POLITECNICO
MILANO 1863

S&C project E. S. Gribaldo, F. Rosa

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	Edoardo Silvio Gribaldo, Federico Rosa
Version:	1.0
Date:	07/01/2025
Download page:	https://github.com/edogriba/GribaldoRosa
Copyright:	Copyright © 2025, E. S. Gribaldo, F. Rosa – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

In today's job market, when looking for a job, having a previous work experience has become an important criteria of evaluation, therefore more and more students try to find an internship. On the other side, many companies are willing to invest into young talents in order to build a skilled workforce that can help them to drive innovation and foster and secure a competitive edge in the future.
Student&Companies is a platform that offers to students the possibility to look up for internships and to companies the possibility of advertising. The matchmaking process of student and companies is enabled through a (proprietary) recommendation algorithm, that creates the most suited matches, through the collection of statistics and feedback.
Moreover Students&Companies allows for the management of the selection process and the monitoring of the execution of the internships. The goal is to provide a platform for both students and companies to make the process of finding, offering and tracking internships easier and more efficient for both parties.

1.2 Scope

The platform S&C has three major stakeholders: students, companies, and universities. On one hand, students upload their CV and indicate their skills and domain of interest on the platform. Then they can proactively look out for internships or they can get suggestions from the platform, when new positions become available. On the other hand, companies can open internship positions, with the related domain and required skills, advertise those positions but also get notified when profiles of students, aligned with the requirements of one of their open positions, appear on the platform. Once both parties express interest on the platform, a contact between the two is established. Then the management of the set up and the conduct of interview starts. When a candidate is selected, the platform tracks the conduction of the internship through feedback, complaints and exchange of information by both parties. Indeed a specific space of the website is devoted to communication between a student and company that have started an internship process together. The system run analytics on statistics collected by the platform itself on both students and companies to guarantee the best match possible between the two. Universities are able to view and monitor their students' internships without having access to any kind of interaction with them.

1.3 Revision History

The following is the revision history of the document:

1. Version 1.0 (07/01/25)

1.4 Reference Documents

This document is based on the following reference documents:

- The document of the 2024/2025 assignment for RASD and DD
- The slides of the course found on WeBeep

1.5 Document Structure

1. **Introduction**
   The introduction identifies the purpose and the scope of the project, explains acronyms, abbreviations that will be used in the document, specifies the version of the document and the reference documents on which the current version it is based.

2. **Architectural Design**
   The architectural design gives a general overview on the structure of the project. First, it describes the application at the highest level possible and lists the taken choices. Then it goes a little bit deeper and focuses on components and their interactions

3. **User Interface Design**
   Here there are some snippets of user interfaces of the website

4. **Requirements Traceability**
   In the requirements traceability section, there are tables clearly linking components and the requirements they contribute to satisfy.

5. **Implementation, Integration and Test Plan Traceability**
   This section presents a precise plan to implement, test and integrate the code written for the S&C platform

6. **Effort spent**
   The effort spent is described by a simple table

7. **References**
   It has pointers to all the resources used or cited in this project

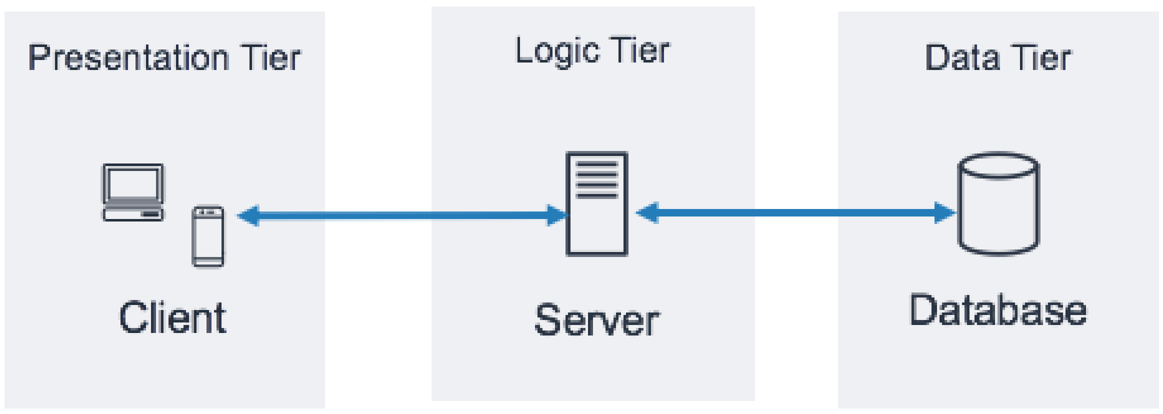
2 Architectural Design

2.1 Overview

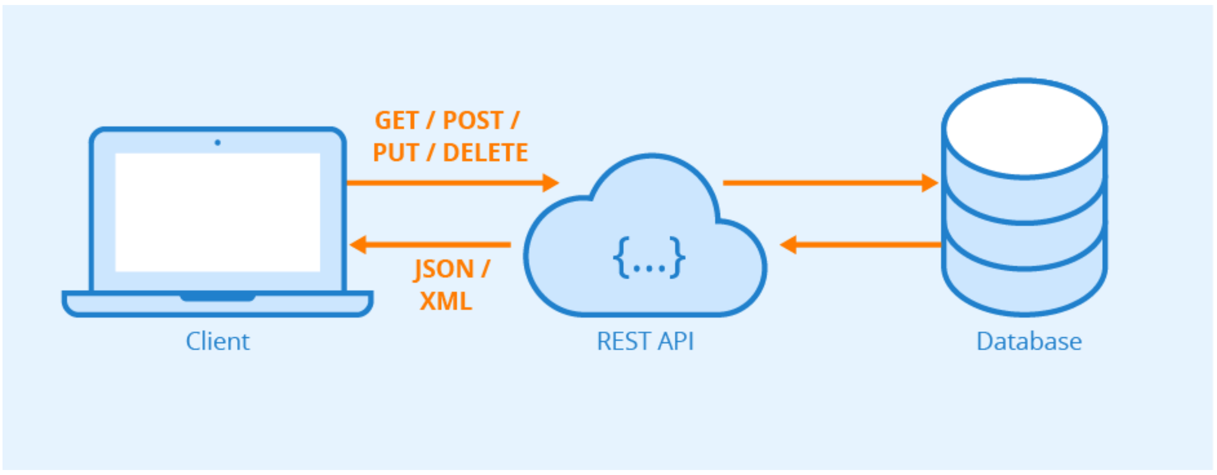
We chose the three tier architecture for S&C because we believe that it is a solid choice aligned with our goals and requirements. The architecture is divided in three:

1. **Presentation Tier**: This is essentially the user interface
2. **Application Tier**: This layer is where the data are transformed and undergo a set of processes to make them available to the user interface
3. **Data Tier**: This layer is where the data are stored and managed remotely.

This is the visualization of the 3-tier logic. [1]



We also chose to use REST APIs to make the presentation layer and application layer communicate. The following image displays how an API works[2]:



2.2 Component View

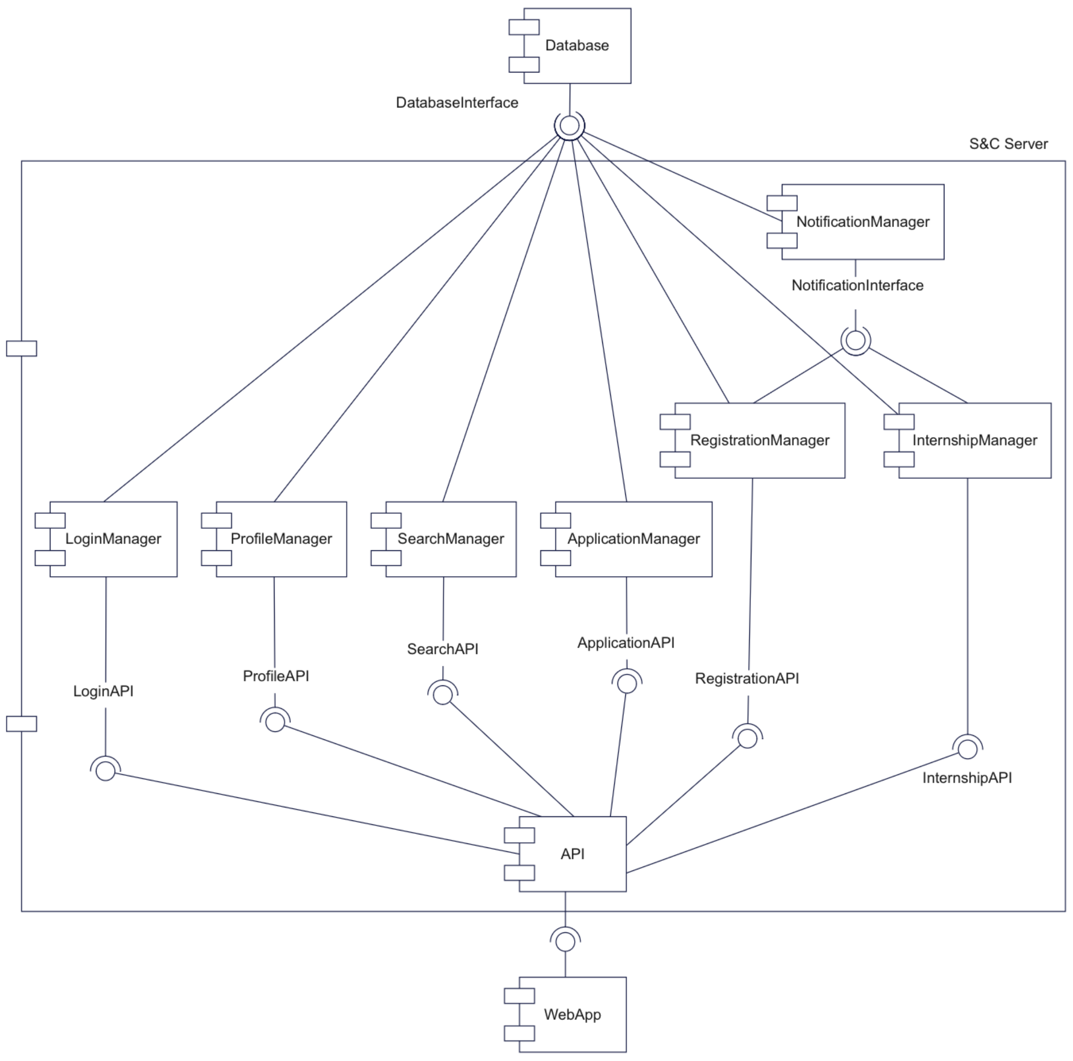
This section aims to give a general representation of the core components of the system and how they interact together without focusing on the internal details. Indeed it is important to grasp the division in modules and the interactions among them. In the component view image it is clearly distinguishable the 3-tier architecture and composed by the WebApp, the Back-End composed by all the modules depicted and listed below and the Database. Since all the logic is implemented in the middle layer, namely the S&C Server, that is where all the modules are and interact. The main components of the back-end are:

1. **WebApp**: it is the component in charge of providing the user interface for interacting with the system. It relies on the API to fetch and send data for various operations.

2. **S&C Server**

   - **SearchManager**: it is the component in charge of receiving and handling the search requests about the internship positions issued by the students
   - **RegistrationManager**: it is the component in charge of managing the registration of the users.
   - **LoginManager**: it is the component in charge of the login and the logout of the users.
   - **NotificationManager**: it is the component in charge of sending the notifications to users including push notifications.
   - **ProfileManager**: it is the component in charge of handling user profile information. It is designed to interact with all the other components in order to keep the profile of the user up to date.
   - **InternshipManager**: it is the component in charge of managing the internships and the internship positions. It manages the internship status, the communication, the assessment process (if needed), publishing of news related to the internships.
   - **ApplicationManager**: it is the component in charge of keeping track of the status application and its details.
   - **API**: it is the component that allows the back-end to interact with the requests from the front-end.

3. **Database**: it is the component in charge of storing all the information relevant to the platform.

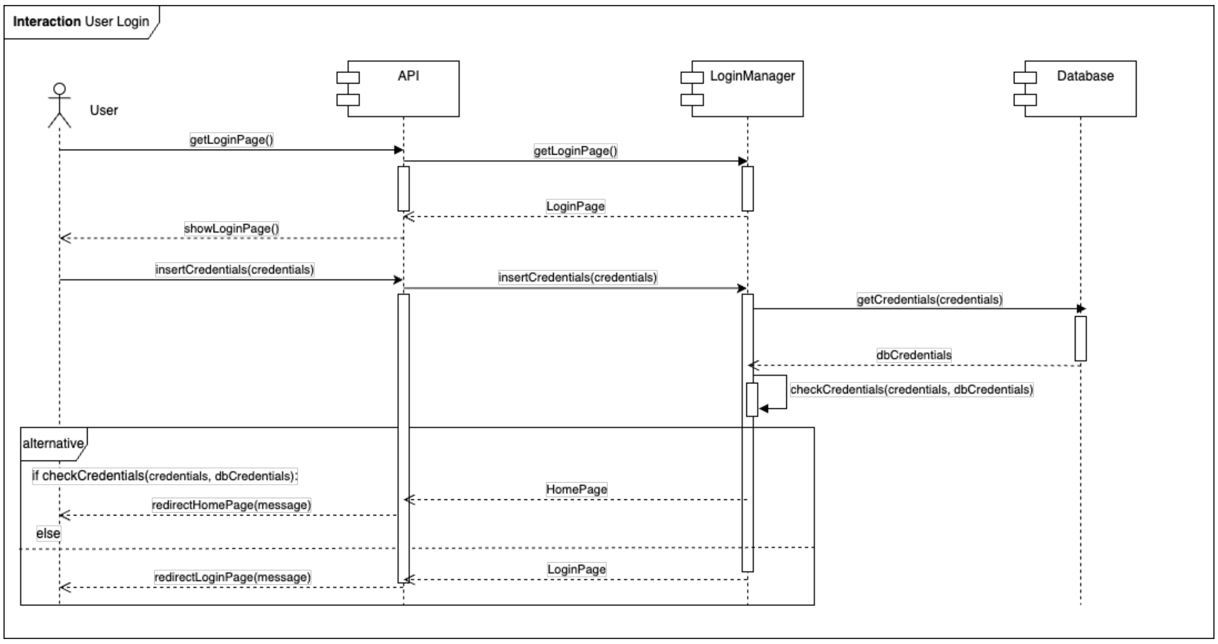


Component View Diagram

2.3 Runtime View

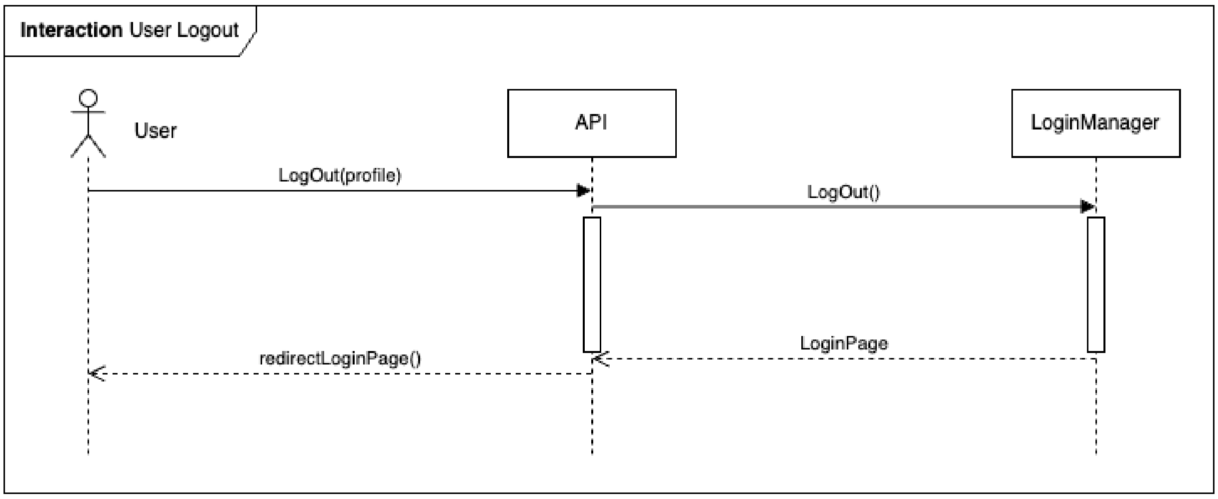
In this subsection there will be listed the interactions diagram along with their textual descriptions.

1. **User Login**


Interaction User Login
User
API
LoginManager
Database
getLoginPage()
getLoginPage()
LoginPage
showLoginPage()
insertCredentials(credentials)
insertCredentials(credentials)
getCredentials(credentials)
dbCredentials
checkCredentials(credentials, dbCredentials)
alternative
if checkCredentials(credentials, dbCredentials):
HomePage
redirectHomePage(message)
else
LoginPage
redirectLoginPage(message)

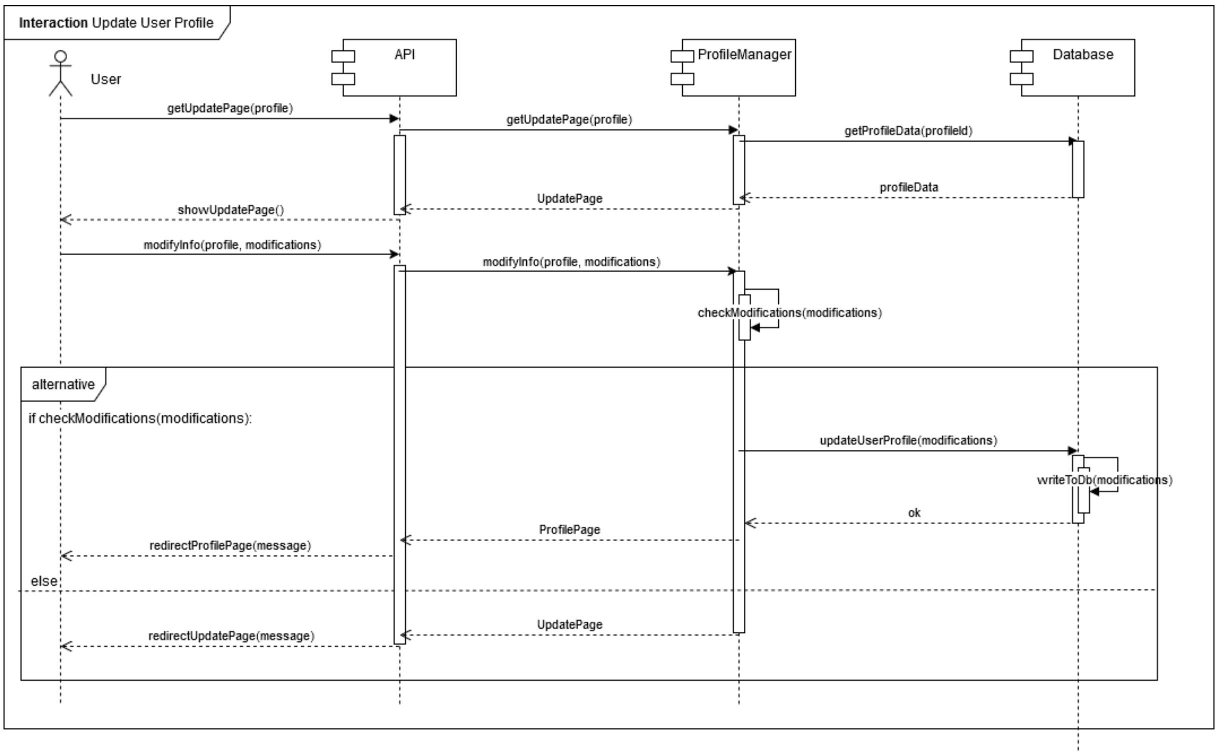
The interaction diagram shows the process of a user logging into S&C. The sequence begins with the user requesting the login page through the **API**. The **API** retrieves and displays the login page for the user to interact with. Once the user inputs their credentials, such as a username and password, these are submitted to the **API**. The **API** then forwards the credentials to the **LoginManager**, which is responsible for handling the authentication process. The **LoginManager** communicates with the **Database** to retrieve the stored credentials associated with the submitted user details. After getting the stored credentials, the **LoginManager** compares them with the provided ones to check the user's identity. If the credentials match, the validation is successful and the user is redirected to the homepage. If the credentials do not match, an error is raised and the user is redirected to the login page.

2. **User Logout**



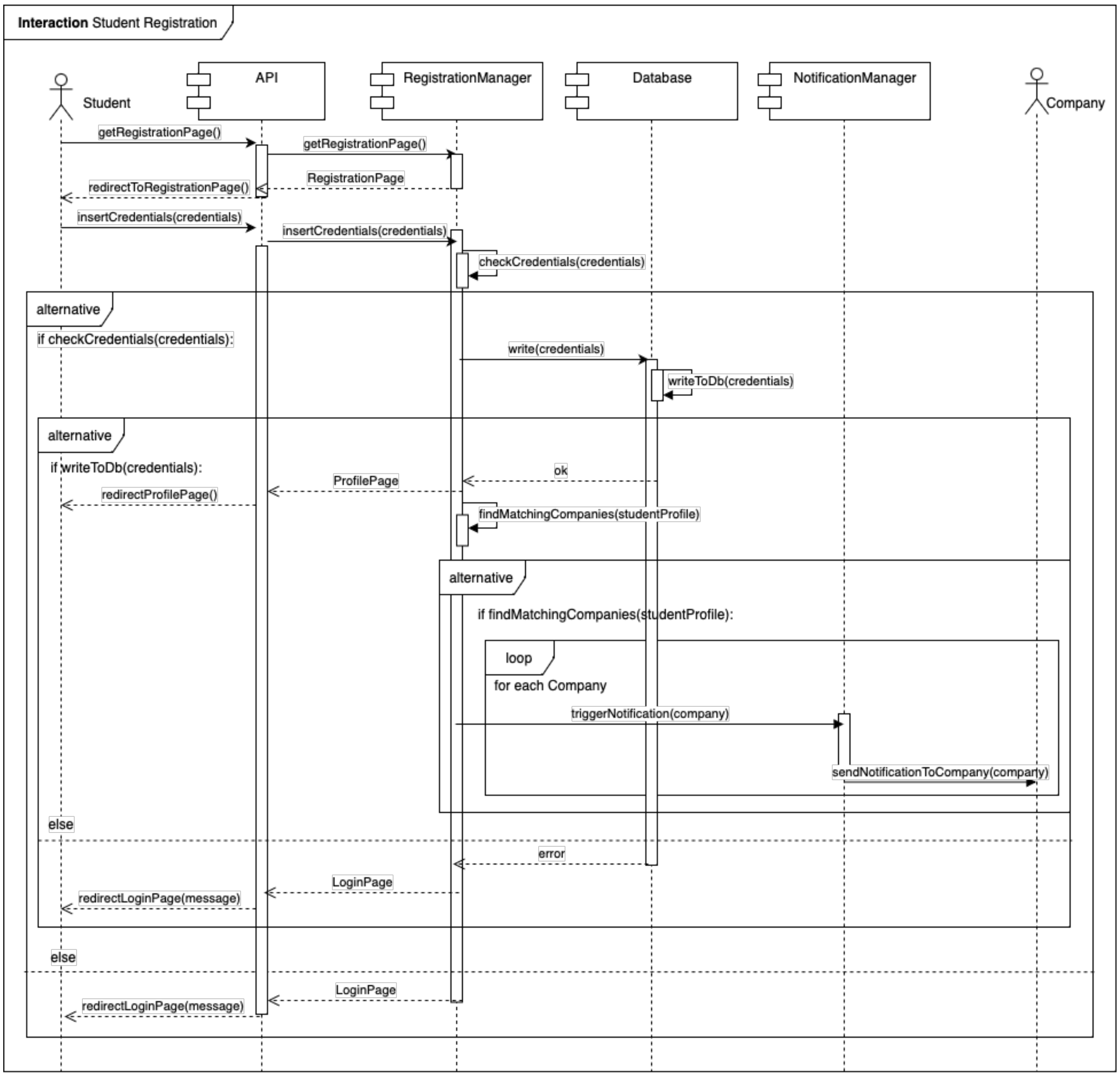
The interaction diagram describes the process of a user logging out from S&C The sequence begins with the user starting the logout operation by sending a request to the **API**, specifying their profile. The **API** forwards this request to the **LoginManager**, which is responsible for handling session management and user authentication. The **LoginManager** then closes the user's session and clears any associated session data. Once the session is successfully invalidated, the **LoginManager** informs the **API** that the user has been logged out. The **API** then redirects the user to the login page, notifies the completion of the logout process.

3. **User Update**


Interaction Update User Profile
User
API
ProfileManager
Database
getUpdatePage(profile)
getUpdatePage(profile)
getProfileData(profileId)
profileData
UpdatePage
showUpdatePage()
modifyInfo(profile, modifications)
modifyInfo(profile, modifications)
checkModifications(modifications)
alternative
if checkModifications(modifications):
updateUserProfile(modifications)
writeToDb(modifications)
ok
ProfilePage
redirectProfilePage(message)
else
UpdatePage
redirectUpdatePage(message)

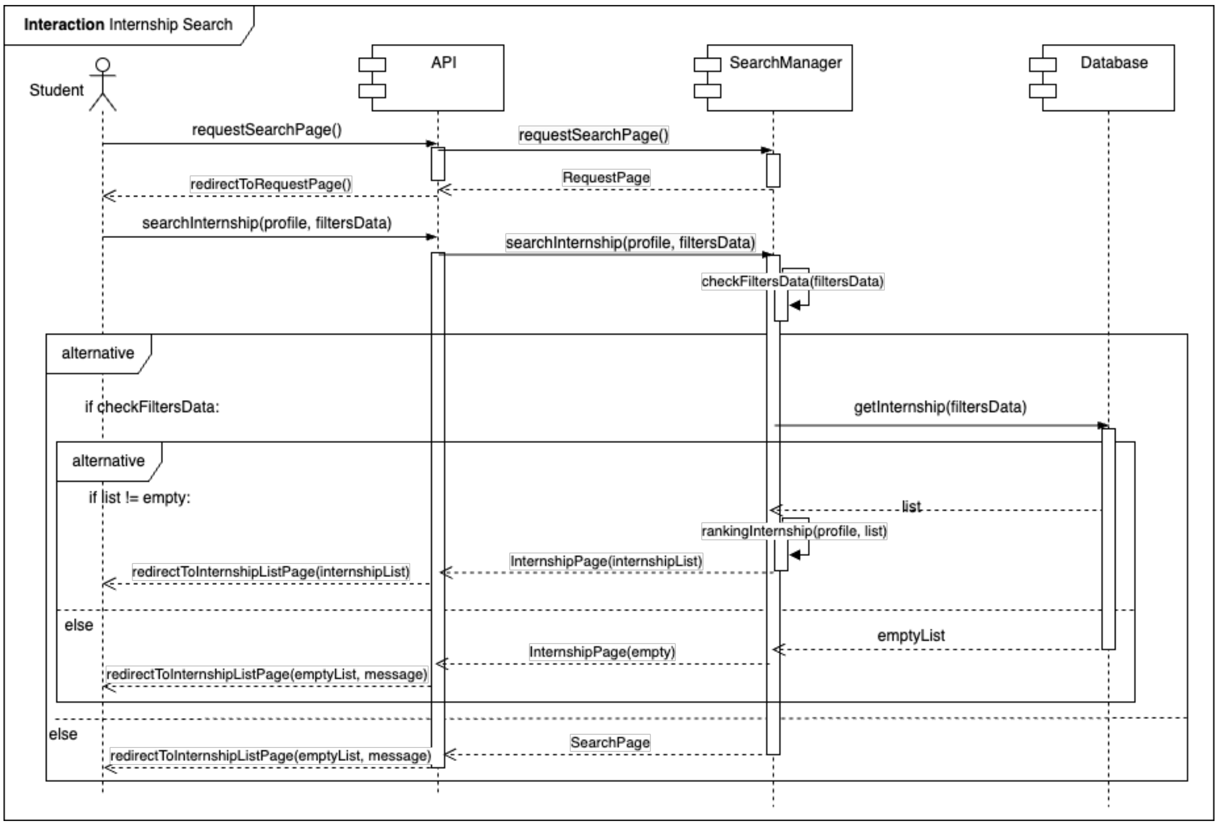
The interaction diagram shows the process of updating a user profile in S&C. The sequence begins with the user requesting the profile update page by sending their profile information to the **API**. The **API** forwards this request to the **ProfileManager**, which retrieves the data that can be updated from the **Database**. The **API** then displays the update page with the pre-compiled data to the user. Once the user makes changes to their profile and submits the modifications, the **API** sends the updated information to the **ProfileManager** for processing. The **ProfileManager** validates the submitted modifications through its internal logic. If the modifications are valid, the **ProfileManager** writes the changes to the **Database** and the user is redirected to their profile page. If the update fails due to an error, the API redirects the user back to the update page with an error message.

4. **Student Registration**


Interaction Student Registration
Student
API
RegistrationManager
Database
NotificationManager
Company
getRegistrationPage()
getRegistrationPage()
RegistrationPage
redirectToRegistrationPage()
insertCredentials(credentials)
insertCredentials(credentials)
checkCredentials(credentials)
alternative
if checkCredentials(credentials):
write(credentials)
writeToDb(credentials)
alternative
if writeToDb(credentials):
ok
ProfilePage
redirectProfilePage()
findMatchingCompanies(studentProfile)
alternative
if findMatchingCompanies(studentProfile):
loop
for each Company
triggerNotification(company)
sendNotificationToCompany(company)
else
error
LoginPage
redirectLoginPage(message)
else
LoginPage
redirectLoginPage(message)

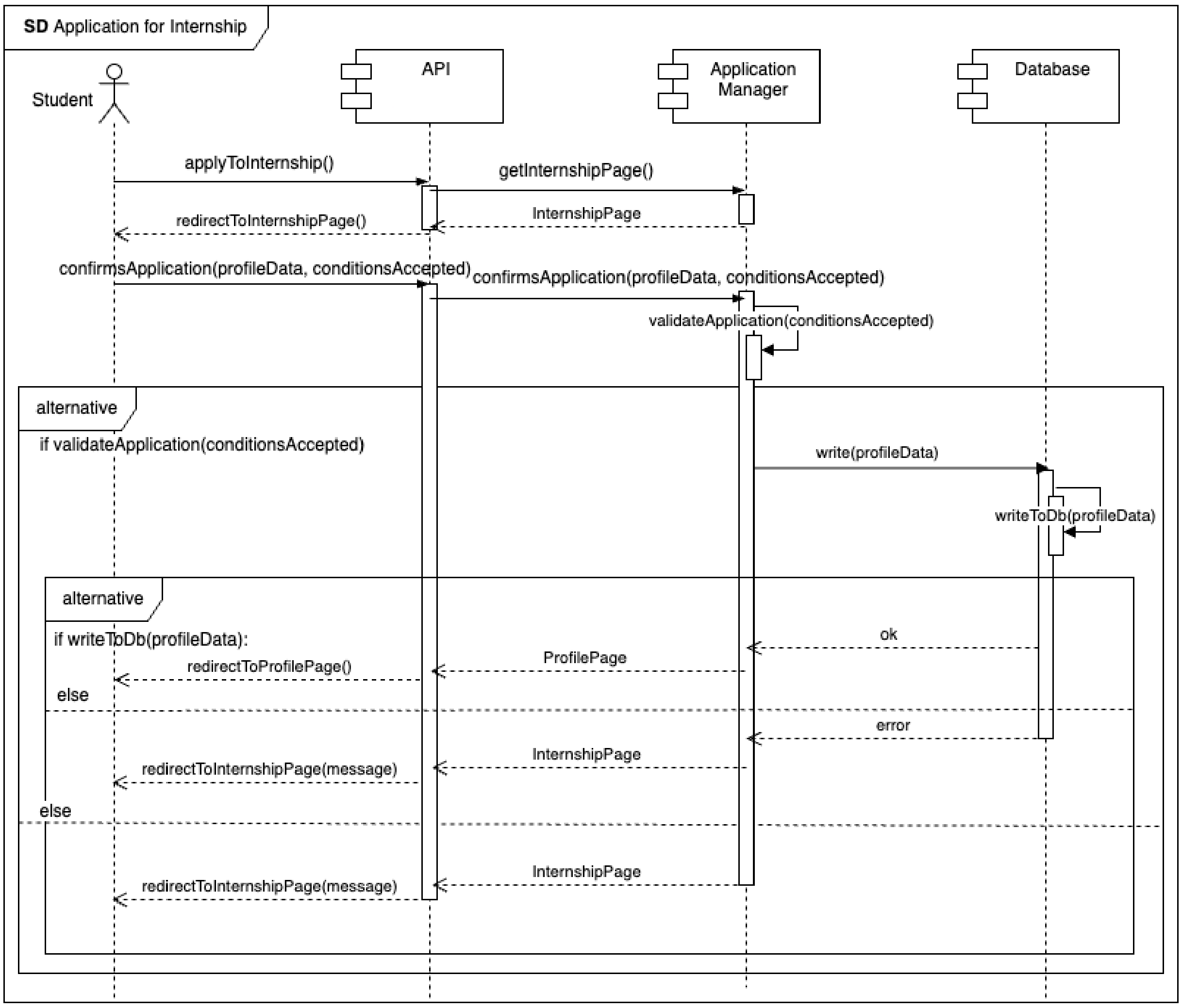
The interaction diagram shows the process of a student registering within S&C. The process begins when the student requests access to the registration page through the **API**. The **API** retrieves the registration interface by interacting with the **RegistrationManager**, which provides the appropriate page. The **API** then displays the registration page to the student. Once the student submits their credentials, the **API** forwards the submitted data to the **RegistrationManager**, which validates the credentials to ensure they meet the required standards. If the credentials are valid, the **RegistrationManager** saves the student's data to the Database. Upon successful storage, the **RegistrationManager** proceeds to find matching companies based on the student's profile. If relevant companies are identified, the **RegistrationManager** triggers the **NotificationManager** to send notifications to these companies, informing them of the newly available student profile. If the credentials are evaluated correct, the **API** redirects the student to their profile page. Otherwise the **RegistrationManager** communicates the problem to the **API**, which redirects the student back to the login page displaying an error message.

5. **Internship Search**


Interaction Internship Search
Student
API
SearchManager
Database
requestSearchPage()
requestSearchPage()
RequestPage
redirectToRequestPage()
searchInternship(profile, filtersData)
searchInternship(profile, filtersData)
checkFiltersData(filtersData)
alternative
if checkFiltersData:
getInternship(filtersData)
alternative
if list != empty:
list
rankingInternship(profile, list)
InternshipPage(internshipList)
redirectToInternshipListPage(internshipList)
else
emptyList
InternshipPage(empty)
redirectToInternshipListPage(emptyList, message)
else
SearchPage
redirectToInternshipListPage(emptyList, message)

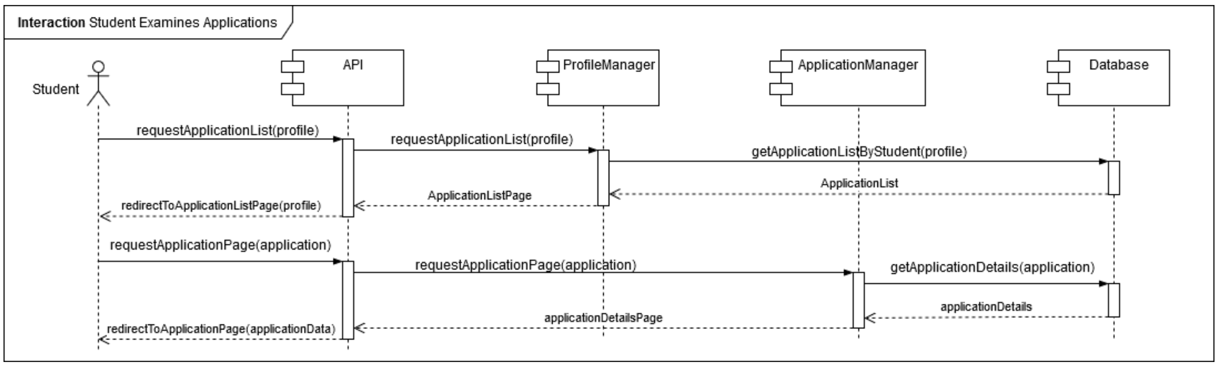
The interaction diagram describes the process of searching for internship positions within S&C. The sequence begins when the user requests access to the internship search page through the **API**. The **API** processes this request and redirects the user to the internship search page. Once on the search page, the user provides search criteria, which may include filters values but also keywords. These data are sent from the **API** to the **SearchManager** to be validated and if they are evaluated correct, the **SearchManager** uses them to query the **Database** for matching internships. If the **Database** returns a list of internships that match the user's criteria, the **SearchManager** runs a ranking algorithm on them to put first the most suited ones to the profile who issued the search, then the **API** redirects the user to the internship results page and displays the available options. If no matches are found, the **API** redirects the user to an empty internship results page, along with a message indicating that no internship meets the specified criteria.

6. **Internship Application**


SD Application for Internship
Student
API
Application Manager
Database
applyToInternship()
getInternshipPage()
InternshipPage
redirectToInternshipPage()
confirmsApplication(profileData, conditionsAccepted)
confirmsApplication(profileData, conditionsAccepted)
validateApplication(conditionsAccepted)
alternative
if validateApplication(conditionsAccepted)
write(profileData)
writeToDb(profileData)
alternative
if writeToDb(profileData):
ok
ProfilePage
redirectToProfilePage()
else
error
InternshipPage
redirectToInternshipPage(message)
else
InternshipPage
redirectToInternshipPage(message)

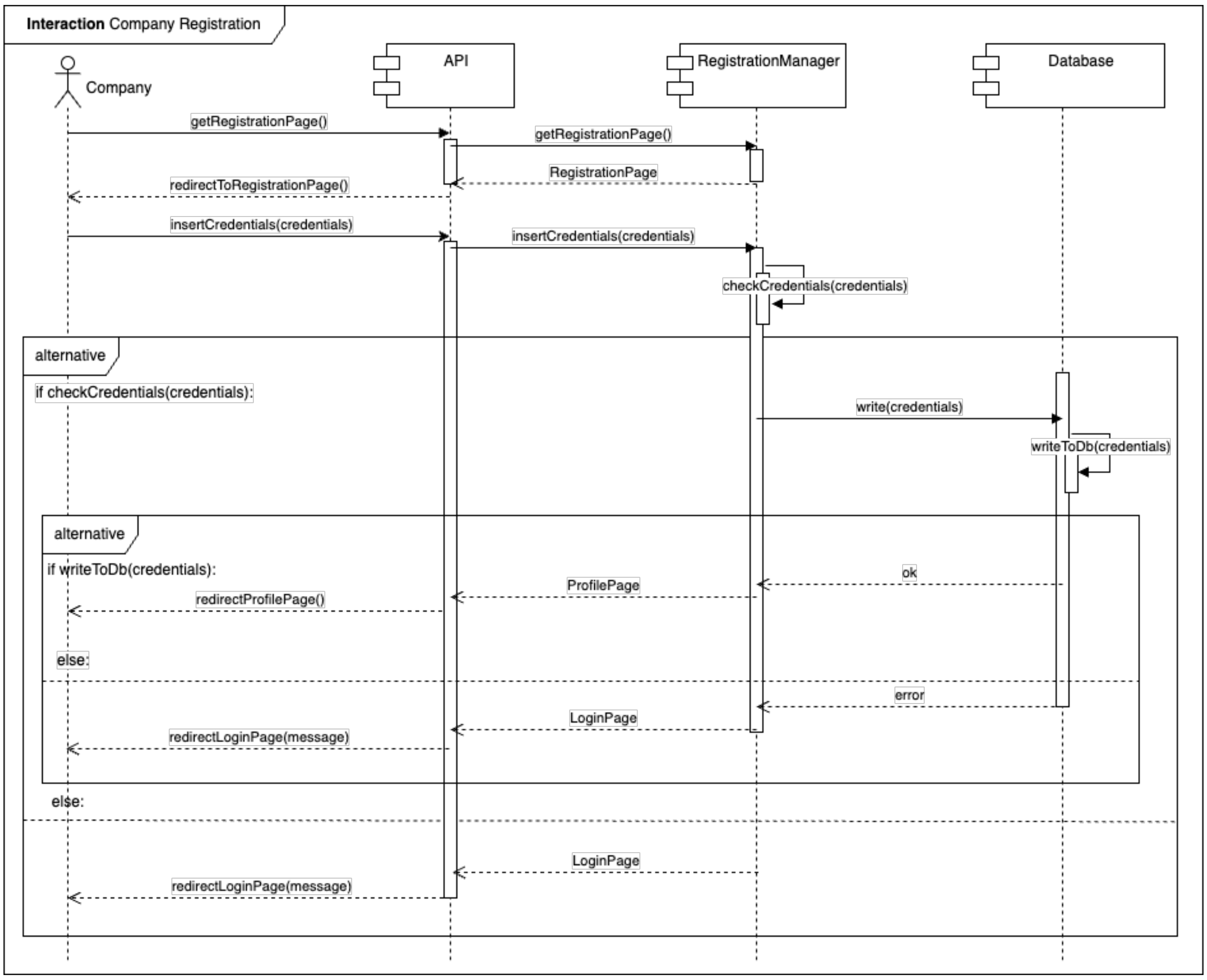
The interaction diagram shows the process of applying for an internship within S&C. The sequence begins when the user accesses the internship application page by interacting with the **API**. The **API** redirects the user to the internship position page where he/she must accept the terms and conditions. Once the user submits their application, the **API** forwards the data to the **ApplicationManager**, which validates the submission. This validation includes ensuring that the user has accepted the terms and conditions of the internship application. If the validation is successful, the **ApplicationManager** writes the relevant profile data to the Database to store the application and then the **API** redirects the user to their profile page to confirm the completion of the application process. If the validation fails the **API** keeps the user into the internship application page showing an error message.

7. **Student Examines Application**


Interaction Student Examines Applications
Student
API
ProfileManager
ApplicationManager
Database
requestApplicationList(profile)
requestApplicationList(profile)
getApplicationListByStudent(profile)
ApplicationList
ApplicationListPage
redirectToApplicationListPage(profile)
requestApplicationPage(application)
requestApplicationPage(application)
getApplicationDetails(application)
applicationDetails
applicationDetailsPage
redirectToApplicationPage(applicationData)

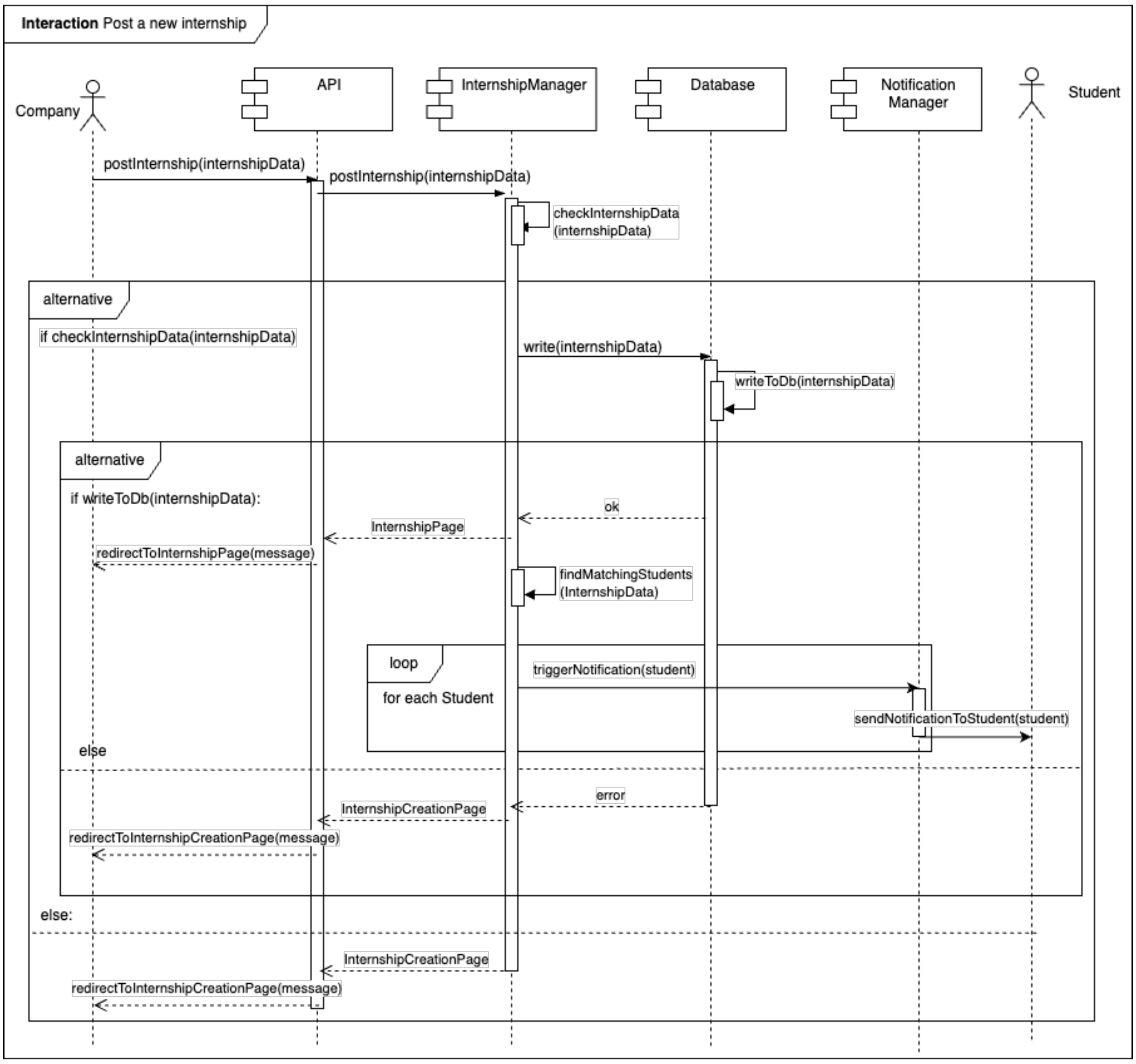
The interaction diagram shows the process of a student examining their internship applications within S&C. The process begins when the user requests to view their application history. This request is handled by the **API**, which redirects the user to the application list page. The **API** communicates with the **ProfileManager** to retrieve the user's profile-related data and then queries the **ApplicationManager** for a list of the student's applications. The **ApplicationManager** consults the **Database** to gather the details of all internship applications associated with the user. Once the information is retrieved, it is sent back to the **API**, which displays the application list to the user on the application list page. If the user selects a specific application to examine in more detail, the **API** processes this request by communicating with the **ApplicationManager**. The **ApplicationManager** retrieves the details of the selected application from the **Database** and sends them back to the **API**. Finally, the **API** redirects the user to a detailed view of the selected application, where they can review the information related to it.

8. **Company Registration**


Interaction Company Registration
Company
API
RegistrationManager
Database
getRegistrationPage()
getRegistrationPage()
RegistrationPage
redirectToRegistrationPage()
insertCredentials(credentials)
insertCredentials(credentials)
checkCredentials(credentials)
alternative
if checkCredentials(credentials):
write(credentials)
writeToDb(credentials)
alternative
if writeToDb(credentials):
ok
ProfilePage
redirectProfilePage()
else:
error
LoginPage
redirectLoginPage(message)
else:
LoginPage
redirectLoginPage(message)

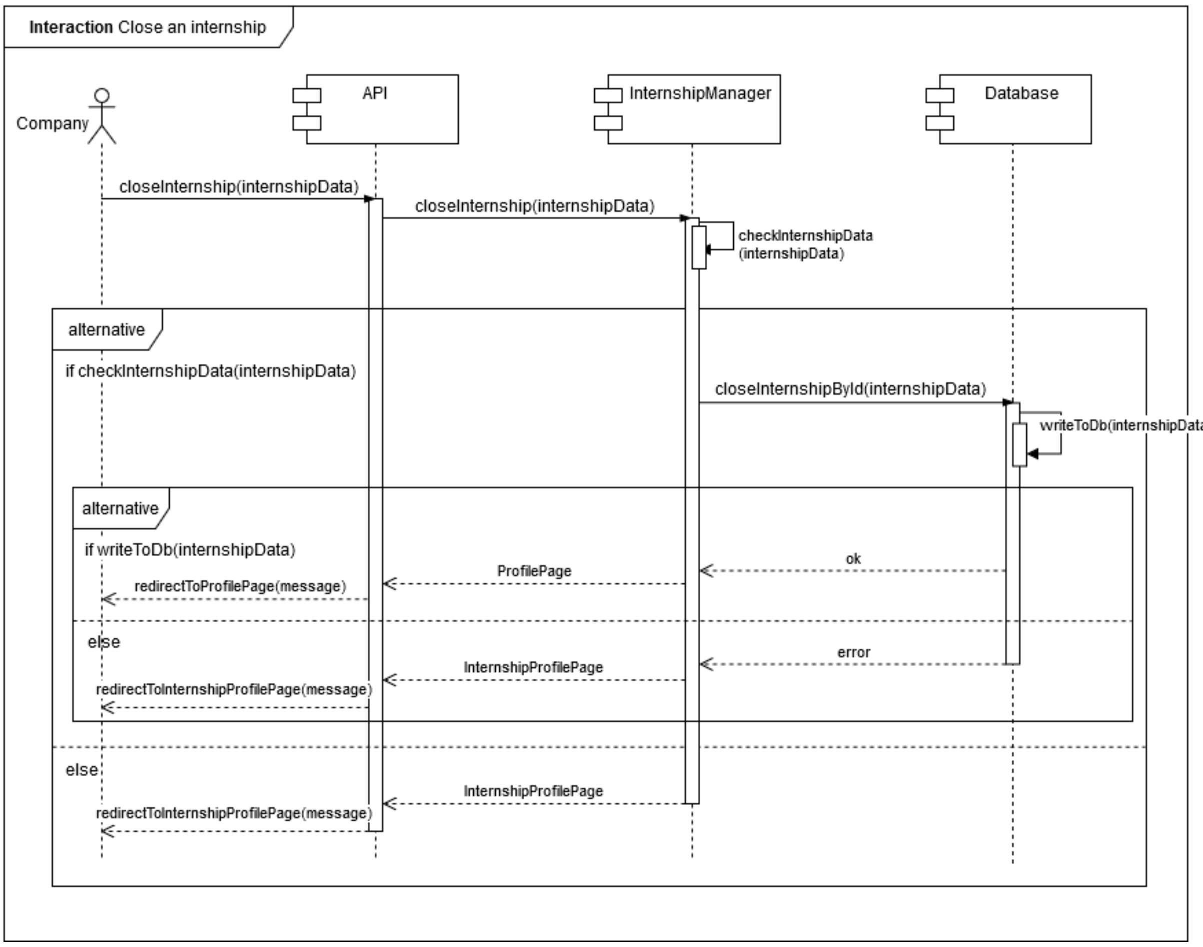
The interaction diagram describes the process of company registration within the S&C. The process begins when a company starting a request to access the registration page. The **API** handles this request and redirects the company to the appropriate page for inputting registration details. Once the company provides its credentials, the **API** forwards the submitted information to the **RegistrationManager** that validates them, checking their correctness and ensuring they meet the necessary requirements. If the validation is successful, the **RegistrationManager** stores the new credentials by writing them to the **Database**. After confirmation of successful storage, the API redirects the company to their newly created profile page. If the credentials fail validation due to errors or missing information, the **RegistrationManager** notifies the API. In this case, the API redirects the company back to the login page with an appropriate error message.

9. **Post a new Internship**


Interaction Post a new internship
Company
API
InternshipManager
Database
Notification Manager
Student
postInternship(internshipData)
postInternship(internshipData)
checkInternshipData (internshipData)
alternative
if checkInternshipData(internshipData)
write(internshipData)
writeToDb(internshipData)
alternative
if writeToDb(internshipData):
ok
InternshipPage
redirectToInternshipPage(message)
findMatchingStudents (InternshipData)
loop
for each Student
triggerNotification(student)
sendNotificationToStudent(student)
else
error
InternshipCreationPage
redirectToInternshipCreationPage(message)
else:
InternshipCreationPage
redirectToInternshipCreationPage(message)

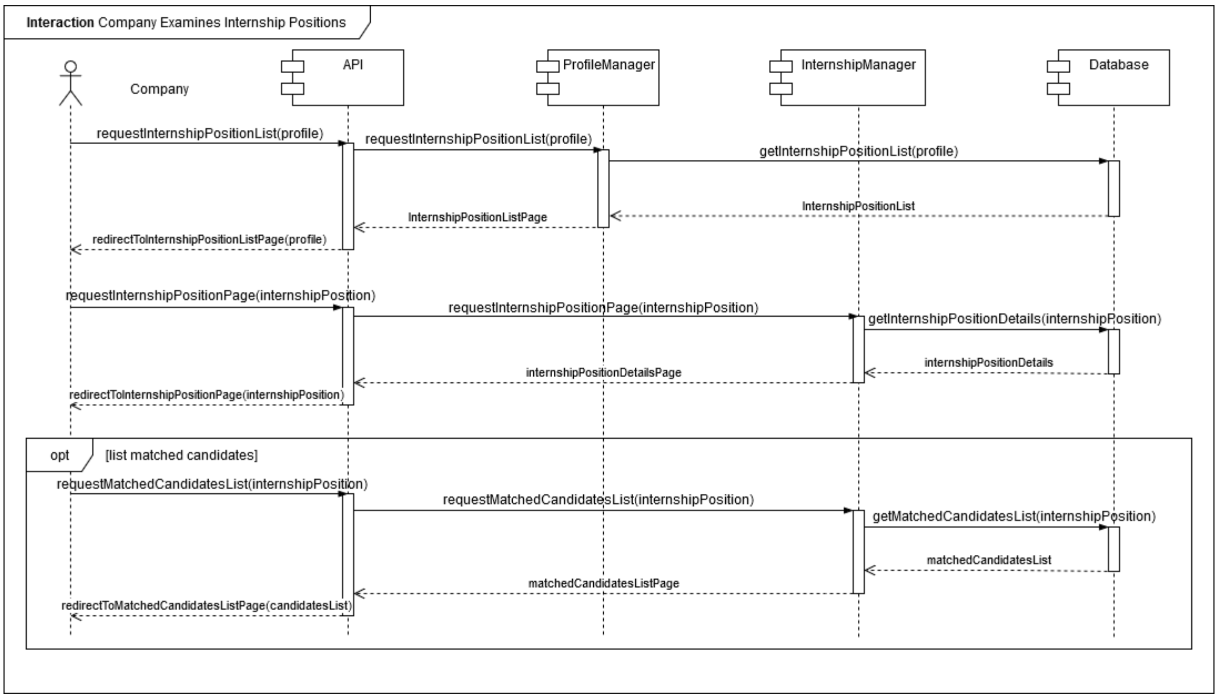
The interaction diagram describes the process of posting a new internship position within S&C. The sequence starts with a company accessing the internship creation page through the **API**. On this page, the user inputs all the relevant details for the internship position, which the API forwards to the **InternshipManager**. The **InternshipManager** is responsible for validating the provided internship data to ensure it meets the required standards and format. If the data passes validation, the **InternshipManager** stores the internship information in the **Database**. After successfully saving the data, the **InternshipManager** proceeds to identify students whose profiles match the specified internship criteria. For each matched student, a notification is triggered by the **InternshipManager** and sent to the student, informing them of the new internship opportunity. In cases where the validation process fails due to incomplete or incorrect data, the **InternshipManager** informs the API of the issue. The API then redirects the user back to the internship creation page, providing a message with details about the error.

10. **Close an Internship**


Interaction Close an internship
Company
API
InternshipManager
Database
closeInternship(internshipData)
closeInternship(internshipData)
checkInternshipData
(internshipData)
alternative
if checkInternshipData(internshipData)
closeInternshipById(internshipData)
writeToDb(internshipData)
alternative
if writeToDb(internshipData)
ok
ProfilePage
redirectToProfilePage(message)
else
error
InternshipProfilePage
redirectToInternshipProfilePage(message)
else
InternshipProfilePage
redirectToInternshipProfilePage(message)

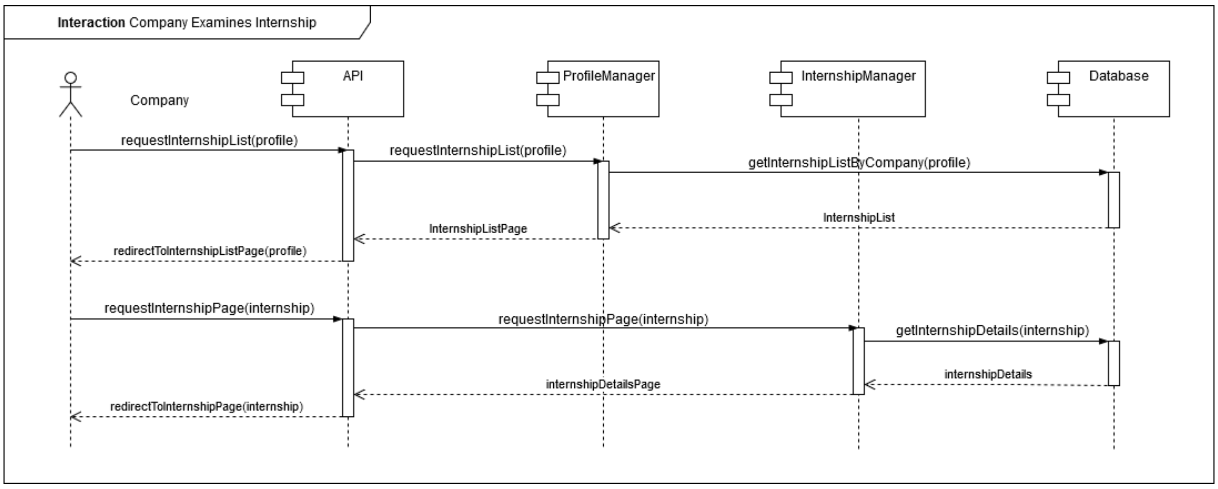
The interaction diagram describes the process of closing an internship position within S&C. The sequence starts with a company pressing "Close" button that inform the **InternshipManager** through the **API** the desire to close a specific internship. The **InternshipManager** is responsible for validating the provided internship data to ensure it meets the required standards, format and the membership of the internship to the specific company. If the data passes validation, the **InternshipManager** changes the internship information in the **Database**. In cases where the validation process fails due to incomplete or incorrect data, the **InternshipManager** informs the **API** of the issue. The **API** then redirects the user back to the internship profile page, providing a message with details about the error, otherwise the company is redirected to its profile page.

11. **Company Examines Internship Positions**


Interaction Company Examines Internship Positions
Company
API
ProfileManager
InternshipManager
Database
requestInternshipPositionList(profile)
requestInternshipPositionList(profile)
getInternshipPositionList(profile)
InternshipPositionList
InternshipPositionListPage
redirectToInternshipPositionListPage(profile)
requestInternshipPositionPage(internshipPosition)
requestInternshipPositionPage(internshipPosition)
getInternshipPositionDetails(internshipPosition)
internshipPositionDetails
internshipPositionDetailsPage
redirectToInternshipPositionPage(internshipPosition)
opt
[list matched candidates]
requestMatchedCandidatesList(internshipPosition)
requestMatchedCandidatesList(internshipPosition)
getMatchedCandidatesList(internshipPosition)
matchedCandidatesList
matchedCandidatesListPage
redirectToMatchedCandidatesListPage(candidatesList)

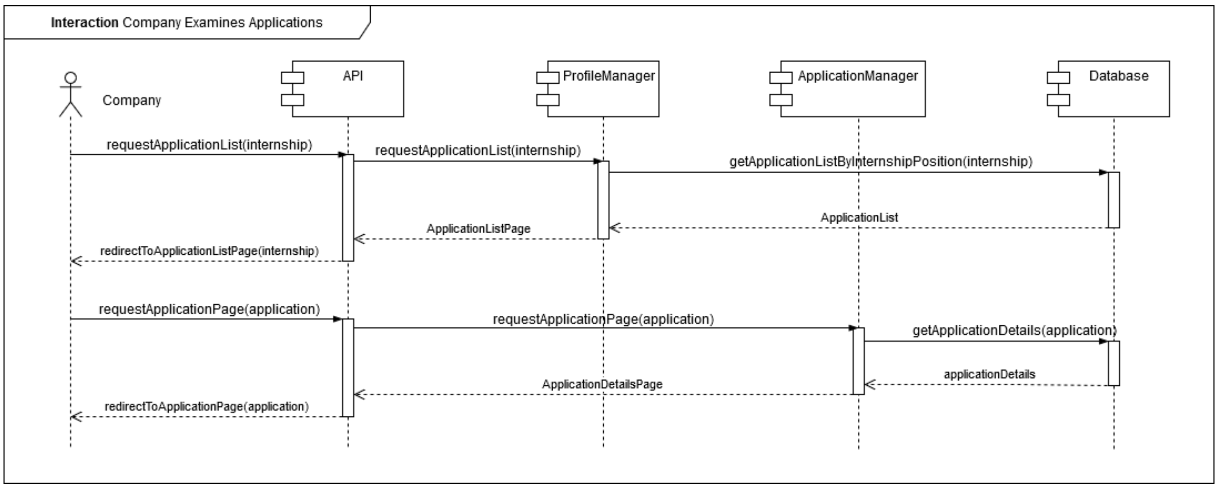
The interaction diagram shows the process of a company examining its internship positions within S&C. The process begins when the user requests to view their internship positions list. This request is handled by the **API**, which redirects the user to the internship positions list page. The **API** communicates with the **ProfileManager** to retrieve the list of the company's internship positions consulting the **Database** to obtain details of all internship positions linked to the user. Once the information is retrieved, it is sent back to the **API**, which displays the internship positions list to the user on the internship positions list page. If the user selects a specific internship position for a detailed examination, the **API** processes this request by communicating with the **InternshipManager**. The **InternshipManager** retrieves the details of the selected internship position from the Database and sends them back to the **API**. Finally, the **API** redirects the user to a detailed view of the selected internship position, where they can review all relevant information. The company has also the option to examine the list of candidates and review their profiles. When requesting the list of candidates for a specific internship, the **API** communicates with the **InternshipManager**, which consults the **Database** to gather the information about each candidate. The data is sent back to the **InternshipManager**, which, through the **API**, redirects the company to the dedicated page displaying the list of matched candidates.

12. **Company Examines Internships**


Interaction Company Examines Internship
Company
API
ProfileManager
InternshipManager
Database
requestInternshipList(profile)
requestInternshipList(profile)
getInternshipListByCompany(profile)
InternshipList
InternshipListPage
redirectToInternshipListPage(profile)
requestInternshipPage(internship)
requestInternshipPage(internship)
getInternshipDetails(internship)
internshipDetails
internshipDetailsPage
redirectToInternshipPage(internship)

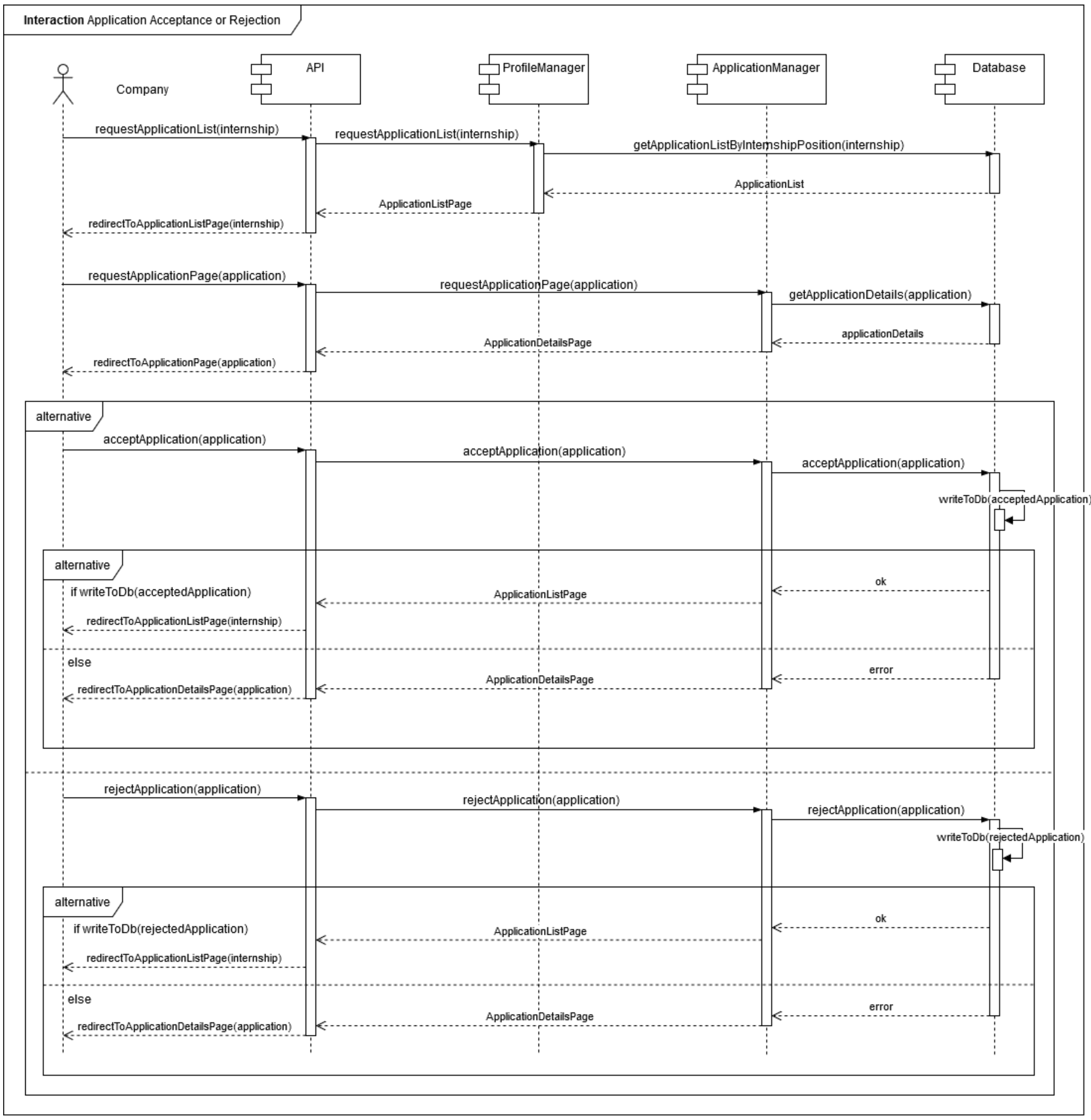
The interaction diagram shows the process of a company examining its internship within S&C. The process begins when the user requests to view their internships history. This request is handled by the **API**, which redirects the user to the internship list page. The **API** communicates with the **ProfileManager** to retrieve the user's profile-related data and then consults the **Database** to gather the details of all internship associated with the user. Once the information is retrieved, it is sent back to the **API**, which displays the internship list to the user on the internship list page. If the user selects a specific internship for a detailed examination, the **API** processes this request by communicating with the **InternshipManager**. The **InternshipManager** retrieves the details of the selected internship from the Database and sends them back to the **API**. Finally, the **API** redirects the user to a detailed view of the selected internship, where they can review all relevant information.

13. **Company Examines Applications**


Interaction Company Examines Applications
Company
API
ProfileManager
ApplicationManager
Database
requestApplicationList(internship)
requestApplicationList(internship)
getApplicationListByInternshipPosition(internship)
ApplicationList
ApplicationListPage
redirectToApplicationListPage(internship)
requestApplicationPage(application)
requestApplicationPage(application)
getApplicationDetails(application)
applicationDetails
ApplicationDetailsPage
redirectToApplicationPage(application)

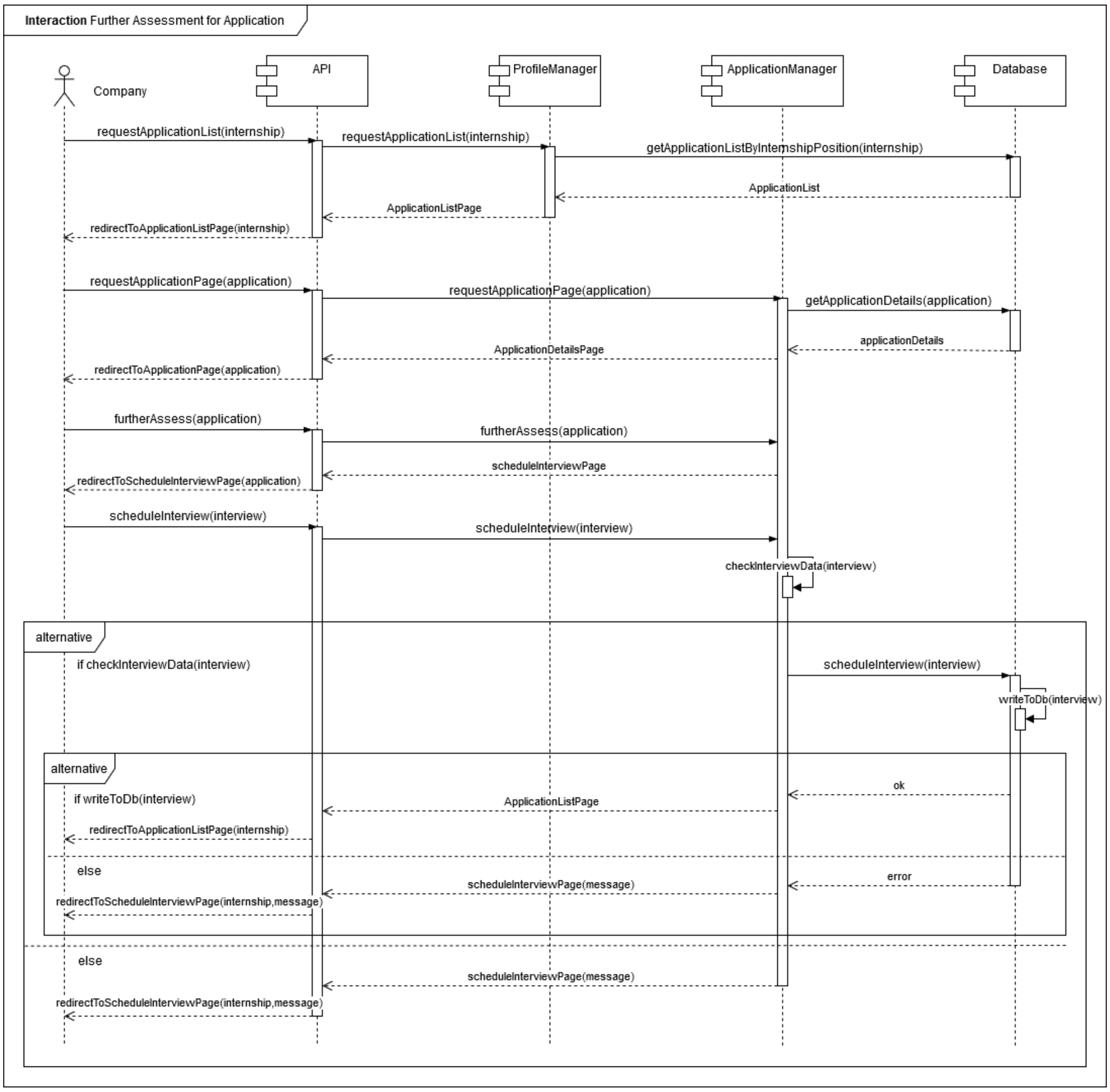
The interaction diagram shows the process of a company examining applications related to a specific internship within S&C. The process begins when the user requests to view their applications history. This request is handled by the **API**, which redirects the user to the application list page. The **API** communicates with the **ProfileManager** to retrieve the internship data and then consults the **Database** to gather details of all applications associated with the internship. Once the information is retrieved, it is sent back to the **API**, which displays the applications list to the user on the applications list page. If the user selects a specific application for a detailed examination, the **API** processes this request by communicating with the **ApplicationManager**. The **ApplicationManager** retrieves the details of the selected application from the **Database** and sends them back to the **API**. Finally, the **API** redirects the user to a detailed view of the selected application, where they can review all relevant information.

14. **Application Acceptance or Rejection**


Interaction Application Acceptance or Rejection
Company
API
ProfileManager
ApplicationManager
Database
requestApplicationList(internship)
requestApplicationList(internship)
getApplicationListByInternshipPosition(internship)
ApplicationList
ApplicationListPage
redirectToApplicationListPage(internship)
requestApplicationPage(application)
requestApplicationPage(application)
getApplicationDetails(application)
applicationDetails
ApplicationDetailsPage
redirectToApplicationPage(application)
alternative
acceptApplication(application)
acceptApplication(application)
acceptApplication(application)
writeToDb(acceptedApplication)
alternative
if writeToDb(acceptedApplication)
ok
ApplicationListPage
redirectToApplicationListPage(internship)
else
error
ApplicationDetailsPage
redirectToApplicationDetailsPage(application)
rejectApplication(application)
rejectApplication(application)
rejectApplication(application)
writeToDb(rejectedApplication)
alternative
if writeToDb(rejectedApplication)
ok
ApplicationListPage
redirectToApplicationListPage(internship)
else
error
ApplicationDetailsPage
redirectToApplicationDetailsPage(application)

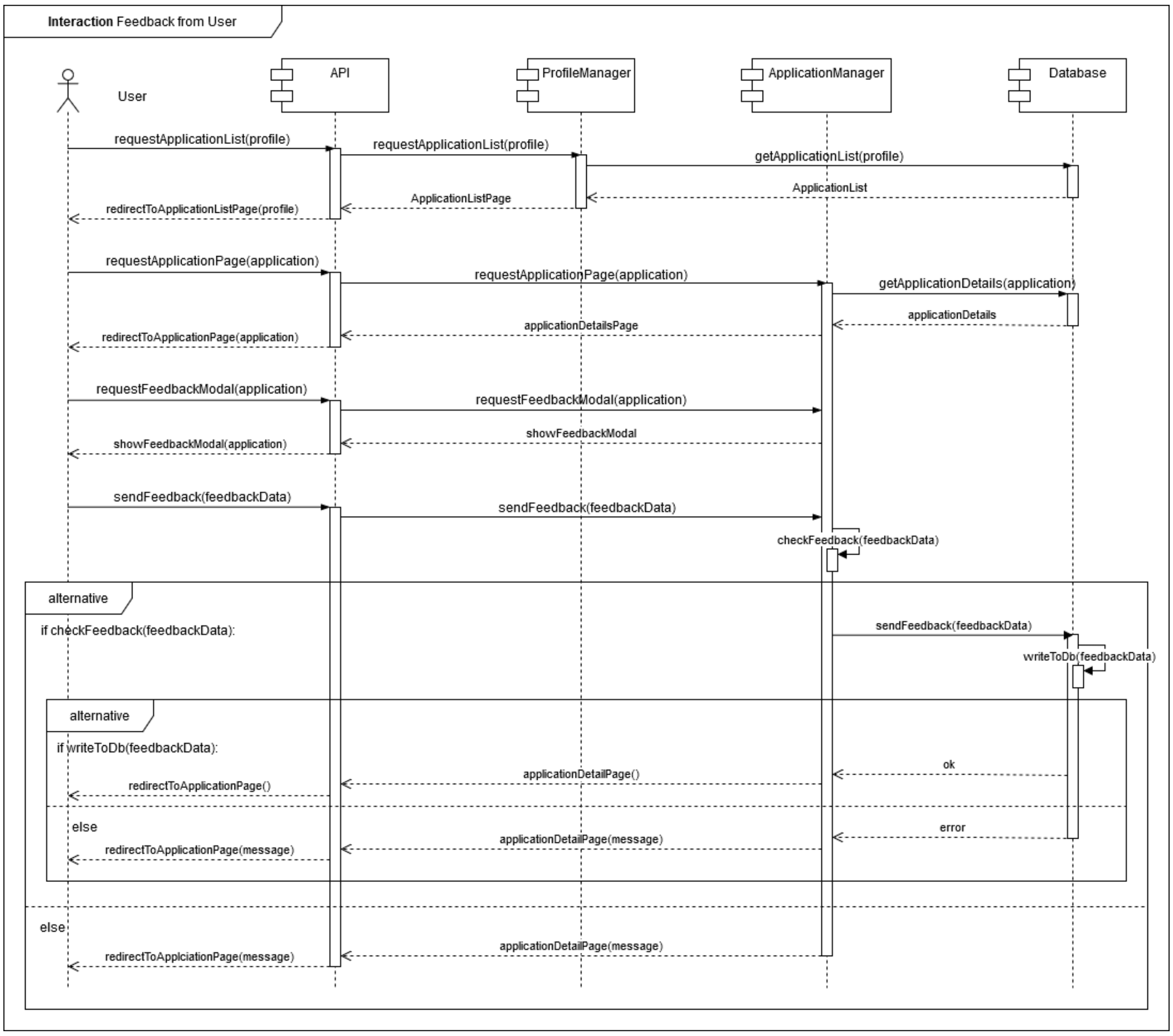
The interaction diagram shows the process of a company examining applications related to a specific internship and deciding to accept or reject a specific application within S&C. When a company requests their application history, the **API** redirects them to the applications list page by retrieving internship data via the **ProfileManager** and querying the **Database** for application details. The **API** displays the list, and if the user selects an application for review, it communicates with the **ApplicationManager** to fetch detailed information from the **Database**. The **API** then presents the detailed application view to the user. At this point the company can choose to accept or reject the application by sending this decision to the **ApplicationManager** through the **API**. The **ApplicationManager** stores this new information into the **Database**. If the operation is successful, the **Database** sends a *success message*, which allows the **ApplicationManager** to redirect the company back to the application list page through the **API**. If there is any error during the write operation in the **Database**, the company is redirected to the application details page to address the issue.

15. **Further Assessment for Application**


Interaction Further Assessment for Application
Company
API
ProfileManager
ApplicationManager
Database
requestApplicationList(internship)
requestApplicationList(internship)
getApplicationListByInternshipPosition(internship)
ApplicationList
ApplicationListPage
redirectToApplicationListPage(internship)
requestApplicationPage(application)
requestApplicationPage(application)
getApplicationDetails(application)
applicationDetails
ApplicationDetailsPage
redirectToApplicationPage(application)
furtherAssess(application)
furtherAssess(application)
scheduleInterviewPage
redirectToScheduleInterviewPage(application)
scheduleInterview(interview)
scheduleInterview(interview)
checkInterviewData(interview)
alternative
if checkInterviewData(interview)
scheduleInterview(interview)
writeToDb(interview)
alternative
if writeToDb(interview)
ok
ApplicationListPage
redirectToApplicationListPage(internship)
else
error
scheduleInterviewPage(message)
redirectToScheduleInterviewPage(internship,message)
else
scheduleInterviewPage(message)
redirectToScheduleInterviewPage(internship,message)

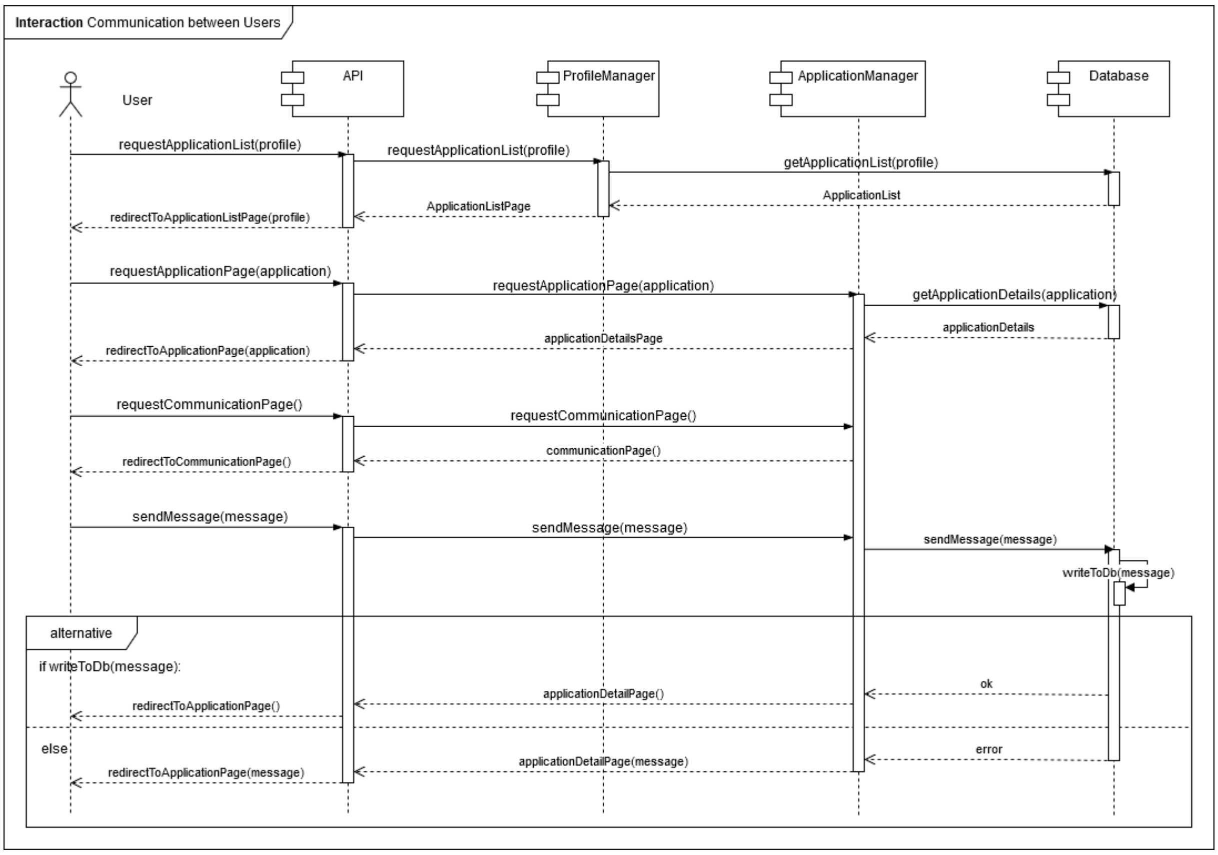
The interaction diagram shows the process of a company examining applications related to a specific internship and deciding to have the candidate further evaluated within S&C. When a company requests their application history, the **API** redirects them to the applications list page by retrieving internship data via the **ProfileManager** and querying the **Database** for application details. The **API** displays the list, and if the user selects an application for review, it communicates with the **ApplicationManager** to fetch detailed information from the **Database**. The **API** then presents the detailed application view to the user. At this point the company can decide to schedule an interview for a specific application by informing the **ApplicationManager** through the **API**, which redirects the user to the page where they can fill in all the required information to schedule the interview. This data is first sent to the **API** and then to the **ApplicationManager** for validation. If the check is successful, this information is stored in the **Database**. If the operation is successful, the **Database** sends a *success message*, which allows the **ApplicationManager** to redirect the company back to the application list page through the **API**. If there is any error during the write operation in the **Database**, the company is redirected to the schedule interview page to address the issue.

16. **Feedback from User**


Interaction Feedback from User
User
API
ProfileManager
ApplicationManager
Database
requestApplicationList(profile)
requestApplicationList(profile)
getApplicationList(profile)
ApplicationList
ApplicationListPage
redirectToApplicationListPage(profile)
requestApplicationPage(application)
requestApplicationPage(application)
getApplicationDetails(application)
applicationDetails
applicationDetailsPage
redirectToApplicationPage(application)
requestFeedbackModal(application)
requestFeedbackModal(application)
showFeedbackModal
showFeedbackModal(application)
sendFeedback(feedbackData)
sendFeedback(feedbackData)
checkFeedback(feedbackData)
alternative
if checkFeedback(feedbackData):
sendFeedback(feedbackData)
writeToDb(feedbackData)
alternative
if writeToDb(feedbackData):
ok
applicationDetailPage()
redirectToApplicationPage()
else
error
applicationDetailPage(message)
redirectToApplicationPage(message)
else
applicationDetailPage(message)
redirectToApplciationPage(message)

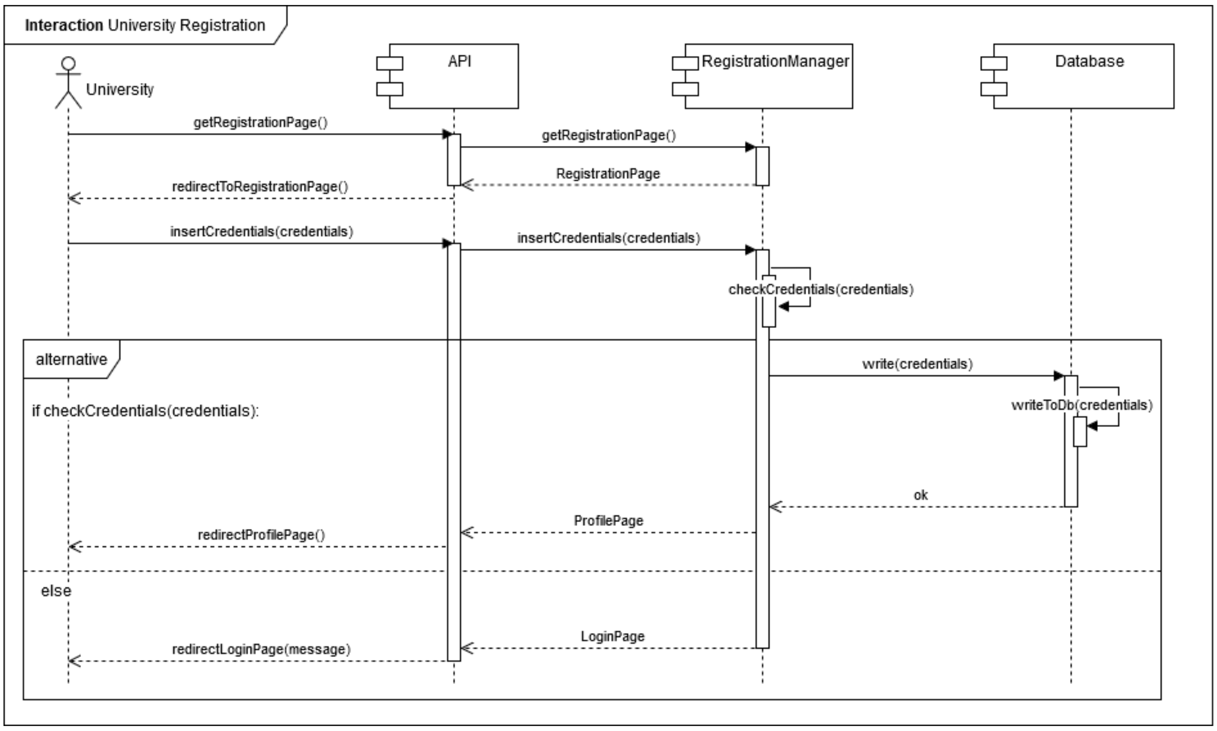
The interaction diagram shows the process of a user examining applications related to a specific internship and deciding to send a feedback within S&C. When a company requests their application history, the **API** redirects them to the applications list page by retrieving internship data via the **ProfileManager** and querying the **Database** for application details. The **API** displays the list, and if the user selects an application for review, it communicates with the **ApplicationManager** to fetch detailed information from the **Database**. The **API** then presents the detailed application view to the user.
At this point the user can decide to create a feedback for a specific application by informing the **ApplicationManager** through the **API**, which redirects the user to the page where they can fill in all the required information. This data is first sent to the **API** and then to the **ApplicationManager** for validation. If the check is successful, this information is stored in the **Database**. If the operation is successful, the **Database** sends a *success message*, which allows the **ApplicationManager** to redirect the user back to the application page through the **API**. If there is any error during the write operation in the **Database**, the user is redirected to the application page with a message to address the issue.

17. **Communication between Users**


Interaction Communication between Users
User
API
ProfileManager
ApplicationManager
Database
requestApplicationList(profile)
requestApplicationList(profile)
getApplicationList(profile)
ApplicationList
ApplicationListPage
redirectToApplicationListPage(profile)
requestApplicationPage(application)
requestApplicationPage(application)
getApplicationDetails(application)
applicationDetails
applicationDetailsPage
redirectToApplicationPage(application)
requestCommunicationPage()
requestCommunicationPage()
communicationPage()
redirectToCommunicationPage()
sendMessage(message)
sendMessage(message)
sendMessage(message)
writeToDb(message)
alternative
if writeToDb(message):
ok
applicationDetailPage()
redirectToApplicationPage()
else
error
applicationDetailPage(message)
redirectToApplicationPage(message)

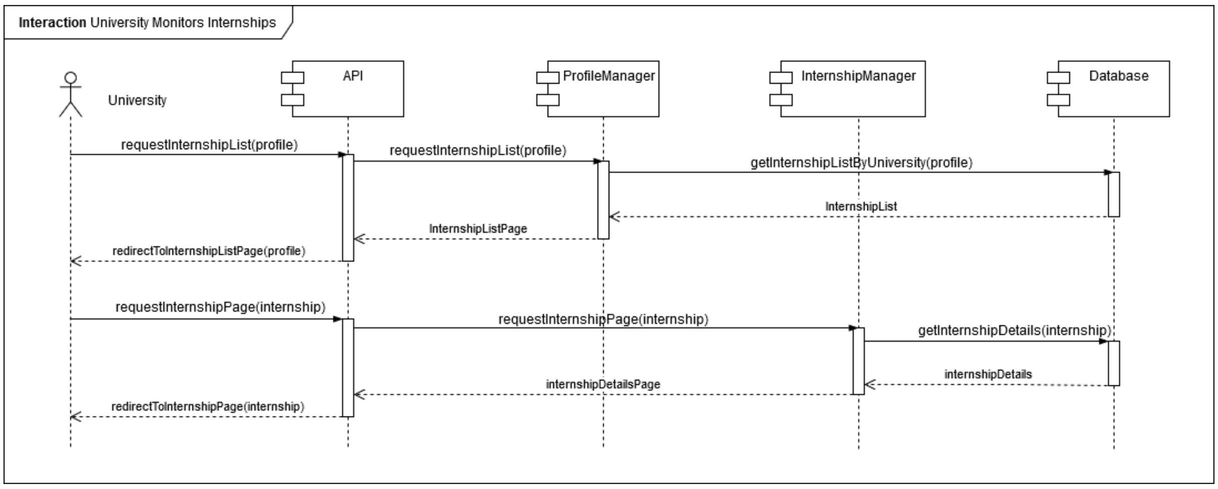
The interaction diagram shows the process of a user examining applications related to a specific internship and deciding to send a message within S&C. When a company requests their application history, the **API** redirects them to the applications list page by retrieving internship data via the **ProfileManager** and querying the **Database** for application details. The **API** displays the list, and if the user selects an application for review, it communicates with the **ApplicationManager** to fetch detailed information from the **Database**. The **API** then presents the detailed application view to the user.
At this point the user can decide to create a message for a specific application by informing the **ApplicationManager** through the **API**, which redirects the user to the page where they can write the message. This data is first sent to the **API**, then to the **ApplicationManager** and finally is stored in the **Database**. If the operation is successful, the **Database** sends a *success message*, which allows the **ApplicationManager** to redirect the user back to the application page through the **API**. If there is any error during the write operation in the **Database**, the user is redirected to the application page with a message to address the issue.

18. **University Registration**


Interaction University Registration
University
API
RegistrationManager
Database
getRegistrationPage()
getRegistrationPage()
RegistrationPage
redirectToRegistrationPage()
insertCredentials(credentials)
insertCredentials(credentials)
checkCredentials(credentials)
alternative
if checkCredentials(credentials):
write(credentials)
writeToDb(credentials)
ok
ProfilePage
redirectProfilePage()
else
LoginPage
redirectLoginPage(message)

The interaction diagram describes the process of university registration within the S&C. The process begins when a university initiates a request to access the registration page. The **API** handles this request and redirects the university to the appropriate page for inputting registration details. Once the university provides its credentials, the **API** forwards the submitted information to the **RegistrationManager** that validates them, checking their correctness and ensuring they meet the necessary requirements. If the validation is successful, the **RegistrationManager** stores the new credentials by writing them to the **Database**. After confirmation of successful storage, the **API** redirects the university to their newly created profile page. If the credentials fail validation due to errors or missing information, the **RegistrationManager** notifies the **API**. In this case, the **API** redirects the university back to the login page with an appropriate error message.

19. **University Monitors Internships**


Interaction University Monitors Internships
University
API
ProfileManager
InternshipManager
Database
requestInternshipList(profile)
requestInternshipList(profile)
getInternshipListByUniversity(profile)
InternshipList
InternshipListPage
redirectToInternshipListPage(profile)
requestInternshipPage(internship)
requestInternshipPage(internship)
getInternshipDetails(internship)
internshipDetails
internshipDetailsPage
redirectToInternshipPage(internship)

The interaction diagram shows the process of a university examining internships of its students within S&C. The process begins when the user requests to view their internships history. This request is handled by the **API**, which redirects the user to the internship list page. The **API** communicates with the **ProfileManager** to retrieve the user's profile-related data and then consults the **Database** to gather the details of all internship associated with the user. Once the information is retrieved, it is sent back to the **API**, which displays the internship list to the user on the internship list page. If the user selects a specific internship for a detailed examination, the **API** processes this request by communicating with the **InternshipManager**. The **InternshipManager** retrieves the details of the selected internship from the Database and sends them back to the **API**. Finally, the **API** redirects the user to a detailed view of the selected internship, where they can review all relevant information.

2.4 Component Interfaces

1. **API**:

    - *Response* getLoginPage()
    - *Response* insertCredentials(*Dict* credentials)
    - *Response* LogOut()
    - *Response* getUpdatePage(*Dict* profileData)
    - *Response* modifyInfo(*Dict* profileData, *Dict* modifications)
    - *Response* getRegistrationPage()
    - *Response* insertRegistrationCredentials(*Dict* profileData)
    - *Response* requestSearchPage()
    - *Response* searchInternship(*Dict* filtersData, *Dict* relevantProfileInfo)
    - *Response* getInternshipPage()
    - *Response* confirmsApplication(*Dict* profileData, *Bool* conditionsAccepted)
    - *Response* requestApplication(*Dict* profileData)
    - *Response* requestApplicationPage(*Int* applicationId)
    - *Response* postInternship(*Dict* internshipData)
    - *Response* closeInternship(*Int* internshipId)
    - *Response* requestInternshipPositionList(*Dict* profileData)
    - *Response* requestInternshipPositionPage(*Int* internshipPositionId)
    - *Response* requestMatchedCandidatesList(*Int* internshipPositionId)
    - *Response* requestInternshipList(*Dict* profileData)
    - *Response* requestInternshipPage(*Int* internshipId)
    - *Response* requestApplicationList(*Int* internshipPositionId)
    - *Response* acceptApplication(*Int* applicationId)
    - *Response* rejectApplication(*Int* applicationId)
    - *Response* furtherAssess(*Int* applicationId)
    - *Response* scheduleInterview(*Dict* interviewData)
    - *Response* requestFeedbackModal(*Int* applicationData
    - *Response* sendFeedback(*Dict* feedbackData)
    - *Response* requestCommunicationPage()
    - *Response* sendMessage(*String* message)

2. **LoginManager**

    - *Dict* getCredentials(*Dict* credentials)
    - *Bool* checkCredentials(*Dict* doubleCredentials)

3. **RegistrationManager**

    - *Bool* checkCredentials(*Dict* credentials)
    - *Bool* write(*Dict* credentials)
    - List[*Int*] findMatchingCompanies(*Dict* credentials)

4. **SearchManager**
   - *Bool* checkfiltersData(*Dict* credentials)
   - *List*[*Dict*] getInternship(*Dict* filtersData)
   - *List*[*Dict*] rankingInternship(*Dict* relevantProfileInfo, *List[Dict]* internshipList)
5. **NotificationManager**
   - *Void* sendNotificationToCompany(*int* companyId);
   - *Void* sendNotificationToStudent(*Int* studentId);
6. **ProfileManager**
   - *Dict* getProfileData(*Int* userID);
   - *Bool* checkModifications(*Dict* modifications);
   - *Void* updateUserProfile(*Dict* modifications);
   - *List[Dict]* getApplicationListByStudent(*Int* studentId);
   - *List[Dict]* getApplicationListByInternshipPosition(*Int* internshipPosition)
   - *List[Dict]* getInternshipPositionList(*Int* companyId);
   - *List[Dict]* getInternshipListByCompany(*Int* companyId);
   - *List[Dict]* getInternshipListByUniversity(*Int* universityId);
7. **ApplicationManager**
   - *Bool* validateApplication(*Bool* conditionsAccepted);
   - *Void* writeNewApplication(*Int* studentId, *Int* internshipPositionId);
   - *Dict* getApplicationDetails(*Int* applicationId);
   - *Void* acceptApplication(*Int* applicationId);
   - *Void* rejectApplication(*Int* applicationId);
   - *Bool* checkInterviewData(*Dict* interview);
   - *Bool* checkFeedback(*Dict* feedback);
   - *Void* sendFeedback(*Dict* feedback);
   - *Void* sendMessage(*Dict* message);
8. **InternshipManager**
   - *Bool* checkInternshipData(*Dict* internshipData);
   - *Void* writeNewInternship(*Dict* internshipData);
   - *List[Int]* findMarchingStudents(*Dict* internshipData);
   - *Void* triggerNotification(*Int* studentId);
   - *Void* closeInternshipById(*Int* internshipId);
   - *Dict* getInternshipPositionDetails(*Int* internshipPositionId);
   - *Dict* getInternshipDetails(*Int* internshipId);
   - *List[Dict]* getMarchedCandidates(*Dict* internshipPosition);

2.5 Selected Architectural Styles and Patterns

2.5.1 3-Tier Architecture

As already mentioned in the overview above, the decision to go for the 3-tier architecture is based on the belief that this type of architecture is well suited to the kind of application that it is going to be developed. The first immediate benefit comes from the modularization which is the key feature of this type of architecture and from which indeed its name derives. The 3 independent layers are:

- **Presentation layer**: this corresponds to the front-end which is composed by the web pages and web interfaces with which the users interact.
- **Application layer**: this corresponds to the back-end which is where the logic of the application resides.
- **Data layer**: this corresponds to the database which is the place where data are stored permanently.

The choice of implementing a three-tier architecture has several advantages.

- The various tiers can be developed simultaneously and independently by different people or teams of people making the development phase faster
- The division in three tiers makes the separation of different functions more clear and makes the code easier to be understood and maintained
- The separation in three allows each layer to be scaled independently from the others, thus enhancing the overall scalability of the web application

To facilitate the communication between the presentation and the application layers we decided to adopt the use of REST APIs. The use of REST APIs is, in general, a choice towards flexibility, indeed they bring many advantages such as:

- **Decoupling**: it allows the developers to separate the scope and the concerns of the modules in three different layers, making the maintenance of the web application and the deployment of new features easier.
- **Easier future integration with third party services**: if for any reason in the future it will be necessary to add third party services they could be effortlessly integrated into the system.
- **Standard formats of communication**: REST APIs use standard HTTP methods and the JSON format to communicate which are the common standard for the world wide web.
- **Scalability**: each tier can be scaled independently from the others (up to a certain point of course).
- **Caching Support**: if needed in the future caches can be added to speed up the application since REST API integrate seamlessly with caches.

Since our app it is not huge, a monolithic structure will suffice. Indeed microservices are suited to those kinds of applications who suffer of scalability issues. S&C requirements are far less constraining than the ones of big tech companies; therefore we believe that the development of a microservice based application that can request a lot of expertise and ability, would not be a good design choice.

2.6 Other Design Decisions

2.6.1 Standard Compliance

S&C should be GDPR compliant, therefore when deployed into production must use cloud providers that keep the European citizen's data in the EU. Moreover, it will use encryption libraries (such as Flask BCrypt) to securely store sensitive information.

2.6.2 Security

The code infrastructure will use a secure-by-design approach, creating secure code and test it really extensively in order to discover bugs and vulnerabilities before the deployment. Some libraries that are going to be used towards the goal of making the web application secure are Flask Cors, JWT, BCrypt on the back-end and custom validation code on the front-end. More information on the testing plan will be discussed in section 5 of this document.

2.6.3 Reliability

S&C must be reliable and therefore enforce data consistency.For this reason we thought that a SQL database is a good choice. For prototyping SQLite is definitely the best choice, when going to production solid choices are PostgreSQL and MySQL.

2.6.4 Availability

S&C must be available as much as possible. Therefore, it is important that the code written allows the server to be up and running smoothly.

2.6.5 Maintainability

S&C code base should be easily maintainable. In order to ensure that, first of all, no legacy code will be allowed. Legacy code slow the web app down: all the lines of code in the project must have a purpose. Modular programming is strongly recommended. Modules enhance readability of the code, allow for decoupling of components and, in general, help when in case of debugging. GitHub will be used as a versioning control software to keep track of the changes of the code and their authors. Comments are also strongly recommended: commenting functions, APIs and more generally anything that it is not immediately clear is a good programming practice.

3 User Interface Design

The followings are some snippets of the user interface design of S&C web pages.

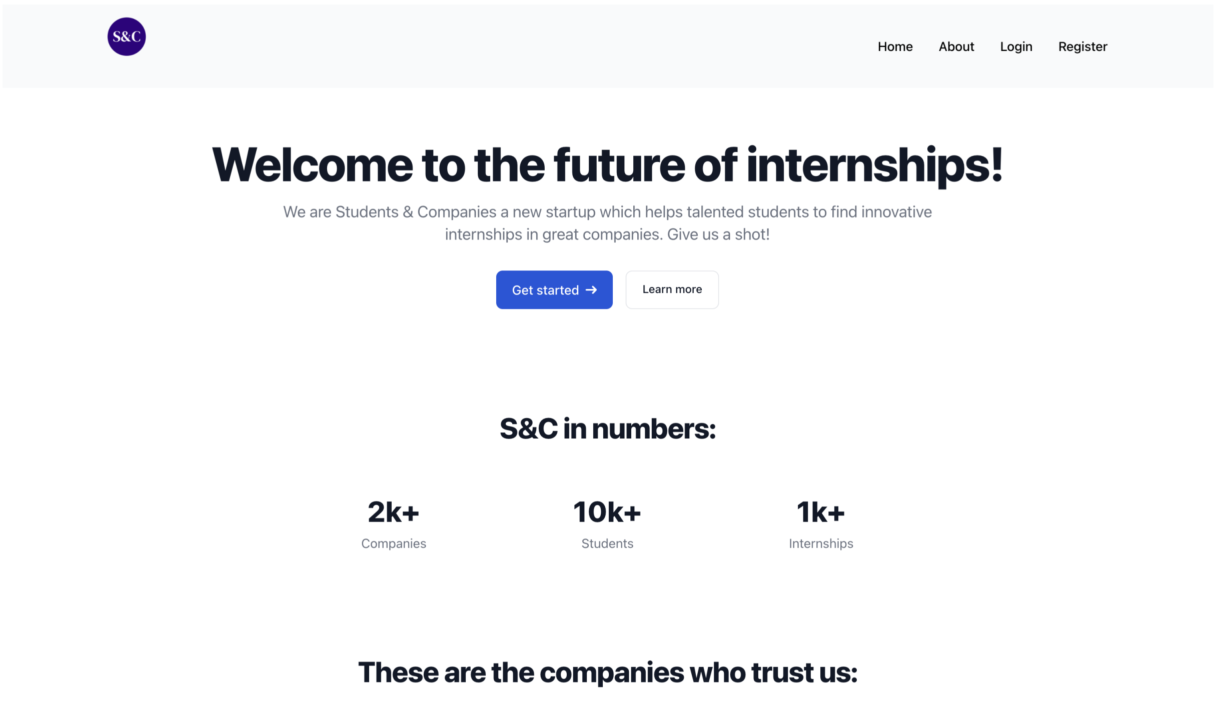

S&C
Home
About
Login
Register
Welcome to the future of internships!
We are Students & Companies a new startup which helps talented students to find innovative internships in great companies. Give us a shot!
Get started →
Learn more
S&C in numbers:
2k+
Companies
10k+
Students
1k+
Internships
These are the companies who trust us:

Figure 1: This is the UI of the Home Page of S&C

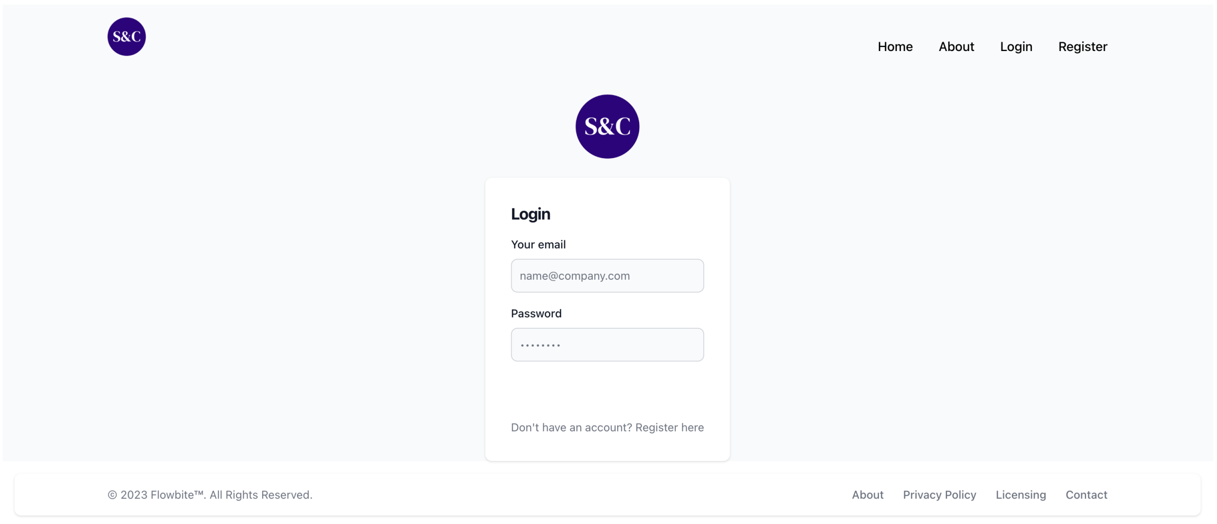
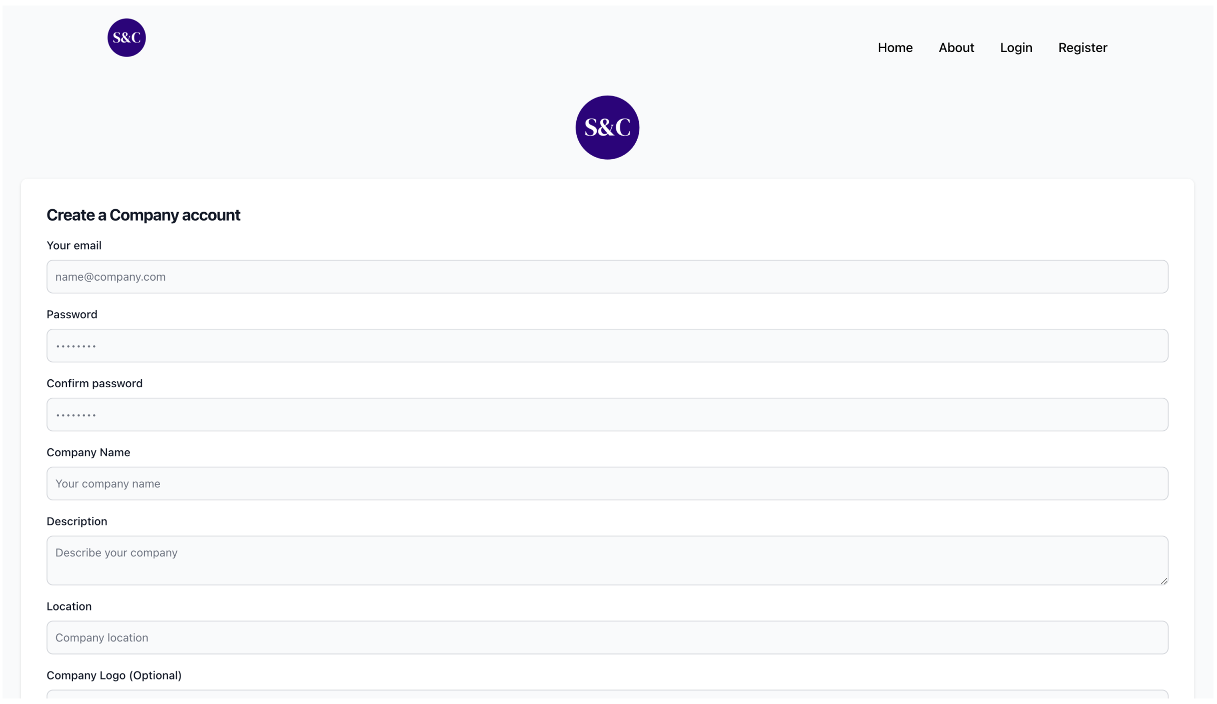


Figure 2: This is the UI of the Login Page of S&C



Create a Company account

Your email

name@company.com

Password

••••••••

Confirm password

••••••••

Company Name

Your company name

Description

Describe your company

Location

Company location

Company Logo (Optional)

Figure 3: This is the UI of the Registration Page of S&C

4 Requirements Traceability

In this section there are tables that show the mapping between the requirements and the modules of S&C.

1. Tab relative to **user registration**

Requirements	[R1] S&C shall allow users to sign up. [R2] S&C shall allow users to fill in their profile information when signing up.
Components	WebApp API RegistrationManager Database

2. Tab relative to user **login** and **logout**

Requirements	[R3] S&C shall allow users to log in. [R4] S&C shall allow users to log out.
Components	WebApp API LoginManager Database

3. Tab relative to **profile examination** by users

Requirements	[R5] S&C shall allow users to update their profile information. [R6] S&C shall allow users to examine their own internships. [R8] S&C shall allow students to examine their own applications.
Components	WebApp API ProfileManager Database

4. Tab relative to **internship positions search**

Requirements	[R7] S&C shall allow students to examine open internship positions [R9] S&C shall allow students to search for a specific kind of internship position [R12] When a student searches for an internship, S&C shall list the different positions aligned with the student's profile.
Components	WebApp API SearchManager Database

5. Tab relative to **applications** to internship positions

Requirements	[R10] S&C shall allow students to apply for an internship position. [R13] If an application has been accepted by a company, S&C shall allow the student who made the application to confirm or refuse the internship. [R14] If a company signals that it needs an assessment of the students' skills, S&C shall allow the company to schedule an interview. [R15] After an interview has been scheduled, S&C shall allow the student to access the link and see the date of the meeting. [R16] S&C shall allow the students to see the status of their applications. [R18] S&C shall allow companies to accept or reject applications for their internship positions
Components	WebApp API ApplicationManager ProfileManager Database

6. Tab relative **notifications**

Requirements	[R11] If an internship position suitable for a student is opened, S&C shall notify the student. [R20] If a new profile of a student who could interest a company becomes available on the platform, S&C shall notify the company.
Components	WebApp API NotificationManager RegistrationManager InternshipManager Database

7. Tab relative to **internship position** and **internship management**

Requirements	[R17] S&C shall allow companies to open and examine its internship positions. [R19] S&C shall allow companies to close internship positions. [R21] S&C shall allow companies to leave private notes to students who are doing an internship with it. [R22] S&C shall allow companies to send news about the internship to the students who participate in it. [R23] S&C shall allow both parties to communicate problems in a specific space of the website. [R24] S&C shall allow students who took an internship with a company to rate the internship and vice versa. [R25] S&C shall allow students who took an internship with a company to give suggestions to the company and vice versa. [R26] S&C shall allow companies to send news about the internship to students who are carrying it out. [R27] S&C shall allow both parties involved in an internship to make complaints in a dedicated space. [R28] S&C shall allow universities to monitor the status of the internship of their students. [R29] S&C shall allow universities to see details of the internships of their students.
Components	WebApp API ProfileManager InternshipManager Database

5 Implementation, Integration and Test Plan Traceability

5.1 Overview

This chapter explains how the platform will be implemented and tested. The primary goal of testing is to identify and resolve as many bugs as possible that may have been introduced by the development team (Section 5.5). In addiction, this section outlines the key implementation strategies used to develop the project (Section 5.4), provides an overview of its most significant features (Section 5.2), and maps these features to the components described earlier (Section 5.3).

5.2 Features identification

The features to implement are delivered according to the project requirements. Some requirements require the creation of new components, while others require only minor modifications to existing functionalities. Below is a summary of the most important features, grouped into a collection of functionalities that address a specific requirement or a set of closely related requirements. This structure ensures a clear understanding of the system's scope and priorities.

1. **User Authentication and Session Management**:
   - **Sign-In, Sign-Up, and Logout**
     Implement sign-in, sign-up, and logout functionalities for all user types: companies, students, and universities. These core features ensure secure access to the platform and a seamless user experience.
2. **Profile Management**:
   - **Modification of User Profile**
     Allow companies, students, and universities to edit their profiles, updating key information like contact details, descriptions, or preferences.
3. **Internship Life-cycle Management**:
   - **Creation of an Internship Position**
     Enable companies to post internship positions, specifying details such as title, description, duration, and requirements.
   - **Creation of an Internship**
     Transform an application to an internship position into an actual internship after the company accepts it and the student confirms. This step marks the beginning of the internship life-cycle.
   - **Closing of an Internship Position**
     Provide companies with the ability to close internship positions once filled or no longer needed.
   - **Closing of an Internship**
     Allow companies to close internships after completion, recording the result for reporting and feedback purposes.
4. **Application Management**:
   - **Creation of an Application for an Internship Position**
     Students can apply to internship positions, submitting necessary documents and information through the platform.
   - **Acceptance, Rejection, or Further Assessment of Applications**
     Companies can evaluate applications, marking them as accepted, rejected, or requiring further assessment.

- **Acceptance or refusal of an Internship by a Student**
  Students must confirm or decline internships after their application is accepted by a company.

5. **Search and Viewing Functionalities**

   - **Search for Internship Positions**
     Students can search for internship positions using filters like location, duration, company, and skills required. The search is enhanced by a recommendation algorithm.
   - **View Applications**
     Companies can view applications for their internship positions to access student suitability.
   - **View Internship Positions**
     Students can view detailed information about available internship positions, including requirements and company details.
   - **View Internships**
     Enable students, companies, and universities to view details about active or completed internships, facilitating transparency and monitoring.

6. **Communication and Feedback**

   - **Communication During an Internship**
     Allow students and companies to exchange messages during an internship to discuss progress, challenges, and updates.
   - **Feedback Between Students and Companies**
     After an internship ends, both the student and the company can provide feedback in the form of ratings and comments.

7. **University Monitoring**

   - **Monitoring of Student Internships by Universities**
     Universities can track the progress of internships involving their students, ensuring academic and professional alignment.

5.3 Features Mapping

This section connects the features identified in the system with the components responsible for their implementation. Each feature is mapped to one or more components from the component diagram, indicating their roles in ensuring the functionality of the system.

1. **Sign-In, Sign-Up, and Logout**

   - ***LoginManager***: Manages authentication processes for signing in and logging out.
   - ***RegistrationManager***: Handles new user registration and validation.
   - ***NotificationManager***: Notifies companies with internship positions that match a student's profile after the student registers.
   - ***Database***: Stores and retrieves user profile information.
   - ***API***: Interfaces with the WebApp to process requests related to authentication and registration.

2. **Modification of User Profile**

   - ***ProfileManager***: Handles profile data retrieval and updates.
   - ***Database***: Stores and retrieves user profile information.

- ***API***: Acts as the communication layer between the WebApp and the ProfileManager.

3. **Creation of an Internship Position**
   - ***InternshipManager***: Responsible for creating and managing internship positions.
   - ***NotificationManager***: Notifies students that match with the new internship position.
   - ***Database***: Stores and retrieves internship position details.
   - ***API***: Facilitates interaction between the company and the InternshipManager.
4. **Creation of an Internship**
   - ***InternshipManager***: Converts an internship position into an internship when a student is accepted.
   - ***ApplicationManager***: Tracks and manages the applications related to an internship.
   - ***Database***: Maintains records of internships and applications.
   - ***API***: Acts as the communication layer between the WebApp and the InternshipManager.
5. **Closing of an Internship Position**
   - ***InternshipManager***: Handles closing internship positions.
   - ***Database***: Updates the status of the internship position.
   - ***API***: Facilitates requests from the WebApp to close internship positions.
6. **Closing of an Internship**
   - ***InternshipManager***: Manages the termination process for internships.
   - ***Database***: Updates the internship status.
   - ***API***: Acts as the communication layer between the WebApp and the InternshipManager.
7. **Creation of an Application for an Internship Position**
   - ***ApplicationManager***: Manages application submissions.
   - ***Database***: Records application data.
   - ***API***: Handles communication between the student interface and the ApplicationManager.
8. **Acceptance, Rejection, or Further Assessment of Applications**
   - ***ApplicationManager***: Processes the application status updates.
   - ***Database***: Updates the application status.
   - ***API***: Interacts with the WebApp for user actions.
9. **Acceptance or rejection of an Internship by a Student**
   - ***InternshipManager***: Updates internship status based on student decisions.
   - ***Database***: Records the final status of the internship.
   - ***API***: Interacts with the WebApp for user actions.
10. **Search for Internship Positions**
    - ***SearchManager***: Handles search queries and applies filters.
    - ***Database***: Provides data for search results.

- ***API***: Processes search requests from the WebApp.

11. **View Applications**

    - ***ApplicationManager***: Retrieves application details.
    - ***Database***: Stores application data.
    - ***API***: Displays application information to the company.

12. **View Internship Positions**

    - ***InternshipManager***: Provides data about available internship positions.
    - ***Database***: Stores internship position details.
    - ***API***: Facilitates the interaction.

13. **View Internships**

    - ***InternshipManager***: Provides internship details to the user.
    - ***Database***: Stores internship records.
    - ***API***: Handles request from the WebApp.

14. **Communication During an Internship**

    - ***ProfileManager***: Sends and tracks communication.
    - ***Database***: Logs messages or communication details.
    - ***API***: Provides the interface for communication.

15. **Feedback Between Students and Companies**

    - ***InternshipManager***: Tracks feedback data.
    - ***Database***: Stores feedback comments and ratings.
    - ***API***: Facilitates interaction between parties for feedback submission.

16. **Monitoring of Student Internships by Universities**

    - ***ProfileManager***: Provides university access to students' progress.
    - ***InternshipManager***: Shares updates on the internship.
    - ***Database***: Stores monitoring records and internship status.
    - ***API***: Facilitates interaction between parties for feedback submission.

5.4 Implementation Plan

Our implementation strategy combines two complementary approaches: **bottom-up implementation** and **incremental implementation**, tailored to our small team size and limited time frame. These approaches ensure efficient progress and prioritize delivering functional, high-priority feature early in the development process.

5.4.1 Combined Implementation Strategy

1. **Bottom-Up Implementation**:
   - This approach focuses first on developing core, low-level functionalities. These foundational components provide the building blocks for higher-level features, ensuring stability and reliability as development progresses.
2. **Incremental Implementation**:
   - The features are added one by one, starting with those of highest priority. By addressing critical functionalities first, we ensure that the system delivers value early and minimizes interdependencies between tasks. This reduces the risk of blocked tasks and keeps the workflow smooth.

5.4.2 Adapted Scrum-Like Workflow

Although our team consists of only two members and the available time is short, we adapt the principles of the Scrum framework to suit our needs. This lightweight Scrum-like setup emphasizes flexibility, iterative development, and frequent communication.

1. **Planning**:
   - Identify and prioritize all required tasks using *MoSCoW* prioritization (Must-have, Should-have, Could-have, Won't-have).
   - Divide the workload into manageable units, keeping in mind the dependencies of the tasks.
   - Agree on short, focused iterations (e.g., mini-sprints lasting a few days).
2. **Development Iterations**:
   - At the beginning of each mini-sprint, select a subset of tasks based on their priority and feasibility.
   - During the iteration:
     - Implement core functionalities using a bottom-up approach.
     - Incrementally add prioritized features, testing each one after implementation.
3. **Daily Sync-Ups**:
   - Instead of traditional standups, hold daily check-ins (e.g., 10 minutes) to discuss progress, challenges, and next steps.
   - This practice ensures both team members stay aligned and can quickly address any roadblocks.
4. **Testing and Integration**:
   - Test each feature immediately after its implementation to identify and fix issues early.
   - Perform integration tests frequently to ensure that all components work together as intended.
5. **Review and Adjustment**:
   - At the end of each iteration, review the progress made, and adjust plans for the next iteration as needed.
   - Incorporate any new insights or changes to the scope of the project into the plan.

6. **Delivery**:
    - Performing periodic delivery of functional increments, ensuring that a working product is always available, even in the early stages.
    - Prioritize the most critical features to maximize the value of the project in a limited time.

5.4.3 Focus on Efficiency and Flexibility

Given our small team size and limited time, this Scrum-like workflow prioritizes efficiency, with frequent and direct communication replacing more formalized Scrum ceremonies. By combining bottom-up and incremental strategies, we ensure a steady development pace and maximize the chance of meeting deadlines with a functional and high-quality product.

5.5 Test Plan

This section outlines the test plan for the main features of the platform. The testing approach is designed to ensure robust functionality, secure operation, and seamless user experience. Each feature is tested using **unit testing** combined with **integration testing** to verify the interactions between components. Automated tests are prioritized where applicable to accelerate development and improve accuracy.

5.5.1 Testing Strategy

To ensure comprehensive coverage, **unit testing** is used for validating the implementation of individual features, while **integration testing** ensures proper interaction between components (e.g., *APIs* and *Managers*). For critical workflows like authentication, **security testing** is added to verify data protection and session management. Testing adheres to the guidelines outlined in the ***ISTQB Testing Standards*** (see [3]) and uses tools like *Selenium* for UI testing and *Postman* for API validation.

5.5.2 Testing Methodology

1. **Profile Management**:
    - **Unit Testing**: Validate that profile modification functionality works for all user types and includes form validation for fields like email addresses and contact numbers.
    - **Regression Testing**: Ensure changes to profile data are reflected in the database and on relevant interfaces.
    - **Test Cases**:
        - Edit profile details with valid and invalid input.
        - Verify updates across all connected components.
2. **Profile Management**:
    - **Unit Testing**: Validate that profile modification functionality works for all user types and includes form validation for fields like email addresses and contact numbers.
    - **Regression Testing**: Ensure changes to profile data are reflected in the database and on relevant interfaces.
    - **Test Cases**:
        - Edit profile details with valid and invalid input.
        - Verify updates across all connected components.
3. **Internship Life-cycle Management**:

- **Integration Testing**: Validate interactions between *InternshipManager*, *ApplicationManager*, and *NotificationManager* during the internship life-cycle.
- **Unit Testing**: Confirm that life-cycle transitions (e.g., from position creation to closing) are seamless.
- **Test Cases**:
  - Create internship positions with valid/invalid inputs.
  - Transform internship positions into internship after application acceptance.
  - Verify data consistency after closing internships.

4. **Application Management**:
   - **Unit Testing**: Test application creation, submission, and updates through all stages of the review process (acceptance, rejection, or further assessment).
   - **Boundary Testing**: Validate limits for input fields.
   - **Test Cases**:
     - Apply to positions with valid/invalid information.
     - Change the status of an application and verify user notifications.
     - Test acceptance/rejection handling by students and companies.
5. **Search and Viewing Functionalities**:
   - **Usability Testing**: Ensure the search interface is intuitive and filters function as expected.
   - **Test Cases**:
     - Search internship positions with various filters.
     - Test viewing detailed internship positions, applications, and internship statuses.
     - Validate access permissions for viewing by user role (e.g., student, company, university).
6. **Communication and Feedback**:
   - **Unit Testing**: Test real-time messaging between students and companies, ensuring no messages are lost.
   - **Usability Testing**: Validate that feedback forms are accessible and store data correctly in the system.
   - **Test Cases**:
     - Send messages between users during internships and check delivery information.
     - Submit feedback with valid/invalid inputs and verify system storage.
7. **University Monitoring**:
   - **Integration Testing**: Ensure universities can view internship details and track student progress through the *InternshipManager*.
   - **Unit Testing**: Validate that only authorized university users can access monitoring tools.
   - **Test Cases**:
     - View student internships with valid credentials.
     - Test access control for unauthorized university users.

6 Effort Spent

Project Working Hours

Project Section	**Gribaldo (hours)**	**Rosa (hours)**	**Total Hours**
1) Introduction **2) Architectural Design** **3) User Interface Design** **4) Requirements Traceability** **5) Implementation, Integration and Test Plan**	3 14:30 3 2 1:30	1 13:30 1 1 7	4 28 4 3 8:30
Reading and Checking	3	3:30	6:30
Brainstorming (together)	5	5	10
Total Hours	32	32	64

Table 1: Working Hours per Project Section

References

[1] AWS. 3-tier architecture. Technical report. URL: https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/three-tier-architecture-overview.html.

[2] AsteraSoftware. Rest api. Technical report. URL: https://www.astera.com/type/blog/rest-api-definition/.

[3] ISTQB. ISTQB Standard, Dec., 2024. URL: https://www.istqb.org.