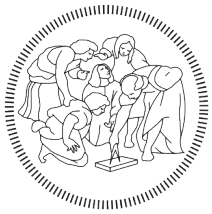
Design Document

Students&Companies project Andrea Bellani Alessandro Capellino

Andrea Bellani, Alessandro Capellino

January 7, 2025



POLITECNICO
MILANO 1863

Contents

List of Figures

List of Tables

Introduction 1

1.1 Purpose

During their university studies, in order to start entering the workforce, a student might decide to apply for an internship related to their field of study. Similarly, companies offering internships may be interested in finding students that are adequate for them. To facilitate the matching between students and companies, a new platform called *Students and Companies* (S&C) is to be developed. S&C allows companies to look for suitable students by publish internship advice on the platform, while students can look for internships that interest them. Moreover, the platform implements recommendation mechanism to help student and companies to find each other. Once the contact is established, S&C can provide support to the students selection process.

1.2 Scope

As mentioned in the RASD document, there are two main users categories that interact with the system: *Companies* and *Students*. The companies publish announcements about the internships they want to offer where they specify *projects* that will be carried out and the *terms* of the offer. The system itself informs the companies about the availability of students who may be suitable for their internships (based on their profile).

Students, on the other hand, may use the platform to look for internships and S&D can also notify them if there are new internships that could meet their interests, but they can still independently search through all the available internships.

Once a *contact* is established and accepted by the two parties, the student selection process begins. At this point the company defines selection steps and schedules the interviews for each student. Once the selection is over, the system collects feedback and suggestions from both students and companies.

Finally, both students and companies can monitor the progress of the internships by providing information on its development and any issues that may arise.

1.3 Definitions, Acronyms and Abbreviations

1.3.1 Definitions

- **internship advice** : a call for application related to an internship that will be offered by a company;
- **recommendation** : the mechanism related to the fact that the system both informs students whether new internship advice that might interest them are published and notifies companies of the presence of students that might be suitable for their internships;
- **project** (of an internship advice) : the definition of the application domain, the set of tasks to be performed and the set of the most relevant adopted technologies (if any) for an internship;
- **terms** (of an internship advice) : the set of benefits offered by an internship (e.g. paid/not paid, training, lunch voucher...);
- **selection process** : each internship advice is followed by a sequence of selection steps.

1.3.2 Acronyms

- S&C: Students&Companies, the name of the platform;
- UML: Unified Modeling Language;
- CV: Curriculum Vitae.
- DB: Database
- RDBMS: Relational Database Management System
- API: Application Programming Interface

1.3.3 Abbreviations

- Gn: Goal number n;
- Rn: Requirement number n;
- Dn: Domain assumption number n;
- WPn: World Phenomena number n;
- SPn: Shared Phenomena number n;
- UC: Use Case.
- UI: User Interface

1.4 Revision history

First version
07/01/2024

1.5 Reference documents

The Documents used to deliver the RASD document are the following:

- the Specification of RASD and DD assignment of Software Engineering 2;
- the class slides on WeBeep, in particular slides on UML diagrams, software architectures styles and the part related to Integration and Testing

1.6 Document structure

1. **Introduction**: this section provides a brief introduction to the purpose of the platform to be developed, S&C in this case, focusing in particular on the most important goals which the system has to achieve and on the various phenomena identified, as we mentioned in the RASD document;
2. **Architectural Design** : this section provide a description on the general architecture chosen for the system, at different levels of granularity: it focuses on the component of the architecture, on how they are related (showing also how the most important functionalities of the platform are provided) and on the deployment view;
3. **User Interface Design** : in this section it is explained how the User Interface is designed to provide the functionalities of the platform
4. **Requirement Traceability** : in this section is explained how the goals described in the RASD document are satisfied showing the interaction between the requirements and the components of the architecture;
5. **Implementation, Integration and Test Plan**: this section provides a description of how the components of the system are integrated and then tested; this information are useful for the developers of the platform
6. **Effort Spent**:report of the time spent by any group member in any document section;
7. **Software Used**: list of software and documents used to develop the document.

Architectural Design 2

2.1 Overview

From an implementation perspective, it was decided to develop the S&C platform as a web application.For this reason, the architecture chosen for its development is a 3-tier architecture. It is a software design pattern that separates applications into three distinct layers:

1. **Presentation Tier**: the user interface (UI) where users interact with the application;
2. **Application Tier**: the tier in which the data are processed;
3. **Data Tier**: manages the storage, retrieval, and manipulation of data, involving databases.

This architecture offers significant advantages in terms of scalability, maintainability, and especially modularity, allowing each layer to be developed and updated independently of the others.

Going into detail, the data layer employs two different types of databases: a relational database is used to manage the structured data of our system, which constitutes the majority of the data utilized by the platform (users, internship advice, internships, notifications, etc.). For the management of unstructured data, namely CVs and selection-related questionnaires, a non-relational database was chosen. Specifically, MongoDB was selected, as its JSON-based data storage format is perfectly suited for hierarchical and non-rigid structures, such as CVs and questionnaires.

Here we present an high-level view of the architecture:

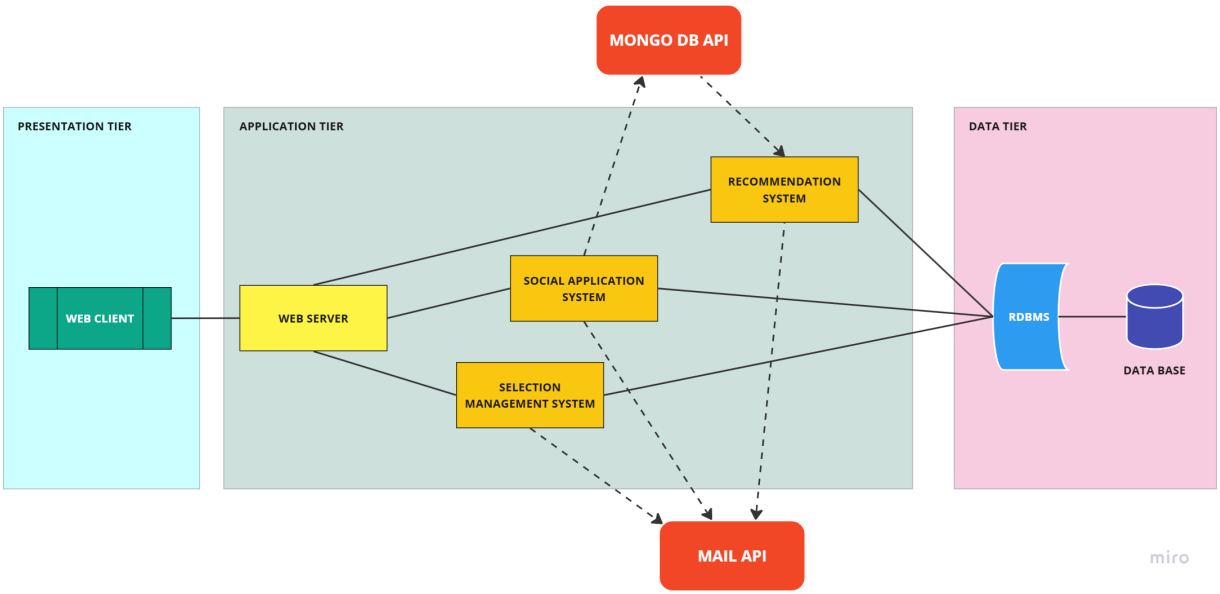


Figure 2.1: High-level architecture

Note that the mail client is not developed by used, it is only shown here for completeness.

2.2 Component View

As mentioned, the platform is developed used a 3-tier architecture approach with 2 DB(a RDBMS and MongoDB); As for the back-end components, four main macro-components can be identified, which are:

1. **Social Application System**: It provides the ”social” functionalities of the platform, such as user registration/login, searching for internship advice, viewing profiles, applying for/sending an application proposal, sending feedback/complaints, and so on. It is divided in several components:
    a) **Authentication Manager**: It handles user authentication (initial registration, subsequent logins);
    b) **Application Manager**: It is responsible for managing application proposals, acceptances, and application requests;
    c) **Advice Presenter**: It provides functionalities for viewing individual internship advice and the list of all internship advice published on the platform;
    d) **Advice Publisher**: It provides functionalities for publishing and deleting an internship advice;
    e) **Feedback Manager**: It provides functionalities for viewing feedback questionnaires and receiving the corresponding responses;
    f) **Internship Manager**: It displays the ongoing internships (both the list and individual ones) and manages user complaints;
    g) **Profile Manager**: it provides functionalities for managing a profile, from creation to modification and deletion
    h) **Profile Presenter**: It allows viewing profiles
2. **Recommendation System**: It is the component responsible for the recommendation functionality offered by the platform, which involves analyzing the necessary information (feedback, CVs for students, etc.), determining which internships/students may be of interest to a specific user, and finally sending this information to the interested user. It is divided into 3 sub-components:
    a) **Recommendation Interface**: It is the component that retrieves the necessary information from the databases to develop the recommendation;
    b) **Recommendation Analyzer**: t is the component that performs the analysis on the information obtained from the recommendation interface (feedback, CVs, etc.);
    c) **Recommendation Presenter**: It inserts the recommendation results into the database and triggers the sending of notifications to inform that the recommendation has been made
3. **Selection Management System**: It provides functionalities related to the entire selection process, from its configuration to the sending of results. It is divided in 3 sub-components:
    a) **SP Initializer**: It provides functionalities for configuring the selection process (dates, metrics, questionnaires, etc.);
    b) **SP Manager**: It manages the insertion of answers in the questionnaires and the subsequent finalization of the selection process
    c) **SP presenter**: It displays interview dates and the results of the selection process
4. **Notification System**: It is the component responsible for managing all notifications generated and sent by the system; it handles the creation, sending, and triggering of notifications with the mail service. It is divided in 2 sub-components:
    a) **Notification presenter**: It allows viewing notifications (both individual ones and the list);
    b) **Notification Generator**: It is the component that generates a notification and triggers the sending of an email;

there are other important components of the system:

1. **DBMS**: stores all the structured data of the platform
2. **WebServer**: it is the component related to routing the incoming request to the appropriate internal component.
3. **MongoDB API**: A programming interface that allows developers to interact with a MongoDB database through HTTP requests, using CRUD operations (Create, Read, Update, Delete) on data stored in a MongoDB database;
4. **SES API**: Amazon Simple Email Service (SES) is an email sending and receiving service offered by Amazon Web Services (AWS). The SES API allows to easily integrate email sending functionality into applications/websites.

for the front-end, there are 2 clients which interact with the back-end:

1. **WebClient**: the UI of the application; it interacts with the back-end through REST API, which is based on a set of principles and constraints that allow systems to communicate over the web using standard HTTP methods such as GET, POST, PUT, DELETE, etc.

2. **Client Mail**: the client used to access and manage the mail; it communicates with the back-end (in particular with the Notification system) through SES API

the overall component diagram is shown in the following figure:

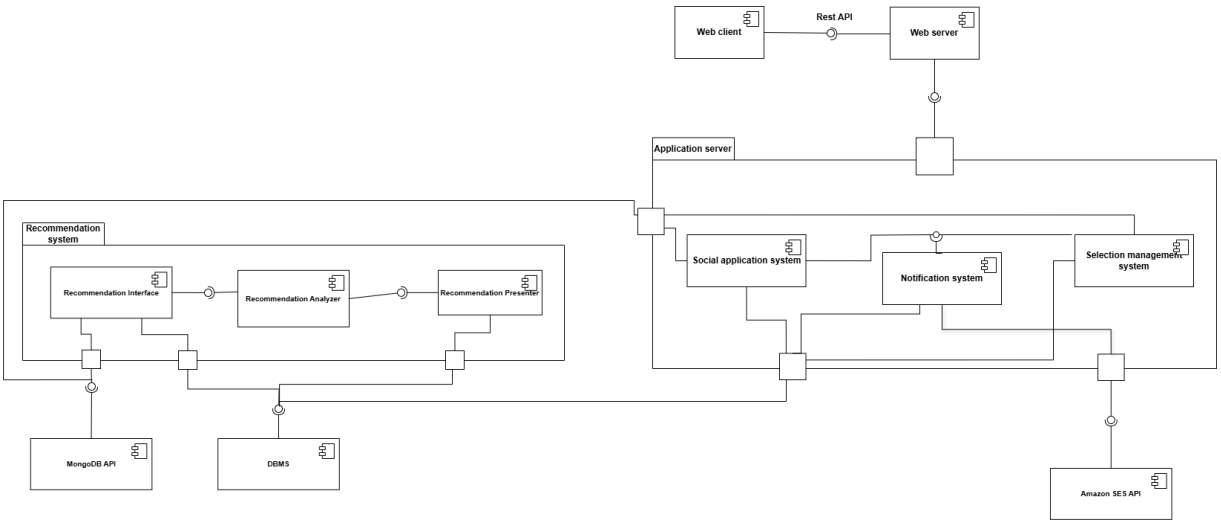


Figure 2.2: System UML component diagram

and here the detailed structure of each macro-component of the *Application server* (*Recommendation system* components are not further subdivided):

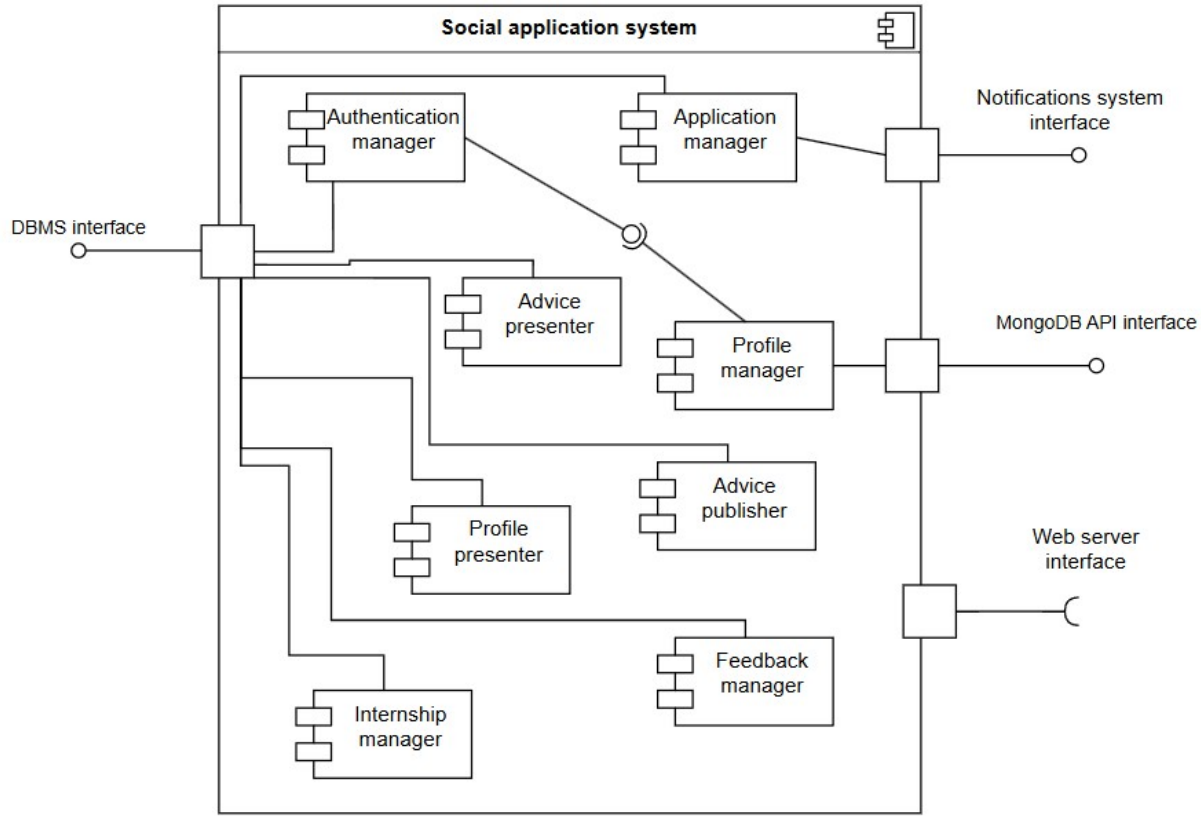


Figure 2.3: Social application system UML component diagram

Note that in the social application system component diagram each sub-component is implicitly connected to the web server with the designated interface.

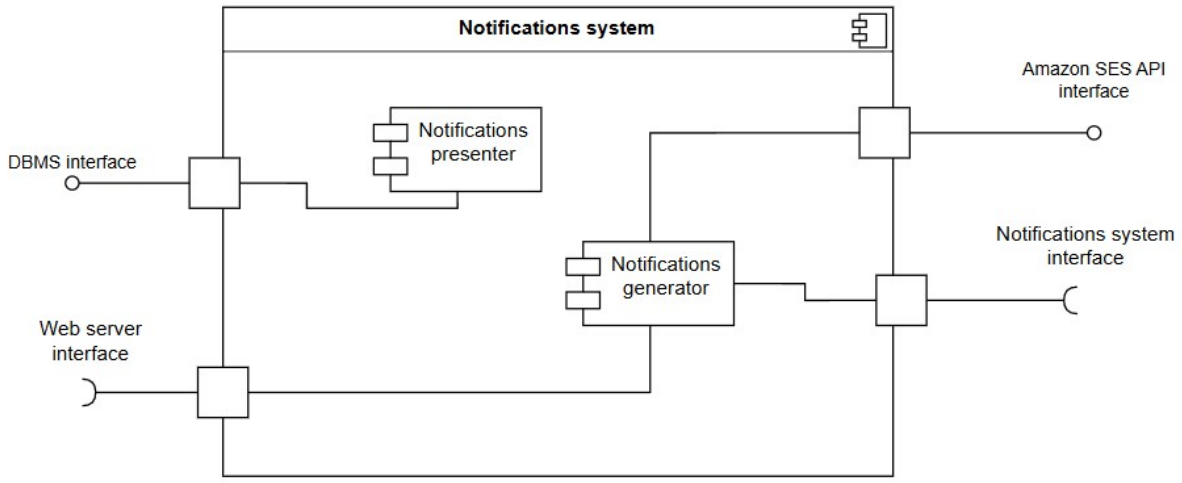


Figure 2.4: Notifications system UML component diagram

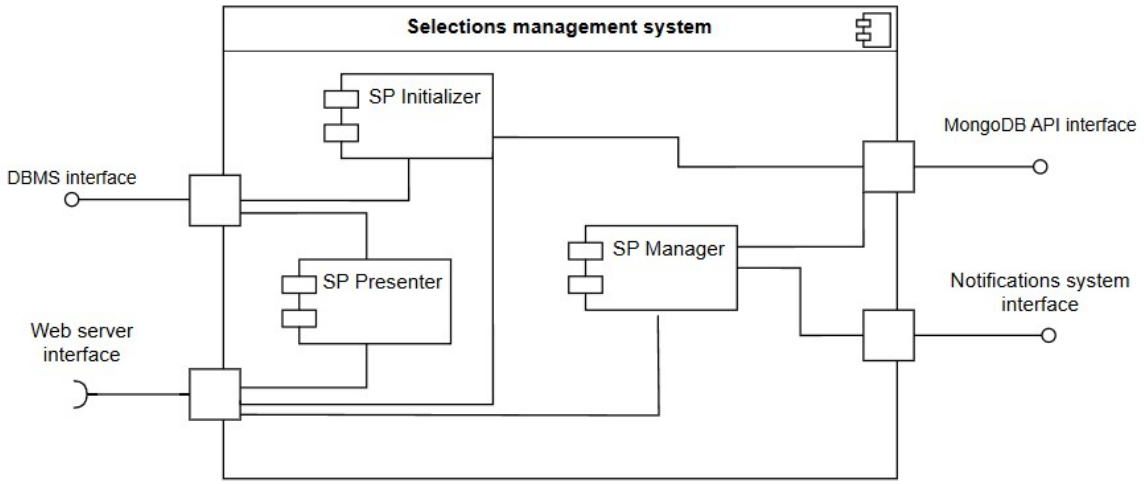


Figure 2.5: Selections management system UML component diagram

2.3 Deployment View

For what concern the system deployment, our solution had to meet the non-functional requirements stated in the RASD. In order to ensure it, we decided to deploy the business and data tier on the Amazon AWS cloud (business tier is managed by AWS EC2 while data tier is managed by Amazon SE3).

For each non-functional concern mentioned in the RASD that interests the deployment choices, we considered a solution that would have met it, in particular:

- availability concerns:
  - presence of a load balancer: provided by AWS EC2;
  - large-scale availability: provided by AWS EC2;
- security concerns:
  - DDoS prevention: AWS Shield;
  - communication security: HTTPS communication is permitted through AWS EC2;
  - firewalling: provided by AWS EC2.

Moreover, in the design phase we chose to deploy the recommendation system or an independent node, in this way the heaviness of the recommendation analysis can't impact the application server performances.

Note that as we explained in other sections, the recommendation system is not queried from the application server. Recommendation results are inserted in the relational data base from the recommendation system independently from the application server behavior. Therefore, these two subsystems have not to be synchronized: they don't update the results and don't wait results of each other.

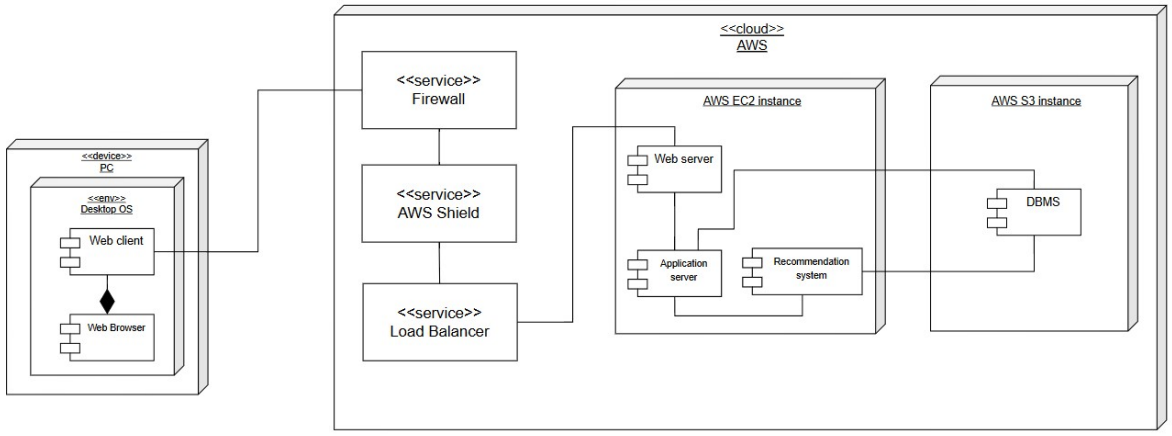


Figure 2.6: UML Deployment diagram

2.4 Runtime View

[UC1] - Student Registration

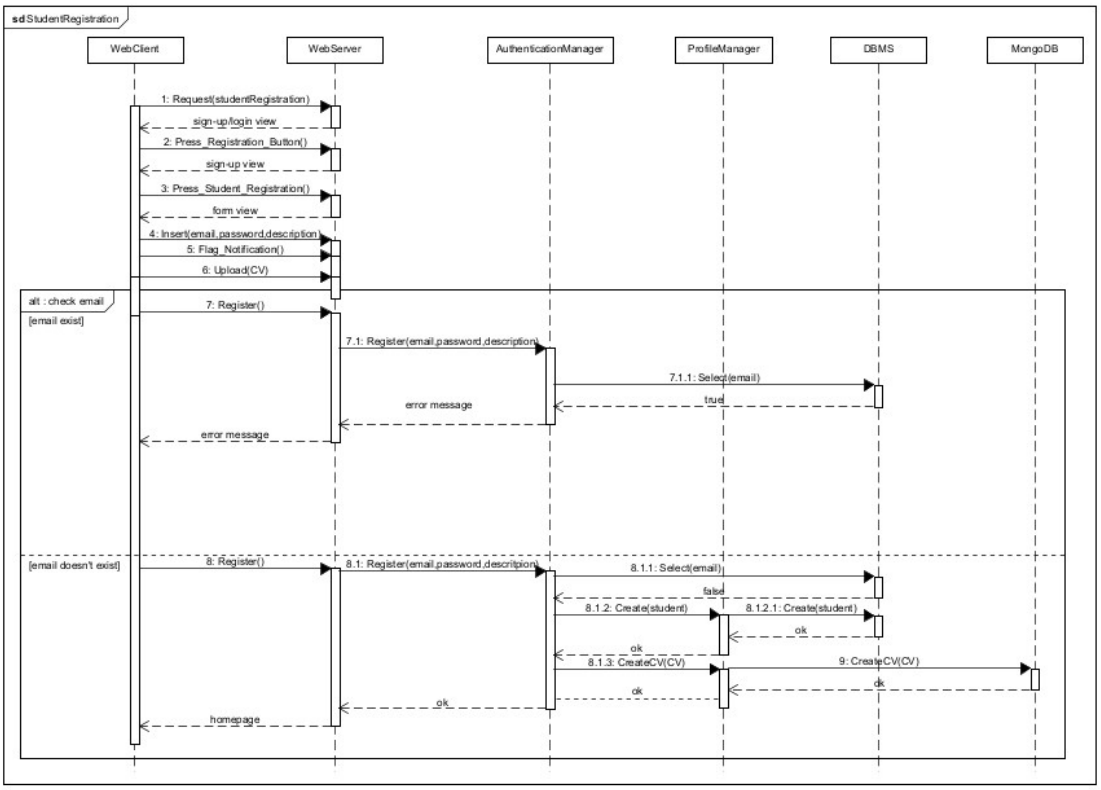


Figure 2.7: [UC1] - Student Registration

The figure shows the student registration process; the request is sent to Authentication Manager, which is also responsible for receiving the registration form completed by the student. Once received, Authentication Manager queries the DBMS to check if the student's email is already present in the database. If it is not present, the student will be saved in the database, while the CV will be stored in MongoDB as an unstructured data. If the email is already in the database, an error message will be displayed to the student.

[UC2] - Company Registration

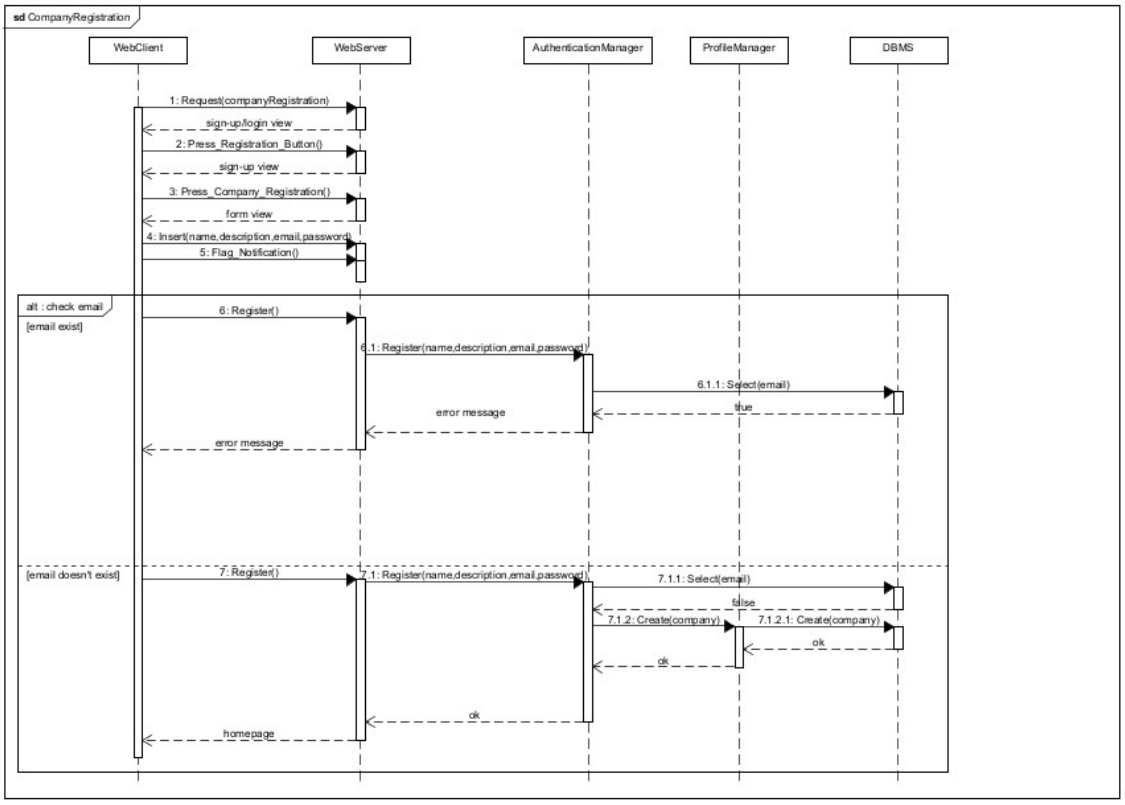


Figure 2.8: [UC2] - Company Registration

The figure shows the company registration process; the flow is similar to the student registration case, with the difference that in this case, different data is saved (the company has a name and a description), and all the data is stored in the relational database since they are all structured data.

[UC3] - User Login

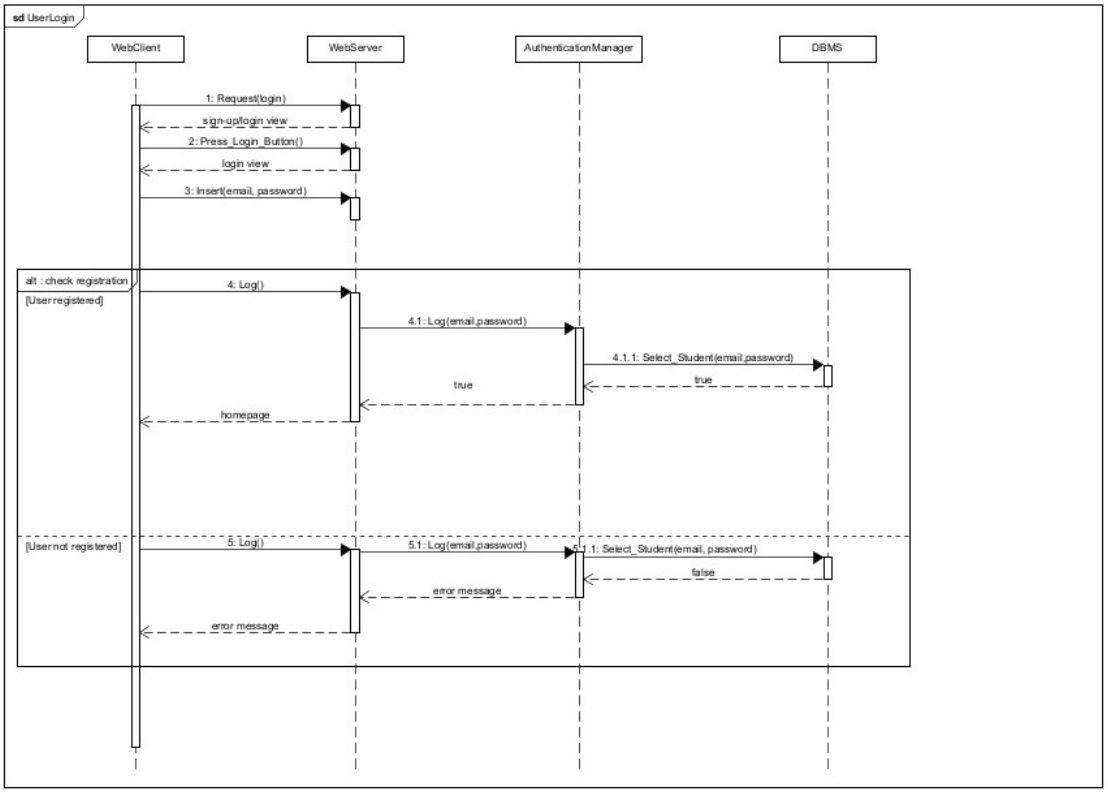


Figure 2.9: [UC3] - User Login

The figure shows the user login process, which is similar for both students and companies; in this case, the request is handled by the Authentication Manager, which selects the student from the database to allow login. If the student is not found, an error message is displayed.

[UC4] - Publish Internship Advice

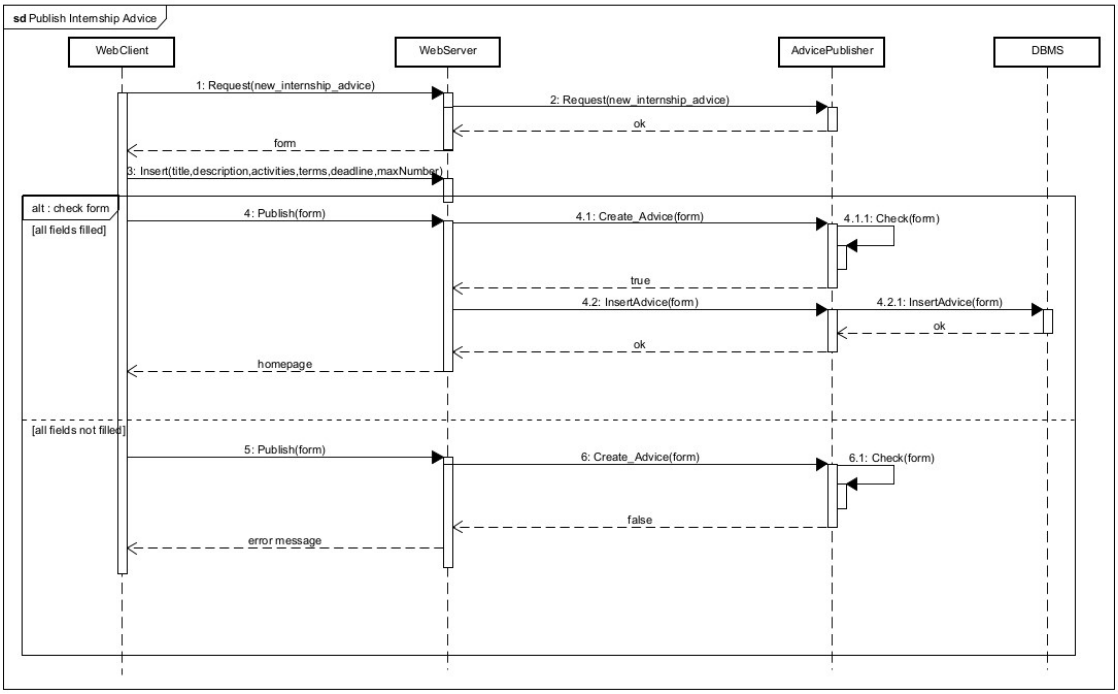


Figure 2.10: [UC4] - Publish Internship Advice

the figure shows the publish internship advice process; the web server redirect the request to the publisher advice component, which create a new advice and put it in the relational DB; before that, the publisher advice check if the form is already filled: in the negative case, it presents to the client an error message.

[UC5] - Search of all Internships

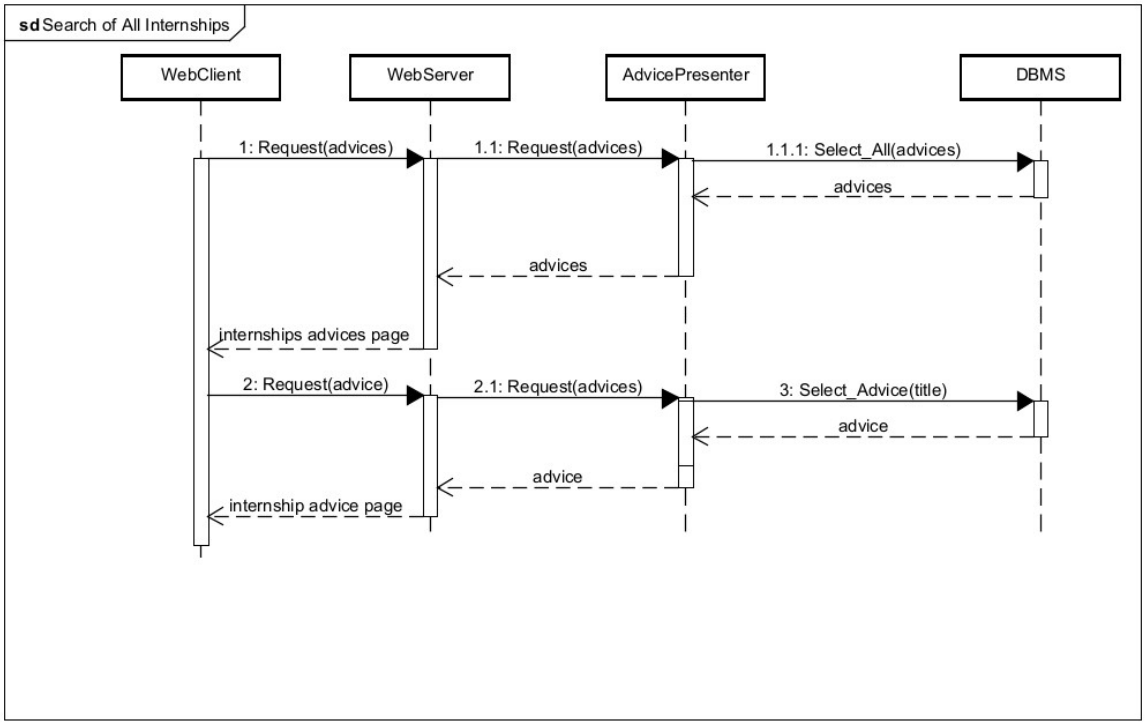


Figure 2.11: [UC5] - Search of all Internships

the figure shows the process of search on all the internships: the web server sends the request to the advice presenter, which selects all the advice from the relational DB; then the client requests a specific advice and the flow is the same: the advice presenter is called and it selects the advice (by title) from the relational DB

[UC6] - Search of All Companies

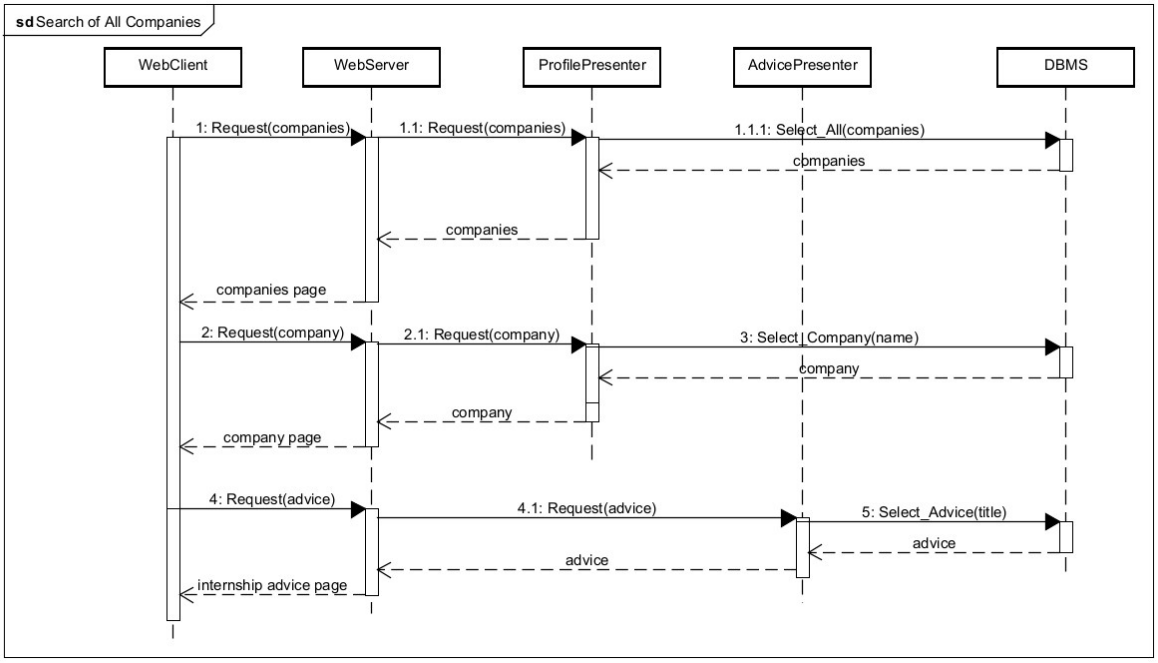


Figure 2.12: [UC6] - Search of All Companies

the figure shows the process of search on all the companies: the web server sends the request to the profile presenter, which selects all the companies from the relational DB; then the client requests a specific advice: the advice presenter is called and it selects the advice (by title) from the relational DB and it is presented to the client

[UC7] - Search an Internship by Name

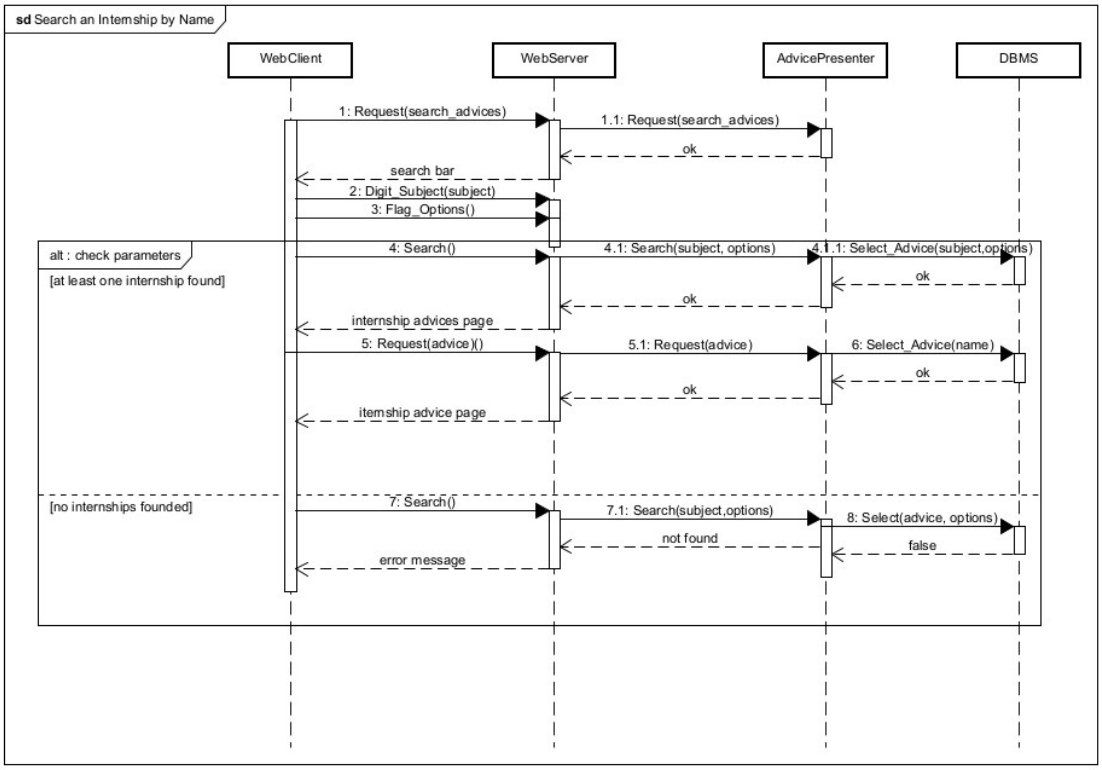


Figure 2.13: [UC7] - Search an Internship by Name

the figure shows the process of proactive search of an internship; the request is redirect to the advice presenter, which receive then from the client the subject searched and the options flag; then this component query the relational DB to find if exists at least one advice that match the subject and the option: if it is true, return the list of this advice and then the client requests a specific one (as we mentioned before); if it is false, return an error message.

[UC8] - Internship Recommendation

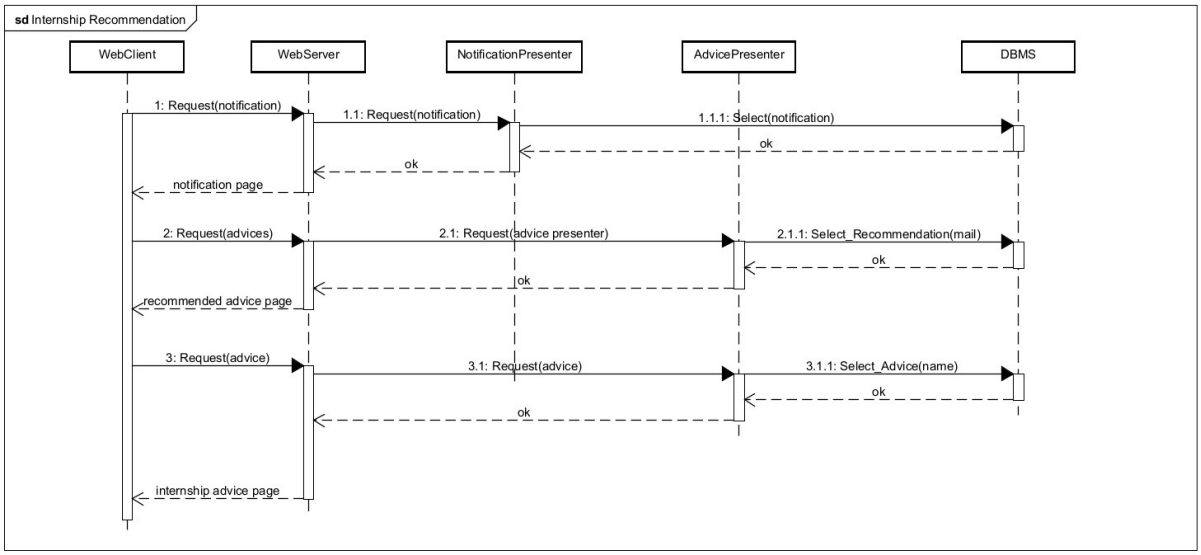


Figure 2.14: [UC8] - Internship Recommendation

the figure shows how the students can look for internships advice recommended; first of all, the client displays a notification: the web server sends the notification to the notification presenter, which retrieves it from the database. Subsequently, to view the recommended advice, the web server forwards the request to the advice presenter, which retrieves the recommended advice related to the student from the database and then presents it to the client. Finally, the client selects an advice in the manner described earlier.

[UC9] - Students Recommendation

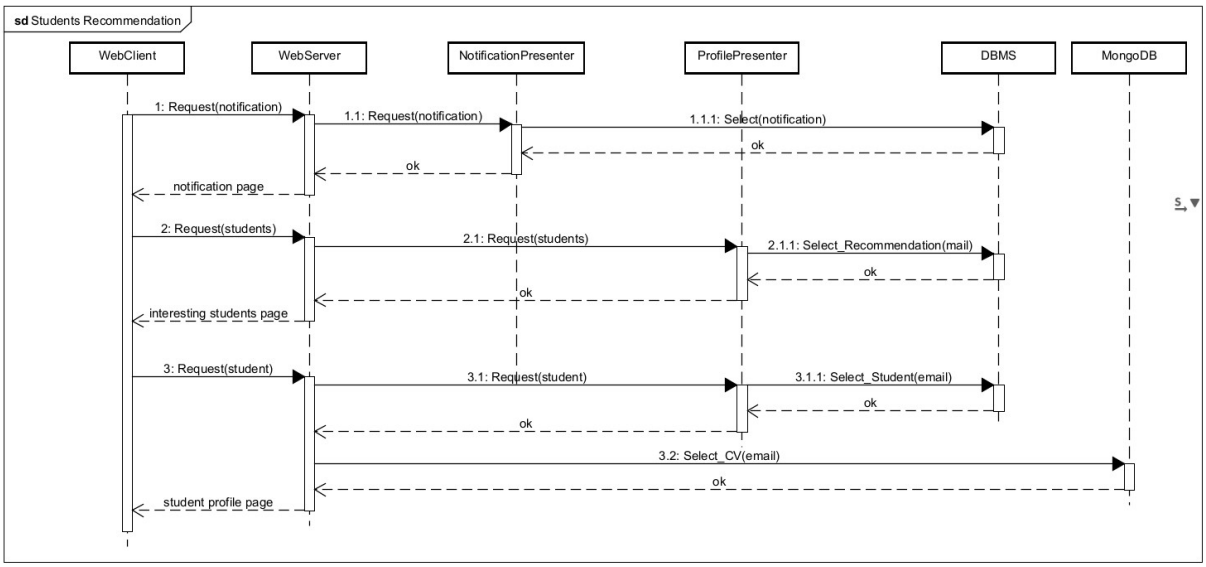


Figure 2.15: [UC9] - Students Recommendation

the figure shows how the companies can look for students recommended; first of all, the client displays a notification: the web server sends the notification to the notification presenter, which retrieves it from the database. Subsequently, to view the recommended students, the web server forwards the request to the profile presenter, which retrieves the recommended students related to the company from the database and then presents it to the client. Finally, the client selects a students(note that it has to retrieve also the CV from MongoDB, instead the other data are retrieved from the relational DB)

[UC10-a] - Student Application Process-send application

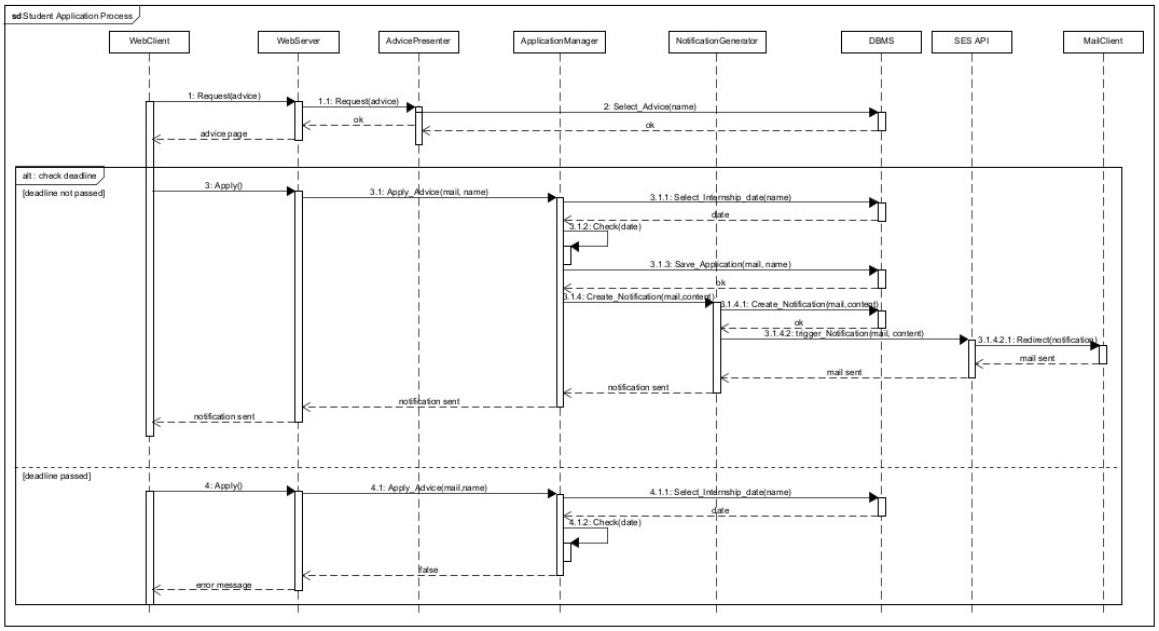


Figure 2.16: [UC10-a] - Student Application Process-send application

[UC10-b] - Student Application Process-accept application

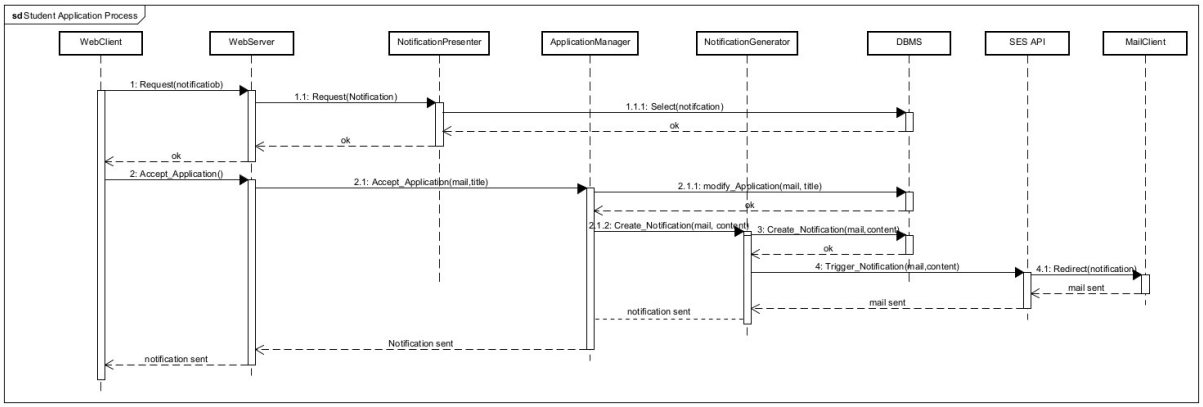


Figure 2.17: [UC10-b] - Student Application Process-accept application

In these two sequence diagrams, the entire student application process is depicted. Both the application submission phase and the acceptance of an application utilize the Application Manager component, which is responsible for inserting the newly created application into the relational database. The Notification Generator component then triggers the email sending process and sends a notification to the recipient company, which accepts the application by sending a response notification (Figure 2). It should be noted that the company can also reject the application, but the modeling is identical to the acceptance process.

[UC11-a] - Company Send of Proposal-send proposal

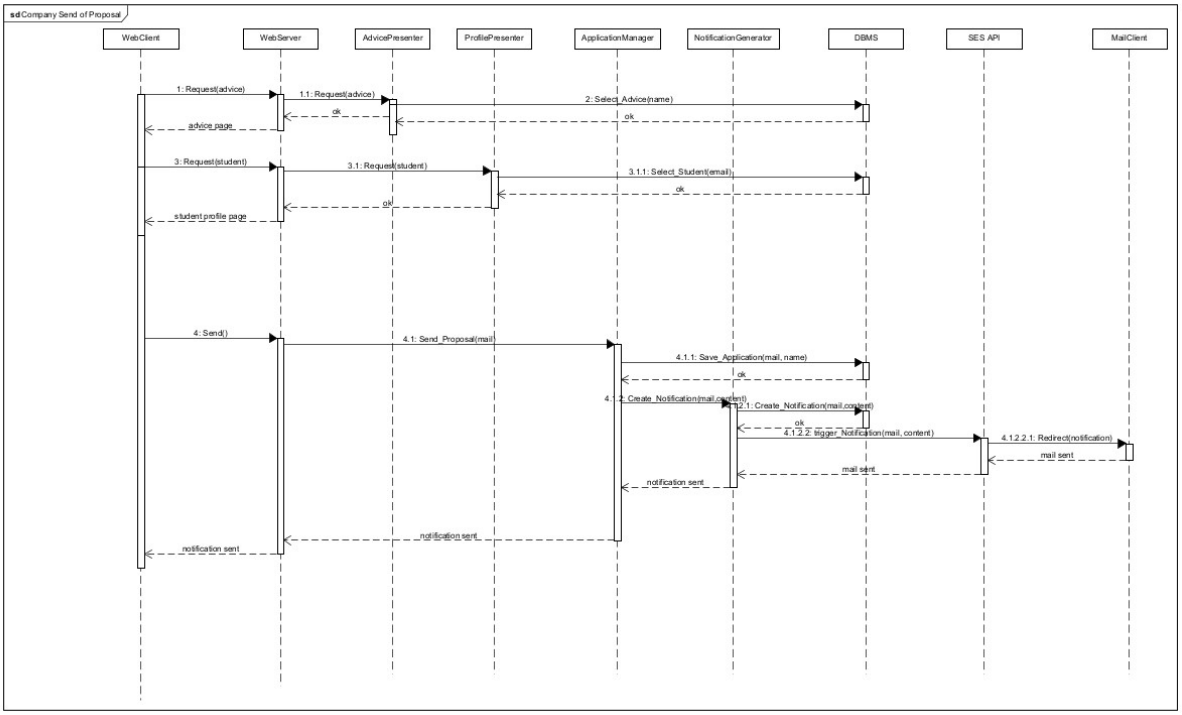


Figure 2.18: [UC11-a] - Company Send of Proposal - send proposal

[UC11-b] - Company Send of Proposal-accept proposal

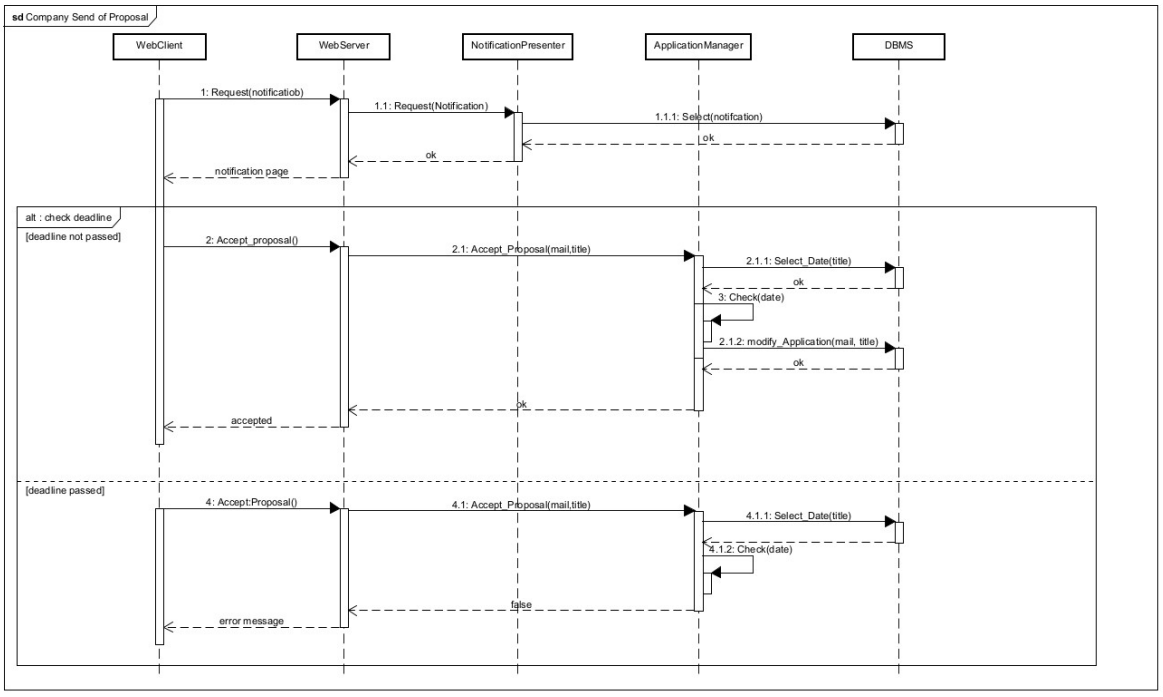


Figure 2.19: [UC11-b] - Company Send of Proposal - accept proposal

The two sequence diagrams show the entire process of submitting a proposal by a company. In both figures, it is the Application Manager component that handles the creation of a new application, sending the notification

(with email trigger), and managing the acceptance. The acceptance can only occur if the deadline has not yet passed; otherwise, an error message is displayed.

[UC12] - Selection Process Configuration

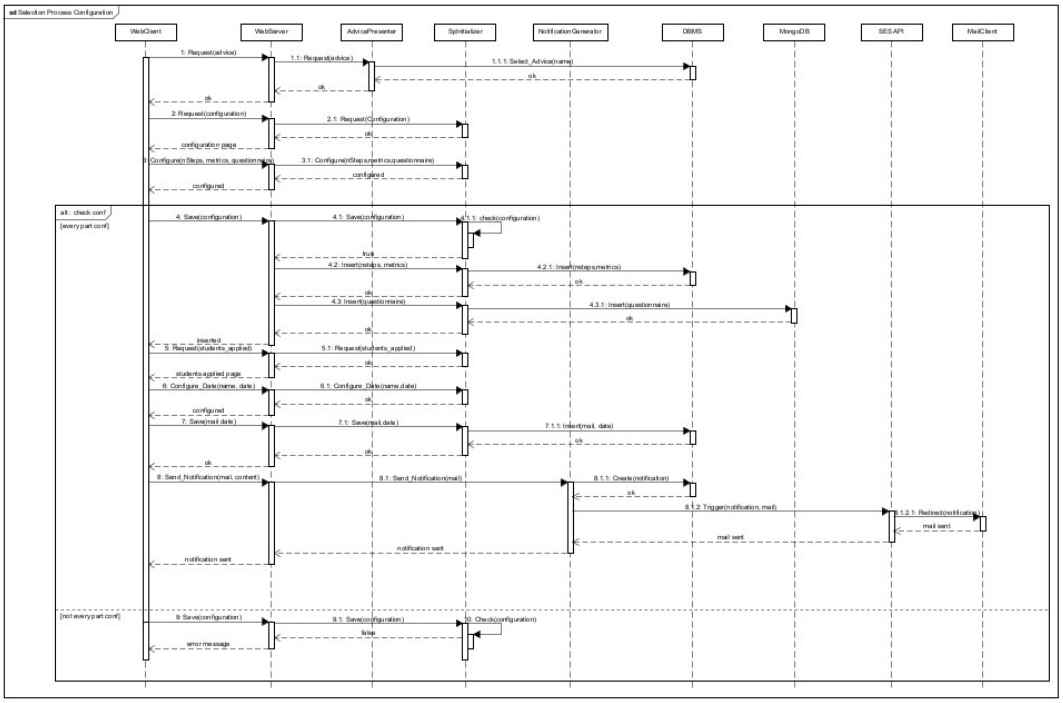


Figure 2.20: [UC12] - Selection Process Configuration

the figure shows the selection process configuration; initially, the client requests the advice it wants to view (and the advice presenter retrieves it from the database). Subsequently, to initialize the selection process, a request is sent to the SP initializer, which displays the configuration to be completed. The client completes the configuration by entering the number of steps, metrics, and defining the questionnaires for the questions. The configuration is then saved by the SP initializer in the databases: metrics and the number of steps are stored in the relational database, while the questionnaires are saved in MongoDB's non-relational database (as they are unstructured data).

Afterward, similarly, the interview dates for each student are defined and saved in the relational database. Finally, the client sends a notification: the Notification Generator handles generating the notification, then triggers the SES API, which directs the email to the student's Mail client. If the configuration is incomplete, an error message will be displayed to the client.

[UC13] - Selection Process Running

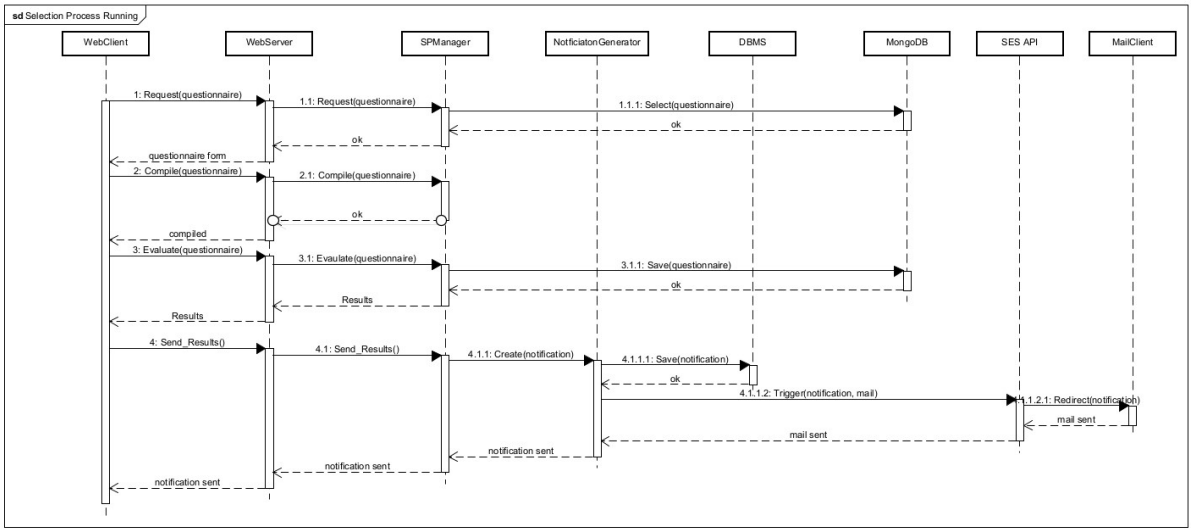


Figure 2.21: [UC13] - Selection Process Running

the figure shows the running of the selection process; the main component is SP Manager which retrieve from MongoDB the questionnaires and finally triggers the Notification Generator which create a new notification representing the sending of the results to an user; the questionnaires are saved in MongoDB after the entering of the answers

[UC14] - Collection of Student Selection Process Feedback

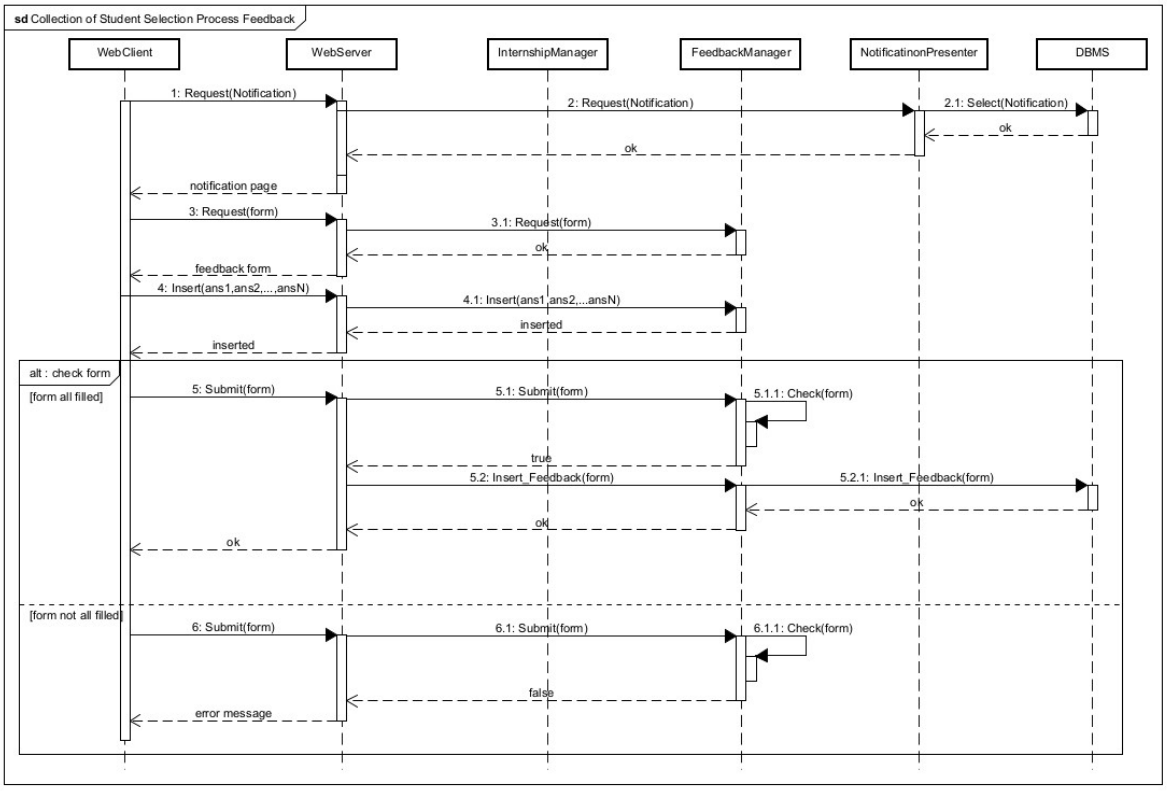


Figure 2.22: [UC14] - Collection of Student Selection Process Feedback

the figure shows the collection of students selection process feedback; the client requests the notification containing the form to be filled out to provide feedback (the form is displayed by the Feedback Manager). The client fills out the form and submits it to the Feedback Manager, which handles querying the database to insert the feedback. If the form is incomplete, the Feedback Manager will display an error message to the client.

[UC15] - Collection of Company Selection Process Feedback

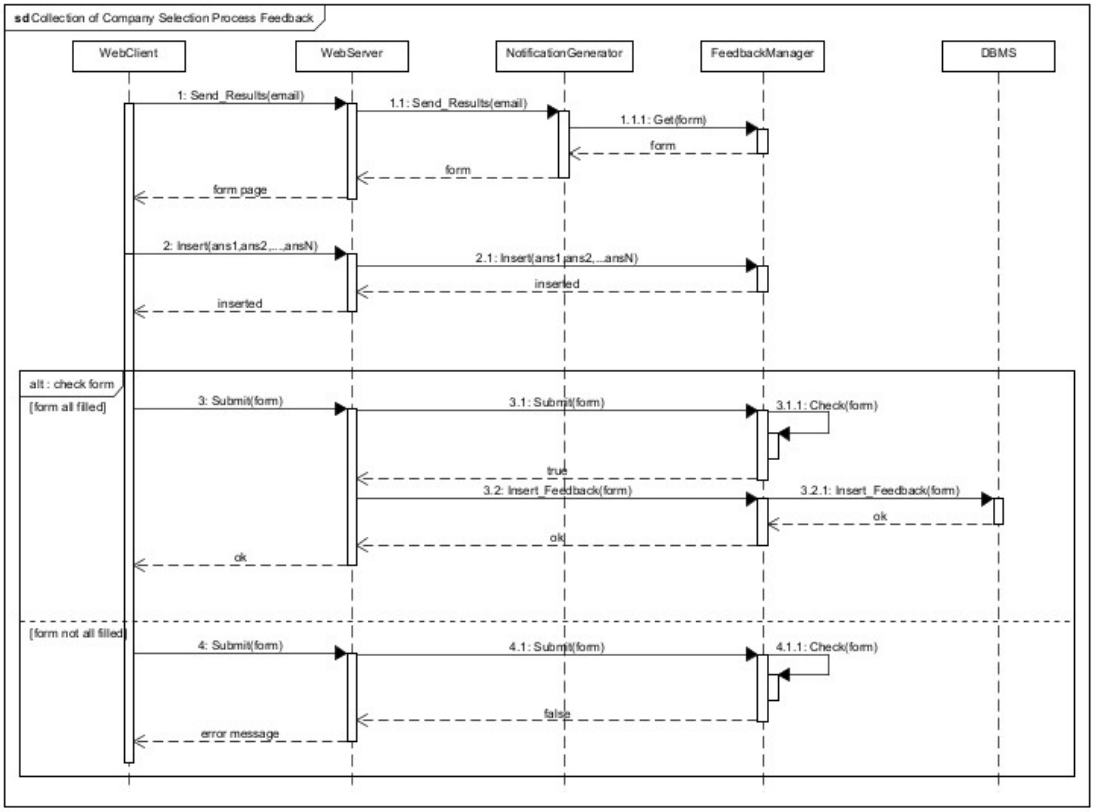


Figure 2.23: [UC15] - Collection of Company Selection Process Feedback

the figure shows the collection of company selection process feedback; after sending the result notification, the client obtain the form from the Feedback Manager. The client fills out the form and submits it to the Feedback Manager, which handles querying the database to insert the feedback. If the form is incomplete, the Feedback Manager will display an error message to the client.

[UC16] - Collection of Internships Feedback

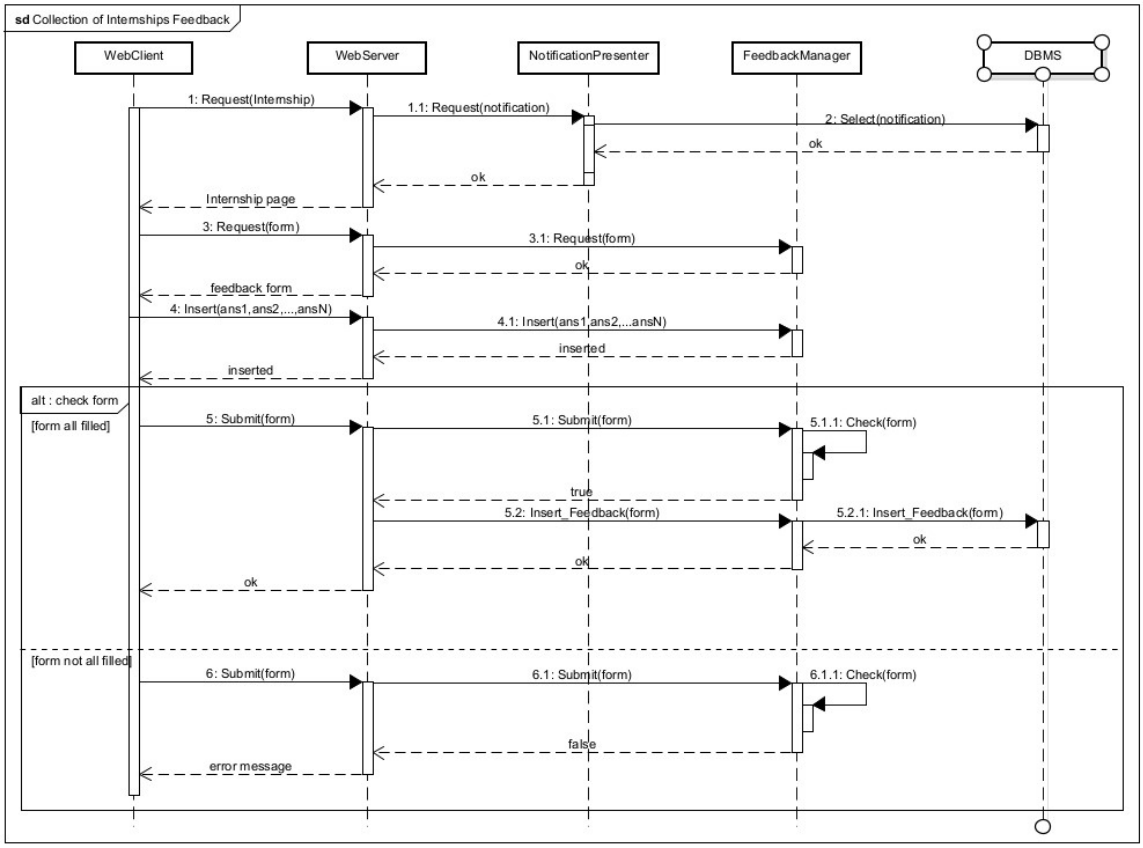


Figure 2.24: [UC16] - Collection of Internships Feedback

the figure shows the collection of Internships feedback; the client request the notification regarding the closing of the internship(offered by Notification Presenter). Subsequently, the client requests the form to provide feedback, shown by the Feedback Manager. The client then fills out the form, and the Feedback Manager saves the results in the relational database, provided the form is fully completed. Otherwise, the Feedback Manager shows an error message to the client.

[UC17] - Collection of Complaints

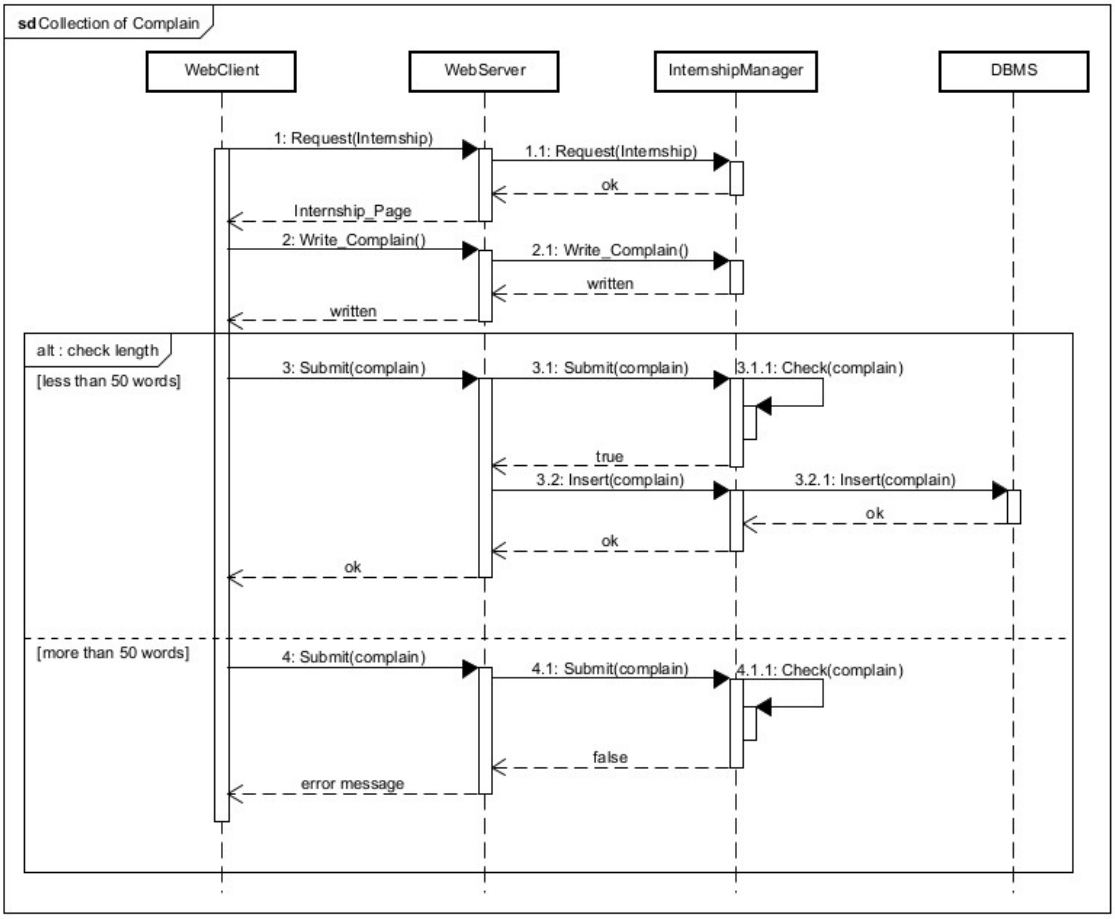


Figure 2.25: [UC17] - Collection of Complaints

the figure shows the collection of complaints written by the user; he flow is similar to the one described for the collection of internship feedback, with the difference that, in this case, the Feedback Manager performs a check on the length of the complaint (which must be less than 50 words).

[UC18] - Ongoing internship closing

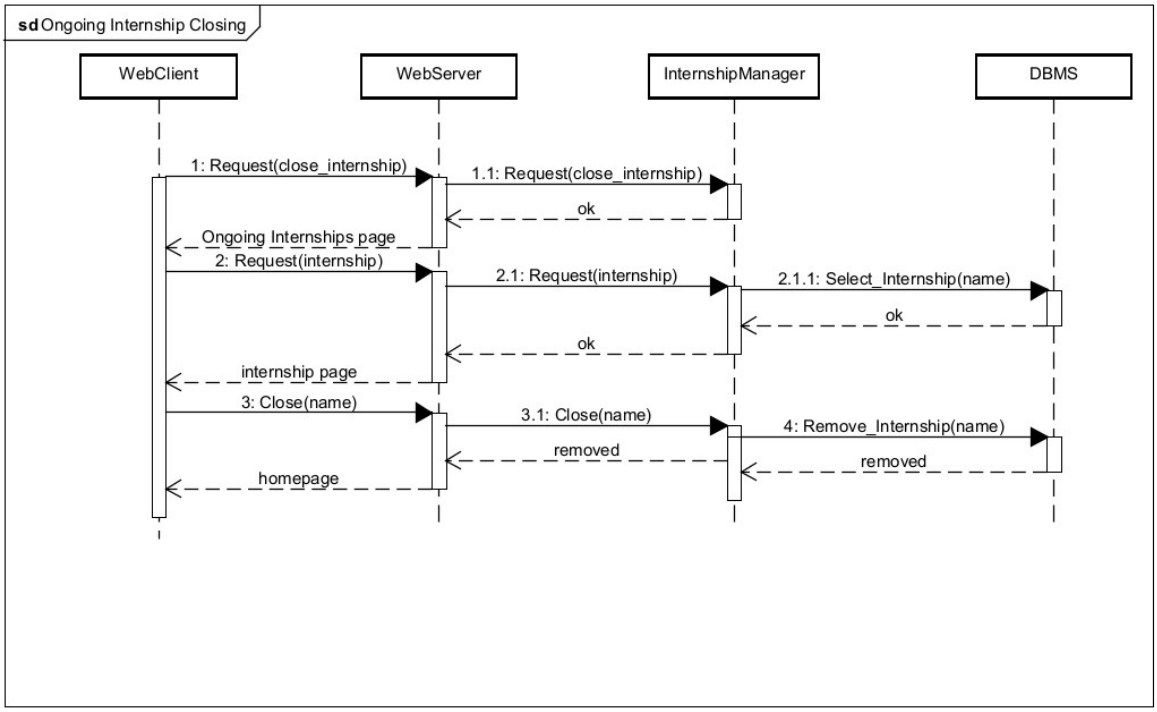


Figure 2.26: [UC18] - Ongoing internship closing

The figure shows the process of closing an internship when it is completed. The Client, when requesting to close an internship, invokes the Internship Manager component, which is responsible for removing the internship from the relational database.

[UC19] - Recommendation Process

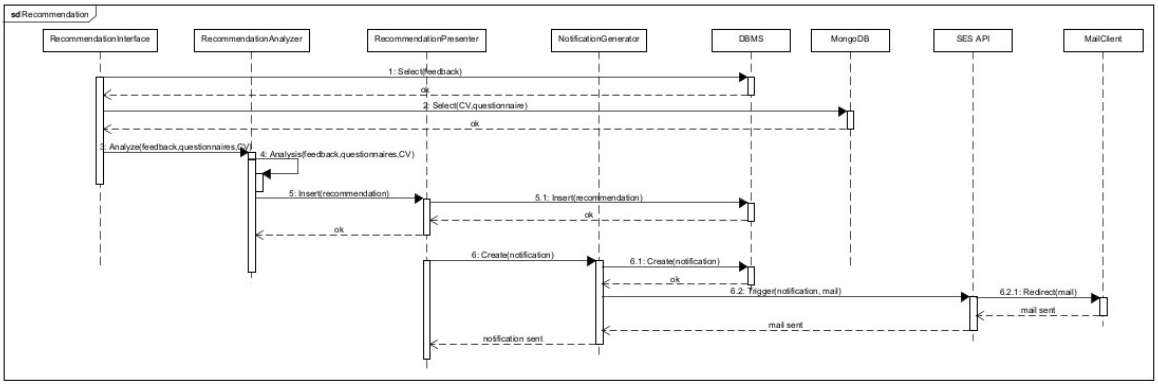


Figure 2.27: [UC19] - Recommendation Process

The figure provides a detailed view of the recommendation process. The Recommendation Interface is the component responsible for retrieving data from the two databases and passing it to the Recommendation Analyzer, which performs the actual analysis. The Recommendation Presenter saves the results in the relational database and triggers the sending of an email to those users who wish to be notified. It is noted that the recommendation process, along with the subsequent delivery of results to users, occurs once a week.

2.5 Component Interfaces

Authentication Manager

- Register(email,password,description)
- Register(name, description,email,password)
- Login(email,password)

Application Manager

- Apply_Advice(mail,name)
- Send_Proposal(mail)
- Accept_Application(mail,title)
- Accept_Proposal(mail,title)

Advice Presenter

- Request(advices)
- Request(search_advices)
- Search(subject, options)
- Request(advice)
- Request(notification)

Advice Publisher

- Request(new_internship_advice)
- Create_Advice(form)
- insert_Advice(form)

Feedback Manager

- Request(form)
- Insert(ans1,ans2,..., ansN)
- Submit(form)
- Insert_Feedback(form)

Internship Manager

- Request(internship)
- Write_Complaint()
- Submit(complain)
- Insert(complain)
- Request(close_internship)
- Close(name)

Profile Manager

- Create(student)
- Create_CV(CV)
- Create(company)

Profile Presenter

- Request(companies)
- Request(company)
- Request(students)
- Request(student)

Recommendation Analyzer

- Analyze(feedback,questionnaires,CV)

Recommendation Presenter

- Insert(recommendation)

Notification Presenter

- Request(notification)

Notification Generator

- Create_Notification(mail, content)

SP Initiallizer

- Request(configuration)
- Confgiure(nSteps, metrics, questionnaire)
- Save(configuration)
- Insert(nSteps, metrics)
- Insert(questionnaire)
- Request(student_applied)
- Configure_Date(email,date)
- Save(mail, date)

SPManager

- Request(questionnaire)
- Compile(questionnaire)
- Evaluate(questionnaire)
- Send_Results()

SP Presenter

- Request(date)
- Request(results)

SES API

- Trigger_Notification(mail,content)

Mail_Client

- Redirect(notification)

DBMS

- Select(email)
- Create(student)
- Create(company)
- Select_Student(email,password)
- Insert_Advice(form)
- Select_All(advices)
- Select_Advice(title)
- Select_Advice(subject,options)
- Select(notification)
- Select_Recommendation(mail)
- Select_Internship_Date(mail)
- Save_Application(mail,name)
- Create_Notification(mail,content)
- Modify_Application(mail,title)
- Insert(nStesp,metrics)
- Insert(mail,date)
- Insert(recommendation)
- Insert_Feedback(form)
- Insert(complain)
- Select_Internship(name)
- Remove_Internship(name)

MongoDB

- CreateCV(CV)
- Select_CV(email)
- Insert(questionnaire)
- Select(questionnaire)
- Save(questionnaire)

2.6 Data Logic Model

Starting from the high-level class diagram presented in the RASD, the logic scheme of the database is designed:

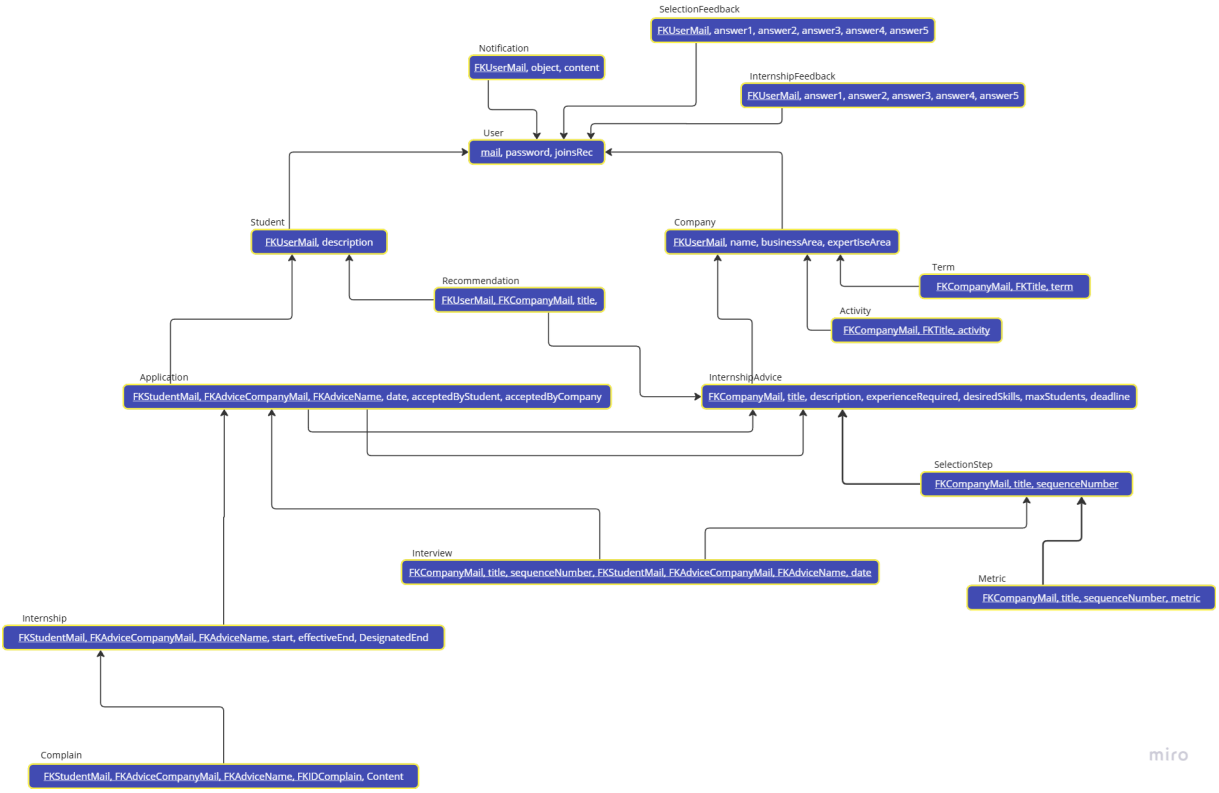


Figure 2.28: Data logic model scheme

Classes are almost one-to-one mapped with tables (with reification of vector attributes), the only relevant differences are:

- feedback-tables has an attribute for each questionnaire answer, this is not a concern since questionnaires are static (each questionnaire always has the same set of questions);
- the *Recommendation* table was added, in order to store the recommendation analysis results;
- selection process questionnaires and CVs are missing since they are stored in Mongo DB (they are far more suitable for an unstructured data representation).

2.7 Selected Architectural Styles and Patterns

2.7.1 3-tier architecture

As mentioned in section 2.1, the architecture used for the system implementation is a 3-tier architecture, which offers significant advantages in terms of modularity and scalability. Regarding the data layer, a rela-

tional database was chosen to manage structured data, while a non-relational database was adopted to handle unstructured data more flexibly, such as CVs and questionnaires with the responses from the selection process.

2.7.2 Model-View-Controller(MVC) pattern

As previously mentioned, S&C was designed to be developed as a web application. Consequently, one of the most commonly used patterns for building this type of application is the MVC pattern. This pattern divides the application into three main components:

- **Model**: Manages the data, business logic, and rules of the application. It acts as the interface between the database and the rest of the application.
- **View**: Handles the presentation layer, defining how the data is displayed to the user.
- **Controller**: Serves as an intermediary between the Model and the View, processing user inputs, updating the Model, and determining which View to display.

the use of this pattern once again provides significant advantages in terms of modularity and separation of concerns. Additionally, it enhances the reliability of the developed system and promotes reusability. Finally, having a clear separation between the Model, View, and Controller makes it much easier to perform functional testing of the system.

3 User Interface Design

While in the RASD we defined platform's proper UI structure, in this section we formalize navigation maps and present mock-up for each page:

- public pages:
  - registration pages:
    - **WelcomePage**
    - **RegistrationMenuPage**
    - **StudentRegistrationPage**
    - **CompanyRegistrationPage**
  - logic pages:
    - **LoginPage**
- private pages:
  - profile pages:
    - **StudentProfilePage**
    - **CompanyProfilePage**
  - notifications pages:
    - **NotificationsPage**
  - companies pages:
    - **CompanySearchPage**
  - advice-related pages:
    - **AdviceSearchPage**
    - **PersonalAdvicePage**
    - **PublishAdvicePage**
    - **AdviceDetailsPage**
    - **InterestingAdvicePage**
  - invites pages:
    - **InvitesPage**
  - applications pages:
    - **ApplicationsPage**
  - process-related pages:
    - **ConfigProcessPage**
    - **ProcessManagementPage**
    - **StepsManagementPage**
    - **ViewStatsPage**
    - **ProcessFinalizationPage**
  - interviews pages:
    - **InterviewsPage**
  - internships pages:
    - **InternshipsPage**
    - **InternshipDetailsPage**

Moreover, in this chapter we also specified how the sections defined in the User Interfaces section of the RASD are mapped onto the sections effectively designed.

3.1 Public pages

3.1.1 Public pages navigation map

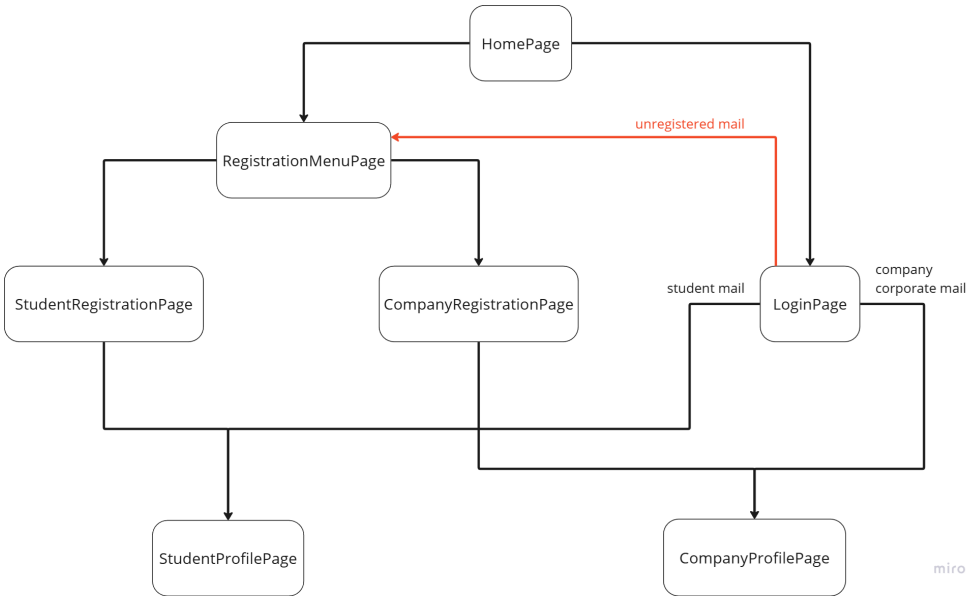


Figure 3.1: Public pages navigation map

3.1.2 Registration pages mock-up

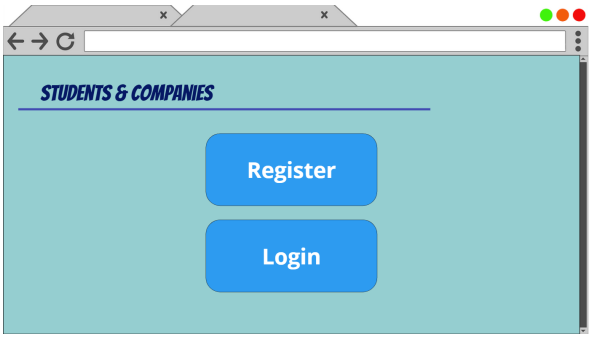


Figure 3.2: WelcomePage and RegistrationMenuPage mock-up

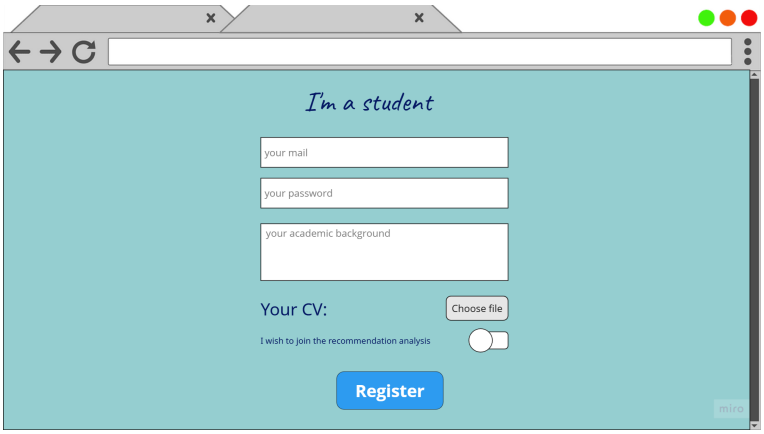


Figure 3.3: StudentRegistrationPage mock-up

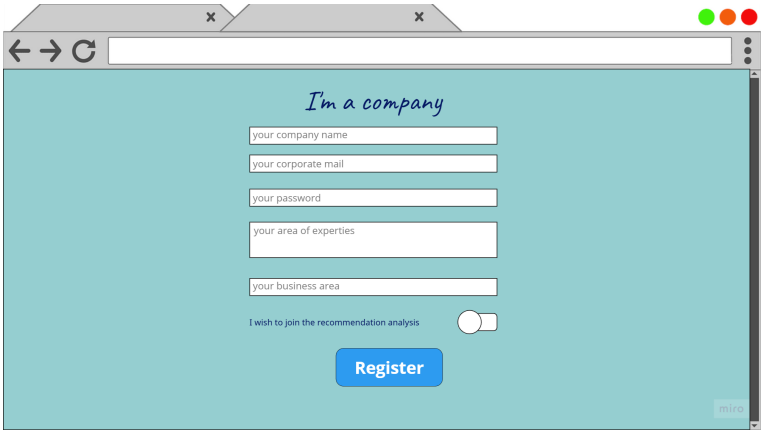


Figure 3.4: CompanyRegistrationPage mock-up

3.1.3 Login pages mock-up

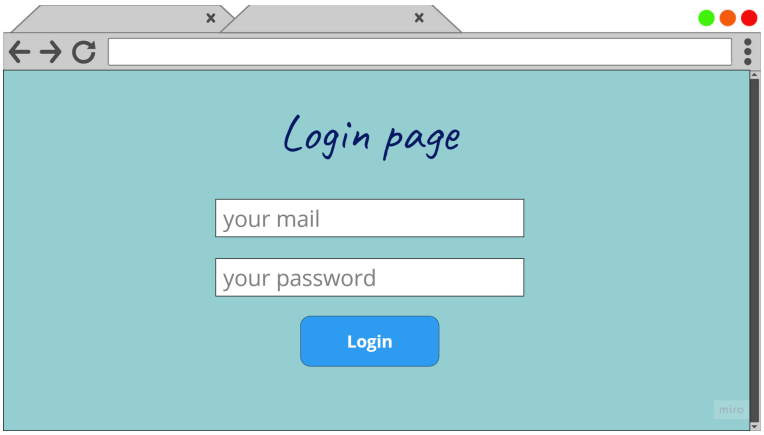


Figure 3.5: LoginPage mock-up

3.2 Private pages

Note that for what concerns user navigation in private pages:

- as mock-up show, private sections can be accessed by means of a side-bar (this explains why navigation maps are not connected graphs);
- navigation maps present the designated possibilities to navigate throw sections, while mock-ups often omit navigation buttons/links (e.g. InternshipDetailsPage CompanyProfilePage). Therefore, to implement navigation hyperlinks always refer to maps.

3.2.1 Private pages students navigation map

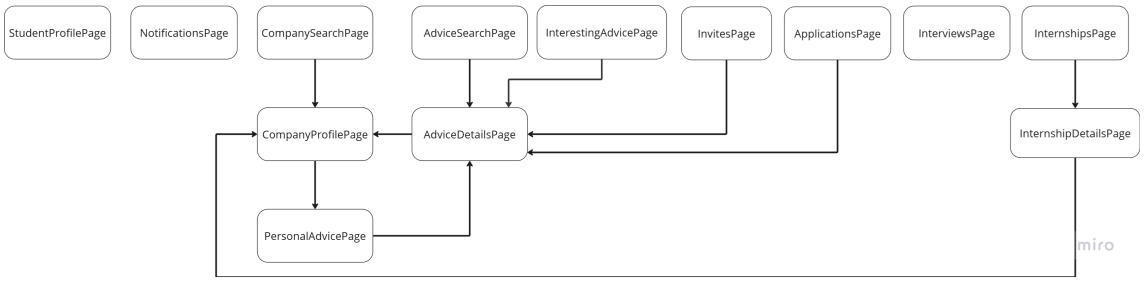


Figure 3.6: Private pages students navigation map

3.2.2 Private pages companies navigation map

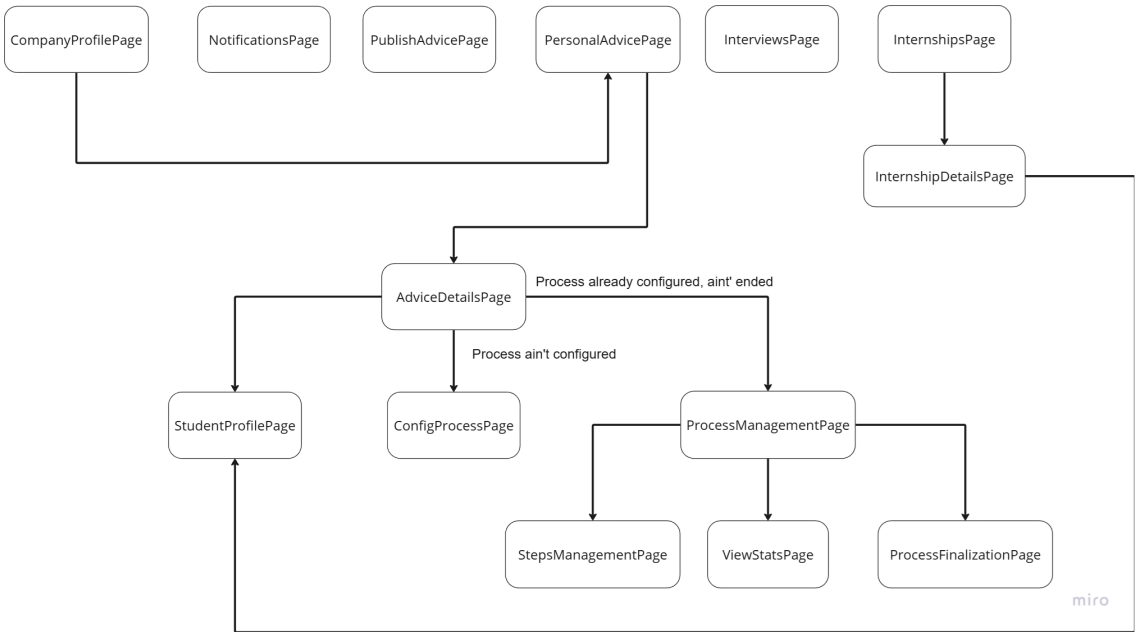


Figure 3.7: Private pages companies navigation map

3.2.3 Profile pages

3.2.3.1 Profile pages mock-up

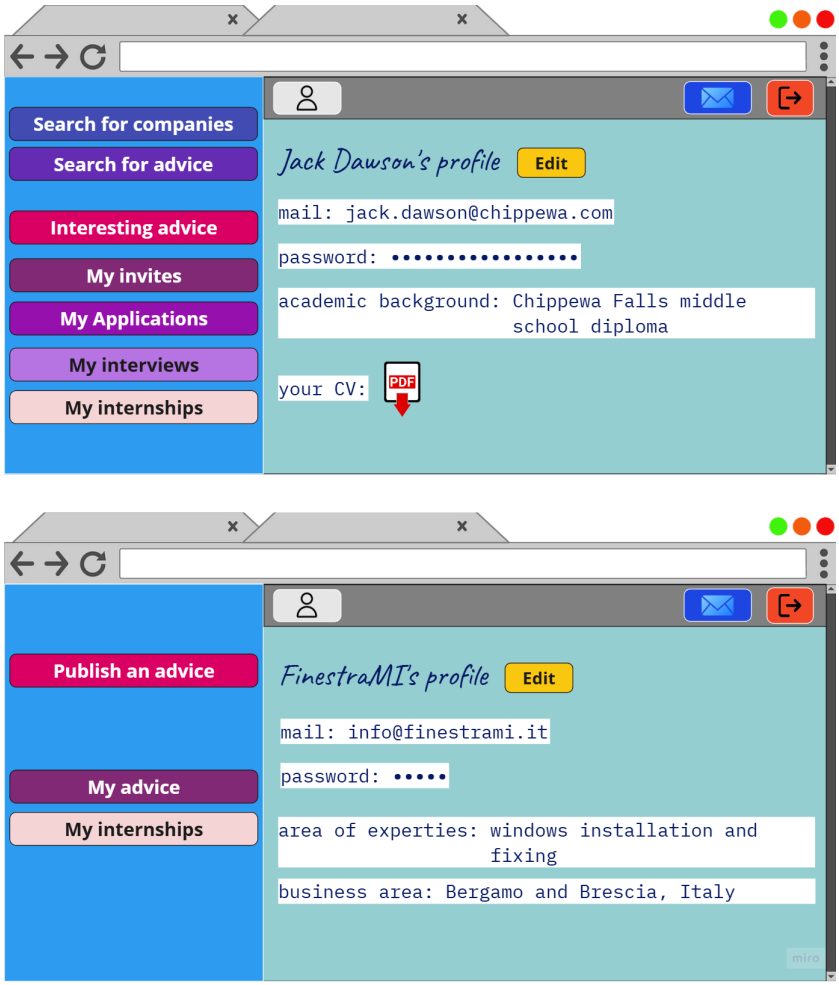


Figure 3.8: Profiles pages (students above, companies below) mock-up from the profile's owner view point

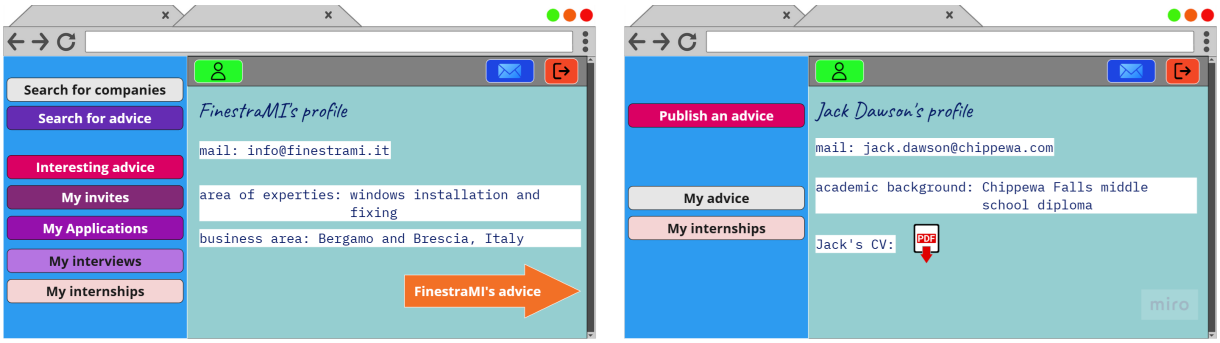


Figure 3.9: Profiles pages (students on the left, companies on the right) mock-up from other users view point

3.2.4 Notifications pages

3.2.4.1 Notifications pages mock-up

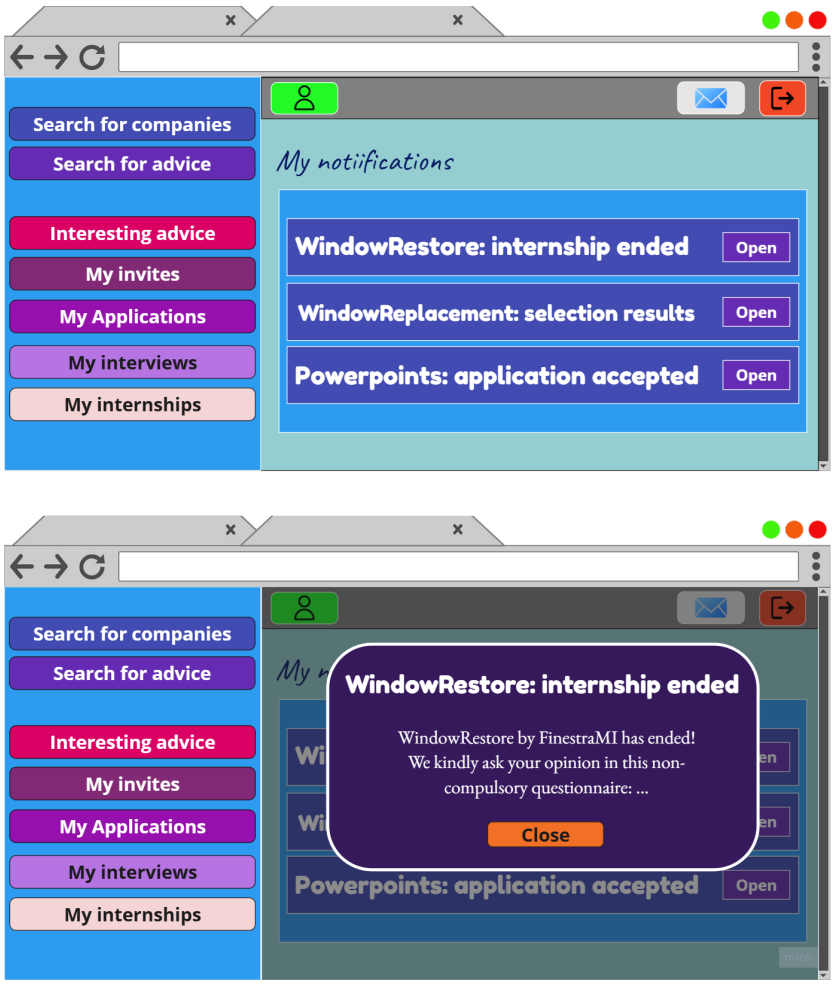


Figure 3.10: NotificationsPage and Notification modal box mock-up

Non-compulsory questionnaires are shown on a single window accessible from a link in the notification related to them:

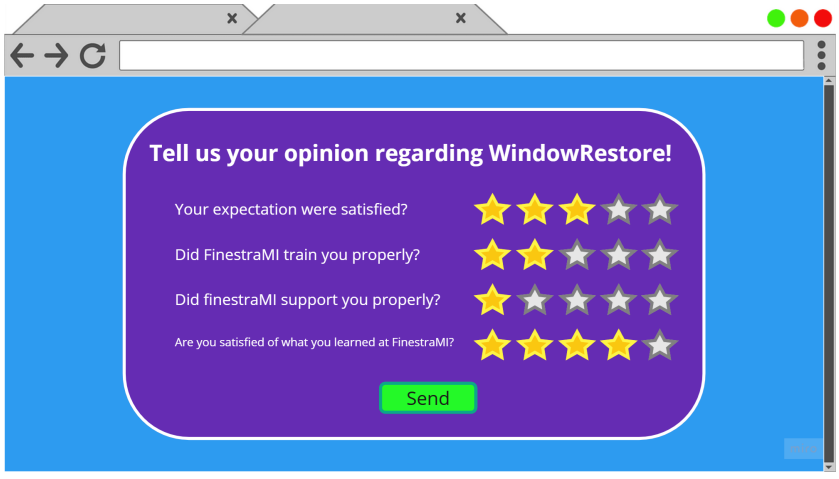


Figure 3.11: Feedback questionnaire window mock-up

3.2.5 Companies pages

3.2.5.1 Companies pages mock-up

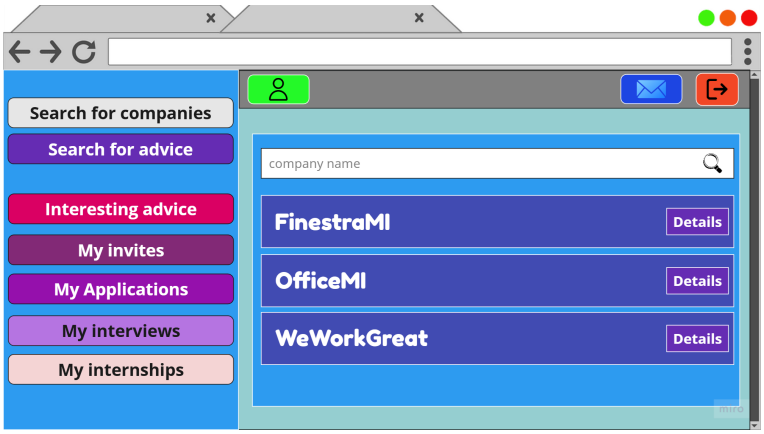


Figure 3.12: CompanySearchPage mock-up

3.2.6 Advice-related pages

3.2.6.1 Advice-related pages mock-up

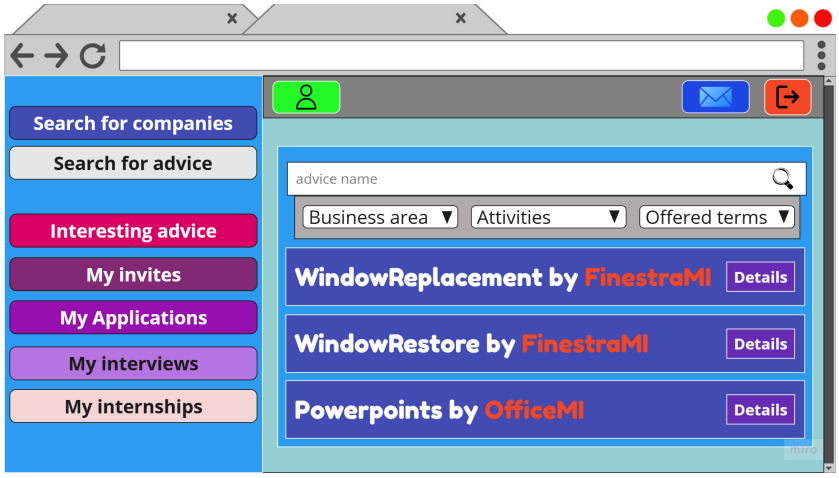


Figure 3.13: AdviceSearchPage mock-up

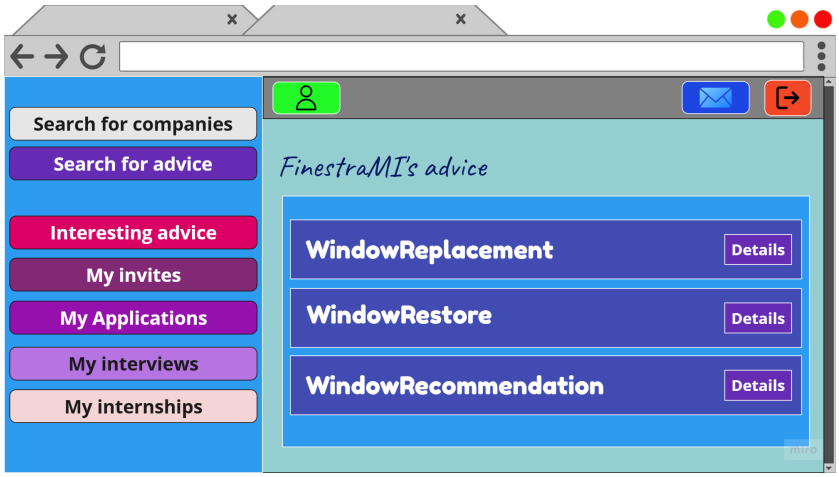


Figure 3.14: PersonalAdvicePage mock-up

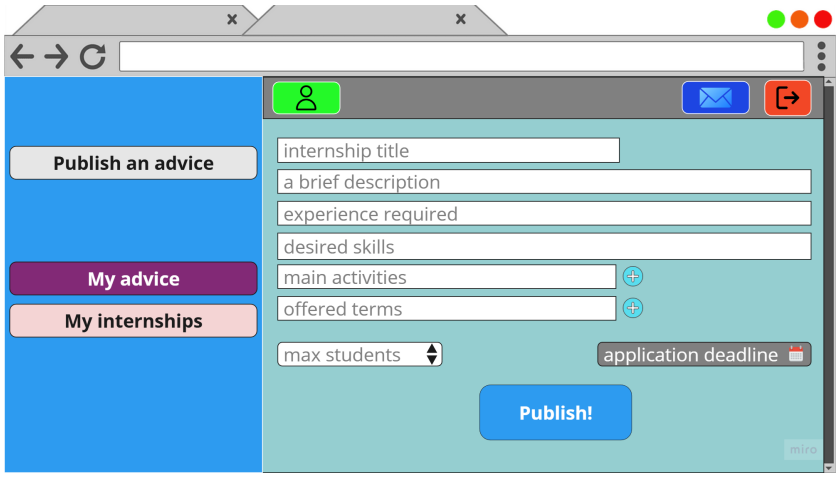


Figure 3.15: PublishAdvicePage mock-up

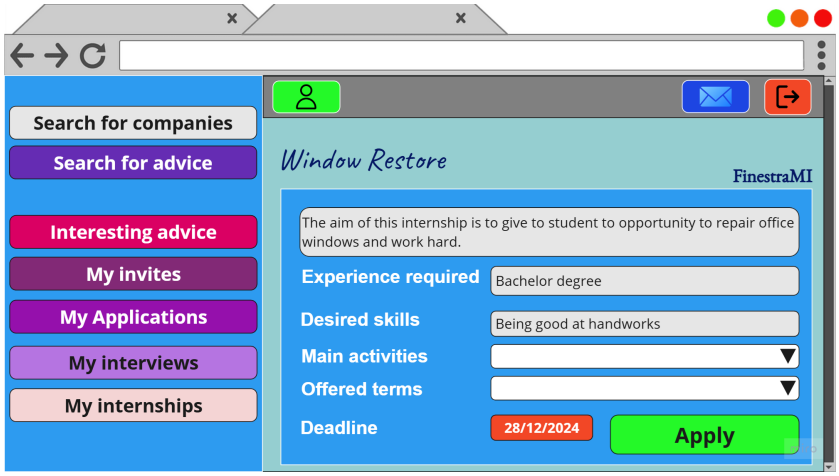


Figure 3.16: AdviceDetailsPage (students view-point) mock-up

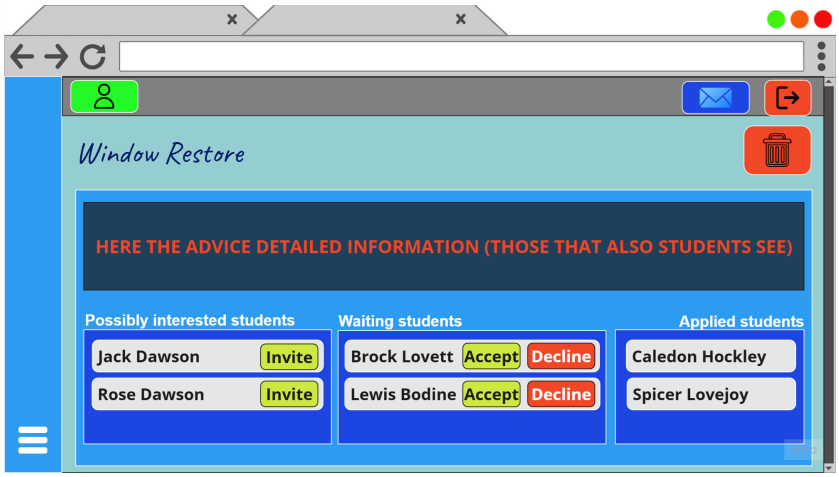


Figure 3.17: AdviceDetailsPage (company view-point) mock-up

The trash button will turn into a *Configure selection process* button once the advice deadline has last.

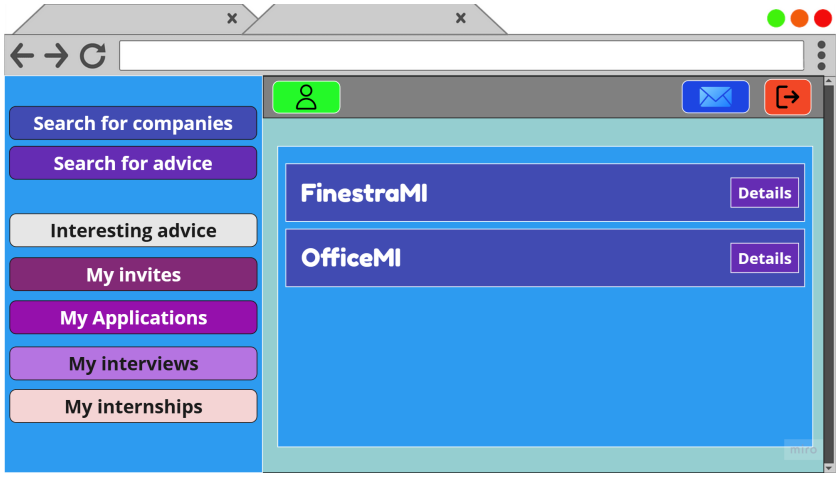


Figure 3.18: InterestingAdvicePage mock-up

3.2.7 Invites and Applications pages

3.2.7.1 Invites and Applications pages mock-up

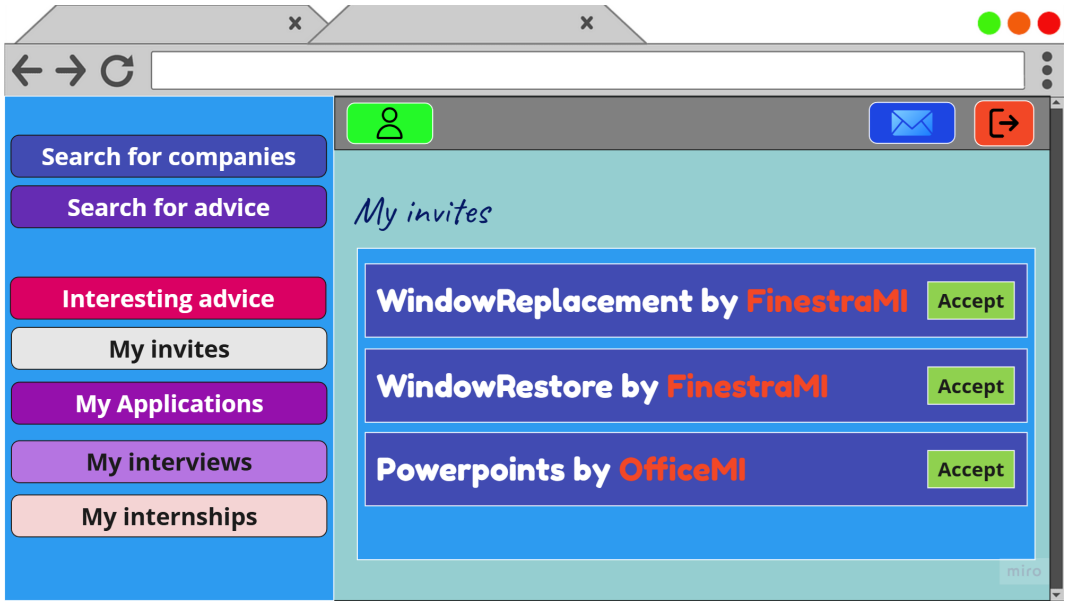


Figure 3.19: InvitesPage and ApplicationsPage mock-up

3.2.8 Process-related pages

3.2.8.1 Process-related pages mock-up

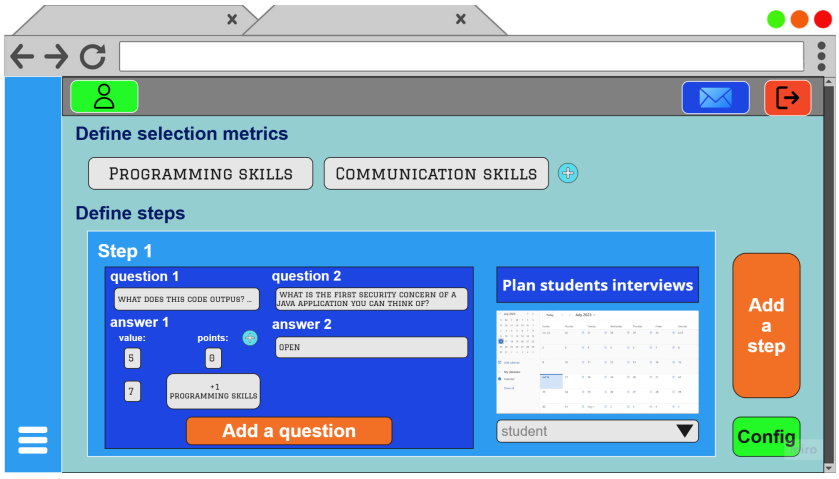


Figure 3.20: ConfigProcessPage mock-up

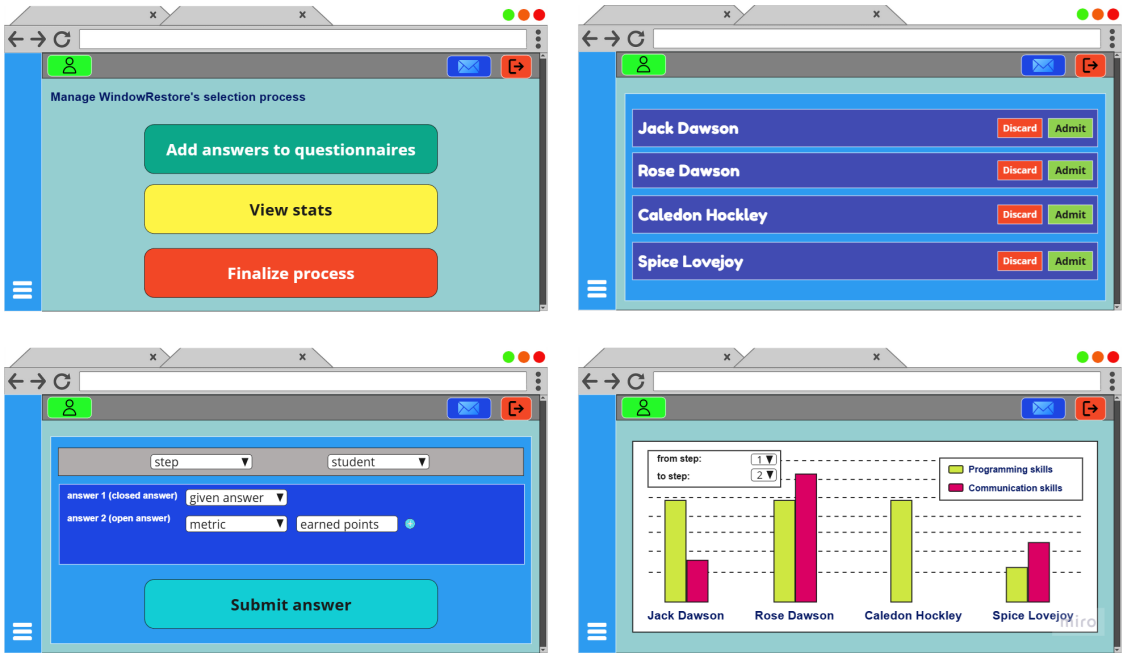


Figure 3.21: ProcessManagementPage, StepsManagementPage, ViewStatsPage and ProcessFinalizationPage mock-up

3.2.9 Interviews pages

3.2.9.1 Interviews pages mock-up

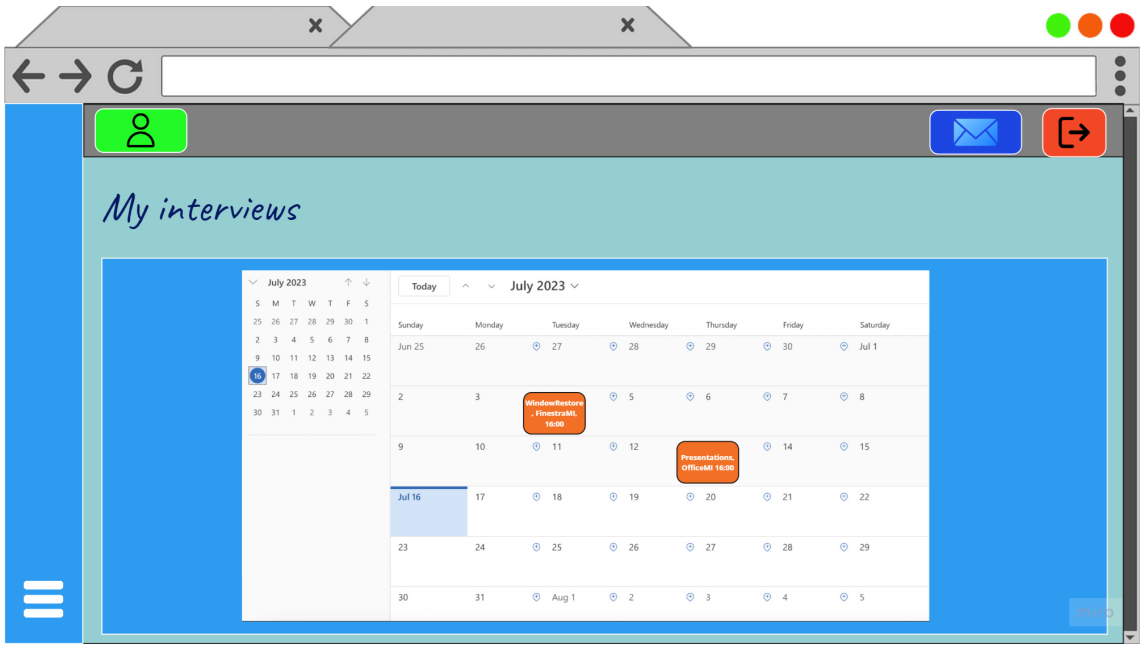


Figure 3.22: InterviewsPage mock-up

3.2.10 Internships pages

3.2.10.1 Internships pages mock-up

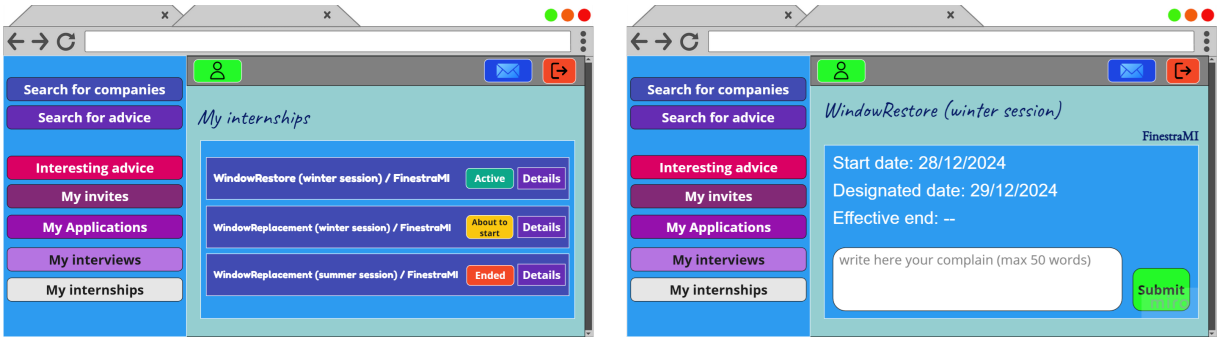


Figure 3.23: InternshipsPage and InternshipDetailsPage mock-up (from students view point)

In fact there is not much difference in what internships owners see. Companies can end instantly any ongoing internship that belongs to them.

3.3 Requirements UI sections mapping

This mapping considers the UI sections stated in the version 1.2 of the RASD, which has not been uploaded yet

Table 3.1: Requirements UI sections mapping

RASD UI section	DD pages
registration	WelcomePage, RegistrationMenuPage, StudentRegistrationPage, CompanyRegistrationPage
login	LoginPage
profiles	StudentProfilePage, CompanyProfilePage
notifications	NotificationsPage
internship (of a company)	PersonalAdvicePage
ongoing	InternshipsPage, InternshipDetailsPage
view internships	AdviceSearchPage
view companies	CompanySearchPage
interesting internships	InterestingAdvicePage
invites section	InvitesPage
publish new internship	PublishAdvicePage
selection process management	ConfigProcessPage, ProcessManagementPage, StepsManagementPage, ViewStatsPage, ProcessFinalizationPage

4 Requirements traceability

Here we detail how each requirement stated in the RASD is effectively ensured through the components:

4.1 General requirements

Table 4.1: General requirements traceability table

Requirement	Component(s)
R00000	NotificationGenerator, SES API

4.2 Requirements related to goal G1

Table 4.2: requirements related to goal G1 table

Requirement	Component(s)
R10101	AuthenticationManager, ProfileManager
R10102	AuthenticationManager, ProfileManager
R10103	AuthenticationManager, ProfileManager, MongoDB API (for CV)
R10104	AuthenticationManager, ProfileManager
R10105	AuthenticationManager
R10106	ProfileManager, MongoDB API (for CV)
R10107	ProfileManager
R10201	AuthenticationManager, ProfileManager
R10202	AuthenticationManager, ProfileManager
R10203	AuthenticationManager, ProfileManager
R10204	AuthenticationManager
R10205	ProfileManager
R10301	AdvicePublisher
R10302	AdvicePublisher
R10401	AdvicePresenter
R10402	ProfilePresenter
R10403	AdvicePresenter
R10404	AdvicePresenter
R10501	RecommendationInterface, RecommendationAnalyzer, RecommendationPresenter, NotificationGenerator
R10601	RecommendationInterface, RecommendationAnalyzer, RecommendationPresenter, NotificationGenerator
R10701	ApplicationManager
R10702	NotificationGenerator
R10801	ApplicationManager
R10901	ApplicationManager, NotificationGenerator
R11001	ApplicationManager
R11002	ApplicationManager
R12001	NotificationGenerator, NotificationPresenter, FeedbackManager
R12002	FeedbackManager
R11301	InternshipManager, FeedbackManager

4.3 Requirements related to goal G2

Table 4.3: Requirements related to goal G2 traceability table

Requirement	Component(s)
R20101	SPInitializer, MongoDB API
R20102	SPInitializer
R20201	SPPresenter, NotificationGenerator
R20202	SPManager, MongoDB API
R20203	SPManager
R20204	SPManager, MongoDB API
R20205	SPPresenter
R20206	SPManager
R20207	SPManager
R20209	SPManager, NotificationGenerator
R20210	SPManager

4.4 Requirements related to goal G3

Table 4.4: Requirements related to goal G3 traceability table

Requirement	Component(s)
R30101	InternshipManager
R30102	InternshipManager
R30103	InternshipManager
R30104	NotificationGenerator, InternshipManager
R30104	InternshipManager

5 Implemention, Integration and Test Plan

5.1 Overview

in this chapter is explained how S&C platform will be implemented and tested; regarding the implementation, the most important strategies used to carry out the project are described. As for the testing phase, it is essential because it allows for identifying the maximum number of bugs in the code written by the platform's developers.

5.2 Implementation Plan

This section explains the implementation strategy that is most advisable to follow in the development of the system. Specifically, the strategy follows a top-down approach, which involves developing the various sub-components of the architecture step by step, starting with those that need to be integrated first and progressing to the last ones. Before being added to the architecture, these sub-components must be thoroughly tested to ensure their implementation is free of bugs and integrates well with the other components already present in the architecture. Once validated, the sub-component is incorporated into the architecture.

This strategy allows multiple development teams to work in parallel: one team can focus on creating a single component, testing it, and then integrating it into the software architecture.

5.2.1 Feature Identification

The features to be implemented are described, starting from the requirements defined in the RASD document and, in particular, from the identified product functions. Below, we summarize the most relevant ones:

[F1]sign-up and login
These two features are essential to test in order to ensure that users can register or log in correctly. It is important to separately test the registration processes for companies and students, as they provide different data during registration (for instance, students upload their resumes, interfacing with MongoDB, whereas companies do not).

The subsequent login functionality can be implemented and tested for a generic user since the functionality is similar for both students and companies.

[F2]recommendation mechanism
This is the core feature of the system: it provides recommendations of interesting students for companies and relevant internship advice for students. It is crucial to properly test this functionality to prevent the wrong students from being recommended to companies and irrelevant internship advice from being provided to students.

[F3]Internship proposal management
This is the functionality to be developed to allow students to apply for internships and companies to send application proposals to students. Implementing this feature also enables students to proactively search for interesting internship advice through the various methods already presented (searching for all internship advice, searching for all companies, or searching for advice by selecting different options).

[F4]selection process management and internship monitoring
This is another key functionality of the system: it allows companies to configure a selection process, save students' responses during interviews, and choose the appropriate students for their internships. Subsequently, both students and companies can monitor the progress of the internship they are involved in. It is necessary to properly test this functionality because it forms the basis for selecting students for an internship.

[F5]notification management
It is mandatory to implement a notification mechanism involving the email service: each notification triggers the sending of an email to the respective user. The platform also includes a notifications section where users can view more detailed content of the notifications.

5.3 Component Integration and Testing

This section details the steps to follow for the development of the platform. It lists the order in which each component must be developed and validated before being integrated into the overall architecture.

UI and Data Layer

The first part to be implemented is, of course, the relational database and the MongoDB database for managing unstructured data. These databases will store the data from our platform. In parallel, the UI can be developed, which is necessary to enable user interaction with the system.

Once the presentation layer and data layer have been implemented, the application layer can be developed. The components should be developed and tested in the following order:

Sign-up and Login

The first two components to implement and test are the *Authentication Manager* and the *Profile Manager*, as the initial functionality to be provided is user registration and authentication.

Internship Proposal Management, Recommendation and Notification Management

Subsequently, the components *Advice Publisher, Advice Presenter, and Profile Presenter* are added. These components manage the publication of advice and subsequent application requests. In parallel, the components related to the recommendation system (*Recommendation Interface, Analyzer, and Presenter*) and notification management (*Notification Presenter and Notification Generator*) must also be developed.

This integration is crucial because the application process and the sending of proposals must be supported by recommendations received by the user, delivered through the notification system. Therefore, it is necessary to conduct integrated testing of these components before adding them to the architecture.

Once validated, the system allows users to publish advice and apply for them. Additionally, users can receive recommendations related to internships or students, if they choose to opt in for these notifications.

Selection Process Management

The components related to the management of the selection process— *SP Initializer, SP Manager, and SP Presenter*—are then implemented and thoroughly tested. These components must be integrated and tested in conjunction with the notification system, as the selection process sends results to users via notifications.

feedback and complaint management

Subsequently, components related to feedback management are added, as feedback must be collected at the end of the selection process. The Feedback Manager component is introduced for this purpose. Additionally, the Internship Manager component is added to handle ongoing internships.

These components must be thoroughly tested and validated, as they also enable the system to manage user complaints effectively.

5.4 System Testing

As mentioned earlier, S&C must be tested to ensure that the implemented features align with the platform's requirements. During the development phase, each component must be tested individually as it is created before being integrated into the architecture. However, as previously stated, it is not sufficient to test and validate individual components alone. It is essential to test a set of these sub-components together and, ultimately, the entire developed system.

The primary goal is to verify that the system meets all the requirements defined in the RASD document (both functional and non-functional). The following steps can be followed:

- **Functional Testing**: the idea is to run the software as described in the use-cases in RASD document and verified if they are satisfied
- **Performance Testing**: the objective is find the bottlenecks that affect in negative way the time response of the system; to perform this tests is necessary load the system with the expected workload and identify optimizations if possible
- **Usability Testing**: it is a method of testing the features of a website, app, or other digital product by observing real users as they attempt to complete tasks on it.

When testing the system, the involvement of system stakeholders becomes crucial. They can first participate in an alpha test, followed by a beta test, before the official release of the platform. The feedback obtained from these tests is essential to identify areas that need improvement or adjustment prior to the official launch.

- **Alpha Testing**: This initial phase of testing is typically conducted by a select group of internal stakeholders or developers. It focuses on identifying major bugs or functional issues early on. The goal is to ensure the system works as intended before being presented to a larger audience.
- **Beta Testing**: This phase is conducted with a larger group of users, often including external stakeholders, who test the system under real-world conditions. Feedback from beta testers helps identify any remaining issues and provides valuable insights into user experience, usability, and system performance.

The feedback gathered from both the alpha and beta tests plays a crucial role in refining the platform, making necessary improvements, and ensuring that it is robust and ready for the official release.

6 Effort Spent

Table 6.1: Effort spent overview

	Andrea		Alessandro	
Week	**Hours**	**Category**	**Hours**	**Category**
23/12/2024 - 29/12/2024	16	User interface and high-level structure	4	high-level structure
30/12/2024 - 05/01/2025	2	Data logic model	12	Introduction, data logic model, sequence diagrams
06/01/2025 - 07/01/2025	8	Final revision, requirements mapping, component view, deployment view, effort spent and software used compilation	8	Final revision, requirements mapping, integration and test plan, component view
TOTAL	**26**		**24**	

7 Software Used

Table 7.1: Software used overview

Software	Version	Usage
TeXstudio	4.6.3	Writing Latex
PdfLaTeX	-	Latex compilation
XeLaTex	-	Latex compilation
Astah UML	9.2.0/0248cd	UML diagrams
Alloy Analyzer	6.1.0	Alloy modeling
miro.com	-	User interface maps and mock-up, high-level view, logic scheme
draw.io	-	Component diagrams, deployment diagram

This document was written over the template kaobook designed by Federico Marotta (https://github.com/fmarotta/kaobook), with few adjustments by us.

———————————————————————