



# **POLITECNICO**

## **MILANO 1863**

### **Design Document**

Software Engineering 2  
Emanuele Cimino - Gabriele Lorenzetti



## Contents

<b>Table of Contents</b>	<b>2</b>
<b>List of Figures</b>	<b>4</b>
<b>List of Tables</b>	<b>4</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Scope	5
1.2 Definitions, acronyms, abbreviations	5
1.2.1 Definitions	5
1.2.2 Acronyms	5
1.2.3 Abbreviations	6
1.3 Revision history	6
1.4 Reference Documents	6
1.5 Overview	6
<b>2 Architectural Design</b>	<b>7</b>
2.1 Overview: high-level components and interactions	7
2.2 Component view	9
2.3 Deployment view	11
2.3.1 Why this approach?	12
2.4 Component interfaces	13
2.5 Runtime view	15
2.6 Selected architectural styles and patterns	21
2.6.1 Tiers and Devices description	21
2.7 Other design decisions	22
<b>3 User Interface Design</b>	<b>23</b>
<b>4 Requirements traceability</b>	<b>27</b>
4.1 Functional Requirements	27

4.2	Non-functional Requirements . . . . .	28
<b>5</b>	<b>Implementation, Integration and test Plan . . . . .</b>	<b>29</b>
5.1	Component priority . . . . .	29
5.2	Integration . . . . .	30
5.3	Build and test plan . . . . .	30
<b>6</b>	<b>Effort Spent . . . . .</b>	<b>32</b>

List of Figures

1	Architectural Design . . . . .	8
2	Component view . . . . .	9
3	Deployment view . . . . .	11
4	Component interfaces . . . . .	13
5	Integration . . . . .	30

List of Tables

1	Requirements . . . . .	27
2	Component priority . . . . .	29
3	Effort Recap . . . . .	32

# 1 Introduction

## 1.1 Scope

Student & Companies is an innovative web platform designed to provide an immersive and visually engaging experience for both students and companies looking for an internship or a student to train.

The main purpose of the platform is to provide a friendly and smart environment in which students can find their way in the nonacademic world and help companies find their best pair to train.

- **Intuitive User Experience (UX):** The platform is crafted to ensure an intuitive and user-friendly experience. Through a visually appealing interface, users will find it easy to navigate, explore new proposed internships, and track their experiences.
- **User-Friendly Tools:** Recognizing the work of both users, the platform plays the role of facilitates the selection process. From finding an available and pairing internship to the handling of the selection process, the design empowers users to improve their chances of making a good choice.

## 1.2 Definitions, acronyms, abbreviations

### 1.2.1 Definitions

- **User:** It is a generic user of the platform, who wants to offer a job (Company), or try to find an offer (Student).
- **Complaint:** It is a form that can be compiled from users to inform their counterparts about problems.
- **Feedback:** It is a form that must be compiled from users to evaluate their fit during and after the internship with their counterparts.
- **Review:** It is a form that must be compiled from users at the end of the internship to leave a comment and evaluate with a score their counterpart.
- **Statistical analysis:** Analysis based on the previous general choices of students, companies and feedback.
- **Filters:** Used by the student to make an advanced search for the apprenticeship, applying the so-called filters, such as minimum salary or company name.

### 1.2.2 Acronyms

- **S&C:** Student and Companies
- **DBMS:** DataBase Management System
- **REST:** REpresentetional State Transfer
- **DTO:** Data Transfer Object
- **DMZ:** Demilitarized zone
- **DDoS:** Distributed Denial of Service

### 1.2.3 Abbreviations

- **[RV<sub>n</sub>]**: It refers to the n-th runtime view provided
- **[R<sub>n</sub>]**: It refers to the n-th requirement

## 1.3 Revision history

- Version 1: 07/01/2025

## 1.4 Reference Documents

- Slides of the course “Software Engineering 2”
- Specification document "01. Assignment RDD AY 2024-2025”
- Unified Modeling Language specifications - <https://www.omg.org/spec/UML/>
- "RASD” by Emanuele Cimino, Gabriele Lorenzetti
- Drawing UML Sequence Diagram by using pgf-umlscd - Yuan Xu

## 1.5 Overview

- **Section 1: Introduction**

This section offers a brief introduction to the document presented here, including all the definitions, acronyms and abbreviations that will be found reading it.

- **Section 2: Architectural Design**

This section offers a more detailed description of the architecture of the system. The first part provides a high-level description of the system, together with a description of its components. Then, the deployment of the system is presented and described in detail. Finally, the main runtime views of the system are exposed and outlined.

- **Section 3: User Interface Design**

This section is useful for graphical designers and contains several concepts of the visual appearance of S&C platform, referring to the client-side experience.

- **Section 4: Requirements Traceability**

This section acts as a bridge between the RASD and DD document, providing a complete mapping of the requirements (functional and non-functional) described in the RASD to the logical modules presented in this document.

- **Section 5: Implementation, Integration and Test Plan**

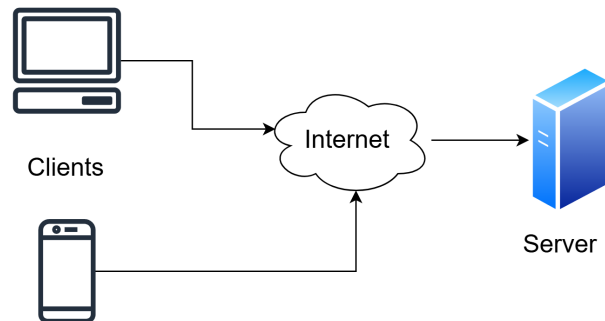
The last section is addressed to the developer team and describes the procedures followed to implement, test and integrate the components of S&C: there will be a complete report about how to implement and test them.

- **Section 6: Effort spent**

Assigns the effort in hours spent by each member of the group on the various activities related to the development of the document.

## 2 Architectural Design

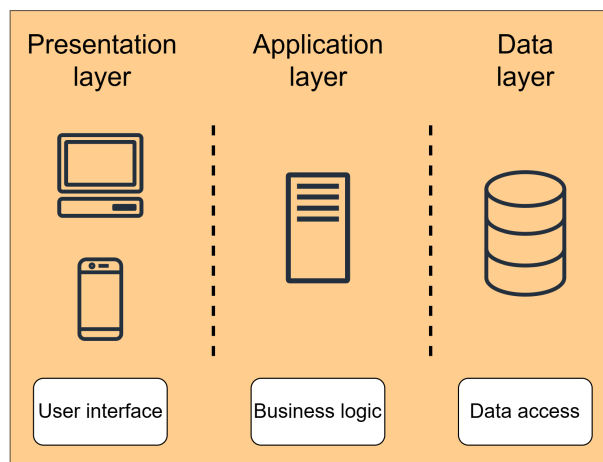
### 2.1 Overview: high-level components and interactions



S&C is a web application that follows the commonly known client-server paradigm. In particular, the client is thin, which means that it does not manage the application business logic, but only the presentation layer. The server, instead, is fat and contains all the data management and business logic.

The system consists of multiple layers responsible for specific functions:

- **Presentation Layer:** focuses on the user interface and interaction.
- **Application Layer:** manages the business logic and data flow between the presentation and data layers.
- **Data Layer:** manages data persistence and database access.



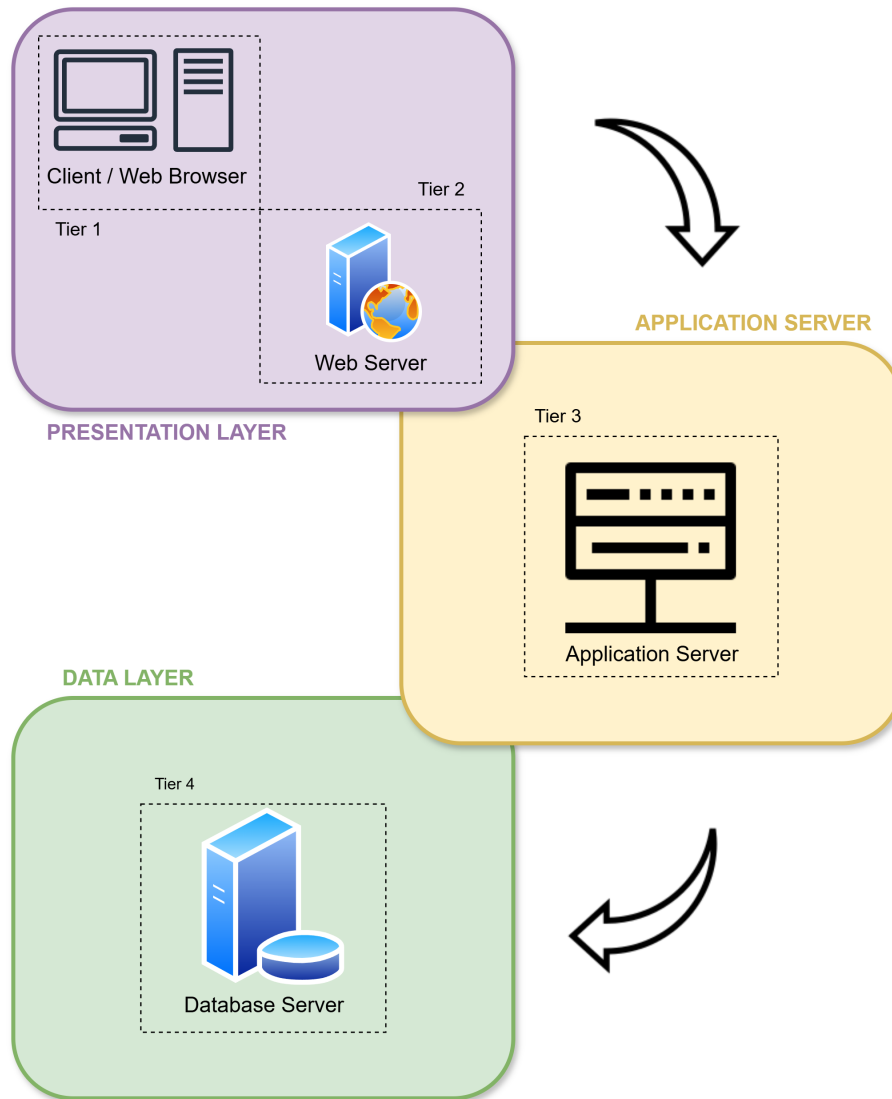


Figure 1: Architectural Design

The system will be designed with a 4-tier architecture (two for the presentation layer, one for the application layer and one for the data layer, for further details see section 2.6.1) with the aim to focus on the following goals:

- **Adaptable Scalability:** With more tiers, it is easier to vertically scale the system, aligning with the growth of the user population of the application.
- **Flexible technology:** Every node can use his own specific technology, without influencing the others. Moreover, this kind of architecture simplifies the upgrade process, which is a valuable attribute considering the nature of the platform.
- **Simplified Maintainability:** The division of responsibilities simplifies maintenance. For example, changes at the presentation level can be handled independently of changes to the business logic or data level.
- **Performance Boost:** The distribution of functionalities across multiple layers enhances performance by allowing each layer to be optimized independently from the others.



## 2.2 Component view

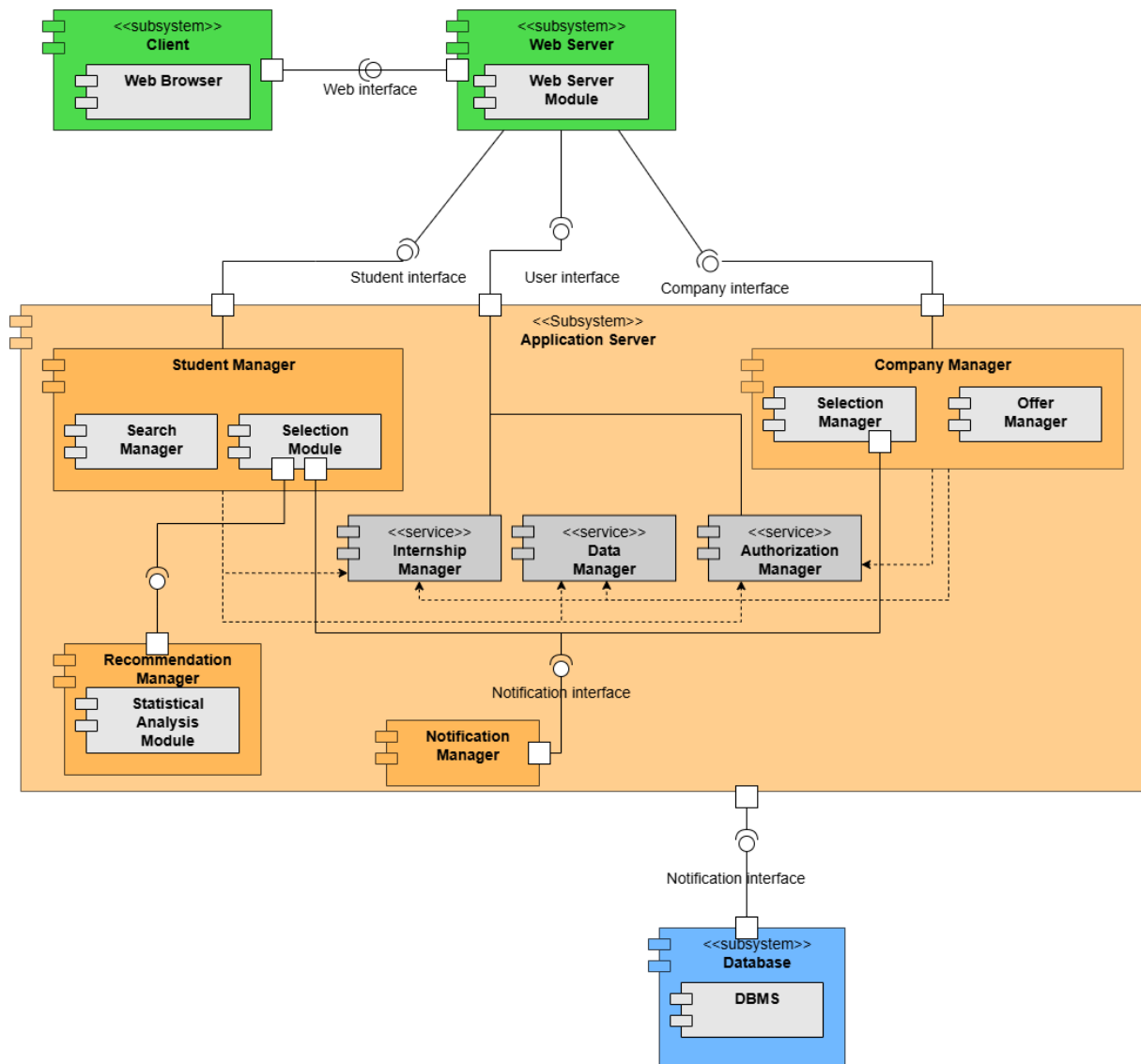


Figure 2: Component view

- **Authorization Manager**

This component handles requests regarding authentication. This includes signing up and logging in. Moreover, the aim of this component is to manage authorizations to prevent users from forwarding requests they are not allowed to.

- **Company Manager**

The Company Manager serves as the central hub for handling requests from the company. It includes the following sub-components:

- **Selection Manager**

This sub-component is dedicated to overseeing aspects of the selection phase. Such as sending forms and checking expiry dates with the automatic rejection of the student in case of such.

- **Offer Manager**

This sub-component handles requests related to offer management. This includes the creation

of offers with related forms and expiration dates.

- **Student Manager**

The Student Manager is designed to handle requests specifically from students. It includes the following sub-components:

- **Search Manager**

This sub-component is dedicated to the searches that the student can do: by scrolling, with the search bar and applying various filters.

- **Selection Module**

Dedicated to selection-related activities, like filling out forms.

- **Notification Manager**

The Notification Manager serves as the centralized component responsible for sending notifications to users in response to various events, such as the various forms and acceptances or other relevant information. This component ensures timely and effective communication with users.

- **Recommendation Manager**

It takes care of managing the recommendation function thanks also to the Statistical Analysis Module, for both students and companies.

- **Internship manager**

It takes care of managing all the functions related to the internship development.

- **Data Manager**

It takes care of modify data on user request.

## 2.3 Deployment view

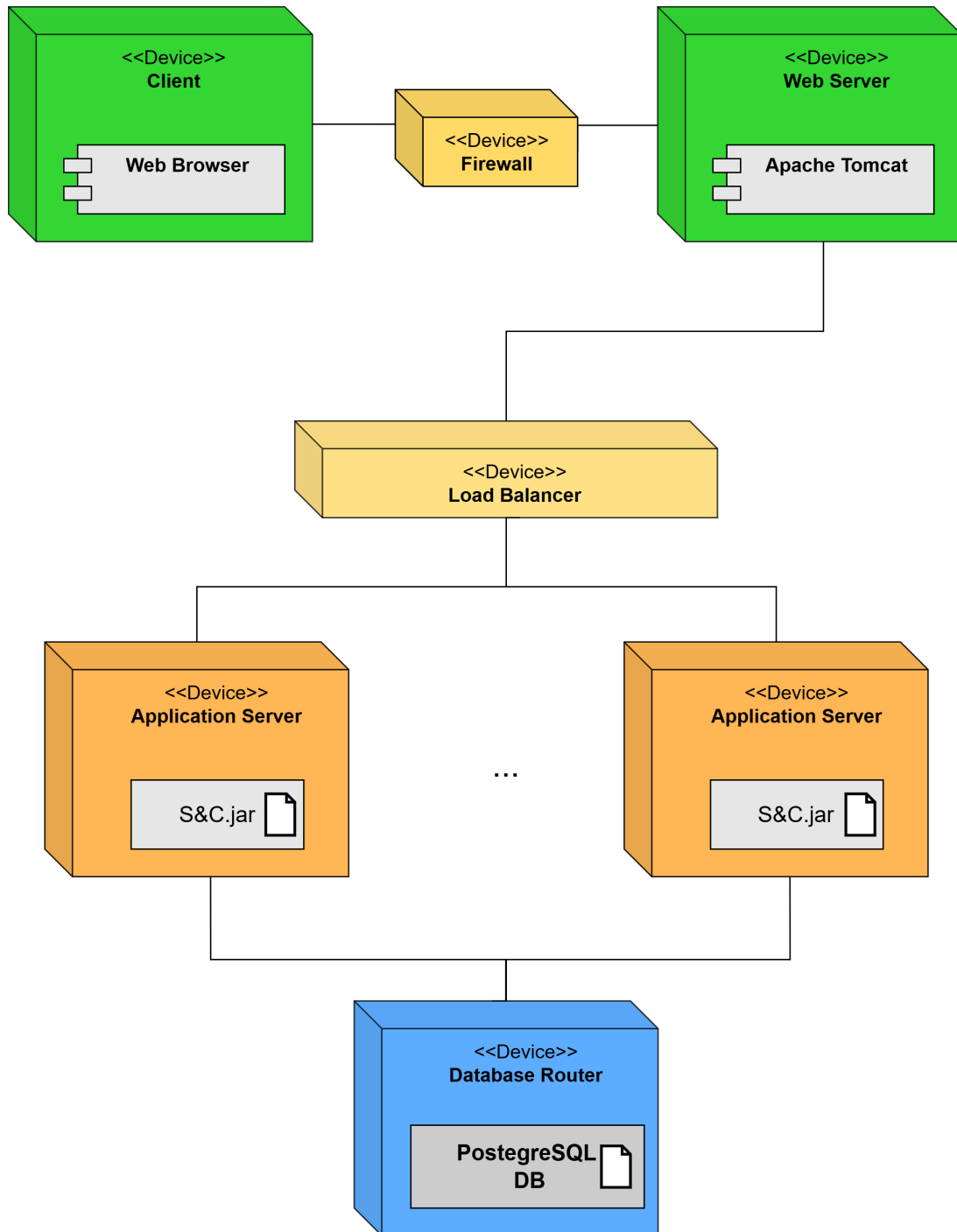


Figure 3: Deployment view

This diagram provides a detailed view of the physical architecture of the system, depicting how the nodes interact and position themselves.

### 2.3.1 Why this approach?

- In the Component view, the main components of the system have been described, with a deep description of the roles they have inside of the application server. We can see that it is not in charge of many computational tasks, a lot of them are also carried out by the DBMS, since they are reading/writing operation. Consequently, the communication-related timing, i.e. the time taken to perceive a request and deliver a response, prevails over the pure computational time, i.e. the time taken to fulfill that request.
- The idea is to deploy a single machine in charge of managing the entire logic of the application, processing requests coming from the Web Server and querying the Database Server to answer them. Thanks to the single machine in charge, the cost of the initial deployment is limited since the number of machines to be bought and installed is lowered: this could help either in delivering a prototype and in observing the growth of the user base.
- Vertical scalability (scale-up) is facilitated: the system can be easily strengthened vertically by upgrading the single machine of the Application Server. This could be a good compromise to follow the initial growth of the platform with reduced costs.
- To fulfill the potential expansion of the platform, the microservices-approach can be reconsidered in the future. In this case, the application layer of the system would be split into different servers, each managing a different area of the system logic. The most reasonable deployment for S&C as a microservices-based system might consist in assigning each main component of the Application Server (Search Manager, Selection Module, Selection management, Offer Management, Internship Manager and Authorization Manager) to a different machine. Communication between different components would be achieved by implementing additional endpoints, according to the RESTful paradigm. In this way the system would be scaled out and optimized.

## 2.4 Component interfaces

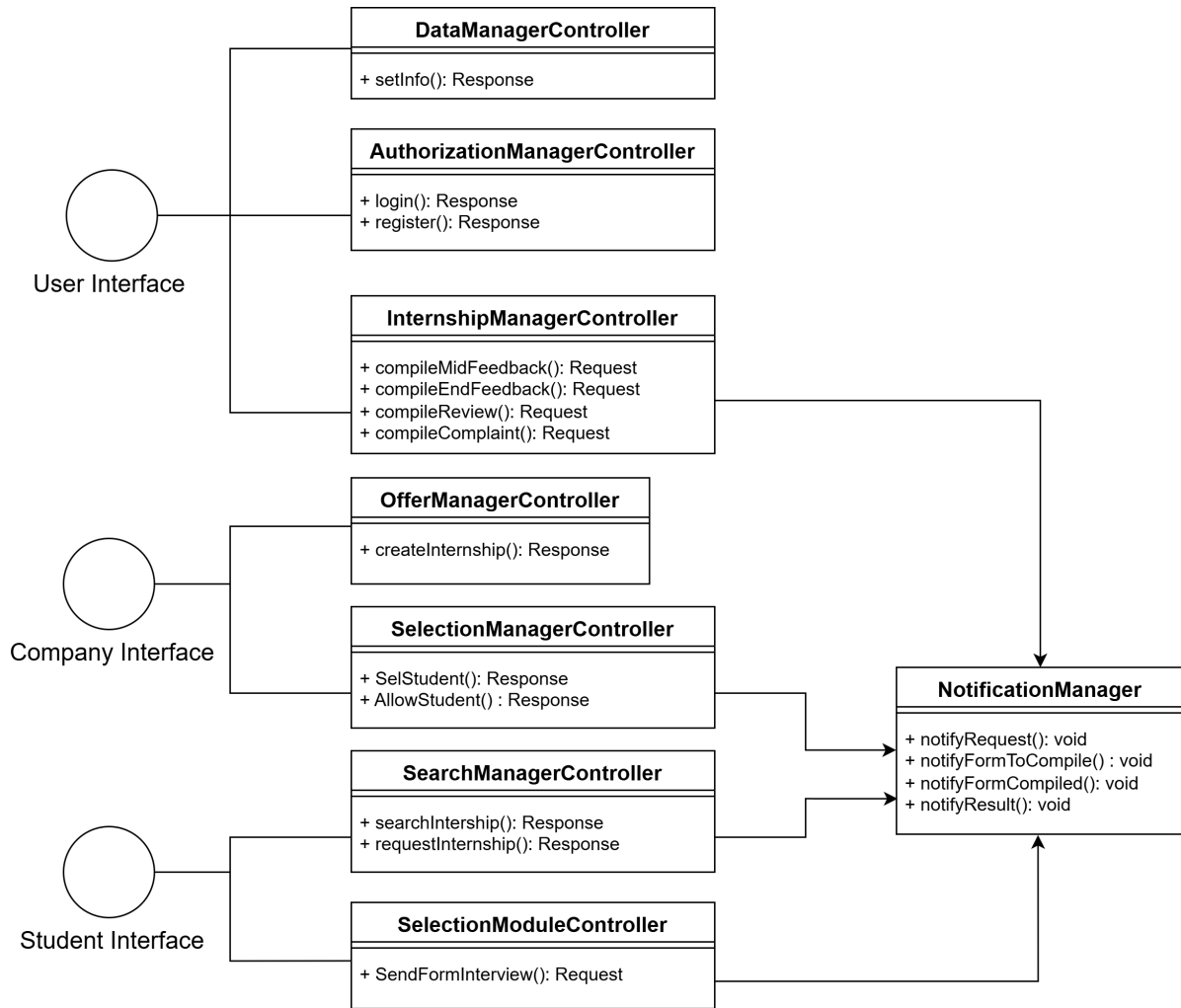


Figure 4: Component interfaces

- **User Interface**

This interface provides access to generic user-related functionalities that are common to both students and companies. It is managed by the *AuthorizationManagerController*, which is responsible for handling log-in and registration, and by the *InternshipManagerController*, which is responsible for internship-related actions to fulfill. Users information are requested or sent through methods returning or sending serialized objects or strings as responses.

- **Company interface**

This interface provides access to company-related functionalities. They are handled by the *OfferManagerController* that handles the creation of new internships by a company and also the selection management by a company through the *SelectionManagerController*.

- **Student interface**

This interface provides access to student-related functionalities. They are handled by the *SearchManagerController* that handles the search functions to find a suitable internship and also the selection module to help a student retrieve information about a requested/recommended internship. *SelectionModuleController*

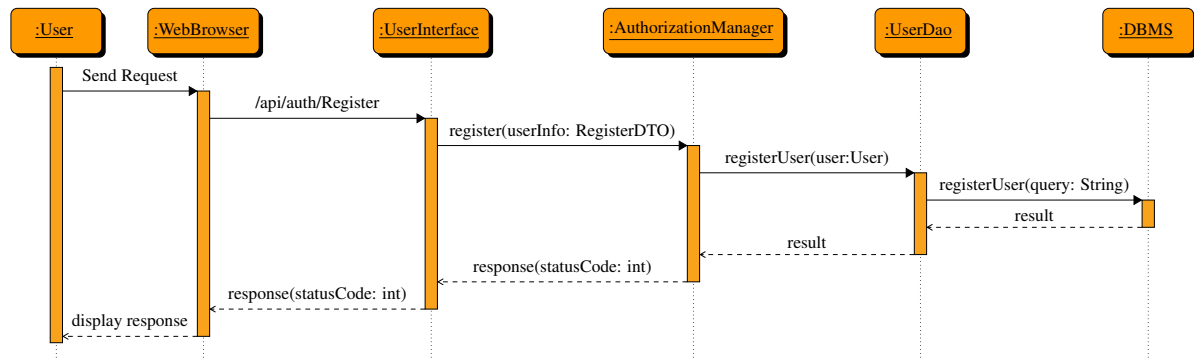
- **Notification management**

The *NotificationManager* is there to help giving a common notification interface to the various actions that happens in the system during its all functioning.

## 2.5 Runtime view

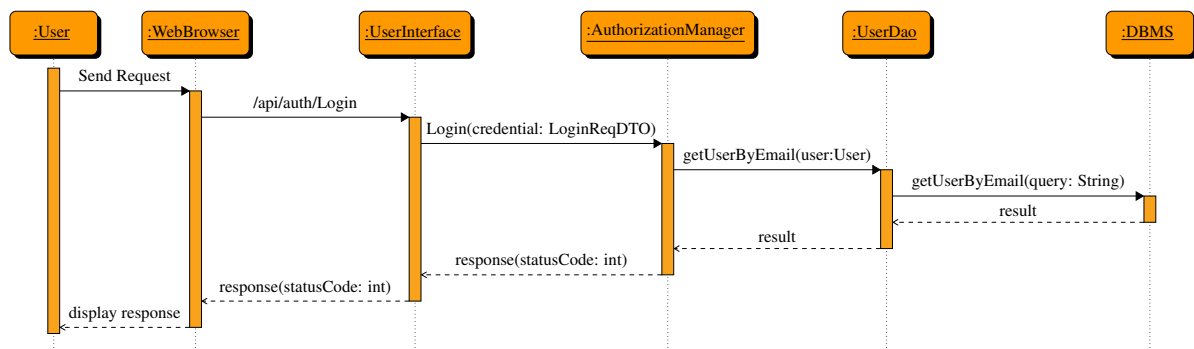
In this section, all the runtime views corresponding to the use cases in the RASD are represented and described.

### [RV1]: Register

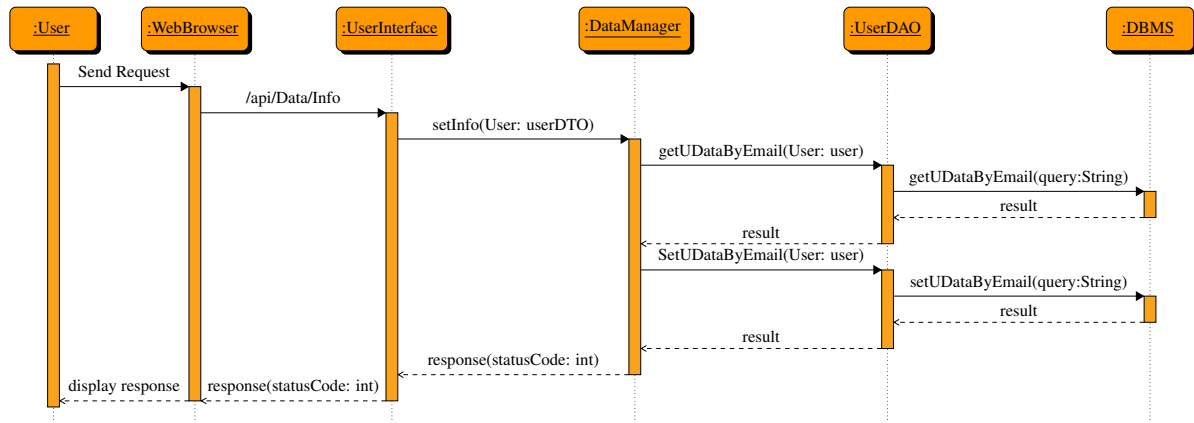


Initially, users enter their information via a dedicated form and use the Web Browser to send a request to the server. This request is received by the User Interface and managed through the *register()* method within the Authorization Manager component. This method will attempt to store the new user's data in the database through the UserDao. If the operation is successful, the user will receive a response with a 200 status code. Otherwise, if the provided email already exists in the database, the user will receive a response with a 422 status code.

### [RV2]: Login



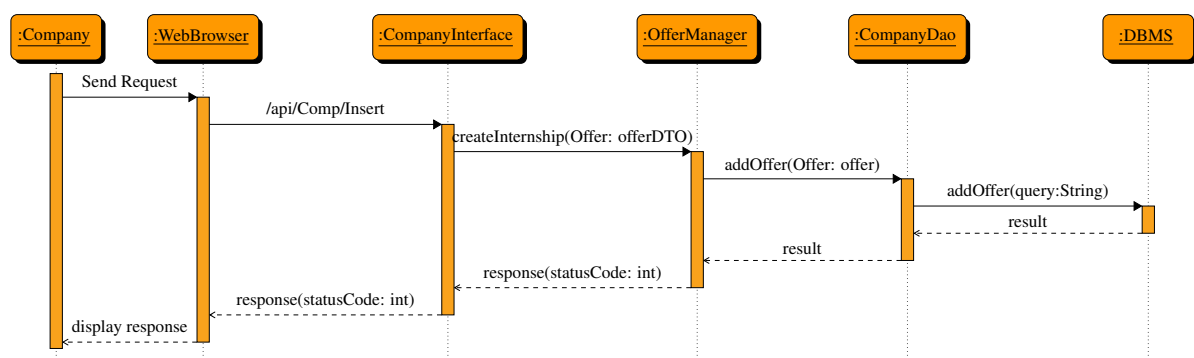
Initially, the user enters their credentials via a dedicated form and uses the Web Browser to send a request to the server. This request is received by the User Interface and managed through the *login()* method within the Authorization Manager component. This method will verify the correctness of the credentials by checking the database through UserDao. If the operation is successful, the user will receive a response with a 200 status code. Otherwise, if the credentials are incorrect, the user will receive a response with a 422 status code.

**[RV3]:** Modify Info

Here a user modifies his/her personal information through the system. The process begins when the user sends a request through the browser to the user interface, using the `/api/Data/Info` endpoint. The user interface handles the request by invoking the `DataManager`'s `setInfo()` method, providing the new data to update.

Before making any changes, the `DataManager` retrieves the user's current information. To do so, it invokes the `UserDAO`'s `getUDataByEmail()` method, which consults the database through the `DBMS`. Once the existing data is obtained, the `DataManager` starts updating the data by calling the `UserDAO`'s `SetUDataByEmail()` method, transmitting the new data. The `UserDAO` then updates the database using the `DBMS`.

Once the update is complete, the result is returned from the `DBMS` to the `UserDAO`, then to the `DataManager`, and finally to the user interface. The process ends by sending a response to the user's browser, which indicates the outcome of the operation through a status code: 200 if the operation is successful, 422 if there is an error.

**[RV4]:** Create an internship

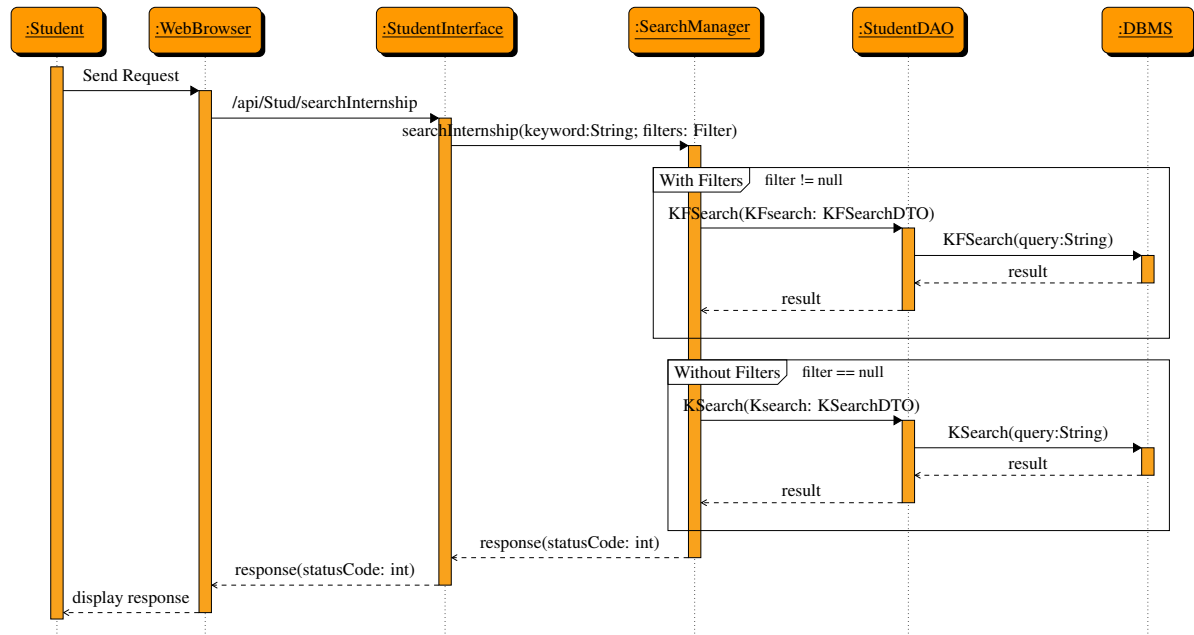
The process starts with the company sending a request through the browser, accessing the `/api/Comp/Insert` endpoint. The request is forwarded to the `CompanyInterface`, which handles the communication with the application layer. The `CompanyInterface` calls the `OfferManager`'s `createInternship()` method, passing the details of the internship offer.

The `OfferManager` is responsible for the offer management logic. It calls the `addOffer` method of the `CompanyDao` to save the offer data. The `CompanyDao` translates this operation into a SQL query and sends it to the `DBMS`, which records the offer in the database. After completing the operation, the result is returned from the `DBMS` to the `CompanyDao`, then to the `OfferManager`, and finally to the



## CompanyInterface.

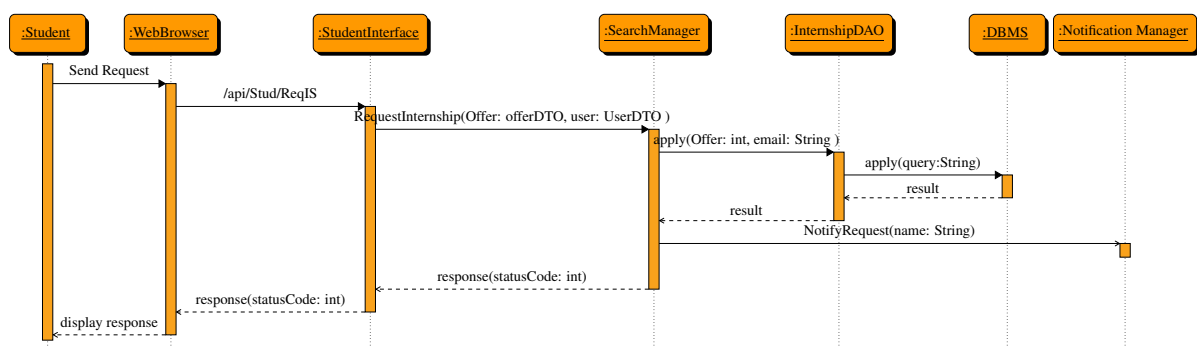
At the end of the process, the CompanyInterface sends a response to the company's browser, indicating the outcome of the operation via a status code: 200 if the operation is successful, 422 if there is a problem.

**[RV5]:** Searching an internship

The process starts when the student sends a search request through the browser, using the /api/Stud/searchInternship endpoint. The request is forwarded to the StudentInterface, which handles communication with the application layer. The StudentInterface then calls the SearchManager's *searchInternship()* method, passing the search keyword and, if any, filters.

If filters are specified, the SearchManager performs a filtered search (KFSearch) by invoking the corresponding method of the StudentDAO. This constructs a SQL query and sends it to the DBMS, which returns the results. Otherwise, the SearchManager performs a simple search (KSearch), following the same flow but without considering the filters.

Once the search is complete, the results are returned from the DBMS to the StudentDAO, then to the SearchManager, and finally to the StudentInterface, which sends them to the student's browser. The browser then displays the search results, allowing the student to explore them.

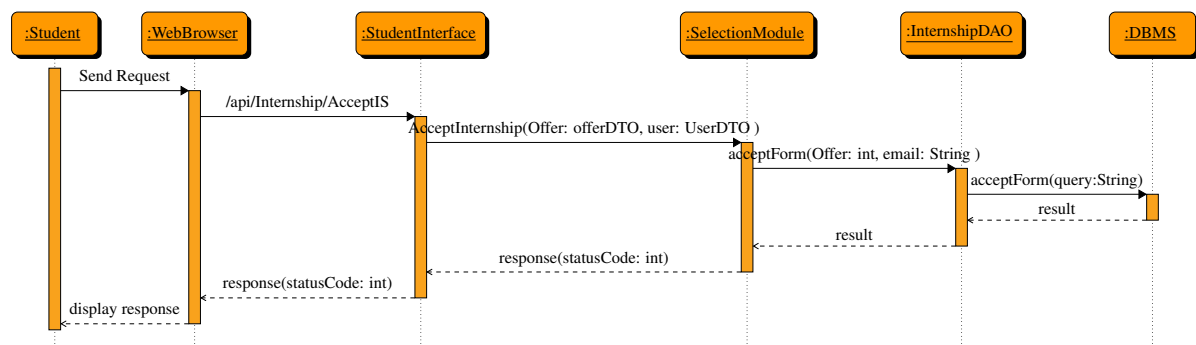
**[RV6]:** Applying for an internship

The process begins with the student sending a request through the browser, using the `/api/Stud/ReqIS` endpoint. The browser forwards this request to the `StudentInterface`, which handles communication with the application layer. The `StudentInterface` then calls the `SearchManager`'s `RequestInternship()` method, providing the internship offer ID and the student's email.

The `SearchManager` handles the application logic and calls the `InternshipDAO`'s `apply` method to record the application details. The `InternshipDAO` translates this into a SQL query and sends it to the `DBMS`, which records the application in the database. The result of this operation is then returned from the `DBMS` to the `InternshipDAO` and then to the `SearchManager`.

Once the registration is complete, the `SearchManager` sends a notification to the `Notification Manager` via the `NotifyRequest()` method, to inform the company of the new application. Finally, a response containing the status code is sent from the `StudentInterface` to the browser, allowing the student to view the outcome of the operation: 200 if the operation is successful, 422 if there is an error.

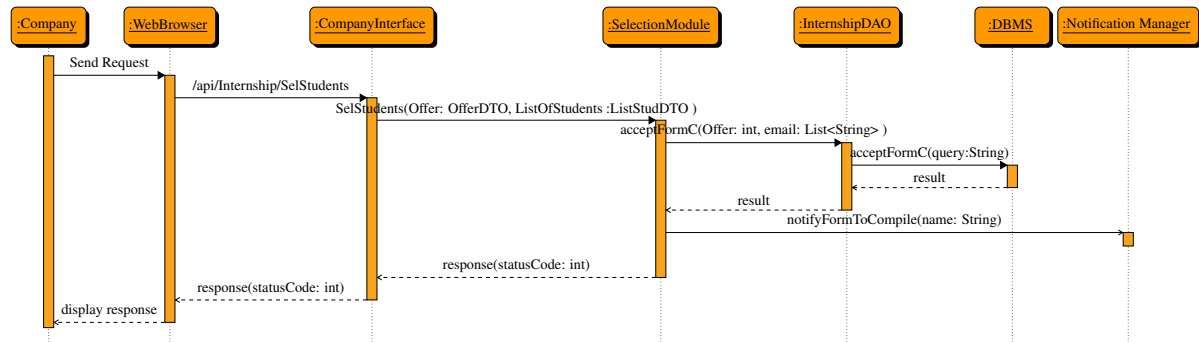
**[RV7]:** Student accept an internship



The process begins with the student sending a request through the browser using the `/api/Internship/AcceptIS` endpoint. This request is forwarded to the `StudentInterface`, which handles communication with the application layer. The `StudentInterface` calls the `AcceptInternship()` method of the `SelectionModule`, passing the offer identifier and the student's email.

The `SelectionModule` processes the request and calls the `acceptForm` method of the `InternshipDAO` to record the acceptance of the offer. The `InternshipDAO`, in turn, constructs a SQL query and sends it to the `DBMS`, which updates the accepted offer data in the database. Once the operation is complete, the result is returned by the `DBMS` to the `InternshipDAO`, then to the `SelectionModule`, and finally to the `StudentInterface`.

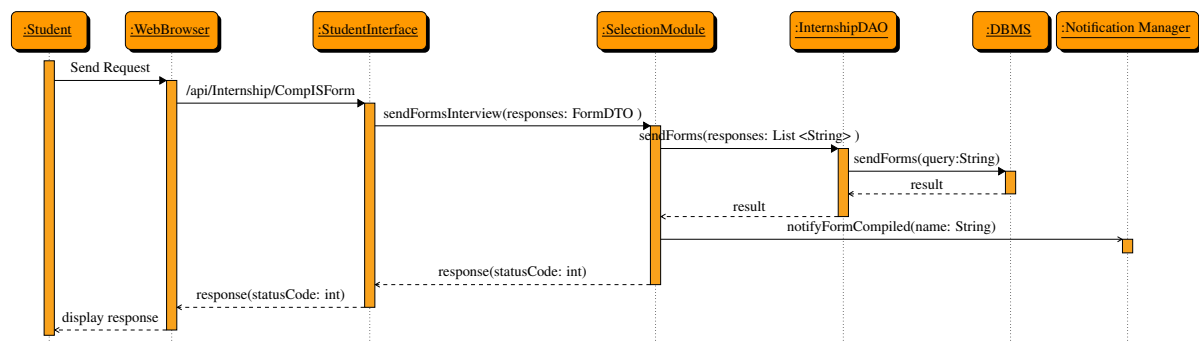
At the end of the process, the `StudentInterface` sends a response to the student's browser containing the status code: 200 if the operation is successful, 422 if there is an error.

**[RV8]:** Company select an internship and accepts some trainee

After viewing the candidates, the company sends a new request to the `/api/Internship/SelStudents` endpoint. This request is forwarded to the `CompanyInterface`, which calls the `SelStudents()` method of the `SelectionModule`, passing the offer ID and the list of selected students.

The `SelectionModule` processes the selection and calls the `InternshipDAO`'s `acceptFormC()` method to update the database with the accepted candidates. The `InternshipDAO` builds a SQL query and executes it through the `DBMS`. Once the operation is completed, the result is propagated back to the browser.

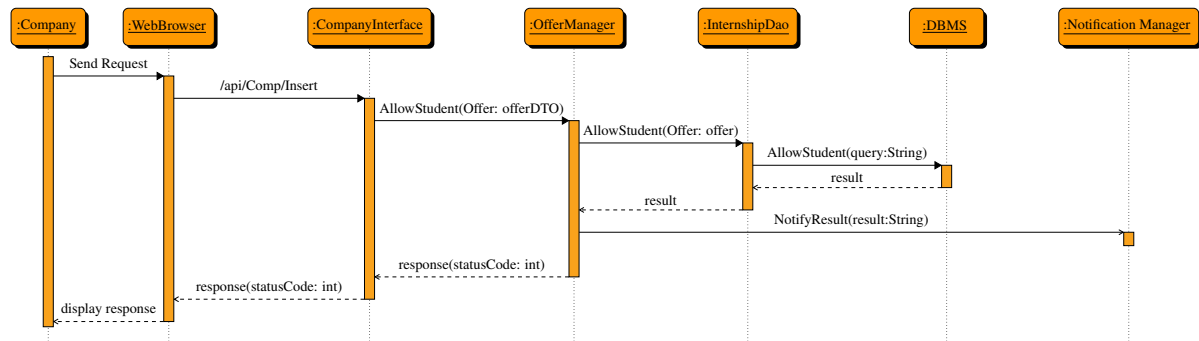
Finally, the `SelectionModule` call the `Notification Manager` to inform the students, ensuring that they are updated on the company's decision.

**[RV9]:** Student compile a form

The process begins when the student sends a request via the browser to the `/api/Internship/CompISForm` endpoint. The request is forwarded to the `StudentInterface`, which handles communication with the application layer. The `StudentInterface` calls the `SelectionModule`'s `sendFormsInterview()` method.

The `SelectionModule` handles the logic for sending the answers and calls the `InternshipDAO`'s `sendForms()` method to store the data in the database. The `InternshipDAO` constructs a SQL query and sends it to the `DBMS`, which stores the answers in the database. After completing the operation, the result is returned from the `DBMS` to the `InternshipDAO`, then to the `SelectionModule`, and finally to the `StudentInterface`.

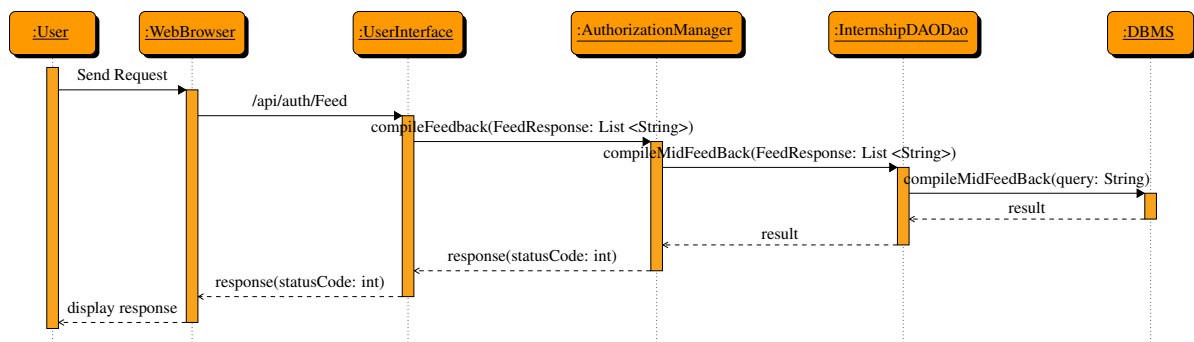
Once the data has been successfully stored, the `SelectionModule` call the `Notification Manager` via the `notifyFormCompiled` method, to inform the company that the student has completed the form. Upon completion, a response containing the status code is sent from the `StudentInterface` to the browser, allowing the student to view the outcome of the operation: 200 if the operation is successful, 422 if there is an error.

**[RV10]:** Company see the form and accept trainee

The process begins when the company sends a request through the browser, using the `/api/Comp/Insert` endpoint. The request is forwarded to the `CompanyInterface`, which handles communication with the application layer. The `CompanyInterface` invokes the `OfferManager`'s `AllowStudent()` method, passing an `offerDTO` object containing the offer and authorization details.

The `OfferManager` processes the request and calls the `InternshipDAO`'s `AllowStudent()` method, which constructs a SQL query to record the authorization in the database. The `InternshipDAO` sends the query to the `DBMS`, which stores the authorization information. Once the operation is complete, the result is returned from the `DBMS` to the `InternshipDAO`, then to the `OfferManager`, and finally to the `CompanyInterface`.

At the end of the process, the `CompanyInterface` sends a response to the company's browser, indicating the outcome of the operation via a status code: 200 if the operation is successful, 422 if there is an error.

**[RV11]:** User sends a feedback

The process begins when the user sends a request through the browser to the `/api/auth/Feed` endpoint. This request is forwarded to the `UserInterface`, which handles communication with the application layer. The `UserInterface` calls the `compileFeedback()` method of the `AuthorizationManager`, passing a list of responses provided by the user as part of the feedback.

The `AuthorizationManager` is responsible for the logic for processing the feedback. It calls the `compileFeed()` method of the `InternshipDAO`, which constructs a SQL query to store the feedback in the database. The `InternshipDAO` sends the query to the `DBMS`, which records the feedback. Once the operation is complete, the result is propagated from the `DBMS` to the `InternshipDAO`, then to the `AuthorizationManager`, and finally to the `UserInterface`.

Finally, the `UserInterface` sends a response to the user's browser, indicating the success of the operation via a status code: 200 if the operation is successful, 422 if there is an error.

The collection of information part is always the same for a company for every step of the internship, only change the function.

## 2.6 Selected architectural styles and patterns

- **4-tiered architecture**

The system has been designed in accordance with the 4-tier architecture, therefore it is divided into 4 different tiers. This architectural choice is aimed to confer flexibility to the system and to facilitate scalability, maintainability and improve performance.

- **RESTful architecture**

The RESTful architecture adopted by S&C establishes a framework for structuring and accessing system resources, facilitating interactions between clients (e.g., web browsers) and the server via a client-server model. This architecture prioritizes stateless communication, where the server maintains no information about the state of client, enhancing scalability and reliability.

Moreover, servers have the ability to temporarily extend or customize client functionality by transferring software code to the client (code on demand), reducing the computational load on the servers. For instance, when filling out a registration form on a website, the browser instantly highlights any mistakes made, such as an incorrect email. Additionally, through client-side scripting, all page updates and requests occur on the client-side, improving user experience by allowing more interactivity.

- **Model-View-Controller**

Model-view-controller (MVC) is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements. These elements are the internal representations of information (the model), the interface (the view) that presents information to and accepts it from the user, and the controller software linking the two.

- **Thin client and fat server**

In this architectural paradigm, the emphasis lies in distributing the workload and responsibilities between the client and the server.

This architectural style gives significant computational load and operational responsibilities to the server, which contains the application's business logic, while the client focuses on rendering user interfaces and forwarding user requests to the server. This means that the client doesn't require high computational power which makes the application more accessible to various devices.

### 2.6.1 Tiers and Devices description

As previously mentioned in 2.1, the system adopts a 4-tier architecture which are described below. We also add the description of the firewall and the load balancer.

- **Tier 1. Client**

The first tier is dedicated to the client, here is represented the user interface and the interaction with the application, which is performed via the web browser. It is users entry point users to access and interact with the platform. It includes client-side logic and visual elements that make up the front-end of the application.

- **Firewall**

It is placed between the Client (Tier 1) and the Web Server (Tier 2) and protects the system network from external threats coming from the Internet (such as DDoS attacks etc.). In this way, the system exploits a DMZ approach and ensures security and privacy to the users of the platform.

- **Tier 2. Web server**

The second tier is represented by the Web Server, strategically handling incoming requests from clients and managing the dynamic aspects of the application. Together with the Firewall, it plays

a key role in ensuring the security of the system: it manages the access to the platform, thus preventing users from exploiting functionalities they are not allowed to.

The diagram shows a single instance of the Web Server but, alongside with a possible evolution of the platform, multiple instances of it can be deployed in order to fulfill a bigger load of requests.

- **Load balancer**

It is placed between the Presentation Layer and the Application Layer and it has the duty of dispatching, forwarding and distributing the requests coming from the web server to the multiple instances of the Application Server.

Its primary purpose is to optimize resource utilization and improve system performance, but it is also responsible for guaranteeing the availability of the system (especially of the Application Layer): it enhances fault tolerance by automatically rerouting traffic away from failed or unhealthy servers.

- **Tier 3. Application server**

The third tier is dedicated to the Application Server, the core of the system. It contains all the logic of the application, including its main functionalities and features, and realizes S&C experience.

As shown in the 2 diagram, multiple instances of the entire platform business logic are deployed on different machines in order to supply a potentially high throughput of incoming requests. The load balancer helps to make it possible thanks to the routes that connects them to the correct server, according to its specific strategy.

It is an horizontal scalability kind of deployment, is very suitable for following a rapid increase in the platform user base: when a server reach is maximum load of requests, a new Application server can be deployed with little effort in the load redistribution.

- **Tier 4. Database server**

The core of data management, the Database server, that oversees the storing, managing and updating of the platform, is the fourth and last tier of the system deployment.

In the case of S&C, most of the functions of the platform are simple read/write operations on data, which are carried out by the DBMS itself. For this reason, it is of paramount importance to have the possibility to perform a very large number of transactions at a time.

Another aspect to be considered is the potential amount of data as a consequence of the growth of the platform. The Database Server should be able to provide a considerable capacity to store all the information effortlessly.

Moreover, it is important to guarantee a high level of availability of data and to assure no losses of information, such that users can rely on their correctness and completeness.

Considering all these aspects, it goes without saying that the Database Server should employ a good partitioning and replication system.

## 2.7 Other design decisions

- **OAuth2.0**

OAuth 2.0 is the industry-standard protocol for authorization. It focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.

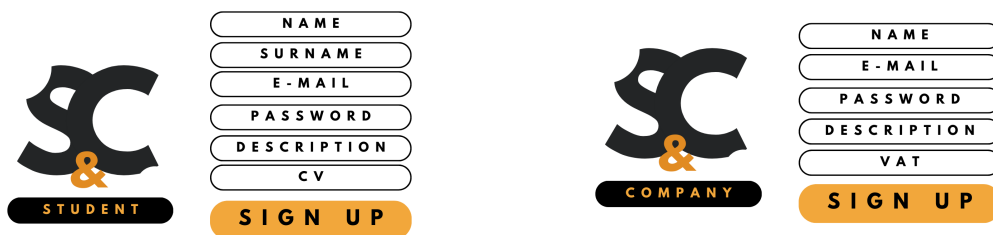
In the context of S&C, OAuth 2.0 is adopted to guarantee that each user type gains access solely to functionalities designed for their role. This ensures that a company, for example, can access and utilize features specifically meant for Company, maintaining strict control over who accesses what within the platform.

### 3 User Interface Design

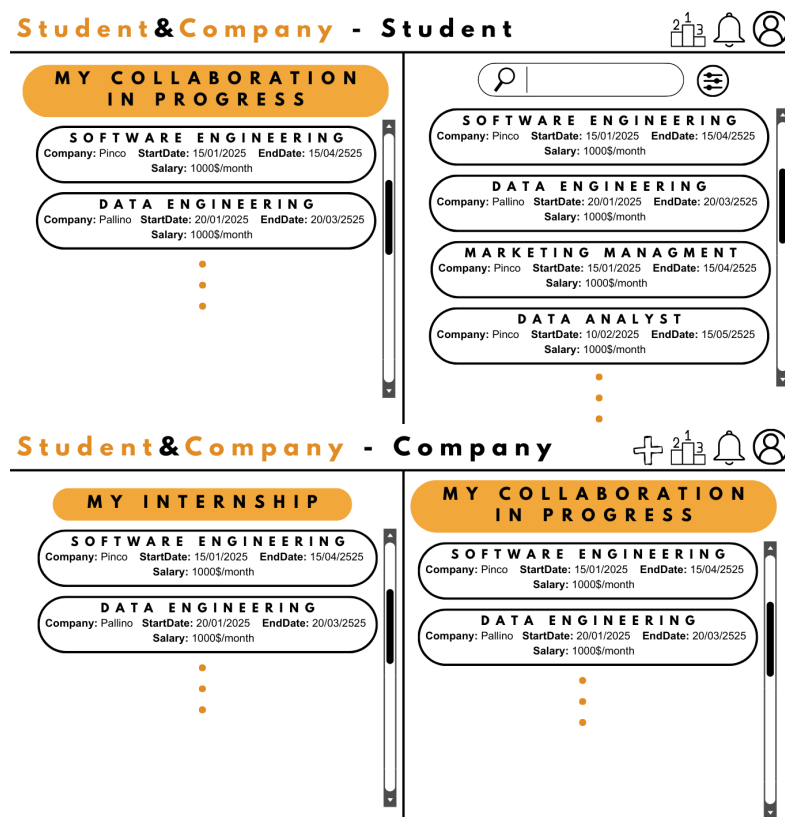
The purpose of this section is to show the design concepts of the main pages of S&C platform, also describing the flow from one to another according to user input.



The sign-in is the same for both students and companies, while the sign-up is different and therefore the choice is shown.







The first image shows the student's sign-up, while the second one shows the company's sign-up.



The first image shows a student's home page: on the left he/she can find the internships that have not yet started or in progress, while on the right he/she can find the list of available ones and the search bar with the filter icon next to it.

The second image shows the home page of a company: on the left there is a list of its internships that have not yet started, while on the right there is a list of those that have started.



**Student & Company - Company**    




**MY INTERNSHIP**

**SOFTWARE ENGINEERING**  
Company: Pinco StartDate: 15/01/2025 EndDate: 15/04/2525  
Salary: 1000\$/month

**DATA ENGINEERING**  
Company: Pallino StartDate: 20/01/2025 EndDate: 20/03/2525  
Salary: 1000\$/month

**ADD A INTERNSHIP**  
Name: ...  
StartDate: ././.  
EndDate: ././.  
Salary: \$  
Qualification required: form  
Interview: form  
Description: form  
**ADD**

By clicking on the "+" icon and filling in the relevant fields, the company can add a new internship pressing the 'ADD' button at the end.




**PROFILE**  
Public  
Private

EndDate: 15/04/2525

By clicking on the profile icon the student can access the private and public page of his/her profile.

In the private one, visible only to the student. on the left he/she can modify his/her information and on the right he/she can see the reviews him/her have left so far and the complaints received.




The public page is visible to both the student and the companies, on the left you can see the non-sensitive information and on the right the reviews received from companies.

**Student & Company - Student**   

**Private page**

NAME  
SURNAME  
E - MAIL  
PASSWORD  
DESCRIPTION  
CV

**My review**  
1 Company: company\_name  
2 Company: company\_name  
**Complaints received**  
1 Company: company\_name

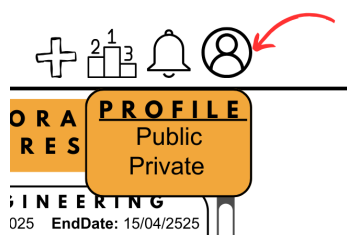
**Student & Company - Student**   

**Public page**

NAME  
SURNAME  
E - MAIL  
DESCRIPTION  
CV

**Review received**  
1 Company: company\_name  
2 Company: company\_name





By clicking on the profile icon the company can access the private and public page of its profile.

In the private one, visible only to the company, on the left it can modify its information and on the right it can see the reviews it has left so far and the complaints received.

The public page is visible to both the student and the companies, on the left you can see the non-sensitive information and on the right the internships with open applications and reviews received from students.

## Student & Company - Company

### Private page

NAME

E - MAIL

PASSWORD

DESCRIPTION

VAT

**My review**

1 Student: student\_name

2 Student: student\_name

**Complaints received**

1 Student: student\_name

## Student & Company - Company

### Public page

NAME

E - MAIL

DESCRIPTION

VAT

**Internships available**

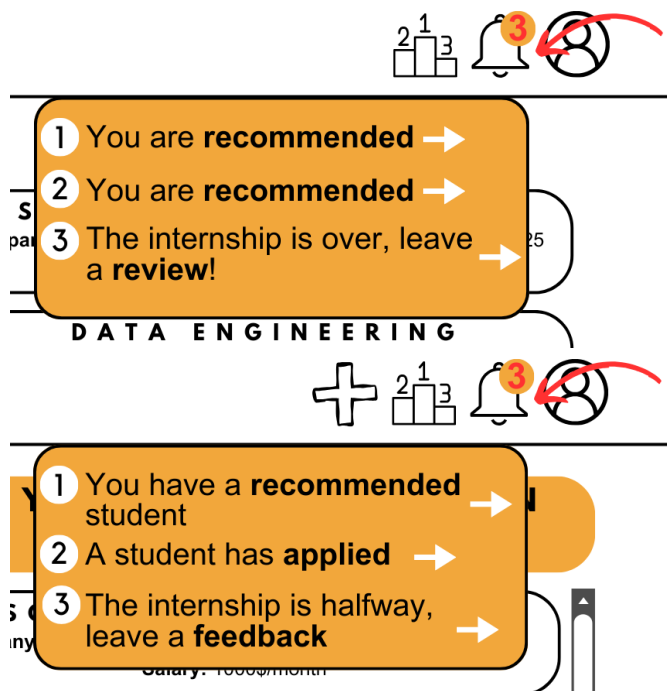
**SOFTWARE ENGINEERING**

Company: Pinco StartDate: 15/01/2025 EndDate: 15/04/2525 Salary: 1000\$/month

**Review received**

1 Student: student\_name

2 Student: student\_name



These two images show what the notifications look like after clicking on the bell icon, for the student and the company respectively.

The number on the icon shows how many unread notifications there are.

**Student&Company - Student**   

## RANKED LIST OF COMPANY

1 Company_name	2 Company_name	3 Company_name
4 Company_name	5 Company_name	6 Company_name
7 Company_name	8 Company_name	9 Company_name
⋮		

**Student&Company - Company**   

## RANKED LIST OF COMPANY

1 Company_name	2 Company_name	3 Company_name
4 Company_name	5 Company_name	6 Company_name
7 Company_name	8 Company_name	9 Company_name
⋮		

By clicking on the podium icon, students and companies can see the ranked list of companies.

## 4 Requirements traceability

### 4.1 Functional Requirements

This section shows the traceability between requirements and components described in component diagram. It highlights which components are responsible for the realization of each requirement.

Requirements	Component
<b>R1:</b> The S&C platform allows users to register	Authorization Manager
<b>R2:</b> The S&C platform allows users to login using their credential	Authorization Manager
<b>R3:</b> The S&C platform allows users to manage their data and modify them	Data Manager
<b>R4:</b> The S&C platform allows companies to insert an internship and the relative interview form	Company Manager: Offer Management
<b>R5:</b> The S&C platform allows users to see others public profiles	Search manager
<b>R6:</b> The S&C platform should provide the "search internship" functionality to students	Student Manager: Search Management
<b>R7:</b> The S&C platform should use the recommendation system to find a match between student and companies	Recommendation Manager
<b>R8:</b> The S&C platform should notify users when a match is found	Notification Manager
<b>R9:</b> The S&C platform should allows student to send the request for a internship	Student Manager: Selection Module
<b>R10:</b> The S&C platform should notify companies when a student requests for a internship	Notification Manager
<b>R11:</b> The S&C platform should allows users to accept or not the match made by the platform	Student Manager: Selection Module
<b>R12:</b> The S&C platform should allows users to terminate the selection process with a rejection	Student Manager: Selection Module / Company Manager: Selection Manager
<b>R13:</b> The S&C platform should send to the students a form related to the chosen internship	Internship Manager
<b>R14:</b> The S&C platform checks all the deadlines	Internship Manager
<b>R15:</b> The S&C platform should allows users to leave feedback in the middle and at the end of internship	Internship Manager
<b>R16:</b> The S&C platform should allows users to leave complaints during the internship	Internship Manager
<b>R17:</b> The S&C platform should allows users to leave a review after the end of the internship	Internship Manager

Table 1: Requirements

## 4.2 Non-functional Requirements

For what concerns system attributes, the design choices presented in this paper illustrate how non-functional features have been accomplished.

In particular:

- **Modularity and Scalability**

The division of the system in components that are responsible for semantically different functions guarantees modularity and scalability, allowing the developing of new components related to new functions, or enriching the existing ones. Furthermore, the service is presented differently for each kind of user, with regards to calls to methods and interface.

- **Availability and Reliability**

Following a scale-out approach, the Application Server will be cloned. Cloning needs the introduction of a new component in the system: the load balancer. It will be able to manage the traffic in order to guarantee the requested service in a better and faster way. More than one physical node that runs in parallel will avoid system downtime due to a failure or maintenance. When a clone is down, there are others in parallel that can supply the service just with some slowdowns.

- **Maintainability**

The simply and easy structure of the various components allows the system to support changes without affecting all the components. The partitioning between the Application Server's components devoted to catching clients' requests and the ones devoted to retrieving data from DBMS simplifies every single component's work and sustains DBMS changes in structure or type.

- **Security**

The adoption of OAuth 2.0 guarantees the compliance with the different roles of the platform (Company, Student) and strengthens the privacy of sensitive data.

Moreover, the integration of an encryption protocol may help masking passwords and keywords, securing the access to private areas of the platform.

- **Portability and hardware limitations**

The system is designed in accordance with the "thin client" paradigm, which ensures the lightness of client-side computational load. Moreover, it is realized as a web platform, which by nature implies its portability on several kinds of machines. Therefore, the system is easily accessible to almost whoever has a PC.

## 5 Implementation, Integration and test Plan

The application is divided as follows:

- Client (Web Browser)
- Web Server
- Application Server
- Database

A bottom-up strategy will be used to implement, integrate and test the platform; this means that the platform development will start from individual components and will build up to the larger systems. This approach allow to decompose the big problem (aka the whole platform implementation) into smaller and manageable tasks. This means that for our problem, the construction starts from the DB, going on to the application server blocks and composing our system.

### 5.1 Component priority

To improve readability, here is defined the priority in which parts of the system should be implemented.

Component	Priority
Student Manager	Very High
Company Manager	Very High
Recommendation Manager	Very High
Authorization Manager	High
Internship Manager	High
Data Manager	Medium
Notification Manager	Low

Table 2: Component priority

## 5.2 Integration

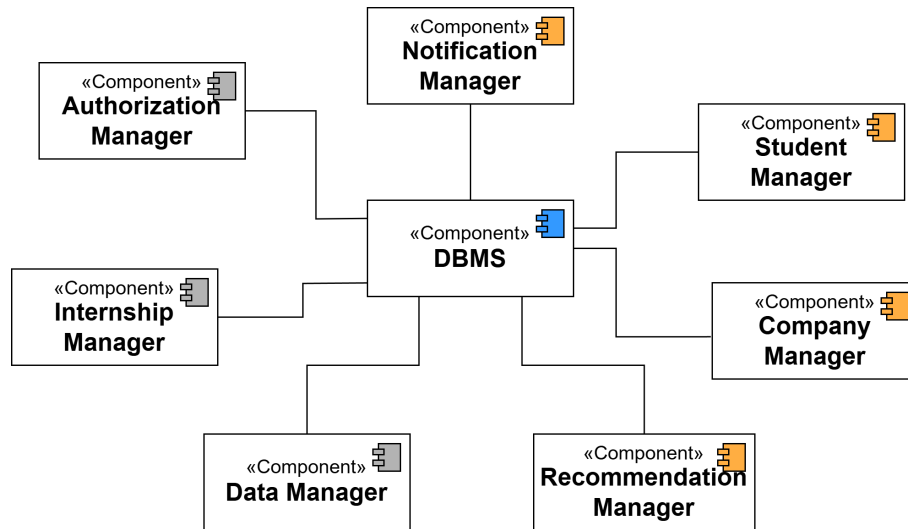


Figure 5: Integration

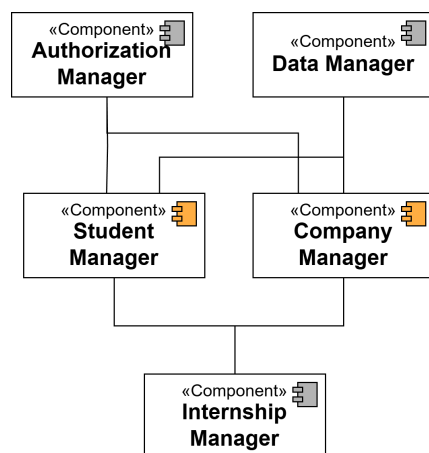
This figure above is introduced to highlight, as mentioned in 2.6.1, that the database is the core of our application and should integrate with all the other modules of the application.

## 5.3 Build and test plan

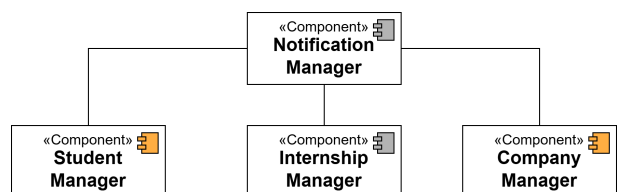
Here are shown the list of integration needed between the various modules of the platform.

As its central in the development of the application, since it handle most of the request and logic of the platform, the DBMS should be the first thing to develop and should be tested before the integration with other module by the developers to ensure its correctly working.

After DBMS implementation, developers must implement student, company and recommendation modules, test them, and only after this phase integrate them. The integration comes via the Data, Authorization and Internship managers implementation. The integration is clear in the Component view image (2).

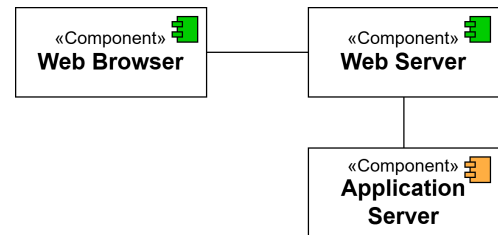


The last component to be implemented is the notification manager, even if it interacts with almost all modules. It has a lower priority of implementation because its functionalities are not critical to other components' functions.



3.

Finally, to achieve client-server communication, the application should be integrated with the web server and consequently, with the web browser.



## 6 Effort Spent

Sections	Cimino	Lorenzetti	Together	Total
Introduction	0	0	3	3
Architectural Design	9	3	9	21
User Interface Design	1	5	2	8
Requirements traceability	1	1.5	1	3
Implementation, Integration and test Plan	1	1	2	4
Total	12	10.5	17	39.5

Table 3: Effort Recap