POLITECNICO
MILANO 1863

Design Document

Deliverable:	DD
Title:	Design Document
Authors:	CHIARA BARONE, OTTAVIA BIAGI, MYRIAM RITA CARAVAGGIO
Version:	1.0
Date:	7-January-2025
Download page:	https://github.com/Chiaaa17/BaroneBiagiCaravaggio
Copyright:	Copyright © 2024, CHIARA BARONE, OTTAVIA BIAGI, MYRIAM RITA CARAVAGGIO – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document contains the design description of the Students&Companies system. It includes the architectural design, the user interface, and a description of all the operations that the system will perform. Additionally, it demonstrates how the requirements and use cases detailed in the RASD document are satisfied by the system's design. This document is intended to be read by system developers, testers, and project managers. It is also intended to serve as a reference for the system's future maintenance.

1.2 Scope

The Students&Companies system is a web application designed to facilitate matching between students and companies for internships, either through recommendations or independent searches. Specifically, the system allows students to search for internships while allowing companies to advertise their internship opportunities. The platform identifies and recommends matches, streamlining direct communication between students and companies. Once contact is established, the system supports the selection process by managing interview scheduling and finalizing decisions. In addition, it offers tools for tracking the progress and outcomes of internships, handling complaints, and ensuring effective communication. Universities can also monitor ongoing internships to maintain oversight and support for their students. A more detailed description of the system is available in the RASD document. This document, instead, provides a comprehensive description of the system's design, detailing how the requirements and use cases outlined in the RASD document are implemented.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- **User** Anyone interacting with the system, it can be a student, a company or an university.
- **Manage** Create, supervise and edit a certain element of the application.
- **Recommendation** The suggestion for students and companies aimed at creating a relationship between the two based on information regarding the internship and the student's CV.

1.3.2 Acronyms

- **ST** - Student
- **CO** - Company
- **UNI** - University
- **S&C** - Students&Companies
- **RASD** - Requirements Analysis and Specification Document
- **DD** - Design Document
- **SAT** - Static Analyzer Tool
- **MVC** - Model View Controller

1.3.3 Abbreviations

1.4 Revision History

1.5 Reference Docoments

The specification document of the project: Assignment RDD AY 2024-2025 The RASD document of the project

2 Architectural Design

2.1 Overview: high-level components and interactions

This section describes the high-level architecture of the **Students & Companies (S&C)** platform, focusing on the primary components and their interactions. The architecture is based on a client-server model with a centralized gateway to manage communication between components.

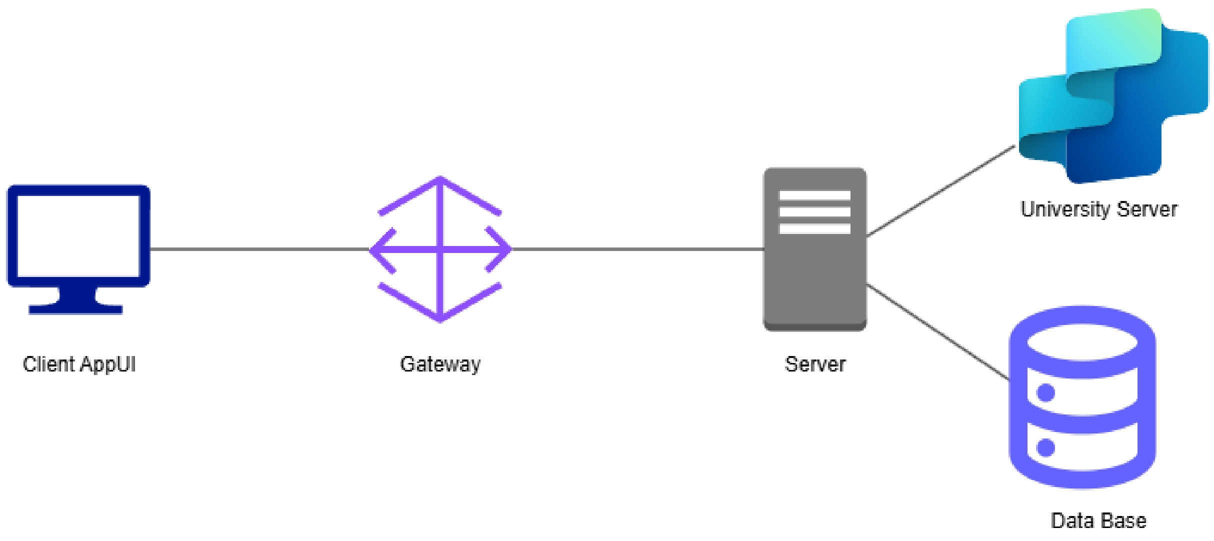


Figure 1: High-Level components interaction

2.1.1 Components and Their Roles

- **Client AppUI**: Represents the user interface that is accessed through a web application. This interface allows users (students, companies, and universities) to perform tasks such as logging in, managing internships, viewing recommendations, and interacting with notifications. Users send requests to the server via the Gateway, initiating actions like authentication or data retrieval.
- **Gateway**: Acts as the central dispatcher, routing all incoming client requests to the appropriate back-end services hosted on the Server. It ensures secure communication and abstracts the complexity of back-end services from the client. Additionally, the Gateway promotes modularity by isolating client-side functionality from server-side logic.
- **Server**: Hosts the core back-end services of the platform. These services manage critical operations such as:
  - User authentication and data management.
  - Internship tracking and recommendations.
  - Feedback collection and notification delivery.

  The server interacts directly with both the University Server and the Database for data exchange and storage.
- **University Server**: An external system integrated with the platform, specifically for handling student authentication. During login or registration, student credentials are validated through the University Server to ensure authenticity.
- **Database**: Stores and manages all persistent data used by the platform. This includes:

– User profiles and account information.
– Internship details and statuses.
– Recommendation data and feedback.

2.1.2 Interactions

The main interactions between the components are as follows:

- The Client AppUI sends requests (e.g., login, internship application publishing) through the Gateway, which forwards them to the Server.
- For student authentication, the Server interacts with the University Server, ensuring valid credentials before granting the registration.
- The Server retrieves or updates data in the Database based on the type of request, such as fetching user profiles, generating recommendations, or logging internship statuses.

2.2 Component View

This section illustrates all components of the system, explaining their roles and how they relate to each other. The system follows a micro-services architecture, where individual services are responsible for specific functionalities, rather than relying on a monolithic server.

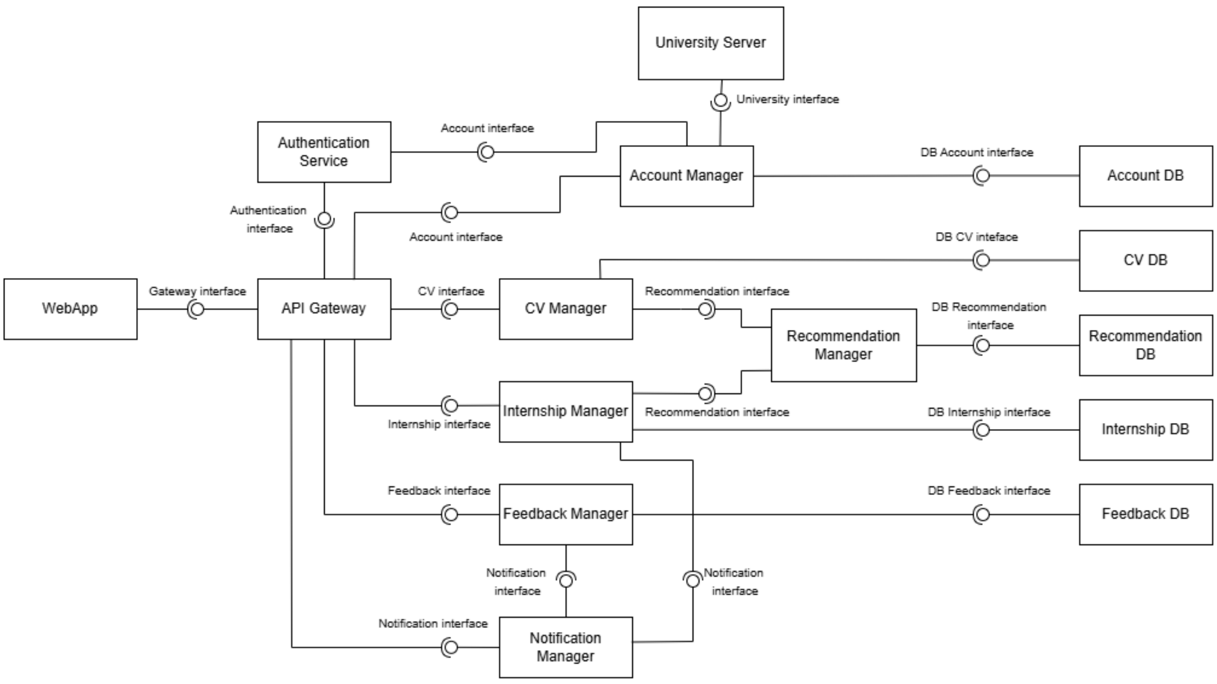


Figure 2: Component View

Client Side

- **WebAppUI:**
  Represents the web application accessible via any modern browser. Users are authenticated through the Authentication Service, which manages the redirection of student registration requests to the University Platform. Additionally, it verifies the user's credentials before granting access to the platform. Once authenticated, users—whether Students, Companies, or Universities—can perform role-specific actions tailored to their needs. Requests are sent through the Gateway Interface, which interacts with the API Gateway to route them to the appropriate micro-service.

Server Side

- **Authentication Service:**
  Manages user authentication and registration.
  - **For students:** The service interacts with the University Platform to verify credentials during registration and handles authentication during the login process.
  - **For companies and universities:** The service independently manages the authentication process.
- **API Gateway:**
  Serves as the dispatcher for all micro-services, receiving requests from the WebAppUI and redirecting them to the appropriate service. The API Gateway ensures that the web application only accesses the functionality necessary for its proper operation.
- **Recommendation Manager:**
  is responsible for generating tailored recommendations for both students and companies. It combines static analysis (e.g., evaluating stored data, such as user profiles, internship descriptions, and historical matches) and dynamic analysis (e.g., assessing real-time interactions, preferences, and feedback) to deliver precise and relevant suggestions.
  - For Students:
    * Suggests internships that align with their profiles and preferences.
    * Provides personalized feedback on their CVs to improve compatibility with internship offers.
  - For Companies:
    * Recommends candidates who match their internship requirements.
    * Provides suggestions to improve the clarity and attractiveness of internship postings.

  The Recommendation Manager performs in-depth analysis and directly integrates data from the Recommendation DB, Internships DB, CV DB, and Account DB to retrieve and process the information necessary for its operations.
- **Internship Manager:**
  Responsible for managing all aspects of internship administration within the platform. It oversees the entire lifecycle of applications, from submission to resolution, ensuring that the status of each application — whether pending, accepted, or rejected — is accurately tracked and updated. The Internship Manager coordinates with the Notification Manager to promptly inform users about changes or updates.
  Beyond tracking applications, the Internship Manager maintains and retrieves data from the Internships DB. This structured data is critical for providing insights to the Recommendation Manager, which uses it to generate precise and tailored recommendations for both students and companies.
  By consolidating internship tracking and data management, the Internship Manager serves as a critical component for delivering a reliable and efficient internship experience to all platform users.
- **Account Manager:**
  Responsible for managing user account information and customizing the WebAppUI based on user roles. It interacts with the Authentication Service for verification and retrieves account data from the Account DB.
- **Notification Manager:**
  The Notification Manager implements the procedures for sending notifications to users. It manages

all notifications within the platform and ensures users are informed about relevant events. It works seamlessly with the Account Manager and other managers to process and deliver notifications. The Notification Manager sends updates to different user groups:

  - **Students:** Notified of new requests, acceptance or rejection of internships, and interview scheduling, moreover of internship interruption.
  - **Companies:** Informed about new applications, the acceptance or rejection of offers, interview scheduling, and internship completions or interruptions.
  - **Universities:** Updated when internships start or end and when feedback about internships is made.

- **Feedback Manager:**
  Responsible for collecting, processing, and managing feedback provided by platform users, including students, companies, and universities. This feedback encompasses various aspects, such as about internship experiences of students, system functionality, and general platform usage.
  The Feedback Manager interacts with the Feedback DB to store and retrieve information and works closely with the Account Manager and Notification Manager to notify users about feedback submissions or updates.

- **CV Manager:**
  Handles operations related to CV management. It retrieves, stores, and updates CVs in the CV DB and interacts with the Recommendation Manager to provide insights for better recommendations. The CV Manager also collaborates with the Notification Manager to inform users about CV-related events, such as required updates.

Databases

- **Account DB:**
  Stores user account information, including personal details and credentials required for system access.

- **Recommendation DB:**
  Contains data relevant to recommendations, such as user preferences, historical matches, feedback from previous recommendations, and internship requirements. This database serves as the foundation for generating tailored suggestions for both students and companies.

- **Internships DB:**
  Stores information about internships, including details about active, completed, and pending internship opportunities.

- **Feedback DB:**
  Stores information about feedback from all users, enabling further analysis and insights for system improvement.

- **CV DB:**
  Contains detailed information about CVs, including versions, feedback, and associated updates.

External Components

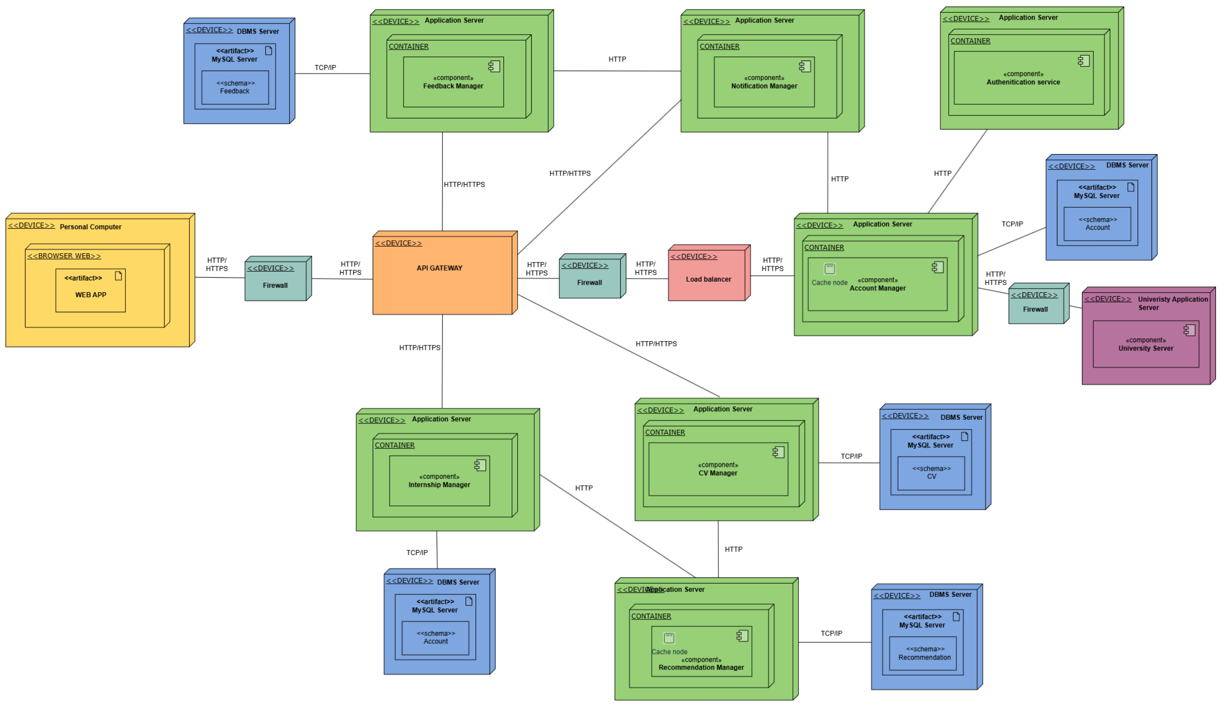
- **University Platform:**
  An external system used by the Authentication Service to verify the credentials of the students during the registration process.

2.3 Deployment View

The architecture of the Students & Companies (S&C) platform is based on a micro-services design, ensuring scalability, modularity, and maintainability. Each micro-service operates independently on a dedicated application server and connects to its own MySQL database using TCP/IP. Communication between micro-services and external components is handled via HTTP/HTTPS to ensure secure data transmission, while internal communication between micro-services also leverages HTTP or HTTPS for enhanced data protection and reliability.

To prevent data congestion and accelerate application performance, a load balancer is placed before the Account Manager, as it is the most frequently accessed component. Additionally, caching mechanisms are implemented for both the Account Manager and Recommendation Manager to improve response times for repeated queries, thereby reducing the overall computational load.

This architecture, leveraging load balancing, caching, and robust communication protocols, ensures a scalable, efficient, and secure platform capable of meeting diverse user needs.



<<DEVICE>> DBMS Server
<<artifact>> MySQL Server
<<schema>> Feedback
TCP/IP
<<DEVICE>> Application Server
CONTAINER
«component» Feedback Manager
HTTP
<<DEVICE>> Application Server
CONTAINER
«component» Notification Manager
<<DEVICE>> Application Server
CONTAINER
«component» Authenitication service
HTTP/HTTPS
HTTP/HTTPS
HTTP
HTTP
<<DEVICE>> DBMS Server
<<artifact>> MySQL Server
<<schema>> Account
TCP/IP
<<DEVICE>> Personal Computer
<<BROWSER WEB>>
<<artifact>> WEB APP
HTTP/ HTTPS
<<DEVICE>> Firewall
HTTP/ HTTPS
<<DEVICE>> API GATEWAY
HTTP/ HTTPS
<<DEVICE>> Firewall
HTTP/ HTTPS
<<DEVICE>> Load balancer
HTTP/ HTTPS
<<DEVICE>> Application Server
CONTAINER
Cache node
«component» Account Manager
HTTP/ HTTPS
<<DEVICE>> Firewall
<<DEVICE>> Univeristy Application Server
«component» University Server
HTTP/HTTPS
HTTP/HTTPS
<<DEVICE>> Application Server
CONTAINER
«component» Internship Manager
HTTP
<<DEVICE>> Application Server
CONTAINER
«component» CV Manager
TCP/IP
<<DEVICE>> DBMS Server
<<artifact>> MySQL Server
<<schema>> CV
TCP/IP
HTTP
<<DEVICE>> DBMS Server
<<artifact>> MySQL Server
<<schema>> Account
<<DEVICE>> Application Server
CONTAINER
Cache node
«component» Recommendation Manager
TCP/IP
<<DEVICE>> DBMS Server
<<artifact>> MySQL Server
<<schema>> Recommendation

Figure 3: Component Deployment View

The diagram shows the deployment view of the system. It illustrates the distribution of the system components between different nodes and how they communicate. The system is composed of four tiers:

- **Client Tier:** Includes the Web browser used by users to access the application.
- **Application Tier:** Hosts key services like the Authentication Service, Account Manager, Notification Manager, CV Manager, and Internship Manager.
- **Data Tier:** Composed of MySQL databases, each dedicated to a specific micro-service (Account DB, CV DB, Recommendation DB, and Internship DB).
- **Evaluation Tier:** Hosts the Recommendation Manager, equipped with a static analyzer for advanced data processing.

Additionally, the system integrates with an external University Server, used to authenticate student registration.

2.3.1 Load Balancing

Given that the S&C platform will potentially have a large number of concurrent users, a load balancer has been implemented to distribute the load among multiple servers. This design choice ensures scalability as more servers can be added to handle increasing loads. Load balancing is particularly critical for the Account Manager, which handles a significant number of user authentication and data management requests. By distributing the workload, the platform minimizes response times and ensures uninterrupted service even during peak usage periods.

2.3.2 Caching Mechanism

To enhance system performance and efficiency, caching mechanisms have been implemented for components that handle a large number of repeated requests.

- **Account Manager:** Frequently accessed user data is cached to reduce the load on the Account DB and improve response times for user-related queries.
- **Recommendation Manager:** Caching plays a crucial role in this component, as it performs computationally intensive tasks such as static analysis and data evaluation. By caching the results of prior analyses, the system avoids redundant computations, thereby reducing latency and improving scalability.

This caching mechanism not only optimizes resource utilization but also ensures faster responses for frequently requested data, contributing to an overall better user experience.

2.3.3 Firewall

To protect the application from external threats, multiple firewalls have been deployed to filter traffic and to protect the more sensible devices. Specifically:

- A firewall is placed between the client and the API Gateway.
- A firewall is implemented between the API Gateway and the load balancer before the Account Manager,
- A firewall is implemented between the Account Manager and the University Server.

This layered security approach strengthens the application's defense by filtering and inspecting incoming traffic before it reaches critical system components, ensuring robust protection against potential threats.

2.4 Runtime Views

This section contains the sequence diagrams of the most important operations of the system. The diagrams include the components that we have already described in the previous section and the external components that are involved in the operations.

2.4.1 Log In

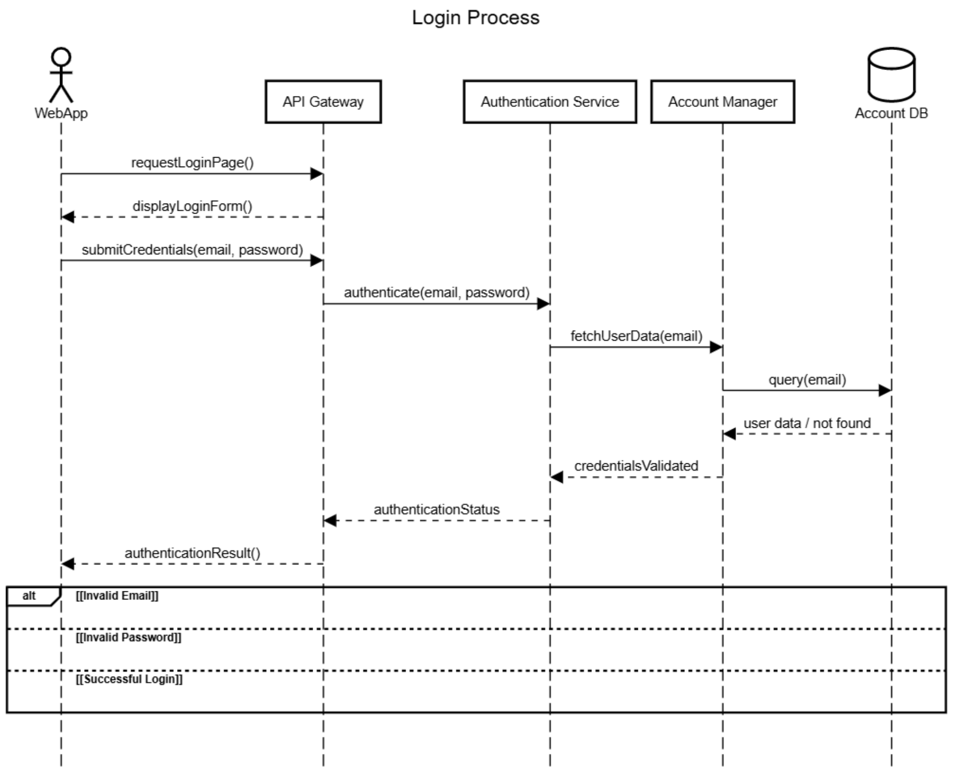


Figure 4: Log In

2.4.2 Register

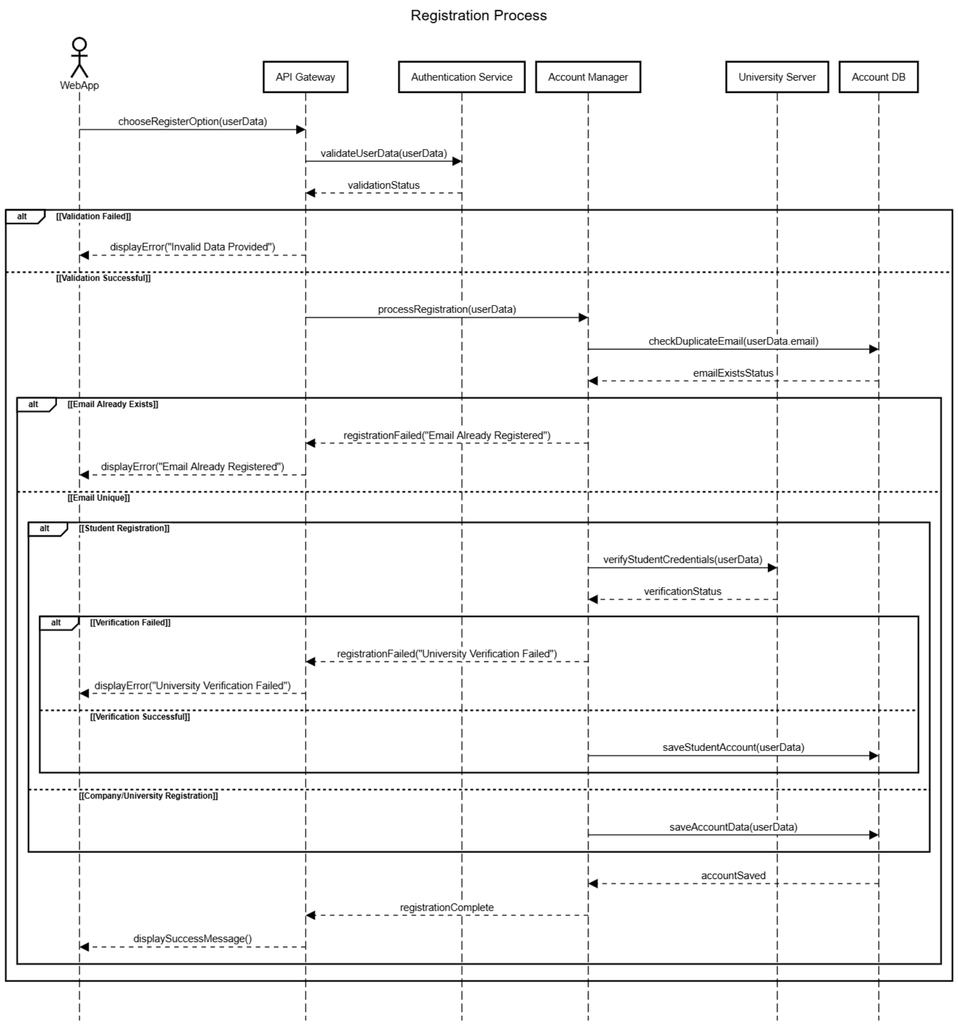
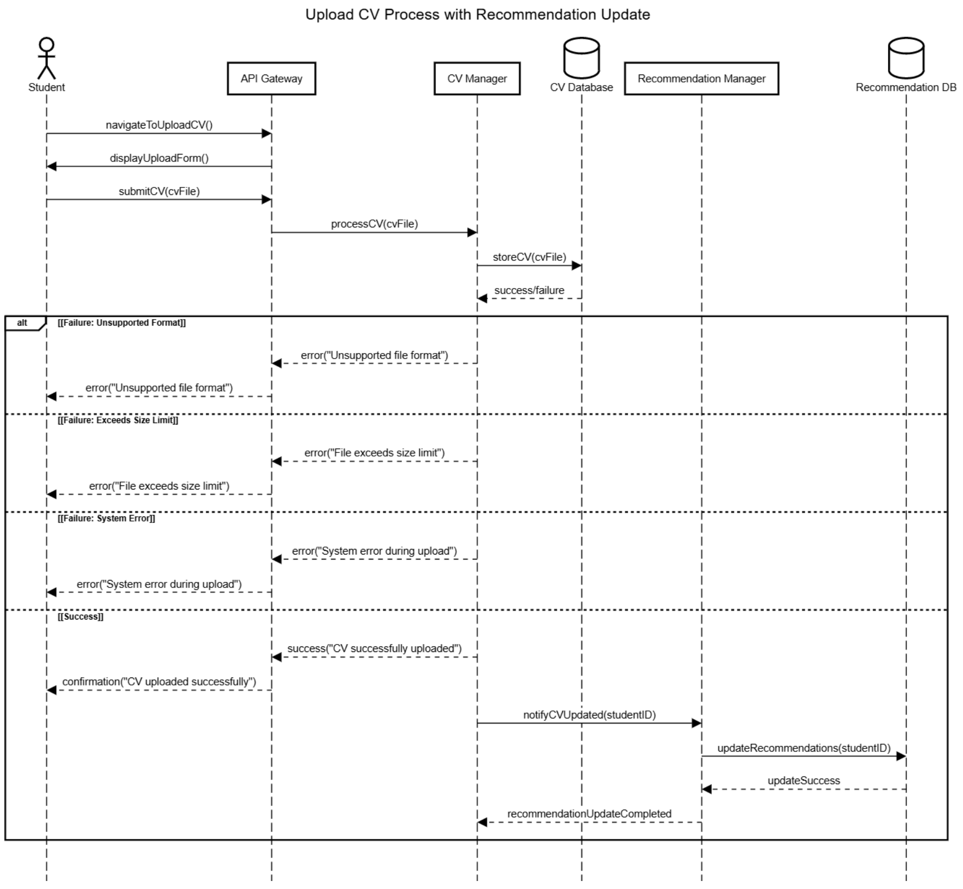

Registration Process
WebApp
API Gateway
Authentication Service
Account Manager
University Server
Account DB
chooseRegisterOption(userData)
validateUserData(userData)
validationStatus
alt
[[Validation Failed]]
displayError("Invalid Data Provided")
[[Validation Successful]]
processRegistration(userData)
checkDuplicateEmail(userData.email)
emailExistsStatus
alt
[[Email Already Exists]]
registrationFailed("Email Already Registered")
displayError("Email Already Registered")
[[Email Unique]]
alt
[[Student Registration]]
verifyStudentCredentials(userData)
verificationStatus
alt
[[Verification Failed]]
registrationFailed("University Verification Failed")
displayError("University Verification Failed")
[[Verification Successful]]
saveStudentAccount(userData)
[[Company/University Registration]]
saveAccountData(userData)
accountSaved
registrationComplete
displaySuccessMessage()

Figure 5: Register

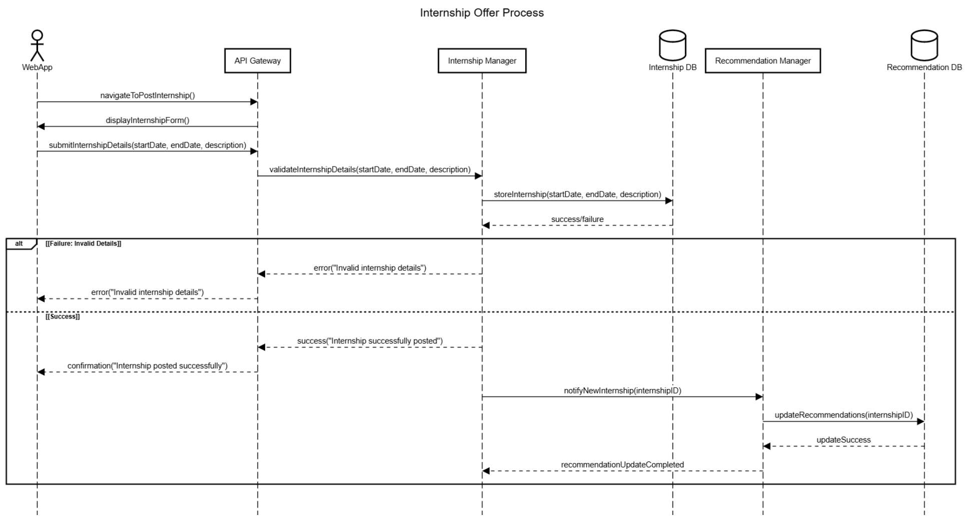
2.4.3 Upload CV



Upload CV Process with Recommendation Update
Student
API Gateway
CV Manager
CV Database
Recommendation Manager
Recommendation DB
navigateToUploadCV()
displayUploadForm()
submitCV(cvFile)
processCV(cvFile)
storeCV(cvFile)
success/failure
alt
[[Failure: Unsupported Format]]
error("Unsupported file format")
error("Unsupported file format")
[[Failure: Exceeds Size Limit]]
error("File exceeds size limit")
error("File exceeds size limit")
[[Failure: System Error]]
error("System error during upload")
error("System error during upload")
[[Success]]
success("CV successfully uploaded")
confirmation("CV uploaded successfully")
notifyCVUpdated(studentID)
updateRecommendations(studentID)
updateSuccess
recommendationUpdateCompleted

Figure 6: Upload CV

2.4.4 Internship Offer



Internship Offer Process
WebApp
API Gateway
Internship Manager
Internship DB
Recommendation Manager
Recommendation DB
navigateToPostInternship()
displayInternshipForm()
submitInternshipDetails(startDate, endDate, description)
validateInternshipDetails(startDate, endDate, description)
storeInternship(startDate, endDate, description)
success/failure
alt
[[Failure: Invalid Details]]
error("Invalid internship details")
error("Invalid internship details")
[[Success]]
success("Internship successfully posted")
confirmation("Internship posted successfully")
notifyNewInternship(internshipID)
updateRecommendations(internshipID)
updateSuccess
recommendationUpdateCompleted

Figure 7: Internship Offer

2.4.5 Recommendation - Student about Internships

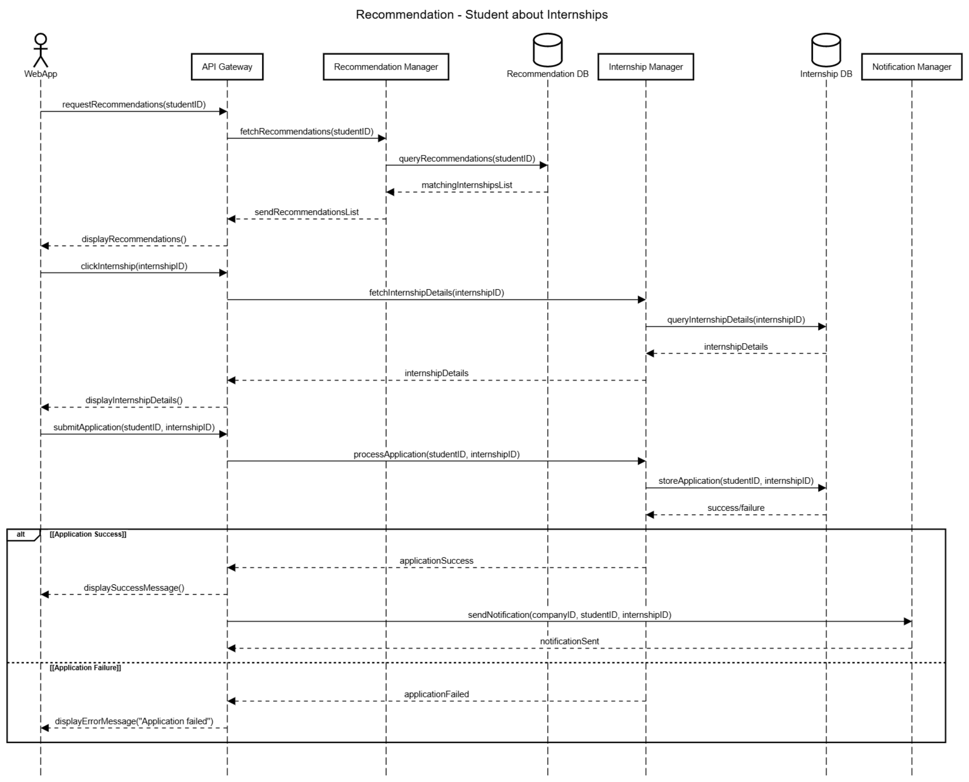

Recommendation - Student about Internships
WebApp
API Gateway
Recommendation Manager
Recommendation DB
Internship Manager
Internship DB
Notification Manager
requestRecommendations(studentID)
fetchRecommendations(studentID)
queryRecommendations(studentID)
matchingInternshipsList
sendRecommendationsList
displayRecommendations()
clickInternship(internshipID)
fetchInternshipDetails(internshipID)
queryInternshipDetails(internshipID)
internshipDetails
internshipDetails
displayInternshipDetails()
submitApplication(studentID, internshipID)
processApplication(studentID, internshipID)
storeApplication(studentID, internshipID)
success/failure
alt
[[Application Success]]
applicationSuccess
displaySuccessMessage()
sendNotification(companyID, studentID, internshipID)
notificationSent
[[Application Failure]]
applicationFailed
displayErrorMessage("Application failed")

Figure 8: Recommendation - Student about Internships

2.4.6 Recommendation - Student about CV

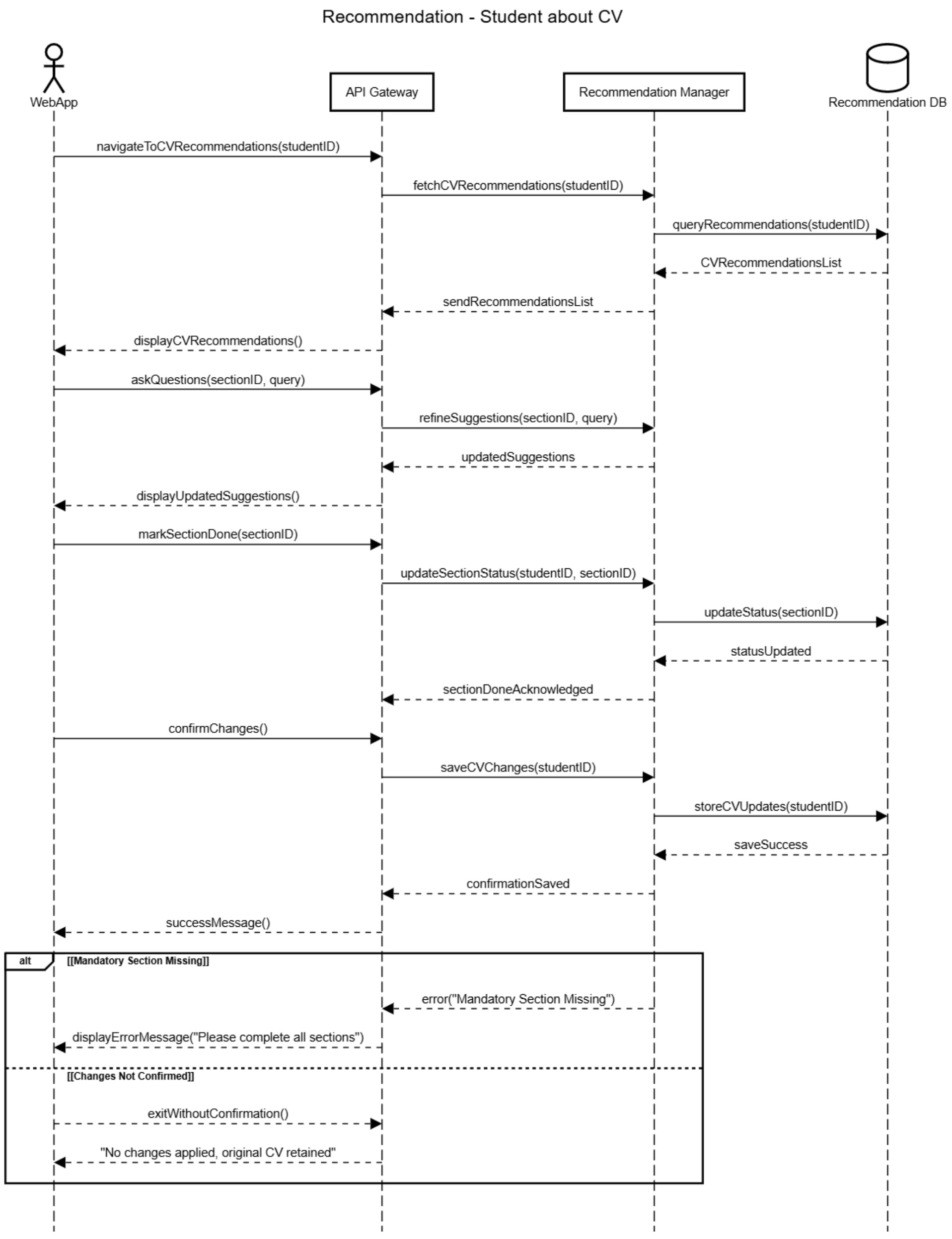

Recommendation - Student about CV
WebApp
API Gateway
Recommendation Manager
Recommendation DB
navigateToCVRecommendations(studentID)
fetchCVRecommendations(studentID)
queryRecommendations(studentID)
CVRecommendationsList
sendRecommendationsList
displayCVRecommendations()
askQuestions(sectionID, query)
refineSuggestions(sectionID, query)
updatedSuggestions
displayUpdatedSuggestions()
markSectionDone(sectionID)
updateSectionStatus(studentID, sectionID)
updateStatus(sectionID)
statusUpdated
sectionDoneAcknowledged
confirmChanges()
saveCVChanges(studentID)
storeCVUpdates(studentID)
saveSuccess
confirmationSaved
successMessage()
alt
[[Mandatory Section Missing]]
error("Mandatory Section Missing")
displayErrorMessage("Please complete all sections")
[[Changes Not Confirmed]]
exitWithoutConfirmation()
"No changes applied, original CV retained"

Figure 9: Recommendation - Student about CV

2.4.7 Recommendation - Company about the Internship Offer

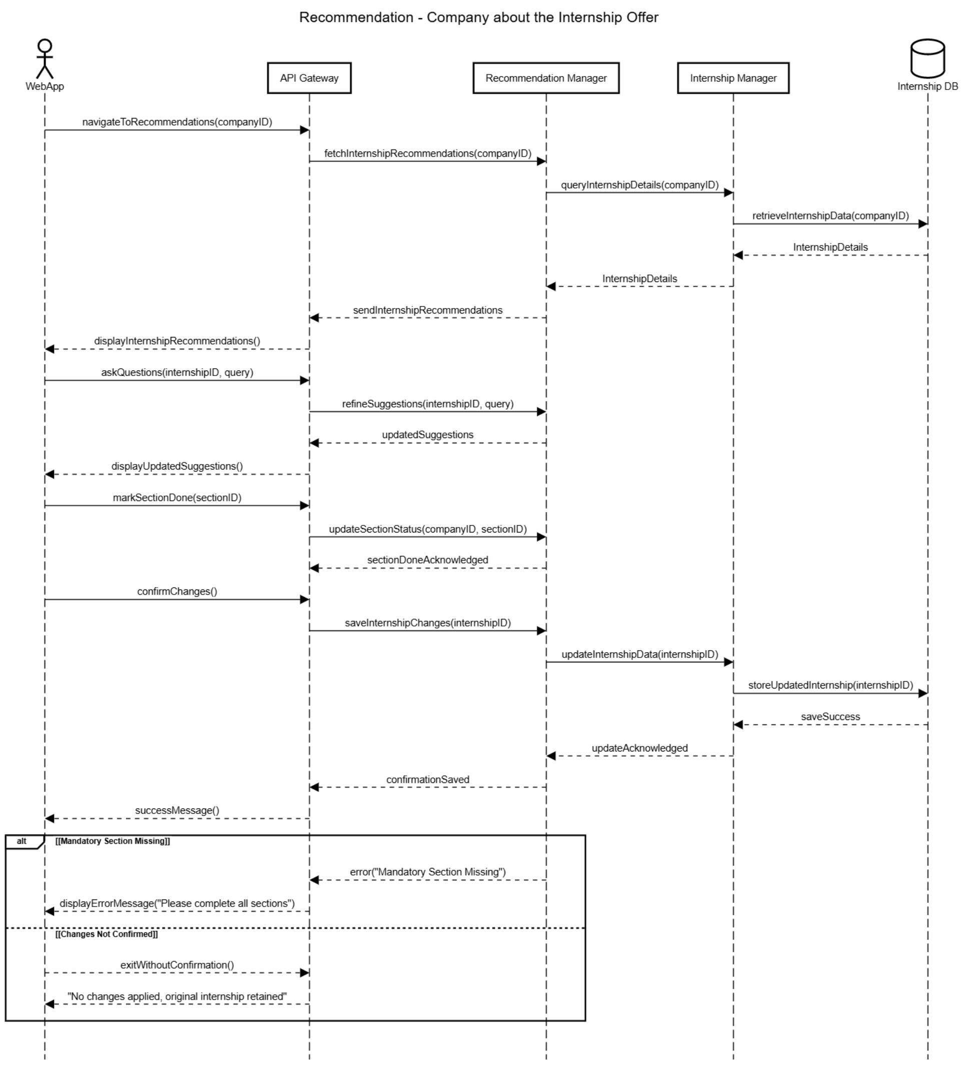

Recommendation - Company about the Internship Offer
WebApp
API Gateway
Recommendation Manager
Internship Manager
Internship DB
navigateToRecommendations(companyID)
fetchInternshipRecommendations(companyID)
queryInternshipDetails(companyID)
retrieveInternshipData(companyID)
InternshipDetails
InternshipDetails
sendInternshipRecommendations
displayInternshipRecommendations()
askQuestions(internshipID, query)
refineSuggestions(internshipID, query)
updatedSuggestions
displayUpdatedSuggestions()
markSectionDone(sectionID)
updateSectionStatus(companyID, sectionID)
sectionDoneAcknowledged
confirmChanges()
saveInternshipChanges(internshipID)
updateInternshipData(internshipID)
storeUpdatedInternship(internshipID)
saveSuccess
updateAcknowledged
confirmationSaved
successMessage()
alt
[[Mandatory Section Missing]]
error("Mandatory Section Missing")
displayErrorMessage("Please complete all sections")
[[Changes Not Confirmed]]
exitWithoutConfirmation()
"No changes applied, original internship retained"

Figure 10: Recommendation - Company about the Internship Offer

2.4.8 Recommendation - Company about Students

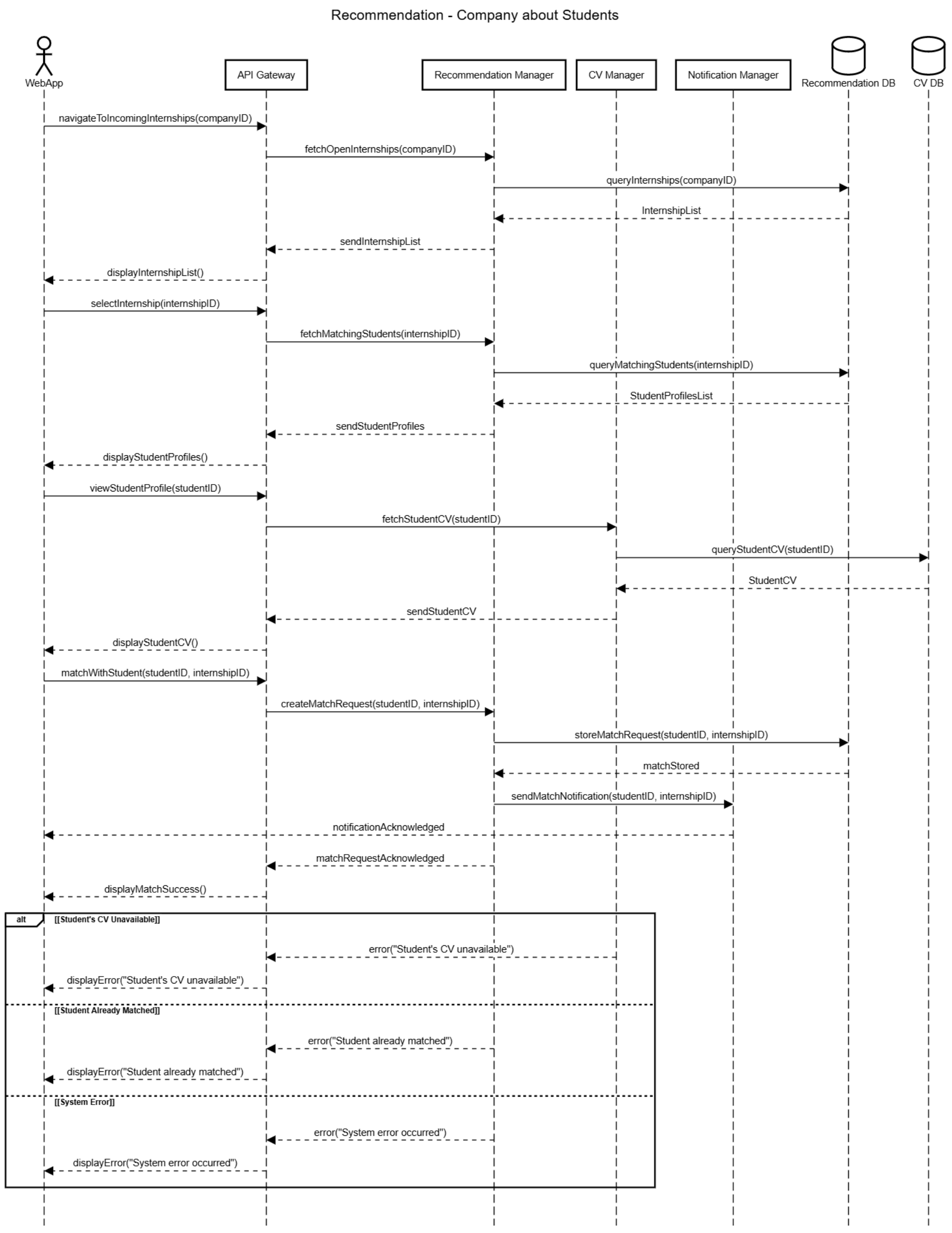

Recommendation - Company about Students
WebApp
API Gateway
Recommendation Manager
CV Manager
Notification Manager
Recommendation DB
CV DB
navigateToIncomingInternships(companyID)
fetchOpenInternships(companyID)
queryInternships(companyID)
InternshipList
sendInternshipList
displayInternshipList()
selectInternship(internshipID)
fetchMatchingStudents(internshipID)
queryMatchingStudents(internshipID)
StudentProfilesList
sendStudentProfiles
displayStudentProfiles()
viewStudentProfile(studentID)
fetchStudentCV(studentID)
queryStudentCV(studentID)
StudentCV
sendStudentCV
displayStudentCV()
matchWithStudent(studentID, internshipID)
createMatchRequest(studentID, internshipID)
storeMatchRequest(studentID, internshipID)
matchStored
sendMatchNotification(studentID, internshipID)
notificationAcknowledged
matchRequestAcknowledged
displayMatchSuccess()
alt
[[Student's CV Unavailable]]
error("Student's CV unavailable")
displayError("Student's CV unavailable")
[[Student Already Matched]]
error("Student already matched")
displayError("Student already matched")
[[System Error]]
error("System error occurred")
displayError("System error occurred")

Figure 11: Recommendation - Company about Students

2.4.9 Selection Process - Company

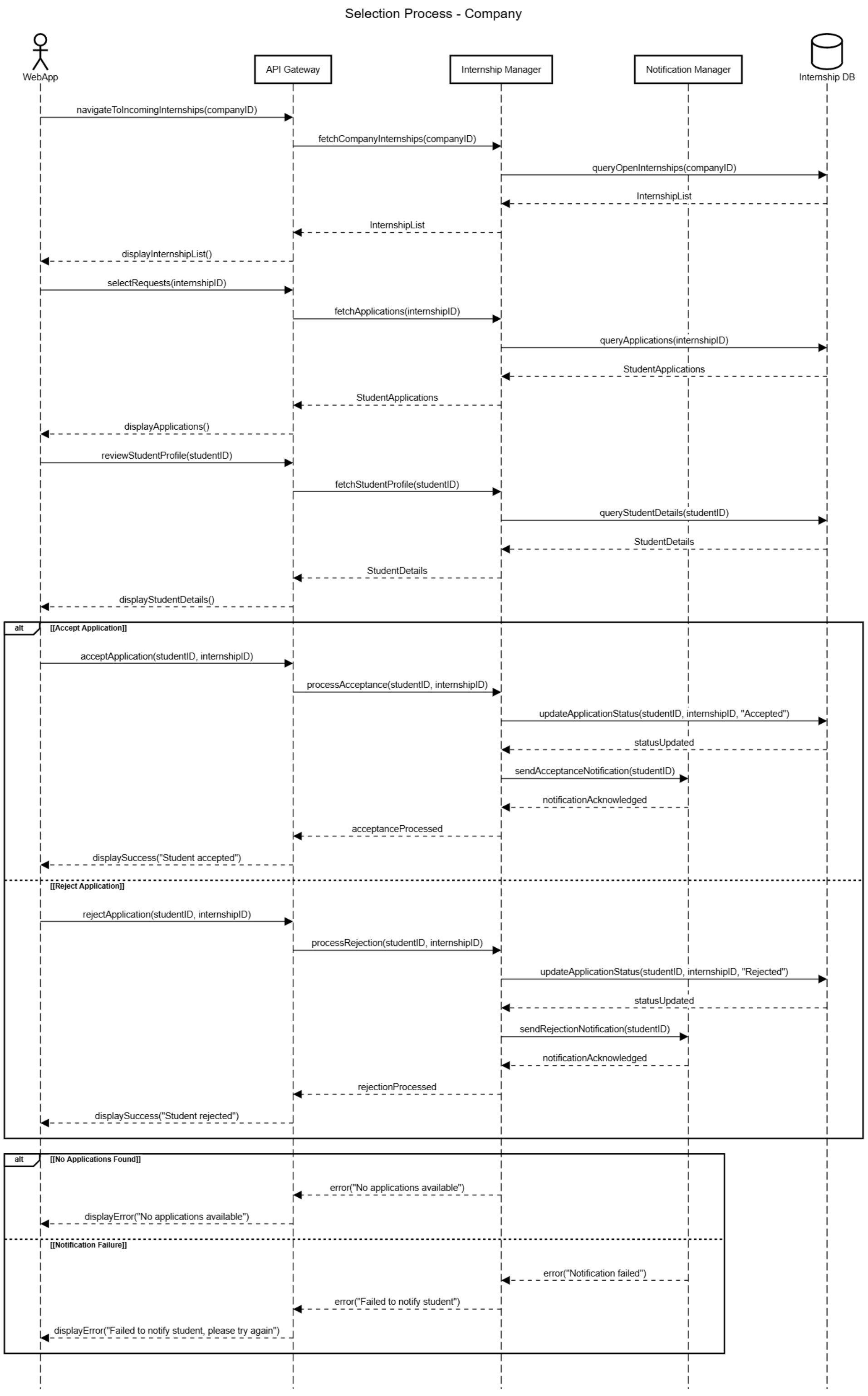
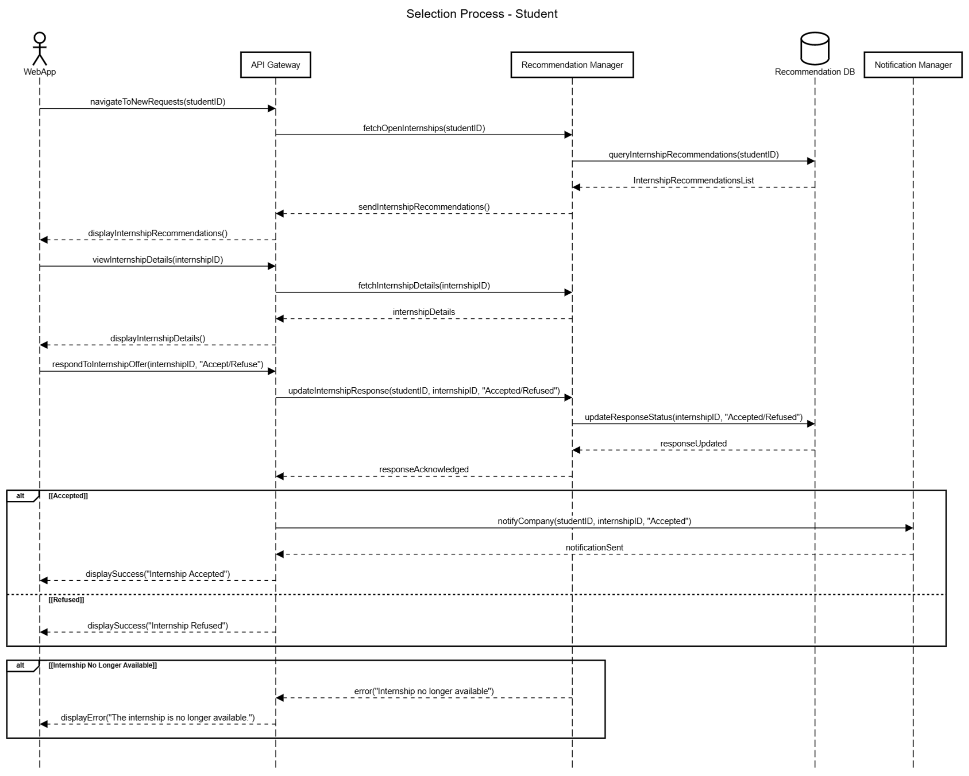

Selection Process - Company
WebApp
API Gateway
Internship Manager
Notification Manager
Internship DB
navigateToIncomingInternships(companyID)
fetchCompanyInternships(companyID)
queryOpenInternships(companyID)
InternshipList
InternshipList
displayInternshipList()
selectRequests(internshipID)
fetchApplications(internshipID)
queryApplications(internshipID)
StudentApplications
StudentApplications
displayApplications()
reviewStudentProfile(studentID)
fetchStudentProfile(studentID)
queryStudentDetails(studentID)
StudentDetails
StudentDetails
displayStudentDetails()
alt
[[Accept Application]]
acceptApplication(studentID, internshipID)
processAcceptance(studentID, internshipID)
updateApplicationStatus(studentID, internshipID, "Accepted")
statusUpdated
sendAcceptanceNotification(studentID)
notificationAcknowledged
acceptanceProcessed
displaySuccess("Student accepted")
[[Reject Application]]
rejectApplication(studentID, internshipID)
processRejection(studentID, internshipID)
updateApplicationStatus(studentID, internshipID, "Rejected")
statusUpdated
sendRejectionNotification(studentID)
notificationAcknowledged
rejectionProcessed
displaySuccess("Student rejected")
alt
[[No Applications Found]]
error("No applications available")
displayError("No applications available")
[[Notification Failure]]
error("Notification failed")
error("Failed to notify student")
displayError("Failed to notify student, please try again")

Figure 12: Selection Process - Company

2.4.10 Selection Process - Student



Selection Process - Student
WebApp
API Gateway
Recommendation Manager
Recommendation DB
Notification Manager
navigateToNewRequests(studentID)
fetchOpenInternships(studentID)
queryInternshipRecommendations(studentID)
InternshipRecommendationsList
sendInternshipRecommendations()
displayInternshipRecommendations()
viewInternshipDetails(internshipID)
fetchInternshipDetails(internshipID)
internshipDetails
displayInternshipDetails()
respondToInternshipOffer(internshipID, "Accept/Refuse")
updateInternshipResponse(studentID, internshipID, "Accepted/Refused")
updateResponseStatus(internshipID, "Accepted/Refused")
responseUpdated
responseAcknowledged
alt
[[Accepted]]
notifyCompany(studentID, internshipID, "Accepted")
notificationSent
displaySuccess("Internship Accepted")
[[Refused]]
displaySuccess("Internship Refused")
alt
[[Internship No Longer Available]]
error("Internship no longer available")
displayError("The internship is no longer available.")

Figure 13: Selection Process - Student

2.4.11 Interviews Management

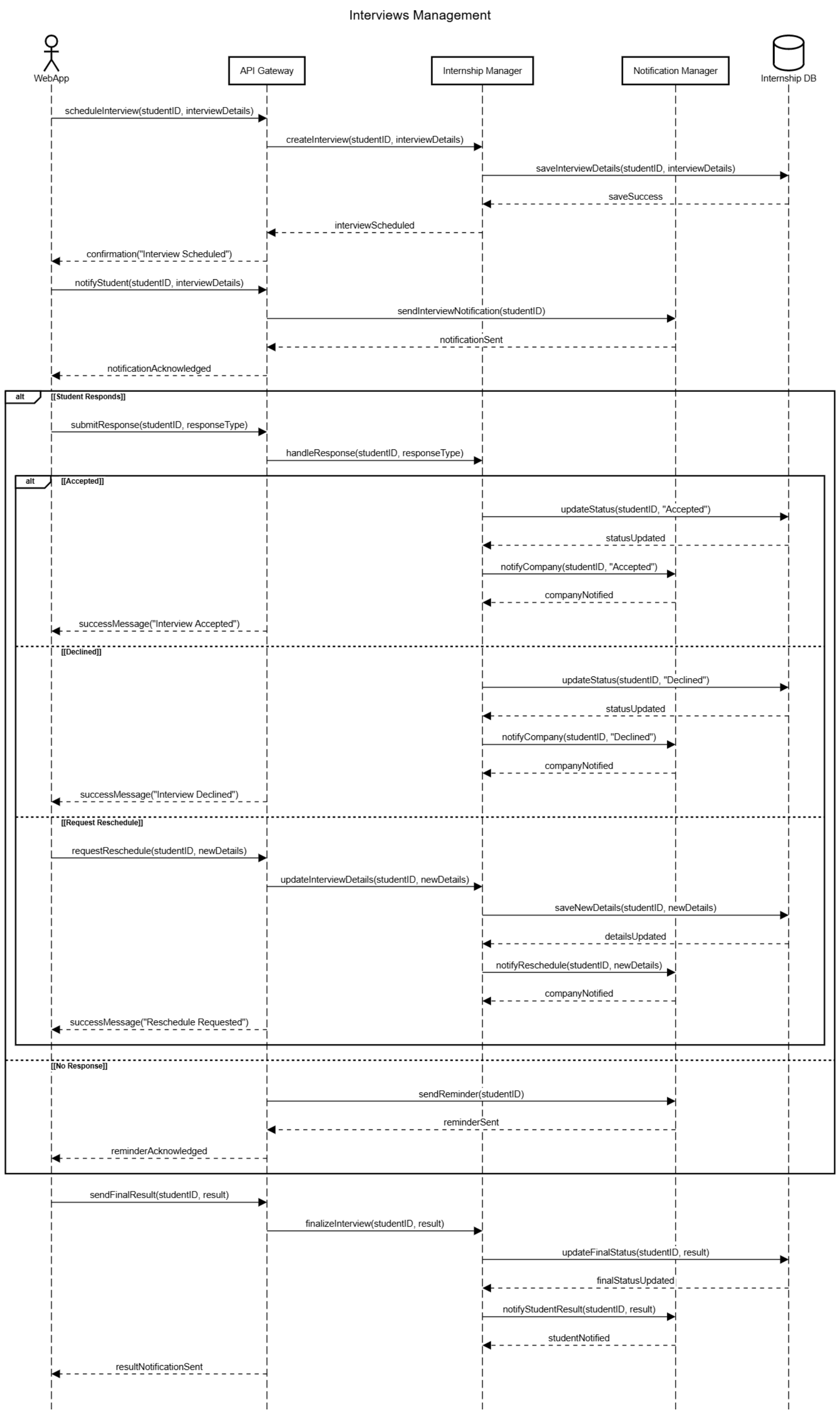

Interviews Management
WebApp
API Gateway
Internship Manager
Notification Manager
Internship DB
scheduleInterview(studentID, interviewDetails)
createInterview(studentID, interviewDetails)
saveInterviewDetails(studentID, interviewDetails)
saveSuccess
interviewScheduled
confirmation("Interview Scheduled")
notifyStudent(studentID, interviewDetails)
sendInterviewNotification(studentID)
notificationSent
notificationAcknowledged
alt
[[Student Responds]]
submitResponse(studentID, responseType)
handleResponse(studentID, responseType)
alt
[[Accepted]]
updateStatus(studentID, "Accepted")
statusUpdated
notifyCompany(studentID, "Accepted")
companyNotified
successMessage("Interview Accepted")
[[Declined]]
updateStatus(studentID, "Declined")
statusUpdated
notifyCompany(studentID, "Declined")
companyNotified
successMessage("Interview Declined")
[[Request Reschedule]]
requestReschedule(studentID, newDetails)
updateInterviewDetails(studentID, newDetails)
saveNewDetails(studentID, newDetails)
detailsUpdated
notifyReschedule(studentID, newDetails)
companyNotified
successMessage("Reschedule Requested")
[[No Response]]
sendReminder(studentID)
reminderSent
reminderAcknowledged
sendFinalResult(studentID, result)
finalizeInterview(studentID, result)
updateFinalStatus(studentID, result)
finalStatusUpdated
notifyStudentResult(studentID, result)
studentNotified
resultNotificationSent

Figure 14: Interviews Management

2.4.12 Interviews Participation

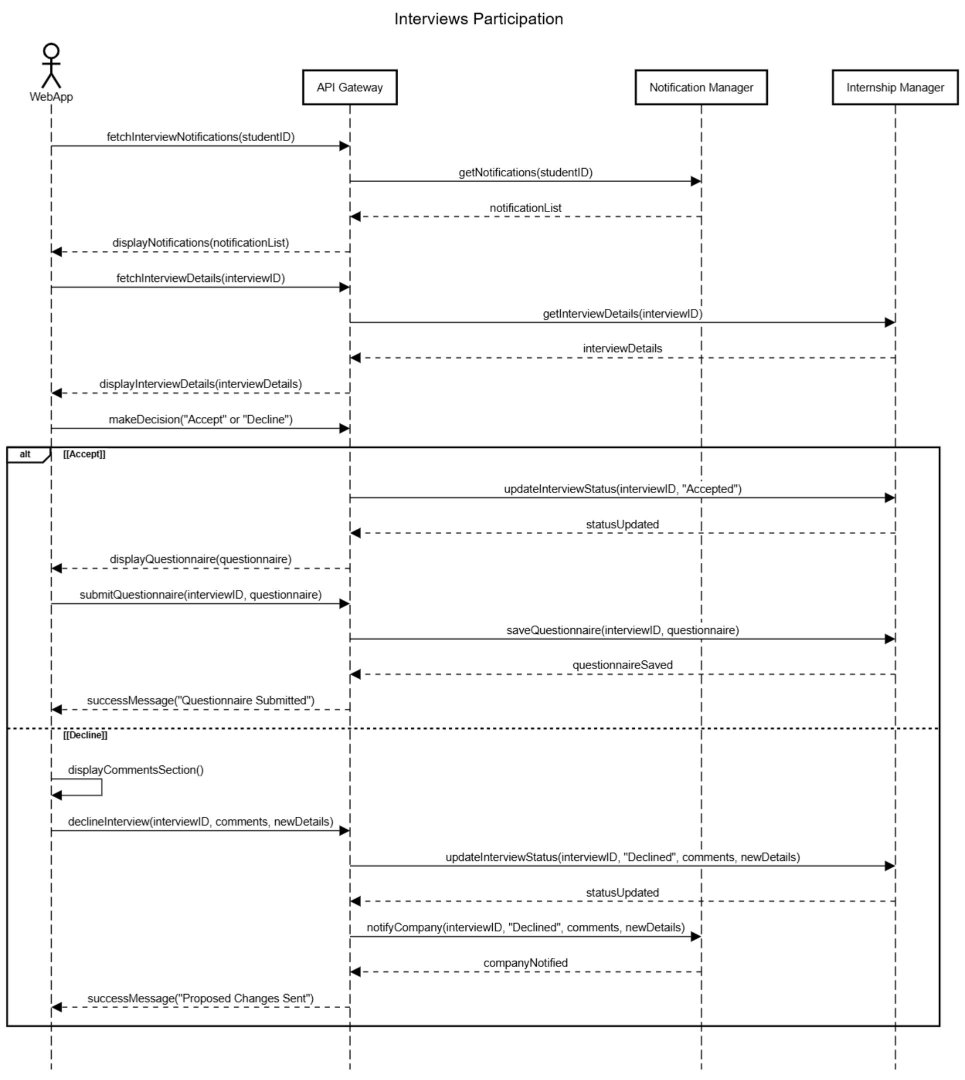

Interviews Participation
WebApp
API Gateway
Notification Manager
Internship Manager
fetchInterviewNotifications(studentID)
getNotifications(studentID)
notificationList
displayNotifications(notificationList)
fetchInterviewDetails(interviewID)
getInterviewDetails(interviewID)
interviewDetails
displayInterviewDetails(interviewDetails)
makeDecision("Accept" or "Decline")
alt
[[Accept]]
updateInterviewStatus(interviewID, "Accepted")
statusUpdated
displayQuestionnaire(questionnaire)
submitQuestionnaire(interviewID, questionnaire)
saveQuestionnaire(interviewID, questionnaire)
questionnaireSaved
successMessage("Questionnaire Submitted")
[[Decline]]
displayCommentsSection()
declineInterview(interviewID, comments, newDetails)
updateInterviewStatus(interviewID, "Declined", comments, newDetails)
statusUpdated
notifyCompany(interviewID, "Declined", comments, newDetails)
companyNotified
successMessage("Proposed Changes Sent")

Figure 15: Interviews Participation

2.4.13 Monitoring of internships

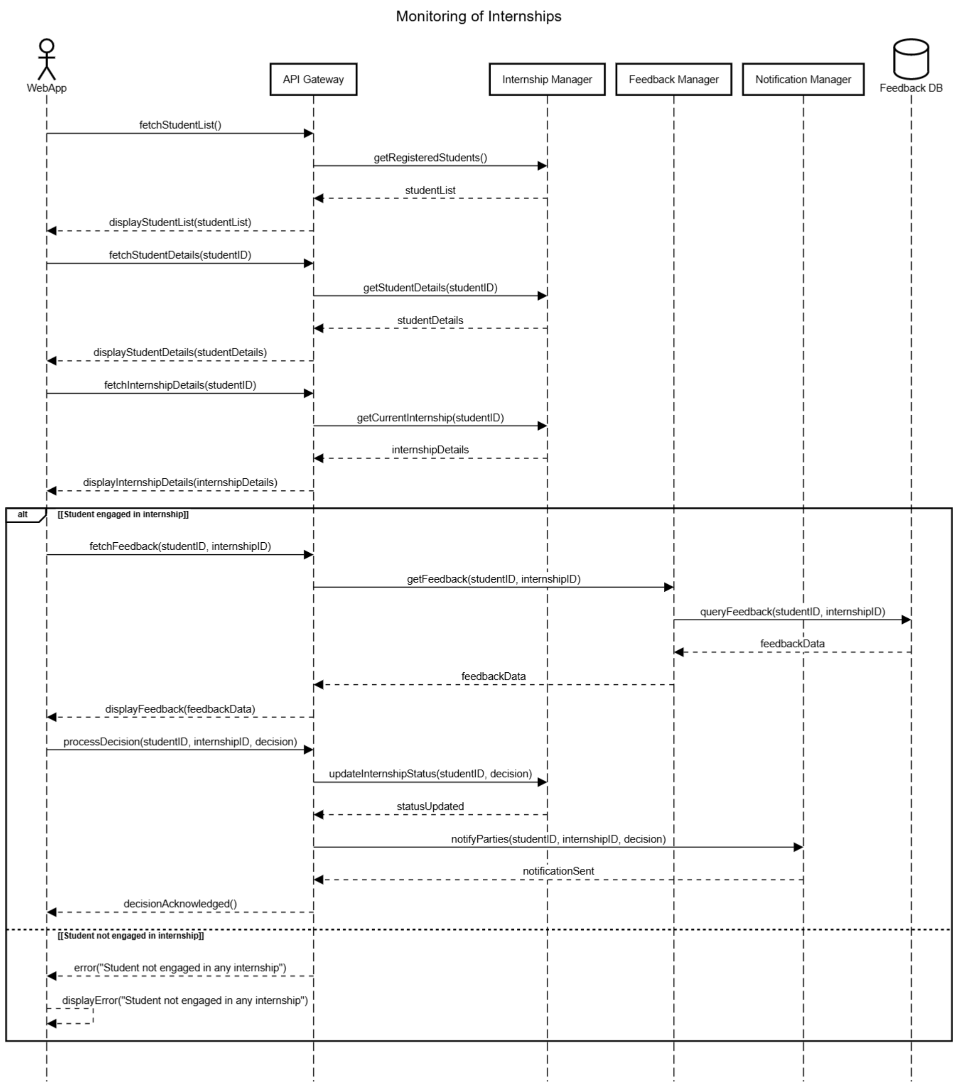

Monitoring of Internships
WebApp
API Gateway
Internship Manager
Feedback Manager
Notification Manager
Feedback DB
fetchStudentList()
getRegisteredStudents()
studentList
displayStudentList(studentList)
fetchStudentDetails(studentID)
getStudentDetails(studentID)
studentDetails
displayStudentDetails(studentDetails)
fetchInternshipDetails(studentID)
getCurrentInternship(studentID)
internshipDetails
displayInternshipDetails(internshipDetails)
alt
[[Student engaged in internship]]
fetchFeedback(studentID, internshipID)
getFeedback(studentID, internshipID)
queryFeedback(studentID, internshipID)
feedbackData
feedbackData
displayFeedback(feedbackData)
processDecision(studentID, internshipID, decision)
updateInternshipStatus(studentID, decision)
statusUpdated
notifyParties(studentID, internshipID, decision)
notificationSent
decisionAcknowledged()
[[Student not engaged in internship]]
error("Student not engaged in any internship")
displayError("Student not engaged in any internship")

Figure 16: Monitoring of internships

2.4.14 Feedback and Suggestions

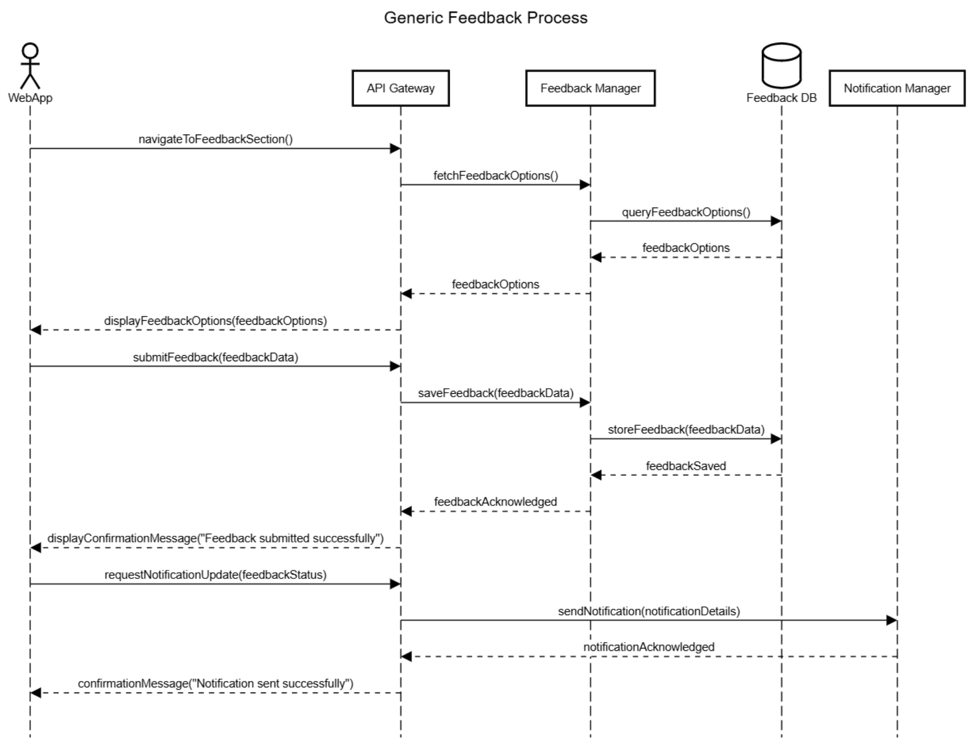

Generic Feedback Process
WebApp
API Gateway
Feedback Manager
Feedback DB
Notification Manager
navigateToFeedbackSection()
fetchFeedbackOptions()
queryFeedbackOptions()
feedbackOptions
feedbackOptions
displayFeedbackOptions(feedbackOptions)
submitFeedback(feedbackData)
saveFeedback(feedbackData)
storeFeedback(feedbackData)
feedbackSaved
feedbackAcknowledged
displayConfirmationMessage("Feedback submitted successfully")
requestNotificationUpdate(feedbackStatus)
sendNotification(notificationDetails)
notificationAcknowledged
confirmationMessage("Notification sent successfully")

Figure 17: Feedback and Suggestions

2.4.15 End of contract

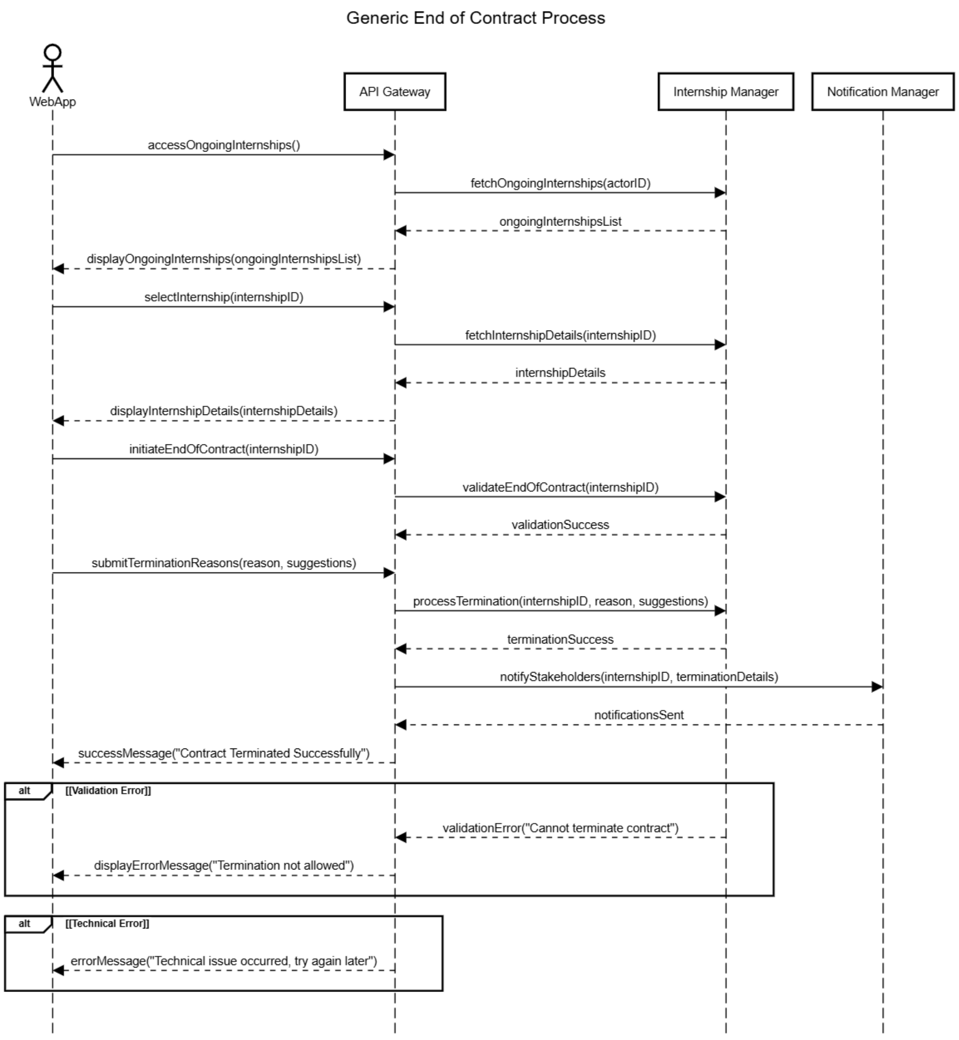

Generic End of Contract Process
WebApp
API Gateway
Internship Manager
Notification Manager
accessOngoingInternships()
fetchOngoingInternships(actorID)
ongoingInternshipsList
displayOngoingInternships(ongoingInternshipsList)
selectInternship(internshipID)
fetchInternshipDetails(internshipID)
internshipDetails
displayInternshipDetails(internshipDetails)
initiateEndOfContract(internshipID)
validateEndOfContract(internshipID)
validationSuccess
submitTerminationReasons(reason, suggestions)
processTermination(internshipID, reason, suggestions)
terminationSuccess
notifyStakeholders(internshipID, terminationDetails)
notificationsSent
successMessage("Contract Terminated Successfully")
alt
[[Validation Error]]
validationError("Cannot terminate contract")
displayErrorMessage("Termination not allowed")
alt
[[Technical Error]]
errorMessage("Technical issue occurred, try again later")

Figure 18: End of contract

2.5 Component Interfaces

This section is a summary of all the methods that each component provides to the rest of the system, including names, return types, and required arguments.

2.5.1 WebApp Functions

- void viewNotifications()
- void selectNotification(interviewID: Integer)
- void navigateToViewStudents()
- void submitDecision(decision: String)
- void navigateToFeedbackAndSuggestions()
- void fetchFeedback(studentID: Integer, internshipID: Integer)
- void viewCurrentInternships()

- void closeCollaboration()
- void fetchInternshipDetails(studentID: Integer)
- void logIn(email: String, password: String)
- void registerStudent(firstName: String, lastName: String, email: String, password: String)
- void registerCompany(name: String, vatNumber: Integer, password: String)
- void registerUniversity(name: String, institutionCode: String, password: String)

2.5.2 Account Manager Functions

- void createAccount(userData: User)
- void getAccountDetails(userID: Integer)
- void updateAccountDetails(userID: Integer, newData: User)

2.5.3 CV Manager Functions

- void saveCV(userID: Integer, cvData: TextFile)
- void updateCV(userID: Integer, cvData: TextFile)
- void getCV(userID: Integer)
- void deleteCV(userID: Integer)

2.5.4 Recommendation Manager Functions

- void generateRecommendations(userID: Integer)
- void getRecommendations(userID: Integer)
- void updateRecommendations(userID: Integer)

2.5.5 Internship Manager Functions

- void createInternship(internship: Internship)
- void getInternshipDetails(internshipID: Integer)
- void updateInternshipDetails(internshipID: Integer, updatedData: Internship)
- void deleteInternship(internshipID: Integer)

2.5.6 Feedback Manager Functions

- void submitFeedback(userID: Integer, feedbackData: Feedback)
- Feedback fetchFeedback(studentID: Integer, internshipID: Integer)
- void updateFeedback(feedbackID: Integer, newData: Feedback)
- void deleteFeedback(feedbackID: Integer)

2.5.7 Notification Manager Functions

- void sendNotification(userID: Integer, notification: Notification)
- List<Notification> getNotifications(userID: Integer)
- void deleteNotification(notificationID: Integer)

2.5.8 Authentication Service Functions

- boolean authenticateUser(email: String, password: String)
- void registerStudent(studentData: Student)
- void registerCompany(companyData: Company)
- void registerUniversity(universityData: University)
- void resetPassword(userID: Integer, newPassword: String)

2.6 Selected Architectural Styles and Patterns

2.6.1 3-tier Architecture

Given that the S&C platform is a web-based application, the client-server architecture emerges as the optimal choice for its design. Specifically, we have adopted a 3-tier architecture that divides the application into three distinct layers: the client, the server, and the database. In this setup, the client is represented by the web browser, which handles user interaction; the server acts as the application server, processing business logic; and the database functions as the data server, managing and storing persistent data.

This architectural approach is particularly advantageous for our application because it provides a clear separation of concerns. By isolating the presentation layer (handled by the client) from the business logic (processed by the server) and the data management (executed by the database), the architecture enhances both maintainability and scalability. The modularity introduced by this separation not only simplifies updates and debugging but also facilitates the scaling of individual components independently, ensuring the application can efficiently handle increased demand or future extensions.

2.6.2 Microservices architecture

The system is structured following a microservice-based architectural approach, which ensures a high level of decoupling between components and supports exceptional scalability. This design divides the system into several independent services, each tailored to address a specific subset of requirements. By adopting this architecture, the need for synchronization among teams is significantly minimized, allowing smaller, more focused development groups to work efficiently. In addition, the codebase is broken into smaller, manageable units, simplifying processes such as development, testing, and debugging. This modularity not only enhances productivity, but also improves the maintainability and adaptability of the system over time.

2.6.3 API gateway

The API Gateway serves as an intermediary between users and the system, offering a unified interface for user interactions while regulating all traffic directed toward the micro-services. By leveraging this design pattern, the internal structure and organization of the system are hidden from the users, providing a seamless and simplified experience. Additionally, the API Gateway can function as a load balancer, efficiently distributing incoming requests across multiple machines delivering the same backend services. This approach enhances system reliability, performance, and scalability by centralizing traffic management and reducing complexity for end users.

2.7 Other Design Decisions

To ensure high availability of the system, the architecture has been designed with redundancy and fault tolerance in mind. The following measures have been implemented:

- **Load Balancers:** Multiple instances of critical services are deployed behind load balancers to distribute traffic evenly, preventing single points of failure and managing high traffic loads effectively.
- **Database Replication:** The relational databases, such as PostgreSQL, are set up with replication to maintain copies of the data. This ensures data availability even if one database node goes offline.
- **Micro-services Isolation:** Each service is deployed independently, and failures in one service (e.g., Notification Service) do not cascade to others, preserving the overall availability of the platform.
- **Auto-Scaling:** The system utilizes auto-scaling features to dynamically adjust the number of service instances based on demand, ensuring consistent performance even during peak usage.

By combining these strategies, the system provides consistent availability and ensures minimal downtime for users.

2.7.1 Notification Timing

The notification system is designed to deliver time-sensitive updates to users efficiently. Notifications are generated and managed using the Notification Service, which adheres to the following timing and delivery principles:

1. **Immediate Delivery:** Notifications related to critical events, such as application updates or interview schedules, are sent immediately after the event is triggered. This ensures that users are informed without delay.
2. **Scheduled Notifications:** For events such as application deadlines or reminders, notifications are queued and scheduled for delivery at a predefined time. The Notification Service manages these schedules efficiently to prevent performance bottlenecks.
3. **Retry Mechanism:** To ensure reliability, a retry mechanism is implemented for failed notification deliveries. For instance, if an email or push notification fails due to a temporary issue, the system retries the delivery after a set interval.
4. **User Preferences:** Users can configure their notification preferences (e.g., frequency, channels like email or push notifications). The system respects these preferences while scheduling and delivering notifications.
5. **Performance Considerations:** The Notification Service operates in an asynchronous manner, using message queues to decouple notification generation from delivery. This prevents delays in other system operations caused by notification handling.

By implementing these notification timing strategies, the system ensures timely and reliable communication with its users.

2.7.2 Data Storage

The database design for this system follows a logical representation of the relationships between various tables, as depicted in the diagram. It is essential to note that while all tables are presented in a unified logical view, they may be distributed across physically separate databases. This approach aligns with the system's micro-services architecture, enabling high decoupling and seamless scalability. Each micro-service has its specific database tailored to its functionality, ensuring optimal performance and easier maintenance.

The tables in the diagram are color-coded based on the physical databases they belong to:

- **Blue tables** (`User`, `Company`, `Student`, `University`, `Session`, `Notification`) are part of the account management database, storing foundational user and entity data in the Account Data Base.

- **Yellow tables** (`Feedback`, `FeedbackLogs`,) relate to feedback management, enabling companies and students to share structured feedback on internships.

- **Green tables** (`Recommendation`, `RecommendationLog`)are used for recommendation services, storing insights and matches between students and internships, with a dedicated log table to track actions like creation and updates.

- **Purple tables** (`Internship`, `InternshipApplication`, `IntenrshipLog`) focus on internship management, holding detailed information about posted opportunities, applications, and corresponding logs to track modifications and updates.

- **Red tables** (`CV Updates`) are part of the CV Data Base, dedicated to handling student profiles and tracking CV updates (e.g., URLs of updated CVs and reasons for updates).

Given the system's structured nature and the need for transactional integrity, a relational database model is employed. Specifically, PostgreSQL is utilized as the database management system. PostgreSQL is an open-source solution, widely recognized for its robustness and ACID compliance. ACID compliance ensures:

- Atomicity: Every transaction is executed completely or not at all.

- Consistency: Transactions bring the database from one valid state to another, maintaining its integrity.

- Isolation: Concurrent transactions execute without interference.

- Durability: Once a transaction is committed, its changes are permanent.

This architecture and database design ensure a scalable, high-performing system capable of handling user accounts, cv and internship postings, recommendations, and feedback effectively. By leveraging PostgreSQL, the system benefits from a stable and reliable foundation for managing critical data.

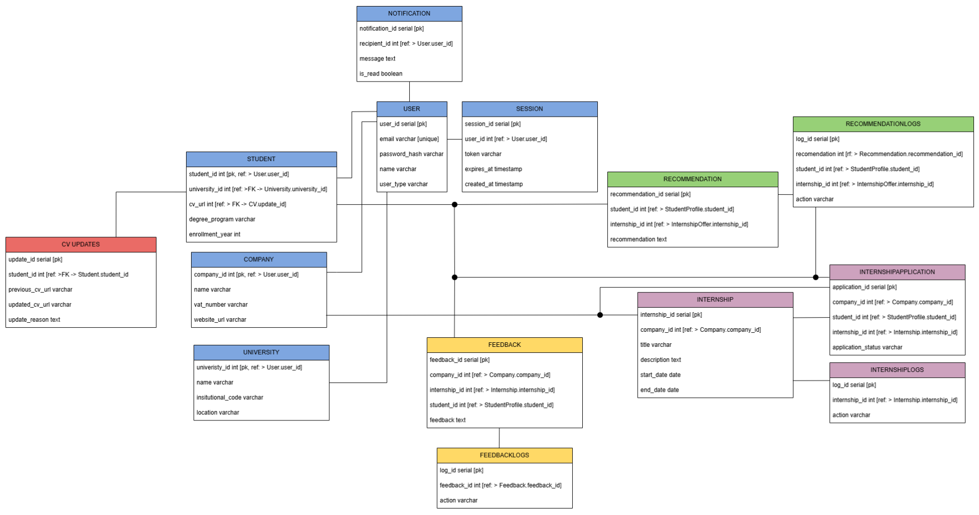

NOTIFICATION
notification_id serial [pk]
recipient_id int [ref: > User.user_id]
message text
is_read boolean
USER
user_id serial [pk]
email varchar [unique]
password_hash varchar
name varchar
user_type varchar
SESSION
session_id serial [pk]
user_id int [ref: > User.user_id]
token varchar
expires_at timestamp
created_at timestamp
STUDENT
student_id int [pk, ref: > User.user_id]
university_id int [ref: >FK -> University.university_id]
cv_url int [ref: > FK -> CV.update_id]
degree_program varchar
enrollment_year int
CV UPDATES
update_id serial [pk]
student_id int [ref: >FK -> Student.student_id
previous_cv_url varchar
updated_cv_url varchar
update_reason text
COMPANY
company_id int [pk, ref: > User.user_id]
name varchar
vat_number varchar
website_url varchar
UNIVERSITY
univeristy_id int [pk, ref: > User.user_id]
name varchar
insitutional_code varchar
location varchar
RECOMMENDATION
recommendation_id serial [pk]
student_id int [ref: > StudentProfile.student_id]
internship_id int [ref: > InternshipOffer.internship_id]
recommendation text
RECOMMENDATIONLOGS
log_id serial [pk]
recomendation int [rf: > Recommendation.recommendation_id]
student_id int [ref: > StudentProfile.student_id]
internship_id int [ref: > InternshipOffer.internship_id]
action varchar
INTERNSHIPAPPLICATION
application_id serial [pk]
company_id int [ref: > Company.company_id]
student_id int [ref: > StudentProfile.student_id]
internship_id int [ref: > Internship.internship_id]
application_status varchar
INTERNSHIP
internship_id serial [pk]
company_id int [ref: > Company.company_id]
title varchar
description text
start_date date
end_date date
INTERNSHIPLOGS
log_id serial [pk]
internship_id int [ref: > Internship.internship_id]
action varchar
FEEDBACK
feedback_id serial [pk]
company_id int [ref: > Company.company_id]
internship_id int [ref: > Internship.internship_id]
student_id int [ref: > StudentProfile.student_id]
feedback text
FEEDBACKLOGS
log_id serial [pk]
feedback_id int [ref: > Feedback.feedback_id]
action varchar

Figure 19: Platform Data Bases Structure

3 User Interface Design

In this section, we will describe the user interface design of the system. We will provide a mock-up of the main pages of the system and a description of the main functionalities. The user interface of the system is designed to be simple and intuitive. As the system is intended to be used with a desktop browser, the interfaces presented here are based on a desktop browser, but the interfaces are thought to be responsive and consequently usable also on mobile devices.

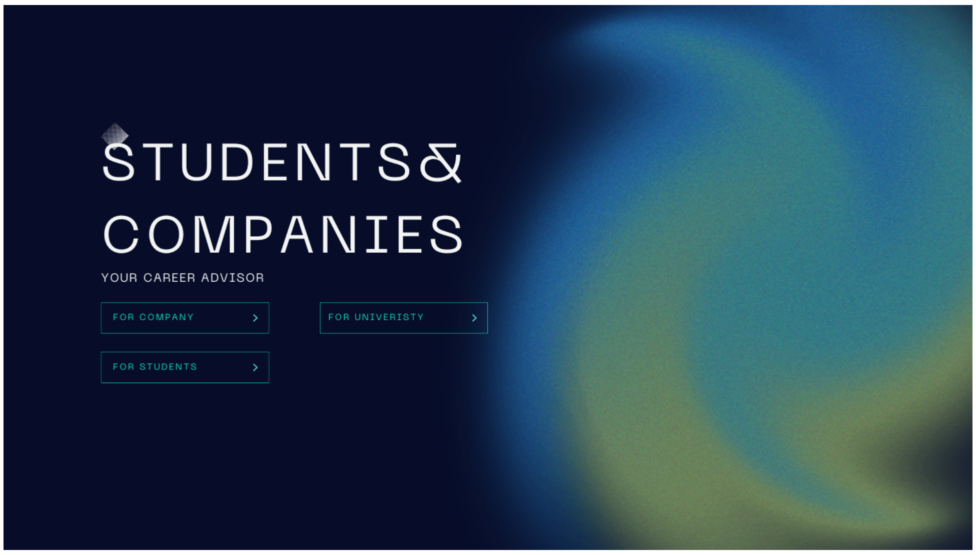


Figure 20: First User Interface

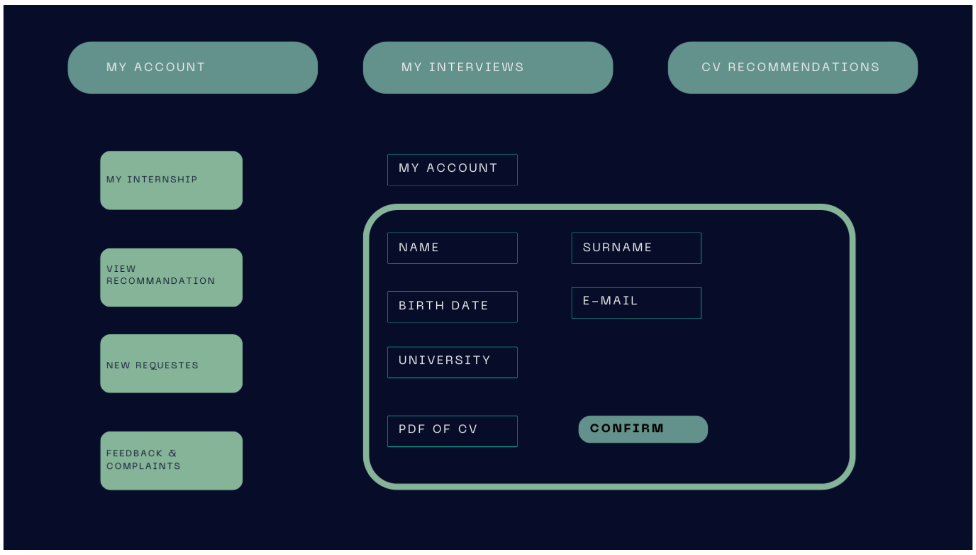


Figure 21: Student Interface: load CV

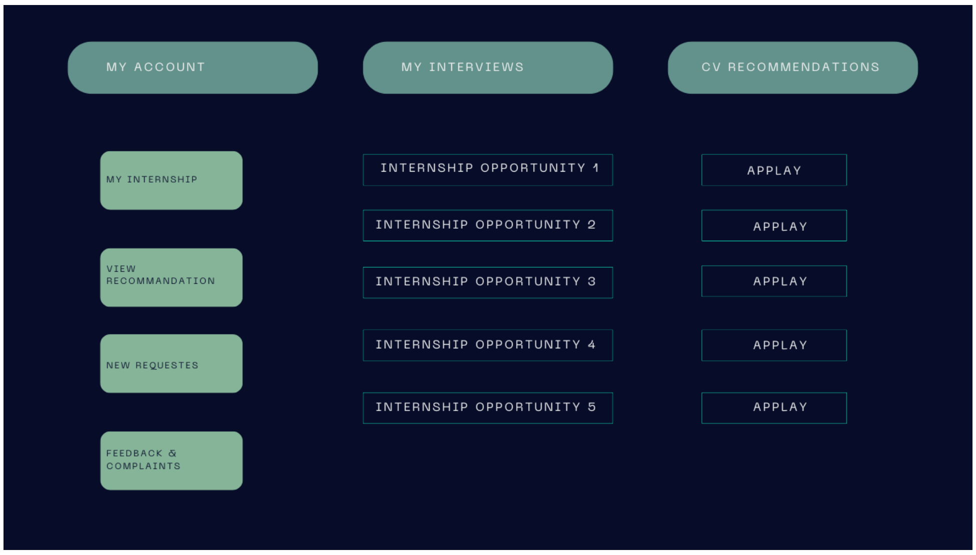


Figure 22: Student Interface: internship opportunities

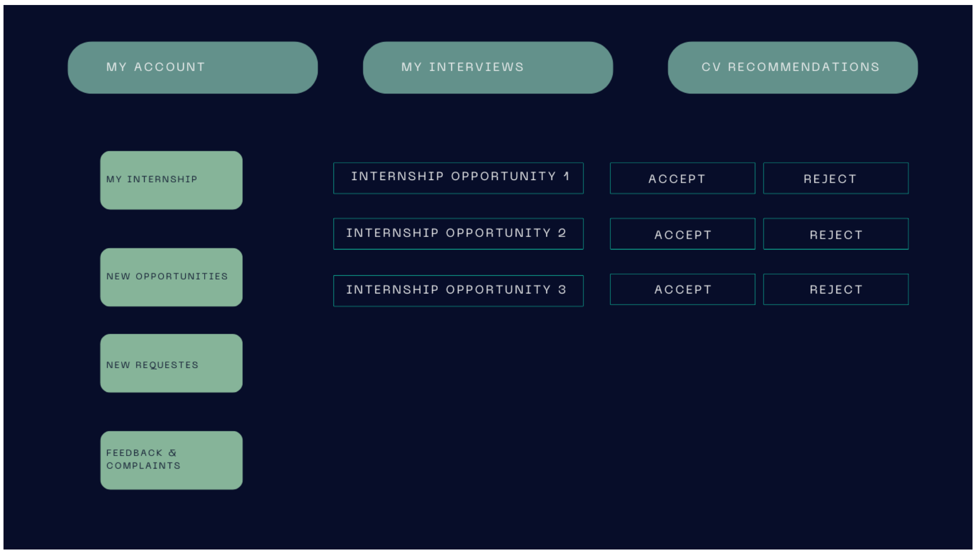


Figure 23: Student Interface: internship requests

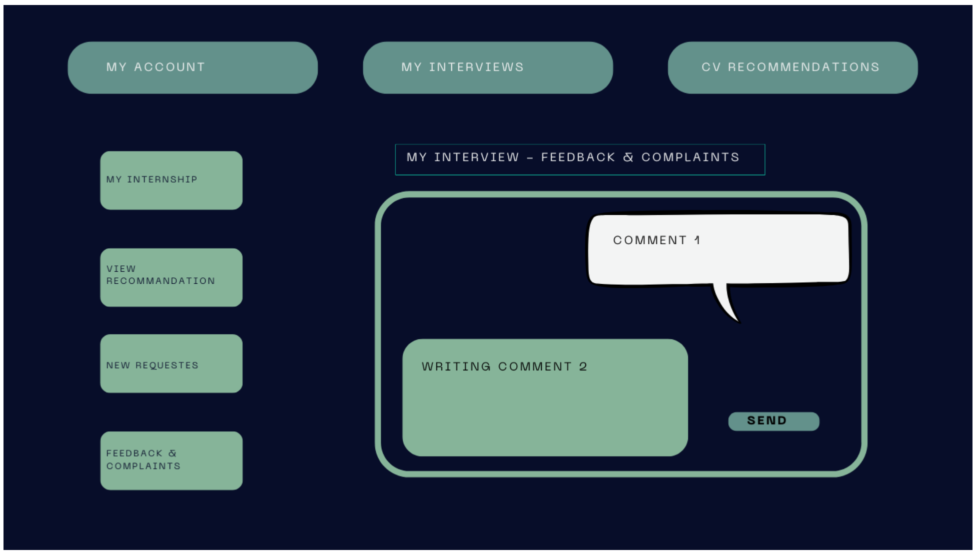


Figure 24: Student Interface: interview feedback&complaints

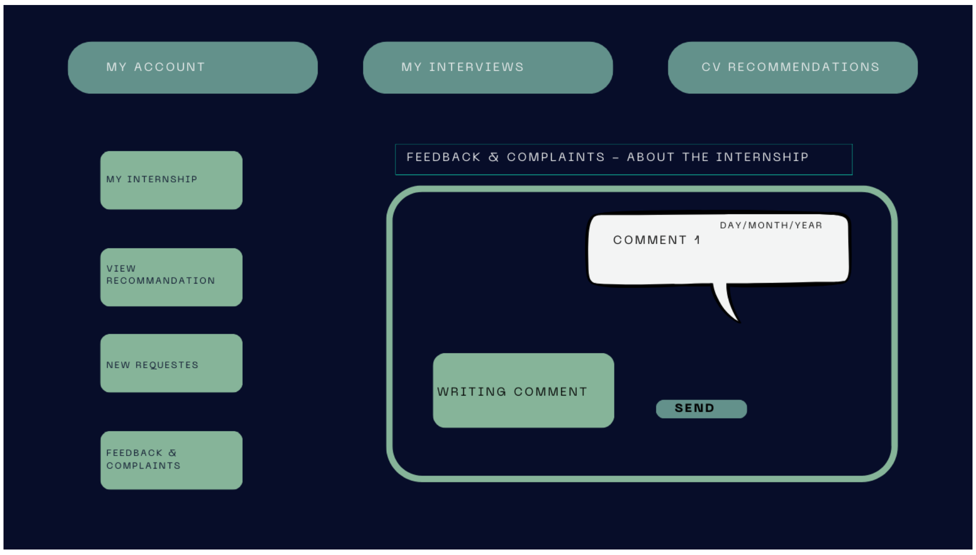


Figure 25: Student Interface: internship feedback&complaints

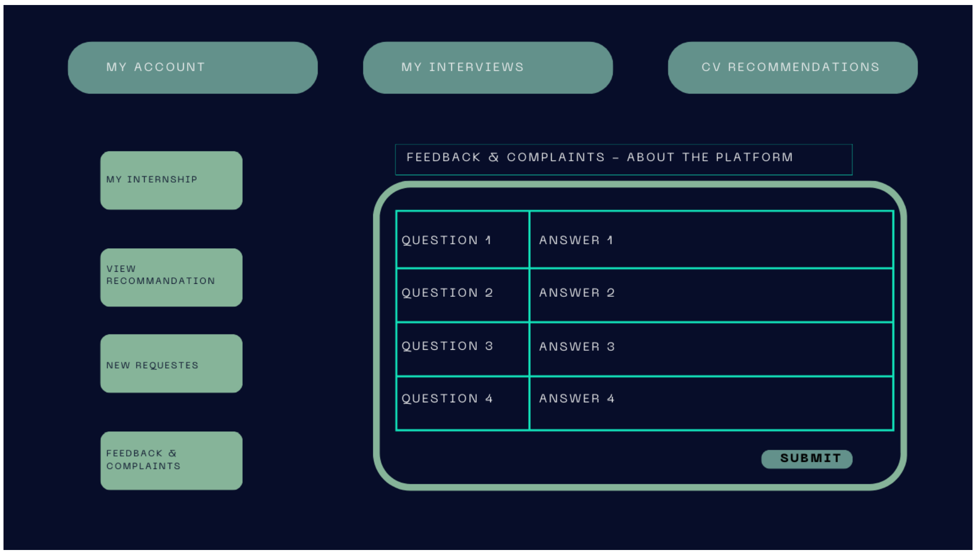


Figure 26: Student Interface: platform feedback&complaints

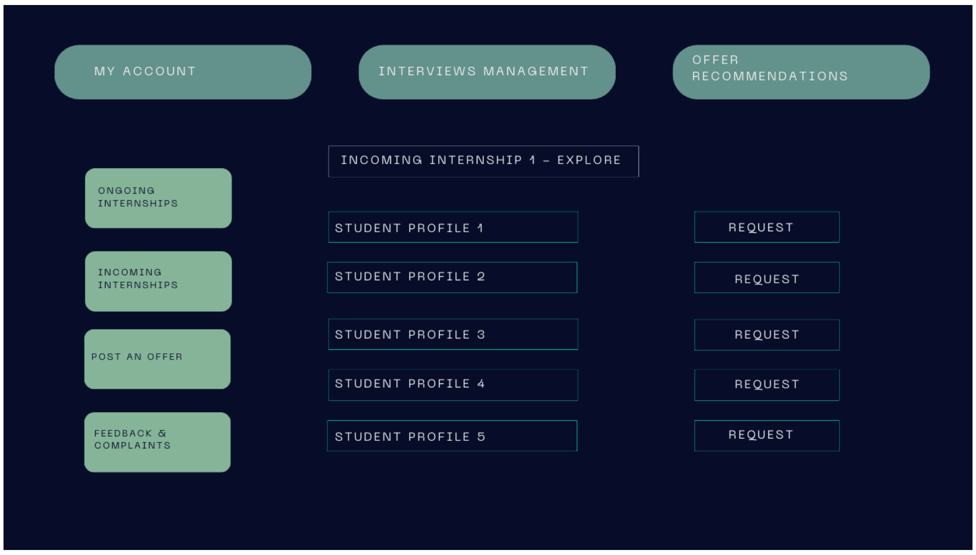


Figure 27: Company Interface: incoming internship recommendations

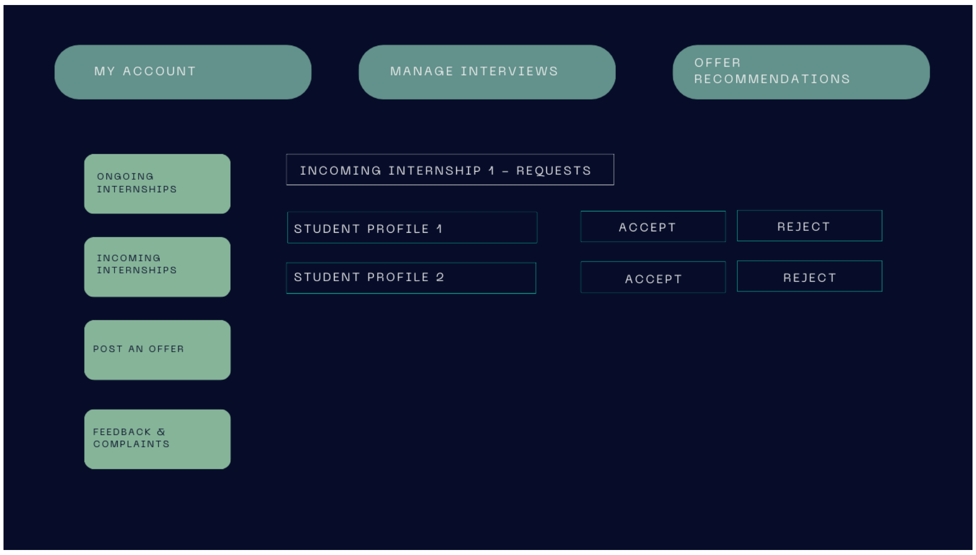


Figure 28: Company Interface: incoming internship requests

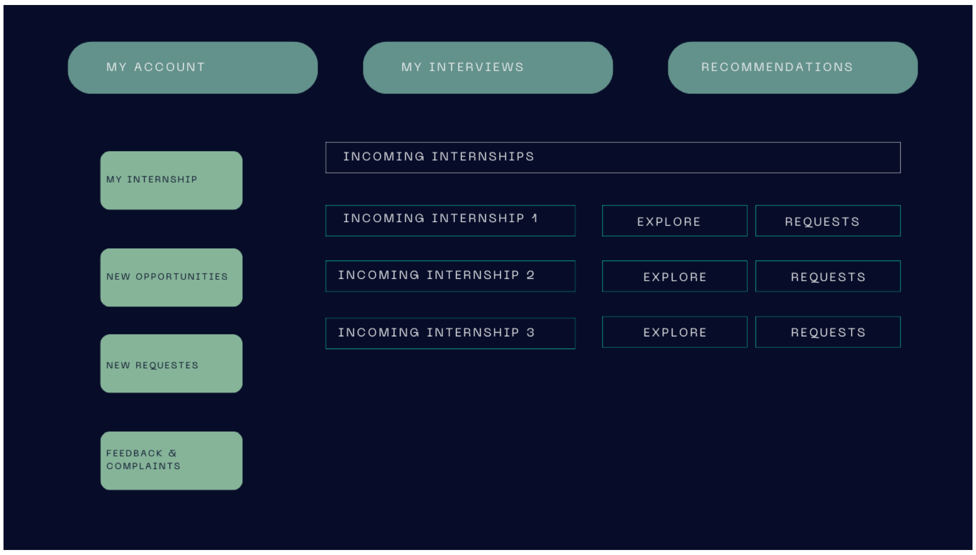


Figure 29: Company Interface: incoming internship

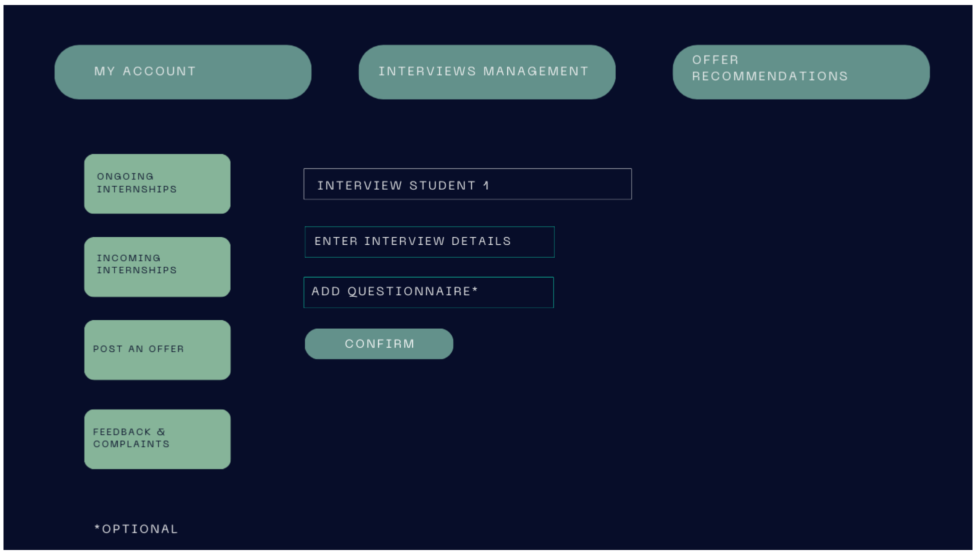


Figure 30: First Interview Management Interface

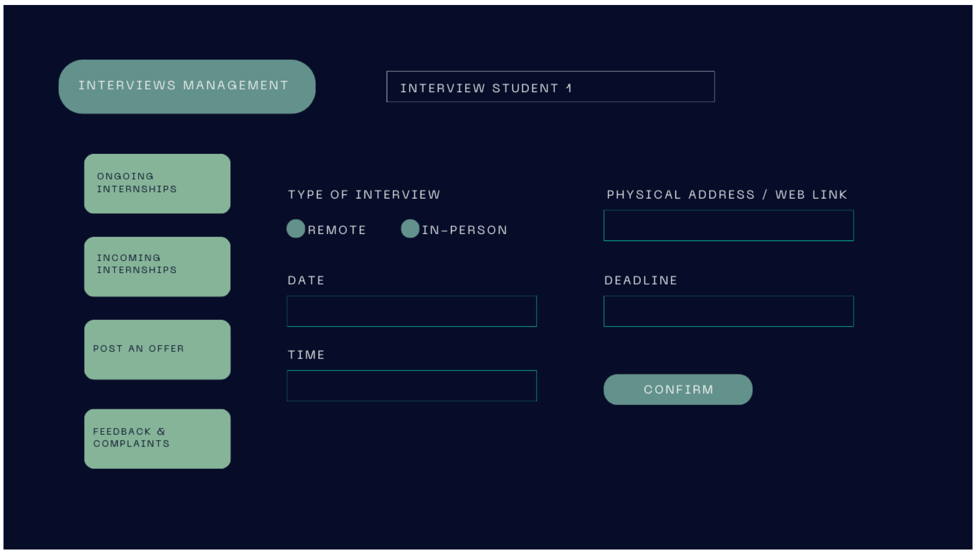


Figure 31: Second Interview Management Interface

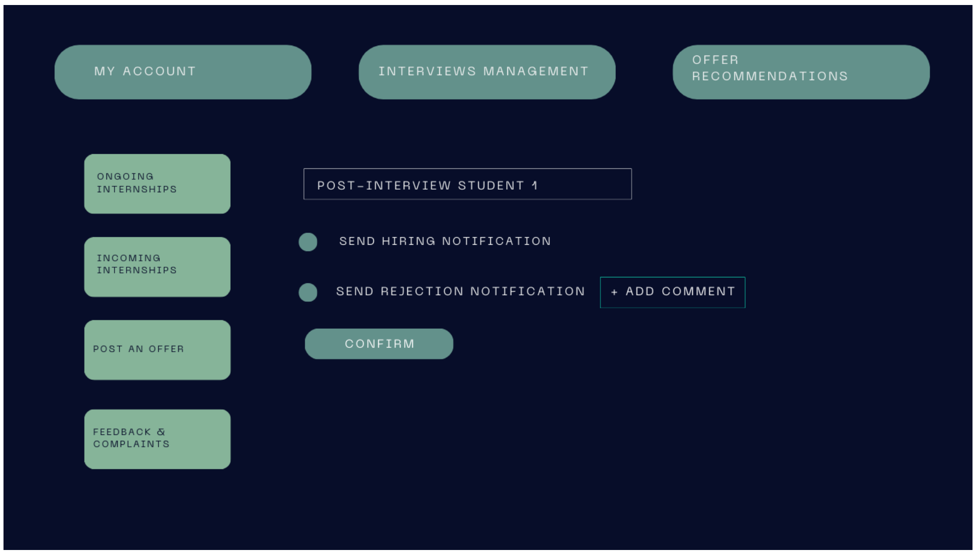


Figure 32: Post Interview Interface

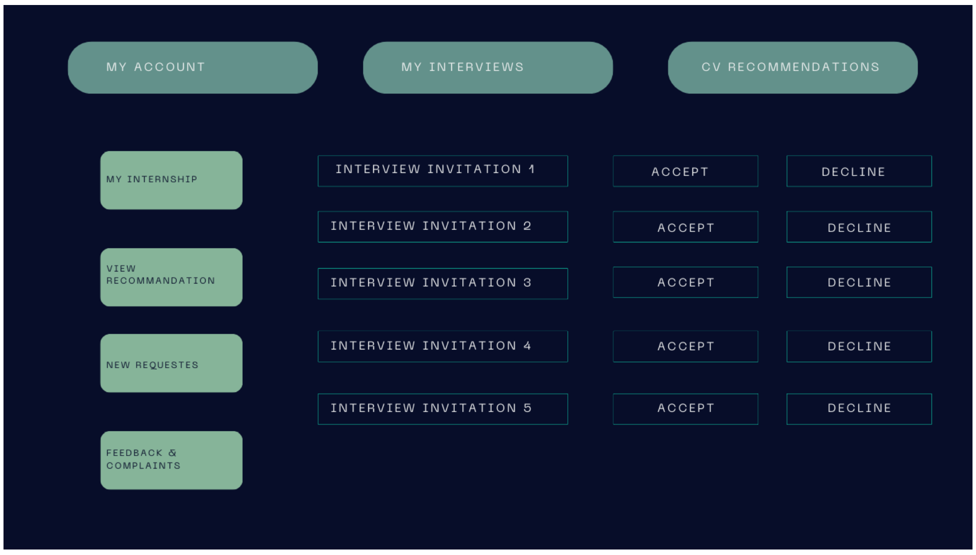


Figure 33: Interview Participation Interface

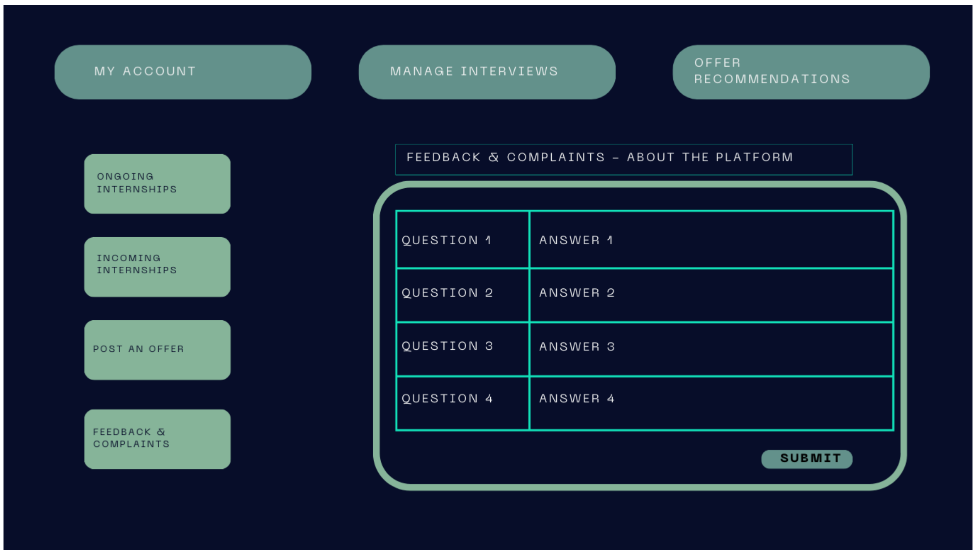
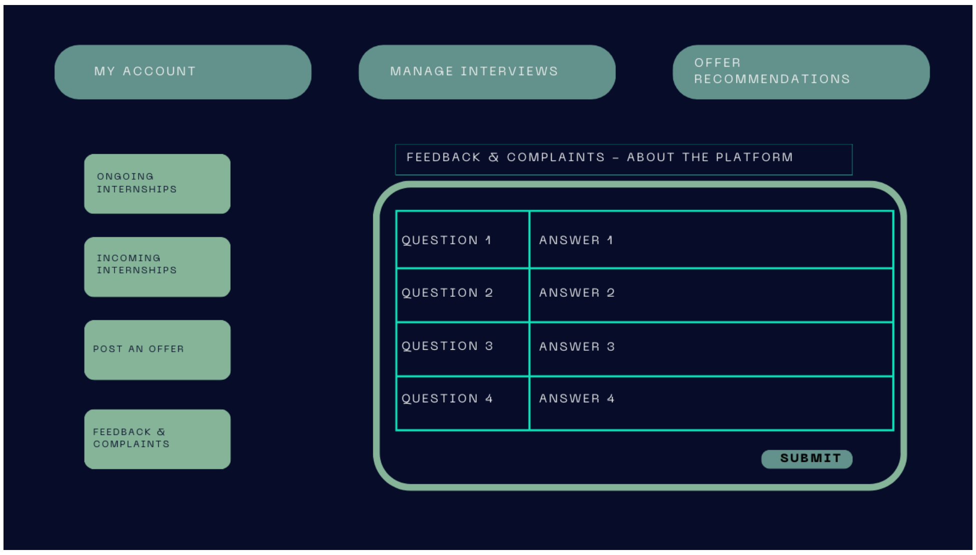


Figure 34: Company Interface: platform feedback&complaints



QUESTION 1	ANSWER 1
QUESTION 2	ANSWER 2
QUESTION 3	ANSWER 3
QUESTION 4	ANSWER 4

Figure 35: Company Interface: platform feedback&complaints

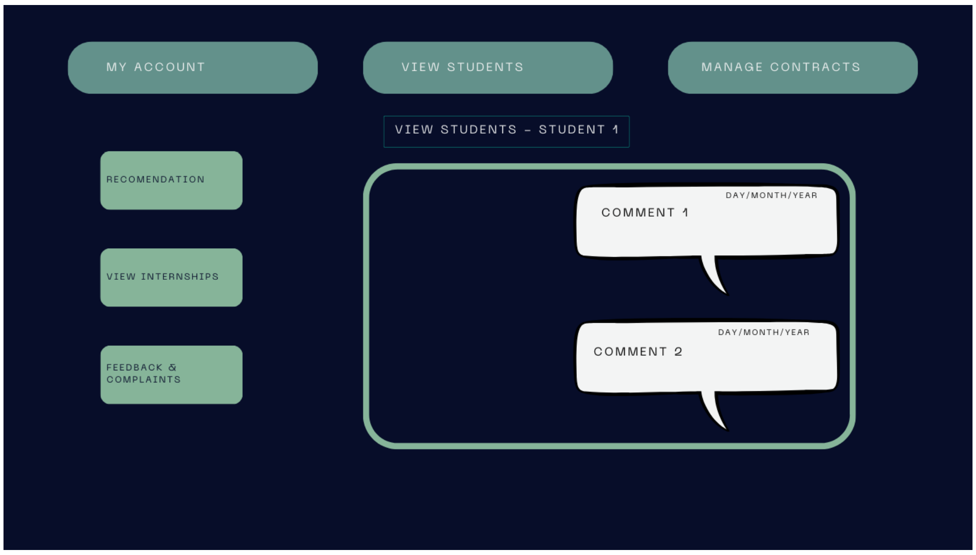


Figure 36: University Interface: view students f&c

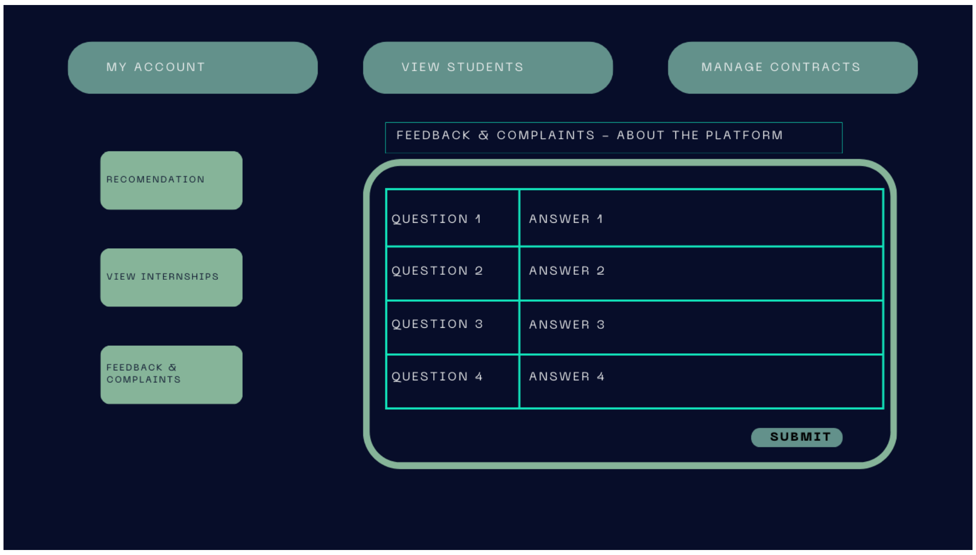


Figure 37: University Interface: platform feedback&complaints

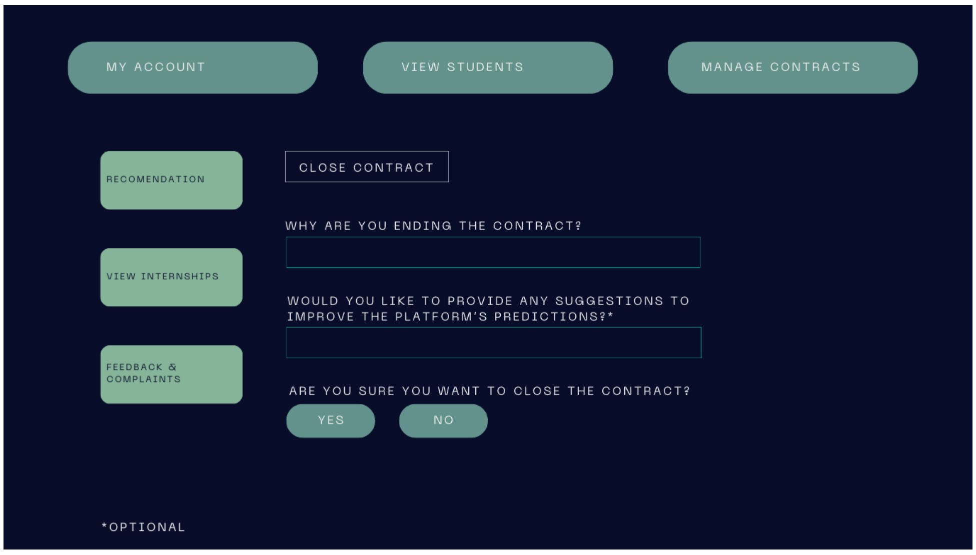

MY ACCOUNT
VIEW STUDENTS
MANAGE CONTRACTS
RECOMENDATION
VIEW INTERNSHIPS
FEEDBACK & COMPLAINTS
CLOSE CONTRACT
WHY ARE YOU ENDING THE CONTRACT?
WOULD YOU LIKE TO PROVIDE ANY SUGGESTIONS TO IMPROVE THE PLATFORM'S PREDICTIONS?*
ARE YOU SURE YOU WANT TO CLOSE THE CONTRACT?
YES
NO
*OPTIONAL

Figure 38: Close Contract Interface

4 Requirements Traceability

In this section of the document it is explained, for each requirement defined in the RASD, what design elements are involved for its fulfillment. In the following a series of tables maps the respective requirement.

Requirements:	**[R1] The system allows users to sign up.** **[R2] The system allows users to log in.** **[R3] The system allows users to log out.** **[R4] The system allows users to reset their password.** **[R34] The system verifies the veracity of the information entered by students, companies and universities who register.**
Components:	• WebAppUI • Application server: – Authentication Service – Account Manager – CV Manager • DBMS: – Account DB – CV DB • University Platform

Table 1: System Requirement and Components for user account management

Requirements:	**[R6] The system allows users to provide feedback about the platform.**
Components:	• WebAppUI • Application server: – Feedback Manager • DBMS: – Feedback DB

Table 2: System Requirement and Components for platform feedback

Requirements:	**[R7] The system allows users to delete their own account.**

Components:	• WebAppUI
	• Application server:
	  – Account Manager
	  – CV Manager
	• DBMS:
	  – Account DB
	  – CV DB

Table 3: System Requirement and Components to delete a user account

Requirements:	**[R8] The system allows students to upload and manage their CVs.** **[R9] The system provides suggestions to students for improving their CVs.** **[R18] The system allows students to edit their CV.** **[R35] The system supports the recommendation process by using statistical analyses to match students and internships.** **[R36] The system collects feedback from students, companies and universities to improve the recommendation mechanisms and optimize the system.**

Components:	• WebAppUI
	• Application server:
	  – Account Manager
	  – Recommendation Manager
	  – CV Manager
	• DBMS:
	  – Account DB
	  – Recommendation DB
	  – CV DB

Table 4: System Requirement and Components to manage Student’s CV

Requirements:	**[R19] The system allows companies to manage internship offers.** **[R20] The system provides suggestions to companies for refining their internship descriptions.** **[R21] The system allows companies to edit their internship descriptions.**

Components:	• WebAppUI
	• Application server:
	  – Recommendation Manager
	  – Recommendation Service
	• DBMS:
	  – Internship DB
	  – Recommendation DB

Table 5: System Requirement and Components to manage internship's offer of Company

Requirements:	**[R5] The system sends automatic notifications to update all parties on relevant changes.** **[R10] The system allows students to interrupt an internship with a company at any time.** **[R17] The system allows students to provide feedback about an internship.** **[R30] The system allows companies to interrupt an internship with a student at any time.** **[R31] The system allows universities to monitor the status of ongoing internships.** **[R32] The system allows universities to read student feedback on an internship.** **[R33] The system allows universities to interrupt an internship in case of complaints and negative feedback from students.**
Components:	• WebAppUI • Application server: – Account Manager – Internship Manager – Feedback Manager – Notification Manager • DBMS: – Account DB – Internships DB – Feedback BD • University Platform

Table 6: System Requirement and Components to manage internship

Requirements:	**[R11] The system allows students to browse available internships.**
Components:	• WebAppUI • Application server: – Internship Manager • DBMS: – Internships DB

Table 7: System Requirement and Components to browse internships

Requirements:	**[R12] The system suggests internships to students based on their expertise and interests.** **[R22] The system informs companies when candidates that match their needs are available.** **[R23] The system allows companies to search for students for an internship independently.** **[R35] The system supports the recommendation process by using statistical analyses to match students and internships.**
Components:	• WebAppUI • Application server: – Internship Manager – Recommendation Manager – CV Manager • DBMS: – Recommendation DB – Internships DB – CV DB

Table 8: System Requirement and Components to suggest internships or students

Requirements:	**[R13] The system allows students to accept, refuse or request the reorganization of interviews proposed by companies.** **[R14] The system allows students to request contact with a company for an internship.** **[R15] The system allows students to view and complete questionnaires provided by companies.** **[R16] The system allows students to accept or reject an internship offer.** **[R24] The system allows companies to request contact with a student for an internship.** **[R25] The system allows companies to provide questionnaires to students.** **[R26] The system allows companies to review completed questionnaires from students.** **[R27] The system allows companies to schedule and manage interviews with students.** **[R28] The system allows companies to monitor the progress of internships.** **[R29] The system allows companies to accept or reject a student for an internship.** **[R37] The system creates contacts between students and companies after mutual acceptance.** **[R38] The system starts and manages the internship selection process.**
Components:	• WebAppUI • Application server: – Account Manager – Internship Manager – Notification Manager • DBMS: – Account DB – Internships DB

Table 9: System Requirement and Components to manage interviews and selection process

Requirements:	**[R35] The system supports the recommendation process by using statistical analyses to match students and internships.** **[R36] The system collects feedback from students, companies and universities to improve the recommendation mechanisms and optimize the system.**
Components:	• WebAppUI • Application server: – Recommendation Service – Recommendation Manager – Feedback Manager – Account Manager – CV Manager • DBMS: – Recommendation DB – Feedback DB – Account DB – CV DB

Table 10: System Requirement and Components for recommendation process and feedback collection

5 Implementation, Integration and Test Plan

The implementation, integration and test plan will follow a bottom-up approach, starting from the components with no dependencies and then integrating them together.

5.1 Implementation Plan

Overview The S&C platform will be developed following a modular approach, using 3-tier architecture combined with a micro-services-based structure. The micro-services architecture facilitates the parallel implementation and testing of the various services by different development teams, significantly reducing dependencies between teams. Each service is developed as an independent and autonomous unit, while interactions between services can be simulated to ensure compatibility and correct integration already in the early stages.

Main Components

1. **Client (Presentation Layer)**: developed using modern front-end technologies (React.js for managing the UI) and manages interaction with the user and communicates with API Gateway to retrieve or send data.

2. **Server (Business Logic Layer)**: made up for independent microservices, developed with Node.js and organized by functional domains. Each microservice handles a specific area. such as authentication, user data management and notifications.

3. **Database (Data Layer)**: data is managed through a relational database (PostgreSQL) for structured data and a NoSQL database (MongoDB) for unstructured content.

4. **API Gateway**: built with a framework like Kong or Express.js to centralize access to microservices, it includes load balancing and request management features.

Implementation Time Plan

Phase	**Task**
Step 1: Setup	Environment Configuration
Step 2: Backend	Microservices Implementation
Step 3: Frontend	User Interface Development
Step 4: API GW	API Gateway Integration
Step 5: Testing	Final verification and validation

Table 11: Implementation Time Plan

5.2 Integration Plan

Integration Strategy The strategy adopted is incremental integration, in which components are developed and tested individually before being integrated into a complete system. During development, interactions between components are simulated to ensure correct operation before full integration.

1. **Step 1**: Test individual micro-services using mock tools to simulate interactions.

2. **Step 2**: Integration between API Gateway and micro-services, with testing of public APIs.

3. **Step 3**: Front-end integration with the API Gateway to verify correct client-server communication.

4. **Step 4**: End-to-end testing of the complete system.

5.3 Test Plan

Testing Strategy The testing plan follows a hierarchical and systematic process:

1. **Unit Test**: Testing individual code units within micro-services to validate their correctness and functionality.
   Tool: Jest (for Node.js)

2. **Integration Test**: Verifying communication between micro-services via the API Gateway.
   Tool: Postman/Newman.

3. **System Test**: Test the complete system, verifying consistency and compliance with requirements.
   Tool: Selenium for automated UI testing.

4. **Performance Test**: Performance measurement under load.
   Tool: Apache JMeter.

During the testing phase, each service is treated as a self-contained unit, rigorously tested to ensure it meets its intended functions. Next, services are integrated to enable end-to-end testing of inter-service interactions without relying solely on simulations.

Test Case Examples

ID Test	**Description**	**Input**	**Expected Result**
T001	User Login	Correct credentials	Access to the system with valid token
T002	New user registration	Valid user data	Account creation and email confirmation
T003	Loading dashboard data	Request via API	Correct data display
T004	API load testing	1000 requests/minute	Response time < 1 second

Table 12: Test Case Examples

Acceptance Criteria

- All tests must pass with a 100% pass rate.
- Performance must meet the minimum requirements defined in the project KPIs.

6 Effort Spent

Discuss the amount of effort and contributions made by each member of the team for document creation and analysis.

Members of group	**Effort spent (hours)**	
Chiara Barone	Introduction	*3h*
	Architectural design	*10h*
	User interface design	*6h*
	Requirements traceability	*6h*
	Implementation, integration and test plan	*1h*
	Reasoning	*2h*
Ottavia Biagi	Introduction	*1h*
	Architectural design	*7h*
	User interface design	*7h*
	Requirements traceability	*10h*
	Implementation, integration and test plan	*3h*
	Reasoning	*10h*
Myriam Caravaggio	Introduction	*8h*
	Architectural design	*5h*
	User interface design	*6h*
	Requirements traceability	*2h*
	Implementation, integration and test plan	*5h*
	Reasoning	*2h*

Table 13: Effort spent by each member of the group

7 References

- **Draw.io** to draw the component diagram and the deployment view.
- **Sequencediagram.org** to draw the sequence diagrams.
- **GitHub** to share and collaborate on the project.
- **Overleaf** for writing this document.
- **Canva** to draw the mock-ups.