

Earn It - International Student Job Platform

Design Project: Design Report



Authors:

Dimitar Popov
Wybe Pieterse
Pranav Chobar
Yordan Tsintsov
Stefan Ilich

Supervisor:

Luís Ferreira Pires

Abstract

This report is a part of the “Design Project” course at the University of Twente. It describes the design process behind the “Earn It” international student job platform. The purpose of this project is to provide easier access to jobs for international students in the Netherlands. It is a collaborative effort with University of Twente students Khattab Raouf and Faisal Nizamudeen. The reason behind Earn Its creation is the struggle that international students face when trying to find employment as companies do not want to waste time and resources on applying for a work visa. This project has found a way to avoid this problem by making use of Self-Employment permits. The report describes all the phases of the project lifecycle - from getting requirements to elaborating and discussing design choices.

Table of Contents

Table of Contents	2
Chapter 1 Introduction	5
Chapter 2 Domain Analysis	7
2.1 Domain introduction	7
2.2 General Knowledge of the Domain and Current Solutions	7
2.3 Software Environment	8
2.4 Stakeholders	8
2.4.1 Students	8
2.4.2 Companies	8
2.4.3 Client	8
2.4.4 Development Team	8
Chapter 3 Requirements Specification	9
3.1 Project Management	9
3.2 Requirement Formulation	9
3.3 Requirement Prioritization	10
3.4 User Requirements	10
3.4.1 Must haves	10
3.4.2 Should haves	11
3.4.3 Could haves	13
3.5 System Requirements	13
Chapter 4 Architectural Design	14
4.1 Preliminary Design Choices	14
4.1.1 Front end	14
4.1.1.1 Vue.JS	14
4.1.2 Back end	15
4.1.2.1 Spring Boot	15
4.1.2.2 PostgreSQL	16
4.1.3 Communication	17
4.1.4 Firebase	18
4.1.4.1 Firebase Storage	18
4.1.4.2 Firebase Authentication	18
4.1.4.2 Firebase Costs	18
4.2 System Overview	19
4.3.1 Home Page	20

4.3.2 Registration Page	20
4.3.3 Login Page	20
4.3.4 Student Dashboard page	20
4.3.5 Student Applications	20
4.3.6 Company Dashboard page	21
4.3.7 Post Vacancy page	21
4.3.8 Profile Page	21
4.3.9 Admin Dashboard page	21
Chapter 5 Detailed Design	21
5.1 System Description	22
5.1.1 Schema Design	22
5.1.2 Relationship between Tables	23
5.1.3 Keys	24
5.1.4 Routing	24
5.1.5 Token Authorisation	25
5.1.6 Spring Security Architecture	27
5.1.7 Back End Architecture	29
5.2 Design Choices	30
5.2.1 Page Navigation	30
5.2.2 User registration	30
5.2.3 Vacancy lifetime	30
5.2.4 Student dashboard	31
5.2.5 Company details	31
Chapter 6 Testing	32
6.1 Test Approach	32
6.1.1 End-to-end Integrated Tests	32
6.2 Technology	32
6.3 Test Cases	33
Chapter 7 Future Work	34
7.1 Features	34
7.2.1 Reviewing	34
7.2.2 Filtering	34
7.2.3 Notification	34
7.2.4 Login with third-party services	34
7.2 OpenAPI Specification	34

Chapter 8 Evaluation	36
8.1 Planning	36
8.2 Responsibilities	37
8.3 Team Evaluation	38
8.4 Conclusion	38
References	39
Appendices	40
Appendix A	41
Appendix B	47

Chapter 1 | Introduction

Finding a job has always been an issue for students all over the world and it is especially harder for those that study abroad. Studying abroad has existed for centuries and it has always had the same issues throughout the years when concerning searching for work. Foreign students usually do not have any professional contacts in the country in which they study, and they must further face additional barriers such as language and cultural differences. Furthermore, they are required to have a work visa or some other type of permit to find a job. This leads to companies hiring mostly native students as it is an easier and less-expensive process. These issues can be mostly seen in countries with many foreign students such as the US, the Netherlands, the UK, and Germany.

The problem of finding work permits has been especially painful for the Netherlands. Over the past 16 years, the Netherlands has been growing rapidly in popularity for international students. For the 2021/22 academic year, 115 thousand students from abroad were enrolled in higher education which is 3.5 times more than were in 2005/06, when the enrolled students were thirty-three thousand [\[1\]](#). A vast majority of these people are non-EU students, which means that for them to land a job, they must first be granted a work visa or some other similar type of permit. This creates a significant issue as EU and Dutch students would be preferred over international ones. Moreover, this leads to foreign students having fewer opportunities in a career aspect and causing unnecessary financial difficulties for them.

For these reasons, University of Twente international students Khattab Raouf(Earn It Chief Executive Officer) and Faisal Nizamudeen(Earn It Chief Technical Officer) produced the idea of Earn It - a start-up from the Netherlands that aims to connect international students with jobs in their field of study. Earn It makes use of self-employment permits to avoid the annoying problem of requiring a work visa to start a job. By using Earn It companies can flexibly and efficiently hire international students without concerning themselves with the paperwork. At the same time, students have a fast way of access to a list of positions related to their field of study for which they can enrol.

The start-up has several goals and motivations. One of the goals is to help increase the chances of international students finding a job and make the overall process easier and faster. Another reason is that this project also allows companies to hire foreign people and thus increase the pool of candidates for job vacancies which can help find an appropriate person for the requested job. Moreover, by diversifying their workforce, companies could improve their efficiency and productivity as studies suggest [\[2\]](#).

The report goes through the whole project lifecycle process. Chapter 2 is an analysis of the system domain. Chapter 3 is a discussion of the system proposal and how the system requirements were decided. Chapter 4 is a thorough analysis of the requirements and specifications of the whole platform including must, should and could haves. Chapter 5 goes over the architectural design of the system - what frameworks, libraries and other types of technologies were used for the project creation and why were they chosen for the system. Chapter 6 covers the project design and how the technologies mentioned in chapter 5 were incorporated. The testing aspect of the project is focused on in chapter 7. It also discusses the testing approach, technologies used and the test results. In chapter 8 possible future improvements and work are covered. Furthermore, the project integration is discussed. Lastly, Chapter 9 gives an evaluation of the project lifecycle by discussing the project planning and results. A team evaluation is also provided. Additionally, the final chapter contains a conclusion of the overall work process.

Chapter 2 | Domain Analysis

Domain analysis is a process in which information used in developing software systems is identified, captured, and organised so that it can be made reusable for the creation of any future systems [3]. This chapter discusses the domain of the system and its stakeholders. All essential information is documented and the context in which the software system is built is thoroughly explained. This was done to ensure that the development was made easier and reusability and future development were guaranteed to be more plausible.

2.1 Domain introduction

The domain in which this system will be applied concerns the hardships that students in the Netherlands face when looking for jobs related to their field of study and in particular, the adversity faced by international non-EU students. With international students constantly increasing and non-EU students having difficulty obtaining work visas, job opportunities are becoming fewer and fewer. This problem led to the birth of Earn It - a platform where students can search for jobs in their field of study and apply for work vacancies. The purpose of Earn It is to support students in finding a job to gain experience within their field of study and support companies in finding suitable employees.

2.2 General Knowledge of the Domain and Current Solutions

The Netherlands requires non-EU residents to have a work visa to be able to apply for jobs. To avoid the paperwork and administrative chores, non-EU students can use self-employment permits so that companies can be more prone to hire them. However, current job-seeking platforms do not take that logistical process into consideration and students are treated as any other potential employee. Furthermore, a platform that is made specifically for students nationwide is yet to arise. Thus, a platform that utilises self-employment permits should be designed to help students find a job while simplifying the job-seeking process for non-EU students. Moreover, to avoid just shifting the disbalance in the opposite direction, such a platform should still accommodate all other types of students. However, the legality of the use of such self-employment permits could pose a problem to the client but is outside of the scope of the development team in this module. The team's objective is to provide the client with a working system that fulfils their requirements and is not concerned, nor responsible for any legal issues regarding the future of this project.

2.3 Software Environment

Before the project group had a meeting with the company Earn It, some specifications were published in the project proposal document. The proposal document contained the key aspects of the system and its technical description. Those aspects were expanded upon in a meeting with Earn It and are shown in the requirements section of this report. In the technical description, it is stated that the technical domain that this system would be functioning is flexible. This meant that we were free to decide the frameworks we would use for the front end and back end. The only requirement that Earn It had was that the front end and back end would communicate with REST API calls. We decided to use modern technologies that allow for efficient and fast implementation of the system while enabling easy scalability in the future.

2.4 Stakeholders

2.4.1 Students

Students will use the platform to search and apply for jobs from vacancies posted by companies. They can upload their CV, skills, and education on the platform so that companies can check if they meet the necessary criteria for the vacancy they are applying to.

2.4.2 Companies

The companies will use the platform to post vacancies for jobs. They should be able to accept and deny applications from students for the offered jobs.

2.4.3 Client

The client is the initiator of the project. In this case, it is the start-up organisation Earn it. The purpose of this organisation is to provide students with an uncomplicated way to apply for jobs and to help companies to find student employees more efficiently. The client gives input on how to design the system. Earn It gives feedback and ideas on how to change/improve the final product.

2.4.4 Development Team

These are the people that implement the software system. They provide a functional online job platform for students as an end product. The team actively communicates with the client to get feedback on the progress of the system implementation and to check if styling changes are necessary, whether new features should be added, or whether the implemented

Chapter 3 | Requirements Specification

3.1 Project Management

To develop this project the team had access to weekly meetings with the client to evaluate the system's progress, stay within the scope and determine priority. To accommodate for potential weekly changes, it was chosen to use Agile practices to manage the project. As the meetings with the client are weekly, it was chosen to go for a sprint length of 1 week so that for each meeting there would be measurable progress. In practice, this meant that the entire team would have meetings twice a week to discuss processes and the sub-teams would have meetings of their own, so including the meeting with the client and supervisor, the team had about 3 meetings a week. The sub-teams that were mentioned earlier are the front-end team and the back-end team, it was decided to split these up so that people could specialise and focus on their experience and qualities instead of doing all facets. To keep the front end and back end compatible with each other the project started by designing a database and having a protocol document where expected behaviour for both sides would be specified.

3.2 Requirement Formulation

For establishing the requirements of the application there was extensive communication with the client to define the scope of the project and what was expected. By specifically communicating with the client about the requirements to have the developers on the same line it resulted in an accurate scope of the project with detailed requirements to not disappoint the client nor make the developer team work unnecessarily. To achieve these detailed requirements specific guidelines were set, these were based on the SMART guidelines [6] often used in project management. The guidelines are specified below:

- Specific
 - Each requirement needs a stakeholder and a detailed description
- Measurable
 - Each requirement needs to be able to be tested
- Attainable
 - Each requirement needs to be possible to implement within the time restriction
- Relevant
 - Each requirement needs to be within the scope of the application

3.3 Requirement Prioritization

During these meetings with the client the requirements were also prioritised, this was done using “*must haves*”, “*could haves*” and “*should haves*”. This tells the project team what the importance is to implement a requirement, *must-haves* are requirements that are needed for the project to reach a minimum viable product (MVP) state and are the most important. This is then followed by “*could haves*” which are important but not essential for the functioning of the application and “*should haves*” which are the least important requirements.

3.4 User Requirements

In this chapter all the requirements for each stakeholder have been specified combined with their implementation details, all *must-have* requirements have been implemented and have no details for that reason.

3.4.1 Must haves

1. A student must be able to register to the platform with a profile containing their personal information and extra details
2. A student must be able to upload their CV
3. A student must be able to view and update their profile
4. A student must be able to log in and register with their email and password
5. A student must be able to see vacancies that they can apply to
6. A student must be able to apply for a vacancy posted by a company
7. A student must be able to see vacancies that they have applied for
8. A student must be able to see the application status of vacancies they have applied for
9. A company representative must be able to register to the platform with a profile containing their company details and a contact person
10. A company representative must be able to view and update their profile

11. A company representative must be able to log in and register with their email and password
12. A company representative must be able to post vacancies
13. A company representative must be able to see students who have applied for a vacancy
14. A company representative must be able to accept and reject students who applied for a vacancy
15. A company representative must be able to download the CV of a student who has applied for their vacancy
16. A company representative must be able to view all the vacancies they have posted in the past
17. A visitor to the platform must be able to view a homepage where they can read information about the platform and register
18. An Earn It representative must be able to log in as an administrator
19. An administrator must be able to view how many vacancies have been posted
20. An administrator should be able to see how many students and companies are registered on the platform

3.4.2 Should haves

1. A company representative should be able to upload their logo
Implemented
2. A student should be able to retract their application to a company
Not implemented, could not be implemented within the time restriction
3. A student should be able to fill in the hours they have worked for a company
Implemented
4. A company representative should be able to view the hours a student has worked for the company

Implemented

5. A company representative or student should be able to export invoices based on the hours this student has worked for this company
Only students can export their invoices, companies cannot
6. A student should be able to filter vacancies based on their study
Not implemented, could not be implemented within the time restriction, specified in chapter 7
7. A student should be able to rate a company they have worked for with a 1–5-star rating
Not implemented, could not be implemented within the time restriction, specified in chapter 7
8. A company should be able to rate a student they have worked with a 1–5-star rating
Not implemented, could not be implemented within the time restriction, specified in chapter 7
9. A student should be able to sort vacancies based on ratings of a company
Not implemented, could not be implemented within the time restriction, specified in chapter 7
10. An admin should be able to delete a student, company and vacancy
User interface for this is implemented, but needs a little more work
11. An admin should be able to edit a student, company and vacancy
Implemented
12. A student or company representative should be able to delete their account
Implemented
13. A student or company representative should be able to update their password
Not implemented, could not be implemented within the time restriction
14. A company representative should be able to delete a vacancy from the company
Not implemented, could not be implemented within the time restriction

3.4.3 Could haves

1. A student or company representative could be able to add a review with more details to their star rating
Not implemented due to prioritisation of other requirements, specified in chapter 7
2. A student or company representative could be able to log in with Google or Microsoft accounts
Not implemented due to prioritisation of other requirements, specified in chapter 7

3.5 System Requirements

This chapter specifies specific requirements that the system must meet like security and specific technical requirements from the client.

1. No one must be able to tamper with the communication
Implemented, security is specified in chapters 4 and 5
2. The system front end must restrict pages to only be viewed by users with the correct permissions
Implemented, explained in chapter 5.1.4
3. The system front end must redirect users if pages with incorrect permissions are being accessed
Implemented, explained in chapter 5.1.4
4. The system must not store passwords in a way that is readable
Implemented, by using Firebase Authentication (chapter 4.1.4.2)
5. A user should only be allowed to perform actions in accordance with their permissions
Implemented and expanded upon in chapters 4 and 5
6. Front end and back end must communicate using a REST API
Implemented, chapter 4.1.3

Chapter 4 | Architectural Design

In the following chapter, it will be explained how all the components that are used in the application fit together, suit the design and fulfil the requirements. The diagram above shows how the system architecture of the application is defined. The application has a front end, and a back end, and uses Firebase and a REST API. It starts with the client being authenticated by Firebase; this allows the front end to send HTTP requests to the back end [7]. The REST API handles this communication. Then Firebase is used to authorise all requests the back end receives. After authorization, the server executes those requests by interacting with the database after which it returns a response to the client using the REST API. In addition, Firebase is also used for the storage of larger files like the CVs of students and logos of companies. Figure 1 shows a visual representation of the system design.

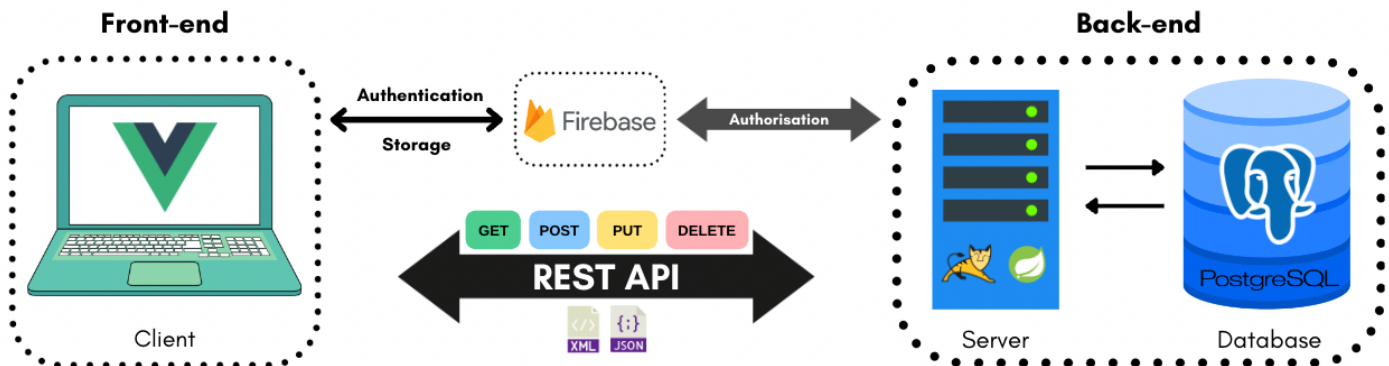


Fig. 1. System design

4.1 Preliminary Design Choices

4.1.1 Front end

4.1.1.1 Vue.JS

For the front end the team wanted to use a front-end development technology based on JavaScript (JS) because the developers had previous experience working with this scripting language thus the learning curve would not be as steep as other options would potentially be. Currently, the most popular technology is the React JS library [5] so choosing that would have

been a reasonable option but the team chose instead to go with the Vue.JS framework which is the fifth most popular JS front-end technology. A reason for this is that one of the team members already works daily with this framework, more technical advantages of why this technology was chosen are explained in chapter 5.

Requirements that the front end had to meet for the team were that it would be high-performance, simple and would be reusable. The high-performance requirement came from the fact that if this product ever grew to be big then slow performance could become an issue to grow this product to this scale, so using one of the fastest and lightweight frameworks around this requirement is met. The need for simplicity came from the fact that the development team is an inexperienced team consisting of students and as reaching an MVP was a high priority for the project a framework with a high learning curve could not be chosen. The reusable requirement works together with the other two requirements, having reusable components in the application reduces the complexity of the program as it is easier to keep the code readable and reduces the amount of repetitive coding which in turn makes the application better maintainable.

4.1.2 Back end

For the back end, we used Java Spring Boot for developing a REST API with an Apache Tomcat server. A PostgreSQL database was used to store information related to the system. The database was interacted with via Java Spring Data JPA. Google Firebase was used for authentication and authorisation purposes as well as storing files such as the student CV PDFs.

4.1.2.1 Spring Boot

It was decided to use Spring Boot for this project because of its many advantages over other solutions. Some of them are as follows:

- Ease of use
- Reduction of development time
- Potential for increased productivity in development
- Simplicity of scalability
- Minimal initialization of the project
- Minimal configuration - avoids a complex XML configuration
- Severe reduction of required boilerplate code
- Spring Data integration, which simplifies interaction with the database dramatically
- Spring Security/Firebase Authentication integration

- Maven integration
- Provides an embedded HTTP server
- The application can be simply run

Spring Boot utilises annotations for binding requests to controllers and handler methods in addition to binding HTTP request parameters to method arguments. This conceptually bridges the front end and the back end, resulting in much shorter, cleaner, simpler, and easier-to-understand source code, which consequently makes the project easier to collaborate on and scale. *Spring Initializer* generates a Spring Boot project by providing a graphical user interface that allows a user to specify the required Spring Boot version, the project metadata as well as all the dependencies the project needs, greatly simplifying the initialization of the project. The Maven integration and provision of an embedded Tomcat server greatly simplify the deployment process. The former also provides a simple way to manage the project, from building it to documenting it.

As previously mentioned, Spring Boot annotations massively reduce the amount of boilerplate code, which provides the ability to implement functionality much faster. However, this is not their only benefit - by automating a large portion of the work done by the server, Spring Boot leaves less room for error than alternatives like managing Tomcat directly, while providing the same amount of flexibility if needed. This leads to much less time fixing bugs and testing basic tasks such as connecting to the database or editing a single row from some table.

There were some risks in developing the project using a framework that was not used during the bachelor of Technical Computer Science at the University of Twente. Using an unfamiliar and new framework required developers to learn while working on the project itself which took more time and slowed down the work speed and efficiency. Furthermore, the team had to find its study materials as the Bachelor program does not offer any information on the technology that was used. However, the many advantages of the Spring framework benefited the workflow of the team immensely, while enabling the team to make an application that the client can expand simply and easily.

4.1.2.2 PostgreSQL

For storing application data, there were many choices for a relational database management system, such as MySQL, PostgreSQL, and MongoDB. When choosing, it was determined which one will fit the project requirements best, and support seamless scalability, while also not having a significant learning curve, allowing to implement the project without jeopardising finishing the project on time. In the end, the decision was that PostgreSQL is the most suitable one.

In comparison to MySQL, PostgreSQL supports more complex data types and allows objects to inherit properties. This property is especially useful since the project will use UUIDs (Universally Unique Identifiers) of 128 bits as unique IDs for every student, company, and account. UUIDs can be easily stored in PostgreSQL databases without any problems. While UUIDs are an available feature in MySQL as well, they cannot be directly assigned to a column as a datatype. First, the ID should be set as a sequence of 128 binary digits and then ran through a function that transforms it into 32 hexadecimal characters, which is the traditional format of a UUID [4]. This makes PostgreSQL preferable over MySQL as it can assign UUIDs directly as datatypes in a column which saves time and complexity. What's more, PostgreSQL's wider variety of data types ensures a much easier scalability process in case the project becomes more complicated in the future. As the project is very specific, a lot of particular data types might be needed for future functionalities, and using PostgreSQL is an excellent way of future-proofing the project for such functionalities.

Furthermore, PostgreSQL is an ideal choice for hosting a database as a company scales (enterprise scope) when compared with MySQL. PostgreSQL supports complex queries and frequent write operations, which MySQL does not. Moreover, PostgreSQL is designed for large database management and has great utilisation of indices and excellent support for complex queries, allowing custom optimization when the application becomes popular and gains a massive number of users.

Another massive advantage of PostgreSQL over its alternatives is that it is an object-relational programming language (ORDBMS). This is an excellent fit for Spring Boot, the Java framework of choice, as it allows for direct manipulation of the database using the framework and creates a bridge between Java's object-oriented programming and the database model.

Lastly, PostgreSQL offers high stability due to its complete compliance with ACID (Atomicity, Consistency, Isolation, Duration) properties. ACID properties ensure that no data is miscommunicated or lost during event failure [15].

4.1.3 Communication

In this section, the communication between the front end and back end will be further elaborated on. As mentioned in the introduction we have a REST API that serves as the communication link between our client and server. REST is an architectural style that defines a set of rules for creating web services. Those services allow the client to send requests to the back end when the user does specific actions like creating a new user, applying to a vacancy, or accepting a student for a vacancy. Below you can find the protocol document that shows all the possible requests the front end can make to the back end and the appropriate responses that will be given. The protocol document is shown in Appendix B.

4.1.4 Firebase

As can be read in the “*Requirements Analysis*” section in this document there were some security, login, and storage requirements for this application. Namely that requests are authenticated, passwords protected, and users can log in using third-party applications and upload a CV. As no developers of the team had experience building an application that could utilise storage while still being scalable the team did research into which solutions were suitable for this. The storage solution that was eventually chosen was Firebase Storage as it would offload the bandwidth load needed for uploading and downloading files to another service that would increase the scalability aspect of the application as the storage and bandwidth capacity would be able to grow as the application grows. As the documentation of Firebase was written extensively which made it easy to implement it was also chosen to use Firebase Authentication for the security of the application. The costs of these services will be discussed in a section below and were agreed upon with the client.

All these advantages together make Firebase cost-effective because by using an existing platform with good documentation the development time is reduced and reduction of bugs and errors this all adds to the scalability and maintainability of the application.

4.1.4.1 Firebase Storage

The front end of the application interacts with Firebase using the Firebase JS SDK which provides robust operations for uploading and downloading files. This means that no matter the network quality these operations restart where they are stopped if they are interrupted reducing the amount of bandwidth needed. This in combination with the scalability of Firebase storage and easy integration with Firebase Authentication to secure the files makes it a good fit for this application. So, in the application Firebase Storage is used to store the CV of students and logos from Companies.

4.1.4.2 Firebase Authentication

Along with Firebase Storage the application also makes use of Firebase Authentication. This has not only the security advantages for Firebase Storage, but also provides a way to quickly set up an expansive login and registration method. The application currently only features the ability to sign up and log in with a password and email but integrating services like Google and Microsoft Sign-in is little work by the Firebase JS SDK. By using this service the passwords are securely encrypted and stored on a different server meaning that the consequences of a leak are reduced. Furthermore, would it be easy to integrate multi-factor authentication in this application with Firebase as there is extensive support for OAuth 2.0 and OpenID connect.

4.1.4.2 Firebase Costs

Firebase is a “*pay as you go*” platform meaning that the platform only charges for what is used, the platform starts with a free tier called “spark plan” which supports 50k monthly active

users (MAU) and 5GB of storage. After that, you pay between \$0.0025 - \$0.0055 per MAU depending on the amount of MAUs the application has. For Firebase storage it costs \$0.026 per GB stored, \$0.12/per GB downloaded, \$0.05/10k upload operations and \$0.004/10k download operations. A quick sum for the costs is outlined below as per the current pricing of Firebase.

Students	Cost	Amount	Total Usage	Total Cost
Monthly Active Users	\$0.0055/MAU	10,000	-	\$55
Upload operations/MAU	\$0.05/10k	2	20,000	\$0.1
Download operations/MAU	\$0.004/10k	2	10,000	\$0.008
Download bandwidth cost	\$0.12/GB	2 * 5MB * 10,000	100GB	\$12
Storage used/MAU	\$0.026/GB	5MB	50GB	\$1.3
			Total:	\$68.408
Companies	Cost	Amount	Total usage	Total Cost
Monthly Active Users	\$0.0055/MAU	10,000	-	\$55
Upload operations/MAU	\$0.05/10k	1	10,000	\$0.1
Download operations/MAU	\$0.004/10k	25	10,000	\$0.4
Download bandwidth cost	\$0.12/GB	25 * 5MB * 10,000	1.25TB	\$1500
Storage used/MAU	\$0.026/GB	5MB	50GB	\$1.3
			Total:	\$1556.8

Fig.2. Firebase costs

This cost overview is based on the estimation that a student would upload and download his CV twice a month, for companies this estimation is set that they download 25 CVs a month. The CV and logo size estimation would be 5MB as that is what the limit of the application is set at and would be a reasonable assumption for a medium-sized pdf so the calculation would be the maximum cost per user and assumes that each MAU reaches those downloads.

Earn It expects that there would be around twenty students for each company on the platform meaning that if the “*spark plan*” from Firebase is exceeded the cost for one hundred students and five companies would be a total cost of \$16,25 of which \$15,56 is for the five companies.

4.3 System Overview

Our system is defined for three types of users: students, companies, and admins. The following sections will describe all the pages and features the system contains and you will be able to understand what each type of user can do.

4.3.1 Home Page

At first, users arrive at the home page. There they can read the benefits of signing up for the platform and register as a student or representative of the company. They can also decide to log in with their already existing account.

4.3.2 Registration Page

When a user enters this page, they have the option to either register as a student or a representative of a company. There is one registration page for both parties, but companies and students need to enter different data due to their different natures. For example, students are going to upload a CV for their profiles while on the other hand companies will be required to upload the logo of their company. For these pages, there are checks if the user entered the correct data in the fields meaning that a password needs to be at least eight characters long and a CV must be uploaded. For each registration field, there is such a check, when a user registers it is also checked if the email address is not yet registered.

4.3.3 Login Page

On this page, the user can log in with their email and password. When logging in the user will first be authenticated by Firebase, if Firebase returns a confirmation the user is logged into the back end with a Firebase token. The authentication is further described in Chapter 6 Detailed Design.

4.3.4 Student Dashboard page

Our landing page for students is the Student Dashboard Page. On this page, students can see all the vacancies posted by companies. Each vacancy contains the needed information to make a decision like company, location, hourly wage, and benefits. Through this page, students can apply for a vacancy and see it on their Student Applications page.

4.3.5 Student Applications

On this page, Students can see all vacancies they have applied for. Here they can see their status regarding this page (Pending, Accepted, Rejected). This status will be set to pending after they apply for a vacancy and then it can only be changed by a company depending on their decision on the application. Furthermore, after students get accepted, they will have two more features enabled. They will be able to enter their hours for the current month and send them to their employer for review. After that, they will be able to export their monthly invoice containing their calculated income.

4.3.6 Company Dashboard page

This is the landing page for companies. They will be able to review all their posted vacancies and for every vacancy, they are going to see who has applied and then accept or reject them. Furthermore, they can download the CVs of all of their applicants to review their education and job history so they can decide whether to accept or reject them. If they accept someone for a vacancy, he will not appear as employed for the vacancy and the company will be able to review the monthly hours he enters.

4.3.7 Post Vacancy page

If a company enters this page, they can post a new vacancy that will appear for all students, and then they will be able to see this vacancy and the applicants on the Company Dashboard page.

4.3.8 Profile Page

Both companies and students have a profile page where they can see all of their current data and edit it to their preferences. The page is different for companies and students. For example, students can download their CVs or upload new ones while companies cannot.

4.3.9 Admin Dashboard page

Finally, there is the Admin Dashboard page which is the only page admin users can access. This page consists of statistics regarding the application like the number of users for each category and all posted vacancies. The admin user is also allowed to change or delete students, companies, and vacancies.

Chapter 5 | Detailed Design

This chapter will describe our system in detail and it will discuss all the different design choices we came across throughout the project.

5.1 System Description

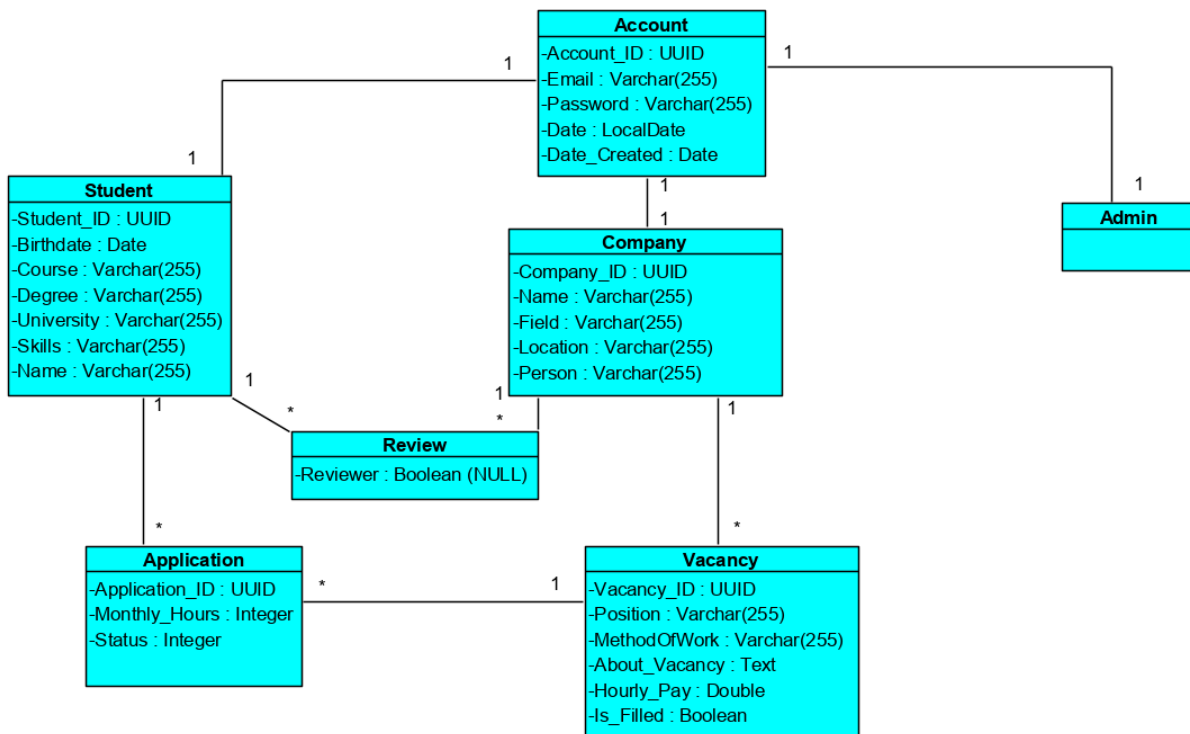


Fig.3. System class diagram

The class diagram above represents the database of the system. Each class in the diagram is a table in the database. The class diagram contains 7 classes, “Account”, “Student”, “Company”, “Admin”, “Review”, “Application”, and “Vacancy”. The class diagram and relationship between the classes are explained in detail below:

5.1.1 Schema Design

Tables for the database were designed by examining the functional requirements and scope while taking the simplicity of query implementations in mind. In the next few paragraphs, all

the tables that the project required will be listed. In the section after that, all the relationships between the tables will be listed.

The functional requirements clearly indicate that a user should be able to register as a student and company. Furthermore, a special admin user should be added to the database. Therefore, Student, Company and Admin tables were defined.

All types of users should have an account that they can log in to via their email and password, regardless of what type of user they are. Thus, to mitigate the need for email and password fields in the tables of all user types, an Account table was created. Each user (Student, Company, Admin) has one account. The relationship between the user and the account is represented in the class diagram with each user having a one-to-one relationship with the Account class.

The Account entity that the Student, Company and Admin have was used instead of a general User entity that the Student and Company would extend, to make the overall model and subsequent queries simpler.

Each company should be able to post multiple vacancies for their open job positions. Furthermore, each student user should be able to apply for a certain vacancy that is listed by a company. Thus, a Vacancy table was defined. This is indicated in the diagram with the Company table having a one-to-many relationship with the vacancy class.

In order to bridge the relationship between Student and Company tables, an application table was defined. As a student can post many applications for many vacancies, the Application table stores the ID of Student and Vacancy. In the class diagram, the relationship is indicated with a one-to-many relation between Student and Application. Furthermore, each vacancy can have many applications from different students, which is indicated by a many-to-one relation between the Application and Vacancy table.

Lastly, every company and student should be able to review each other. Therefore, a Review table that stores the IDs of the company and student was defined. To avoid using two tables, one for reviews from students to companies and one for reviews from companies to students, the Review table has a property that indicates the direction of the relationship. Each student and company can write many reviews. This relationship is indicated by the one-to-many relation between the student/company table with the review table.

5.1.2 Relationship between Tables

To clarify, these are all the relationships between the tables in the database:

- Each type of user (student, company and admin) can have only one account.
- One company can have many vacancies
- One student can have many applications (for different vacancies)
- One vacancy can have many applications (from different students)
- One student can review many companies
- One company can review many students

5.1.3 Keys

Primary keys are unique identifiers of each record in the database, whereas foreign keys are a link to properties of different tables. All of the above-mentioned relationships are formally defined using foreign keys. These are the indexes and foreign key constraints:

- Student - student_id (Primary Key), account_id (Foreign Key to Account).
- Company - company_id (Primary Key), account_id (Foreign Key to Account)
- Admin - account_id (Foreign Key to Account)
- Vacancy - vacancy_id (Primary Key), company_id (Foreign Key to Company)
- Application - application_id (Primary Key), student_id (Foreign key to Student), vacancy_id (Foreign key to Vacancy)
- Review - review_id (Primary Key), student_id (Foreign Key to Student), company_id (Foreign Key to Company)

5.1.4 Routing

As the JavaScript framework that was chosen for this project was Vue.JS it was opted to also include the router library that is officially supported by Vue.JS namely *Vue Router Library*. This library handles all the “routing” for the front-end application, this means that if a user needs to be forwarded from page A to page B this library is used to handle this functionality. This specific library was chosen as it has a simple integration with the Vue.JS framework meaning that the code stays clean because it shares the same design pattern. The function of this router is not only to get a user from page A to page B but also to make sure that a user has the permissions to do so and if not, they are properly redirected, as explained below. This chapter also contains a diagram where a visual representation of the routing is shown.

The implementation of the router is in `/source/router/index.js` here is where the router is initialised and a *beforeEach* is added which is a so-called global navigation guard which can redirect or cancel the navigation process. This specific guard is called every time navigation is

triggered and where it is checked if a user should be allowed with their user status, meaning company, administrator, or student, to go to that navigation. If the user is allowed then a confirmation is returned and the navigation will not be redirected or cancelled but is allowed to continue, if the user is not allowed to access the next navigation, then they are redirected to an appropriate page.

In this implementation the focus was on scalability and maintainability, meaning that if a new page were to be added it would require little effort and restrictions to this page could easily be added. This was achieved by adding permission fields to routes which can easily be used to filter users based on permissions. An example of this is “authRequired”, if this value is true then a user must be logged in to access the route, if it is undefined then it does not matter but if it is false then a logged-in user is not allowed to access the page. For example, the login page should only be accessed by a user who is not logged in.

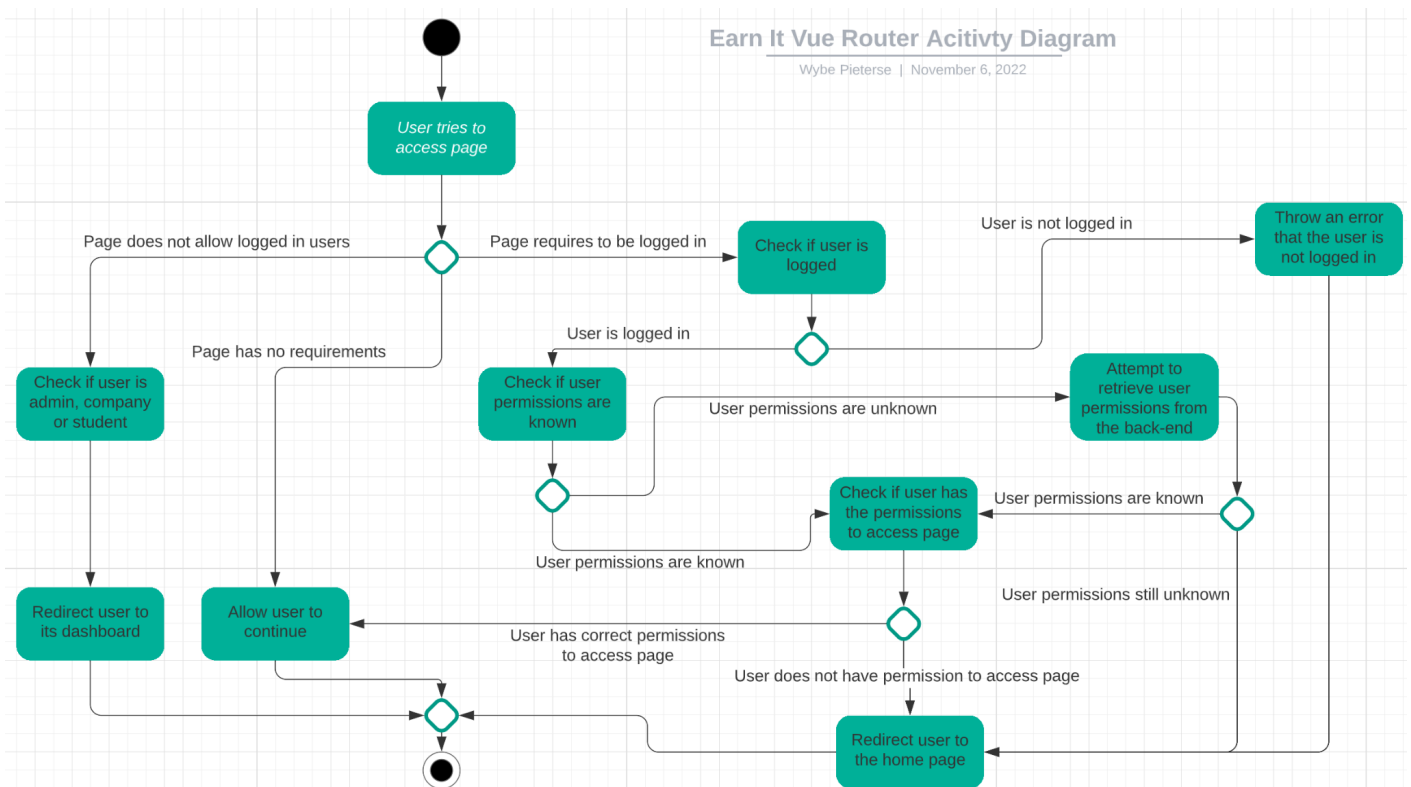


Fig.4. Routing activity diagram

5.1.5 Token Authorisation

Authorization is a process where the resources, services and functions have restricted access. In this case, students, companies, and administrators are in distinct roles which have different sets of permissions to go through the web application. The authorization of the requests was implemented using JWT (Java Web Token) requests API implemented in Spring Boot and Firebase.

According to [\[8\]](#), a Java web token (JWT) is described as a secured way of transmitting information in the form of a JSON object. The information in a JWT is trusted and secured since the token is kept secret using a secured hashing algorithm.

In the code, Google's Firebase ID token was used as the JWT for the web application. Firebase uses the RS256 algorithm for hashing tokens which ensures security. According to [\[9\]](#), RS256 is an asymmetric encryption algorithm that uses a private key to hash the token and the authenticity of the token can be verified using a public key.

If the user wants to access a resource, the user must send a request to the server with a token in the bearer schema for authorization. The server in the back end check's for the token in the authorization header of the request. The token is extracted from the header and sent to the server for further processing. Requests that don't contain the token are rejected.

Each token must go through a token filter where the tokens get verified. After verification, the request can proceed further.

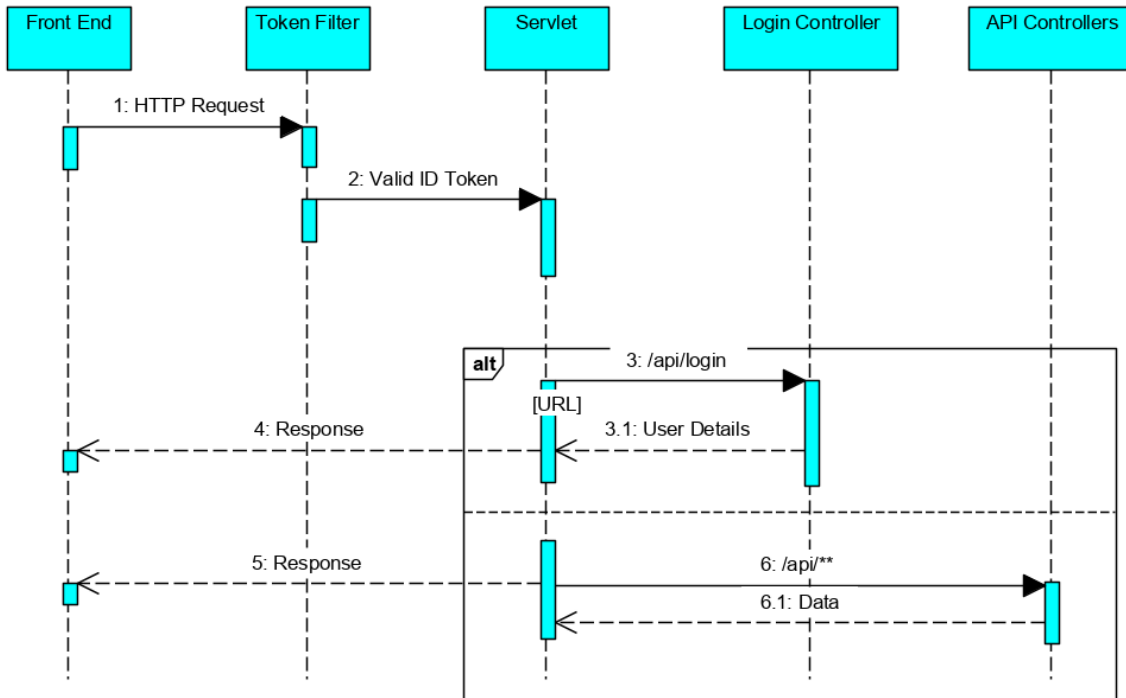


Fig 6. Token Authorization Diagram

The figure explains the authorization system discussed in the above sections. The figure contains 5 lifelines, “FrontEnd”, “TokenFilter”, “Servlet”, “Login Controller” and “API Controllers”.

The “FrontEnd” sends CRUD (Create, Read, Update, Delete) HTTP requests to the back end. First, the request goes through the token filter.

As explained in section 6.2.2, the request must contain a firebase id token in the bearer token part of the HTTP request. The token gets verified in the token filter. Requests containing an invalid token are rejected. If the token is valid then the request passes through the spring security filters and goes into the “Servlet”.

The “Servlet” is an apache tomcat java servlet that processes HTTP requests and manages communication between the front end and back end of a web server. The “Servlet” checks the URL of the request. The request containing the login URL (“/API/login”) is handled by the login controller. The login controller sends “user details” back to the servlet from which it gets sent back to the front end.

The front end requires “user details” to distinguish between the type of user (“Student”, “Company” or “Admin). Other valid requests are passed to the “API Controllers”. “API Controllers” are REST controllers in the spring boot framework which process the incoming

requests and transform them into a model and return a view [\[11\]](#). Each controller class in spring boot has multiple URLs mapped to them.

The “Servlet” checks the URL in the request and sends the request to an appropriate controller. The controller processes the request and sends back a response (containing data) to the servlet which gets passed on to the front end. If the servlet decides that the URL is invalid, the servlet sends back an error message to the front end.

5.1.6 Spring Security Architecture

Spring security architecture works in the form of filters. The request passes through a filter chain before reaching the servlet. Each filter in the chain can either process the request and pass it on to the next filter or terminate the request. Authorization of the incoming requests occurs in the filter chain. Spring boot uses an authorization filter to support the authorization of incoming HTTP requests [\[10\]](#). A token filter was designed which parses the token in the incoming HTTP request.

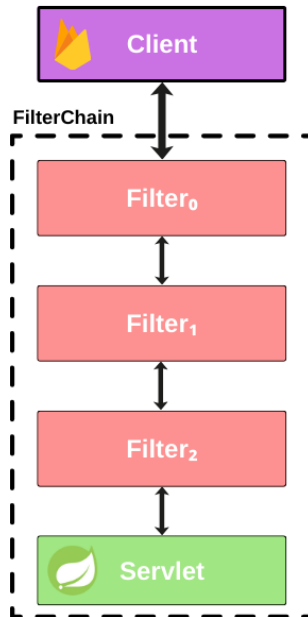


Fig. 5. Spring Security Filter Chain

A security configuration class was designed in Springboot to control the security measures in the application. The class is located in the security folder with the name “*SecurityConfig*”. The security configuration extends *WebSecurityConfigurerAdapter*.

WebSecurityConfigurerAdapter contains methods that allow customising HTTP security [12]. HTTP security allows configuring web-based security for specific HTTP requests. The extension of *WebSecurityConfigurerAdapter* class allows for the customization and configuration of *HttpSecurity*. This is done by overriding the configure method in the security configuration class.

CORS (Cross-origin resource sharing) is disabled to prevent requests originating from different origins. Disabling CORS only allows requests that originate from the host. Enabling CORS will give rise to vulnerabilities that can be exploited through cross-origin requests. Unauthorised requests are handled with the HTTP method of exception handling. Endpoints are configured to allow entry of CRUD and login requests only. Finally, the session is set to stateless to prevent creating any sessions (and attacks such as session fixation attacks) on the server.

5.1.7 Back End Architecture

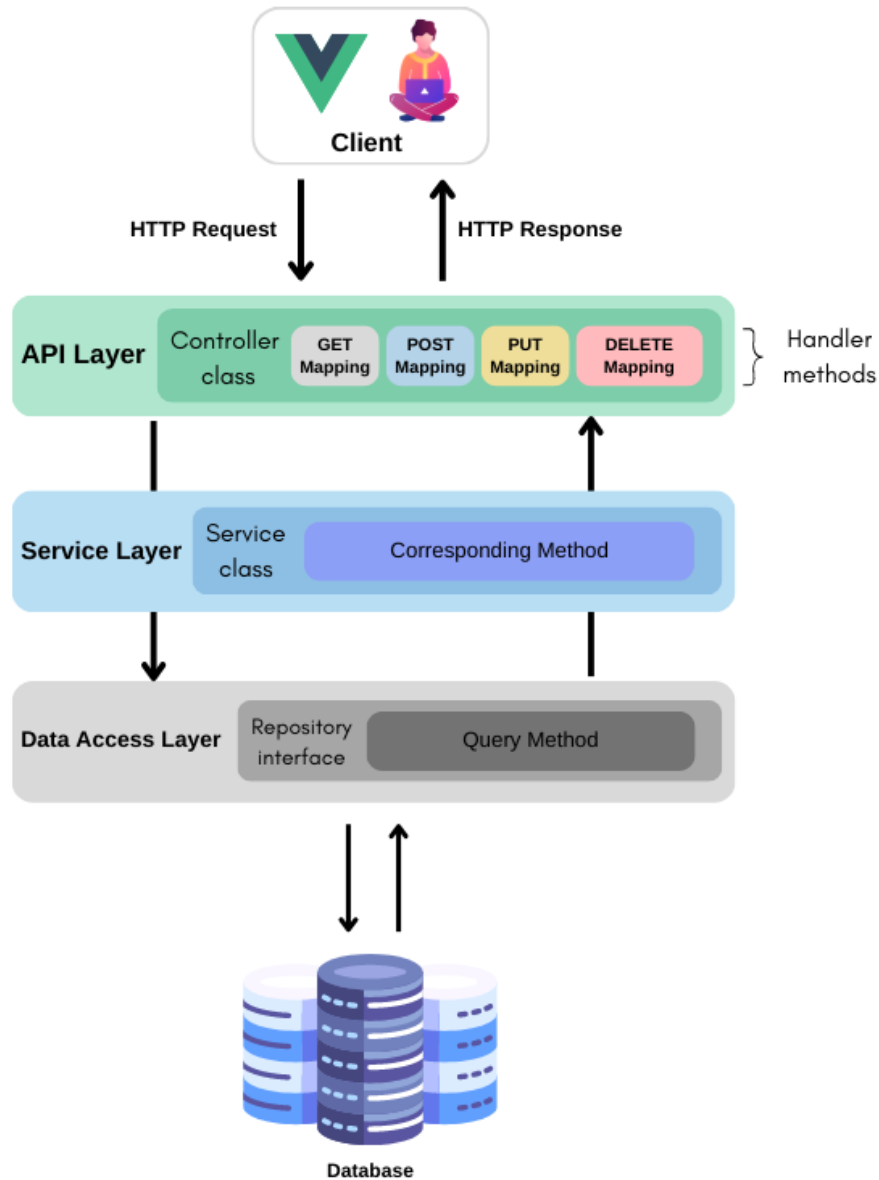


Fig. 7. Back end architecture

In this project, a RESTful application was created using the Spring Java-based open-source framework. Spring Boot was used to create a REST API to establish communication between the client and the Apache Tomcat server over HTTP. Once the client sends an HTTP request, the API is called from the back end, which calls the server. Via a controller class' handler methods, the server then performs all the business logic via services and returns the result to

the client. Direct communication with the database is established via repositories that the services use, which will further explore in the next section.

First off, the client sends an HTTP (GET/PUT/POST/DELETE) request to the server at a specified URL. In the Java Spring implementation, those requests are handled by Controller classes. Those classes utilise annotations[\[16\]](#) that map HTTP requests onto specific handler methods. For example, “@GetMapping(“/api/companies”)” is the annotation that will be used above with the implementation of the method that handles GET requests at that URL. In the method, all relevant information from the HTTP method is processed and passed to the corresponding method of the service class. The service class checks the information for errors and passes them to methods of the repository interface, which directly interacts with the database. The repository interface extends the JpaRepository interface from the Spring Data JPA library to use its wide range of default methods. For more specific queries, methods that use annotations containing JPQL queries can be implemented [\[17\]](#). A visual representation of the backend architecture can be found in Fig. 7.

5.2 Design Choices

5.2.1 Page Navigation

Our first design decision entails how the users would navigate between all pages. The first choice was to use a top navigation bar and the second was to use a sidebar. After discussing this with EarnIT we settled for a sidebar because it looked more professional and it can fit the entire EarnIT logo inside it while the top navigation bar cannot.

5.2.2 User registration

Our second design decision was about the registration of users. The options were to have two separate registration pages on the application or have one that can accommodate both the registration of companies and students. After having a meeting about this we decided that it would be best to have one page where users can quickly change between student and company registration instead of going to a different page.

5.2.3 Vacancy lifetime

Our third design decision concerns when a vacancy should disappear from the dashboard of all students. Our first option was to immediately remove all vacancies for which a student is accepted by a company. The other option we had was to keep those vacancies for a certain period of time before removing them. We decided to go with the second option because if a vacancy immediately disappears when a student is accepted, the other students that have not received an answer may be confused as to what happened with their application. By keeping

filled-in vacancies on the student dashboard, all applicants can see the result of their application

5.2.4 Student dashboard

Our fourth design decision takes into consideration whether are we going to show the basic information about each company on the Student Dashboard. The options were to display that information in the form of rows or cards. After discussing this with EarnIT and between ourselves we decided to use a card design for the dashboard because a card can also fit a company logo while a row would not have enough space.

5.2.5 Company details

Our final design choice is whether to show company details on the Student dashboard or create an entirely new page for that. In this case, we came to the conclusion that it would be preferable to show them on the same page because it is not a lot of information and the new would look rather empty. Also, students will be able to instantly see the details of a vacancy and apply for it instead of loading a new page.

Chapter 6 | Testing

This chapter discusses the testing plan for the front end and back end of the project and the results. It specifies the functionalities for which the tests were performed and explains the approach that was used.

Based on the testing results, some of the system designs were updated and bugs were identified and dealt with.

6.1 Test Approach

The project focuses on writing tests to achieve the maximum coverage of most functionalities in the application. Testing of the system was done using API Testing and Integration testing. API Testing focuses on the back end side of the application. Integration testing focuses on the application as a whole covering both the front end and back end of the application. The tests are described in the sections below.

6.1.1 End-to-end Integrated Tests

The system was tested using end-to-end automated testing. The end-to-end software testing method involves testing the application from start to finish. It mocks a real-life system user that accesses and uses the software. End-to-end testing helps identify bugs and problems before releasing software to users [\[13\]](#). In end-to-end testing, a completely integrated system is used, which also helps in checking the business logic.

6.2 Technology

Cypress is a JavaScript-based front-end testing tool. In this project, cypress is used as the testing framework because of its ease of use and numerous features. Cypress works directly with browsers, which makes testing fast and reliable. At first, the plan was to use Selenium but we decided against it because it was a lot more time-consuming and gave errors with the Vue.JS framework. Furthermore, Cypress provides step-by-step snapshots of commands executed, which proves to be extremely helpful when debugging. Lastly, unlike Selenium, Cypress does not require wait commands in test scripts, making it easier to run code [\[14\]](#).

6.3 Test Cases

The following test cases were used to examine the system:

- Login

The login page was tested using a valid email ID and password. A bearer ID token must be provided for the test to pass. The test passes if the user logs in successfully and the front end receives information about the user from the back end.

- Registration (Student/Company)

A simple registration test was made for both students and companies. The test consists of checking whether the registration for students and companies works correctly. The data for the form is prefilled. The automated test fills up the data on the registration page. The test passes if the user is registered.

- Posting and Applying for a Vacancy (Student/Company)

In this test case scenario, a test was created where a company logs in and creates a vacancy and posts it. Furthermore, the test consists of logging in as a student and applying for the vacancy posted by the company. The test will pass when the student successfully applies for the vacancy.

- Updating Personal Details (Student/Company)

In this test case scenario, a test was created to update personal information as a company or student. A test was created with new details for an existing user. The test log's in as the user and updates personal information with new details. The test will pass when the personal details of the user as updated.

- Administrator Functionality

In this test case scenario, a test case was created to check administration functionality and the administrator homepage. The test will pass when the administrator can successfully log in and view user profiles.

Chapter 7 | Future Work

7.1 Features

7.2.1 Reviewing

The first feature that can be added in the future is a review system between students and companies. Students should be able to leave a review with an explanation for the company they are working for which can help other students decide which company they should apply for. In addition, companies should be able to leave reviews with feedback from students they employed in order to help other companies with employment decisions.

7.2.2 Filtering

Another appropriate decision is for students to be able to filter and sort companies they see based on their preferences. For example, this will help them to only see companies in their region or ones that are searching for employees in their field of study. This will greatly help students in finding companies that are fitting for them.

7.2.3 Notification

Moreover, we believe a notification system should be installed which will help students and companies in having a much faster and cleaner employment process. For instance, students will be receiving notifications on application and their email when a new vacancy is posted near their region or in their field of study. Also, companies will receive notifications and emails when a student applies for their vacancy, making the entire process much faster.

7.2.4 Login with third-party services

Last but not least we want to enable logging in with 3rd party services like your Google or Microsoft account. We believe that users will find this feature very convenient because they will not have to register and go through the process of filling in all their information. With just the click of a few buttons, they will be able to link their existing accounts and quickly log into the platform.

7.2 OpenAPI Specification

During the final presentation of this project feedback was given about how to document the REST API, for this project a document was used with a self-designed method of doing this.

During the feedback, it was advised to use OpenAPI Specification (OAS) for this as this is the standard to document REST APIs. This should be added to the project as it is a useful way to discover how the application communicates without having to look through the source code or network traffic to understand. The result of this is that the application is easier to maintain.

Chapter 8 | Evaluation

This chapter evaluates the project as a whole by discussing its planning, stating responsibilities and task distribution within the team, and assessment of the team. Additionally, the chapter will provide an ending to the design report with a conclusion.

8.1 Planning

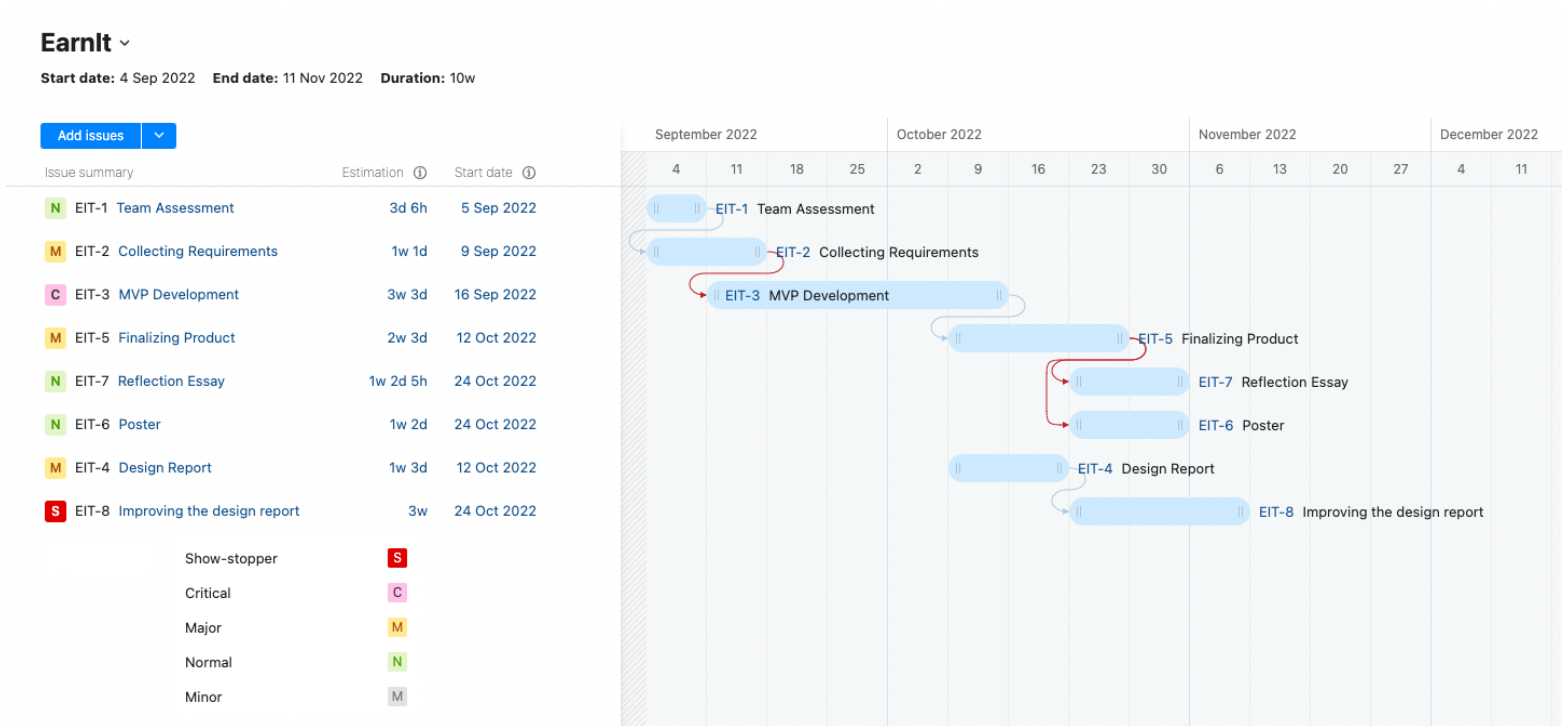


Fig.8. Planning Gantt chart. Created with JetBrains YouTrack software

The project and the team were divided into two separate parts namely, front-end and back-end. Each group had fixed deadlines spread throughout the 10 weeks of the project. The team had benchmarks to achieve throughout the project duration. The Gantt chart shows the distribution of the planning throughout the weeks.

Meetings with the client were held every Friday of each week in the module. During the meetings, the progress of the project was discussed. Furthermore, feedback from the client was used to update the requirements and the scope of the project. During the project lifecycle, additional requests were made by the client which were taken into consideration while planning the development of the product. The client requested monthly hours tracking

functionality for students to be added and to be able to visit students' profiles and download their CVs. Moreover, throughout the whole project, the client made suggestions on the styling of the platform and interface.

In hindsight, there were some problems with the project planning. The planning of the project was focused mainly on developing the system in the first 7 weeks of the module. The deliverables were planned to be completed during the last 3 weeks of the project. Some unexpected reasons such as a few group members falling ill had an influence on the project task schedule.

The focus was to ensure that the deliverable system to the client was completed and working well. However, this led to the completion of the design report being delayed.

8.2 Responsibilities

The team was composed of individuals with various skill sets and experiences in frameworks and technologies that were relevant to the project. Based on the level of experience and skill set, the tasks were evenly distributed throughout the team. Since the team members had a diverse set of skills and knowledge in different fields, this distribution was rather simple and intuitive. The task distribution of the team is as follows:

- **Wybe:** Front end developer, Front end routing, Front end authentication, Firebase, Front end login and registration, Requirements specification, Interface designer and Presentation slides
- **Dimitar:** Back end developer, Database designer, Protocol designer, Student functionalities, Application functionalities, Company functionalities, Vacancy functionalities, Cross-entity functionalities, Invalid input handling, Presentation slides, Poster Design, Head of Back end Management
- **Pranav:** Back end developer, Database designer, Security Implementation, Login Functionality, Protocol designer, Integration Testing, Presentation Slides
- **Yordan:** Back end developer, Back end structure, Database designer, Presentation slides
- **Stefan:** Front end Developer, Communication Manager with Supervisor and Client

The design report was written by everyone in the team, hence there is an equal contribution to the design report by each team member.

8.3 Team Evaluation

The team had mostly the same ideas and values when it came to working on the project. The frequent meetings online and constant communication via services such as Discord and WhatsApp provided the opportunity to accommodate every member's schedule and made the team more coordinated, thus making working on the project easier. Differences in opinions regarding the implementation of some aspects of the system were handled diplomatically and democratically to avoid building unnecessary tension between members. To keep motivation high, team members were encouraging and supportive of each other.

8.4 Conclusion

The project methodology ensured that the system development was completed and delivered on time. Team meetings and project work were mostly online, however, this did not have a negative effect on the development progress as meetings were done on a weekly basis and constant communication between team members was present. This helped immensely in keeping the team spirit strong and the weekly tasks clear and concise.

The aim of the project was to deliver a complete and error-free job-seeking platform for students in the Netherlands, no matter their citizenship. Considering the possibility of the source code being reused in the future, the development of a solid, bug-free system that did not allow for invalid user inputs was essential.

As was already mentioned, the project resulted in a working system that fulfils the client's expectations and requirements. Once the Design Module is finished, Earn It will take the system and either use it directly or take ideas from it and incorporate them into their existing platform.

This project resulted in team members understanding the software development process and its iterations better. Because of the scale and duration of the platform, the project provided insight into the challenges and potential problems that often arise in a project's lifecycle and between teammates. Furthermore, all the team members gained experience and knowledge about conducting interviews and communicating with clients. In addition, the development team was able to learn a lot and work with new technologies that are being used for modern-day projects.

In conclusion, the design project was successful as it helped students to gain valuable knowledge and more experience in working as a team for a client, which led to a working platform suiting the client's requirements.

References

1. <https://www.cbs.nl/en-gb/news/2022/11/40-percent-international-first-year-students-at-dutch-universities#:~:text=Over%20the%20past%2016%20years,number%20stood%20at%2033%20thousand.>
2. L.E. Gomez, Patrick Bernet, 2019, Diversity improves performance and outcomes, *Journal of the National Medical Association*, vol. 111(issue 4), p. 383-392
<https://doi.org/10.1016/j.jnma.2019.01.006>
3. R. Prieto-Diaz, Domain Analysis: An Introduction, *Software Engineering Notes*, vol 15,
<https://dl.acm.org/doi/pdf/10.1145/382296.382703>
4. [MySQL :: Mysql 8.0: UUID support](#)
5. S. Aggarwal, 2018, Modern Web-Development using ReactJS, *International Journal of Recent Research Aspects*, vol. 5(issue 1), p. 133-137
6. SMART goals definitions,
<https://www.atlassian.com/blog/productivity/how-to-write-smart-goals>
7. HTTP request methods definition
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>
8. Mestre, P., Madureira, R., Melo-Pinto, P., & Serodio, C. (2017). Securing RESTful Web Services using Multiple JSON Web Tokens. In Proc. World Congress on Engineering 2017 (pp. 418-23).
9. Elngar, A. A., Arafa, M., Fathy, A., Moustafa, B., Mahmoud, O., Shaban, M., & Fawzy, N. (2021). Items Page. *Journal of Cybersecurity and Information Management (JCIM)* Volume, 6(1_2).
10. Dikanski, A., Steinegger, R., & Abeck, S. (2012, August). Identification and implementation of authentication and authorization patterns in the spring security framework. In *The Sixth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2012)*, p. 14-30
11. <https://spring.io/guides/gs/rest-service/>
12. Gutierrez, F. (2016). Security with Spring Boot. In *Pro Spring Boot*. Apress, Berkeley, CA. p. 177-209
13. Tsai, W. T., Bai, X., Paul, R., Shao, W., & Agarwal, V. (2001, October). End-to-end integration testing design. In 25th Annual International Computer Software and Applications Conference. COMPSAC 2001 (pp. 166-171). IEEE.
14. Taky, M. T. (2021). Automated Testing With Cypress.
15. Conrad, T. (2006). Postgresql vs. MySQL vs. commercial databases: It's all about what you need.
16. [Spring Boot Annotations - javatpoint](#)
17. [JPA - JPQL \(tutorialspoint.com\)](#)

Appendices

Appendix A

Application Design

All page designs from the application.

The image displays two side-by-side registration form designs for a company named 'Student Company'. The forms are set against a dark background with white text and light-colored input fields.

Left Form:

- Title: Student Company
- Field: Email address (value: google@gmail.com)
- Field: Password (masked with dots)
- Field: First Name
- Field: Last Name
- Field: Birthday (format: dd/mm/yyyy, includes a calendar icon)

Right Form:

- Title: Student Company
- Field: Email address (value: google@gmail.com)
- Field: Password (masked with dots)
- Field: Company Name
- Field: Field
- Field: Contact Person

Figure A.1: Registration page design

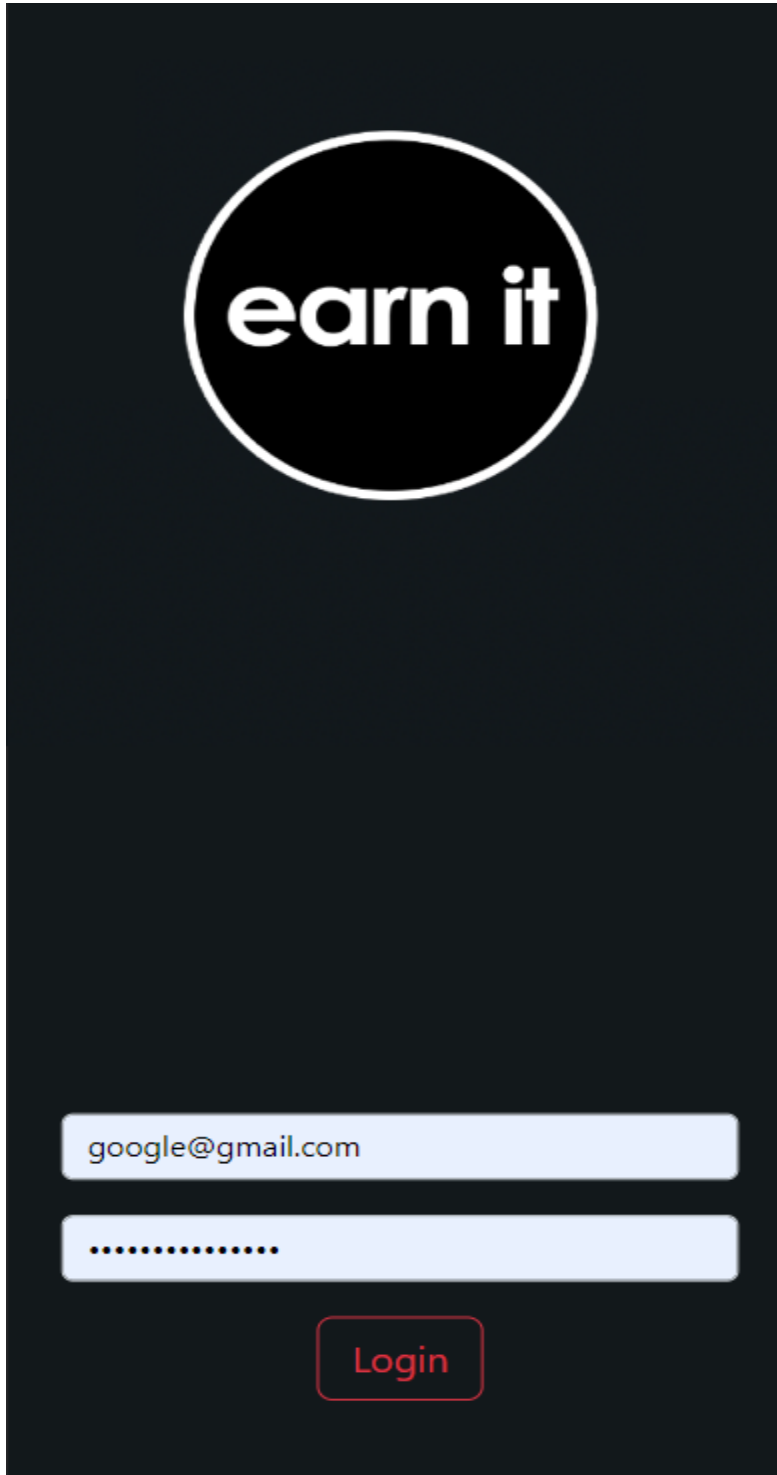


Figure A.2: Login page design

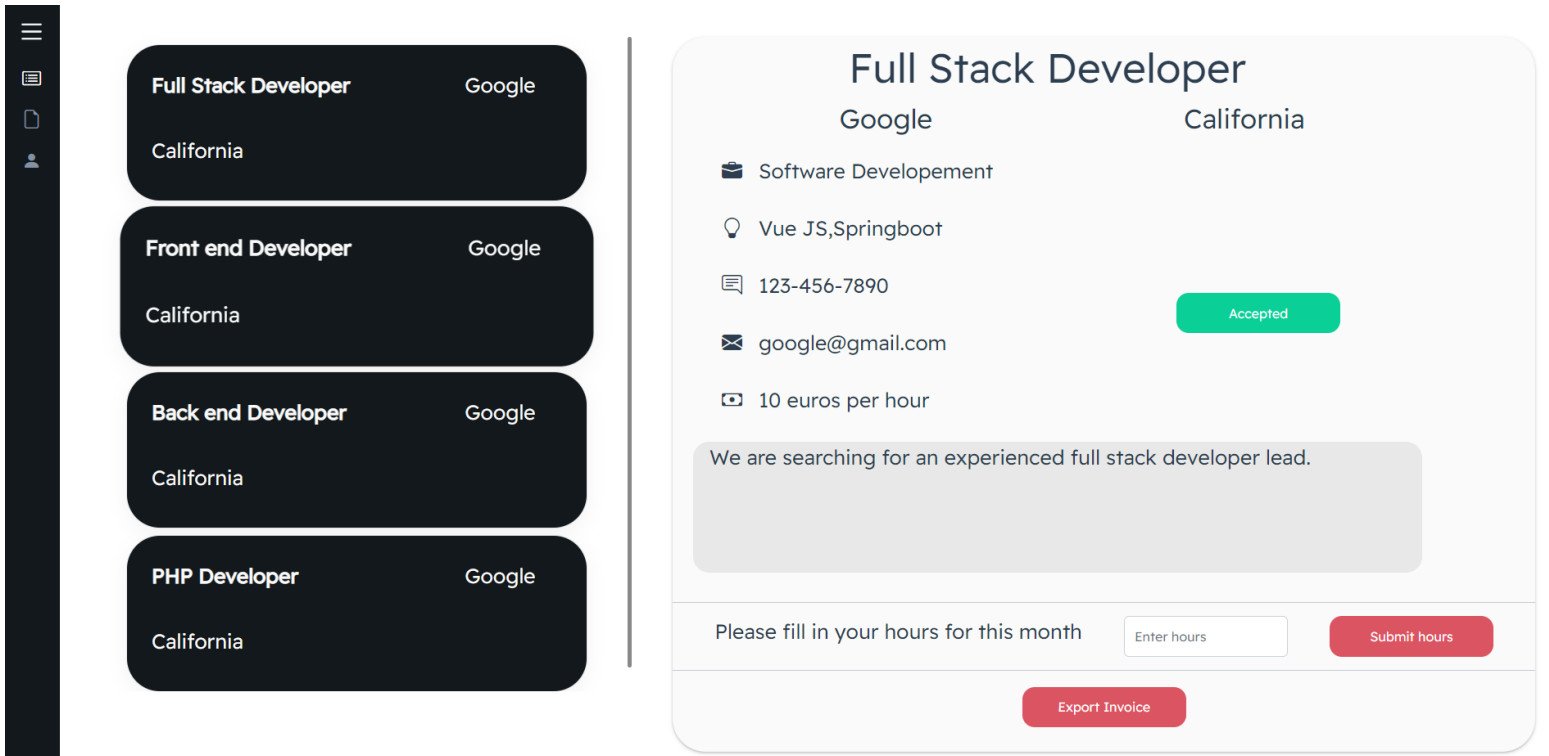


Figure A.3: Student dashboard page

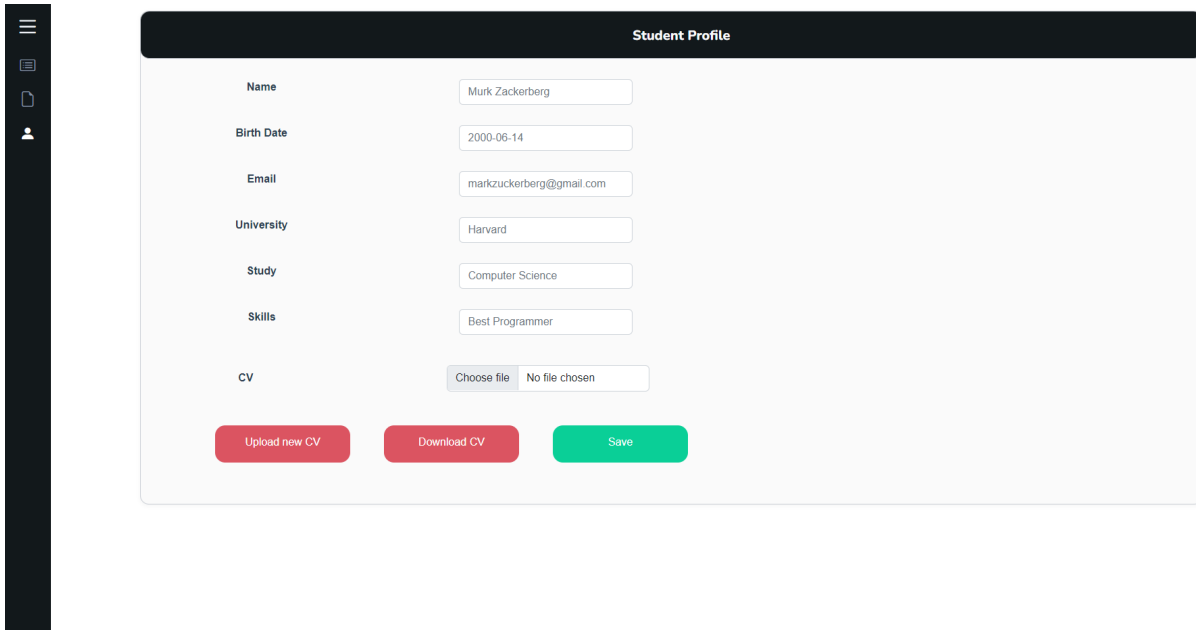


Figure A.4: Student profile Page

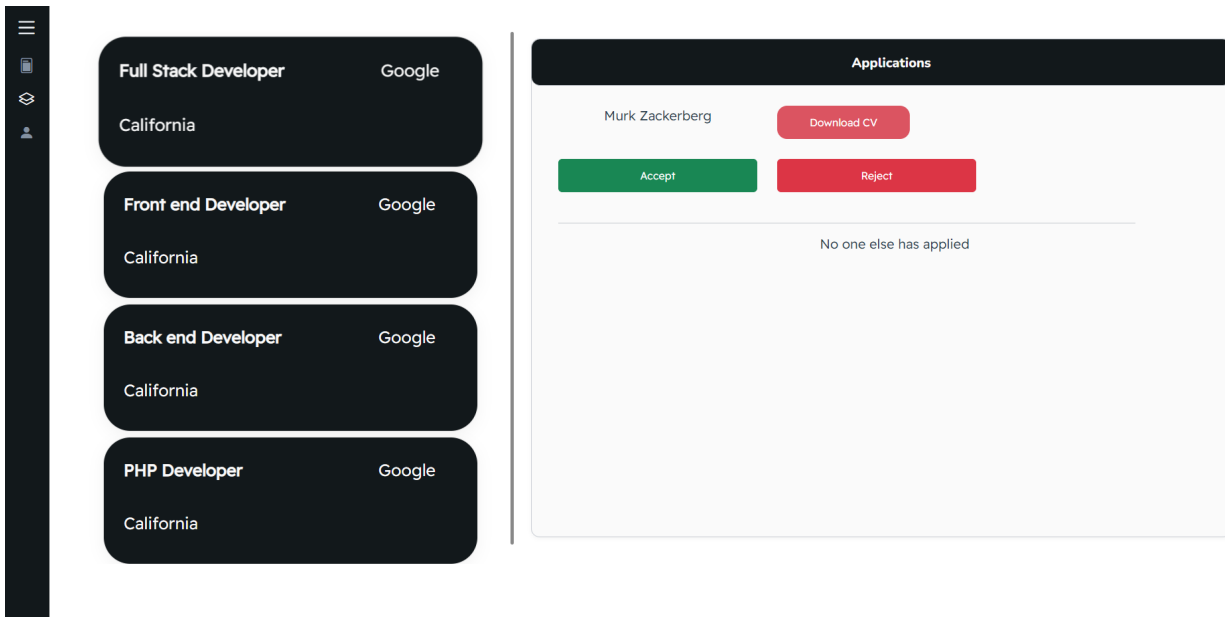
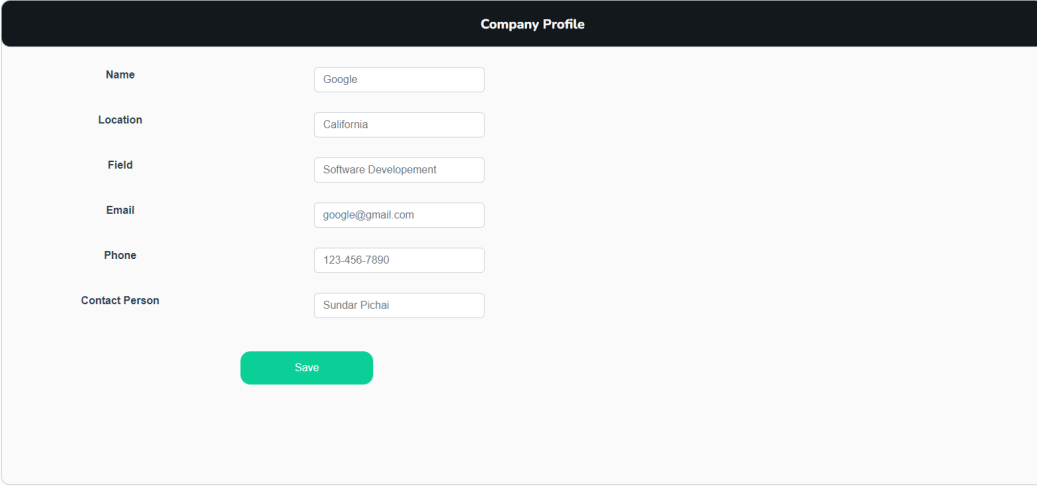


Figure A.5: Company dashboard page

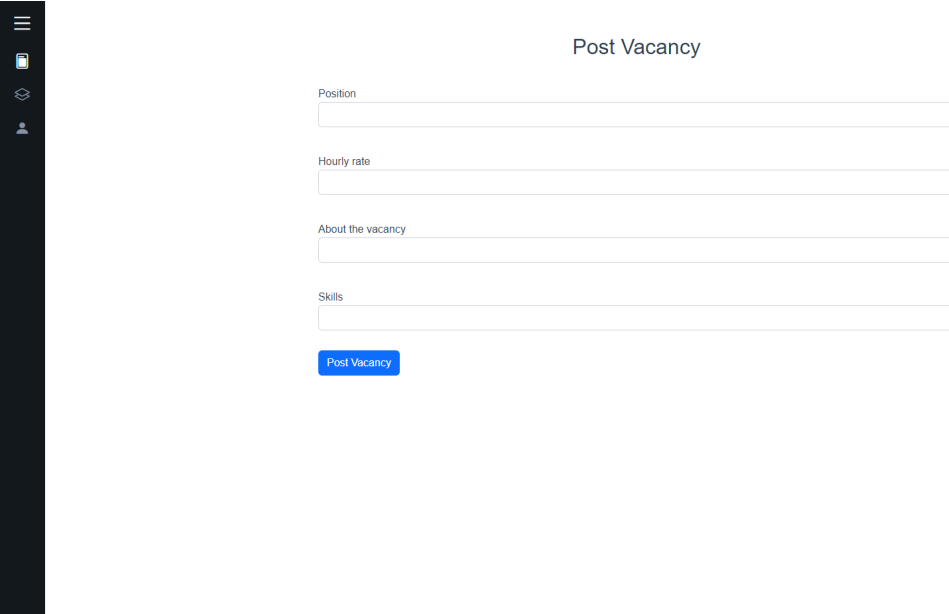


The image shows a 'Company Profile' form with a dark header bar. On the left, there is a vertical sidebar with icons for a menu, document, settings, and user profile. The form contains the following fields:

Name	<input type="text" value="Google"/>
Location	<input type="text" value="California"/>
Field	<input type="text" value="Software Development"/>
Email	<input type="text" value="google@gmail.com"/>
Phone	<input type="text" value="123-456-7890"/>
Contact Person	<input type="text" value="Sundar Pichai"/>

At the bottom of the form is a green 'Save' button.

Figure A.6: Company profile page



The image shows a 'Post Vacancy' form with a dark header bar. On the left, there is a vertical sidebar with icons for a menu, document, settings, and user profile. The form contains the following fields:

Position	<input type="text"/>
Hourly rate	<input type="text"/>
About the vacancy	<input type="text"/>
Skills	<input type="text"/>

At the bottom of the form is a blue 'Post Vacancy' button.

Figure A.7: Post Vacancy page

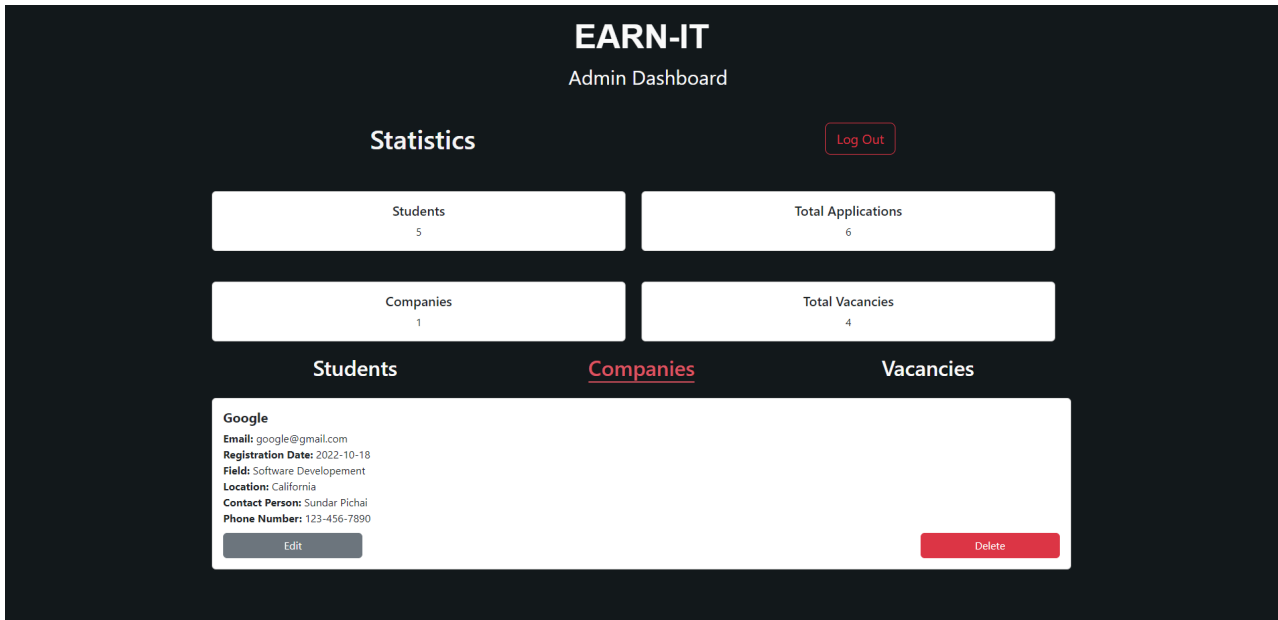


Figure A.8: Admin Dashboard page

Appendix B

Protocol Document

A collection of all the main functionalities of the platform.

Function	Requirements	URL	Fields	HTTP Method	File Type	Expected Response
Login a user	Must	/api/login	<pre>{ "idToken": " " }</pre>	POST	JSON	<pre>{ "userStatus": " ", "uid": " " }</pre>
Add a student	Must (1)	/api/student/create	<pre>{ "email": " ", "password": " ", "name": " ", "dob": " ", "university": " ", "course": " ", "skills": " " }</pre>	POST	JSON	-
Add a company	Must(8)	/api/company/create	<pre>{ "email": " ", "password": " ", "name": " ", "field": " ", "location": " ", "contact_person": " ", "phone": "" }</pre>	POST	JSON	-
Add a vacancy	Must(10)	/api/companies/{company_id}/vacancy/create	<pre>{ "Position": " ", "method_of_work": " " }</pre>	POST	JSON	-

Apply for Vacancy	Must(5)	/api/students/{student_id}/applications /{vacancy_id}	Empty	POST	JSON	-
Response to an Application	Must(13)	/api/vacancies/ changeApplicationStatus/ {application_id}/{status_code}	Empty; {status_code} should be: 0 - not responded 1 - accepted 2 - rejected 3 - withdrawn by student	POST	JSON	-
Get all Vacancies	Must(16)	/api/vacancies	Empty	GET	-	<pre>[- { vacancy_id: "", + company: { ... }, position: "", method_of_work: "", applications: [], about_vacancy: "", hourly_pay: 0, is_filled: false }, + { ... }, + { ... }, + { ... }, + { ... },]</pre>

Get all companies	Must(17)	../api/companies	-	GET	-	<pre>[- { company_id: "", - account: { account_id: "", account_type: 2, email: "", password: "", date_created: "" }, name: "", field: "", location: "", contact_person: "", phone: "" }, + { ... }, + { ... }]</pre>
-------------------	----------	------------------	---	-----	---	--

Get all students	Must(17)	../api/students	-	GET	-	<pre>[- { student_id: "", - account: { account_id: "", account_type: 1, email: "", password: "", date_created: "" }, name: "", dob: "", university: "", course: "", skills: "" }, + { ... }, + { ... },]</pre>
Get Student	Must(3)	../api/student/{student_id}	-	GET	-	<pre>{ student_id: "", + account: { ... }, name: "", dob: "", university: "", course: "", skills: "" }</pre>
Get Company	Must(8)	../api/company/{company_id}	-	GET	-	<pre>{ company_id: "", + account: { ... }, name: "", field: "", location: "", contact_person: "", phone: "" }</pre>
Get Vacancy		../api/vacancy/{vacancy_id}	-	GET	JSON	<pre>{ vacancy_id: "", + company: { ... }, position: "", method_of_work: "", applications: [], about_vacancy: "", hourly_pay: 0, is_filled: false }</pre>

Get all vacancies by ID (for company)	Must(14)	../api/vacancies/{company_id}	-	GET	-	<pre> [- { vacancy_id: "", + company: { ... }, position: "", method_of_work: "", applications: [], about_vacancy: "", hourly_pay: 0, is_filled: false }, + { ... }, + { ... }, + { ... }, + { ... },] </pre>
Get all vacancies a student has applied for	Must(7)	/api/student/vacanciesAppliedFor/{student_id}	-	GET	-	<pre> [- { vacancy_id: "", + company: { ... }, position: "", method_of_work: "", applications: [], about_vacancy: "", hourly_pay: 0, is_filled: false }, + { ... },] </pre>

Get all students applied for vacancy	Must(11)	/api/vacancies/appliedStudents/{vacancy_id}	-	GET	-	<pre>[- { student_id: "", - account: { account_id: "", account_type: 1, email: "", password: "", date_created: "" }, name: "", dob: "", university: "", course: "", skills: "" }, + { ... }, + { ... },]</pre>
Update student profile	Must(3)	../api/students/edit/{student_id}	<pre>{ "name": " ", "dob": " ", "university": " ", "skills": " ", "course": " " }</pre>	PUT	JSON	-
Update Company Profile	Must(8)	../api/companies/edit/{company_id}	<pre>{ "name": " ", "field": " ", "location": " ", "contact_person": " ", "phone": " " }</pre>	PUT	JSON	-

Update Account Credentials	Must	/api/account/edit/{account_id}	<pre>{ "email": " ", "password": " " }</pre>	PUT	-	-
Delete Student Profile	Must	../api/students/delete/{student_id}/{account_id}	-	DELETE	-	-
Get application by ID	Must	../api/applications/{id}	-	GET	JSON	<pre>{ student_id: "", + account: { ... }, name: "", dob: "", university: "", course: "", skills: "" }</pre>
Change Student Monthly Hours	Should	/api/students/{student_id}/monthly_hours/{vacancy_id}/{monthly_hours}	-	PUT	-	-