lock account.

5.4.13. Staff - Company management pape

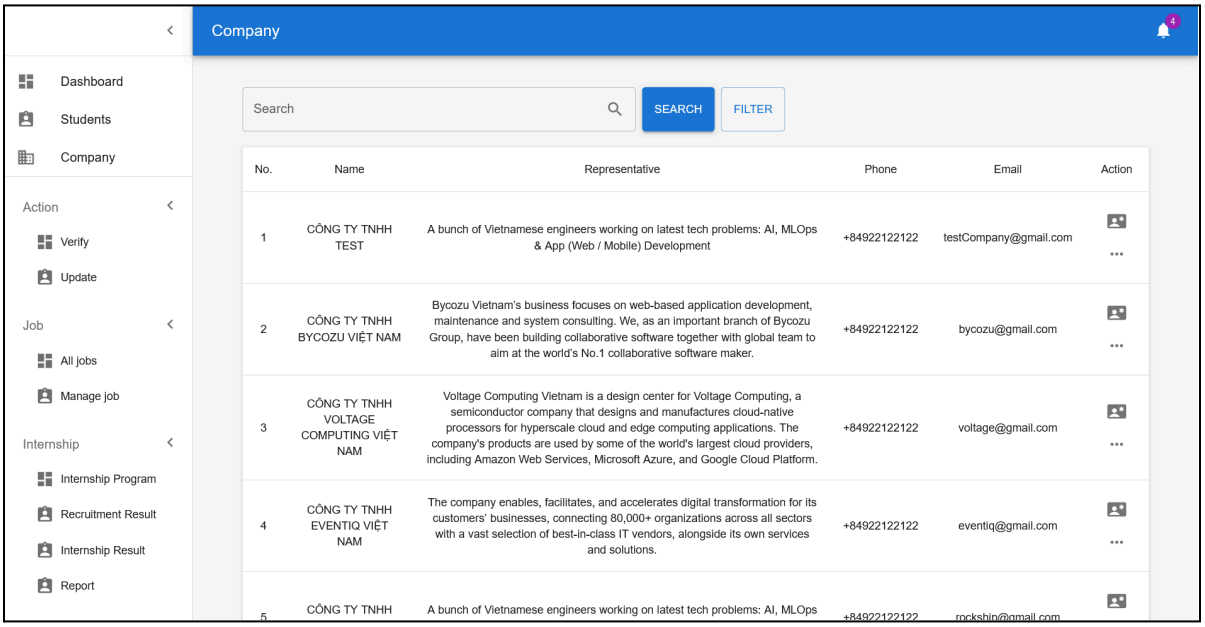


Figure 5.17: Staff - Company management page

Similar to the student list page, there is a page that displays information of all companies on the website.

5.4.14. Staff - Verify company information page

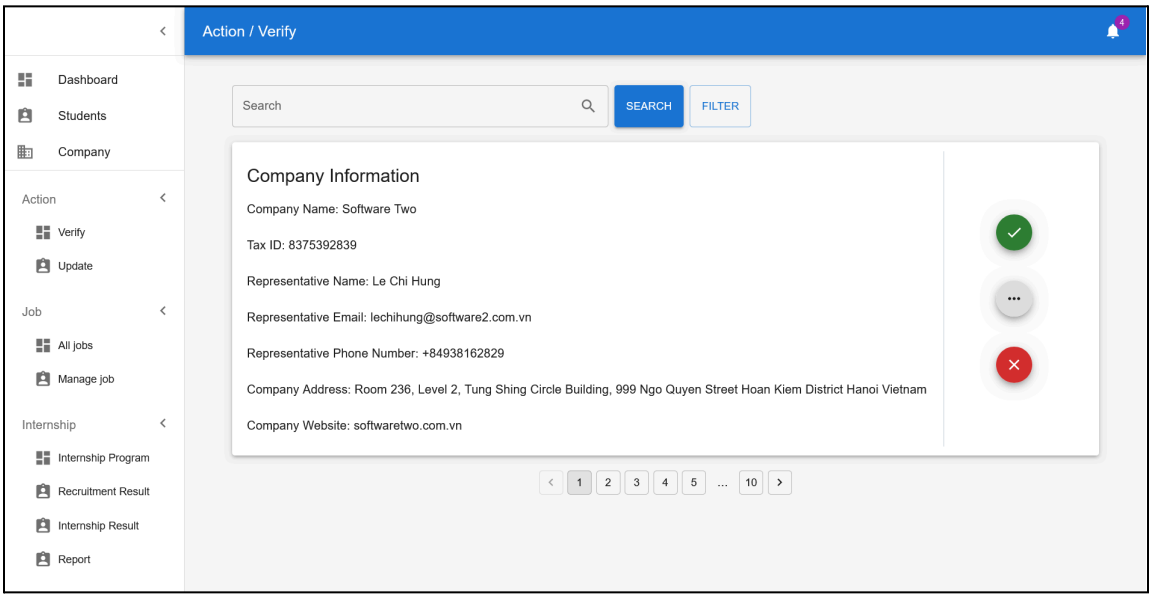


Figure 5.18: Staff - Verify company information page

In the Action/Verify page, staff will see cards displaying all detailed information of pending for the verification company, staff can choose to verify, deny, or some more actions such as message the company.

5.5. User Interface Prototypes

5.5.1. Home Page

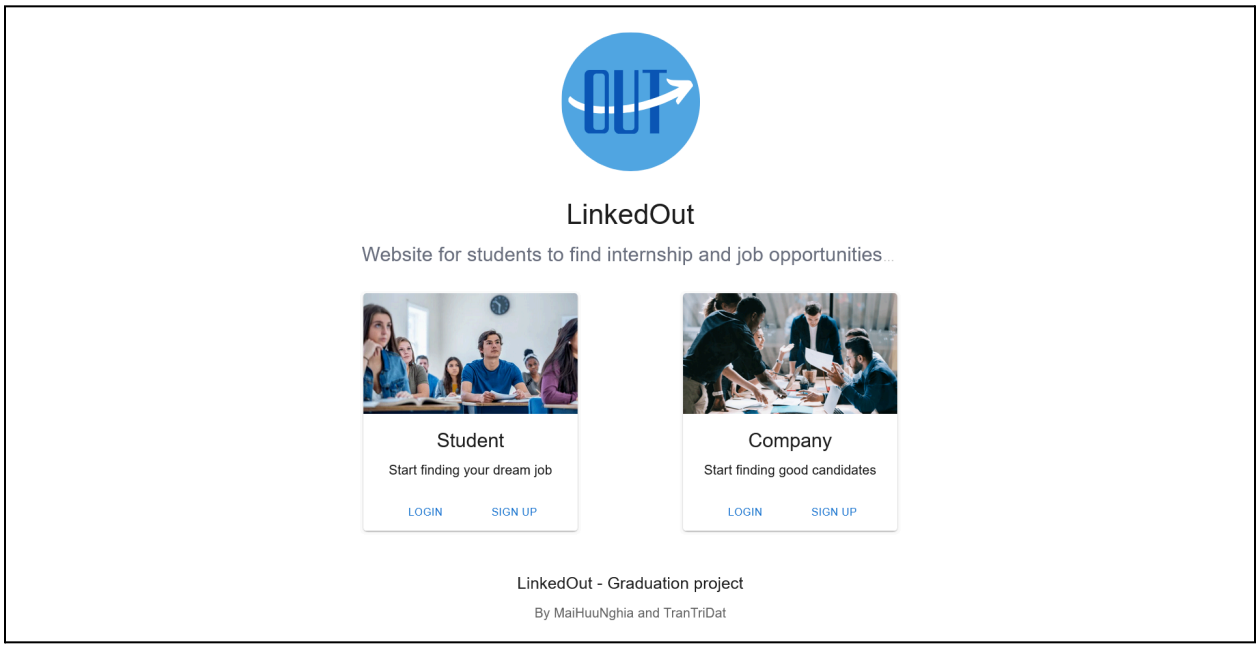


Figure 5.19: Home page

The page display our logo

5.5.2. Student - Login Page

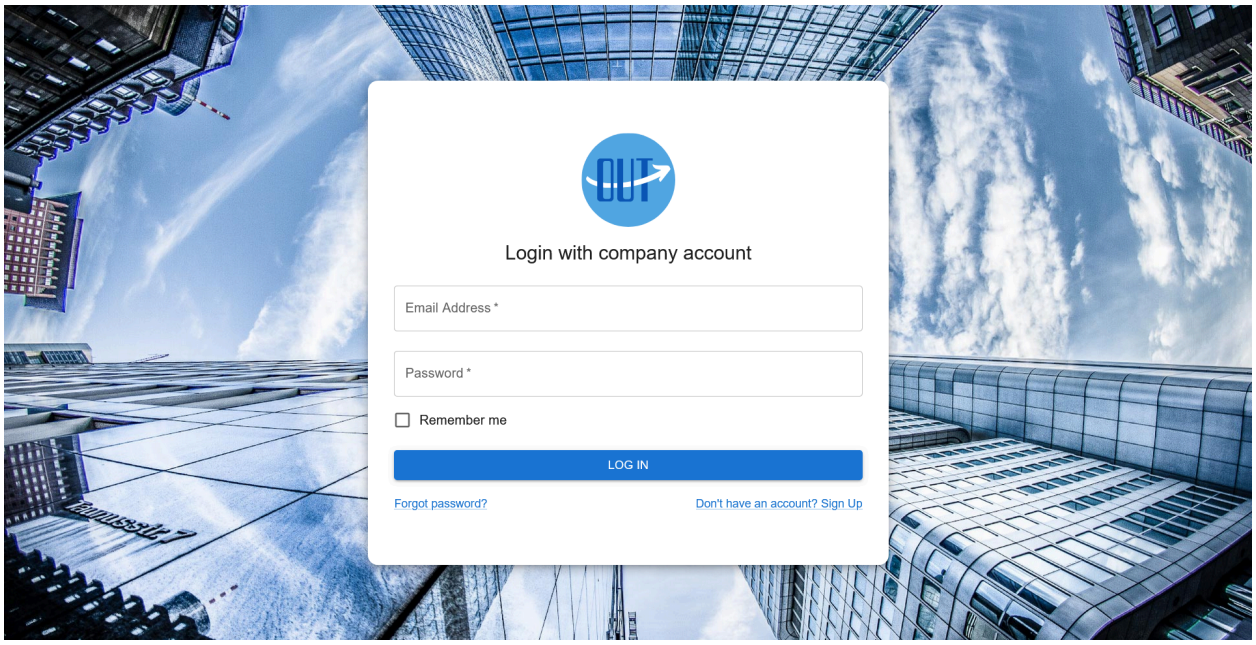


Figure 5.20: Student - Login page

5.5.3. Student - Sign-up Page

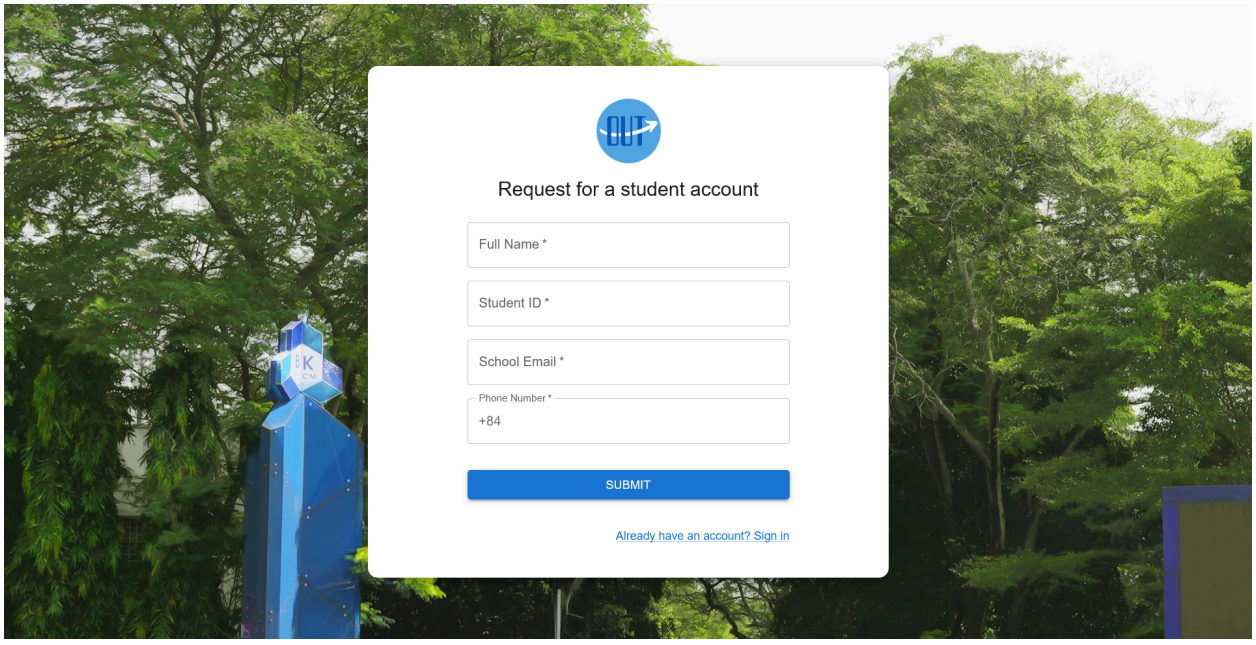


Figure 5.21: Student - Sign-up Page

5.5.4. Student - Home Page

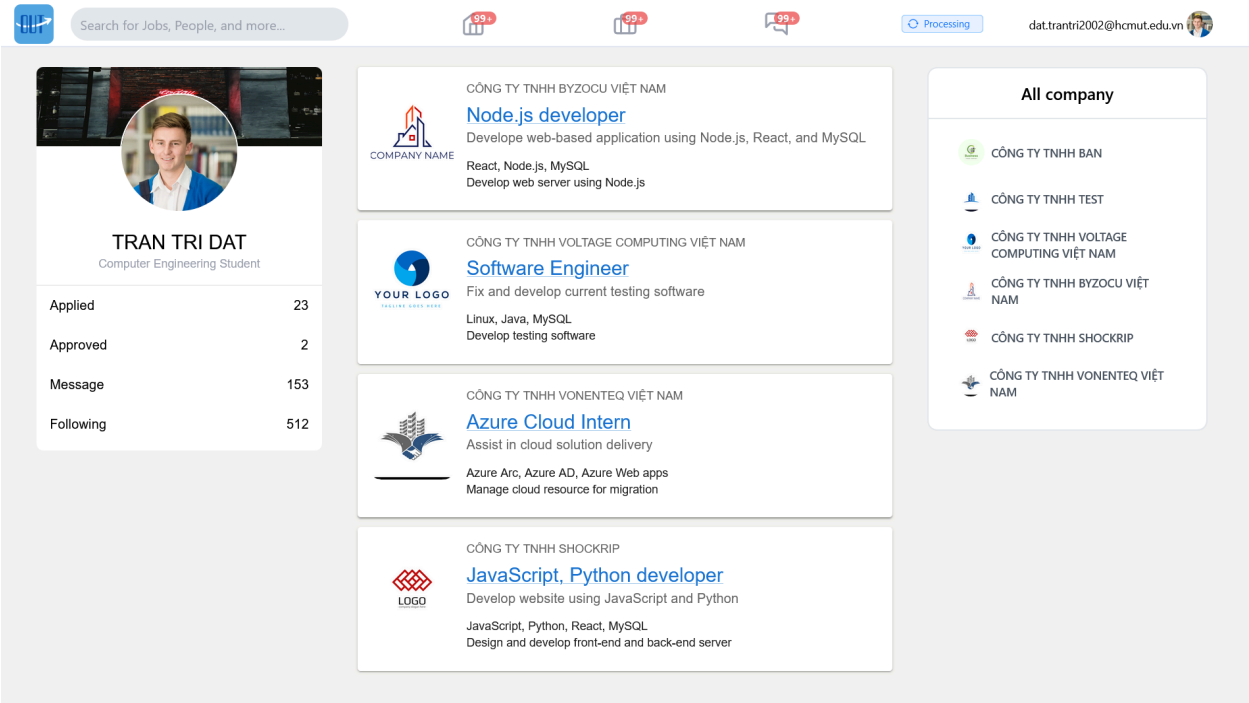


Figure 5.22: Student - Home page

5.5.5. Student - Jobs Page

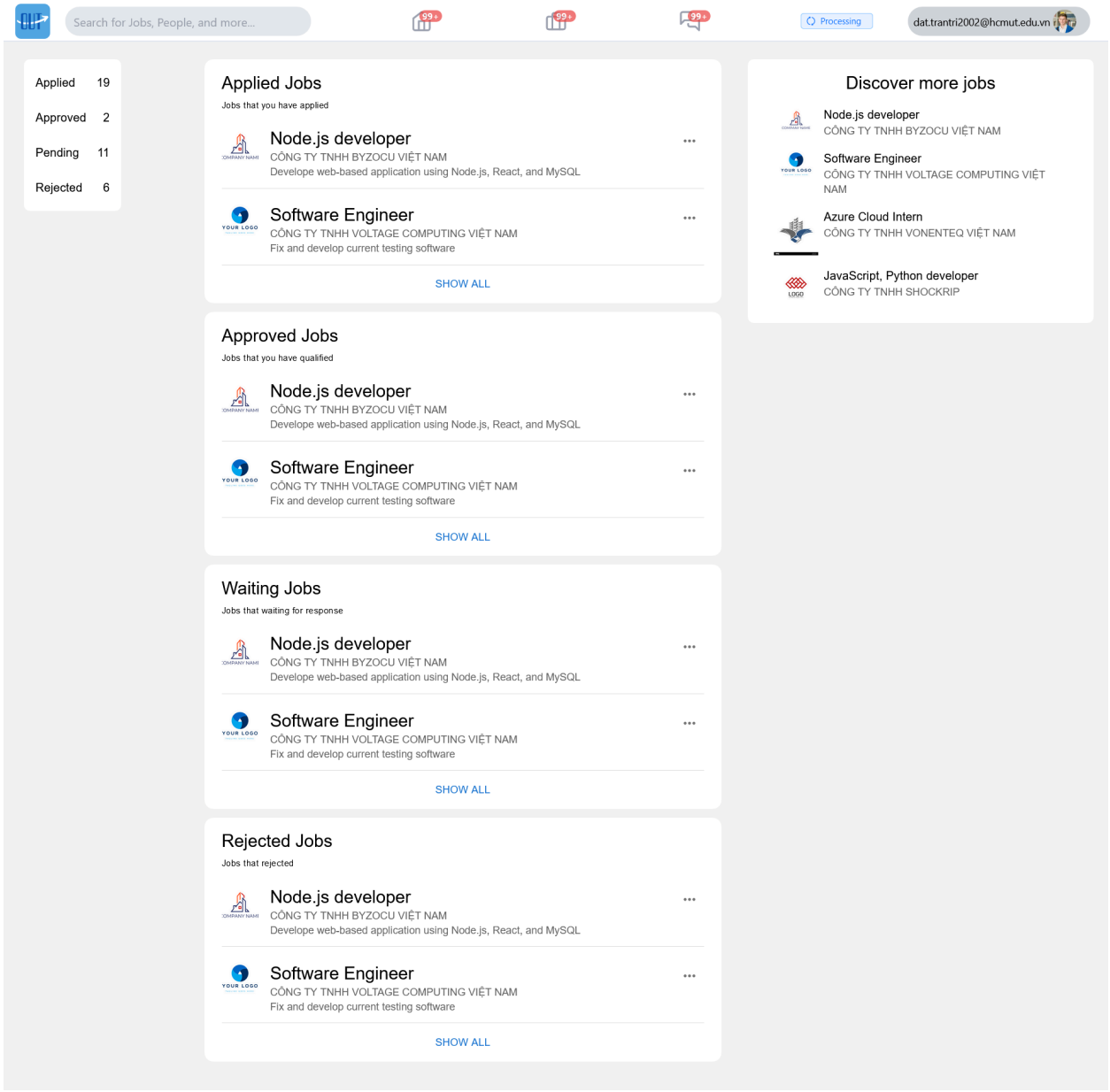


Figure 5.23: Student - Jobs page

5.5.6. Student - Job Display

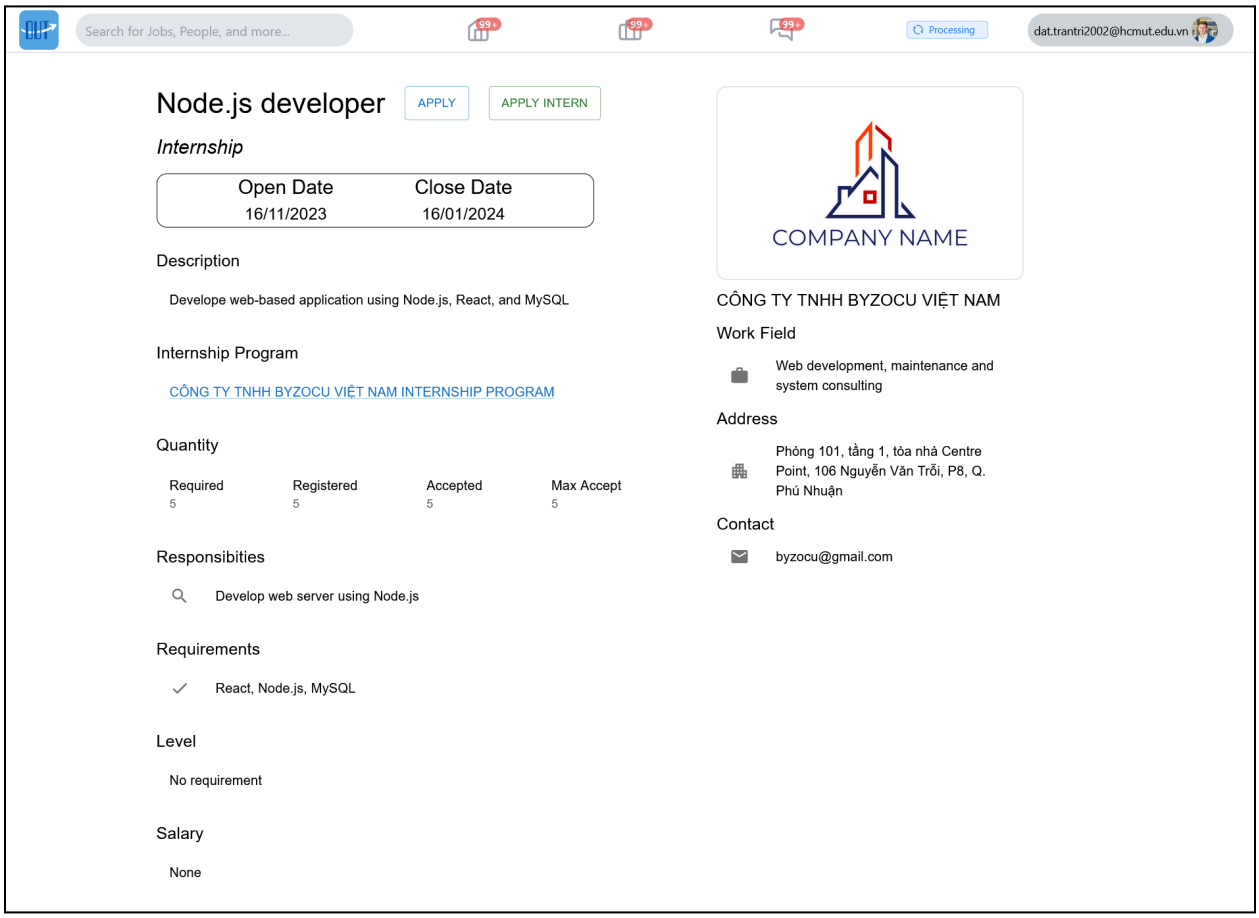


Figure 5.24: Student - Job Display

5.5.7. Student - Profile

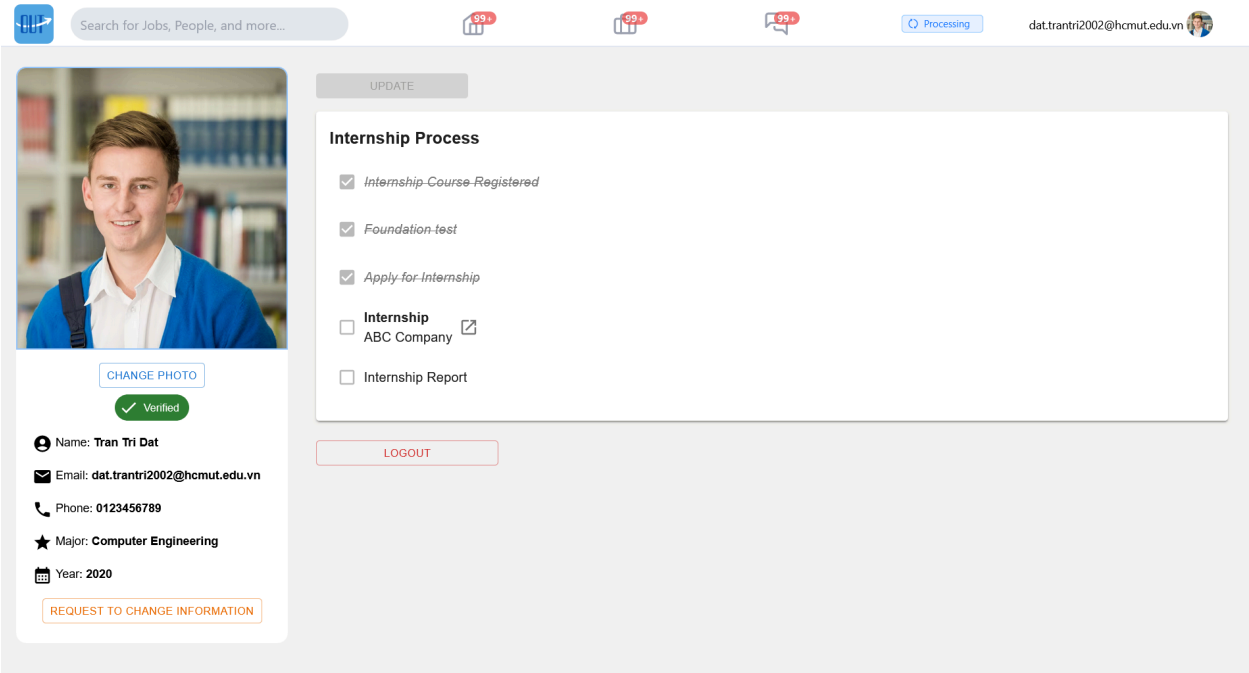


Figure 5.25: Student - Profile

5.5.8. Student - Profile Change Profile picture

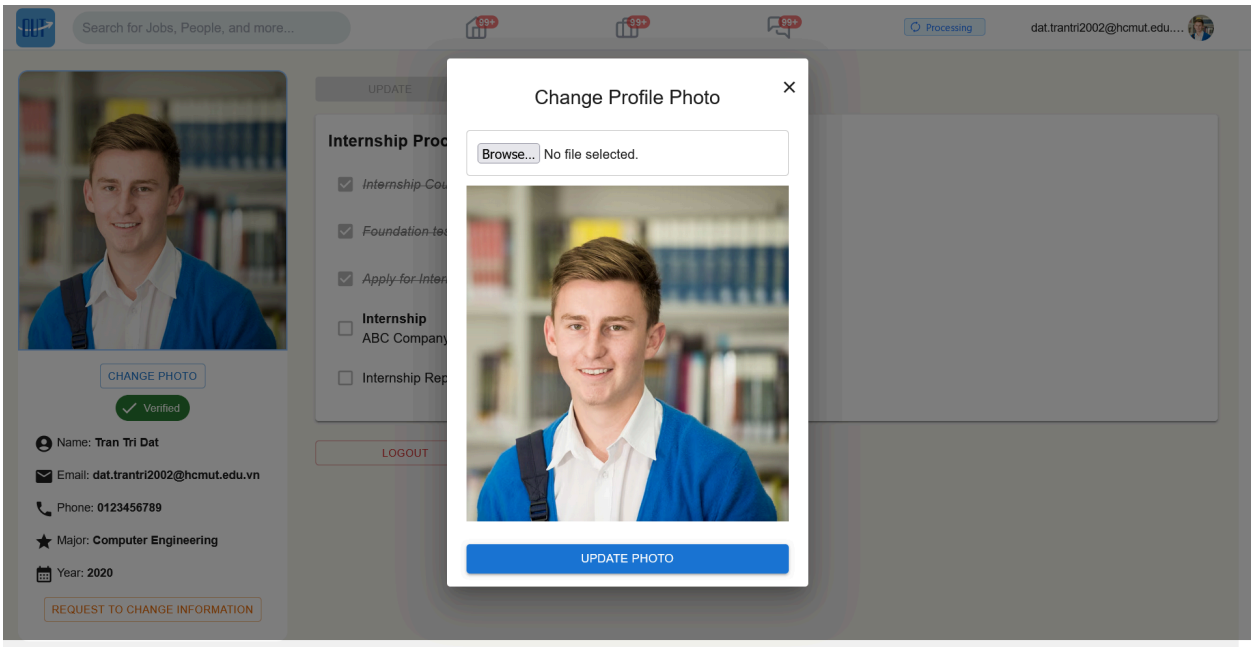


Figure 5.26: Student - Profile Change Profile picture

5.5.9. Student - Profile Update Information

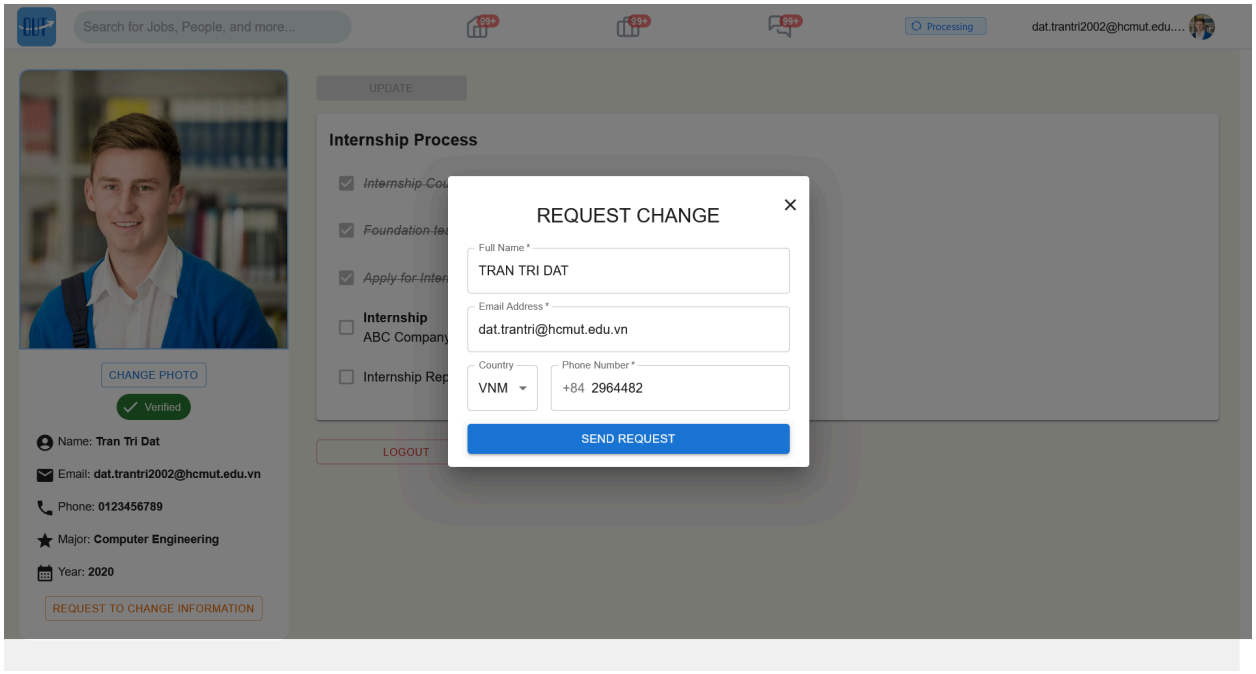


Figure 5.27: Student - Profile Update Information

5.5.10.Student - Message

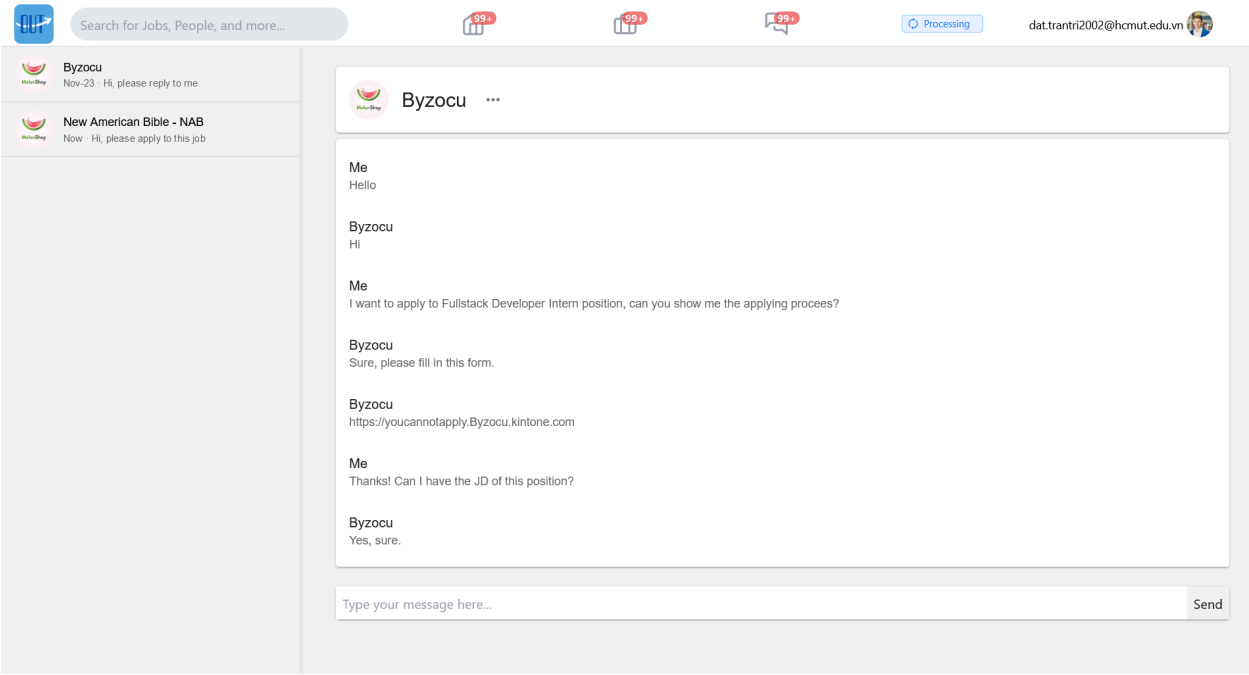


Figure 5.28: Student - Message

5.5.11. Company - Login Page

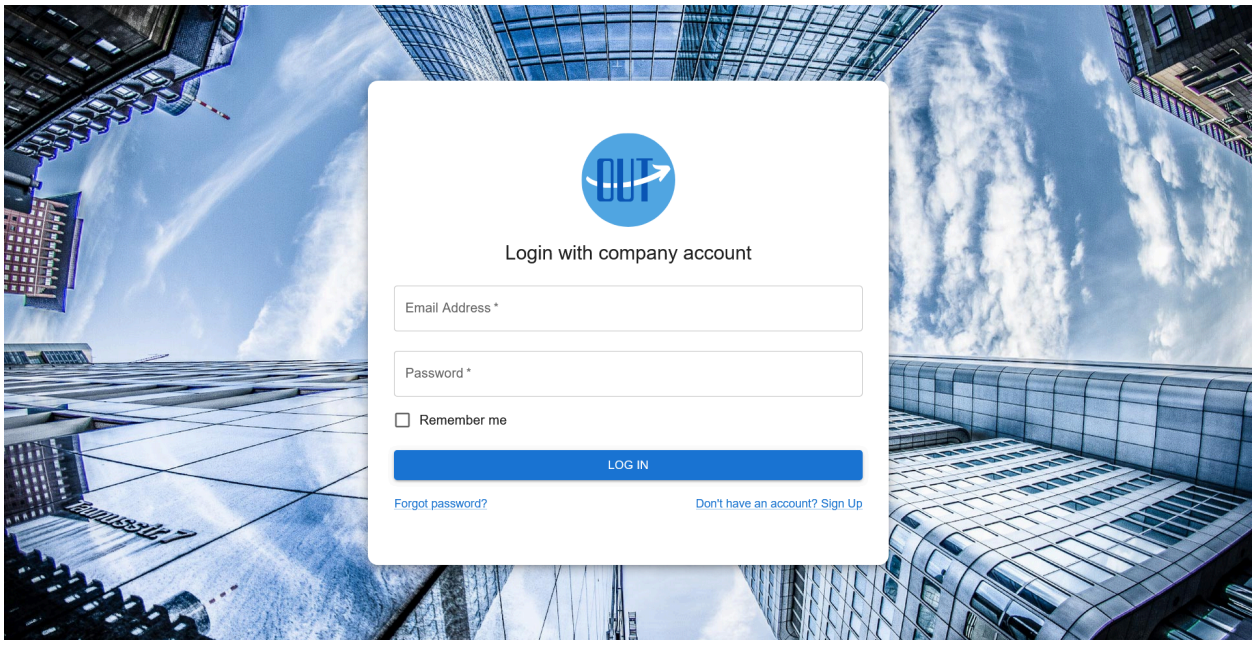


Figure 5.29: Company - Login Page

5.5.12.Company - Sign-up Page

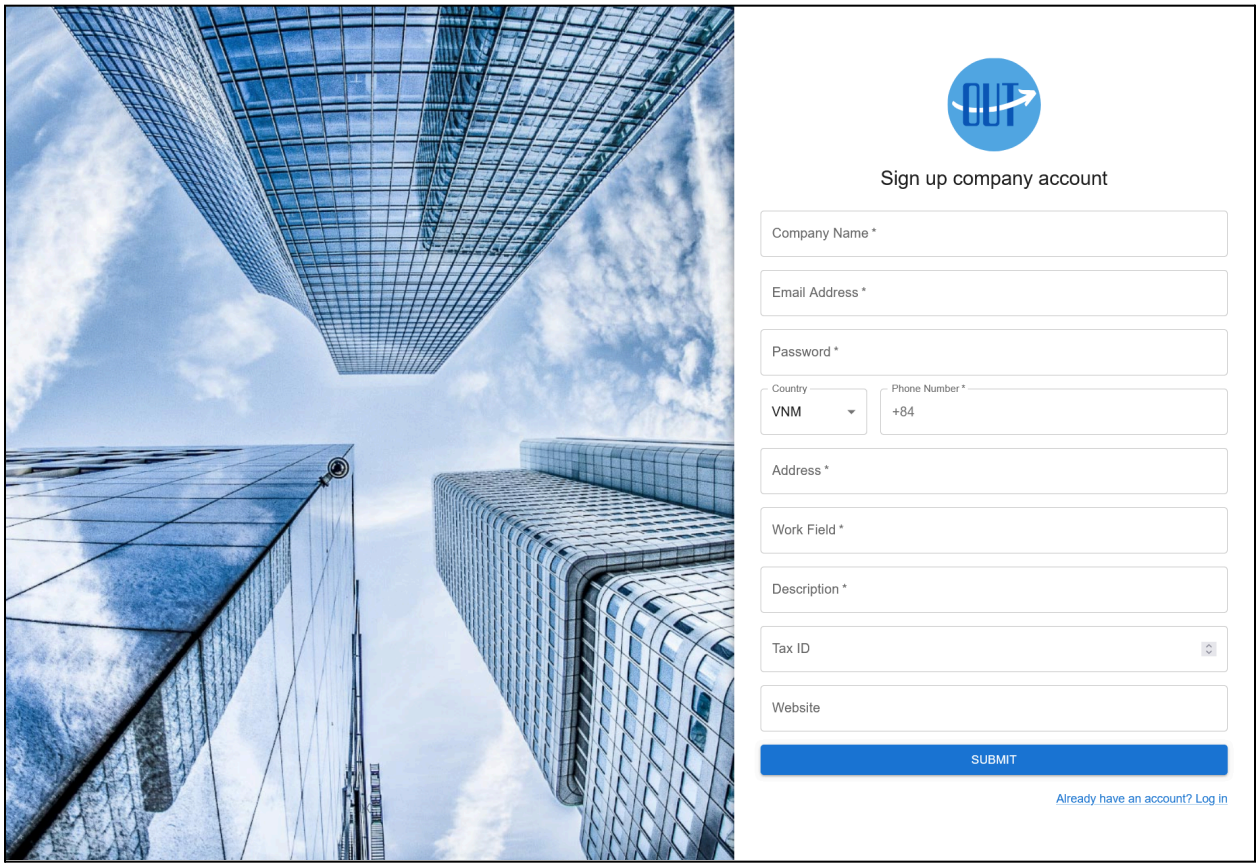


Figure 5.30: Company - Sign-up Page

5.5.13. Company - Home Page

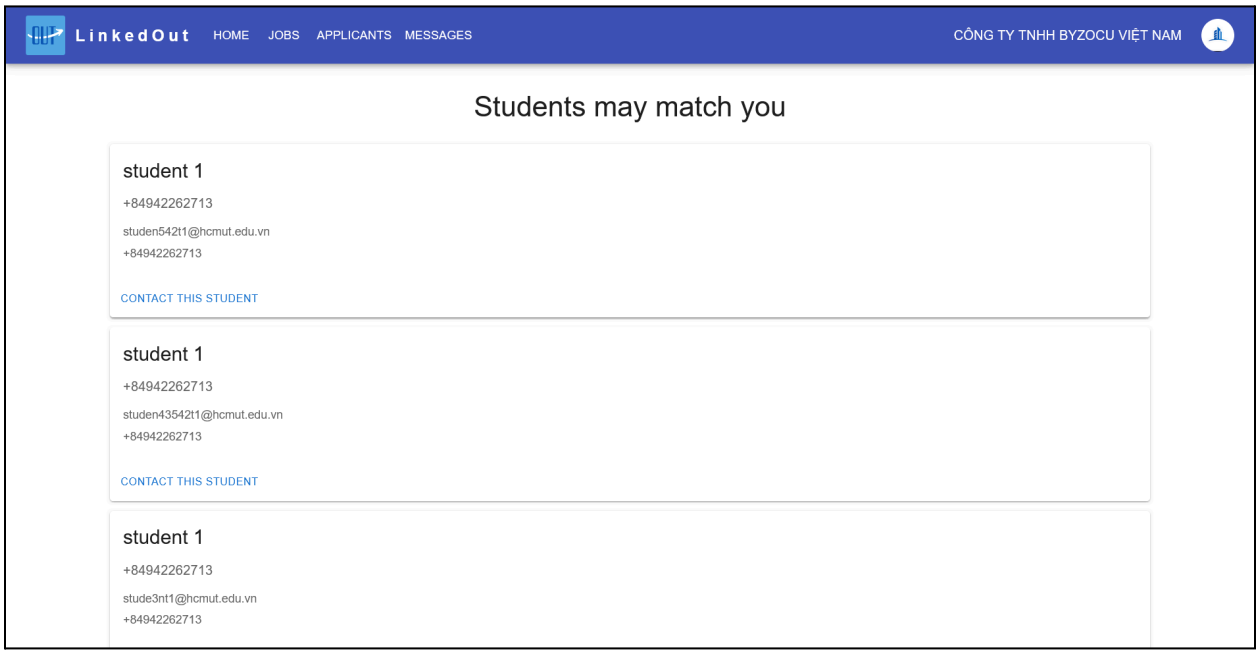


Figure 5.31: Company - Home Page

5.5.14.Company - Jobs Page

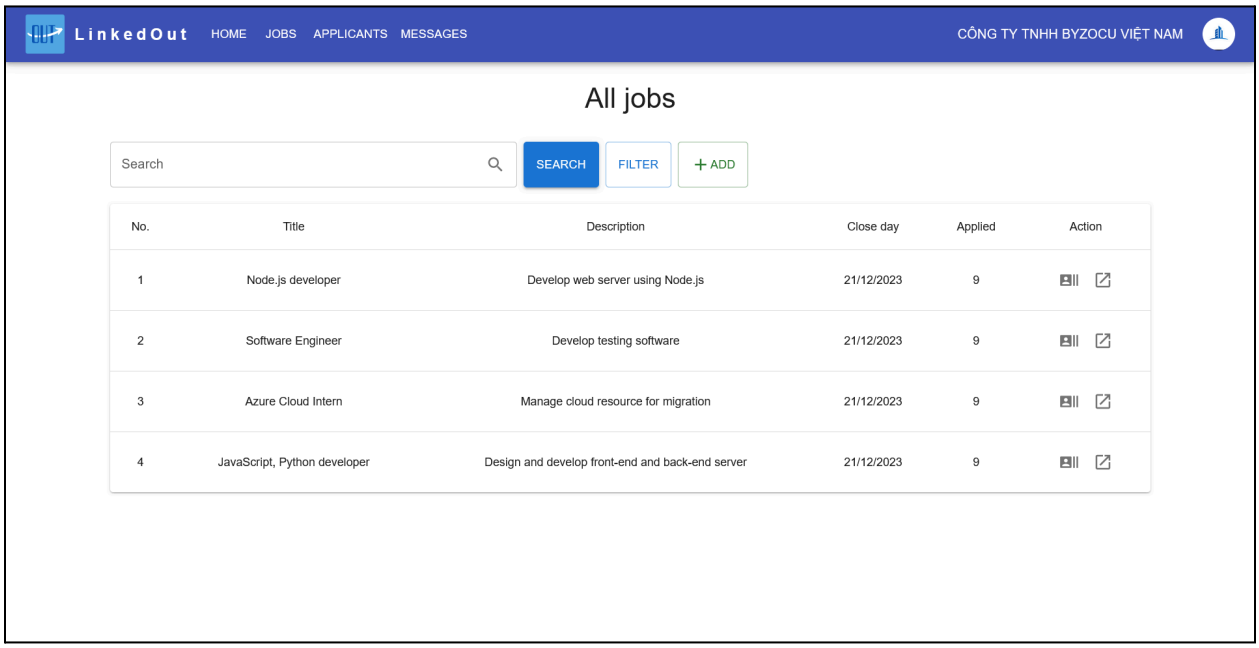


Figure 5.32: Company - Jobs Page

5.5.15.Company - Job Display

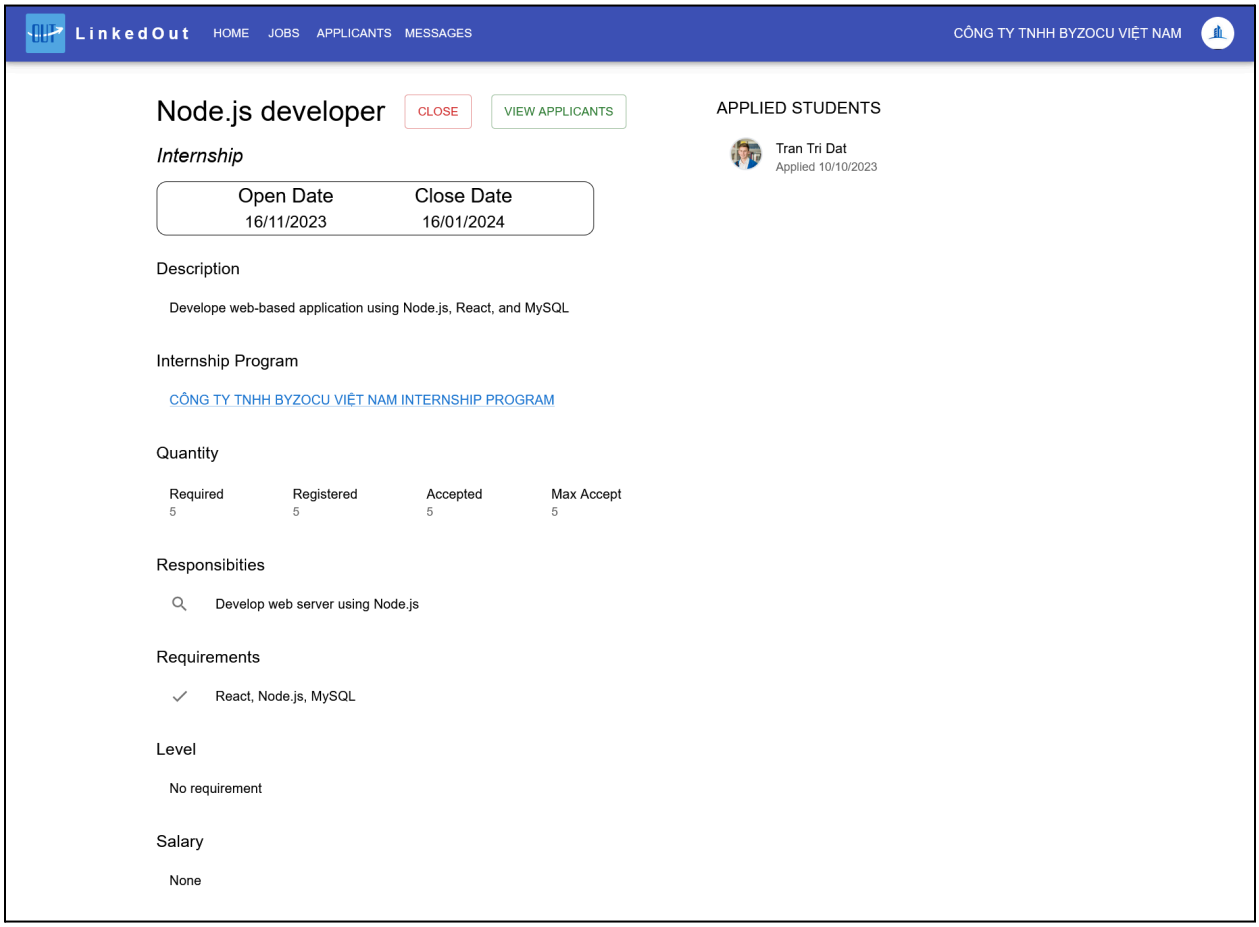


Figure 5.33: Company - Job Display

5.5.16.Company - Add Job

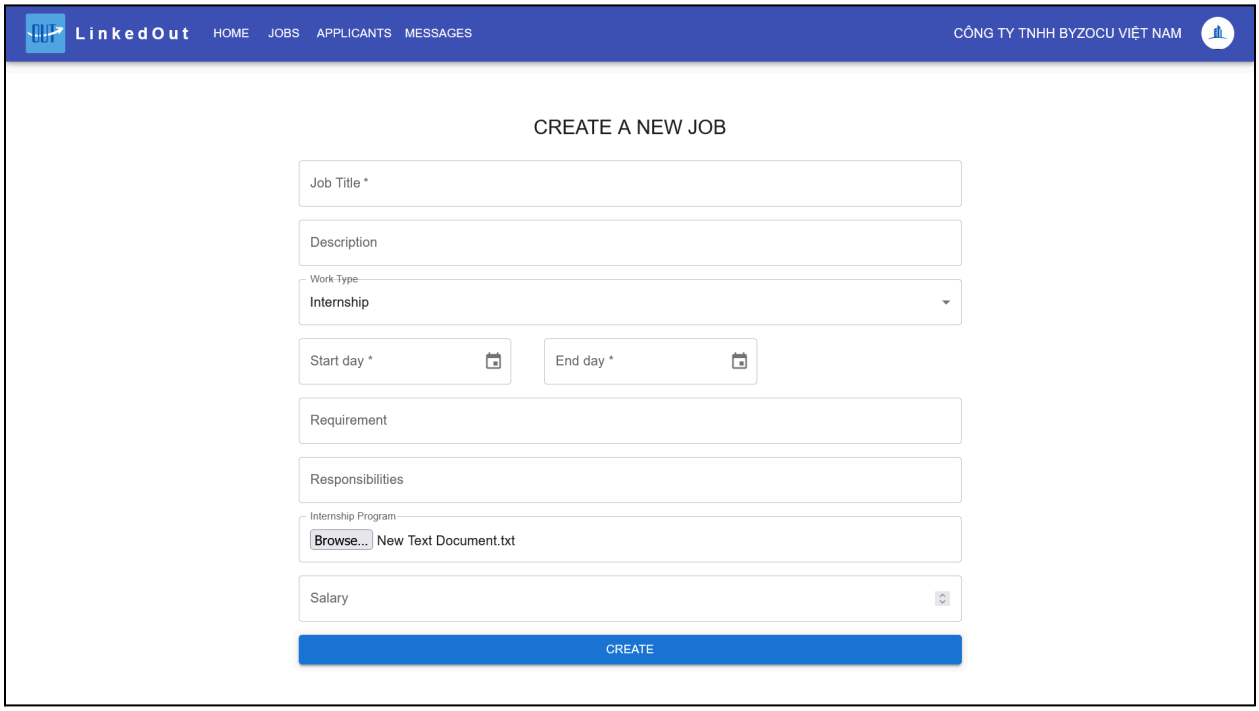


Figure 5.34: Company - Add Job

5.5.17.Company - Applicants Page

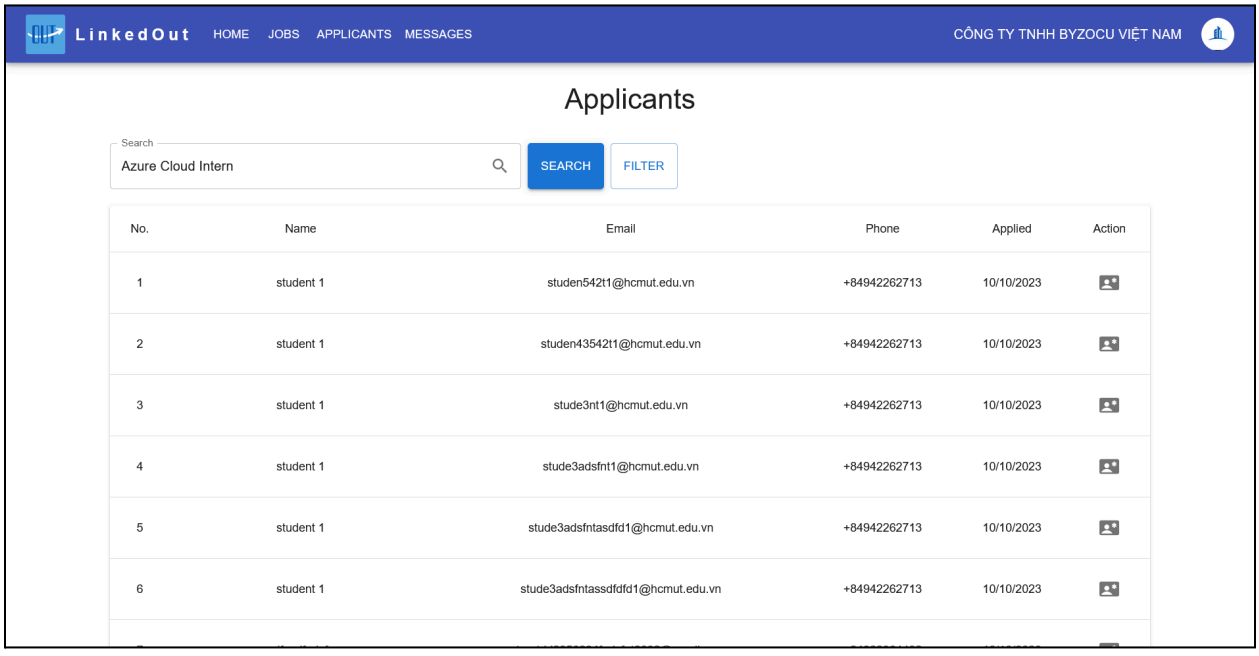


Figure 5.35: Company - Applicants Page

5.5.18.Company - Display Applicant

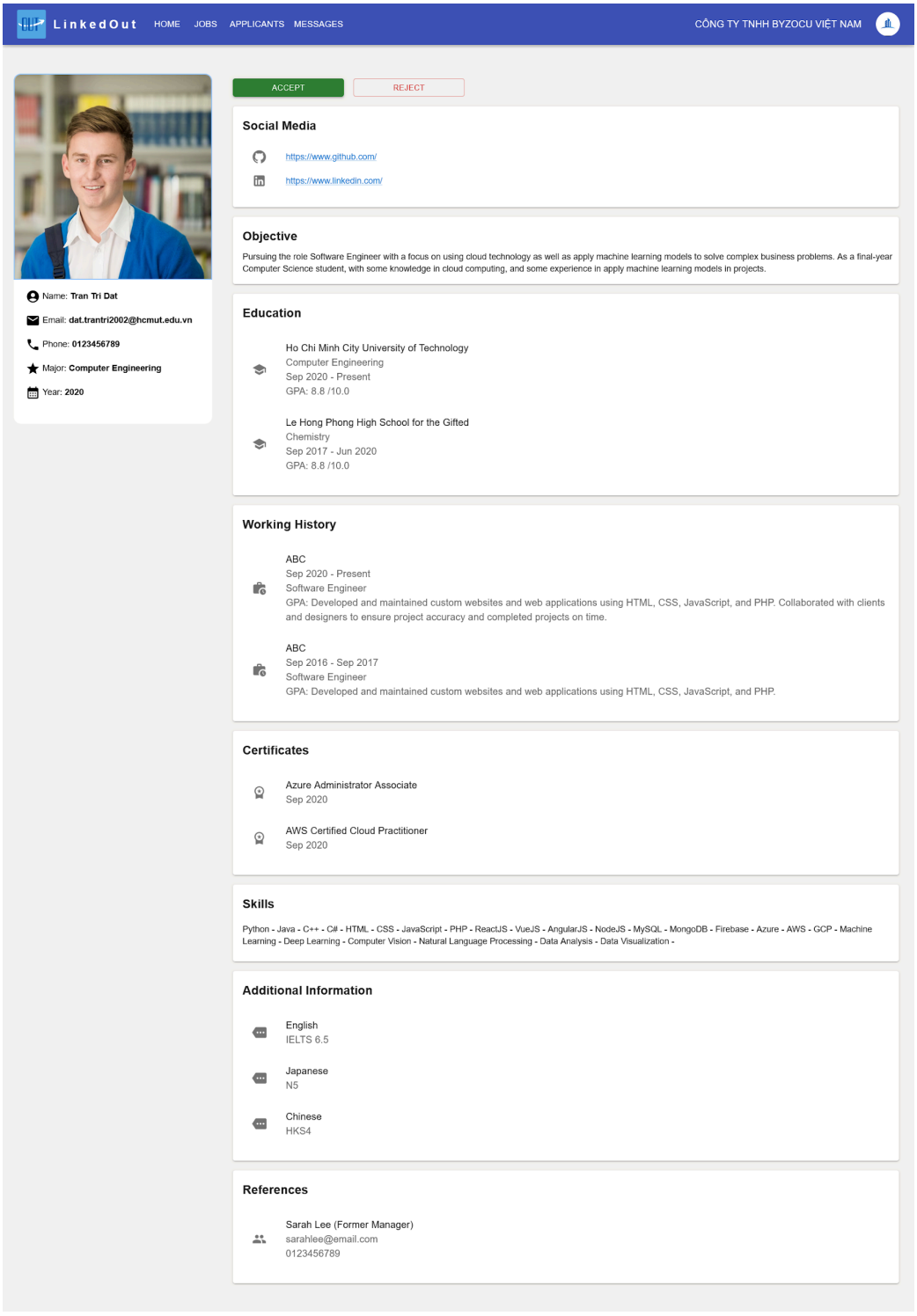


Figure 5.36: Company - Display Applicant

5.5.19.Company - Message Page

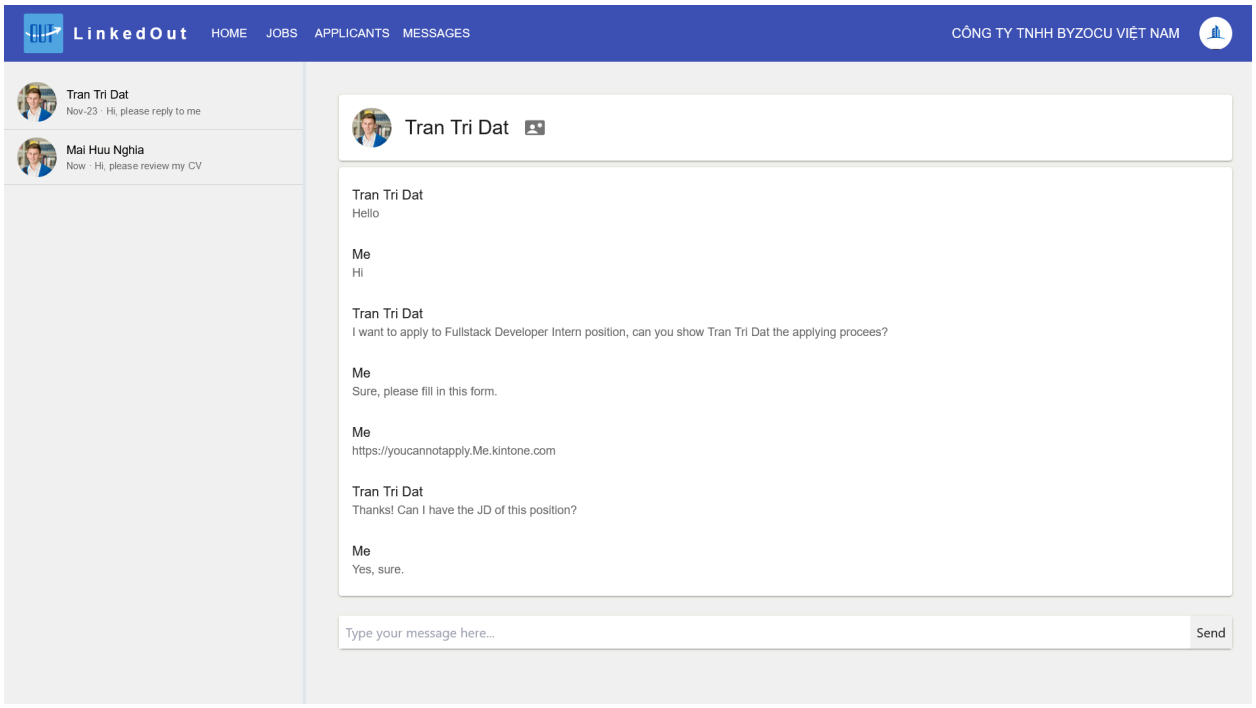


Figure 5.37: Company - Message Page

5.5.20.Company - Update Page

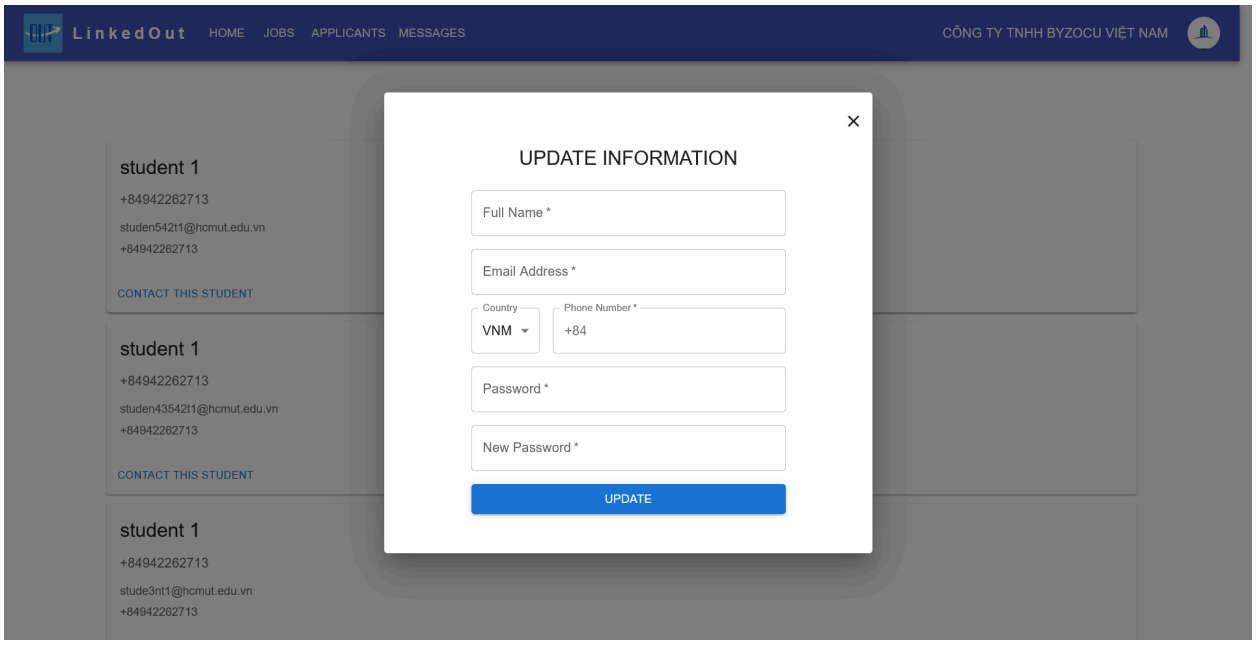


Figure 5.38: Company - Update Page

5.5.21.Staff - Dashboard Page

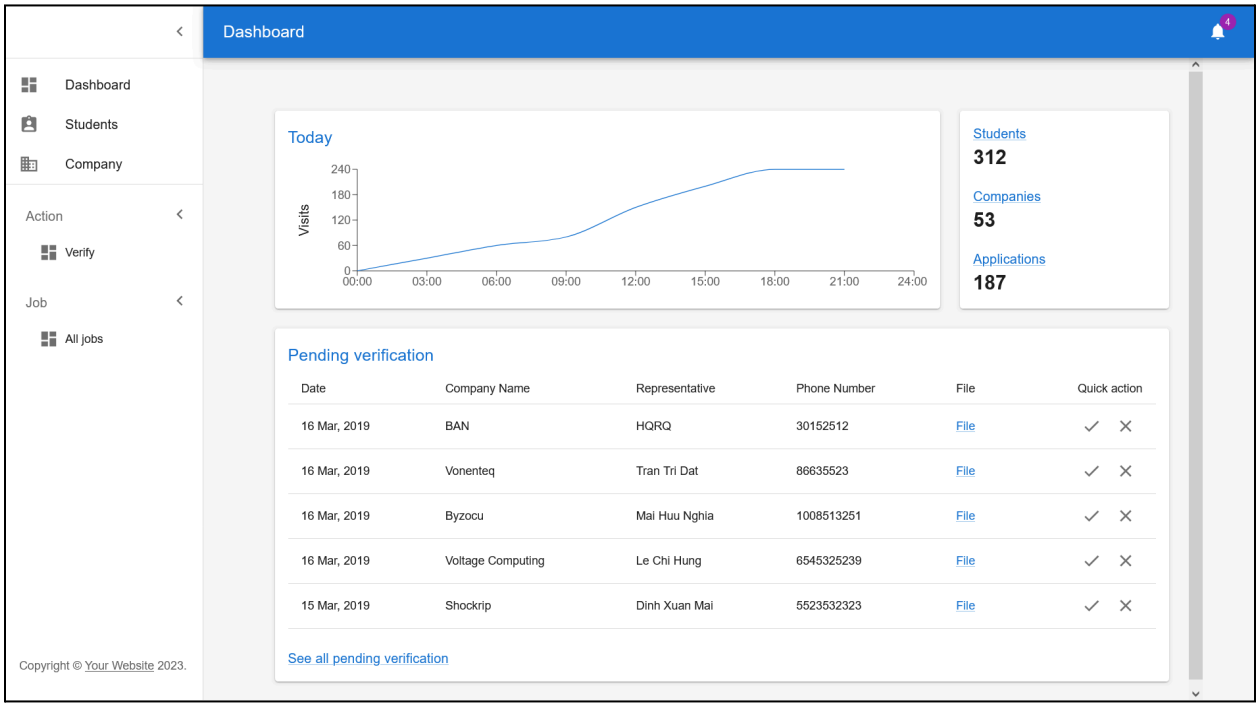


Figure 5.39: Staff- Dashboard Page

5.5.22. Staff - Company Page

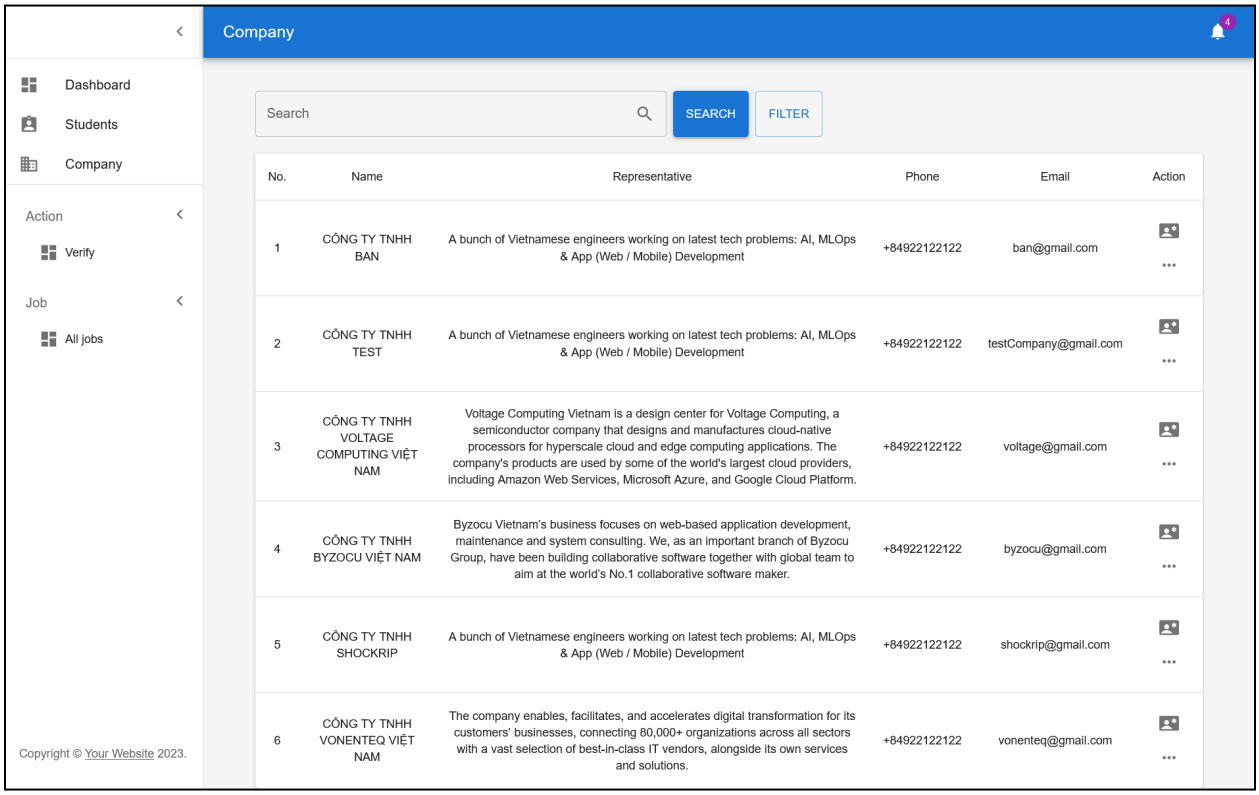


Figure 5.40: Staff- Company Page

5.5.23.Staff - Student Page

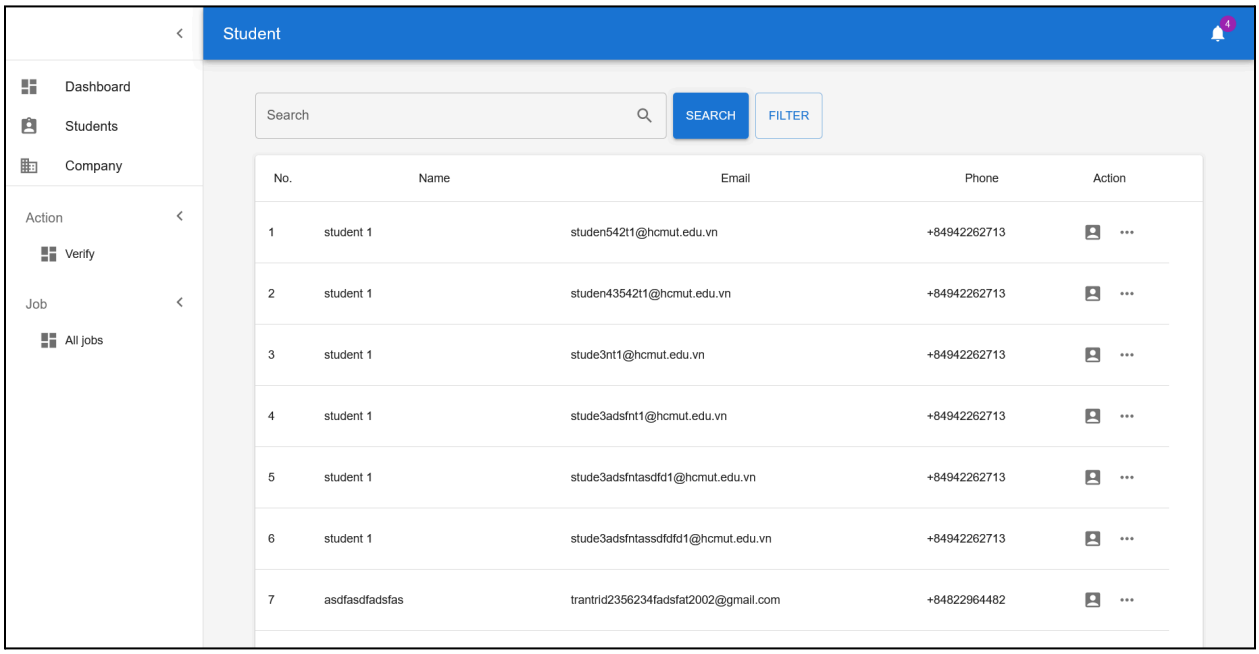


Figure 5.41: Staff- Student Page

5.5.24. Staff - Verify Page

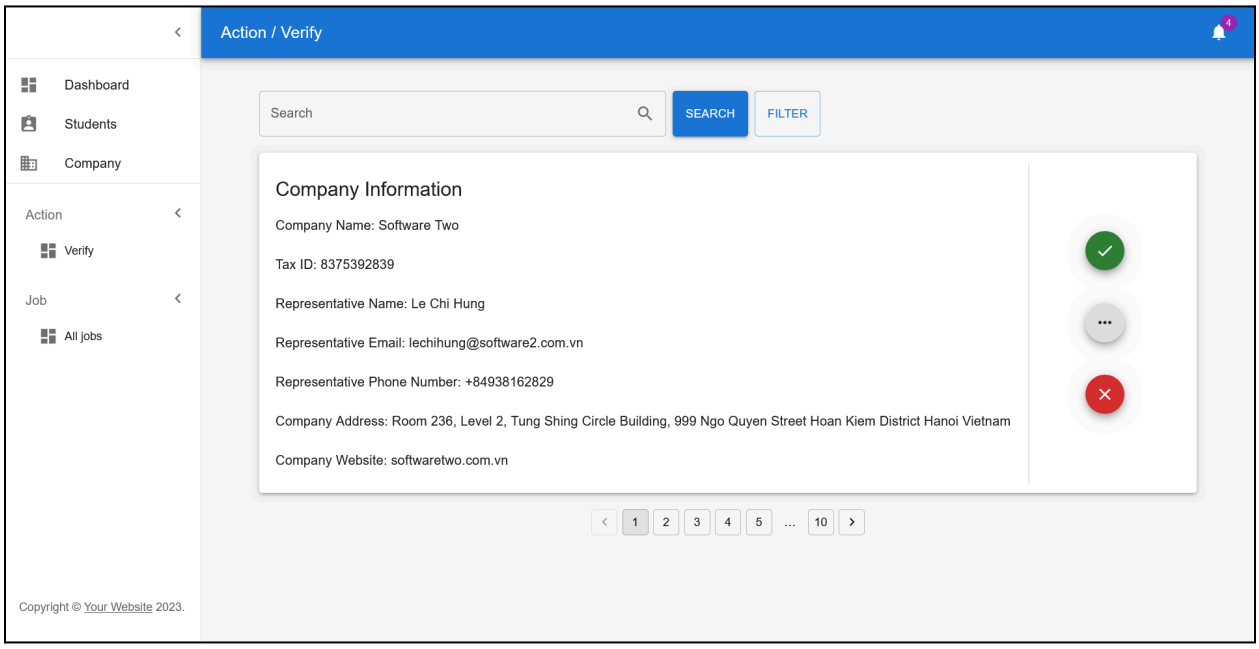


Figure 5.42: Staff- Verify Page

5.5.25. Staff - Jobs Page

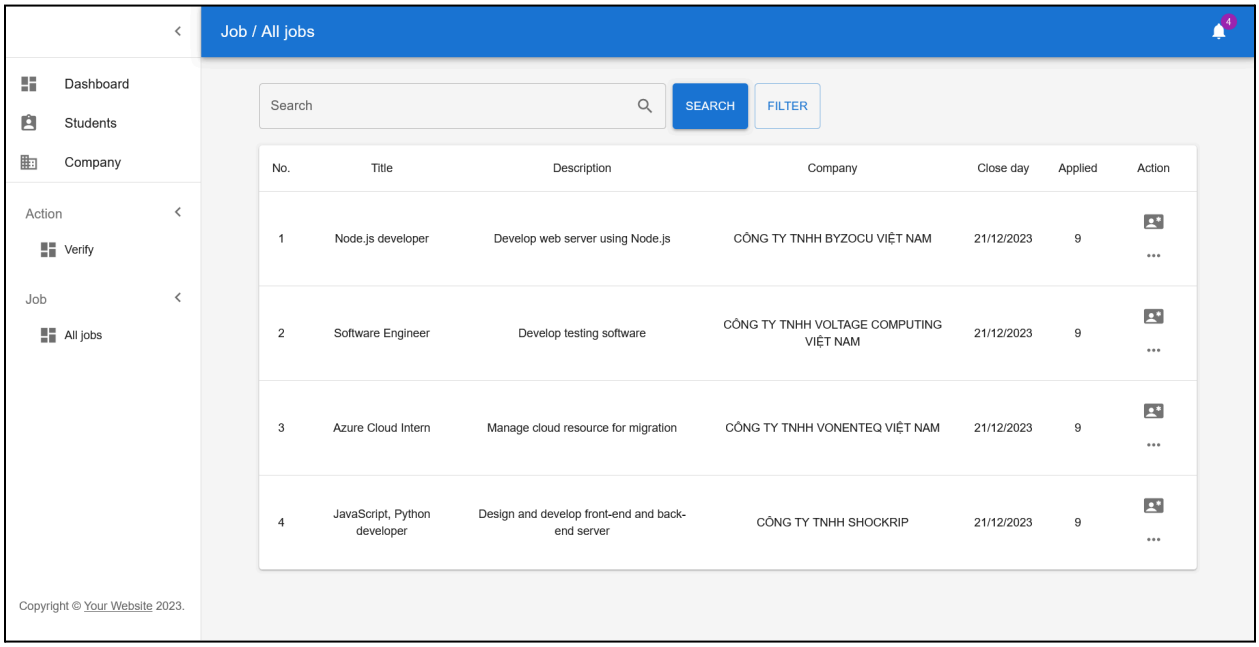


Figure 5.43: Staff- All Jobs Page

6. Technologies

6.1. Front-end

The front-end development of our project is a crucial component that ensures an intuitive and responsive user interface. The technologies chosen for this task are ReactJS, Material-UI, Zustand, Axios, Tailwind CSS, and Jest. Each of these technologies plays a specific role in enhancing the development process and delivering a seamless user experience.

6.1.1. ReactJS

ReactJS [3], developed and maintained by Facebook, is a JavaScript library for building user interfaces. It utilizes a declarative syntax to describe the state of a UI and efficiently updates the DOM when that state changes.

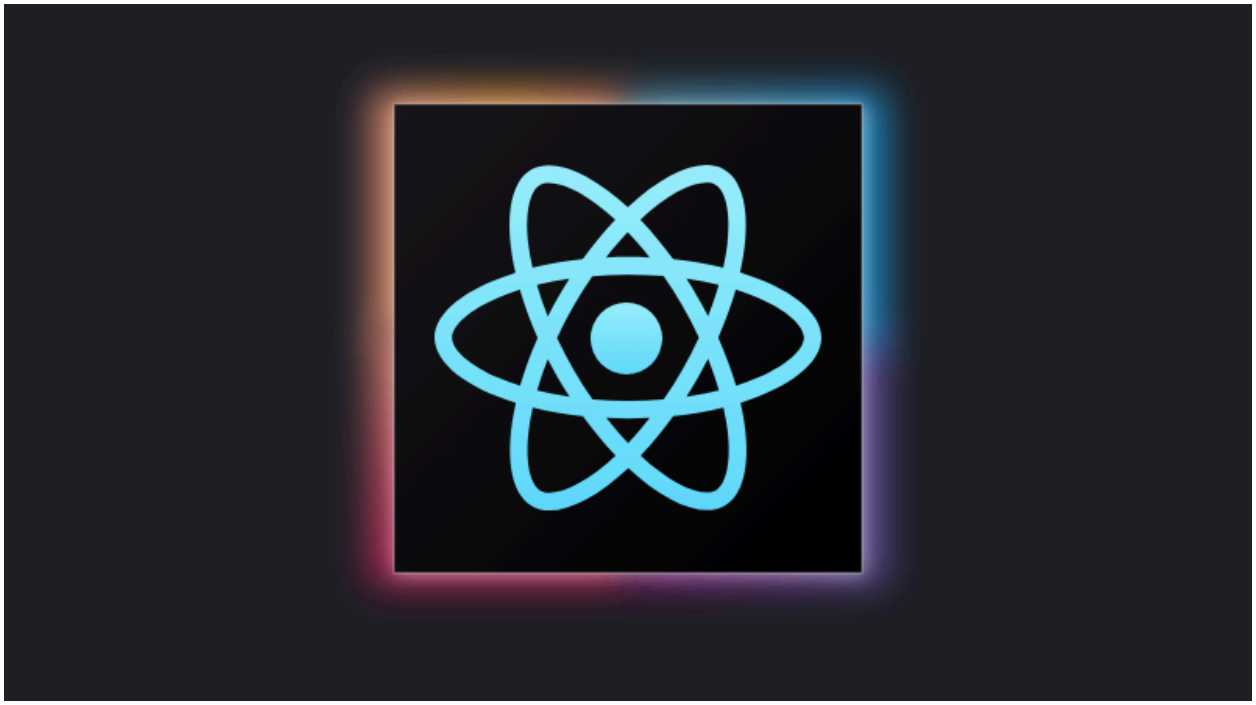


Figure 6.1: React Logo

Why do we use ReactJS [4]?

- **Component-Based Architecture:** React encourages a modular and reusable code structure through its component-based architecture. Each component encapsulates a piece of the user interface, making it easier to manage and update.
- **Virtual DOM:** React's Virtual DOM efficiently updates the actual DOM, reducing the need for direct manipulations. This results in faster rendering and improved application performance.
- **Large Ecosystem:** The extensive React ecosystem includes a variety of libraries and tools, fostering a rich development environment. This ecosystem provides solutions for routing, state management, and more.

Some pros of ReactJS:

- Declarative Syntax: Simplifies the process of understanding and maintaining code.
- Virtual DOM: Enhances performance by minimizing direct DOM manipulations.
- Reusability: Encourages the creation of modular and reusable components.
- Community Support: Active community with ample resources and third-party libraries.

6.1.2. MaterialUI

Material-UI [5] is a React UI framework that implements Google's Material Design principles. It offers a set of pre-designed and customizable components to maintain a consistent and visually pleasing UI.

Figure 6.2: Material UI Logo

Why do we use MaterialUI [6]:

- **Consistent Design:** Material-UI ensures a uniform design language throughout the application, following the principles of Material Design. This consistency enhances the overall user experience.
- **Customization:** Material-UI components are highly customizable, allowing developers to adapt the UI to specific project needs. This flexibility is crucial for maintaining brand identity.

- **Responsive:**The components provided by Material-UI are designed to be responsive, ensuring a seamless user experience across various devices and screen sizes.

Pros:

- Theming: Easy theming options for consistent branding.
- Pre-designed Components: Saves development time with ready-made, styled components.
- Active Community: Large community support and ongoing development.

6.1.3. Zustand

Figure 6.3: Zustand Logo

Zustand [7] is a state management library for React applications. It aims to provide a simple and effective solution for managing state in a React-based project.

Why do we use Zustand [8]:

- **Simplicity:** Zustand is known for its simplicity. It offers a minimalistic API, making it easy to understand and integrate into React applications.
- **Performance:** Designed with a focus on performance, Zustand is lightweight and ensures optimal performance with a small bundle size.
- **React Integration:** Zustand seamlessly integrates with React, aligning with the React philosophy and making it a natural choice for state management in React applications.

Pros:

- Lightweight: Minimal footprint contributes to faster loading times.

- Simple API: Easy to learn and use for managing state in React applications.

6.1.4. Axios

Axios [9] is a promise-based HTTP client for the browser and Node.js, allowing developers to make asynchronous HTTP requests.

Figure 6.4: Axios Logo

Why Axios [10]:

- **Simplicity:** Axios provides a straightforward and consistent API for handling HTTP requests. Its syntax is clean and easy to understand, simplifying the process of making HTTP calls.
- **Interceptors:** Axios allows the use of interceptors, enabling developers to manipulate request and response data. This feature is valuable for customization, error handling, and global configuration.
- **Promise-based:** Being promise-based, Axios facilitates the handling of asynchronous operations, providing a cleaner and more organized code structure.

Pros:

- Concise Syntax: Clean and concise syntax for making HTTP requests.
- Interceptors: Enhanced control over request and response handling.
- Wide Adoption: A widely used and well-supported library.

6.1.5. Tailwind CSS

Tailwind CSS [11][12] is a utility-first CSS framework that provides low-level utility classes to build designs directly in the markup.

Figure 6.5: Tailwind CSS Logo

Why Tailwind CSS:

- **Utility-First Approach:** Tailwind CSS follows a utility-first approach, allowing developers to build designs rapidly by applying pre-defined utility classes. This approach enhances speed and flexibility in styling.
- **Customization:** Tailwind CSS is highly customizable through configuration. Developers can easily create and maintain a tailored design system that aligns with the project's specific requirements.
- **Responsive Design:** The framework includes built-in features for creating responsive designs, ensuring a seamless user experience across various devices.

Pros:

- Rapid Development: Speeds up the development process with utility classes.
- Flexibility: Easy customization to match specific design requirements.
- Responsive Design: Built-in features for creating responsive layouts.

6.1.6. Overall Benefits

- Integration: The selected technologies seamlessly integrate with each other, providing a cohesive development environment.
- Performance: The combination of React, Zustand, Axios, and the lightweight nature of Tailwind CSS contributes to a performant front-end.
- Developer Experience: The simplicity of Zustand and the utility-first approach of Tailwind CSS enhance the developer experience, promoting efficiency and maintainability.
- Testing: Jest facilitates the testing process, ensuring the reliability and correctness of the application.

In conclusion, the chosen stack combines the power of ReactJS for building dynamic and modular user interfaces, Material-UI for consistent and visually appealing design, Zustand for lightweight state management, Axios for efficient HTTP requests, Tailwind CSS for rapid and customizable styling, and Jest for robust testing. This stack is well-suited for a modern, responsive, and maintainable front-end application.

6.2. Backend

6.2.1. NestJS

NestJS [13] is a full-featured, extensible Node.js framework that combines elements of Object-Oriented Programming (OOP), Functional Programming (FP), and Functional Reactive Programming (FRP). It is built with TypeScript and takes inspiration from Angular to provide a modular and scalable architecture.

Figure 6.6: NestJS Logo

Why NestJS [13]:

- **Modularity and Structure:** NestJS organizes code into modules, controllers, and services, fostering a modular and organized codebase. This structure makes it easier to manage complex applications by breaking them into smaller, manageable components.

- **TypeScript Support:** TypeScript brings static typing to JavaScript, reducing common runtime errors and providing a more robust development experience. NestJS leverages TypeScript for strong typing, interfaces, and decorators, enhancing code quality and maintainability.
- **Decorators and Dependency Injection:** NestJS makes extensive use of decorators and dependency injection. Decorators streamline the process of defining routes, middleware, and other components, while dependency injection simplifies the management of dependencies and promotes code reusability.
- **Middleware and Guards:** NestJS supports middleware and guards, allowing developers to inject logic before or after the request reaches the route handler. This enables the implementation of authentication, logging, and other cross-cutting concerns.

Pros:

- Modular Architecture: Facilitates scalability and maintainability by organizing code into modules.
- TypeScript: Enhances code quality, readability, and developer productivity.
- Decorators and Dependency Injection: Streamlines code structure and promotes maintainability.
- Middleware and Guards: Enables the implementation of cross-cutting concerns.

6.2.2. Typeorm

TypeORM [14] is an Object-Relational Mapping (ORM) library for TypeScript and JavaScript. It simplifies database interactions by providing a high-level, TypeScript-friendly API to perform CRUD operations on various relational databases.

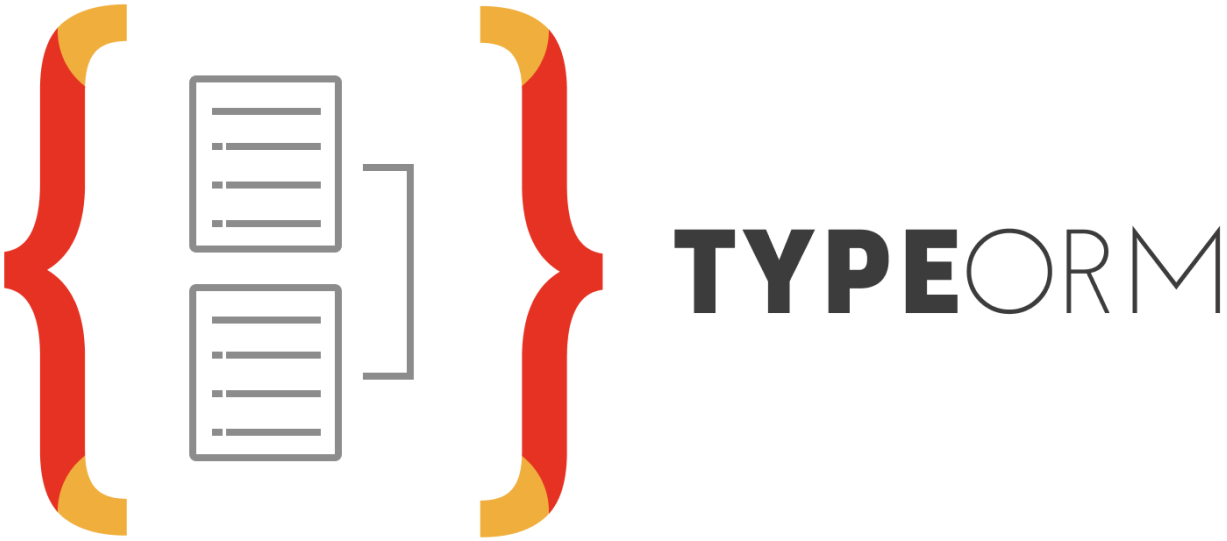


Figure 6.7: Typeorm Logo

Why TypeORM [14]:

- **TypeScript Integration:** TypeORM seamlessly integrates with TypeScript, allowing developers to use TypeScript features such as decorators, interfaces, and types. This integration ensures type safety during database interactions, reducing the likelihood of runtime errors.
- **Database Agnosticism:** TypeORM supports a variety of databases, including PostgreSQL, MySQL, SQLite, and MongoDB. This agnosticism allows developers to choose a database that best fits the project's requirements, and it facilitates easy migration between databases if needed.

- **Entity Relationship Mapping (ERM):** TypeORM simplifies database interactions through Entity Relationship Mapping (ERM). Developers can define entities in their code that map to tables in the database. This abstraction makes it easier to work with databases by using familiar programming concepts.
- **Migrations:** TypeORM provides migration support, allowing developers to version-control and manage database schema changes over time. This is crucial for maintaining consistency between the application's codebase and the underlying database structure.

Pros:

- TypeScript Integration: Ensures type safety and consistency between the application and the database.
- Database Agnosticism: Provides flexibility in choosing and switching between different databases.
- Entity Relationship Mapping: Simplifies database operations with a clear mapping between code and database entities.
- Migrations: Facilitates version control and management of database schema changes.

6.2.3. Overall benefit

- **Seamless Integration:** NestJS and TypeORM are designed to work seamlessly together, providing a unified and efficient development experience for backend development.
- **Consistent TypeScript Environment:** The use of TypeScript in both NestJS and TypeORM ensures a consistent and type-safe environment, reducing the likelihood of bugs and enhancing developer productivity.
- **Modular and Scalable Backend:** NestJS's modular structure combined with TypeORM's entity-based approach results in a backend that is both modular and scalable. This is crucial for managing complexity in large applications.
- **Flexibility in Database Choice:** TypeORM's support for multiple databases gives developers the flexibility to choose the most suitable database for the project's specific requirements.
- **ORM Convenience:** TypeORM's ORM capabilities simplify database interactions, allowing developers to focus more on application logic and less on the intricacies of database communication.

In conclusion, the combination of NestJS and TypeORM provides a powerful and flexible backend solution. NestJS's modular and TypeScript-friendly architecture, coupled with TypeORM's simplified database interactions, creates a development environment that is not only efficient and scalable but also maintains consistency throughout the entire application stack. This backend stack is well-suited for building robust, maintainable, and scalable server-side applications.

6.3. Database: PostgreSQL

Figure 6.8: PostgreSQL Logo

Choosing the right database management system (DBMS) is a critical decision for any project, as it directly impacts performance, scalability, and overall project success. In this report, we will compare PostgreSQL [15] and MySQL, two popular relational database management systems, highlighting the strengths of PostgreSQL over MySQL. Additionally, we will briefly discuss the advantages of MySQL over NoSQL databases.

PostgreSQL vs. MySQL:

- Extensibility and Customization:
    - PostgreSQL excels in extensibility, allowing users to define their data types, operators, and functions. This flexibility is beneficial for projects with complex data requirements.
    - MySQL, while customizable, may not offer the same level of extensibility as PostgreSQL.
- Concurrency Control:
    - PostgreSQL boasts advanced concurrency control mechanisms, making it suitable for applications with high transaction loads. Its Multi-Version

Concurrency Control (MVCC) ensures efficient handling of concurrent database access.

- MySQL, while effective in managing concurrent transactions, may not match the level of sophistication provided by PostgreSQL.

- Data Integrity and Compliance:
  - PostgreSQL is renowned for its robust data integrity enforcement, supporting complex constraints and validations. It complies with the SQL standard more strictly than MySQL.
  - MySQL, while adhering to SQL standards, may be more permissive in certain cases, potentially leading to less strict data integrity.
- Scalability:
  - PostgreSQL is often considered more scalable in terms of handling read-intensive workloads and large datasets. It excels in scenarios requiring horizontal scaling.
  - MySQL is highly scalable as well but may require careful configuration to match the scaling capabilities of PostgreSQL.
- Advanced Features:
  - PostgreSQL supports advanced features such as full-text search, geospatial data, and JSONB (binary JSON) data type. These features make it suitable for a wide range of project requirements.
  - MySQL offers similar features, but PostgreSQL's implementation is often considered more mature and feature-rich.

MySQL vs. NoSQL [16][17]**:**

- Data Model:
  - MySQL follows a relational data model, making it suitable for projects where data relationships are well-defined.

  - NoSQL databases, on the other hand, offer flexibility in terms of data modeling, making them a better choice for projects with evolving or undefined data structures.
- Scalability and Performance:
  - MySQL is vertically scalable, meaning it can handle increased load by adding more resources to a single server. It is well-suited for projects with predictable workloads.
  - NoSQL databases, like MongoDB or Cassandra, excel in horizontal scalability, making them a preferred choice for projects requiring distributed databases and dynamic scaling.
- Complex Queries and Joins:
  - MySQL supports complex queries and joins, making it suitable for projects with a need for relational data analysis.
  - NoSQL databases sacrifice some of this flexibility for the sake of performance and scalability, often avoiding complex joins in favor of simpler data retrieval methods.

Conclusion:

In conclusion, PostgreSQL stands out for its extensibility, concurrency control, data integrity, and support for advanced features. While MySQL is a robust relational database, its advantages over NoSQL databases lie in its well-defined data model, scalability, and support for complex queries and joins. The choice between PostgreSQL and MySQL ultimately depends on the specific requirements of the project, with PostgreSQL offering a compelling option for projects demanding a high degree of customization and advanced features.

6.4. Cloud Service: Microsoft Azure

Microsoft Azure, a top-tier cloud computing platform, provides a multitude of services that enable smooth and scalable implementation of web projects. This analysis delves into the benefits of utilizing Microsoft Azure for web project implementation, with a focus on Azure Cache for Redis and Azure Blob Storage for effective caching and file storage, respectively.

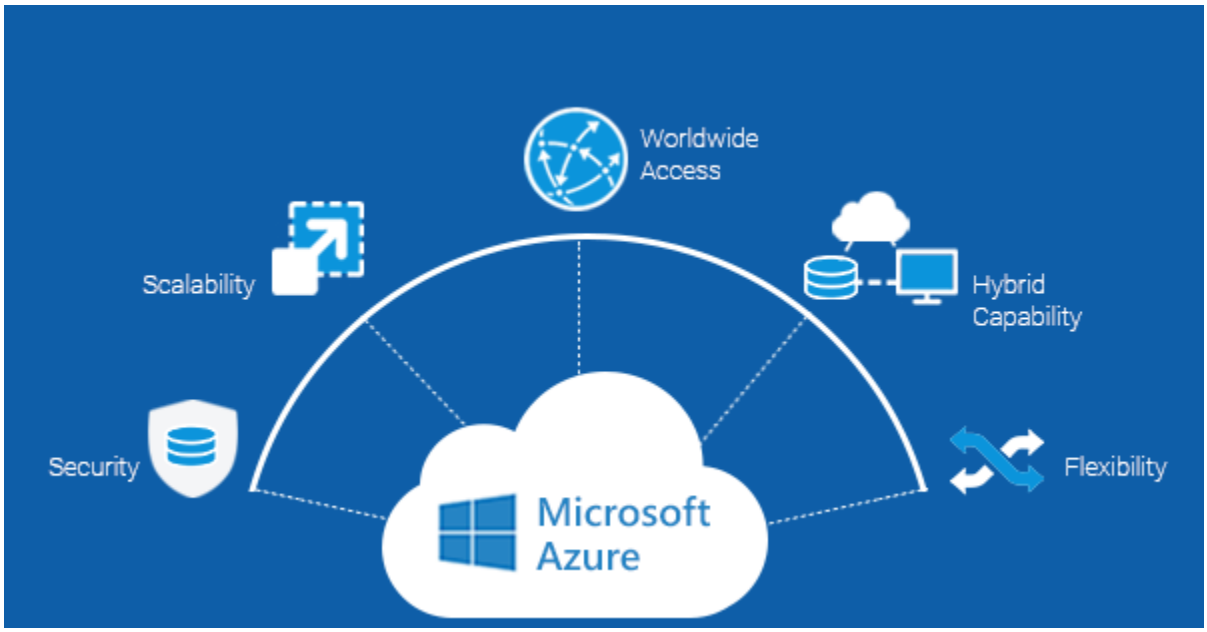


Figure 6.9: Microsoft Azure Benefits

Microsoft Azure for Web Project Implementation:

- Scalability and Adaptability:
    - Azure offers an extremely scalable and adaptable infrastructure, which allows your web project to expand smoothly as user requirements grow. The capacity to scale resources up or down guarantees optimal performance and cost efficiency.
- Worldwide Reach:

- Azure has an extensive global network of data centers, allowing you to launch your project closer to your users. This geographical spread decreases latency and enhances the overall user experience, making it a perfect choice for global projects.

- Integrated Development Tools:
  - Azure integrates effortlessly with popular development tools and supports a broad range of programming languages. This compatibility ensures that your existing development workflows can transition smoothly to Azure, simplifying the implementation process.
- Security and Compliance:
  - Azure places a high priority on security and compliance, offering robust features such as identity management, encryption, and compliance certifications. This is vital for preserving the integrity and confidentiality of sensitive data in your web project.

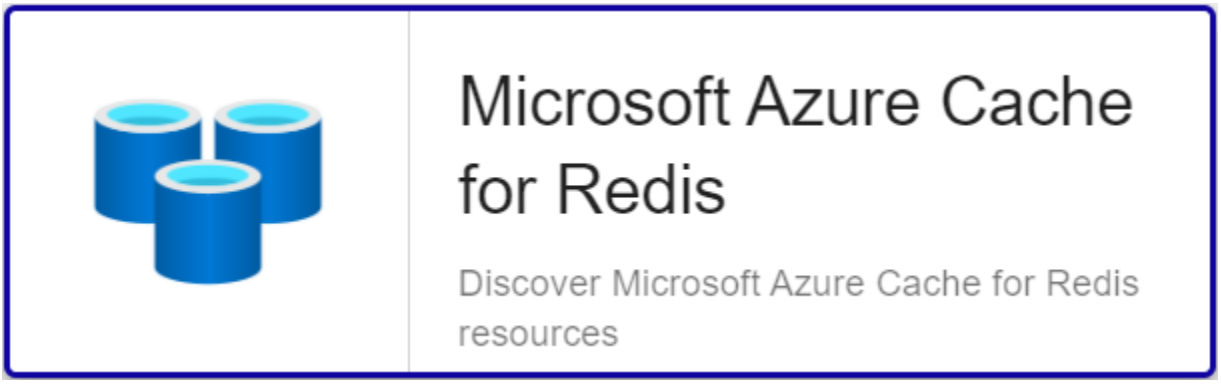


Figure 6.10: Microsoft Azure Redis

Azure Cache for Redis [18]**:**

- High-Speed Caching:
  - Azure Cache for Redis offers an in-memory data store, enabling high-speed caching. This is especially beneficial for web projects where rapid access to frequently used data is crucial, enhancing response times and overall system performance.
- Distributed and Highly Reliable:
  - Redis in Azure is distributed and highly reliable, ensuring that your cached data is resistant to failures. This reliability is essential for maintaining consistent and reliable access to cached information in a web project.
- Support for Various Data Structures:
  - Azure Cache for Redis supports a range of data structures, including strings, hashes, lists, and more. This flexibility allows developers to select the most appropriate data structures for their specific caching requirements.

Figure 6.11: Microsoft Azure Blob Storage Logo

Azure Blob Storage for File Storage [19]**:**

- Scalable and Cost-Effective Storage:
  - Azure Blob Storage offers scalable and cost-effective storage for various types of data, making it a perfect solution for storing files related to your web project. The pay-as-you-go pricing model ensures cost efficiency.
- Redundancy and Durability:
  - Azure Blob Storage provides built-in redundancy and durability features. Your stored files are replicated across multiple data centers, ensuring data integrity and high availability, even in the event of hardware failures or other unexpected issues.
- Integration with Web Applications:
  - Azure Blob Storage integrates seamlessly with web applications, allowing for direct access to stored files. This integration simplifies the implementation of features such as image and document uploads, ensuring a smooth user experience.

Conclusion:

Within the specific scope of our internship connection project, Microsoft Azure emerges as the most fitting technological solution. Our primary goal of streamlining internship processes and enhancing communication between students, faculty, and businesses aligns seamlessly with Azure's global reach, integrated development tools, and security features.

Azure Cache for Redis contributes to the project's efficiency by providing high-performance caching, crucial for automating administrative tasks and facilitating smoother communication channels. Simultaneously, Azure Blob Storage's scalable and cost-effective file storage capabilities meet our specific need for efficient data management associated with internship experiences.

In summary, Microsoft Azure is not only a technological choice but a strategic enabler tailored to our project's objectives. Its scalability, integrated tools, and specialized services offer a precise fit for our mission of creating a more accessible and streamlined internship experience for students, faculty, and businesses involved. Azure stands out as the ideal solution within the specific project scope, addressing our challenges and contributing to the project's success.

7. Implementation

7.1. Front-end implementation

7.1.1. Set up

7.1.1.1. "index.html" file

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml"
href="./src/shared/assets/LinkedOut-Logo.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>LinkedOut</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

- "*<meta name="viewport" content="width=device-width, initial-scale=1.0">*" ensures the page is responsive and scales to different screen sizes.
- "* *" acts as a placeholder for the React application's content. This is where the React app will render its components.

7.1.1.2. "main.tsx" file

```
import React from 'react'
import ReactDOM from 'react-dom/client'
```

```
import App from './App.tsx'

import './index.css'



ReactDOM.createRoot(document.getElementById('root')!).render(

  <React.StrictMode>

    <App />

  </React.StrictMode>

)
```

- "*ReactDOM.createRoot(document.getElementById('root')!)*" creates a React root container within the element with ID "root" in the HTML document.
- "*render(<React.StrictMode><App /></React.StrictMode>)*" renders App component within a React.StrictMode element. React.StrictMode helps in detecting potential issues in React code during development.

7.1.1.3. "index.css" file

```
@tailwind base;

@tailwind components;

@tailwind utilities;
```

Includes directives for Tailwind CSS:

- *@tailwind base*: Includes all of Tailwind's foundational styles, such as resets, box sizing, and flexbox settings. These styles are essential for making layout work properly and are always included.
- *@tailwind components*: Includes all of Tailwind's predefined UI components, such as buttons, forms, and modals.
- *@tailwind utilities*: Includes all of Tailwind's utility classes, which are small, single-purpose CSS classes that can combine to create complex styles.

7.1.1.4. "App.tsx" file

```
import { Suspense} from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Providers from "./Providers";


const queryClient = new QueryClient()
function App() {
  return (
    <QueryClientProvider client={queryClient}>
      <Suspense fallback={<div>Loading...</div>}>
        <Providers>
          <BrowserRouter>
            <Routes>
              {/* Routes */}
            </Routes>
          </BrowserRouter>
        </Providers>
      </Suspense>
    </QueryClientProvider>
  )
}
export default App
```

The App.tsx file serves as the central component in this React application, orchestrating its functionalities and defining its overall structure. It shoulders several key responsibilities:

- Navigation: Utilizing React Router, App.tsx maps specific URLs to corresponding components.

- Data Management: By integrating react-query, App.tsx facilitates asynchronous data fetching, ensuring data is readily available when needed.
- User Authentication: To control access to sensitive information and functionalities, App.tsx employs PrivateRoute, granting access only to authorized users.
- Global State Management: App.tsx wraps the application in a Providers component, likely serving as a centralized point for managing global state and injecting context across the application.
- Error Handling and Loading: To ensure a smooth user experience, App.tsx utilizes Suspense to handle potential asynchronous data fetching errors and display a placeholder ("Loading...") while data is retrieved.

7.1.2. Routes

Our website uses React Router Dom for implementing the routing functionality. This is the summary table of our website routes and pages.

Route	**Page**
/	Home Page
/student	Student Home Page
/student/profile	Student Profile
/student/profile/update	Student Update Profile
/student/jobs	Student Jobs Page
/student/jobs/:jobId	Student Job Display
/student/company	Student Companies Page
/student/company/:companyId	Student Company Display
/staff	Staff Home Page

/company	Company Home Page
/company/jobs	Company Jobs Page
/company/jobs/add	Company Add Job
/company/jobs/:jobId	Company Job Display
/company/applicant	Applicants Page
/company/applicant/:applicantId	Display Applicant
/company/internship	Company Internships Page
/company/internship/:internshipId	Company Display Internship
/company/setting	Company Setting Page
/login/student	Student Login
/login/company	Company Login
/signup/student	Student Sign-up
/signup/company	Company Sign-up
/*	Not Found Page

Table 7.1: Frontend implementation: List of route

7.1.3. Pages implementation

7.1.3.1. Home Page

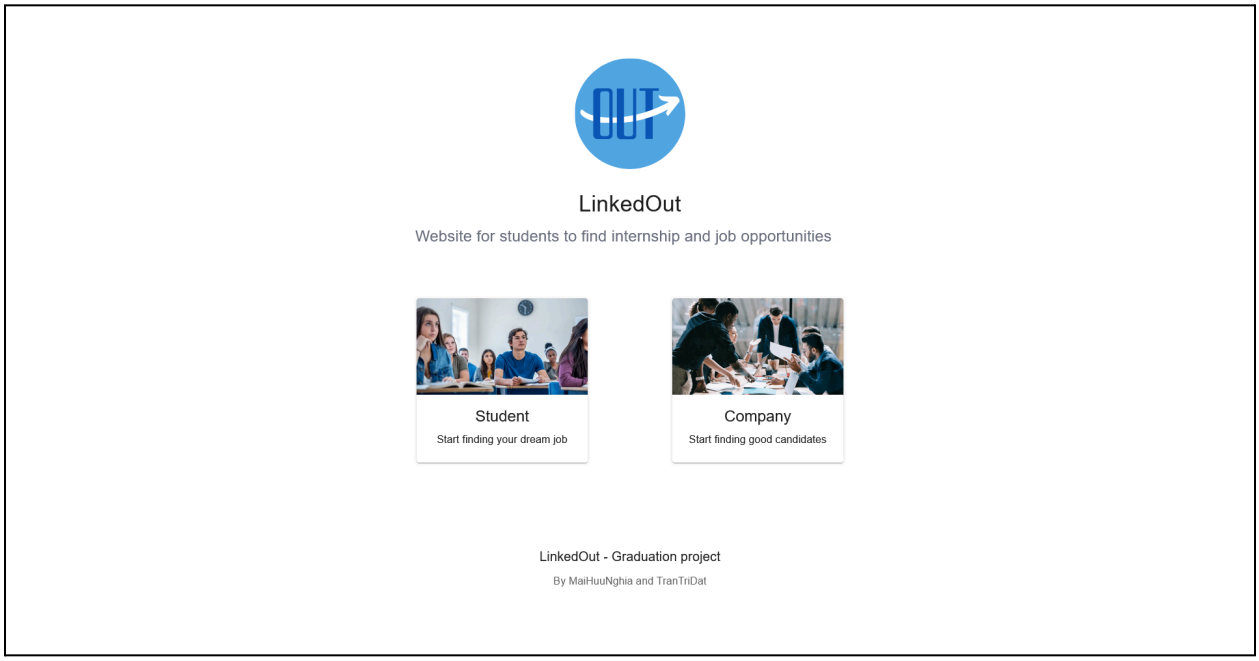


Figure 7.1: Pages implementation - Home Page

This is the Homepage of LinkedOut website, which displays our website logo, name, slogan, and 2 cards for Student and Company for login and signup.

- Button “Login” in “Student” card: Navigate to Student Login page
- Button “Sign up” in “Student” card: Navigate to Student Sign Up page
- Button “Login” in “Company” card: Navigate to Company Login page
- Button “Sign up” in “Company” card: Navigate to Company Sign Up page

7.1.3.2. Student Login Page

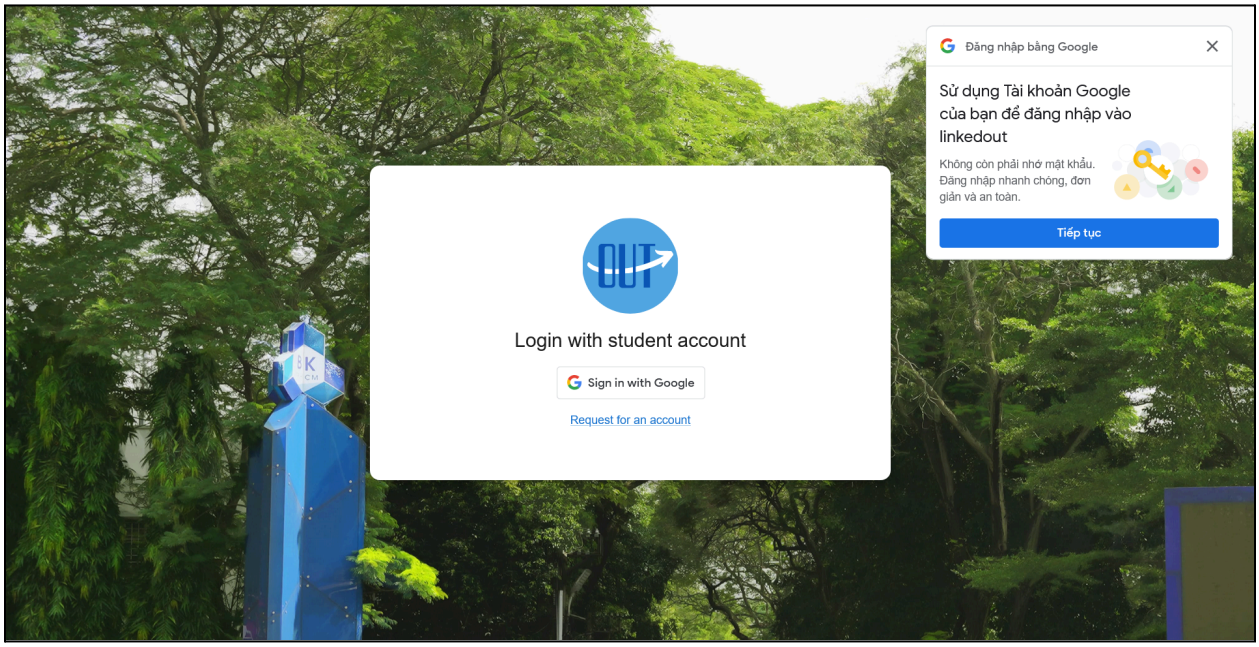


Figure 7.2: Pages implementation - Student Login Page

This page is a place for students to login with their pre-registered Google account.

- When clicking the "Sign in with Google" button, a window will appear for students to login with their email address.
  This action will use the "GoogleLogin" component, from "@react-oauth/google".
  This component will then call api "/api/v1/auth/google-login".
  If login successfully, it will navigate to the Student Home Page with the logged in account.
- When clicking the "Request for an account", it will navigate to the Student Sign Up page.

7.1.3.3. Student Sign Up page

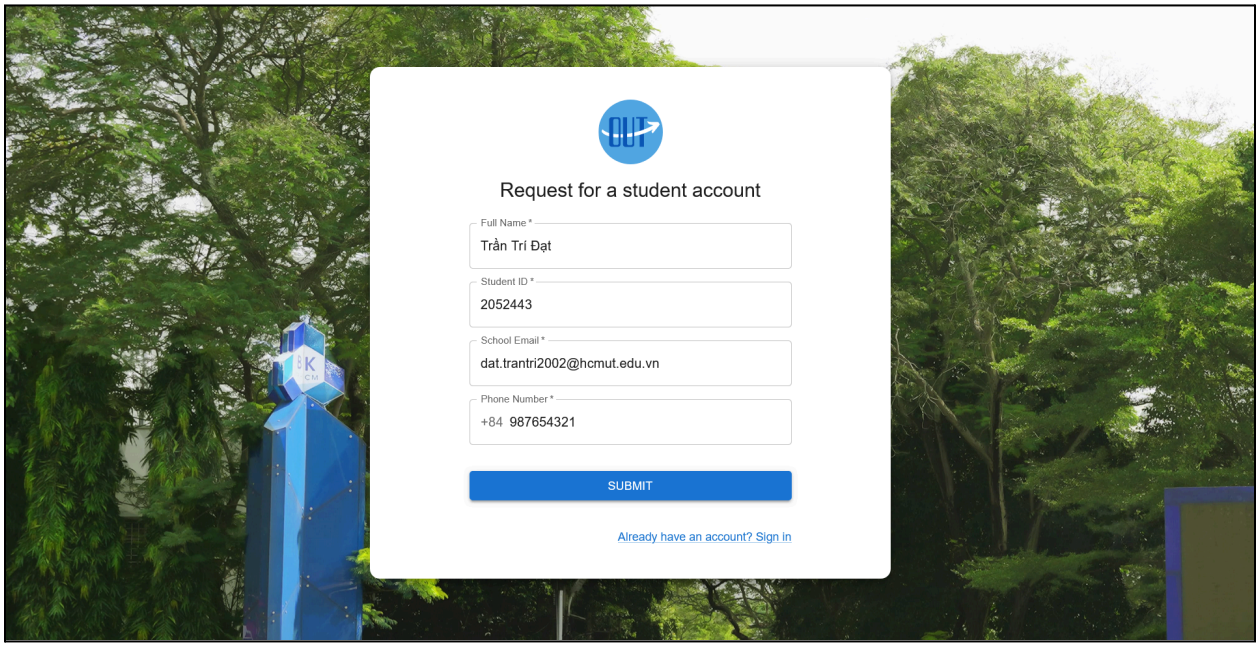


Figure 7.3: Pages implementation - Student Sign Up page

This page is for students to fill in their information to request with the staff for an account to start using the website.

- After filling out all the information, and clicking the “Submit” button, the system will check for validation of information and call the api “/api/v1/student”.
  After the success request, it will navigate back to the Homepage.
- Clicking “Already have an account? Sign in”, it will navigate to the Student Login page.

7.1.3.4. Company Login Page

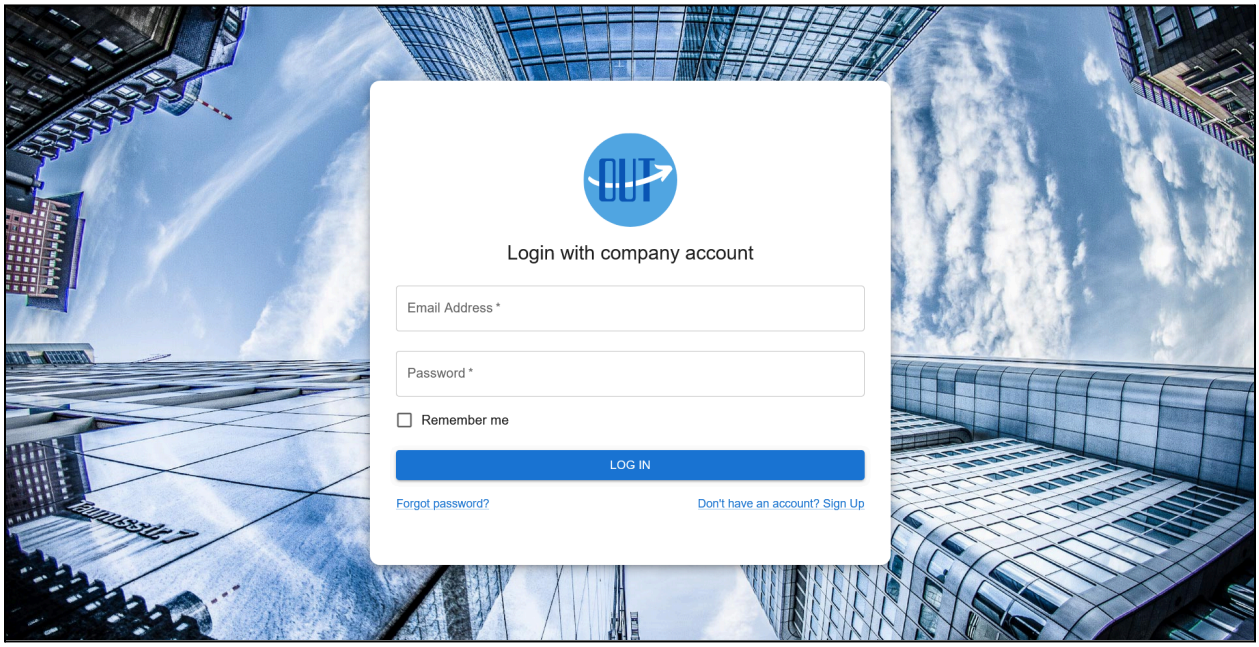


Figure 7.4: Pages implementation - Company Login Page

This page is for company account users to login to the website using their email address and password.

- After filled in those informations and click the “Log In” button. It will call api “/api/v1/company/login” to check.
  If successful, it will navigate to Company Home Page
- If clicking “Don’t have an account? Sign Up”. It will navigate to the Company Sign Up page.

7.1.3.5. Company Sign Up page

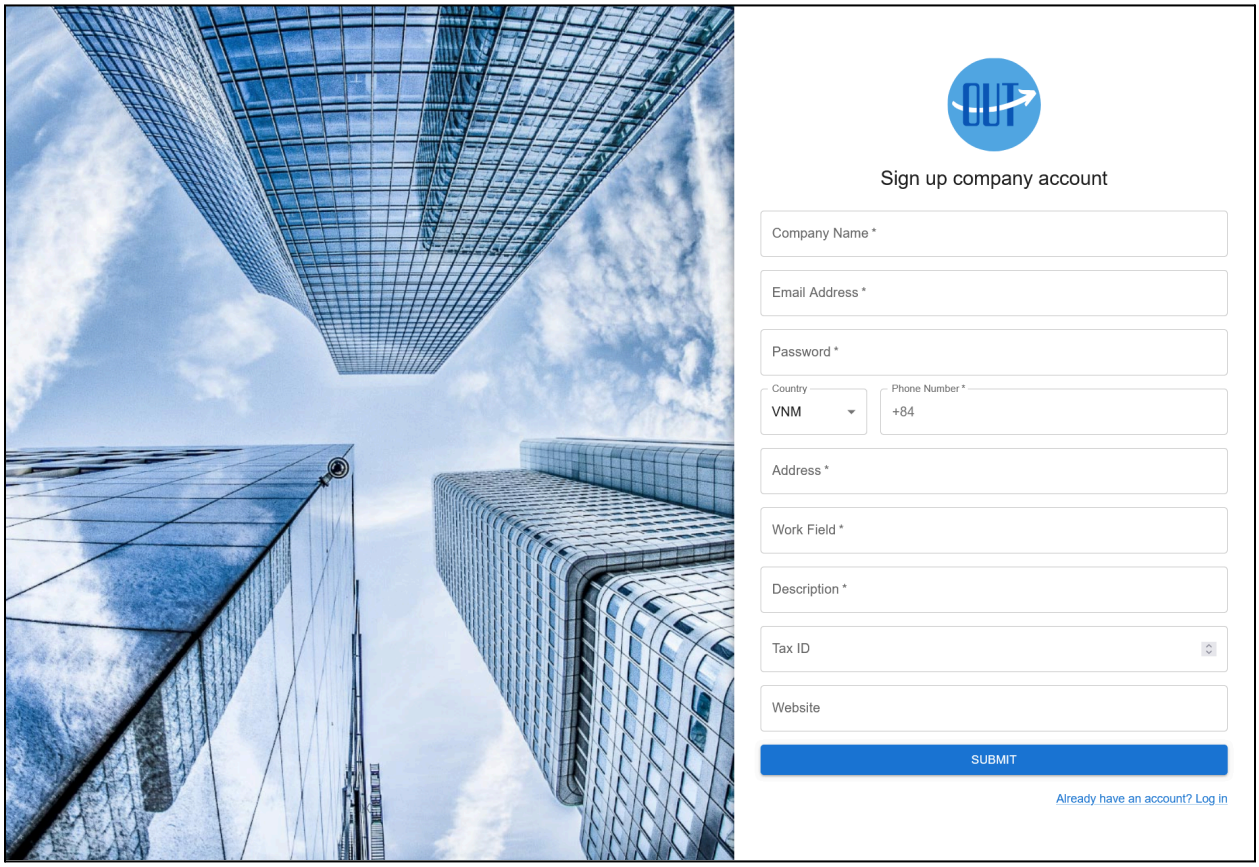


Figure 7.5: Pages implementation - Company Sign Up page

This page allows company account users to fill in information to sign up for an account.

- After filling out all the information and clicking the “Submit” button, it will call api “/api/v1/company” to sign up for an account.
- If clicking the button “Already have an account? Log in”, it will navigate to the Company Login Page.

7.1.3.6. Student Home Page

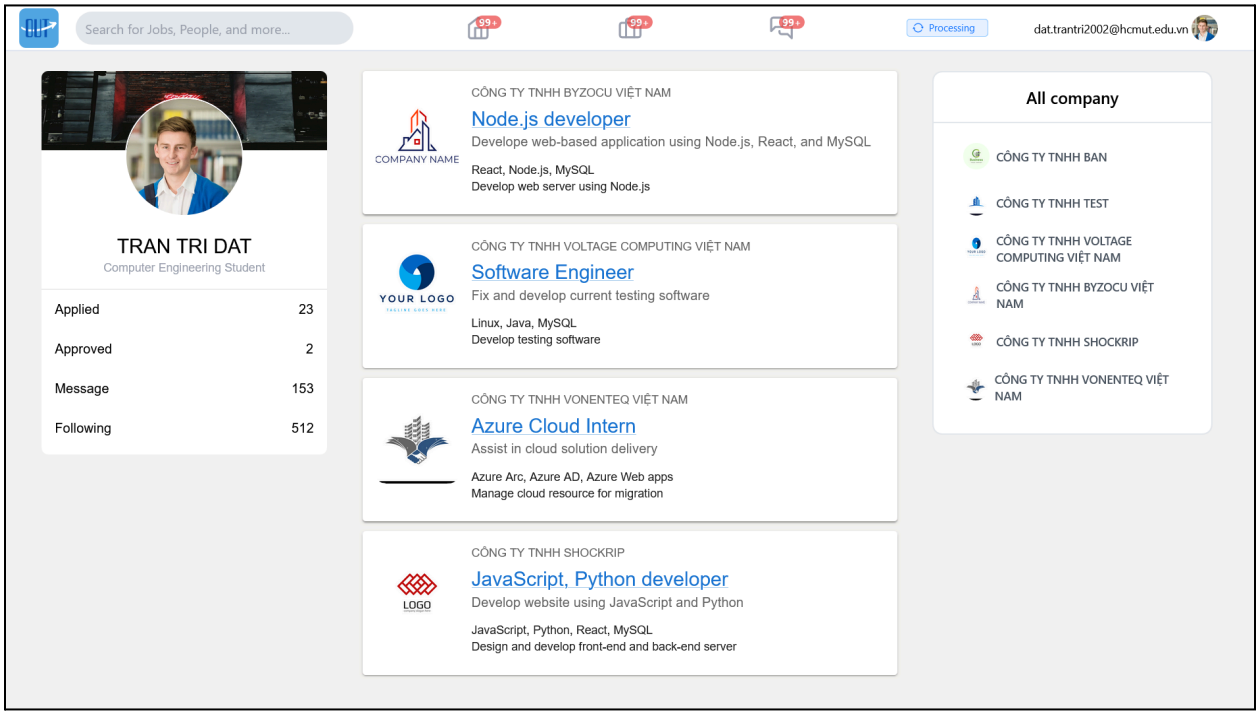


Figure 7.6: Pages implementation - Student Home Page

After the Student has logged in, the student home page will display some information about the students and some actions they can perform.

This page displaying:

- Student information, fetched from back-end with api “api/v1/student/me”
- Jobs that recommended for the student, fetched by api “api/v1/job”
- All connected companies, fetch with api “api/v1/company”

This page allows actions:

- If clicking the email address button (“My profile” button), it will redirect to the Student Profile page.
- If clicking a job title, it will redirect to the Student Job Display page of the selected job.
- If clicking button  , it will navigate to the Student Jobs Page.

- If clicking button 

, it will navigate to the Student Message Page.

7.1.3.7. Student Jobs Page

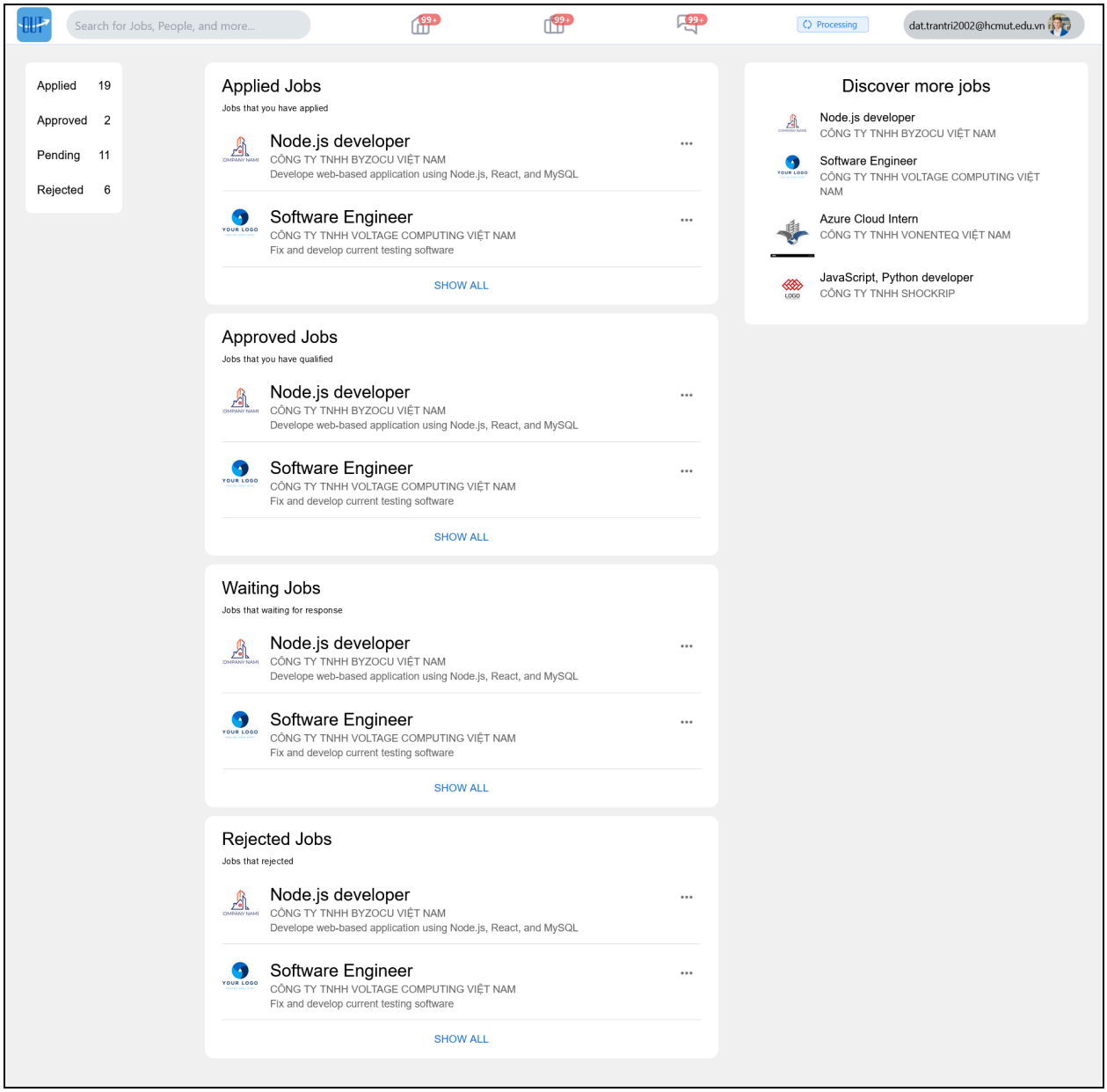


Figure 7.7: Pages implementation - Student Jobs Page

This page displays all information about jobs, including:

- Applied Jobs: Display jobs that the student applied for.

- Approved Jobs: Display jobs that the student applied for and have been approved.
- Waiting Jobs: Display jobs that the student applied for and waiting for processing.
- Rejected Jobs: Display jobs that the student applied for and have been rejected.
- Discover more jobs: Display all others potential jobs that are suitable for the student.

This page allows actions:

- If clicking on a job, it will redirect to the Student Job Display page of the selected job.

7.1.3.8. Student Job Display

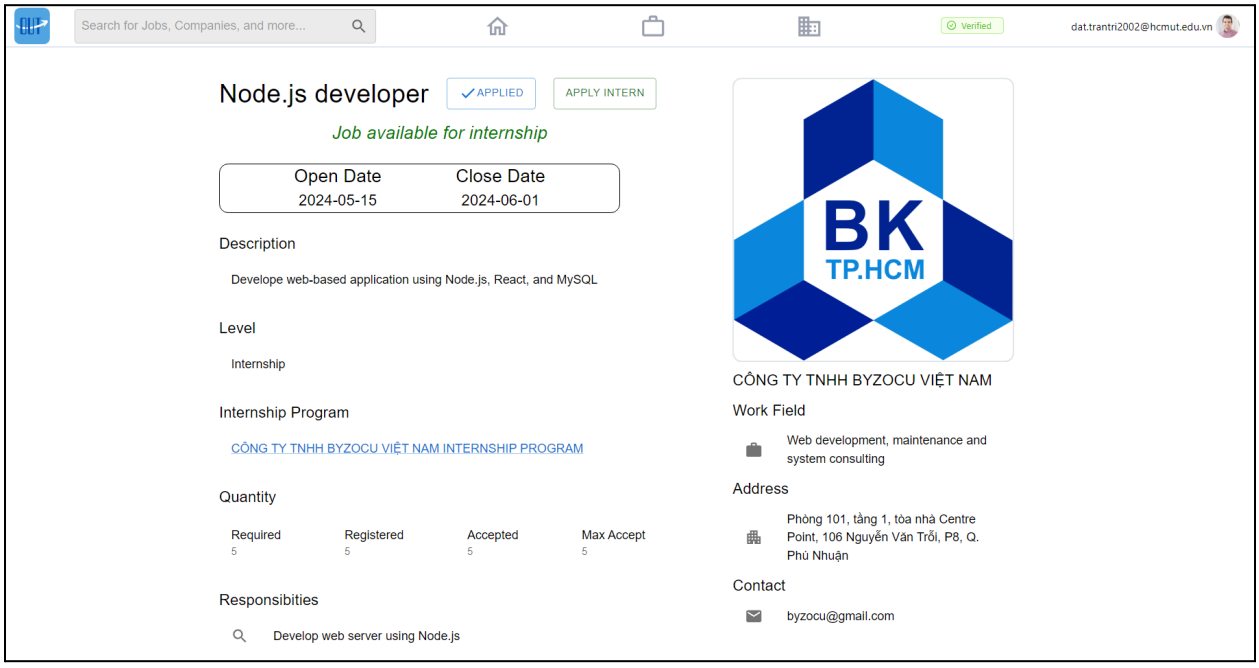


Figure 7.8: Pages implementation - Student Job Display

This page displays all information about the selected job. This information is fetched with api “api/v1/job/<str:uid>”.

Allows actions:

- If clicking “Apply” button, it will apply to the job.
- If clicking “Apply intern” button, it will check the status of the student and apply to the job.

7.1.3.9. Student Companies Page

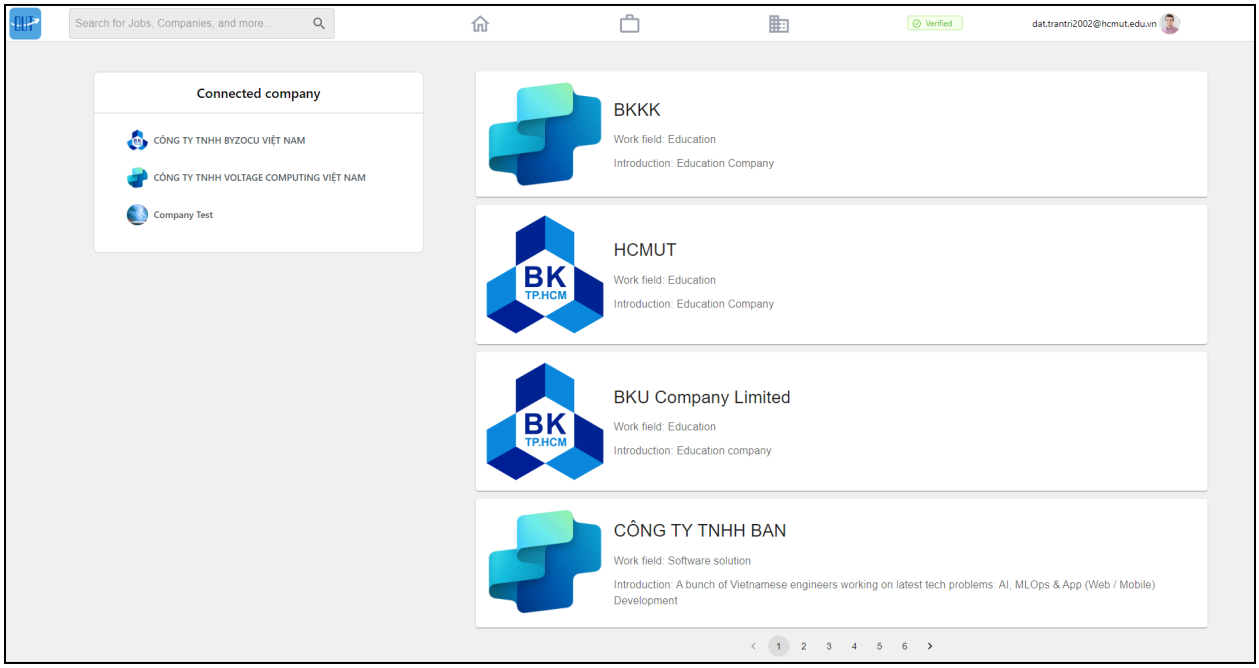


Figure 7.9: Pages implementation - Student Companies Page

This page displays connected companies and lists all companies.

Student then can click on a company to see it profile

7.1.3.10. Student Profile Page

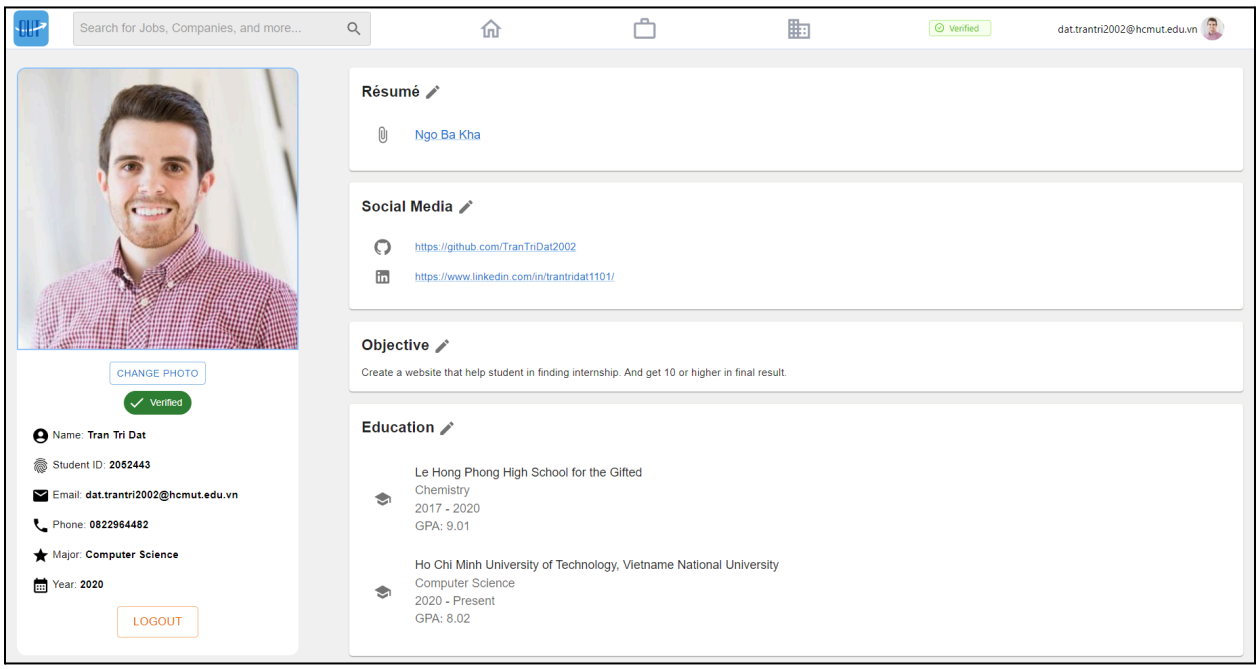


Figure 7.10: Pages implementation - Student Profile Page

This page shows the student information with some internship information. This information is fetched from the api “api/v1/student/me”

Actions:

- If clicking “Change photo” button, it will display a dialog box to update the avatar. This update is done with api “api/v1/student/<str:uid>”.

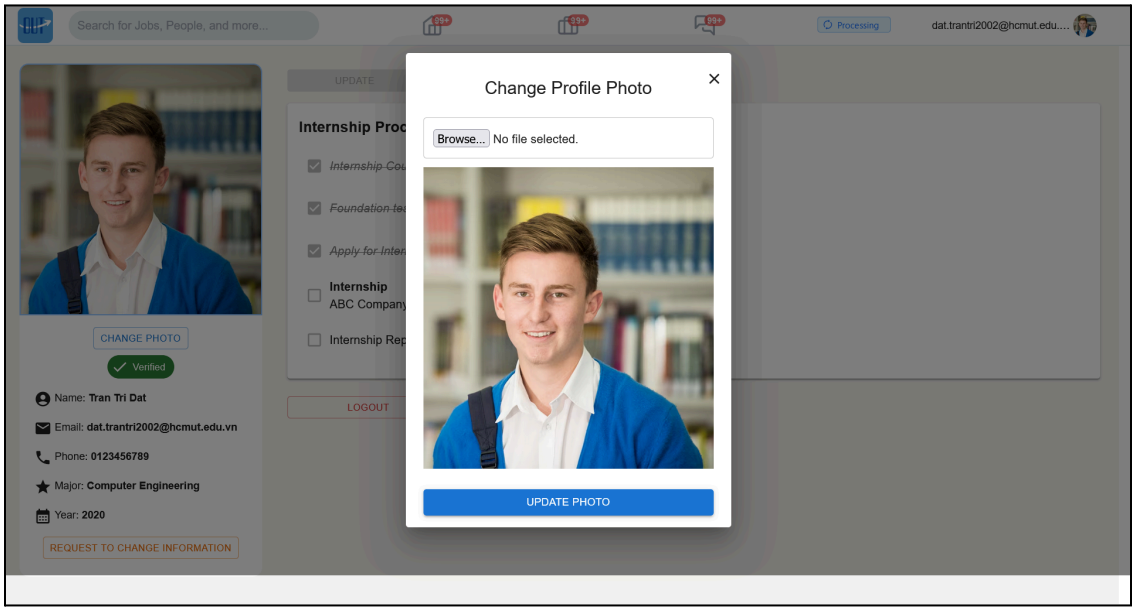


Figure 7.11: Pages implementation - Student Profile Page - Update avatar

- If clicking " " button, it will display a dialog box to update the field.

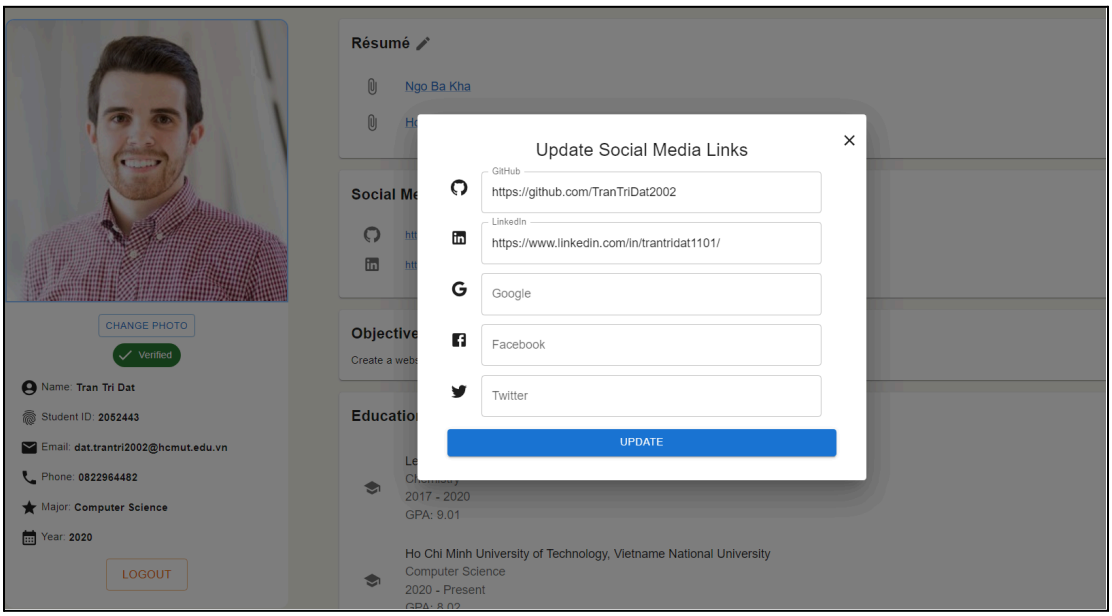


Figure 7.12: Pages implementation - Student Profile Page - Update information

- If clicking "Logout" button, it will log out of the account. This is done with api "api/v1/student/logout"

7.1.3.11. Company Home Page

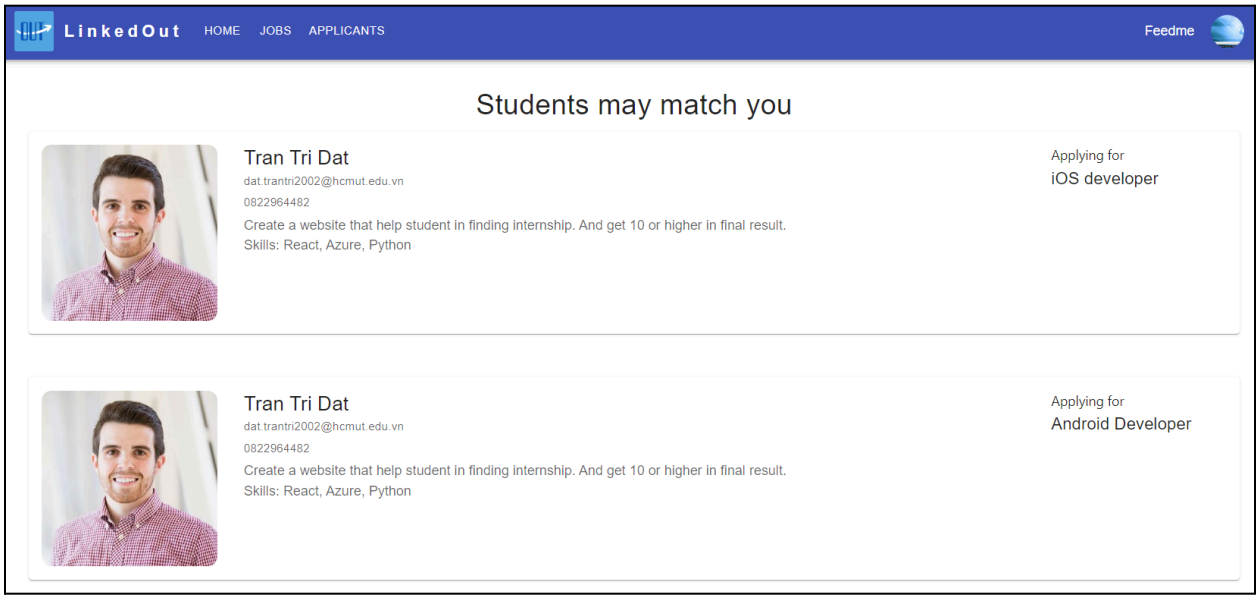


Figure 7.13: Pages implementation - Company Home Page

This page display:

- Company name with the logo. This information is fetched from api "api/v1/company/me"
- Some page navigation buttons on navigation bar
- A list of students that are most suitable for the company.

Allows actions:

- If clicking "Jobs" button, it will navigate to the Company Jobs Page.
- If clicking "Applicants" button, it will navigate to the Company Applicants Page.
- If clicking "Messages" button, it will navigate to the Company Message Page.
- If clicking company logo, it will show a list, include 2 actions "Settings" and "Logout"
    - If clicking "Settings", it will show a dialog to update company information.

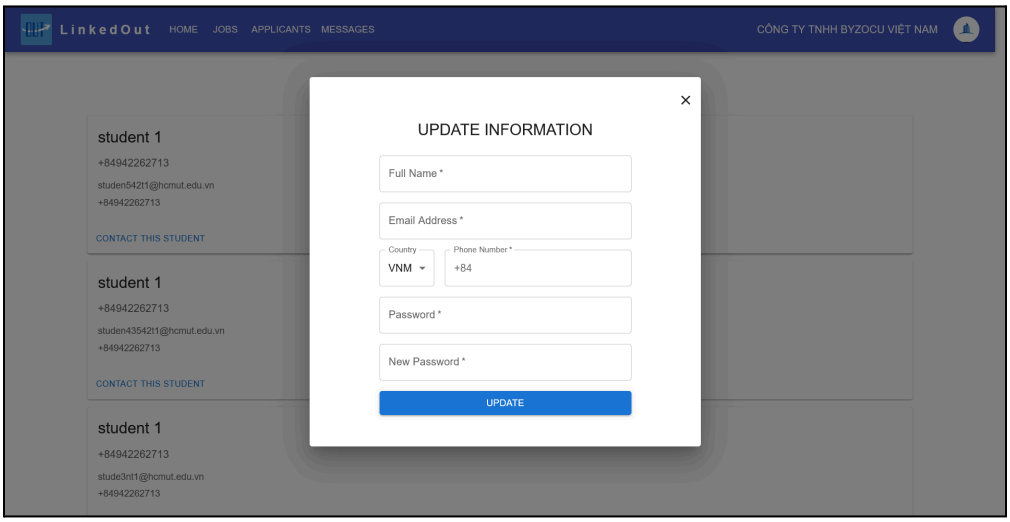


Figure 7.14: Pages implementation - Update company information

Update information can be done with api “api/v1/company/<str:uid>”

- If clicking “Logout”, it will log out of the account, using api “api/v1/company/logout”

- If clicking on a job, it will navigate to the Company Job Display page, with the selected job.

7.1.3.12. Company Jobs Page

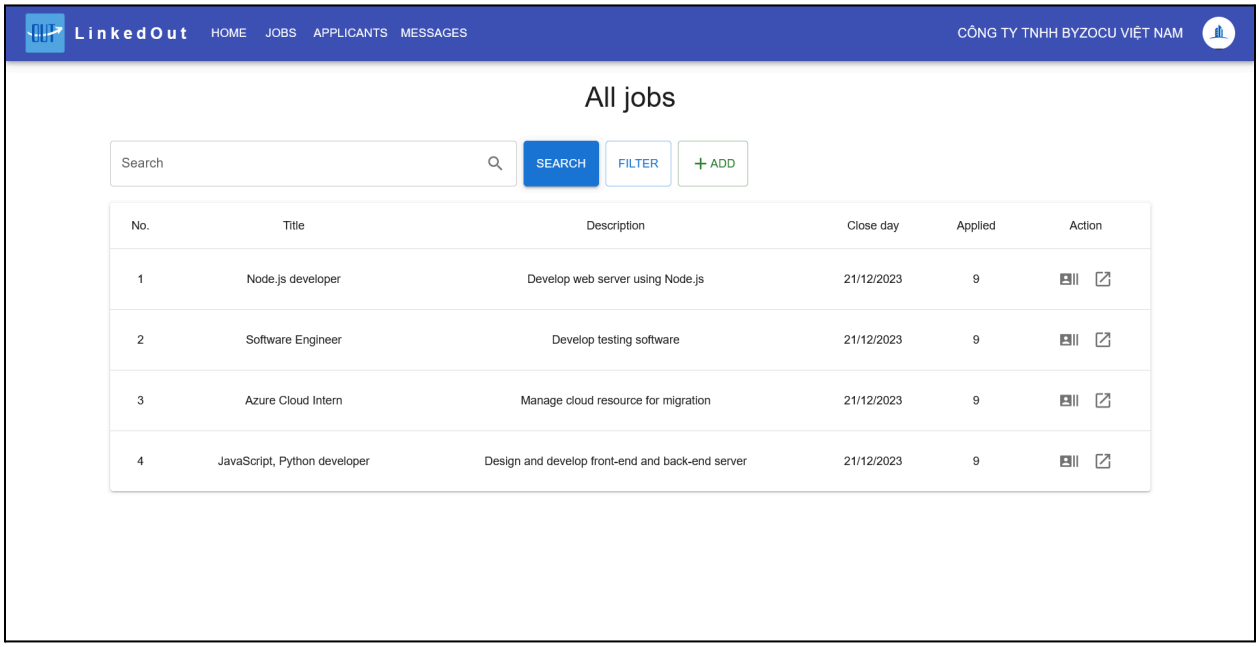


Figure 7.15: Pages implementation - Company Jobs Page

This page displays all jobs that the company posted.

Allow actions:

- Enter a job name and click the “Search” button. It will perform search and display matching jobs using api “api/v1/job/ search”
- Click the “Filter” button, it will display some conditions to filter out jobs.
- Click the “Add” button, it will navigate to the Company Add Job page.
- Click button , it will navigate to the Company Applicants page with applicants that applied to the selected job.
- Click button , it will navigate to the Company Job Display page of the selected job.

7.1.3.13. Company Add Job Page

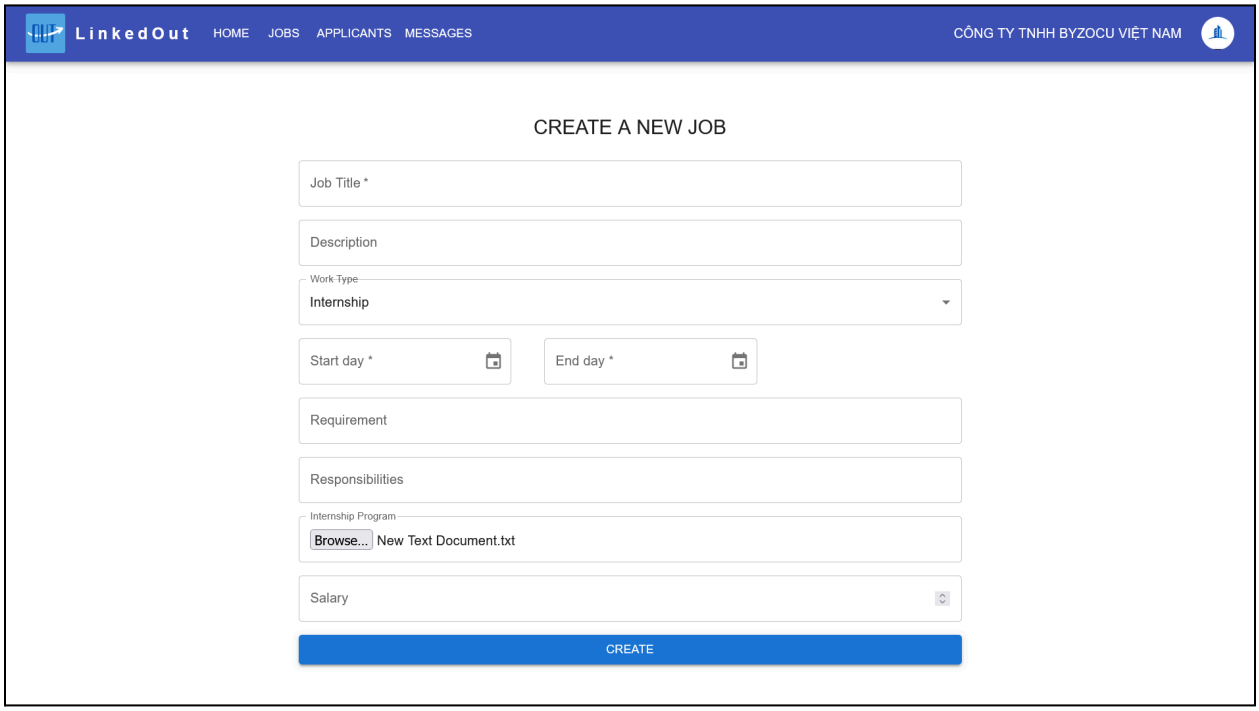


Figure 7.16: Pages implementation - Company Add Job Page

This page displays a form for companies to create new jobs. This action is done by api "api/v1/job/"

7.1.3.14. Company Job Display

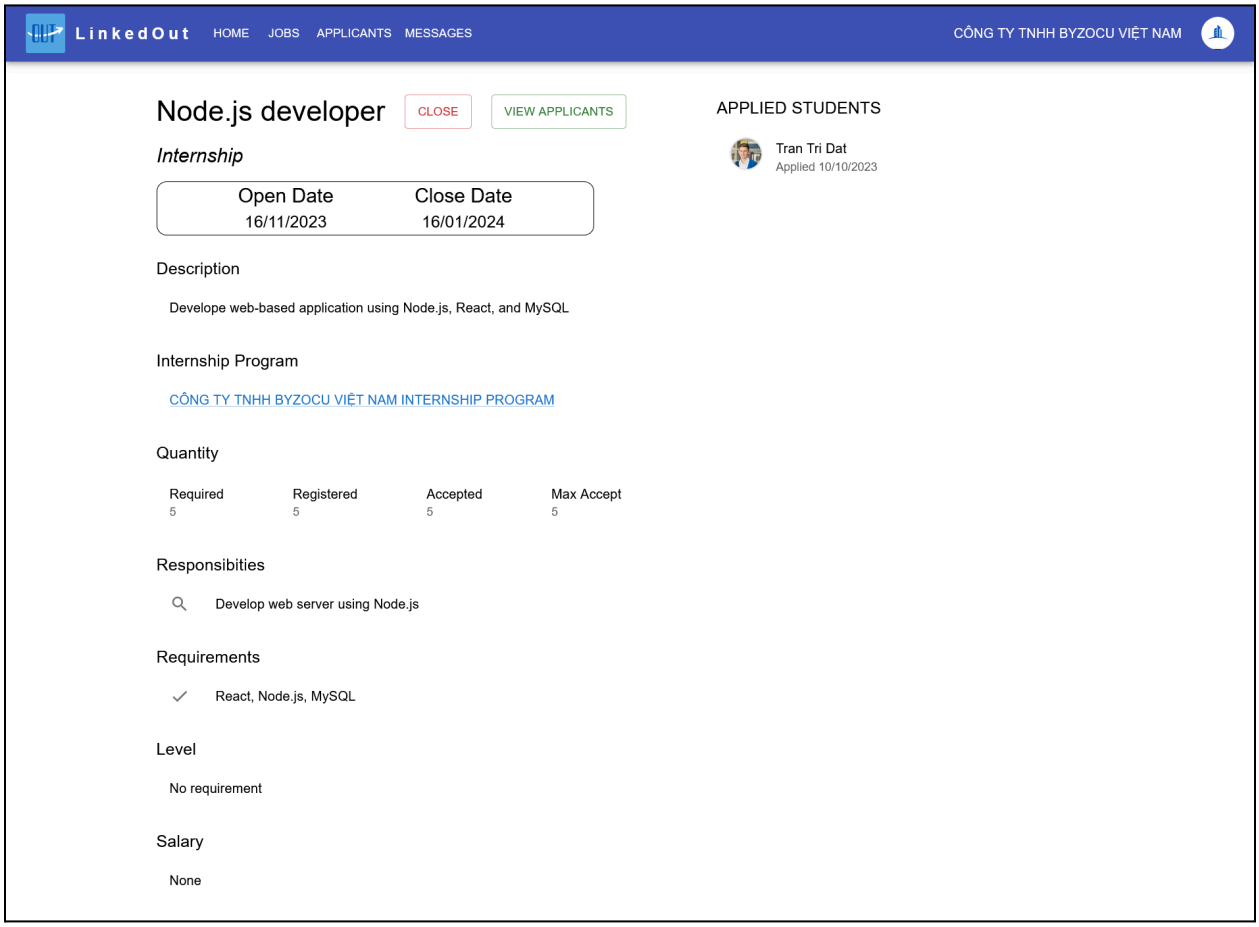


Figure 7.17: Pages implementation - Company Job Display

This page displays information about selected jobs. This information is fetched from api "api/v1/job/<str:uid>". The page also displays a list of applicants that applied for the job.

Allows actions:

- Click the "Close" button, it will close the job, applicants cannot apply to this job. This is done with api "api/v1/job/<str:uid>"
- Click the "View applicants" button, it will navigate to the Company Applicants Page, and show a list of applicants that applied to the job.
- Click on an applicant in the applicant list, it will navigate to the Company Display Applicant page of the selected applicant.

7.1.3.15. Company Applicants Page

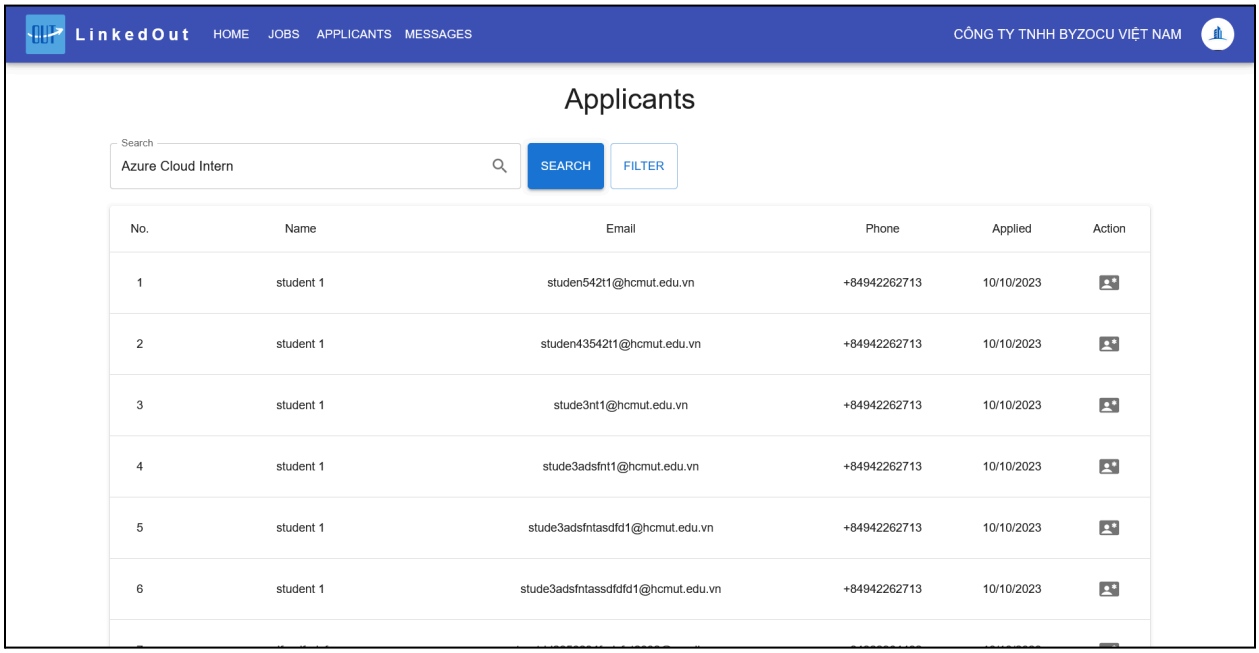


Figure 7.18: Pages implementation - Company Applicants Page

This page displays all applicants that applied for jobs at the company.

Allow actions:

- Enter a name and click the “Search” button. It will perform search and display matching applicants.
- Click the “Filter” button, it will display some conditions to filter out applicants.
- Click button  , it will navigate to the Company Display Applicant page of the selected applicant.

7.1.3.16. Company Display Applicant

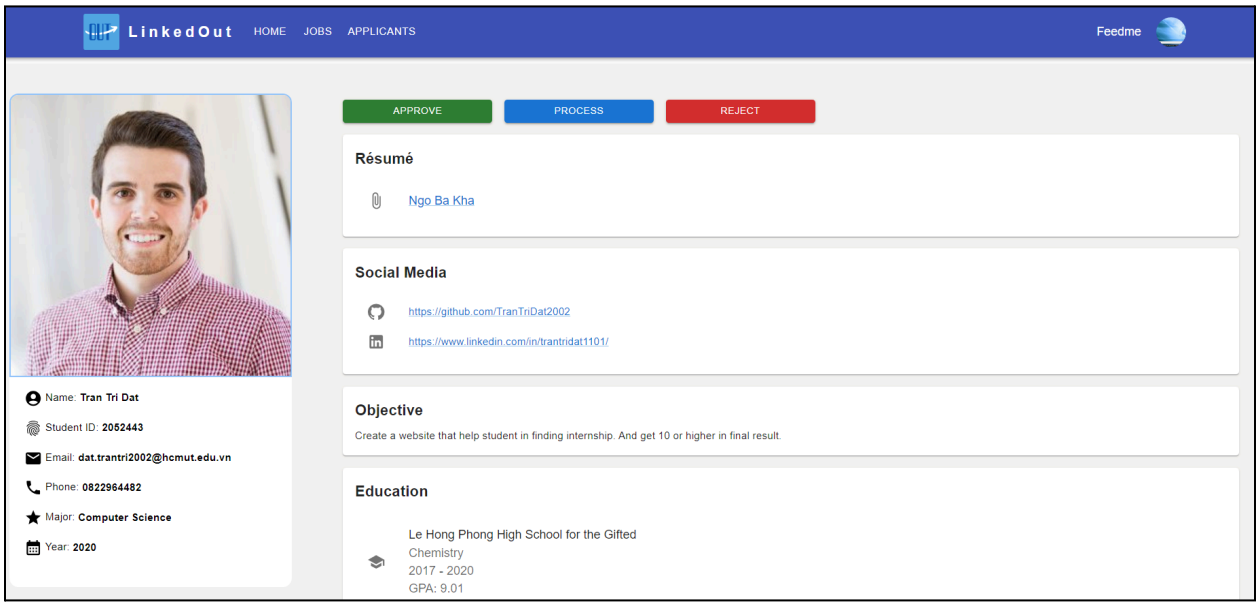


Figure 7.19: Pages implementation - Company Display Applicant

This page displays all information of the selected applicant. Information is fetched from api “api/v1/student/<str:uid>”.

Actions:

- Click the “Accept” button, the student will be accepted with the applied job.
- Click the “Process” button, the student will be processed with the applied job.
- Click the “Reject” button, the student will be rejected with the applied job.

7.1.3.17. Staff Home Page

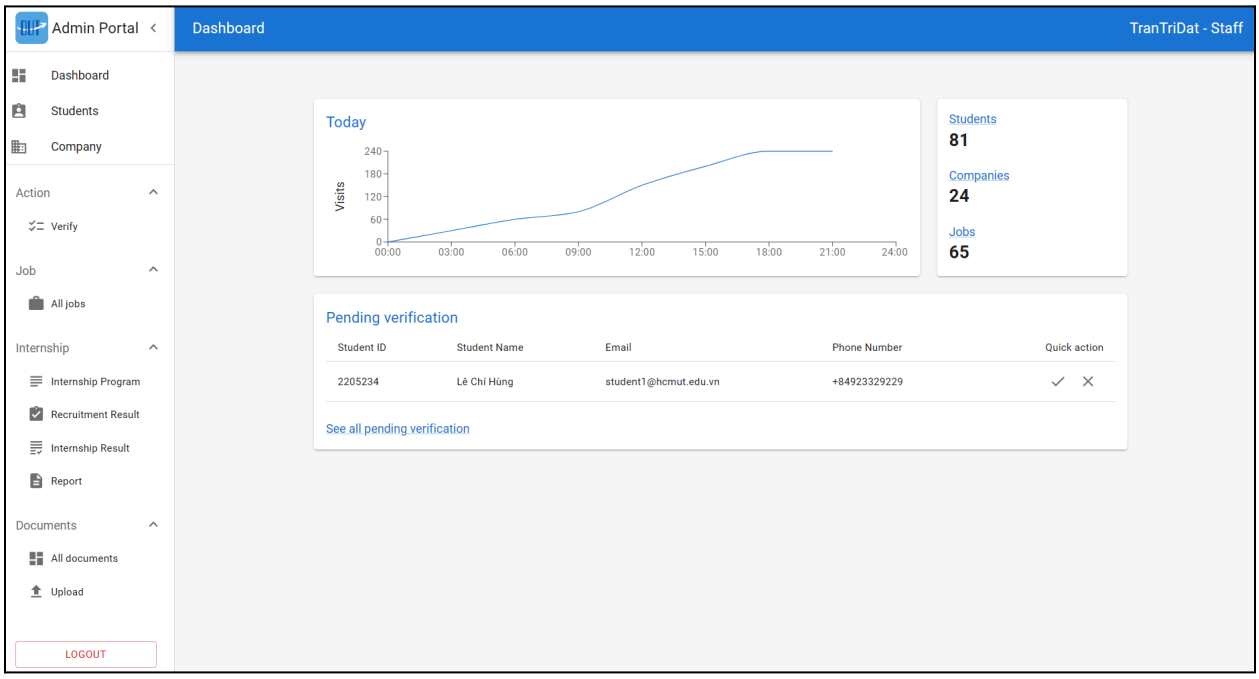


Figure 7.20: Pages implementation - Staff Home Page

This page displays:

- Overview of the number of students, and companies using the website. This information is get from api “api/v1/student/” and “api/v1/company/”
- Number of applications are created on the website.
- A quick view of pending verifications.

Actions:

- Click “accept” or “reject” action in the Pending verification section, it will update the status of companies or students or jobs using apis “api/v1/company/<str:uid>”, “api/v1/student/<str:uid>”, and “api/v1/job/<str:uid>”.
- Click the text “Students”, it will display the Staff - Student screen.
- Click the text “Companies”, it will display the Staff - Company screen.
- Click the text “See all pending verification”, it will display Staff - Verify screen.
- Click “Students” tab, it will display the Staff - Student screen

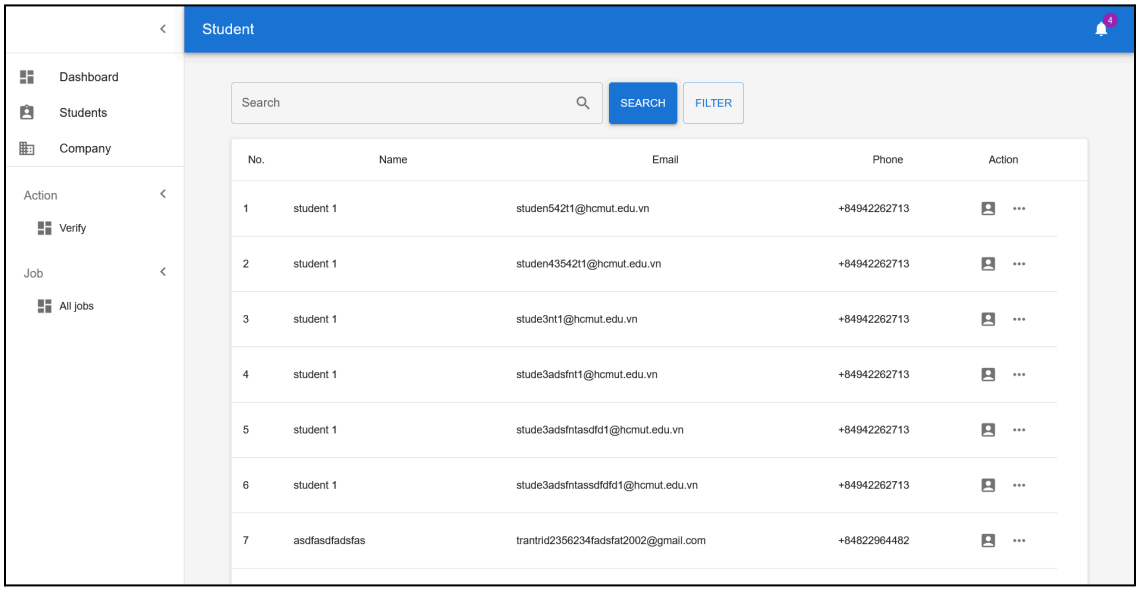


Figure 7.21: Pages implementation - Staff - Student screen

This screen displays a list of all students using the website.

Actions:

- Enter a name and click the “Search” button. It will perform search and display matching students.
- Click the “Filter” button, it will display some conditions to filter out students.
- Click on button , it will display detailed information of the selected student.
- Click on button , it will display some further actions for the selected student.

- Click “Company” tab, it will display the Staff - Company screen

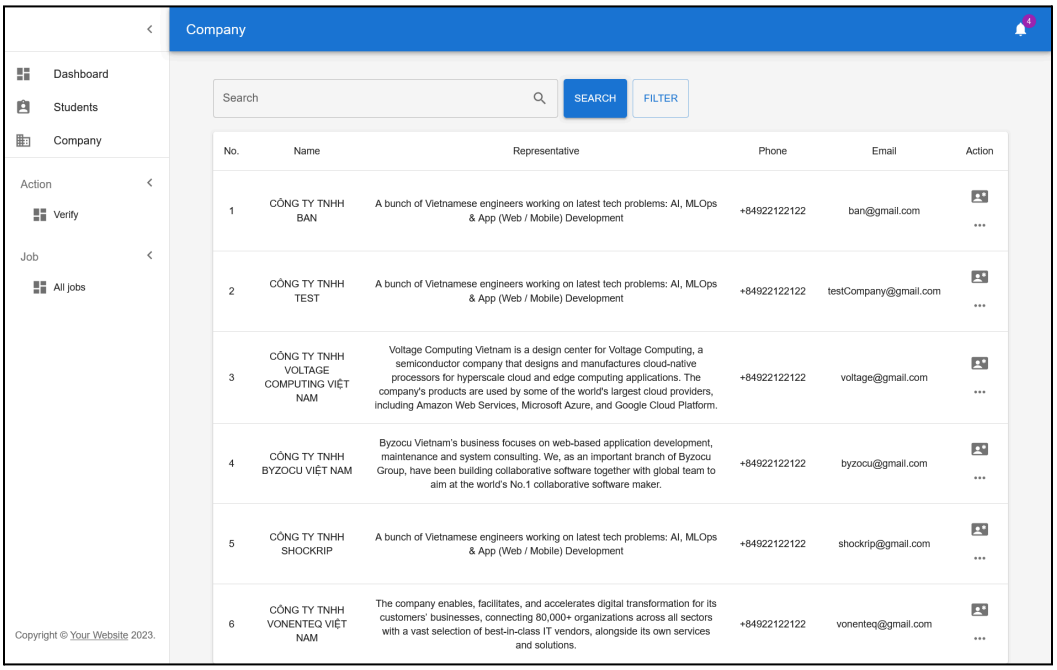


Figure 7.23: Pages implementation - Staff - Company screen

This screen displays a list of all companies using the website.

Actions:

- Enter a name and click the “Search” button. It will perform search and display matching companies.
- Click the “Filter” button, it will display some conditions to filter out companies.
- Click on button , it will display detailed information of the selected company.
- Click on button , it will display some further actions for the selected company.

- Click “Verify” tab, it will display the Staff - Verify screen

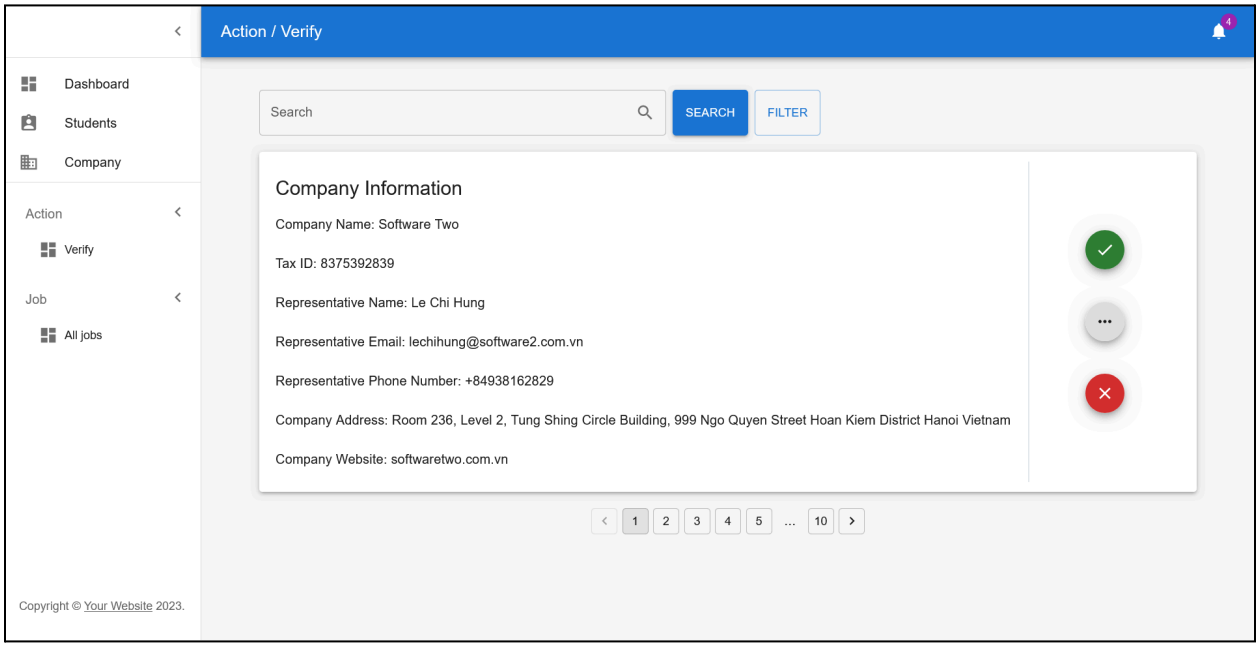


Figure 7.24: Pages implementation - Staff - Verify screen

This screen displays detailed requests of companies, students, and jobs.

Actions:

- Enter a name and click the “Search” button. It will perform search and display matching companies or students or jobs.
- Click the “Filter” button, it will display some conditions to filter out companies or students or jobs.
- Click on button , it will accept the request, update information of companies or students or jobs.
- Click on button , it will reject the request, update information of companies or students or jobs.
- Click on button , it will display some further actions.

- Click “All jobs” tab, it will display the Staff - All Jobs screen

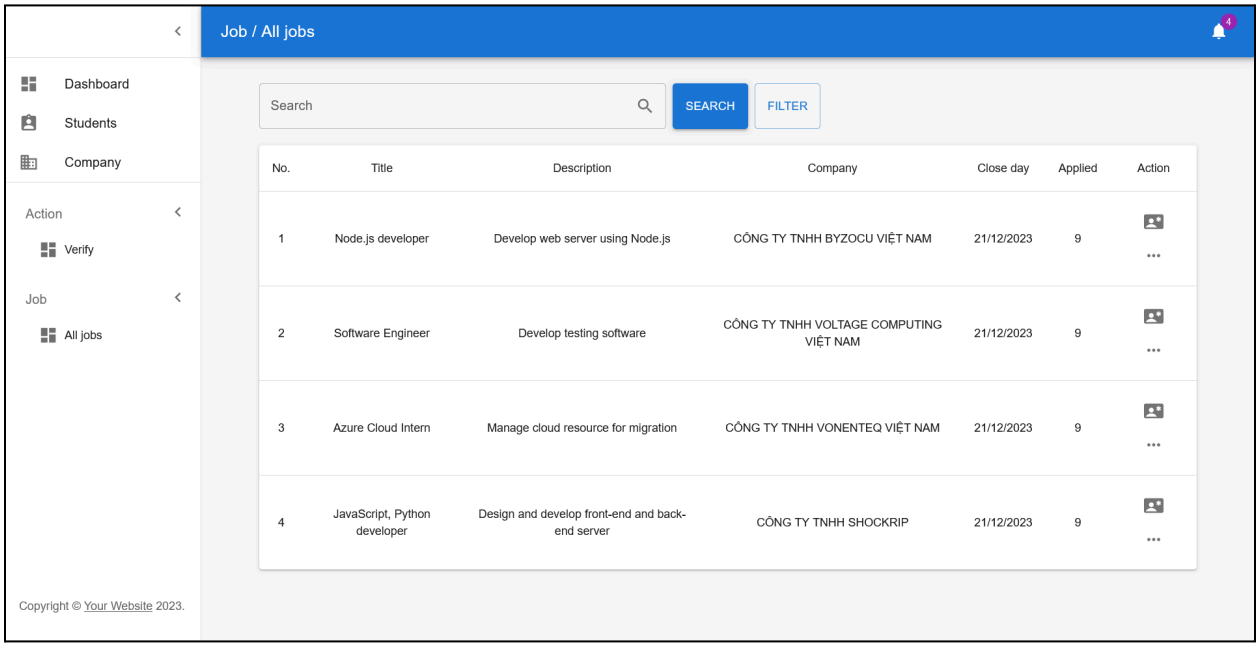


Figure 7.25: Pages implementation - Staff - All Jobs screen

This screen displays a list of all jobs of companies.

Actions:

- Enter a name and click the “Search” button. It will perform search and display matching jobs.
- Click the “Filter” button, it will display some conditions to filter out jobs.
- Click on button , it will display detailed information of the selected job.
- Click on button , it will display some further actions for the selected job.

7.2. Database implementation

Figure 7.25: Database implementation schemas

7.3. Back-end implementation

7.3.1. Authentication and Authorization

Our application utilizes JSON Web Tokens (JWT) [20] for authentication, providing a secure and efficient way to manage user authentication and access control. JWT is a widely adopted industry standard that enables the secure transmission of user identity information between the client and server. By implementing JWT, we ensure that user authentication is handled in a stateless and scalable manner.

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

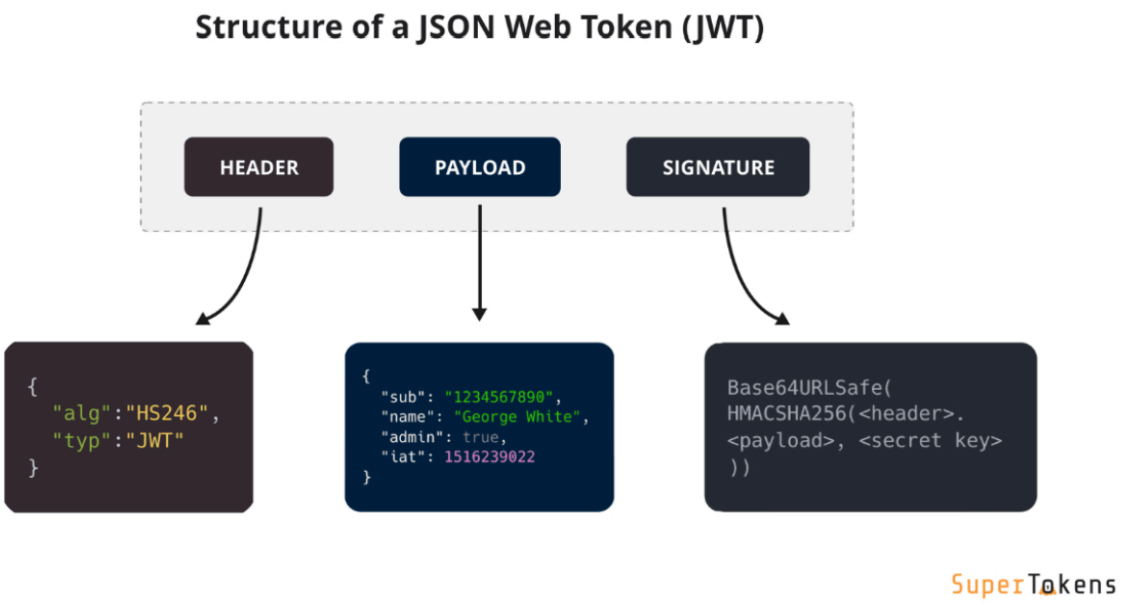


Figure 7.27: Structure of JSON Web Token

Benefits of JWTs:

- More compact: JSON is less verbose than XML, so when it is encoded, a JWT is smaller than a SAML token. This makes JWT a good choice to be passed in HTML and HTTP environments.
- More secure: JWTs can use a public/private key pair in the form of an X5.09 certificate for signing. A JWT can also be symmetrically signed by a shared secret using the HMAC algorithm.
- More common: JSON parsers are common in most programming languages because they map directly to objects.

Our JWT token payload:

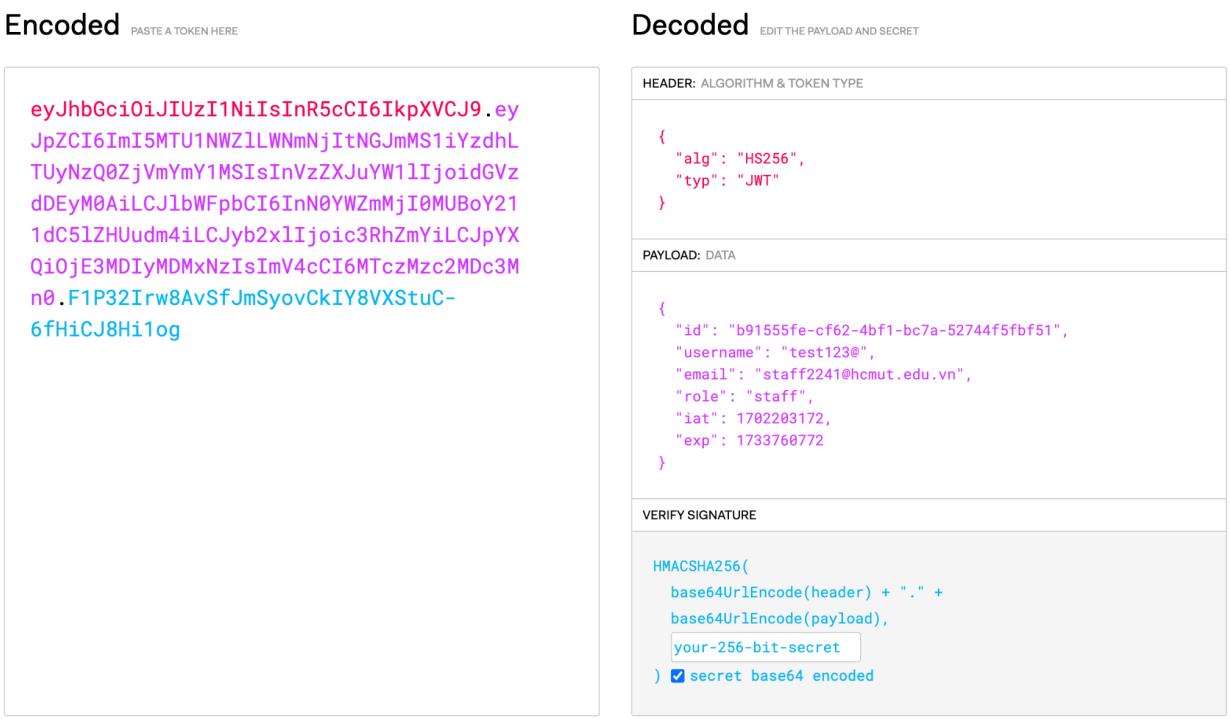


Figure 7.29: JWT token payload

7.3.2. API summary

Staff routes:

Method	Route	Parameter	Description
GET	api/v1/staff/me	Header: - Bearer Token	Get current login staff
GET	api/v1/staff/	Header: - Bearer Token	Get all staff account
GET	api/v1/staff/<str:uid>	Header: - Bearer Token Request parameter: - uid: user's id	Get staff account by user's id
POST	api/v1/staff/	Header: - Bearer Token Request parameter: - name: user's name - email: user's email - phoneNumber: user's phone number - myfile: user's avatar	Create a new staff's account
POST	api/v1/staff/login	Header: - Bearer Token Request parameter: - email: user's email	Login to the staff's account
POST	api/v1/staff/logout	Header: - Bearer Token Request parameter: - uid: user's id	Logout of the staff account
PUT	api/v1/staff/<str:uid>	Header: - Bearer Token Request parameter: - name: user's name - email: user's email - phoneNumber: user's phone	Edit staff information

		number - myfile: user's avatar	
DELETE	api/v1/staff/<str:uid>	Header: - Bearer Token Request parameter: - uid: user's id	Delete a staff account

Table 7.2: Backend implementation: API of staff routes

Student routes:

Method	**Route**	**Parameter**	**Description**
GET	api/v1/student/me	Header: - Bearer Token	Get current login student
GET	api/v1/student/	Header: - Bearer Token	Get all student account
GET	api/v1/student/<str:uid>	Header: - Bearer Token Request parameter: - uid: user's id	Get student account by user's id
POST	api/v1/student/	Header: - Bearer Token Request parameter: - name: user's name - email: user's email - phoneNumber: user's phone number - myfile: user's avatar	Create a new student's account
POST	api/v1/student/login	Header: - Bearer Token Request parameter: - email: user's email	Login to the student's account
POST	api/v1/student/logout	Header: - Bearer Token Request parameter:	Logout of the student account

		- uid: user's id	
PUT	api/v1/student/<str:uid>	Header: - Bearer Token Request parameter: - name: user's name - email: user's email - phoneNumber: user's phone number - myfile: user's avatar	Edit student information
DELETE	api/v1/student/<str:uid>	Header: - Bearer Token Request parameter: - uid: user's id	Delete a student account

Table 7.3: Backend implementation: API of student routes

Company routes:

Method	**Route**	**Parameter**	**Description**
GET	api/v1/company/me	Header: - Bearer Token	Get current login company
GET	api/v1/company/	Header: - Bearer Token	Get all company account
GET	api/v1/company/<str:uid>	Header: - Bearer Token Request parameter: - uid: user's id	Get company account by user's id
POST	api/v1/company/	Header: - Bearer Token Request parameter: - name: user's name - password: user's password - email: user's email - phoneNumber: user's phone number	Create a new company's account

		- myfile: user's avatar	
POST	api/v1/company/login	Header: - Bearer Token Request parameter: - password: user's password - email: user's email	Login to the company's account
POST	api/v1/company/logout	Header: - Bearer Token Request parameter: - uid: user's id	Logout of the company account
PUT	api/v1/company/<str: uid>	Header: - Bearer Token Request parameter: - name: user's name - password: user's password - email: user's email - phoneNumber: user's phone number - myfile: user's avatar	Edit company information
DELETE	api/v1/company/<str: uid>	Header: - Bearer Token Request parameter: - uid: user's id	Delete a company account

Table 7.4: Backend implementation: API of company routes

Job routes:

Method	**Route**	**Parameter**	**Description**
GET	api/v1/job	Header: - Bearer Token	Get all job in the system
GET	api/v1/job/company	Header: - Bearer Token of company	Get all job by company's id
GET	api/v1/job/<str: id>	Header: - Bearer Token	Get a job by job's id

		Request parameter: - jobId: job's id	

POST	api/v1/job/	Header:
		- Bearer Token
		Request parameter:
		- title: user's name
		- workType: user's password
		- quantity: user's email
		- descriptions: description of a job
		  - aboutUs: information about the company
		  - responsibilities: what you will need to be response when taking the job
		  - requirements: some entry criterias that you have to meet to apply for this job
			Create a new job

POST	api/v1/job/search	Header: - Bearer Token Request parameter: - title: title of the job - level: level of the job - workType: working type of the job - companyName: company's name of the job	Search job with criterias

PUT	api/v1/job/<str:uid>	Header:
		- Bearer Token
		Request parameter:
		- title: user's name
		- workType: user's password
		- quantity: user's email
		- descriptions: description of a job
		  - aboutUs: information about the company
		  - responsibilities: what you will need to be response when taking the job
		  - requirements: some entry
			Edit job information

		criterias that you have to meet to apply for this job	
DELETE	api/v1/job/<str:uid>	Header: - Bearer Token Request parameter: - uid: job's id	Delete a job

Table 7.5: Backend implementation: API of job routes

Job Applicants routes:

Method	**Route**	**Parameter**	**Description**
GET	api/v1/job_applicants/applicant/<str:id>	Header: - Bearer Token Request parameter: - jobApplicantsId: job applicant’s id	Get a job applicants by job applicant's id
GET	api/v1/job_applicants/candidate/<str:id>	Header: - Bearer Token Request parameter: - candidateId: candidate's id	Get all job applicants by candidate's id
GET	api/v1/job_applicants/<str: id>	Header: - Bearer Token Request parameter: - jobId: job’s id	Get all job applicants by job's id
POST	api/v1/job_applicants/<str: id>	Header: - Bearer Token Request parameter: - resumeId: resume’s id - jobId	Apply for a job
PUT	api/v1/job_applicants/update/<str:id>	Header: - Bearer Token Request parameter: - status: job applicant status - jobId	Update for a job applicant

Table 7.6: Backend implementation: API of job_applicants routes

Internship routes:

Method	Route	Parameter	Description
GET	api/v1/internship/<str: id>	Header: - Bearer Token Request parameter: - internshipId	Get an internship by internship's id
GET	api/v1/internship/candidate/<str: id>	Header: - Bearer Token Request parameter: - candidateId: candidate's id	Get all internship by candidate's id
GET	api/v1/internship	Header: - Bearer Token	Get all internship
POST	api/v1/internship/<str: id>	Header: - Bearer Token Request parameter: - resumeId: resume's id - internshipId	Apply for an internship
PUT	api/v1/internship/<str: id>	Header: - Bearer Token Request parameter: - result - process - internshipId	Update for an internship

Table 7.7: Backend implementation: API of internship routes

App routes:

Method	Route	Parameter	Description
GET	api/v1/app	Header: - Bearer Token Request parameter: - search	Search

Table 7.8: Backend implementation: API of search routes

7.3.3. Cloud service

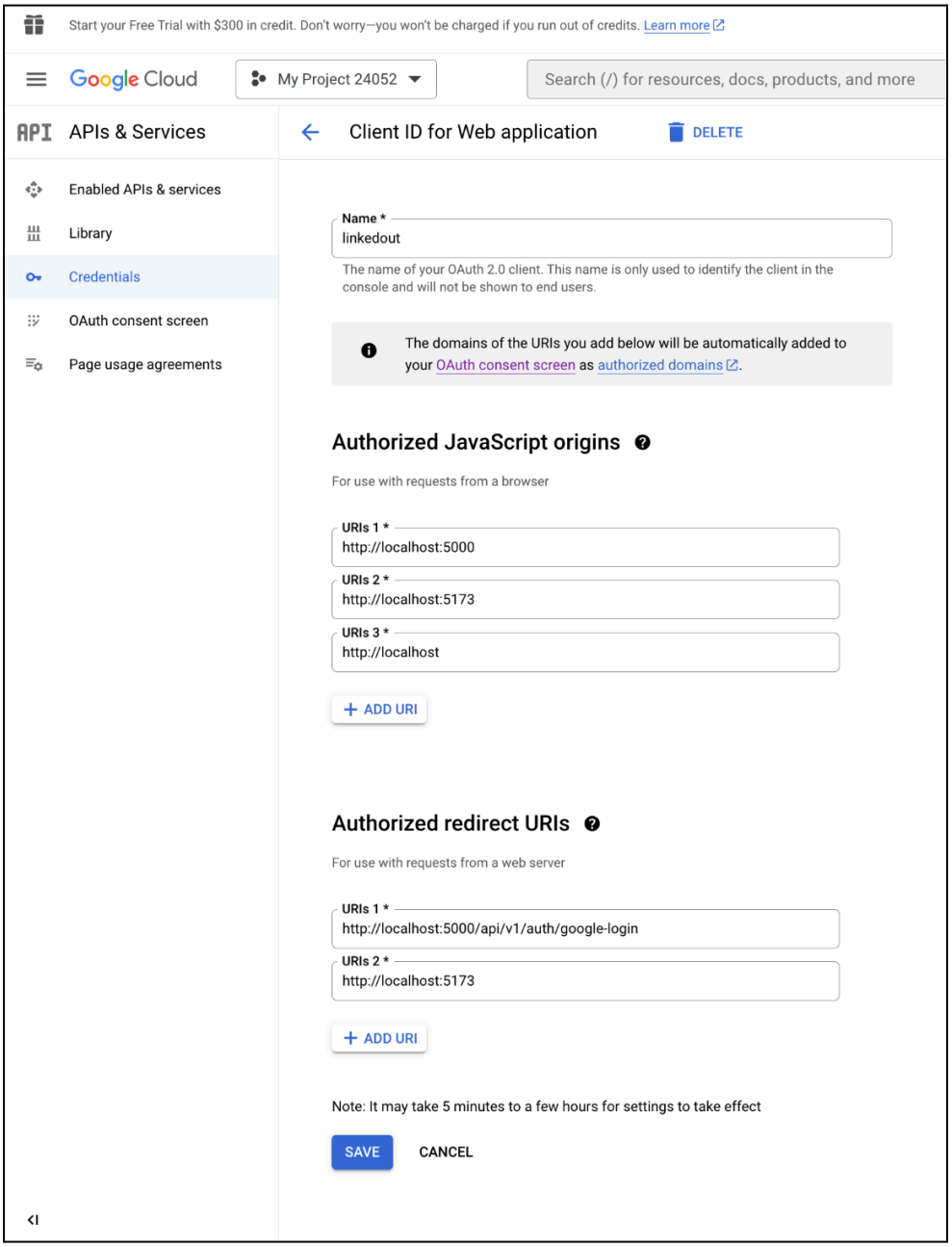


Figure 7.29: Google OAuthentication implementation

This is our Google OAuthentication implementation, the image above is the API & SERVICE for Google Cloud for setup the OAuth

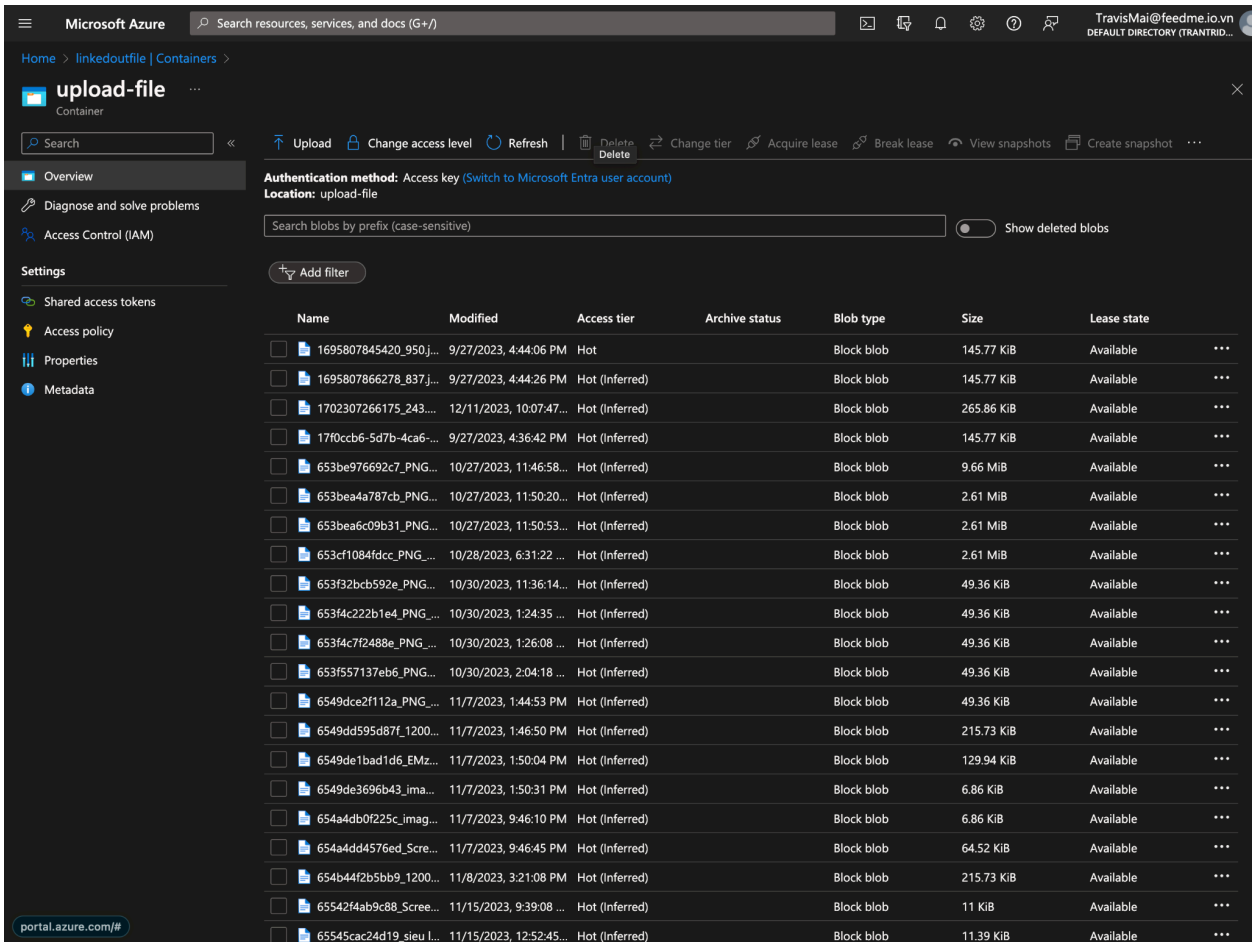


Figure 7.30: Azure Blob Storage deployment

This is our deployment for Azure Blob Storage for storing our user file such as avatar, resume, …

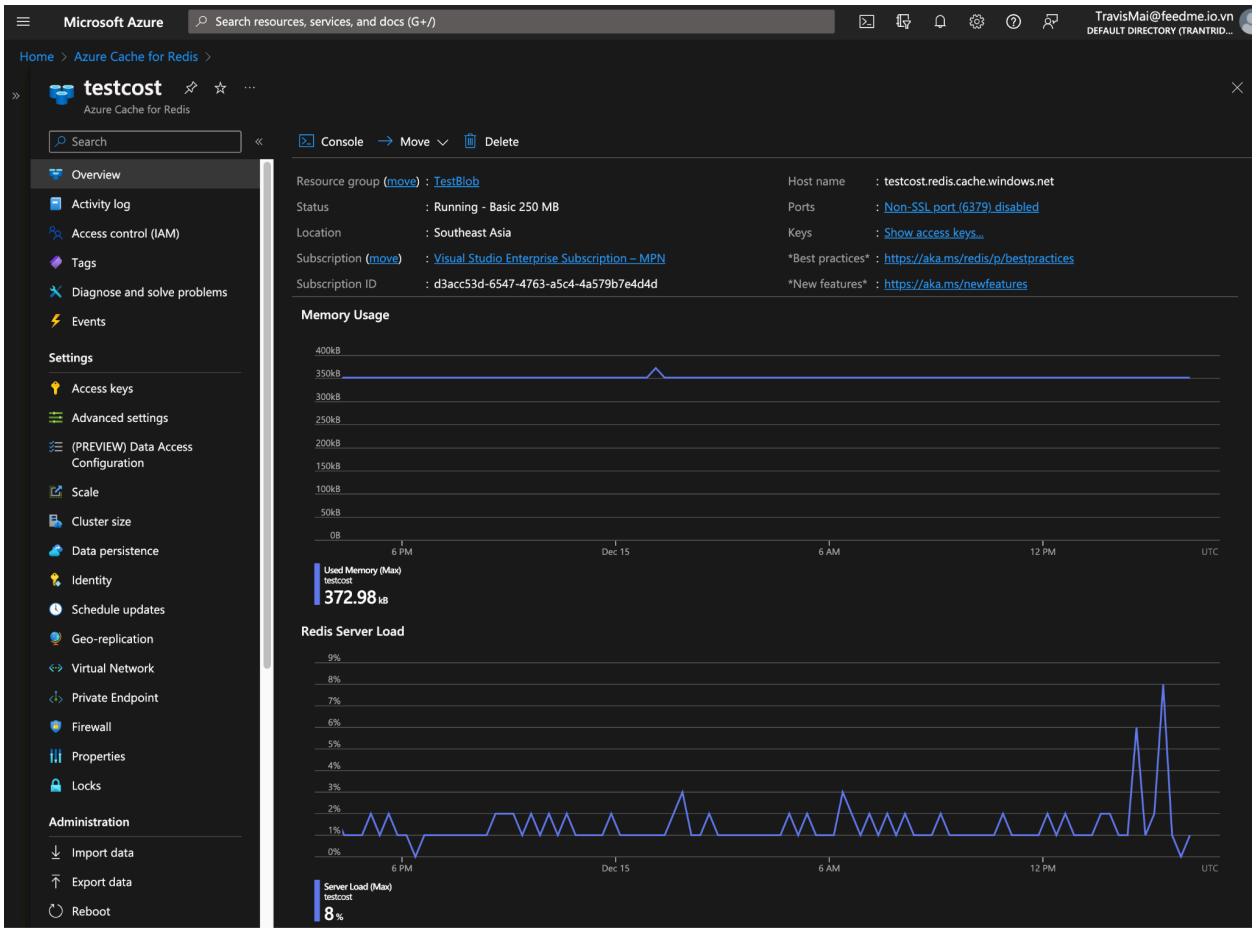


Figure 7.31: Azure Cache for Redis deployment

This is our Azure Cache for Redis deployment in Cloud service

8. Testing

8.1. Unit testing by Jest

8.1.1. API testing

```
PASS test/unit_test/controller/authController.spec.ts (10.208 s)
  AuthController
    checkRole
      ✓ should return true if the role matches the token payload (29 ms)
      ✓ should return false if the role does not match the token payload (3 ms)
      ✓ should return 403 if token verification fails (1 ms)
    login
      ✓ should return student and token if role is student (2 ms)
      ✓ should return staff and token if role is staff (4 ms)

PASS test/unit_test/controller/jobController.spec.ts (10.353 s)
  JobController
    findAll
      ✓ should return cached data if available (9 ms)
      ✓ should return job list and cache it if not cached (5 ms)
      ✓ should return 404 if no jobs found (1 ms)
      ✓ should handle errors
    findAllByCompanyId
      ✓ should return jobs for the company
      ✓ should return 404 if no jobs found for the company (1 ms)
      ✓ should handle errors
    findOne
      ✓ should return job by id (1 ms)
      ✓ should return cached job by id
      ✓ should return 404 if job not found
      ✓ should handle errors (1 ms)
      ✓ should return 400 for invalid UUID format
    create
      ✓ should create a new job (1 ms)
      ✓ should handle errors during job creation
    update
      ✓ should update a job (1 ms)
      ✓ should return 404 if job not found for update
      ✓ should handle errors during job update
      ✓ should return 400 if the id is invalid
    delete
      ✓ should delete a job (1 ms)
      ✓ should delete a job
      ✓ should return 404 if job not found for deletion
    findJobsWithCriteria
      ✓ should return jobs based on criteria
      ✓ should return 404 if no jobs found based on criteria
      ✓ should handle errors during finding jobs with criteria
```

```
PASS test/unit_test/controller/appController.spec.ts (9.631 s)
  AppController
    ✓ should be defined (26 ms)
    ✓ should search (3 ms)
    ✓ should return error (2 ms)

PASS test/unit_test/service/internshipService.spec.ts (9.643 s)
  InternshipService
    ✓ should be defined (14 ms)
    ✓ should create a new internship (6 ms)
    ✓ should find all internships (2 ms)
    ✓ should find one internship by id (2 ms)
    ✓ should update an internship (4 ms)
    ✓ should delete an internship (4 ms)
    ✓ should find all internships by company id (1 ms)
    ✓ should find all internships by student id (2 ms)

PASS test/unit_test/service/studentService.spec.ts (9.76 s)
  StudentService
    ✓ should be defined (9 ms)
    ✓ should return a list of students (1 ms)
    ✓ should return a student by id
    ✓ should create a student (1 ms)
    ✓ should update a student
    ✓ should delete a student
    ✓ should find a student by email (1 ms)
    ✓ should find a student by anything
    ✓ should find a resume by student id and resume id
    ✓ should find a resume by student id and resume id

PASS test/unit_test/service/staffService.spec.ts (9.984 s)
  StaffService
    ✓ should be defined (15 ms)
    ✓ should find all staff (2 ms)
    ✓ should find one staff by id (3 ms)
    ✓ should create a new staff (3 ms)
    ✓ should update a staff (3 ms)
    ✓ should delete a staff (3 ms)
    ✓ should find staff by email (3 ms)
    ✓ should get a random staff (2 ms)

PASS test/unit_test/controller/jobApplicantsController.spec.ts (10.103 s)
  JobApplicantsController
    applyJob
      ✓ should apply for a job (6 ms)
      ✓ should delete a job if already applied (1 ms)
      ✓ should return 404 if student not found
      ✓ should return 500 INTERNAL SERVER ERROR if unexpected error occur (2 ms)
    findAllCandidateByJobId
      ✓ should return all candidates for a job (1 ms)
      ✓ should return 404 if job not found
      ✓ should return 404 if no applicant found (1 ms)
      ✓ should return 500 if unexpected error occur (1 ms)
    findAllJobByCandidateId
      ✓ should return all jobs applied by a candidate (2 ms)
      ✓ should return 404 if no job found
      ✓ should return 500 if unexpected error occur (1 ms)
    findOne
      ✓ should return a job applicant
      ✓ should return 404 if job applicant not found (1 ms)
      ✓ should return 500 if unexpected error occurs
    update
      ✓ should update a job applicant
      ✓ should return 404 if job applicant not found (1 ms)
      ✓ should return 500 if unexpected error occurs
```

```
PASS test/unit_test/controller/internshipController.spec.ts (10.481 s)
  InternshipController
    ✓ should be defined (11 ms)
    applyInternship
      ✓ should apply for an internship and return created internship (1 ms)
      ✓ should return 404 if student not found (1 ms)
      ✓ should handle errors
    findAll
      ✓ should return all internships (1 ms)
      ✓ should return 404 if no internships found
      ✓ should handle errors (1 ms)
    findOne
      ✓ should return an internship by id
      ✓ should return 404 if internship not found (1 ms)
      ✓ should handle errors
    update
      ✓ should update an internship and return updated internship
      ✓ should handle errors
    findByCandidateId
      ✓ should return all internships by candidate id
      ✓ should return 404 if no internships found for candidate
      ✓ should handle errors

PASS test/unit_test/controller/studentController.spec.ts (10.52 s)
  StudentController
    getMyProfile
      ✓ should return a student profile in cache data (8 ms)
      ✓ should return a student profile in database (1 ms)
      ✓ should return an error when there is no student in the database
      ✓ should throw error when fetch a student profile in database (1 ms)
    findAll
      ✓ should return a list of students in cache
      ✓ should return a list of students in database (1 ms)
      ✓ should throw error with unexpected error with findAll (1 ms)
      ✓ should throw 404 error when no student in the list
    findOne
      ✓ should return a student by id in cache (2 ms)
      ✓ should throw 400 error if the id is invalid
      ✓ should return a student by id in database (1 ms)
      ✓ should throw 404 NOT FOUND if there is no student match (1 ms)
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error
    create
      ✓ should create new student successfully (1 ms)
      ✓ should throw 400 BAD REQUEST if the student is already registered
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error (1 ms)
    update
      ✓ should throw 400 error if the id is invalid
      ✓ should throw 404 NOT FOUND when no student found
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error (1 ms)
      ✓ should update student information successfully
    delete
      ✓ should delete student successfully
      ✓ should throw 404 NOT FOUND if the student is not found (1 ms)
      ✓ should throw 400 BAD REQUEST if the id is invalid
    login
      ✓ should login successfully
      ✓ should login successfully when cache data available
      ✓ should throw 401 UNAUTHORIZED if the student is not found (1 ms)
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error
    logout
      ✓ should logout successfully
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error
```

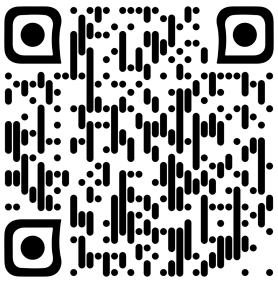
```
PASS test/unit_test/controller/staffController.spec.ts (10.58 s)
  StaffController
    getMyProfile
      ✓ should return a staff profile in cache data (15 ms)
      ✓ should return a staff profile in database (1 ms)
      ✓ should return an error when there is no staff in the database (1 ms)
      ✓ should throw error when fetch a staff profile in database (1 ms)
    findAll
      ✓ should return a list of staffs in cache (3 ms)
      ✓ should return a list of staffs in database (1 ms)
      ✓ should throw error with unexpected error with findAll (1 ms)
      ✓ should throw 404 error when no staff in the list
    findOne
      ✓ should return a staff by id in cache (1 ms)
      ✓ should throw 400 error if the id is invalid
      ✓ should return a staff by id in database (1 ms)
      ✓ should throw 404 NOT FOUND if there is no staff match (1 ms)
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error
    create
      ✓ should create new staff successfully (1 ms)
      ✓ should throw 400 BAD REQUEST if the staff is already registered
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error
    update
      ✓ should throw 400 error if the id is invalid (1 ms)
      ✓ should throw 403 UNAUTHORIZED when id not match (1 ms)
      ✓ should throw 404 NOT FOUND when no staff found (1 ms)
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error
      ✓ should update staff information successfully (1 ms)
    delete
      ✓ should delete staff successfully
      ✓ should throw 404 NOT FOUND if the staff is not found (1 ms)
      ✓ should throw 400 BAD REQUEST if the id is invalid
    login
      ✓ should login successfully
      ✓ should login successfully when cache data available
      ✓ should throw 401 UNAUTHORIZED if the staff is not found (1 ms)
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error
    logout
      ✓ should logout successfully (1 ms)
      ✓ should throw 500 INTERNAL SERVER ERROR if there is an unexpected error (1 ms)

PASS test/unit_test/service/azureBlobService.spec.ts
  AzureBlobService
    ✓ should return a block blob client (1 ms)
    ✓ should upload a file
    ✓ should delete a file

PASS test/unit_test/service/authService.spec.ts
  AuthService
    ✓ should be defined (7 ms)
    ✓ should generate jwt token (2 ms)
    ✓ should check if token is blacklisted (2 ms)
    ✓ should return user information from google login (2 ms)
    ✓ should return no user from google login (3 ms)

PASS test/unit_test/service/redisService.spec.ts
  RedisService
    ✓ should be defined (6 ms)
    ✓ should get object by key (2 ms)
    ✓ should set object by key value (1 ms)
    ✓ should delete object by key (2 ms)
```

```
PASS test/unit_test/controller/companyController.spec.ts (11.029 s)
  CompanyController
    ✓ should be defined (5 ms)
    getMyProfile
      ✓ should return the profile of the company (2 ms)
      ✓ should return the profile of the company from cache
      ✓ should return 404 if company not found (1 ms)
      ✓ should handle errors
    findAll
      ✓ should return all companies
      ✓ should return companies from cache (1 ms)
      ✓ should return 404 if no companies found
      ✓ should handle errors (1 ms)
    findOne
      ✓ should return a company by ID
      ✓ should return a company from cache (1 ms)
      ✓ should return 404 if company not found (1 ms)
      ✓ should return 400 for invalid UUID
      ✓ should handle errors
    create
      ✓ should create a new company and return the company along with a JWT token (1 ms)
      ✓ should return conflict if email already exists (1 ms)
      ✓ should handle errors
    update
      ✓ should update an existing company (1 ms)
      ✓ should return unauthorized if credentials are invalid
      ✓ should return 404 if company not found (1 ms)
      ✓ should handle errors
      ✓ should throw 400 BAD REQUEST if id is invalid
    delete
      ✓ should delete an existing company (3 ms)
      ✓ should return 404 if company not found
      ✓ should return 400 for invalid UUID format
    login
      ✓ should login a company and return a token (1 ms)
      ✓ should return 401 for invalid credentials
      ✓ should return token from cache if already authenticated (1 ms)
      ✓ should handle errors
    logout
      ✓ should logout a company and delete the token
      ✓ should handle errors

PASS test/unit_test/service/jobApplicantService.spec.ts
  JobApplicantsService
    ✓ should be defined (3 ms)
    ✓ should create a new job applicant (1 ms)
    ✓ should find a job applicant by id (1 ms)
    ✓ should find all job applicants by job id (1 ms)
    ✓ should find all job applicants by candidate id (2 ms)
    ✓ should find a job applicant by job id and candidate id (2 ms)
    ✓ should delete a job applicant (1 ms)
    ✓ should update a job applicant (1 ms)
```

```
PASS test/unit_test/service/companyService.spec.ts
  CompanyService
    ✓ should be defined (7 ms)
    ✓ should find all companies (2 ms)
    ✓ should find one company by id (2 ms)
    ✓ should create a new company (1 ms)
    ✓ should update a company (1 ms)
    ✓ should delete a company (1 ms)
    ✓ should find company by email (1 ms)
    ✓ should find company by anything (1 ms)

PASS test/unit_test/service/databaseInitializeService.spec.ts
  DatabaseInitializerService
    ✓ should be defined (5 ms)
    ✓ should call createQueryRunner (1 ms)
    ✓ should call queryRunner.connect (2 ms)
    ✓ should call queryRunner.query (1 ms)
    ✓ should call queryRunner.release (1 ms)

PASS test/unit_test/service/appService.spec.ts
  AppService
    ✓ should be defined (2 ms)
    ✓ should search (2 ms)

Test Suites: 19 passed, 19 total
Tests:       224 passed, 224 total
Snapshots:   0 total
Time:        11.522 s
Ran all test suites.
```

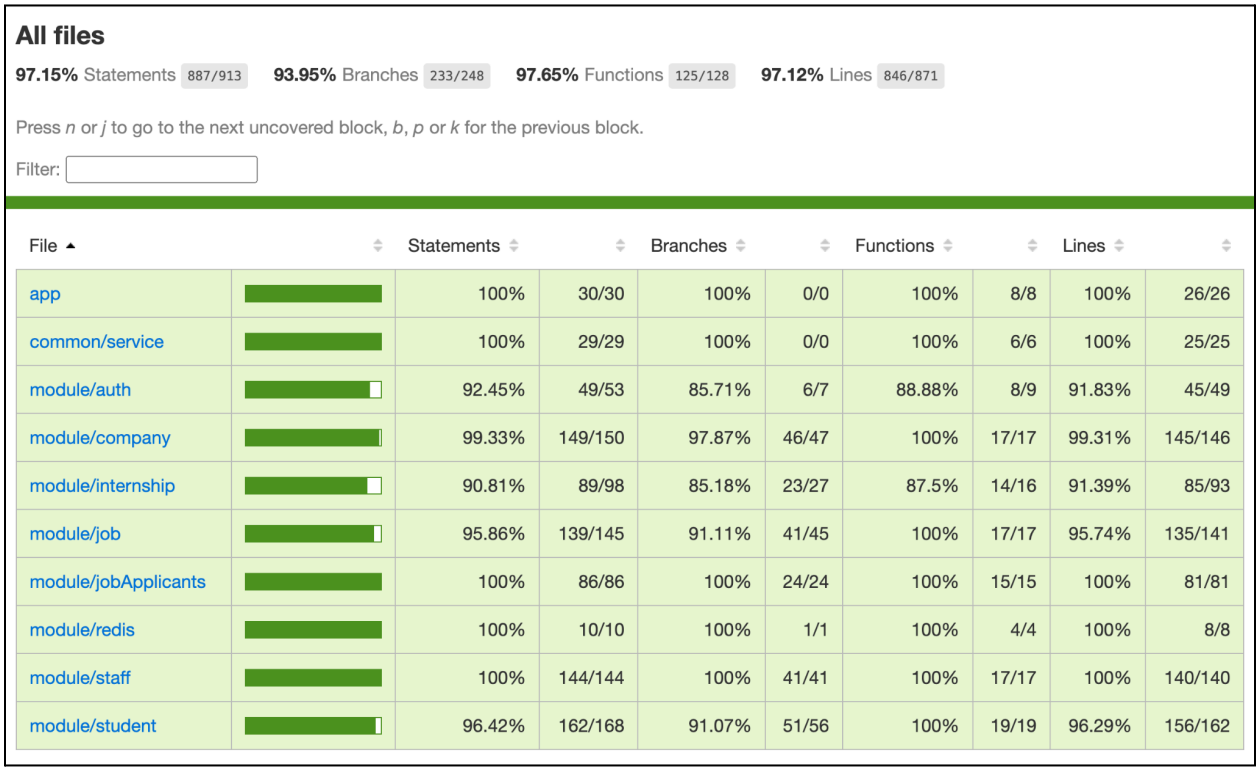
Figure 8.1: 224 tests for the APIs

8.1.2. Test coverage



Test coverage report of API logic code: https://travismai.github.io/LinkedOut/lcov-report/

Link to unit test folder:
https://github.com/TravisMai/LinkedOut/tree/main/backend/test/unit_test



All files

97.15% Statements 887/913 **93.95%** Branches 233/248 **97.65%** Functions 125/128 **97.12%** Lines 846/871

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
app	100%	30/30	100%	0/0	100%	8/8	100%	26/26
common/service	100%	29/29	100%	0/0	100%	6/6	100%	25/25
module/auth	92.45%	49/53	85.71%	6/7	88.88%	8/9	91.83%	45/49
module/company	99.33%	149/150	97.87%	46/47	100%	17/17	99.31%	145/146
module/internship	90.81%	89/98	85.18%	23/27	87.5%	14/16	91.39%	85/93
module/job	95.86%	139/145	91.11%	41/45	100%	17/17	95.74%	135/141
module/jobApplicants	100%	86/86	100%	24/24	100%	15/15	100%	81/81
module/redis	100%	10/10	100%	1/1	100%	4/4	100%	8/8
module/staff	100%	144/144	100%	41/41	100%	17/17	100%	140/140
module/student	96.42%	162/168	91.07%	51/56	100%	19/19	96.29%	156/162

Figure 8.2: Test coverage report for backend unit test

The test coverage overview report provides a comprehensive evaluation of the backend components, focusing on controllers and service files. Key highlights from the analysis include:

- **Statement Coverage**: Achieving a commendable **97.15%** coverage demonstrates thorough validation of individual code statements.
- **Branch Coverage**: With **93.95%** coverage, decision branches have been extensively tested, ensuring robust logic flow under various conditions.
- **Function Coverage**: The high coverage rate of **97.65%** for functions reflects meticulous testing of all functional aspects of the codebase.

- **Line Coverage**: At **97.12%**, line coverage underscores the detailed scrutiny applied to each line of code, minimizing the risk of undiscovered bugs.

These impressive coverage metrics not only validate the reliability and stability of backend functionalities but also instill confidence in the quality of the application. Moving forward, leveraging these insights will guide refinement of test suites, address coverage gaps, and prioritize testing efforts effectively, ensuring ongoing code quality and reliability amidst evolving business requirements and codebase changes.

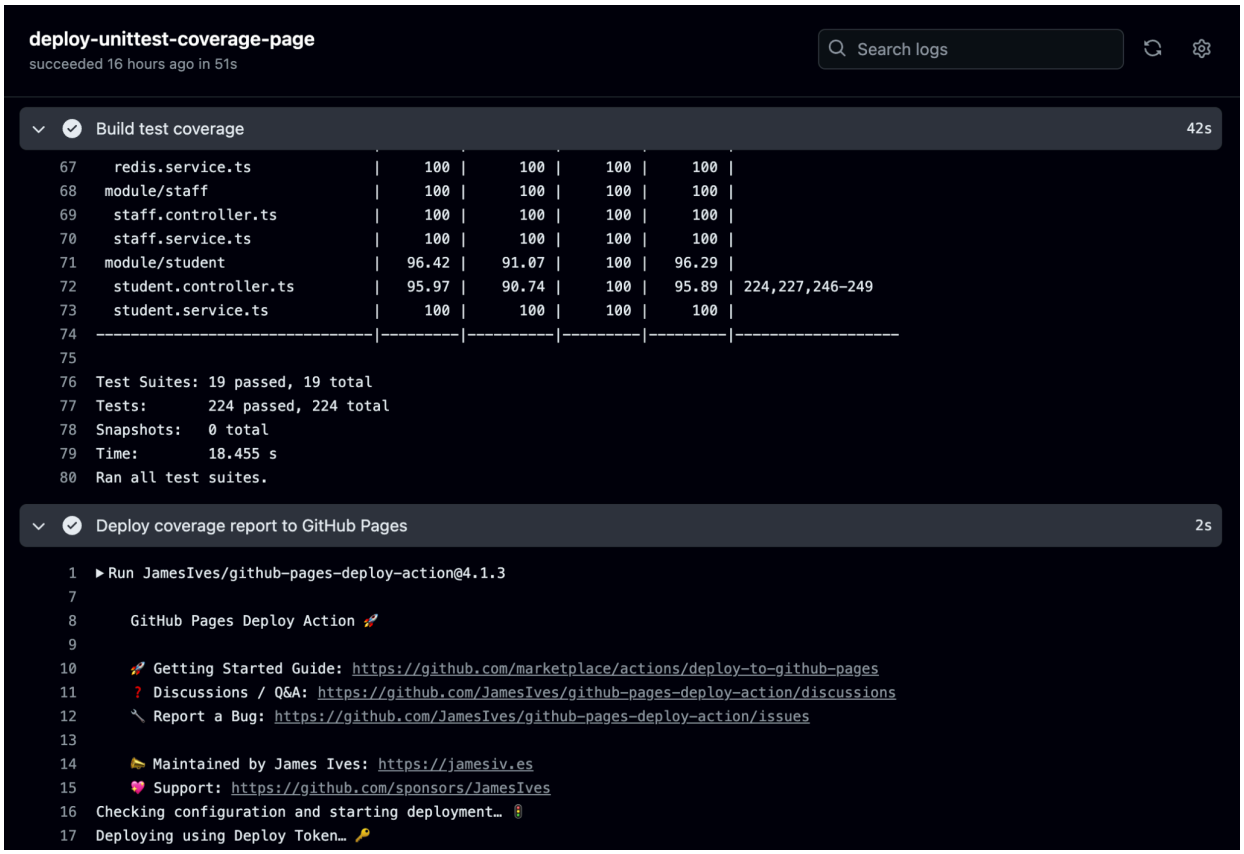


Figure 8.3: Github Action deploy unit test coverage job to Github Page

8.2. Workload testing by Apache Jmeter

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Get current logged in student	300	102	0	2048	129.91	0.00%	1.00304	1.13	0.45	1154
Get all student	300	68	0	937	95.38	0.00%	1.00433	34.04	0.44	34703.5
TOTAL	300	85	0	1492.5	112.645	0	1.003685	17.585	0.445	17928.75

Table 8.1: Statistic report of the workload testing for 2 APIs

The workload test conducted with Apache JMeter involved simulating 300 users interacting with the system, each with a ramp-up period of 300 seconds and a single loop. Analysis of the test results revealed that for the operation of retrieving information about the currently logged-in student, the system exhibited an average response time of 102 milliseconds, with a maximum of 2048 milliseconds and a minimum of 0 milliseconds. The average throughput for this operation was approximately 1.00304 requests per second. Similarly, for the task of retrieving details of all students, the average response time was slightly lower at 68 milliseconds, with a maximum of 937 milliseconds and a minimum of 0 milliseconds, and an average throughput of approximately 1.00433 requests per second. These results indicate that the system can efficiently handle a significant workload from concurrent users, with consistent response times and throughput rates, ensuring stable performance under realistic conditions.

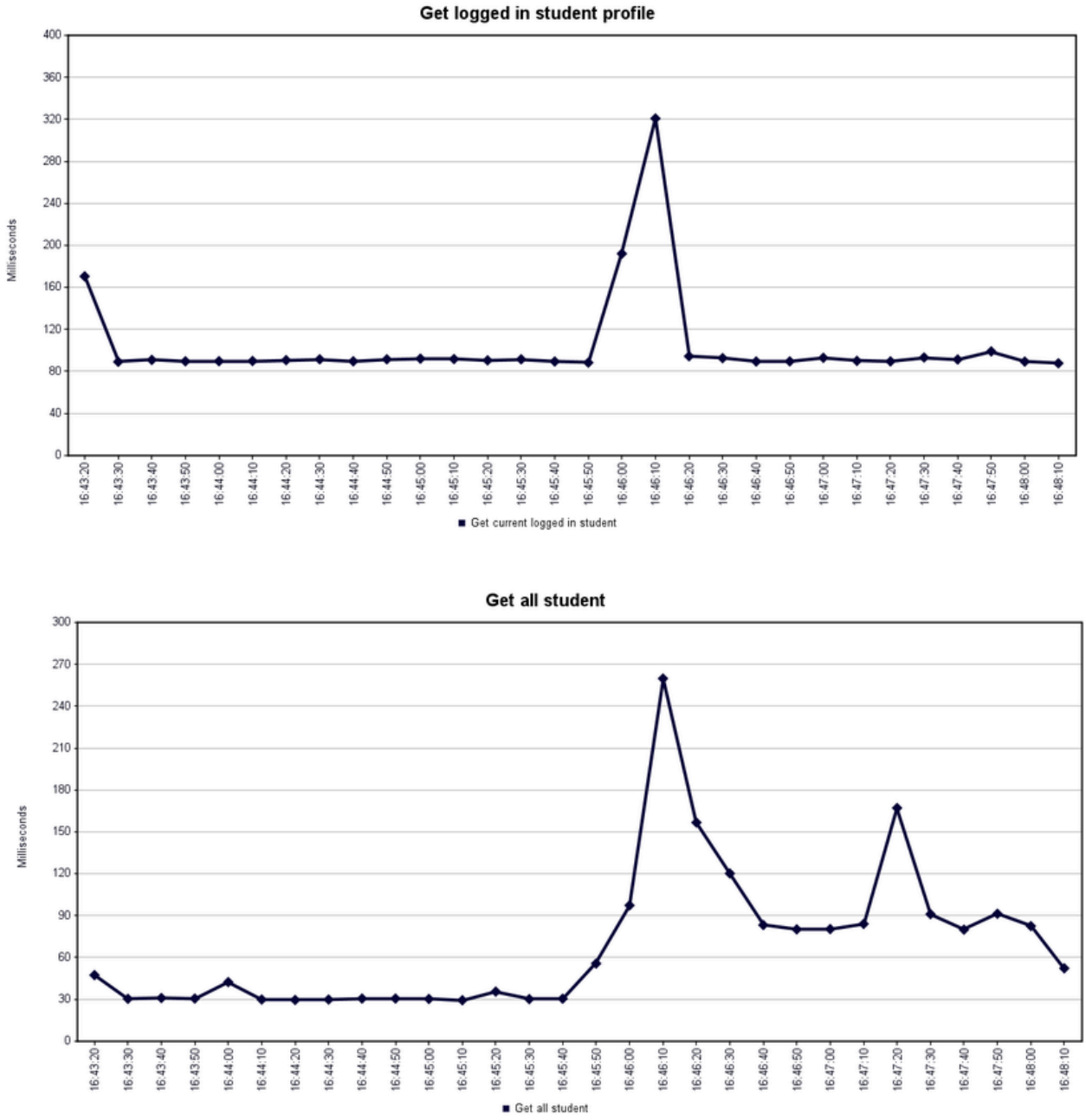


Figure 8.4: Time graph report for workload testing

8.3. Manual testing

Test Case ID	Test Description	Result
STD-01	Verify successful login with valid credentials	Pass
STD-02	Verify failed login with invalid credentials	Pass
STD-03	Verify successful signup with valid information	Pass
STD-04	Verify signup with already existing email address	Pass
STD-05	Verify signup with empty mandatory fields	Pass
STD-06	Verify successful application submission with complete and valid information	Pass
STD-07	Verify application is not available as no resume is created	Pass
STD-08	Verify successful application submission with complete and valid information	Pass
STD-09	Verify application is not available as no resume is created	Pass
STD-10	Verify report upload for current internship job	Pass
STD-11	Verify successful upload of a profile picture with a valid image format	Pass
STD-12	Verify successful update of profile	Pass
STD-13	Verify search results	Pass
STD-14	Verify successful logout after clicking the logout button	Pass
STD-15	Verify user state after logging out	Pass
CPN-01	Verify successful login with valid credentials	Pass
CPN-02	Verify failed login with invalid credentials	Pass
CPN-03	Verify successful signup with valid information	Pass
CPN-04	Verify signup with already existing email address	Pass
CPN-05	Verify signup with empty mandatory fields	Pass

Test Case ID	Test Description	Result
CPN-06	Verify successful creation of a new job position with all required fields	Pass
CPN-07	Verify creation of a new job position with empty required fields	Pass
CPN-08	Verify successful open a job	Pass
CPN-09	Verify successful close a job	Pass
CPN-10	Verify successful update to a job information	Pass
CPN-11	Verify successful job deletion with confirmation	Pass
CPN-12	Verify successful approve a job applicant	Pass
CPN-13	Verify successful process a job applicant	Pass
CPN-14	Verify successful rejection of a job applicant	Pass
CPN-15	Verify successful upload of a document for an internship application	Pass
CPN-16	Verify successful assignment of evaluation scores of Internship	Pass
CPN-17	Verify successful update of profile	Pass
CPN-18	Verify search results	Pass
CPN-19	Verify successful logout after clicking the logout button	Pass
CPN-20	Verify user state after logging out	Pass
STF-01	Verify successful login with valid credentials	Pass
STF-02	Verify failed login with invalid credentials	Pass
STF-03	Verify successful account verification	Pass
STF-04	Verify successful account disable	Pass
STF-05	Verify successful account enable	Pass

Test Case ID	Test Description	Result
STF-06	Verify successful update of student enrollment status	Pass
STF-07	Verify successful account deletion with confirmation	Pass
STF-08	Verify successful account verification	Pass
STF-09	Verify successful account disable	Pass
STF-10	Verify successful account enable	Pass
STF-11	Verify successful account deletion with confirmation	Pass
STF-12	Verify successful account verification	Pass
STF-13	Verify successful account deletion	Pass
STF-14	Verify successful job deletion with confirmation	Pass
STF-15	Verify search results	Pass
STF-16	Verify successful logout after clicking the logout button	Pass
STF-17	Verify user state after logging out	Pass

Table 8.2: Summary of Manual testing results

8.3.1. Student - Login testing

Test Case ID: STD-01

Test Description: Verify successful login with valid credentials

Steps:

1. Choose "Sign in with Google"
2. Perform login with Google pop-up screen using an existing account.

Expected result: The system successfully authenticates the user and redirects to the Student Homepage.

Result: User is redirected to the Student homepage.

Test Case ID: STD-02

Test Description: Verify failed login with invalid credentials

Steps:

1. Choose "Sign in with Google"
2. Perform login with Google pop-up screen using an non-existing account.

Expected result: The system fails to authenticate the user and displays an error message indicating an invalid credential. The user remains on the login page.

Result: An error message regarding an invalid credential is displayed, and the user remains on the login page.

8.3.2. Student - Sign Up testing

Test Case ID: STD-03

Test Description: Verify successful signup with valid information

Steps:

1. Open the Student signup page.
2. Enter a non-existing email address in the email field.
3. Enter Student ID, Full Name, Phone Number, Major, Year, Class Code
4. (Optional) Choose a image file for Avatar
5. Click the "Submit" button.

Expected result: The system successfully registers the user and displays a confirmation message or redirects them to the Student homepage. A pending verification is sent to Staff.

Result: User receives a confirmation message or is redirected to the Student homepage upon successful signup. A pending verification is sent to Staff.

Test Case ID: STD-04

Test Description: Verify signup with already existing email address

Steps:

1. Open the Student signup page.
2. Enter an already existing email address in the email field.
3. Enter Student ID, Full Name, Phone Number, Major, Year, Class Code
4. (Optional) Choose a image file for Avatar
5. Click the "Submit" button.

Expected result: The system fails to register the user and displays error messages indicating which email address is registered.

Result: Error messages regarding email address is registered are displayed, and the user remains on the signup page.

Test Case ID: STD-05

Test Description: Verify signup with empty mandatory fields

Steps:

1. Open the Student signup page.
2. Leave one or more mandatory fields empty
3. Click the "Submit" button.

Expected result: The system fails to register the user and displays error messages indicating which mandatory fields are missing.

Result: Error messages regarding missing mandatory fields are displayed, and the user remains on the signup page.

8.3.3. Student - Job apply testing

Test Case ID: STD-06

Test Description: Verify successful application submission with complete and valid information

Steps:

1. Open job display page.
2. Click the “Apply” button.
3. On the pop-up dialog, choose a Resume to apply.
4. Click the “Apply” button.

Expected result: The system successfully submits the application and the “Apply” button is turned to “Applied”. Company received the application.

Result: Job is successfully applied and the “Apply” button is turned to “Applied”. Company received the application.

Test Case ID: STD-07

Test Description: Verify application is not available as no resume is created

Steps:

1. Open job display page.
2. Click the “Apply” button.

Expected result: The system displays the message "You don't have any resume to apply". And the "Apply" button is disabled.

Result: User received the message "You don't have any resume to apply".

8.3.4. Student - Internship apply testing

Test Case ID: STD-08

Test Description: Verify successful application submission with complete and valid information

Steps:

1. Open job display page.
2. Click the "Apply Intern" button.
3. On the pop-up dialog, choose a Resume to apply.
4. Click the "Apply" button.

Expected result: The system successfully submits the application and the "Apply" button is turned to "Applied", "Apply Intern" button is turned to "Applied for Intern". Company received the application.

Result: Job is successfully applied and the "Apply" button is turned to "Applied", "Apply Intern" button is turned to "Applied for Intern". Company received the application.

Test Case ID: STD-09

Test Description: Verify application is not available as no resume is created

Steps:

1. Open job display page.

2. Click the “Apply Intern” button.

Expected result: The system displays the message “You don’t have any resume to apply”. And the “Apply” button is disabled.

Result: User received the message “You don’t have any resume to apply”.

8.3.5. Student - Upload Internship report testing

Test Case ID: STD-10

Test Description: Verify report upload for current internship job

Steps:

1. Open Internship job page.
2. Click “Upload report”.
3. In the pop-up dialog, browse a file to upload.
4. Click “Upload”

Expected result: The system successfully uploads the report. Display name of the uploaded file.

Result: User successfully uploaded the report. Name of the uploaded file is displayed.

8.3.6. Student - Update information testing

Test Case ID: STD-11

Test Description: Verify successful upload of a profile picture with a valid image format

Steps:

1. Access the profile display section.
2. Click the "Change Photo" button.

3. Select a valid image file (e.g., JPG, PNG) from your device.
4. Click the "Update Photo" button or perform the upload action.

Expected result: The system successfully uploads the profile picture and displays it on the profile section.

Result: The profile picture is uploaded, displayed on the profile, and retains its original format.

Test Case ID: STD-12

Test Description: Verify successful update of profile

Steps:

1. Access the profile display section.
2. Click the "  " button next to an existing information in a profile field (e.g., Objective, Social Media).
3. Enter valid information.
4. Click the "Update" button or perform the update action.

Expected result: The system successfully saves the edited information and updates the profile section.

Result: The edited information is saved and reflected in the updated profile view.

8.3.7. Student - Search testing

Test Case ID: STD-13

Test Description: Verify search results

Steps:

1. Enter a search phrase
2. Click the search button or press Enter on your keyboard.

Expected result: The system displays a search results drop down containing items matching the search phrase.

Result: The search results drop down appears with content matching the keywords or the entire phrase.

8.3.8. Student - Logout testing

Test Case ID: STD-14

Test Description: Verify successful logout after clicking the logout button

Steps:

1. Access the profile display section.
2. Click on the “Logout” button.

Expected result: The system successfully logs the user out and redirects them to the Home page.Any session information is cleared.

Result: User is redirected to the Home page upon clicking logout. Session information is cleared.

Test Case ID: STD-15

Test Description: Verify user state after logging out

Steps:

1. Access the profile display section.
2. Click on the “Logout” button.
3. Attempt to access the previously visited page requiring authentication.

Expected result: After logging out, the system should no longer recognize the user as logged in. Attempting to access restricted pages should redirect the user to the login page.

Result: The user cannot access restricted pages after logging out and is redirected to the login page.

8.3.9. Company - Login testing

Test Case ID: CPN-01

Test Description: Verify successful login with valid credentials

Steps:

1. Open the Company Login page.
2. Enter a valid username in the username field.
3. Enter the corresponding password for the username in the password field.
4. Click the "LogIn" button.

Expected result: The system successfully authenticates the user, displays a successful login message, and redirects to the Company Homepage.

Result: User sees a successful login message and is redirected to the Company homepage.

Test Case ID: CPN-02

Test Description: Verify failed login with invalid credentials

Steps:

1. Open the Company Login page.
2. Enter an invalid username and/or password in the password field.
3. Click the "LogIn" button.

Expected result: The system fails to authenticate the user and displays an error message indicating an invalid credential. The user remains on the login page.

Result: An error message regarding an invalid credential is displayed, and the user remains on the login page.

8.3.10. Company - Sign Up testing

Test Case ID: CPN-03

Test Description: Verify successful signup with valid information

Steps:

1. Open the Company signup page.
2. Enter a non-existing email address in the email field.
3. Enter valid information in mandatory fields
4. (Optional) Enter valid information in optional fields and choose a image file for Avatar
5. Click the "Submit" button.

Expected result: The system successfully registers the user and displays a confirmation message or redirects them to the Company homepage.

Result: User receives a confirmation message or is redirected to the Company homepage upon successful signup.

Test Case ID: CPN-04

Test Description: Verify signup with already existing email address

Steps:

1. Open the Company signup page.
2. Enter an existing email address in the email field.
3. Enter valid information in mandatory fields
4. (Optional) Enter valid information in optional fields and choose a image file for Avatar
5. Click the "Submit" button.

Expected result: The system fails to register the user and displays error messages indicating which email address is registered.

Result: Error messages regarding email address is registered are displayed, and the user remains on the signup page.

Test Case ID: CPN-05

Test Description: Verify signup with empty mandatory fields

Steps:

1. Open the Company signup page.
2. Enter an existing email address in the email field.
3. Leave one or more mandatory fields empty
4. (Optional) Enter valid information in optional fields and choose a image file for Avatar
5. Click the "Submit" button.

Expected result: The system fails to register the user and displays error messages indicating which mandatory fields are missing.

Result: Error messages regarding missing mandatory fields are displayed, and the user remains on the signup page.

8.3.11. Company - Create new job testing

Test Case ID: CPN-06

Test Description: Verify successful creation of a new job position with all required fields

Steps:

1. Navigate to the Job page.
2. Click the “Add” button to open the create a new job page.
3. Fill out all mandatory fields for the job position.
4. Optionally, fill out any additional details.
5. Click the "Create" button.

Expected result: The system successfully creates the new job position and displays a confirmation message, redirects to the Job page.

Result: The new job position is created successfully, and all details are displayed accurately in the job listing.

Test Case ID: CPN-07

Test Description: Verify creation of a new job position with empty required fields

Steps:

1. Navigate to the Job page.
2. Click the “Add” button to open the create a new job page.
3. Leave one or more mandatory fields empty
4. Optionally, fill out any additional details.

5. Click the "Create" button.

Expected result: The system fails to create the new job position and displays error messages indicating which mandatory fields are missing.

Result: Error messages regarding missing mandatory fields are displayed, and the user remains on the create job page.

8.3.12. Company - Job actions testing

Test Case ID: CPN-08

Test Description: Verify successful open a job

Steps:

1. Navigate to a Job display page.
2. Click the “Open Job” button.

Expected result: The system successfully opens the job, “Open Job” button turns to the “Close Job” button.

Result: “Open Job” button turns to the “Close Job” button.

Test Case ID: CPN-09

Test Description: Verify successful close a job

Steps:

1. Navigate to a Job display page.
2. Click the “Close Job” button.

Expected result: The system successfully closes the job, “Close Job” button turns to the “Open Job” button.

Result: "Close Job" button turns to the "Open Job" button.

Test Case ID: CPN-10

Test Description: Verify successful update to a job information

Steps:

1. Navigate to a Job display page.
2. Click the "Update Job" button.
3. In the pop-up dialog, enter new information about the job.
4. Click the "Update" button.

Expected result: The system successfully updates the job information. The updated information is reflected in the job listing or job detail page.

Result: The job information is updated accurately, and the new information is displayed correctly.

Test Case ID: CPN-11

Test Description: Verify successful job deletion with confirmation

Steps:

5. Navigate to a Job display page.
6. Click the "Delete Job" button.
7. In the pop-up dialog, click the "Delete" button.

Expected result: The system successfully deletes the job and displays a confirmation message or redirects to a success page. The deleted job should no longer be accessible in the job listing or search results.

Result: The job is deleted after confirmation, and it's no longer accessible in the system.

8.3.13. Company - Applicant actions testing

Test Case ID: CPN-12

Test Description: Verify successful approve a job applicant

Steps:

1. Navigate to an Application display page.
2. Click the “Approve” button.

Expected result: The system successfully marks the applicant as approved. The applicant's status should reflect their approval within the applicant list or profile.

Result: The applicant is approved, their status is updated.

Test Case ID: CPN-13

Test Description: Verify successful process a job applicant

Steps:

1. Navigate to an Application display page.
2. Click the “Process” button.

Expected result: The system successfully marks the applicant as processing. The applicant's status should reflect their procession within the applicant list or profile.

Result: The applicant is in processing, their status is updated.

Test Case ID: CPN-14

Test Description: Verify successful rejection of a job applicant

Steps:

3. Navigate to an Application display page.
4. Click the “Reject” button.

Expected result: The system successfully marks the applicant as rejected. The applicant's status should reflect their rejection within the applicant list or profile.

Result: The applicant is rejected, their status is updated.

8.3.14. Company - Internship actions

Test Case ID: CPN-15

Test Description: Verify successful upload of a document for an internship application

Steps:

1. Navigate to an Internship display page.
2. Click the “Upload file” button.
3. In the pop-up dialog, select a new file to upload.
4. Click the “Submit” button.

Expected result: The system successfully uploads the document and displays file name in the Internship information page.

Result: The resume is uploaded, and the document’s name is displayed.

Test Case ID: CPN-16

Test Description: Verify successful assignment of evaluation scores of Internship

Steps:

1. Navigate to an Internship display page.
2. Click the “Upload file” button.
3. In the pop-up dialog, enter the scoring point.
4. Click the “Submit” button.

Expected result: The system successfully saves the assigned scores and displays the result in the Internship information page.

Result: The scores are saved, displayed accurately.

8.3.15. Company - Update information testing

Test Case ID: CPN-17

Test Description: Verify successful update of profile

Steps:

1. Access the profile display and update by clicking on the Settings option of the drop down box when clicking on the avatar.
2. Enter valid information.
3. Click the "Update" button or perform the update action.

Expected result: The system successfully saves the edited information and updates the profile section.

Result: The edited information is saved and reflected in the updated profile view.

8.3.16. Company - Search testing

Test Case ID: CPN-18

Test Description: Verify search results

Steps:

1. Enter a search phrase
2. Click the search button or press Enter on your keyboard.

Expected result: The system displays a search results drop down containing items matching the search phrase.

Result: The search results drop down appears with content matching the keywords or the entire phrase.

8.3.17. Company - Logout testing

Test Case ID: CPN-19

Test Description: Verify successful logout after clicking the logout button

Steps:

1. Clicking on the avatar.
2. Click the “Logout” option of the drop down box.

Expected result: The system successfully logs the user out and redirects them to the Home page.Any session information is cleared.

Result: User is redirected to the Home page upon clicking logout. Session information is cleared.

Test Case ID: CPN-20

Test Description: Verify user state after logging out

Steps:

1. Clicking on the avatar.
2. Click the “Logout” option of the drop down box.
3. Attempt to access the previously visited page requiring authentication.

Expected result: After logging out, the system should no longer recognize the user as logged in. Attempting to access restricted pages should redirect the user to the login page.

Result: The user cannot access restricted pages after logging out and is redirected to the login page.

8.3.18. Staff - Login testing

Test Case ID: STF-01

Test Description: Verify successful login with valid credentials

Steps:

1. Choose "Sign in with Google"
2. Perform login with Google pop-up screen using an existing account.

Expected result: The system successfully authenticates the user and redirects to the Staff Homepage.

Result: User is redirected to the Staff homepage.

Test Case ID: STF-02

Test Description: Verify failed login with invalid credentials

Steps:

1. Choose "Sign in with Google"

2. Perform login with Google pop-up screen using an non-existing account.

Expected result: The system fails to authenticate the user and displays an error message indicating an invalid credential. The user remains on the login page.

Result: An error message regarding an invalid credential is displayed, and the user remains on the login page.

8.3.19. Staff - Student actions testing

Test Case ID: STF-03

Test Description: Verify successful account verification

Steps:

1. Navigate to the Student page.
2. Click to open a Student dialog.
3. Click the “Verify” button.

Expected result: The system successfully verifies the account. The term “(Is not verified)” next to the student name has disappeared.

Result: The account is verified, the term “(Is not verified)” next to the student name has disappeared. The student receives confirmation, and can perform full functionality.

Test Case ID: STF-04

Test Description: Verify successful account disable

Steps:

1. Navigate to the Student page.

2. Click to open a Student dialog.
3. Click the “Disable” button.

Expected result: The system successfully disables the account. The term “(Disabled)” next to the student name appeared.

Result: The account is disabled, the term “(Disabled)” next to the student name appeared. The student can no longer use the account.

Test Case ID: STF-05

Test Description: Verify successful account enable

Steps:

1. Navigate to the Student page.
2. Click to open a Student dialog.
3. Click the “Enable” button.

Expected result: The system successfully enables the account. The term “(Disabled)” next to the student name has disappeared.

Result: The account is enabled, the term “(Disabled)” next to the student name has disappeared. The student can now use the account.

Test Case ID: STF-06

Test Description: Verify successful update of student enrollment status

Steps:

1. Navigate to the Student page.
2. Click to open a Student dialog.

3. Click the “Status” field to open the drop down list.
4. Choose the new status.
5. Click the “Update” button.

Expected result: The system successfully updates the student's enrollment status. The updated status is reflected in the student's profile or enrollment record.

Result: The status is updated accurately, and the new status is displayed correctly.

Test Case ID: STF-07

Test Description: Verify successful account deletion with confirmation

Steps:

1. Navigate to the Student page.
2. Click to open a Student dialog.
3. Click the “Delete” button.
4. On the pop-up text, click “Confirm”

Expected result: A clear confirmation text appears before deletion, explicitly stating the account will be deleted. Upon confirmation, the system successfully deletes the account, the account no longer exists.

Result: The confirmation text is clear, the account is deleted after confirmation, and the user cannot log in anymore.

8.3.20. Staff - Company actions testing

Test Case ID: STF-08

Test Description: Verify successful account verification

Steps:

1. Navigate to the Company page.
2. Click to open a Company dialog.
3. Click the “Verify” button.

Expected result: The system successfully verifies the account. The term “(Is not verified)” next to the company name has disappeared.

Result: The account is verified, the term “(Is not verified)” next to the company name has disappeared. The company receives confirmation, and can perform full functionality.

Test Case ID: STF-09

Test Description: Verify successful account disable

Steps:

1. Navigate to the Company page.
2. Click to open a Company dialog.
3. Click the “Disable” button.

Expected result: The system successfully disables the account. The term “(Disabled)” next to the company name appeared.

Result: The account is disabled, the term “(Disabled)” next to the company name appeared. The company can no longer use the account.

Test Case ID: STF-10

Test Description: Verify successful account enable

Steps:

1. Navigate to the Company page.
2. Click to open a Company dialog.
3. Click the “Enable” button.

Expected result: The system successfully enables the account. The term “(Disabled)” next to the company name has disappeared.

Result: The account is enabled, the term “(Disabled)” next to the company name has disappeared. The company can now use the account.

Test Case ID: STF-11

Test Description: Verify successful account deletion with confirmation

Steps:

1. Navigate to the Company page.
2. Click to open a Company dialog.
3. Click the “Delete” button.
4. On the pop-up text, click “Confirm”

Expected result: A clear confirmation text appears before deletion, explicitly stating the account will be deleted. Upon confirmation, the system successfully deletes the account, the account no longer exists.

Result: The confirmation text is clear, the account is deleted after confirmation, and the user cannot log in anymore.

8.3.21. Staff - Quick verify testing

Test Case ID: STF-12

Test Description: Verify successful account verification

Steps:

1. Navigate to the Verify page.
2. Click the “  ” button.

Expected result: The system successfully verifies the account.

Result: The account is verified, the account can perform full functionality.

Test Case ID: STF-13

Test Description: Verify successful account deletion

Steps:

1. Navigate to the Verify page.
2. Click the “  ” button.

Expected result: The system successfully deletes the account, the account no longer exists.

Result: The account is deleted, and the user cannot log in anymore.

8.3.22. Staff - Job action testing

Test Case ID: STF-14

Test Description: Verify successful job deletion with confirmation

Steps:

1. Navigate to the Job page.
2. Click to open a Job dialog.

3. Click the “Delete” button.
4. On the pop-up text, click “Confirm”

Expected result: A clear confirmation text appears before deletion, explicitly stating the job will be deleted. Upon confirmation, the system successfully deletes the job, the job no longer exists.

Result: The confirmation text is clear, the job is deleted after confirmation.

8.3.23.Staff - Search testing

Test Case ID: STF-15

Test Description: Verify search results

Steps:

1. Enter a search phrase
2. Click the search button or press Enter on your keyboard.

Expected result: The system displays a search results drop down containing items matching the search phrase.

Result: The search results drop down appears with content matching the keywords or the entire phrase.

8.3.24.Staff - Logout testing

Test Case ID: STF-16

Test Description: Verify successful logout after clicking the logout button

Steps:

1. Click the “Logout” button from the left side bar.

Expected result: The system successfully logs the user out and redirects them to the Home page. Any session information is cleared.

Result: User is redirected to the Home page upon clicking logout. Session information is cleared.

Test Case ID: STF-17

Test Description: Verify user state after logging out

Steps:

1. Click the “Logout” button from the left side bar.
2. Attempt to access the previously visited page requiring authentication.

Expected result: After logging out, the system should no longer recognize the user as logged in. Attempting to access restricted pages should redirect the user to the login page.

Result: The user cannot access restricted pages after logging out and is redirected to the login page.

9.Deployment

The deployment process for our project involved setting up key Azure resources, including Virtual Machines, a Storage Account, Azure Redis, and an Azure PostgreSQL Server. We then configured the Nginx server, deployed the frontend and backend applications, and finalized by configuring DNS to assign a domain name to our server, ensuring accessible and efficient operation.

9.1. Azure Cloud Infrastructure

9.1.1. Infrastructure Architect

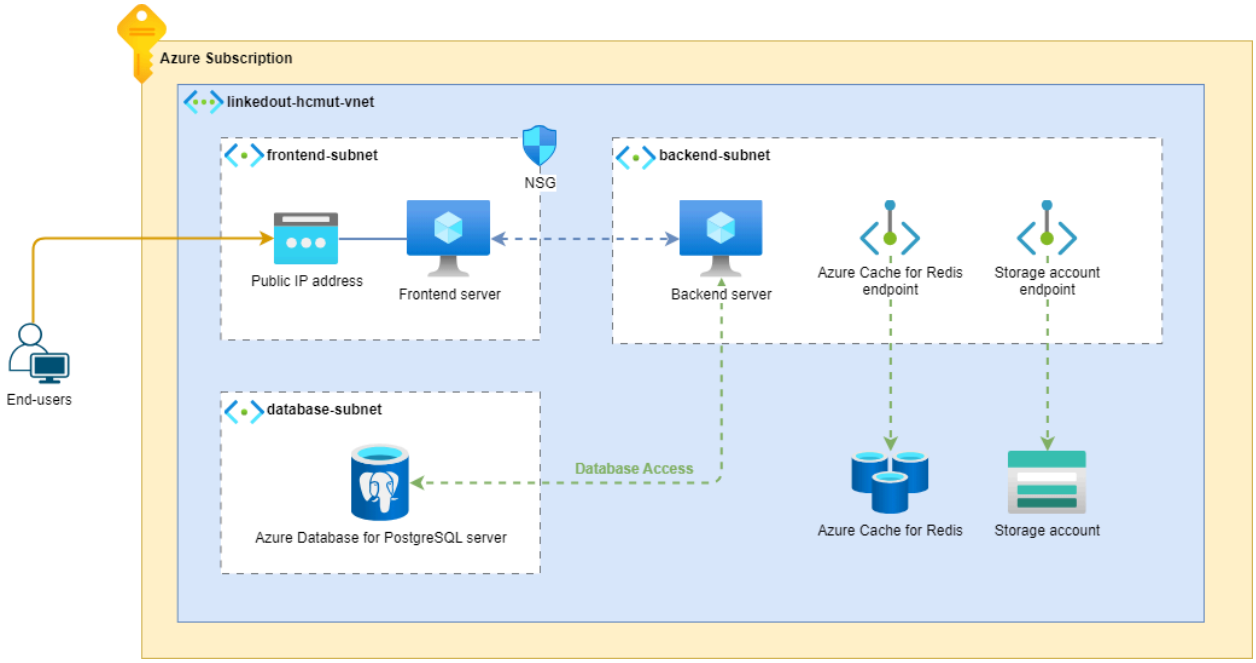


Figure 9.1: Current Azure infrastructure architecture

For the Azure infrastructure, we are using:

- An instance of Virtual machine [21] for hosting the Frontend server
- An instance of Virtual machine for hosting the Backend server
- An instance of Azure Database for PostgreSQL server [22] for the database
- An instance of Azure Cache for Redis for the caching

- An instance of Storage Account for blob files storing

About the connectivity between resources:

- All of the resources are placed inside the same virtual network.
- Only the Frontend server has a public IP address for end-users to access. So it has a Network Security Group [23] (NSG) to ensure security connections.
- Frontend server access to Backend server by internal private IP address.
- The backend server accesses Azure Database for PostgreSQL server by internal private IP address.
- Backend server accesses Azure Cache for Redis by a private endpoint.
- Backend server access Storage Account by a private endpoint [24].

9.1.2. Azure Database for PostgreSQL

Azure Database for PostgreSQL server configuration:

- SKU: Standard_B1ms (1 vCore)
- Tier: Burstable
- Storage Size: 32GB
- PostgreSQL version: 15.6
- Backup Type: Zone redundant - Retention period: 7 days

Additional server settings [25]:

- PostgreSQL extensions - UNACCENT: Normalize keywords, allows accent-insensitive text search.
- PostgreSQL extensions - UUID-OSSP: for generating universally unique identifiers (UUIDs) based on the Open Software Foundation's (OSF) standard.

This Azure Database for PostgreSQL server configuration is designed for cost-effectiveness and scalability while handling moderate workloads. The Standard_B1ms SKU with Burstable tier provides a balance between compute power and cost. The 32GB storage size offers sufficient space for data, and PostgreSQL version 15.6 ensures access to the latest features. Zone-redundant backups with a 7-day retention period safeguard data against accidental loss while optimizing storage usage. Additionally, the UNACCENT and UUID-OSSP extensions enhance the database's functionality for text search and unique identifier generation, streamlining application's operations.

In order to connect your backend server to this database, we set up the following environment variables:

- PGHOST: Set to server's address, <server name>.postgres.database.azure.com.
- PGUSER: Specifies the username for accessing the database.
- PGPORT: Standard port number for PostgreSQL connections, which is usually at port 5432.
- PGDATABASE: Specified database to connect to.
- PGPASSWORD: Password for the user.

9.1.3. Azure Cache for Redis

Azure Cache for Redis configuration:

- SKU: Basic
- Size: 250 MB cache
- Redis version: 6.0

A Basic SKU with a 250 MB cache size was selected for Azure Cache for Redis. This configuration prioritizes cost-effectiveness for development/test environments or

workloads with minimal cache demands. While Redis 6.0 provides the most recent features.

Connection between Azure Cache for Redis and the backend server is facilitated via access keys. Retrieved from the cache instance's authentication settings, hostname, port, and access key are used to configure the backend's client library. This library then establishes a secure connection and enables interaction with the cache for data management tasks. For security, access keys should be stored in environment variables or secure configurations.

9.1.4. Azure Storage Account

Azure Storage Account configuration:

- Type: Block Blob Storage
- SKU: Standard
- Storage Account Type: General Purpose V2
- File Structure: Flat Namespace
- Default access tier: Hot
- Replication: Read-access geo-redundant storage (RA-GRS)
- A Container with Anonymous access level: Container (anonymous read access for containers and blobs)

This configuration prioritizes cost-effective storage of unstructured data with frequent access. Block blobs in a Standard tier, General Purpose v2 account offer flexibility. Flat namespace simplifies management, while Hot access tier ensures data readiness. Read-access geo-redundant storage (RA-GRS) guarantees durability and minimizes downtime. Anonymous container/blob read access allows public access if applicable.

To grant secure access to the container for our backend server, we will utilize Shared Access Signatures (SAS). SAS tokens provide temporary, fine-grained access to storage account resources without requiring users to manage storage account credentials. By incorporating a SAS token within a URL constructed using the storage account information, we can authorize the backend server to interact with the container. This approach enhances security by eliminating the need for storing sensitive account keys on the server.

9.1.5. Azure Virtual Machine

Azure Virtual Machine for Front-end server configuration:

- Tier: Standard
- Size: D2 v3 (2 vCPUs, 8 GB RAM, 50 GB Temporary storage)
- Operating system: Linux (ubuntu 22.04)
- OS Disk: Premium SSD LRS, 30 GiB
- Network interface: Has a public IPv4 address, attached a Network security group

We configure the VM with these settings because they provide an optimal balance of performance and cost-effectiveness for a front-end server. A Standard tier offers a general-purpose compute option, well-suited for handling user requests. The D2 v3 size provides 2 vCPUs and 8 GB RAM, ensuring sufficient resources to manage incoming traffic. Choosing Ubuntu 22.04 as the operating system provides a lightweight and efficient platform for the front-end server. The Premium SSD LRS storage option for the 30 GiB OS disk guarantees fast boot times and application responsiveness, critical for a front-end server. Finally, assigning a public IPv4 address enables external accessibility, allowing users to interact with the front-end server over the internet. Additionally, a Network Security Group (NSG) is attached to further enhance security. This NSG acts as a firewall, meticulously controlling inbound and outbound traffic, safeguarding the front-end server from potential threats.

Azure Virtual Machine for Back-end server configuration:

- Tier: Standard
- Size: D2 v3 (2 vCPUs, 8 GB RAM, 50 GB Temporary storage)
- Operating system: Linux (ubuntu 22.04)
- OS Disk: Premium SSD LRS, 30 GiB
- Network interface: Only has a private IP address.

We configure the back-end server VM with these settings to prioritize security and efficient resource allocation. A Standard tier offers a cost-effective option for back-end processes. The D2 v3 size with 2 vCPUs and 8 GB RAM provides adequate resources for background tasks without exceeding needs. Selecting Ubuntu 22.04 as the operating system ensures a lightweight platform. The Premium SSD LRS storage option for the 30 GiB OS disk maintains fast boot times for the back-end server. For enhanced security, the network interface is configured with a private IP address only. This restricts direct external access, limiting the attack surface and protecting sensitive back-end data.

To facilitate secure development, we leverage SSH connections on port 22. Developers establish a direct and secure SSH tunnel to the front-end server, enabling them to connect and manage their code efficiently. This initial connection importantly bypasses the back-end server, isolating the development environment and enhancing security.

9.2. Server Setting on VM

9.2.1. Set up NGINX server on VM

```
server {

   server_name linkedout-hcmut.feedme.io.vn;

   # Serve the Vite-built frontend

   location / {

       root /home/{username}/linkedout/frontend;

       index index.html;

       try_files $uri $uri/ /index.html;

   }

   # Proxy requests to the backend service

   location /api/v1/ {

       proxy_pass http://localhost:4000;

       proxy_http_version 1.1;

       proxy_set_header Upgrade $http_upgrade;

       proxy_set_header Connection 'upgrade';

       proxy_set_header Host $host;

       proxy_cache_bypass $http_upgrade;

       # CORS configuration

       add_header 'Access-Control-Allow-Origin' '*';

       add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';

       add_header 'Access-Control-Allow-Headers' 'Content-Type';

       add_header 'Access-Control-Max-Age' 1728000;

       add_header 'Access-Control-Allow-Credentials' 'true';

       if ($request_method = 'OPTIONS') {
```

```
            add_header 'Content-Type' 'text/plain charset=UTF-8';
            add_header 'Content-Length' 0;
            return 204;
        }
    }
    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate
/etc/letsencrypt/live/linkedout-hcmut.feedme.io.vn/fullchain.pem; # managed by
Certbot
    ssl_certificate_key
/etc/letsencrypt/live/linkedout-hcmut.feedme.io.vn/privkey.pem; # managed by
Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}
server {
    if ($host = linkedout-hcmut.feedme.io.vn) {
        return 301 https://$host$request_uri;
    } # managed by Certbot
    listen 80;
    listen [::]:80;
    server_name linkedout-hcmut.feedme.io.vn;
    return 404; # managed by Certbot
}
```

This NGINX configuration is set up to serve a web application named "LinkedOut" on the domain "linkedout-hcmut.feedme.io.vn". Here's an overview of the configuration:

1. **Frontend Serving**: Requests to the root URL "/" are served by the Vite-built frontend located in the directory "/home/{username}/linkedout/frontend". NGINX will try to serve static files first, falling back to serving "index.html" for client-side routing.
2. **Backend Proxying**: Requests to the "/api/v1/" endpoint are proxied to a backend service running on localhost port 4000. NGINX is configured to handle WebSocket connections with proxy_http_version and proxy_set_header directives.
3. **CORS Configuration**: CORS (Cross-Origin Resource Sharing) headers are added to allow cross-origin requests from any origin ("*") with specified methods (GET, POST, OPTIONS) and headers (Content-Type). Additionally, CORS preflight requests (OPTIONS method) are handled with a 204 response and appropriate headers.
4. **SSL/TLS Configuration**: The server listens on port 443 (HTTPS) and has SSL/TLS certificates obtained from Let's Encrypt configured for encryption. The "fullchain.pem" and "privkey.pem" files are specified as the SSL certificates for the domain.
5. **Redirect from HTTP to HTTPS**: Requests to the HTTP port 80 are redirected to HTTPS using a 301 redirect.

Overall, this NGINX configuration ensures that the frontend application is served correctly, API requests are proxied to the backend, and appropriate security measures like SSL/TLS encryption and CORS are implemented.

9.2.2. Setting source code and PM2 for thread process handling

- Server Environment Setup:
  - Frontend static code resides at "/home/{username}/linkedout/frontend".
  - Backend code is located at "/home/{username}/linkedout/backend".
- NGINX Configuration:
  - NGINX serves as the front-facing web server.
  - Configured to serve static assets of the frontend application.
  - Static content is distributed from the specified frontend directory.
- PM2 Implementation for Backend:
  - PM2 utilized as a process manager for Node.js applications.
  - Ensures continuous availability and stability of backend services.
  - Manages the lifecycle of backend processes effectively.
- Continuous Availability Assurance:
  - PM2 ensures backend processes remain operational throughout the VM's lifespan.
  - Comprehensive setup enhances reliability and resilience of the hosting environment.
  - Optimizes performance and availability of both frontend and backend applications.

```
linkedout-noventiq@linkedout-hcmut:/etc/nginx/sites-available$ pm2 status
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem
0	backend	default	N/A	fork	2667	2h	1	online	0%	61.9mb

Figure 9.2: PM2 thread processing for backend service

9.2.3. Github Action Automatic Deployment

Github Action source code: https://github.com/TravisMai/LinkedOut/tree/main/.github

This GitHub Actions workflow, named "Deploy project," automates the deployment of both the frontend and backend components of a project upon pushing to the main branch. The workflow is structured into two jobs: frontend-deploy and backend-deploy, ensuring a streamlined deployment process.

Workflow Flow and Main Idea:

Trigger:

- Initiates on any push to the main branch.

Frontend Deployment (frontend-deploy job):

- Runs on ubuntu-latest.
- Steps:
    - Checks out the repository.
    - Sets up SSH access to the frontend host using secrets for SSH credentials.
    - Builds the frontend application.
    - Syncs the built frontend code to the specified directory on the host server.

Backend Deployment (backend-deploy job):

- Depends on the successful completion of the frontend-deploy job.
- Runs on ubuntu-latest.
- Steps:
    - Checks out the repository.
    - Sets up SSH access to the backend host using secrets for SSH credentials.
    - Syncs the backend source code to the specified directory on the host server.
    - Builds the backend application with the provided environment variables.

This workflow ensures that any updates to the main branch trigger a complete deployment of both frontend and backend applications, maintaining consistency and reducing manual deployment efforts.

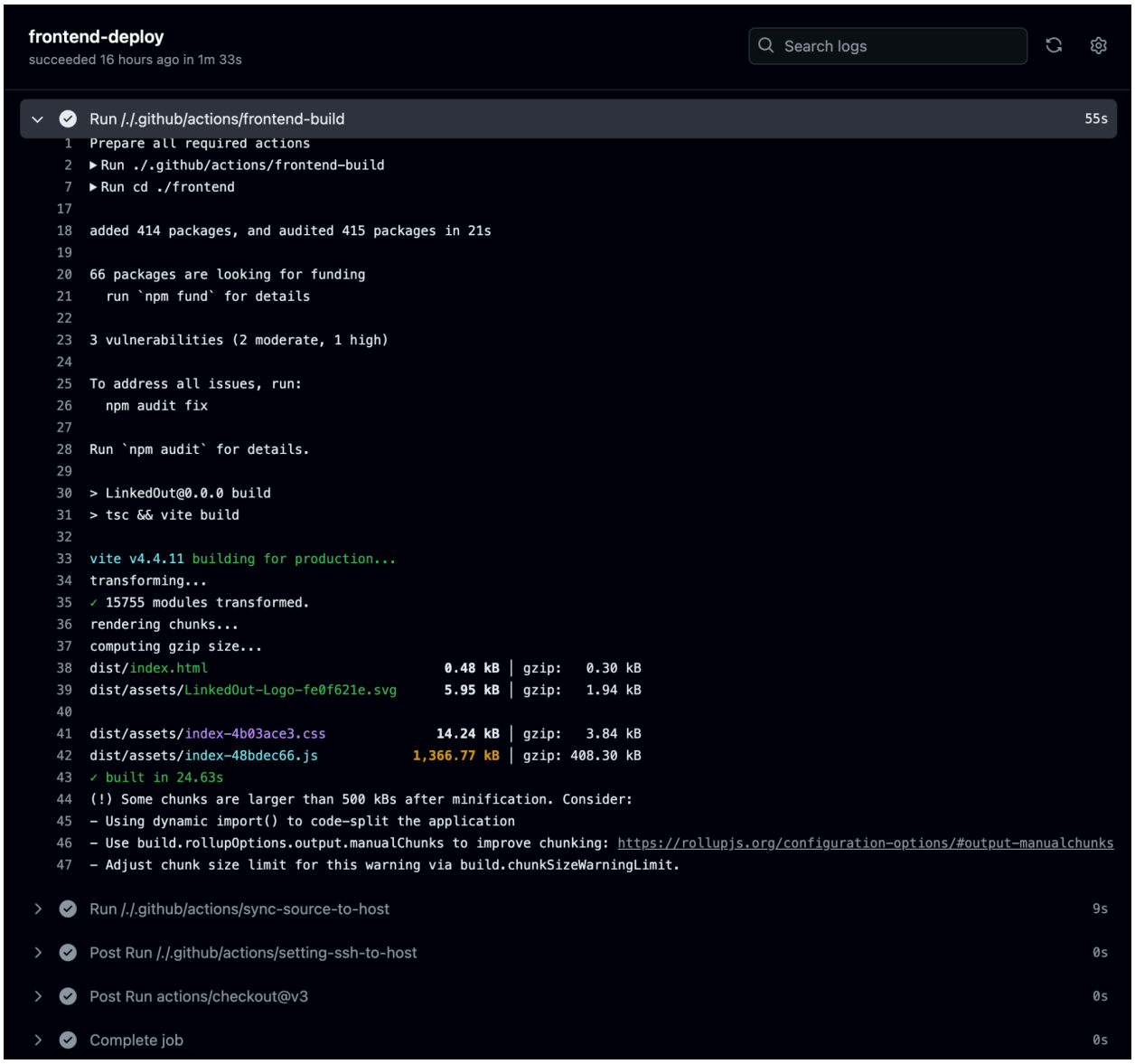


Figure 9.3: frontend-deploy jobs in Github Action

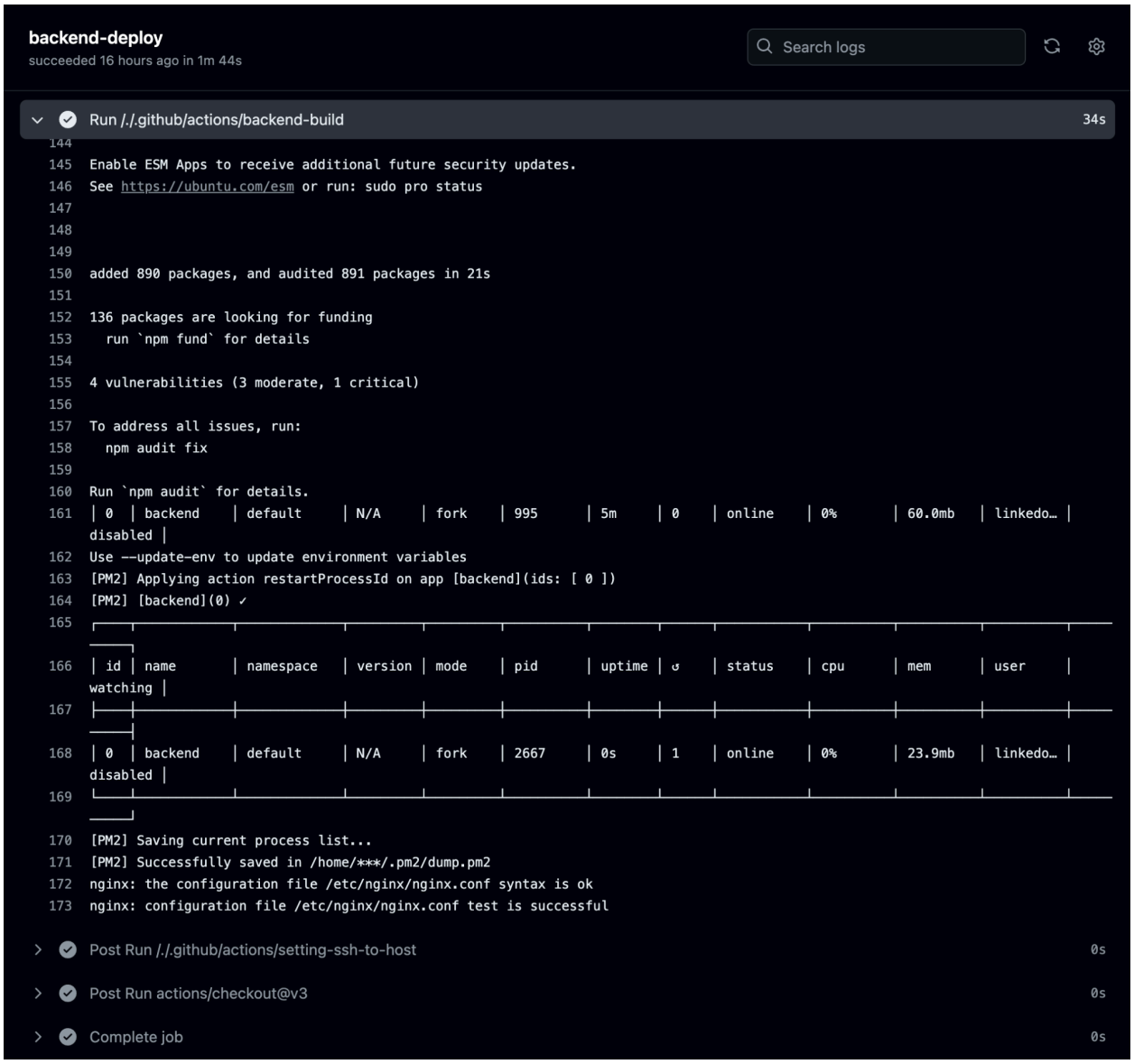


Figure 9.4: backend-deploy jobs in Github Action

9.3. Domain Routing and Hosting

- Domain name: feedme.io.vn
- Subdomain name: linkedout-hcmut
- DNS records:

Host	Type	Value	TTL
linkedout-hcmut	A	<public ip address of frontend VM>	3600

Table 9.1: DNS record

To give our server a publicly accessible address, we procured the domain name "feedme.io.vn" from a domain registrar. This domain name acts as a user-friendly identifier for our server on the internet, replacing the use of a less memorable numerical IP address. Since we require access to a specific service hosted on the server, we further established a subdomain named "linkedout-hcmut" within the main domain. This subdomain allows for granular routing within the server's infrastructure, potentially directing users to a specific application or service. Finally, we leveraged the Domain Name System (DNS) by creating an A record. This record associates the subdomain "linkedout-hcmut" with the public IP address of our Virtual Machine. By querying the DNS, users' web browsers can translate the subdomain name into the corresponding IP address, enabling them to reach the desired service hosted on our server.

10. Conclusion

10.1. LinkedOut - the final result

Source code repository: https://github.com/TravisMai/LinkedOut

LinkedOut url: https://linkedout-hcmut.feedme.io.vn/

Test coverage report: https://travismai.github.io/LinkedOut/lcov-report

10.2. Summary of Achievements

10.2.1. Regarding Topic Research

Regarding topic research, the project has undertaken a comprehensive investigation into the current internship processes and systems, pinpointing key inefficiencies and potential areas for enhancement. This included an in-depth examination of the recruitment processes for both students and companies, as well as the management procedures employed by faculty staff. Additionally, the project highlighted specific inefficiencies within the existing internship frameworks. A detailed analysis of existing internship platforms was also conducted, encompassing the Faculty of Computer Science and Engineering's internship website, the University of Economics HCMC (UEH) Department of Student Affairs (DSA) Career site, and LinkedIn, the social networking site for the business community. Through this meticulous research, the project identified significant gaps and opportunities within the current internship ecosystem. These findings lay the groundwork for developing a more efficient and user-friendly system aimed at better connecting students, faculty, and businesses, thereby enhancing the overall internship experience for all involved parties.

10.2.2.Regarding the Model and Technologies Used

The team has precisely and consistently defined the system's concepts, establishing a solid foundation for its development. Furthermore, comprehensive research has been conducted into various system architectures, models, and the technologies planned for implementation. This thorough groundwork ensures a more efficient and streamlined development process.

10.2.3. Regarding System Analysis and Design

In establishing the system architecture, the team meticulously outlined the key components and interactions within the system, ensuring a coherent and scalable design. This involved identifying the various modules and their relationships, as well as defining the communication protocols and data flow. Furthermore, the design of the data structures involved careful consideration of the information storage and retrieval requirements, ensuring efficiency and reliability in handling data throughout the system. Similarly, the interface design focused on creating intuitive and user-friendly interactions, enhancing the overall usability and accessibility of the system for all stakeholders. Overall, these foundational elements provide a solid framework for the successful implementation of the system's functionalities and pave the way for seamless integration of features and future enhancements.

10.2.4. Regarding the System Implementation Process

The team has successfully implemented a system that meets the initial requirements and functionalities outlined in the project proposal, while also applying the knowledge and understanding acquired throughout the course of this endeavor. However, due to time constraints, the system has not yet been fully refined to its optimal state. Despite this limitation, the team has focused on implementing the core features and functionalities, ensuring that essential aspects of the system are operational and align with the project objectives. Moving forward, there is potential for further refinement and enhancement of the system to fully realize its intended capabilities and deliverables.

10.3. Advantage

Through the implementation and testing process, the LinkedOut system has demonstrated several notable advantages:

- The system effectively fulfills the fundamental requirements of providing a job search website and managing student internship applications. It serves as a platform where businesses can post job openings and discover potential candidates, thereby facilitating efficient recruitment processes.
- Leveraging modern technologies, the system is built to be scalable and adaptable, allowing for seamless development and future enhancements.
- It simplifies the management of student internship outcomes, providing administrators with intuitive tools to oversee and track student progress effectively.

Overall, these strengths underscore the system's capability to meet its objectives and provide a valuable resource for both students and businesses alike in the internship ecosystem.

10.4. Disadvantage

Aside from its strengths, the LinkedOut system still has several drawbacks and limitations:

- The system currently lacks advanced algorithms to recommend suitable candidates for job positions, potentially hindering efficient matching between employers and potential hires.
- Interface optimization across various screen sizes remains incomplete, posing usability challenges for users accessing the platform on different devices.
- Limited support for multiple faculties is a current constraint, with the development primarily focused on the Computer Science and Engineering (CSE) internship system. Efforts are underway to expand support for additional faculties in the future.

10.5. Future improvement and implementation

To enhance the interaction level and refine the article recommendation system, we set the following objectives:

- Optimize candidate recommendation algorithm: Develop and deploy a smarter algorithm to suggest suitable candidates for job positions. Utilize machine learning and data mining techniques to improve the matchmaking between businesses and candidates.
- Improve user experience: Incorporate user feedback and conduct surveys to better understand user needs and preferences. Based on this feedback, we will adjust the interface and features of the system to optimize the user experience.
- Support multi-screen: Integrate responsive and adaptive methods to adjust the interface and user experience across various types of devices and screen sizes.

This includes building flexible and mobile-friendly interfaces for smartphones, tablets, and devices with different screen sizes.

- Leverage Azure's hosting infrastructure [26]: using a more efficient management design, Hub-Spoke Topology. This approach will enhance scalability and security, enabling centralized resource management and streamlined network communication. Adopting this topology ensures our systems can handle increased workloads and maintain high security standards, supporting ongoing growth and innovation.

10.6. Workload

Member	Workload	Completion Rate
Mai Hữu Nghĩa	• Implement backend API • Set up Azure cloud service • Set up NGINX server for deployment • Set up project into Virtual Machine • Set up CI/CD with Github Action • Unit test for backend and deploy test coverage • Database Design • Writing report	100%
Trần Trí Đạt	• Set up Azure cloud service • Azure cloud service management • Set up NGINX server for deployment • Database Design • Design application's UI • Implement application's UI • Manual testing for frontend • Deploy cloud service • Writing Report	100%

Table 10.1: Team member and workload

10.7. Weekly process

Week	Workload
1, 2	• System Description Section • Research Similar Systems, Analyze Their Strengths and Weaknesses • Identify Functional and Non-Functional Requirements for the System
3, 4	• Draw use case diagrams, provide detailed descriptions for each use case using a description table. • Create activity, sequence diagrams.
5, 6	• System Design Section • Database Design • User Interface (UI) Design
7, 8	• Identify and research the necessary technologies and techniques for implementing the system.
9, 10, 11, 12, 13, 14	• Install and prepare for the implementation. • Implement backend, frontend, database • Deploy some cloud services
15, 16, 17, 18	• Implement backend API • Implement user interface • Deploy Azure Blob Storage • Implement CI/CD for checking error on pull request
19, 20, 21	• API testing by Postman • Implement user interface • Ensure user experience • Deploy Azure Cache for Redis
22, 23, 24	• Frontend and backend integration • Checking compatibility and fixing bug • Modify some database entities to meet further requirement
25, 26, 27	• Jest unit testing for backend logic code • Manual testing for user interface • Implement CI/CD for test coverage

28, 29	• Setting environment and host server on Azure Virtual Machine • Deploy Azure Database for PostgreSQL • Deploy project on Azure Virtual Machine • Implement CI/CD Automatic Deployment with Github Action
30	• Finalize and re-check • Update report

Table 10.2: Weekly process

11. Bibliography

1. Hamed Moayeri. *Why and when to use Redis?*. Available at: https://www.linkedin.com/pulse/why-when-use-redis-hamed-moayeri/, 2019. (Accessed: 17/11/2023).
2. Ashish Kumar. *Optimizing performance in nestjs applications.* Available at: https://astconsulting.in/java-script/nodejs/nestjs/optimizing-performance-in-nestjs-applications/, 2023. (Accessed: 17/11/2023).
3. ReactJS Documentation. *Using TypeScript – React.* Available at: https://react.dev/learn/typescript. (Accessed: 25/11/2023).
4. ReactJS Documentation. *Thinking in React.* Available at: https://react.dev/learn/thinking-in-react. (Accessed: 25/11/2023).
5. MaterialUI Documentation. *Material UI - Overview.* Available at: https://mui.com/material-ui/getting-started/. (Accessed: 25/11/2023).
6. MaterialUI Documentation. *Material UI: React components based on Material Design.* Available at: https://mui.com/material-ui/. (Accessed: 25/11/2023).
7. Zustand Documentation. *pmndrs/zustand: Bear necessities for state management in React.* Available at: https://github.com/pmndrs/zustand. (Accessed: 23/11/2023)
8. Shruti Apte. *Zustand or Redux, which one is the best choice?*. Available at: https://blog.saeloun.com/2023/09/13/react-Zustand-vs-Redux/, 2023. (Accessed: 18/11/2023).
9. Axios Documentation. *Getting Started | Axios Docs* Available at: https://axios-http.com/docs/intro. (Accessed: 25/11/2023).
10. Faraz Kelhini. *Axios vs. fetch(): Which is best for making HTTP requests?*. Available at: https://blog.logrocket.com/axios-vs-fetch-best-http-requests/, 2022. (Accessed: 25/11/2023).

11. TailwindCSS Documentation. *Get started with Tailwind CSS.* https://tailwindcss.com/docs/installation. (Accessed: 15/11/2023).
12. TailwindCSS Documentation. *Install Tailwind CSS with Create React App.* https://tailwindcss.com/docs/guides/create-react-app. (Accessed: 16/11/2023).
13. NestJS Documentation. *Documentation | NestJS - A progressive Node.js framework.* Available at: https://docs.nestjs.com/. (Accessed: 15/11/2023).
14. NestJS Documentation. *SQL (TypeORM) | NestJS - A progressive Node.js framework.* Available at: https://docs.nestjs.com/recipes/sql-typeorm. (Accessed: 24/11/2023).
15. PostgreSQL Documentation. *PostgreSQL: Documentation: 16: 1. What Is PostgreSQL?*. Available at: https://www.postgresql.org/docs/current/intro-whatis.html. (Accessed: 24/11/2023).
16. Viblo. *Sự khác biệt giữa Sql và NoSql.* Available at: https://viblo.asia/p/nhung-diem-khac-biet-giua-sql-va-nosql-gGJ59b4rKX2/, 2022. (Accessed: 24/11/2023).
17. Benjamin Anderson, Brad Nicholson. *SQL vs. NoSQL Databases: What's the Difference?*. Available at: https://www.ibm.com/blog/sql-vs-nosql/, 2022. (Accessed: 24/11/2023).
18. Microsoft Documentation (03/01/2023) *Quickstart: Use Azure Cache for Redis in Node.js*. Available at: https://learn.microsoft.com/en-us/azure/azure-cache-for-redis/cache-nodejs-get-started (Accessed: 11/12/2023).
19. Microsoft Documentation (01/31/2023). *Quickstart: Azure Blob storage library - JavaScript*. Available at: https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-nodejs?tabs=managed-identity%2Croles-azure-portal%2Csign-in-azure-cli (Accessed: 11/12/2023).

20. Khoa Bui. *Cùng mình tìm hiểu về Json Web Token (JWT).* Available at: https://viblo.asia/p/cung-minh-tim-hieu-ve-json-web-token-jwt-Rk74axvAVeO, 2023. (Accessed: 18/11/2023).
21. Microsoft Documentation (03/01/2024) *Virtual machines in Azure*. Available at: https://learn.microsoft.com/en-us/azure/virtual-machines/ (Accessed: 02/05/2024).
22. Microsoft Documentation () *Azure Database for PostgreSQL - Flexible Server documentation*. Available at: https://learn.microsoft.com/en-us/azure/postgresql/ (Accessed: 10/05/2024)
23. Microsoft Documentation (27/10/2023) *Network security groups*. Available at: https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview (Accessed: 10/05/2024)
24. Microsoft Documentation (22/06/2023) *Use private endpoints for Azure Storage*. Available at: https://learn.microsoft.com/en-us/azure/storage/common/storage-private-endpoints (Accessed: 09/05/2024)
25. Microsoft Documentation (08/05/2024) *PostgreSQL extensions in Azure Database for PostgreSQL - Flexible Server*. Available at: https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/concepts-extensions (Accessed: 11/05/2024)
26. Microsoft Documentation (06/03/2024). *Hub-spoke network topology in Azure*. Available at: https://learn.microsoft.com/en-us/azure/architecture/networking/architecture/hub-spoke?tabs=cli. (Accessed: 28/04/2024)