POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Design Document

Version 1.0

by
Abdallah Alkhetiar
Daniel Bonardi

December 23, 2024

Document details

Deliverable:	DD
Title:	Design Document
Authors:	Abdallah Alkhetiar and Daniel Bonardi
Version:	1.0
Date:	December 23, 2024
Download page:	github.com/Zero3474/AlkhetiarBonardi.git
Copyright:	

Contents

1 Introduction

1.1 Purpose

This design document outlines the architectural framework, user interface design, and other system components necessary to meet the functional and non-functional requirements described in the Requirement Analysis and Specification Document.
It also serves as a foundation for developers, testers, and stakeholders to understand the system's structure and ensures that the final implementation adheres to the defined goals.
The main purpose of the S&C platform is to grant efficient and effective matchmaking between university students seeking experience through internships and companies offering them. The design aims to ensure:

- **Precision in matching**: The system adopts a sophisticated recommendation system to notify students about suitable internships, based on criteria such as skills and experiences.
- **Smooth user experience**: The system shall grant its users the best experience by smoothly managing all internship-related workflows, including searching, application, selection, feedback collection, and monitoring of ongoing internships.
- **Scalability**: The system's design shall accommodate future adjustments, such as advanced data analytics, integration with external tools, or even an expanded scope.

1.2 Scope

Students&Companies (S&C) is an internship platform designed to connect students with companies based on skills, experiences, and interests.
The platform offers two main ways to establish connections:

- **Recommendation system**
- **Proactive searching**

S&C also supports the selection process and gathers feedback on recommendations and provides spaces for users to address issues, share updates, and track internship progress. More details can be found in the RASD document.

1.3 Definitions, acronyms, abbreviations

TBD

1.4 Revision history

Version 1.0 - WIP

1.5 Reference documents

TBD

1.6 Document structure

Section 1: Introduction
This section is presented with a brief overview of this document's content including the description of the purpose, the scope, and all the definitions, acronyms and abbreviations used.

Section 2: Architectural design
This section is presented with a detailed description of the architectural choices for the system, including a high-level description of the system and its components. In addition it is presented a detailed description of the deployment view and the main runtime views of the system.

Section 3: User interface design
This section is presented with a series of prototypes for the most important pages of the software to help graphic designers. Each image is accurately described to avoid any misunderstanding on the pages' available functionalities.

Section 4: Requirements traceability
This section is highly dependent on the RASD, as it provides a complete mapping of both functional and non-functional requirements described in that document, with the modules introduced and described in this document.

Section 5: Implementation, integration and test plan
The last section describes the procedures followed for implementing, integrating and testing the components of the platform.

2 Architectural design

2.1 Overview

The S&C design is based on two architectural choices:

- A 3 tier architecture
- A thin-client model

First of all the 3 tier architecture is built using 3 layers:

- **Presentation layer**: This layer is in charge of providing all the user interfaces for users to interact with the system.
- **Application layer**: This layer implements the core of the business logic and processes all the requests sent from the presentation layer.
- **Data layer**: This layer is responsible of managing all the data storage.

The choice of the 3 tier architecture what made to grant these key concepts:

- **Modularity**: Components are designed address a specific function required by the system, also making it easier to perform maintenance.
- **Scalability**: Since each module function independently from the others, adding or removing feature should be easier.
- **Security**: By implementing a secure communication protocol, users should not be worried about their sensitive data being stolen.

The thin-client approach instead, ensures the smoothest experience for the users, by keeping the client-side of the application as lightweight as possible. With this approach the client shall focus solely on the rendering of the user interface and the management of basic interactions, the rest of the computation happens on the server-side.
To sum it up, in the client there is only the presentation layer, while the application and data layer are in the server.

2.2 Component view

To make it easier to understand component view, we decided to divide it into 3 diagrams. The first represents a high-level view that distinguishes the client-side from the server-side components. The other 2 diagrams represent a more detailed version of the components presented in the server-side in the first diagram.

2.2.1 High level component view

In the following diagram it is shown how the client-side interacts with the server side through the exposed interfaces.
We can clearly see the 2 sides:

- **Client-side**: Since the in the client-side we only have the presentation layer, we can see that on the left of the diagram there are the Users, that can either be accessing the application through a web browser or a mobile device.
- **Server-side**: On the server-side instead, we have both the application and data layer so we have the most important subsystems of the entire application:

– *Internship management*: Which is the actual biggest component which manages all the actions that involve internships.
– *Account management*: Which simply manages all the interactions that require some data regarding a user.

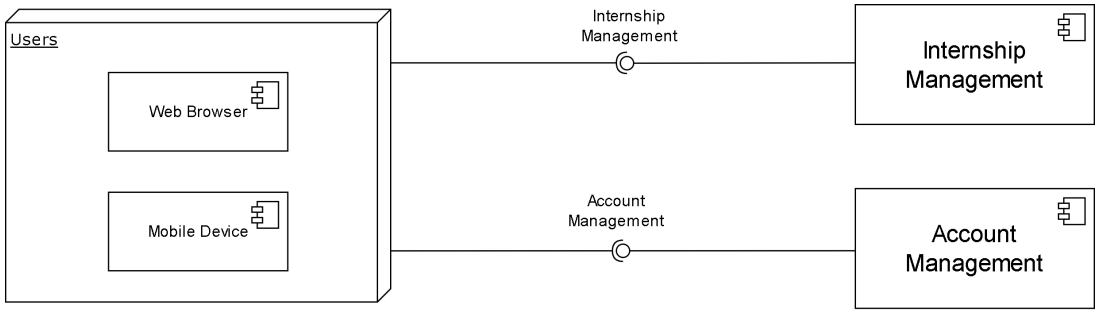

Figure 1: High level component view

2.2.2 Internship management diagram

The following diagram shows how the subsystem Internship Management is composed and how its modules communicate with external ones. This subsystem contains 6 modules:

- **Proactive search**: Allows student to search for specific internship.
- **Internship archival**: Allows companies to archive an internship, and show it on their profile.
- **Feedback**: Allows user to send a feedback.
- **Internship creation**: Allows companies to create an internship, and for this reason it needs to interact with the "Recommendation system" component to notify the compatible students.
- **Complaints**: Allows user to file a complaint, and it also needs to interact with the "Notification system" to notify the universities.
- **Form management**: Manages all the stages of a form, from its creation to the correction, and for this reason it needs to interact with the "Ranking system" component.

At the same time all components, external or internal to the subsystem, must interact with the DBMS to access all the data required for the interaction.

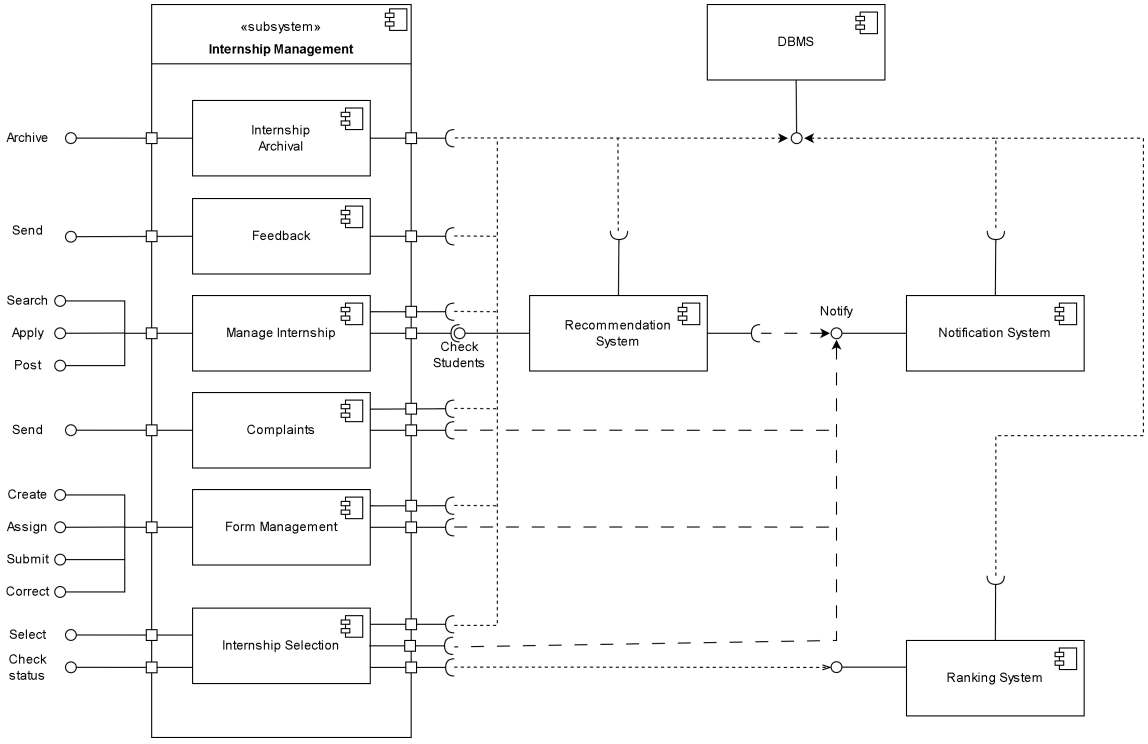

Figure 2: Internship management diagram

2.2.3 Account management diagram

The following diagram shows how the subsystem Account Management is composed and how its modules communicate with external ones. This subsystem contains only 2 modules:

- **Student manager**: Allows student to sign up, login, add a CV and update their profile.
- **Company manager**: Allows a company to sign up, login and update their profile.

In both cases the component need to access both the "Authentication" component to safely complete the login and sign up processes, and also the DBMS to manage all the required data.

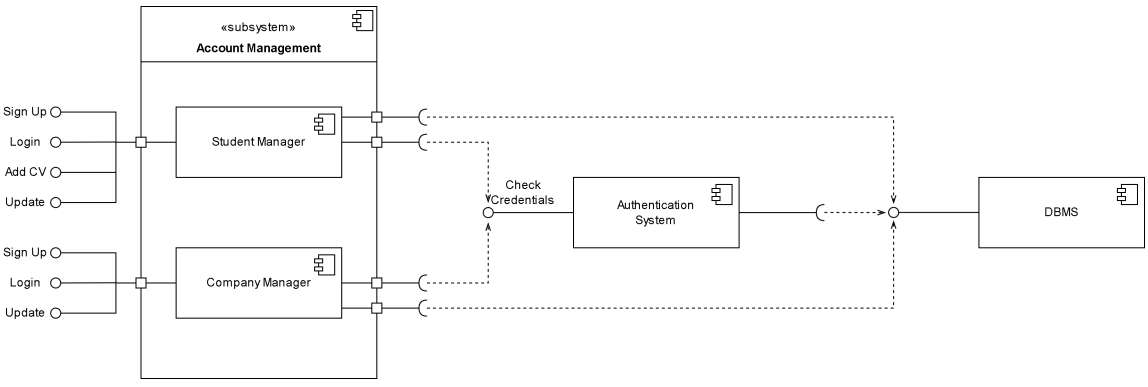

Figure 3: Account management diagram

2.3 Deployment view

2.4 Runtime view

With the following sequence diagrams, we are going to explain and represent all the interactions that happen between the components of the S&C system. But since this representation is still a high-level description of interactions, the function names, parameters, results, errors, and other smaller details will be modified or added during the development phase.

2.4.1 Login

In the login process, the user interacts with the *Web Application* to login in his account. After adding the required information, the data is sent to the *Account Manager* that checks if the credentials are correct through the *Authentication System* that accesses the *DBMS*. If the credentials are correct, the user is shown the home page, otherwise he is shown a message error and can try to login again.

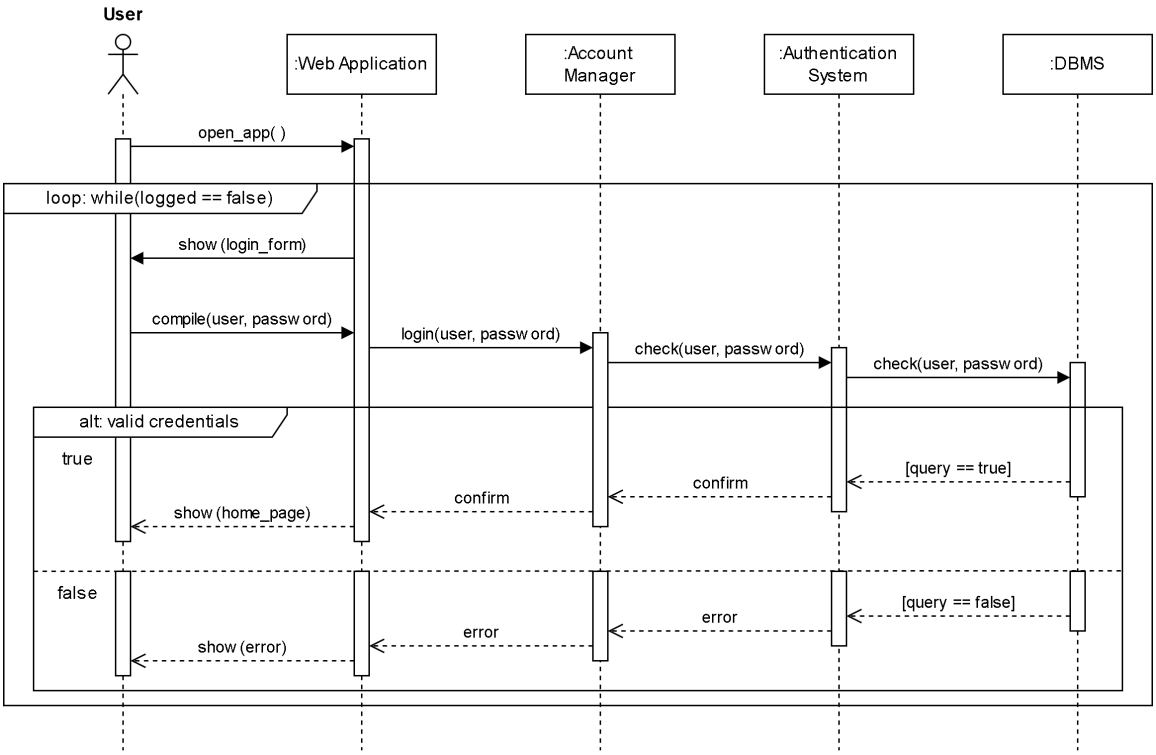

Figure 4: Login sequence diagram

2.4.2 Sign up

In the sign up process, the user interacts with the *Web Application* to create a new account. When entering the application, the user is presented with the login page, so he needs to request for the sign up page. After adding the required information, the data is sent to the *Account Manager* that checks if the credentials already exist in the data base. If the given information are correct, the user is shown the loin page, otherwise he is shown a message error and can try to create his account again.

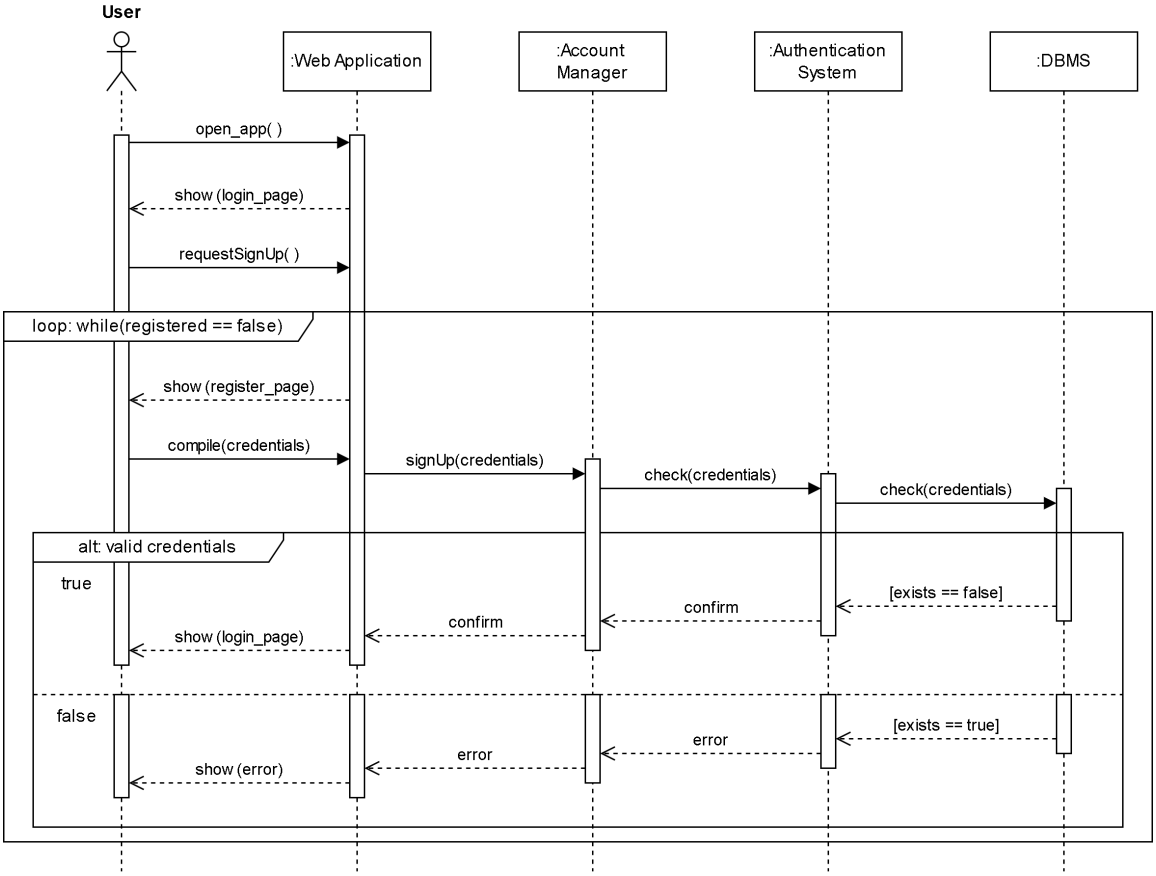

Figure 5: Sign up sequence diagram

2.4.3 Internship creation

In the internship creation process, a company interacts with the *Web Application* to post about a new internship. When entering all the correct information, the *Manage Internship* module interacts with the *Recommendation System* to identify all the student that are deemed compatible with the internship. Once the system has gotten the list, all the specified students are notified through the *Notification System*.

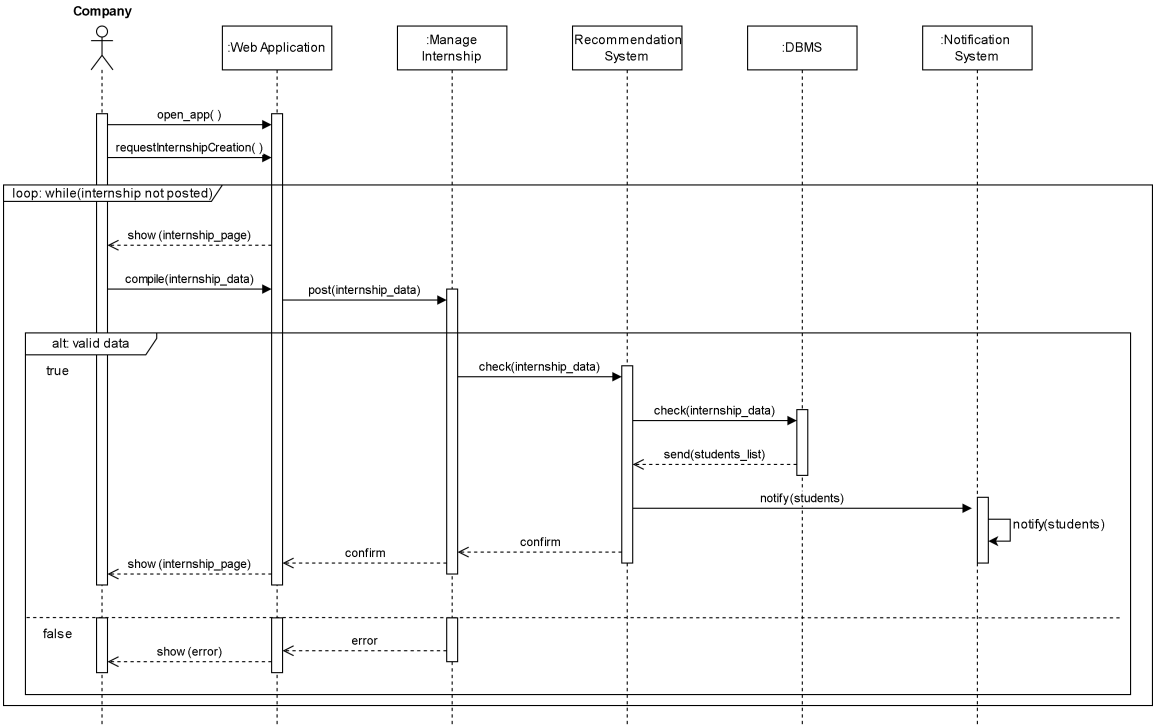

Figure 6: Internship creation sequence diagram

2.4.4 Internship application

In the internship application process, the student interacts with the *Web Application* to search for a specific internship and apply for it. The student searches for specific keywords in the search bar, which activates the *Manage Internship module* and through the *DBMS* it retrieves the correct internships. The results are show on the *Web Application* and the student can apply for the ones that he prefers. When applying to the internship the *Manage Internship* module interacts with the *Notification System* to notify the company of the candidates.

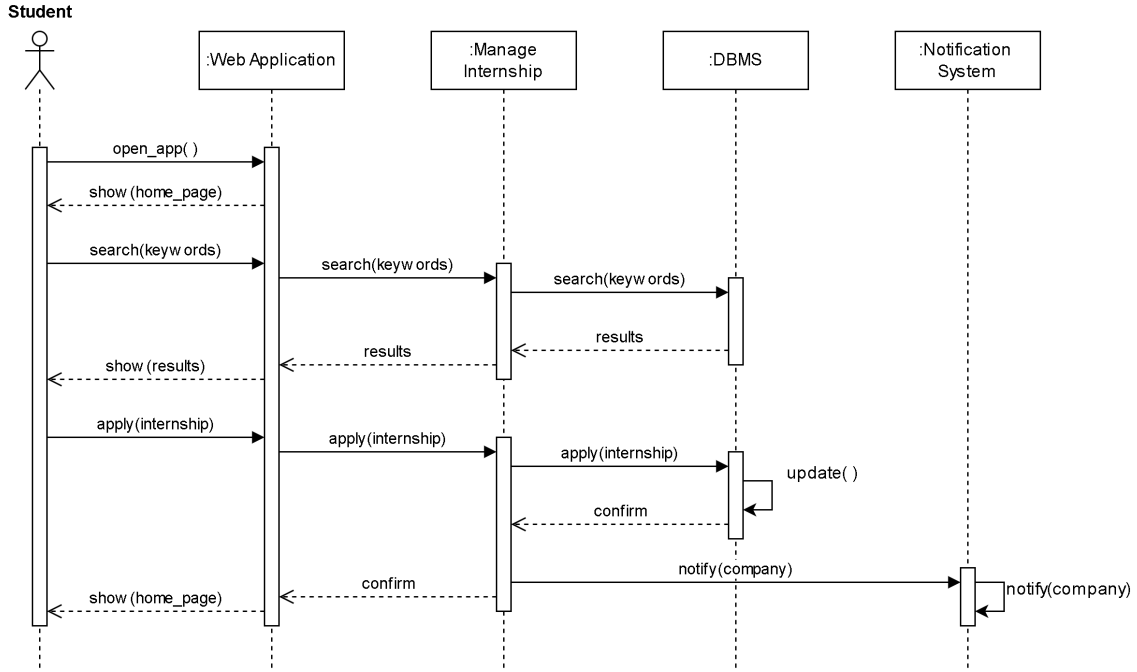

Figure 7: Internship application sequence diagram

2.4.5 Form assignment for internship

In the form assignment process, a company interacts with the *Web Application* to assign a form to an already existing internship. In this case the *Web Application* needs to interact with the *Form Management* module to assign the form to the internship, and the *Notification System* to notify all the candidates of the form to fill. Notice that this flow happens only when assigning a valid form, trying to assign an empty form shows an error message.

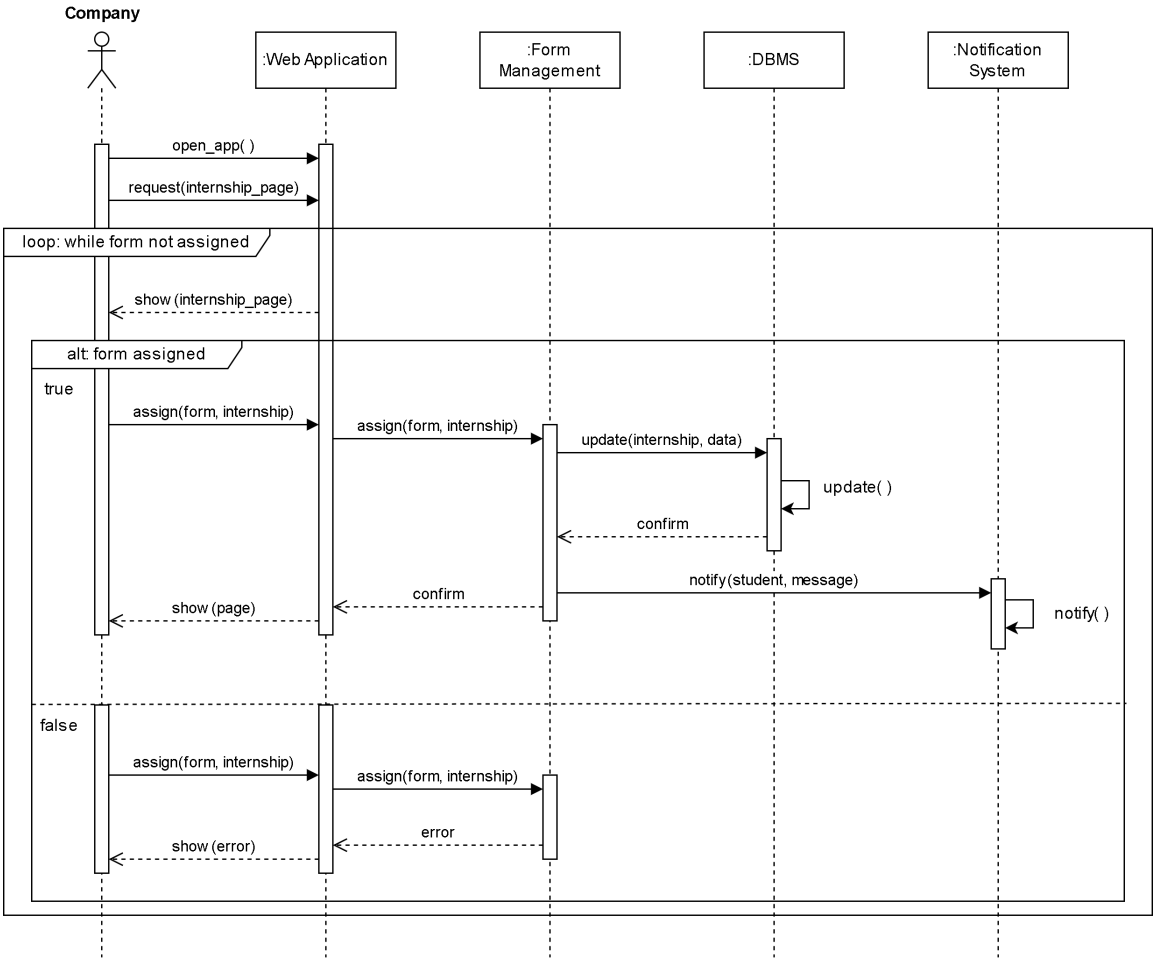

Figure 8: Form assignment sequence diagram

2.4.6 Form compilation

In the form compilation process, the student interacts with the *Web Application* to fill and submit a form for an internship. In this case the *Web Application* needs to interact with the *Form Management* module to save the answer in the data base, and the *Notification System* to notify the company of the compiled forms they need to correct. Trying to submit an form with a missing answer will prevent the submission and show an error message instead.

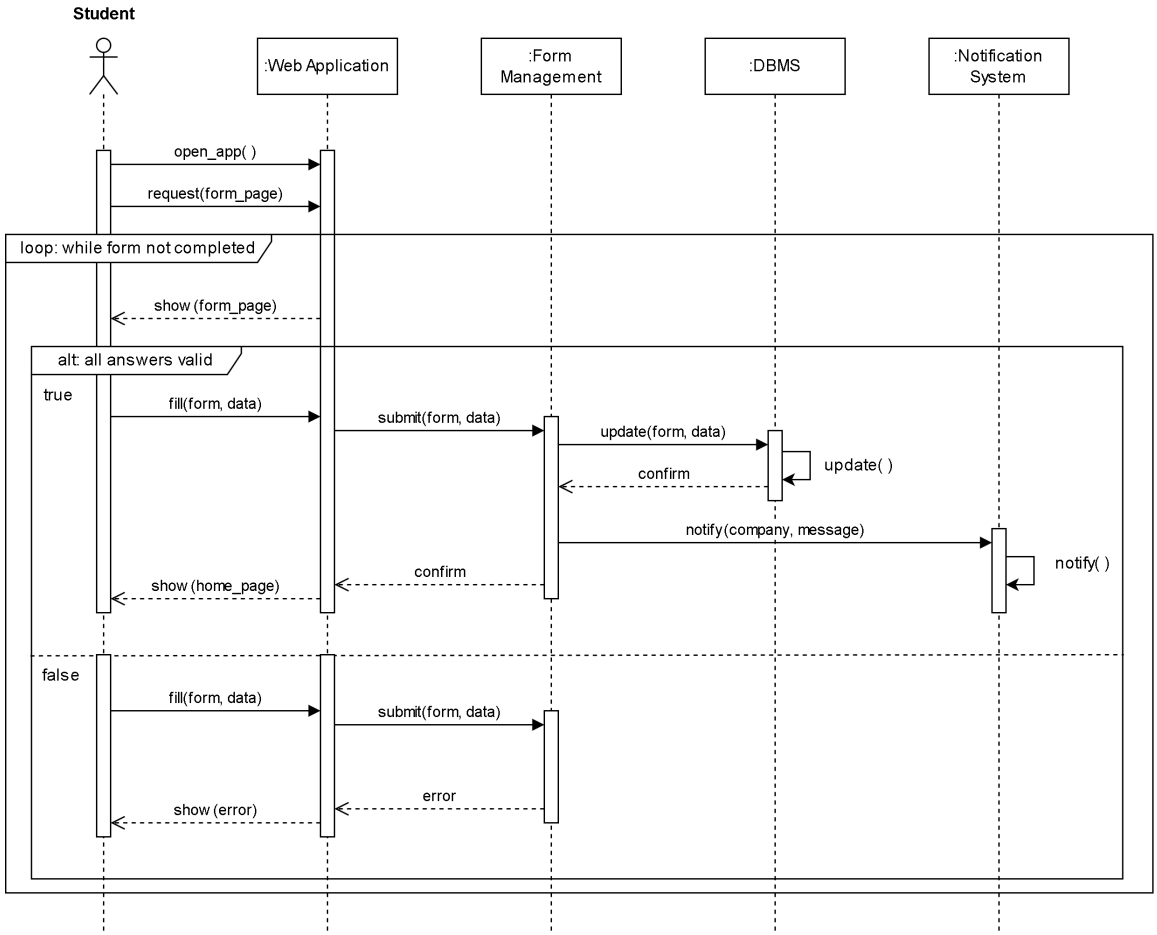

Figure 9: Form compilation sequence diagram

2.4.7 Form correction

In the form correction process a company interacts with the *Web Application* to correct a form submitted by a student. In this case the *Web Application* needs to interact with the *Form Management* module to save the score of the candidate for later use in the ranking system. The score can be submitted only when each individual question is given a valid score.

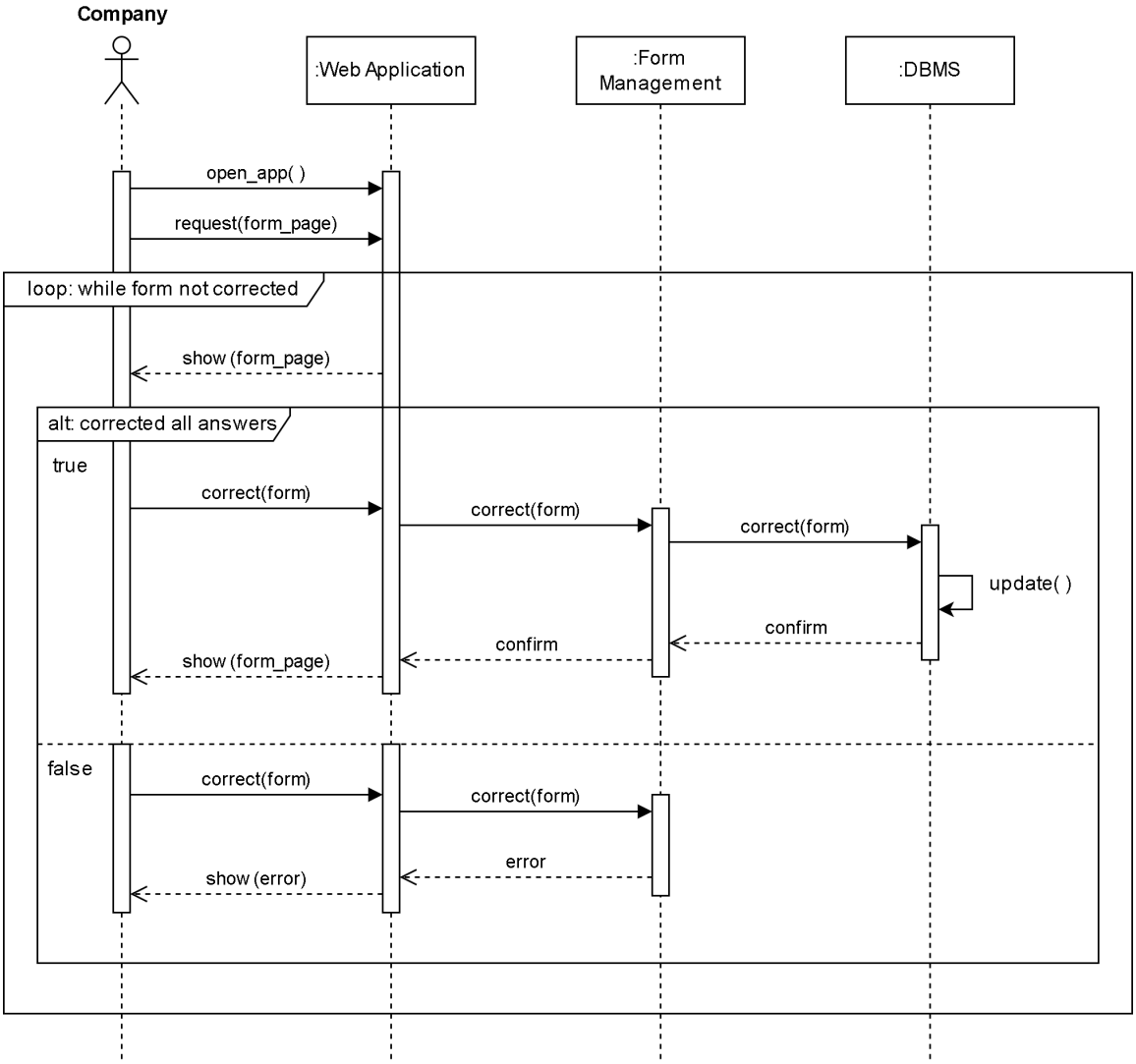

Figure 10: Form correction sequence diagram

2.4.8 Student selection

In the student selection process a company interacts with the *Web Application* to select the candidates that can move to the next phase of the selection process or that were chosen to participate in the actual internship. In this case the *Web Application* needs to interact with the *Internship Selection* module and the *Ranking System* to retrieve from the data base the scores of all the candidates and format the in a readable way. The ranking is shown on the screen of the company application, and they can then choose a list of students that will advance to next phase. For this reason the *Internship Selection* module needs to also interact with the *Notification System*.

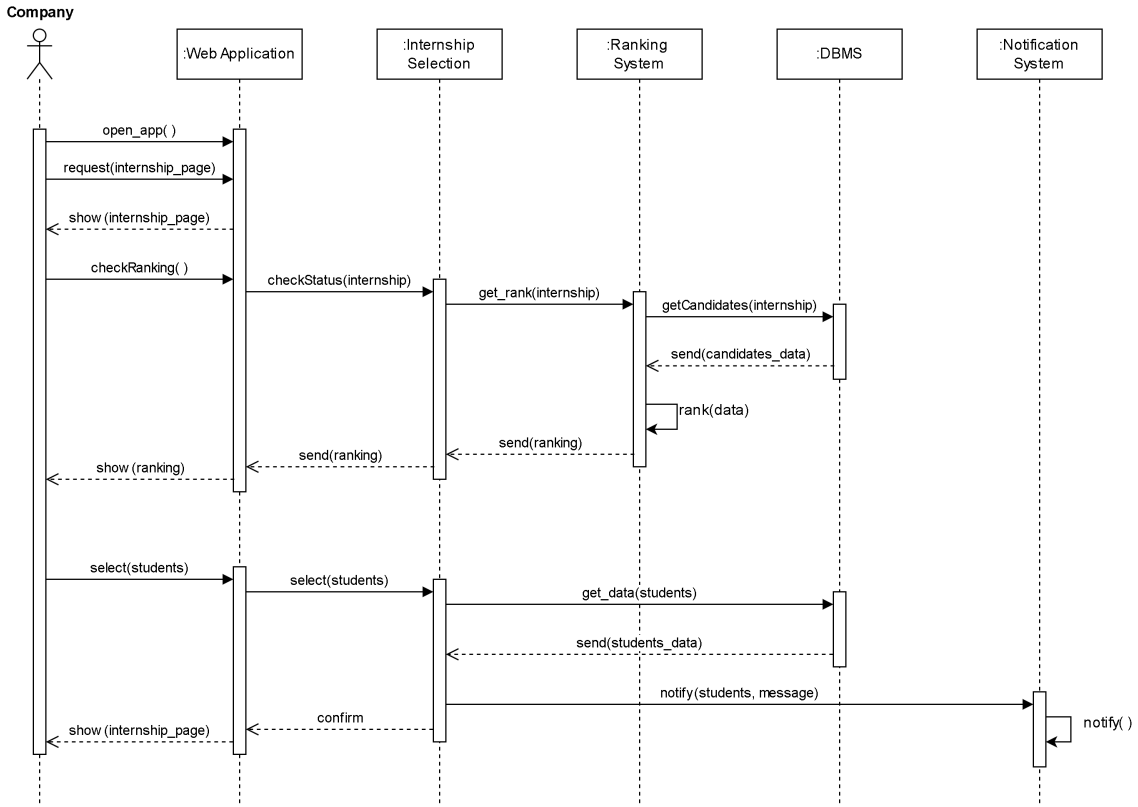

Figure 11: Student selection sequence diagram

2.5 Component interfaces

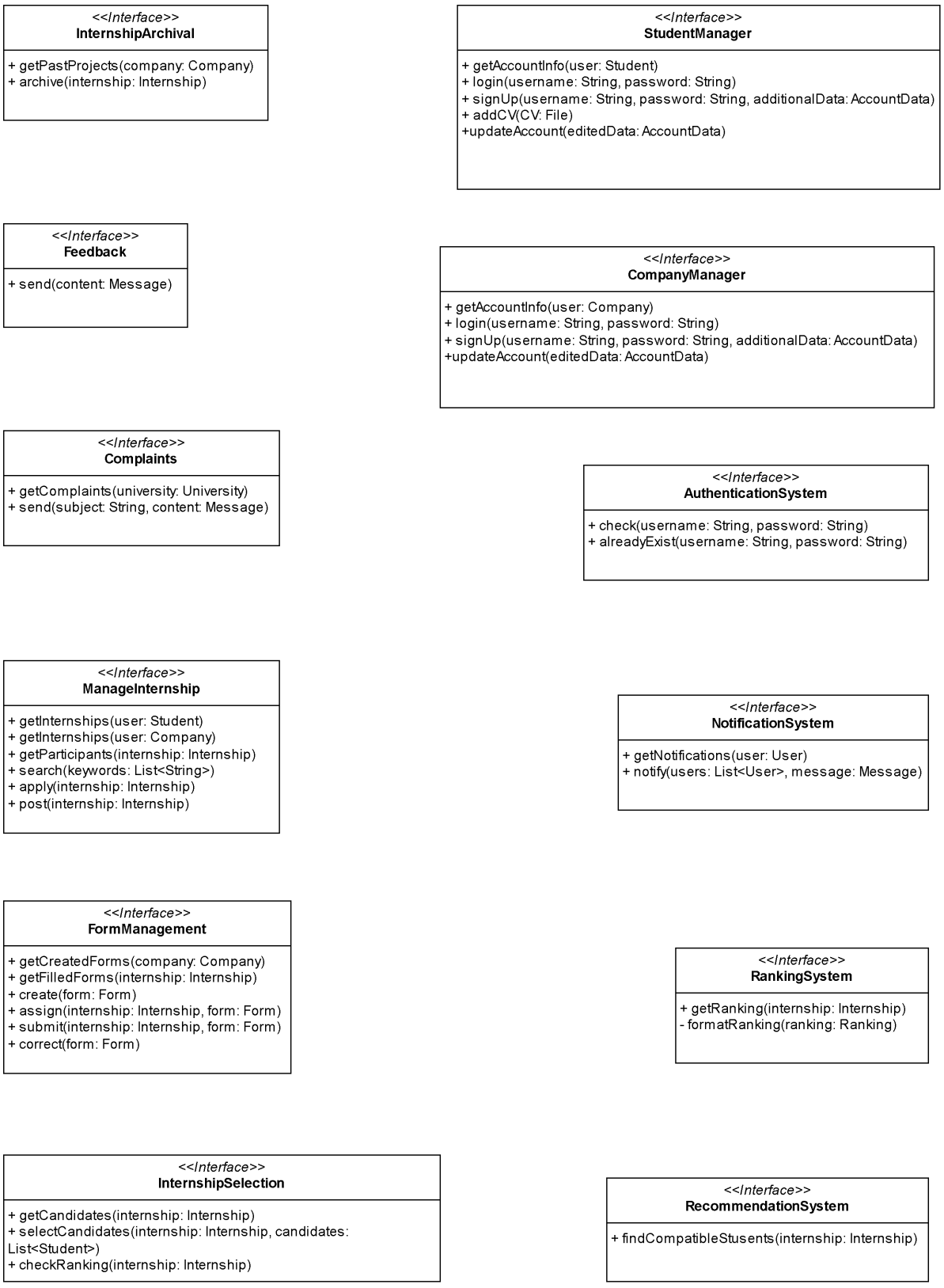

Figure 12: Component interfaces diagram

In figure [12] we can see a list of all the interfaces and their respective exposed methods. In this section we are going to provide a more detailed description of such interfaces.

- **Internship Archival**:
  - *getPastProjects()*: Allows a user to retrieve all past projects of a given company.
  - *archive()*: Allows a company to add a concluded internship to their list of past projects.
- **Feedback**:
  - *send()*: Allows a user to send a feedback to the developers.
- **Complaints**:
  - *getComplaints()*: Allows a university to retrieve all complaints that involve at least one of their student.
  - *send()*: Allows a user to file a complaint.
- **Manage Internship**:
  - *getInternships()*: Overload parameters
    - Student: Allows a student to retrieve all the internship to show in his home page.
    - Company: Allows a user to retrieve all the internships that have been created by a certain company.
  - *getParticipats()*: Allows a company to retrieve all the candidates of an internship.
  - *search()*: Allows a student to retrieve all internships compatible with a list of keywords.
  - *apply()*: Allows a student to apply for a specific internship.
  - *post()*: Allows a company to post regarding a newly created internship.
- **Form Management**:
  - *getCreatedForms()*: Allows a company to retrieve all form they created.
  - *getFilledForms()*: Allows a company to retrieve all forms that have been filled for a specific internship.
  - *create()*: Allows a company to save the form they created.
  - *assign()*: Allows a company to send all the candidates of an internship the form for the first selection phase.
  - *submit()*: Allows a student to submit a form regarding a specific internship.
  - *correct()*: Allows a company to save the score of a corrected form.
- **Internship Selection**:
  - *getCandidates()*: Allows a company to retrieve all the available candidates that want to participate in a given internship.
  - *selectCandidates()*: Allows a company to select a list of candidates, allowing them to move on to the next selection phase.
  - *checkRanking()*: Allows a company to retrieve the ranking for the current selection phase of a given internship.
- **Student Manager**:

  - *getaccountInfo()*: Allows a student to retrieve all the required information to show the profile page.
  - *login()*: Allows a student to try logging in his account using the given username and password.
  - *signUp()*: Allows a student to try creating a new account with the specified information.
  - *addCV()*: Allows a student to add a file in the CV section.
  - *update()*: Allows a student to edit the personal information shown in the profile.

- **Company Manager**:
  - *getaccountInfo()*: Allows a company to retrieve all the required information to show the profile page.
  - *login()*: Allows a company to try logging in his account using the given username and password.
  - *signUp()*: Allows a company to try creating a new account with the specified information.
  - *update()*: Allows a company to edit the personal information shown in the profile.

The following modules contain methods that are called by the main modules, so the users cannot directly interact with them.

- **Authentication System**:
  - *check()*: Checks the correctness of the given credentials. This method is called by the *login* method.
  - *alreadyExist()*: Checks if the given information already exist in the database. This method is called by the *signUp* method.

- **Notification System**:
  - *getNotifications()*: Retrieves all the notifications to be shown to a specified user.
  - *notify()*: Sends a notification to a specified user.

- **Ranking System**:
  - *getRanking()*: Retrieves the ranking of a specified internship. This method is called by the *checkRanking* method and returns the result of *formatRanking*.
  - *formatRanking()*: Allows to format a ranking in a more readable way.

- **Recommendation System**:
  - *findCompatibleStudents()*: Retrieves all the compatible students for a given internship. This method is called by the *post* method of Manage Internship.

2.6 Selected architectural styles and patterns

2.6.1 Architectural styles

We already discussed about the tree-tier architecture with thin client approach and their benefits in the overview section (2.1).
Our choice in terms of communications protocol is HTTPS. First of all we chose it because it is one of the most used across most platforms, so it should not be too much of a problem to

implement and second, is because it is the secure version of HTTP, the main protocol used to share data between a website and a browser. In addition HTTPS grants the security of the communication by encrypting the data to transfer. This is especially important in the case a user has to manage sensitive data regarding his profile.

2.6.2 Patterns

- **Model-View-Controller (MVC)**: The logic in the application layer is structured using the MVC pattern, which allows the to organize the system in three different components:
  - **Model**: Represents all the data logic in the system.
  - **View**: Handles the presentation of the data to the user.
  - **Controller**: Manages the interactions with the model.
- **Observer pattern**: Allows an object, named the subject, to maintain a list of dependent objects, called observers, and notify them whenever happens a change of state. This pattern is especially recommended given the use of the MVC pattern.
- **Strategy pattern**: Allows the selection of an algorithm at runtime. Instead of implementing a single algorithm directly, the system receives runtime instructions as to which in a family of algorithms to use.
- **Factory method**: Uses factory methods to deal with the problem of creating objects without having to specify their exact classes. The creation of an object is achieved by invoking a factory method, instead of calling a constructor.

2.7 Other design decisions

- **OAuth 2.0**: OAuth 2.0, which stands for "Open Authorization", is a standard designed to allow a website or application to access resources hosted by other web apps on behalf of a user. It provides consented access and restricts actions of what the client app can perform on resources on behalf of the user, without ever sharing the user's credentials.
  Important: OAuth 2.0 is an authorization protocol and NOT an authentication protocol, which is why we have a custom *Authentication System* module in the component view and not an authorization one.
- **Notifications API**: Firebase Cloud Messaging (FCM), formerly known as Google Cloud Messaging (GCM), is a cross-platform push notification API for Android, iOS, and web applications. It is one of many tools under the Firebase platform umbrella, allowing you to combine various products for web and mobile production.
  Since it's a Google product it is reliable and easy to implement thanks to a detailed documentation and several tutorials provided by the platform itself.

3 User interface design

The aim of this section is to present the prototypes of the most important pages of the application and explain the features by emulating scenarios of a user interacting with the system.

Login and Sign Up page

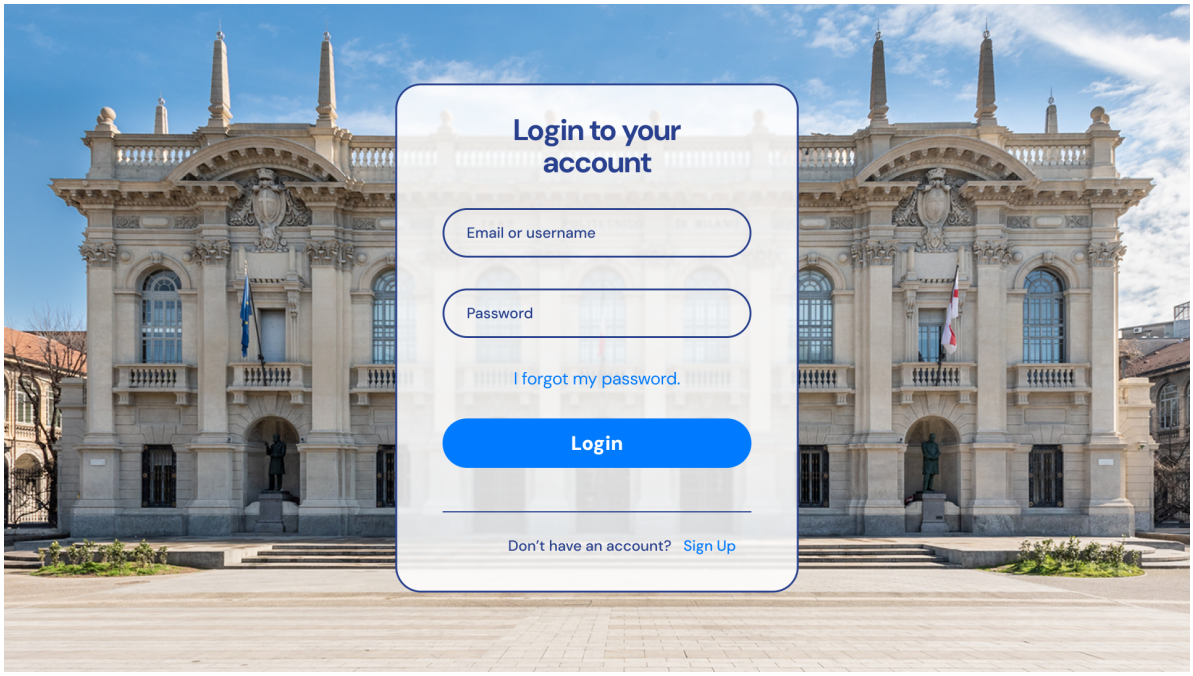

Figure 13: Login prototype page

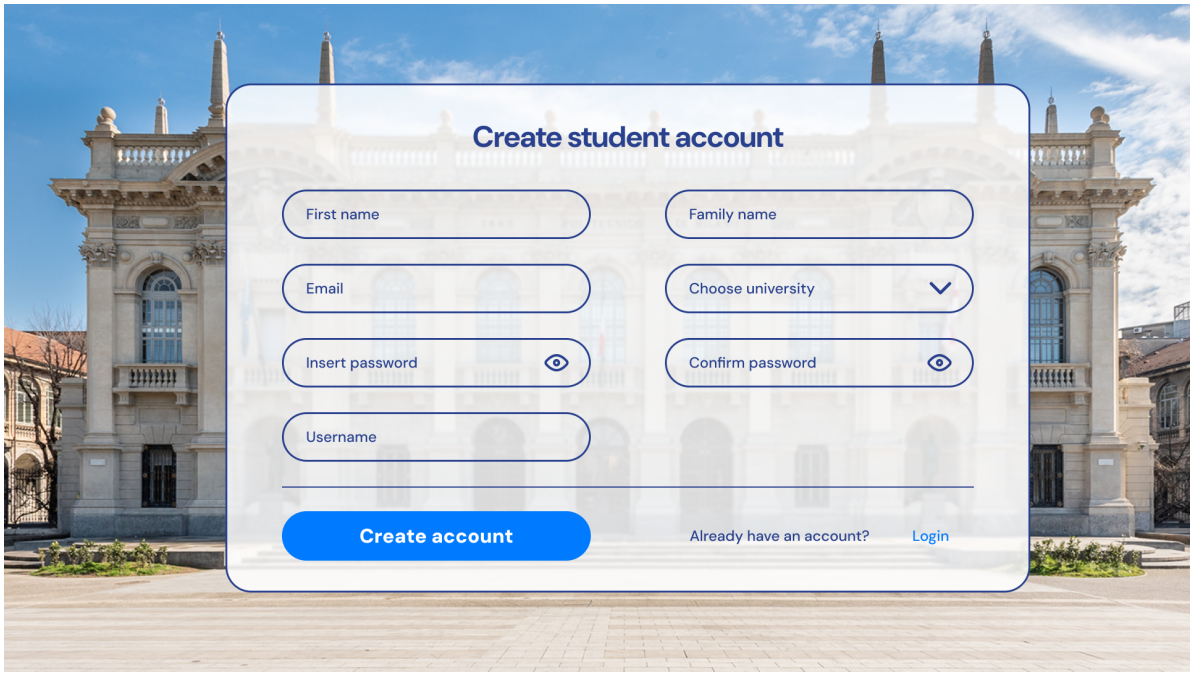

Figure 14: Sign up prototype page

The picture [13] shows the prototype of the login page, allowing to either login by entering the correct credentials, create an account in case the user doesn't have one yet, or reset the password in case it was forgotten.
The picture [14] shows the prototype of the sign up page for students, that allows students to create a new account by entering all the required information. In case the user already has an account there is a button that redirects to the login page. The case of a company creating an account is really similar to the one shown above with the simple difference that the user wont be required to choose a university.

Profile page

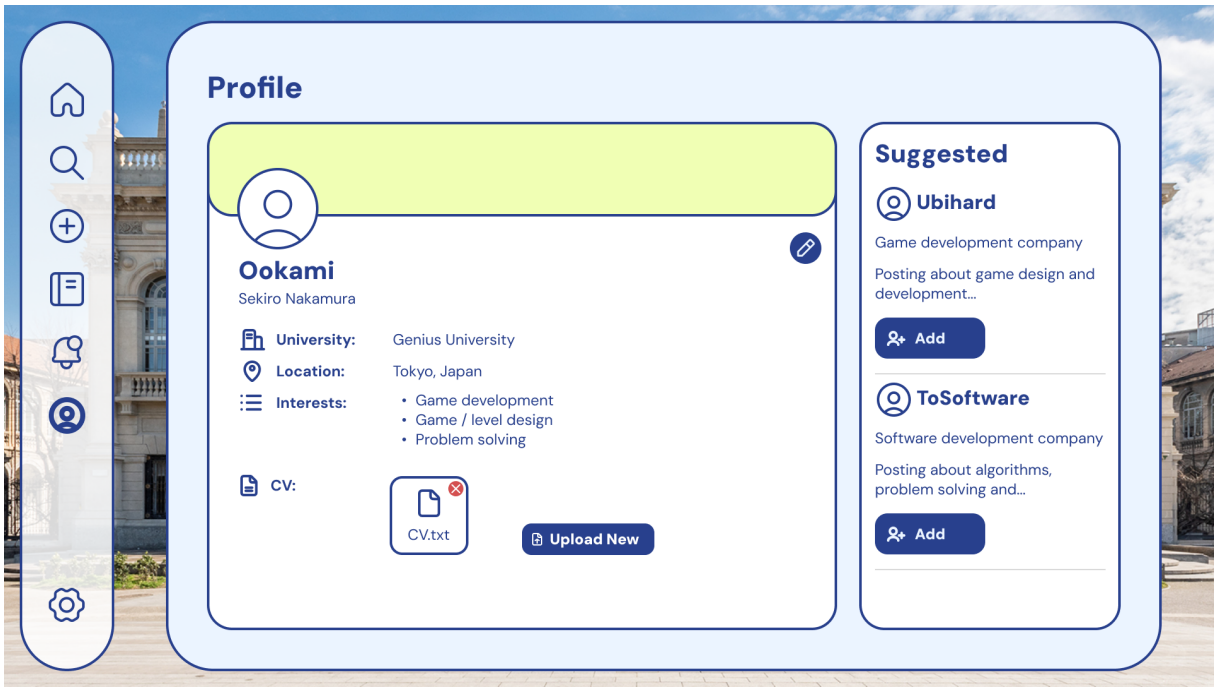

Figure 15: Student profile prototype page

The picture [15] shows the prototype of the profile page of a user logged in as a student. On the left, there is the section containing all the information provided by the user, and through the pencil button on the right, the user can modify the information. On the bottom there is the CV section. By clicking the file icon it opens the file allowing the user to check if the uploaded file is correct, otherwise the button on the right allows to upload a different file.
On the right side of the picture there is the "Suggested" area that show all the companies that might interest the user based on his interests.

Notifications page

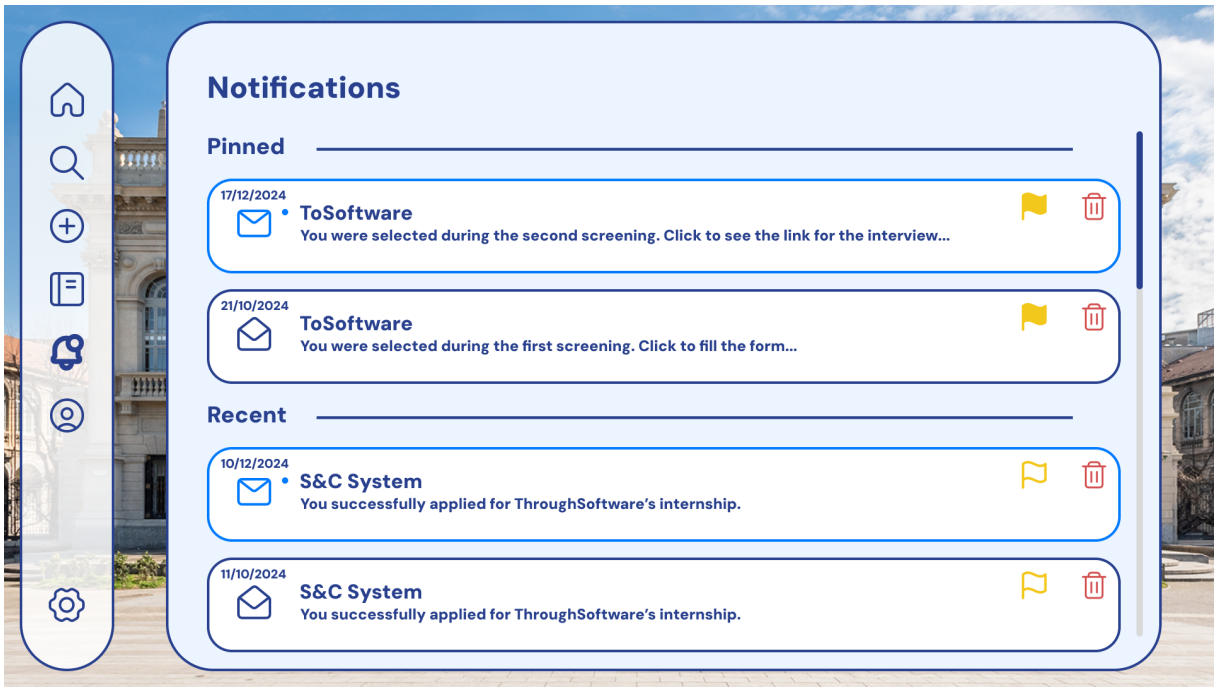

Figure 16: Notifications prototype page

The picture [16] shows the prototype of the notifications page and is show to all three user categories. In general all notifications shown in the page render only a part of the whole message, and clicking on it allows the user to open a page that shows the whole text. In addition, notifications can be pinned to be always shown on top in their specific section by clicking on the yellow flag, or can be also permanently deleted by clicking on the trash can.

Search page

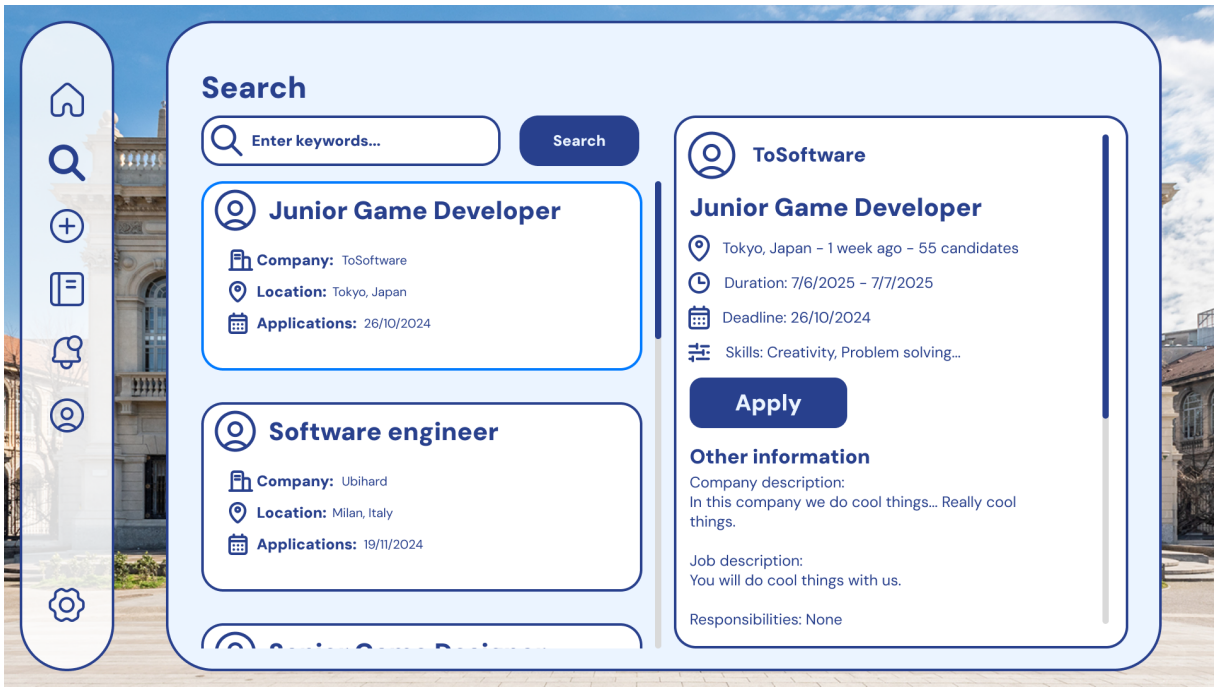

Figure 17: Search prototype page

The picture [17] shows the prototype of the search page, that allow students to proactively search for internship by entering keywords in the search bar.
Right after searching the keywords the internship found are displayed on the left side and only show the basic details. By clicking on the desired internship, all the additional information entered by the company are displayed on the right side. Only when visualizing the details of an internship it is possible to apply for said offer through a dedicated button.

Form creation page

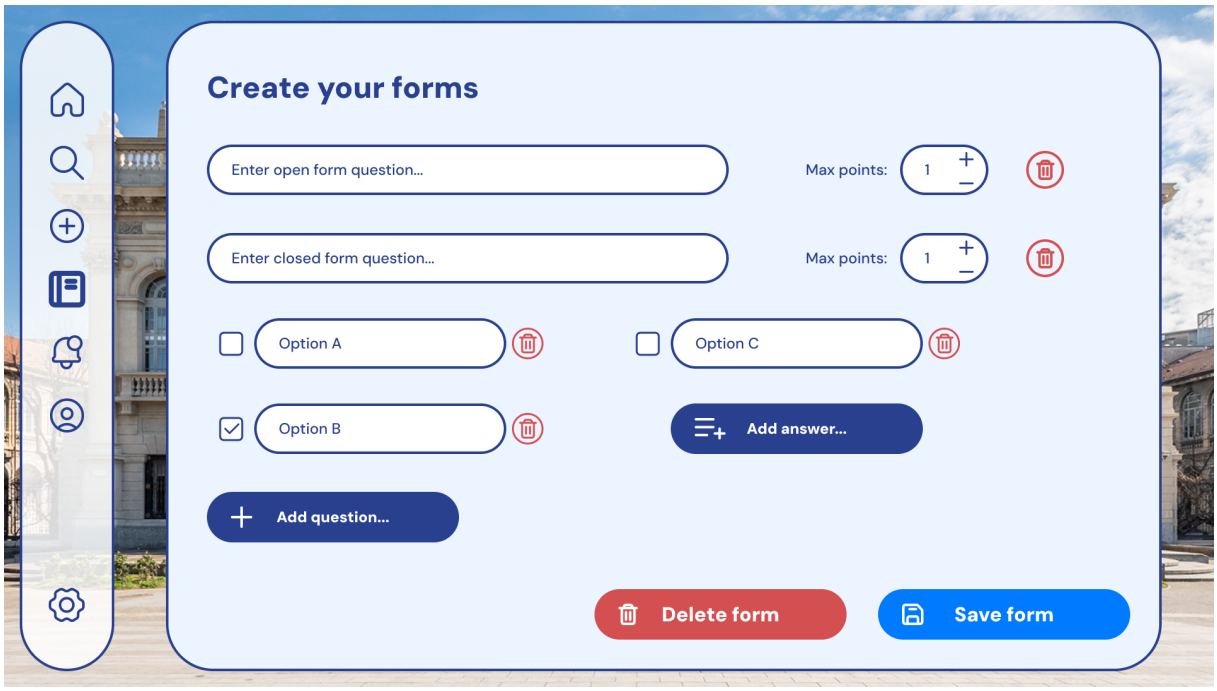

Figure 18: Form creation prototype page

The picture [18] shows the prototype of the form creation page that allows only companies to create custom forms for the candidates to fill during an application. In the form can be added two different types of questions: open and closed.
Open question only require the question text and the number of maximum point.
The closed ones not only require the question text and the maximum points like the open ones, but they also need the options the candidates have to choose from. Each option can be either deleted, marked as correct with the checkbox on the left, or can have its text changed. In addition the company can choose to add more options to the closed question with the *"Add answer..."* button.
Lastly the company can decide to either add questions with the *"Add question..."* button or delete them with the trash can icon on the right.
Once the form has been completed it can be saved or deleted by using the dedicated buttons.

Home page

Figure 19: Home prototype page

The picture [19] shows the prototype of the home page that allows students to see all the available internships posted by various companies.
The section on the left shows all the internships and by interacting with them the student can access a page that shows more details about the post and apply for the internship.
On the right there is a panel that shows the basic data regarding the user that is currently logged in.

Internship creation page

Figure 20: Internship creation prototype page

The picture [20] shows the prototype of the page that allows companies to create and post about an internship.
In this page the company can add all the required information that allows the system to select the suitable students and notify them.

4 Requirement traceability

In this section we are gonna look back at the goals and requirements described in the RASD and map them to the components created in the second chapter of this document. To make it easier to read we wrote again the goals followed by their already mapped requirements and then we added the components.
As we will see each goal is mapped to a single specific component of a subsystem since they were created with this exact purpose to ensure that each component would have a set of task to perform within a single context, providing as much modularity as possible.

G1	**Student can create account**
RA1	The platform shall allow users to register
	Components mapped to the requirements: • Account Management – Student Manager

G2	**Company can create an account**
RA1	The platform shall allow users to register
	Components mapped to the requirements: • Account Management – Company Manager

G2	**Users can update their accounts**
RA3	The platform shall allow users to update their existing account
	Components mapped to the requirements: • Account Management – Student Manager – Company Manager

G4	**Companies can create internships**
RI1	The platform must allow companies to create internships
	Components mapped to the requirements: • Internship Management – Manage Internship

G5	**Students can view all available internships**
RI2	The platform must allow students to view all available internships
RI3	The platform must allow students to search between the available internship
	Components mapped to the requirements: • Internship Management – Manage Internship

G6	**Companies can create forms for students to fill**
RF1	The platform shall allow companies to create custom forms
RF2	The platform shall allow companies to save created forms for later use in an internship selection process
	Components mapped to the requirements: • Internship Management – Form Management

G7	**Recommendation system**
RN1	The platform shall notify students when an internship suitable for them has been posted
	Components mapped to the requirements: • Internship Management – Manage Internship → *findCompatibleStudents()*

G8	**Students can apply for the internships**
RI4	The platform must allow students to apply for any available internship
	Components mapped to the requirements: • Internship Management – Manage Internship

G9	**Form evaluation**
RN7	The platform shall send companies the list of candidates at the end of the application deadline
RF3	The platform shall allow companies to send the candidates that passed the first screening the form for the second screening
RF4	The platform shall allow students to fill and submit the application form
RF5	The platform shall allow companies to evaluate a submitted form by grading each question
	Components mapped to the requirements: • Internship Management – Form Management

G10	**Ranking candidates**
RN2	The platform shall notify students of the result of the first screening
RN3	The platform shall notify students of the result of the second screening
RN4	The platform shall notify students of the final result of the application
	Components mapped to the requirements: • Internship Management – Internship selection → *getRanking()*

G11	**Users can provide feedbacks**
RN5	The platform shall notify students/companies to fill a feedback form after the end of the internship
RR1	The platform shall allow students and companies to write a feedback at the end of an internship
	Components mapped to the requirements: • Internship Management – Feedback

G12	**Complaints management**
RI5	The platform shall allow students/companies involved in an internship to complain about the other party
RN6	The platform shall notify universities whenever a complaint is filed involving one of their students
RR2	The platform shall allow students and companies involved in an internship to write complaints regarding the other party
RR3	The platform shall allow universities to read all complaints that involve their students
	Components mapped to the requirements: • Internship Management – Complaints

5 Implementation, Integration, and Test Plan

This chapter explains how to implement, integrate, and plan the tests for the application designed in this document. The chapter is divided into two fundamental parts. The first one (**Components Development**) describes how the components designed in this document should be implemented, integrated, and tested. The second one (**System Testing**) explains how testing for the system as a whole should be executed.

5.1 Components Development

The system is divided into the following subsystems:

- Web Browser
- Mobile Device
- Account Management
- Internship Management

The first two are designed for the client side of the application, while the latter two handle the server-side functionalities, including data and application layer management, as detailed in Section 2. These subsystems must be implemented, tested, and integrated using a **bottom-up** approach. To ensure development progresses as intended, we provide a prioritized order for development based on dependencies. While client and server-side components can be developed in parallel, final integration and testing will be performed together.

Development Order and Difficulty:

Order	**Stage**	**Difficulty**
1	Account Management	Medium
2	Internship Management	High
3	Web Browser Interface	Low
4	Mobile Device Interface	Medium

Table 1: Development stages and difficulty levels for system components.

Implementation and Testing Plan:

- **Account Management:** Focuses on user authentication, role assignment, and data security. Unit tests will validate functionality, followed by integration tests with the database. Testing scenarios will include edge cases for password complexity, role-based access permissions, and session management.
- **Internship Management:** Handles internship tracking, student assignments, and reporting. This subsystem will undergo rigorous testing, including boundary tests for data handling and stress tests for performance. Additional checks will ensure data integrity during concurrent user operations.
- **Web Browser Interface:** Ensures compatibility across major browsers and responsive design. Functional tests will verify usability and navigation.
- **Mobile Device Interface:** Ensures smooth operation on Android and iOS platforms. Device-specific tests and cross-platform validations will be conducted, covering offline functionalities and seamless synchronization with the server.

5.2 System Testing

System testing involves validating the entire application as an integrated unit to ensure all components function together seamlessly. The following phases will be executed:

- **Integration Testing:** Verifies the correct interaction between subsystems. Test cases will focus on data flow, API interactions, and synchronization between client and server sides, but also to verify the compatibility between the different components that interact with each other as described in section 2.2. Special attention will be given to error-handling mechanisms and fallback operations.
- **Functional Testing:** Ensures the system meets all specified requirements. Scenarios will include user login, internship application workflows, and report generation. Regression tests will be conducted to confirm that new updates do not introduce unintended issues.
- **Performance Testing:** Evaluates system response under various loads. Stress and load tests will simulate peak usage conditions to identify bottlenecks. Metrics such as response time, throughput, and server utilization will be monitored.
- **User Acceptance Testing (UAT):** Conducted with end users to validate the system's functionality, usability, and compliance with expectations. Feedback collected during UAT will be analyzed for final adjustments before deployment.

Test Environment: A staging environment mirroring the production setup will be used for all testing activities. This environment will include:

- Dedicated servers for database and application hosting, configured with realistic data samples.
- Test accounts with varied roles to simulate real-world scenarios, including administrator, student, and external company users.
- Monitoring tools to capture performance metrics and logs, providing detailed insights for debugging and optimization.

Defect Management: Issues identified during testing will be tracked using a defect management tool. Each defect will be prioritized based on its severity and impact on system functionality. Weekly review meetings will ensure progress is monitored, and high-priority defects are addressed promptly.

Deployment Readiness: Before deployment, a final readiness checklist will be verified, including:

- Completion of all planned tests with documented results.
- Resolution of all critical and high-severity defects.
- Approval from stakeholders after successful UAT.

This systematic approach ensures that the application is robust, reliable, and ready for deployment.

6 Effort spent

- **Abdallah Alkhetiar**

Chapter	Effort
1	0 h
2	0 h
3	0 h
4	0 h
5	0 h

- **Daniel Bonardi**

Chapter	Effort
1	0 h
2	0 h
3	0 h
4	0 h
5	0 h

7 References

N-tier architectures
https://dev.to/3bdelrahman
https://blog.nginx.org/blog
https://www.klipfolio.com
OAuth2.0
https://auth0.com