POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering II

Document: **Design Document - Students & Companies**

Author(s): **APPOURCHAUX Léo 11078414**
YE Yumeng 11072406

Advisor: **Prof. DI NITTO ELISABETTA**

Academic Year: 2024-2025

Contents

1 | Introduction

1.1. Purpose

In today's competitive job market, students often face challenges transitioning from university to workplace, struggling to find opportunities that match their skills and career aspirations. At the same time, companies invest significant time and resources in identifying the right candidates. The Students & Companies (S&C) platform addresses these challenges by connecting students, companies and universities in an unified ecosystem. Students can discover internships tailored to their skills and career goals, while companies gain access to a pool of qualified candidates. Universities, on the other hand, can monitor internship progress to ensure educational standards are respected. The platform goes beyond simply matching students and companies. It supports the recruitment process with tools for interviews, tests and feedback. Personalized suggestions and recommendations generated by intelligent algorithms help students enhance their profiles and companies improve their internship offers. The ultimate purpose of the platform is to create a practical, engaging and efficient way for students to gain meaningful experiences, for companies to find the right talent and for universities to maintain high education standards.

1.2. Scope

As mentioned in the RASD, there are three primary users that interact with the system : Students, Companies and Universities.

- Students can create accounts, upload their CVs and search for internship opportunities that align with their skills and career goals. They can apply to internships directly through the platform and participate in recruitment processes, including tests and interviews. Moreover, students receive personalized suggestions to improve their profiles and recommendations for internships based on their qualifications and preferences. They can also provide feedback on the recruitment process or submit complaints when necessary.

- Companies can post detailed internship offers, specifying the required qualifications, skills and benefits.They can manage applications, evaluate candidates through tests or interviews, and provide feedback on student performance. Companies also receive recommendations for suitable candidates and actionable suggestions to enhance their internship offers. Moreover, they can submit complaints or report issues regarding the recruitment process.
- Universities have tools to monitor the internship process, ensuring alignment with educational standards and addressing complaints raised by students or companies. They are responsible for resolving conflicts and maintaining oversight of the internships to ensure compliance with academic and professional guidelines.

The platform acts as a mediator between these users, facilitating efficient communication, seamless recruitment processes and personalized experiences. The system integrates intelligent algorithms to match students with suitable internships and companies with qualified candidates. It also enables real-time notifications, feedback analysis and structured tools for managing every step of the internship journey - from application to final selection.

1.3. Definitions, Acronyms, Abbreviations

- S&C: Students&Companies
- CV: Curriculum Vitae
- UI: User Interface
- API: Application Programming Interface
- DBMS: Database Management System
- SLA: Service Level Agreement
- 2FA: Two-Factor Authentication
- GDPR: General Data Protection Regulation

1.4. Revision History

- Version 1 (22/12/2024)

1.5. Reference Documents

The document is based on the following materials:

- The specification of the RASD and DD assignment of the Software Engineering II course a.a. 2024/25
- Slides of the course on WeBeep

1.6. Document Structure

1. Introduction: it aims to give a brief description of the project. In particular it's focused on the reasons and the goals that are going to be achieved with its development, as previously explained in the RASD;
2. Architectural Design: : This part of the document describes at different levels of granularity how the architecture is designed;
3. User Interface Design: This section shows the wire frame that constitutes the platform UI;
4. Requirements Traceability: in this section it is explained how the goals defined in the previous document are satisfied by the interaction between the requirements and the component described in this document.
5. Implementation, Integration and Test Plan: this section provides a description of the implementation integration and testing details, useful for the developers and producer of the platform;
6. Effort Spent: shows the effort spent to write this document
7. Bibliography: it contains the references to any documents and software used to write this paper.

2 | Architectural Design

2.1. Overview

Our Students & Companies (S&C) platform will use a three-tier architecture to ensure scalability, modularity and ease of maintenance. This architecture divides the system into three distinct layers:

1. Presentation Layer:
    - Contains the web-based user interface, which is tailored for three primary users: Students, Companies, and Universities
    - Ensures intuitive navigation, responsive design and device compatibility
    - Uses technologies like HTML5 and JavaScript frameworks like React
2. Application Layer:
    - Hosts the business logic, including core functionalities like CV processing, recruitment workflows, and recommendation algorithms
    - Implements RESTful APIs for communication with the presentation layer and data layer
    - Supports modularity by encapsulating services such as user authentication, feedback collection, and complaint resolution
    - Technologies such as Node.js or Django can be utilized for server-side processing
3. Data Layer
    - Manages data storage and retrieval through databases:
        (a) Relational databases (e.g., MySQL) for structured data like user profiles, internship offers, and application logs
        (b) NoSQL databases (e.g., MongoDB) for unstructured data such as feedback comments or CV attachments

- Ensures data integrity, security, and redundancy through replication mechanisms.

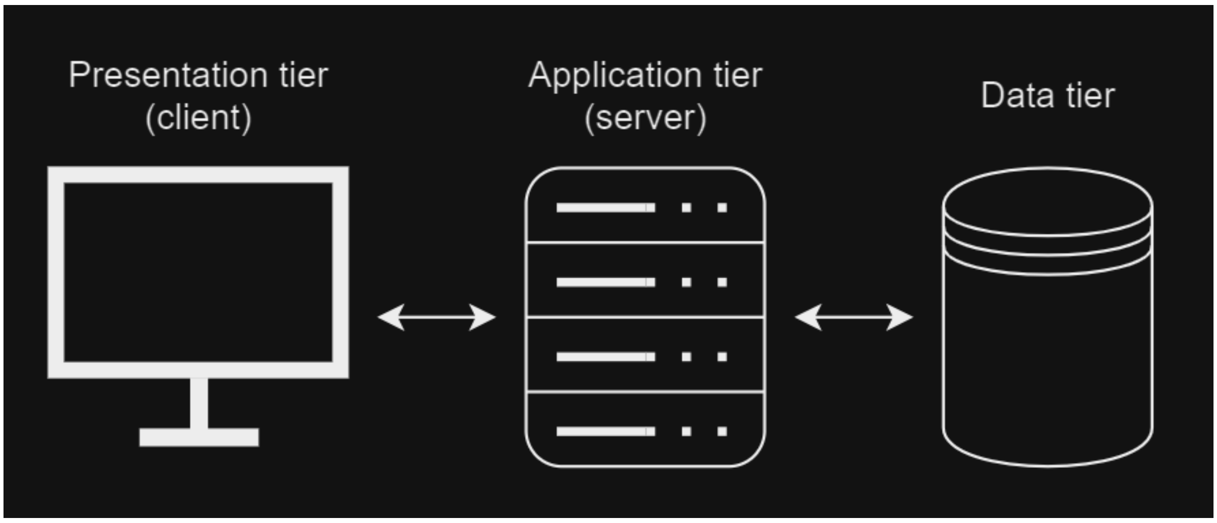


Figure 2.1: Overview

2.1.1. Distributed View

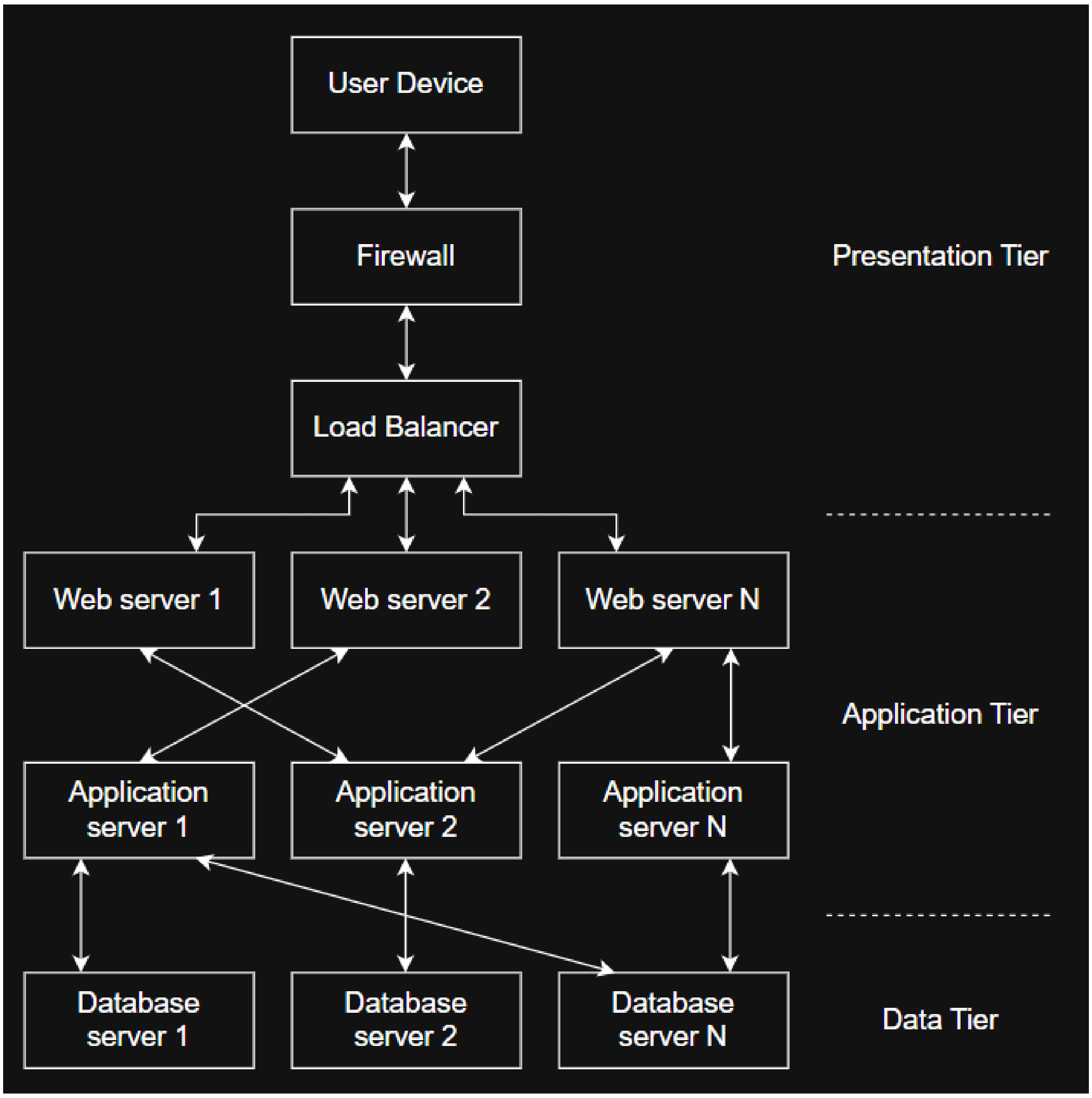


Figure 2.2: Distributed View

2.2. Component view

For the three tier architecture that we will use, the backend will contain an API which will allow the web app to communicate and query the back-end. The backend will also interact with the data layer for storage and retrieval. The main components of the backend are:

- AuthenticationManager: provides the functionalities related to the authentication of the user on the platform

- CVManager: provides the functionalities related to the management of student CVs
- ApplicationManager: provides the functionalities related to the management of student applications
- OfferManager: provides the functionalities related to the management of company offers
- MatchingAlgorithm: takes CVs, offers and feedback to compute potential matches to companies and students, then sends reccomendations to users in the form of a notification
- ComplaintManager: provides the functionalities related to the management of user complaints and discusions
- SuggestionManager: periodicaly sends suggestion on how to improve CVs and Offers to Student and Users
- RecruitingManager: encompasses all of the recruiting process, including interviews, questionnaires, and gathering of answers.
- MonitoringManager: allows the monitoring of the recruiting process to users
- Database: stores all the data of the app
- API: This allows the backend to be called by the frontend, and routes the queries to the appropriate internal components.

The main component of the frontend is:

- WebApp: this is the interface between the app and the user. It contains not only the graphics but also the logic needed to interact efficiently with the API of the backend

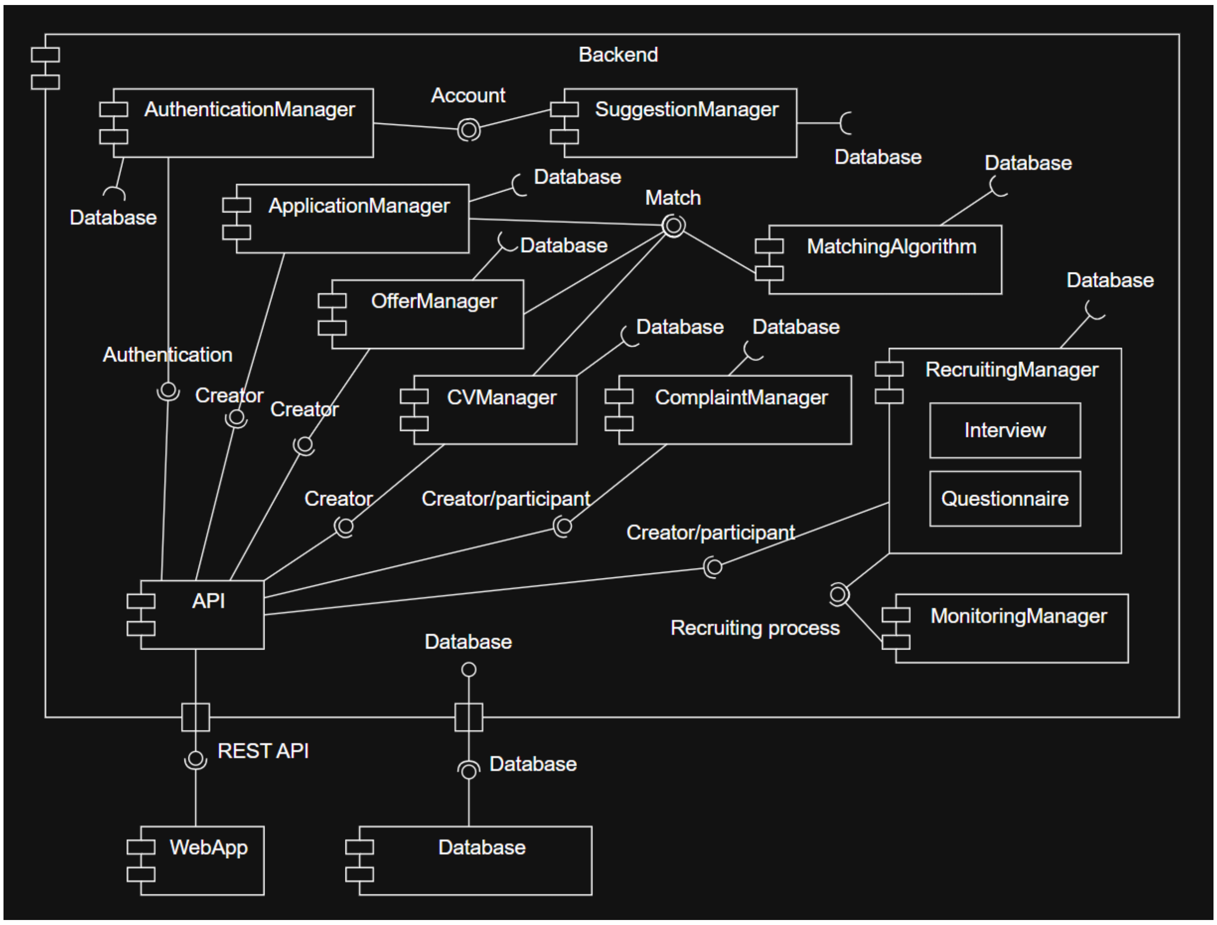


Figure 2.3: Component view

2.3. Deployment view

Here is the deployement showing the topological distribution of the S&C platform.

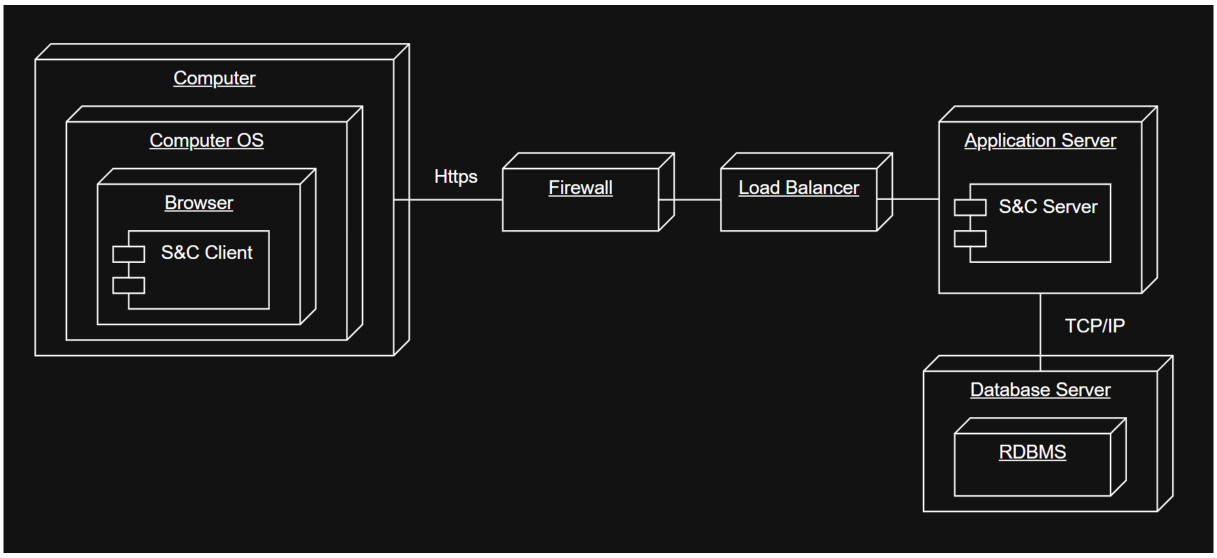


Figure 2.4: Deployment View

The operating system (OS) runs on the computer, the Browser runs inside of the OS, and the S&C client runs inside of the browse. When a request is sent to the backend, it goes through a firewall to filter out eventual malicious requests. The request then goes through a load balancer which redirect trafic to prevent overload. The application server (backend) communicates with the database via TCP/IP.

2.4. Runtime View

[UC1] User Registration

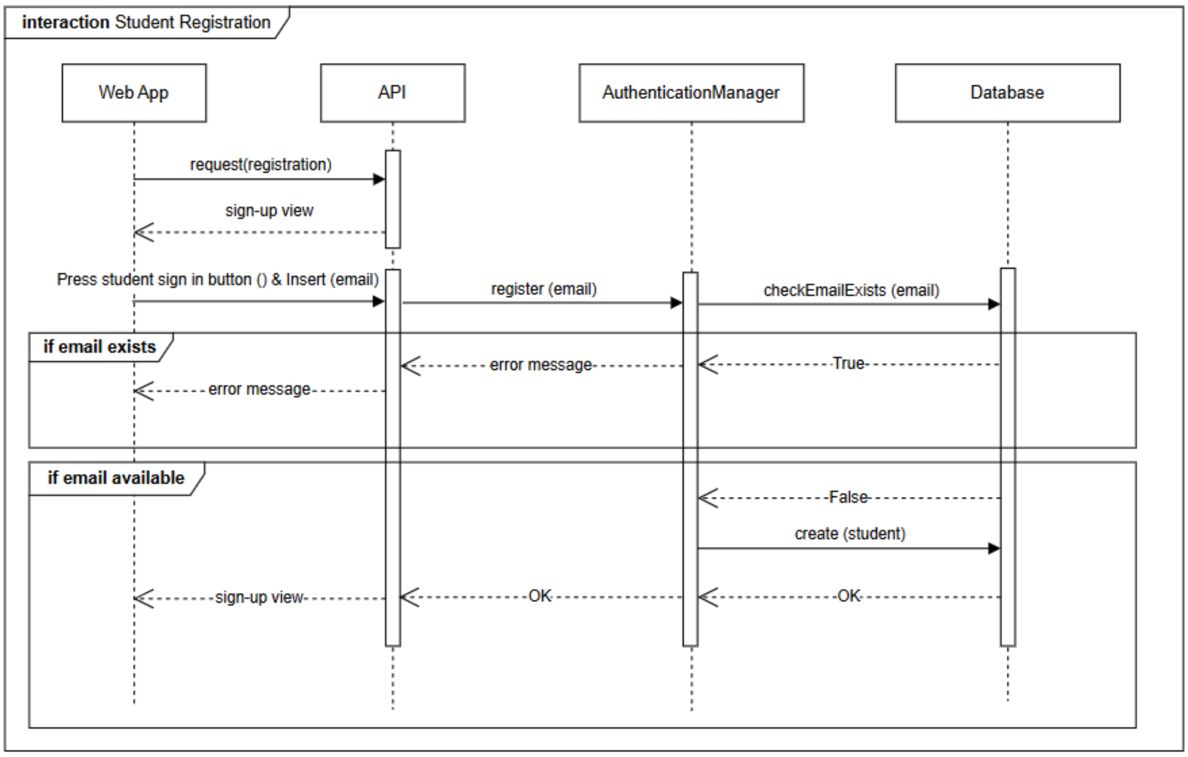


Figure 2.5: User Registration

[UC2] User Login

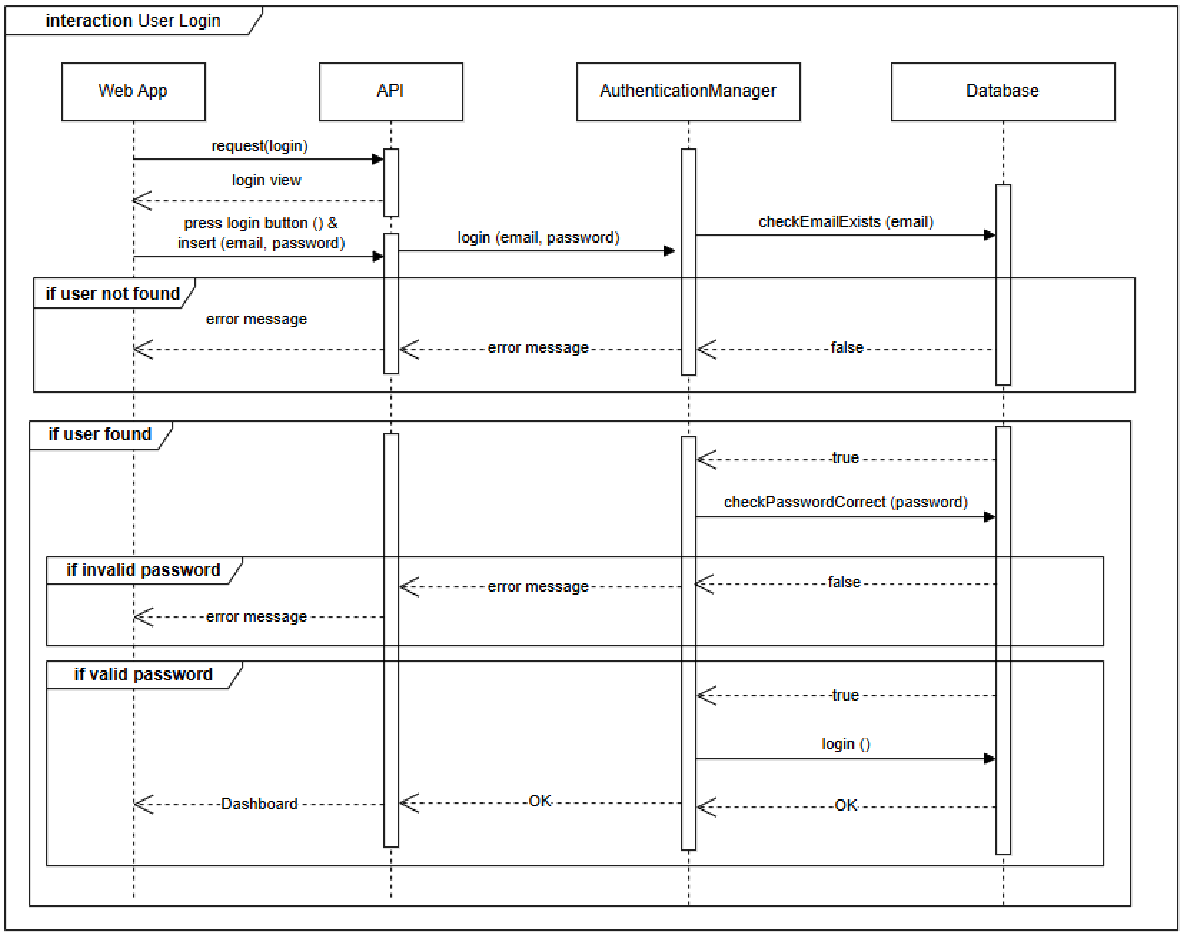


Figure 2.6: User Login

[UC3] Create Internship Offer

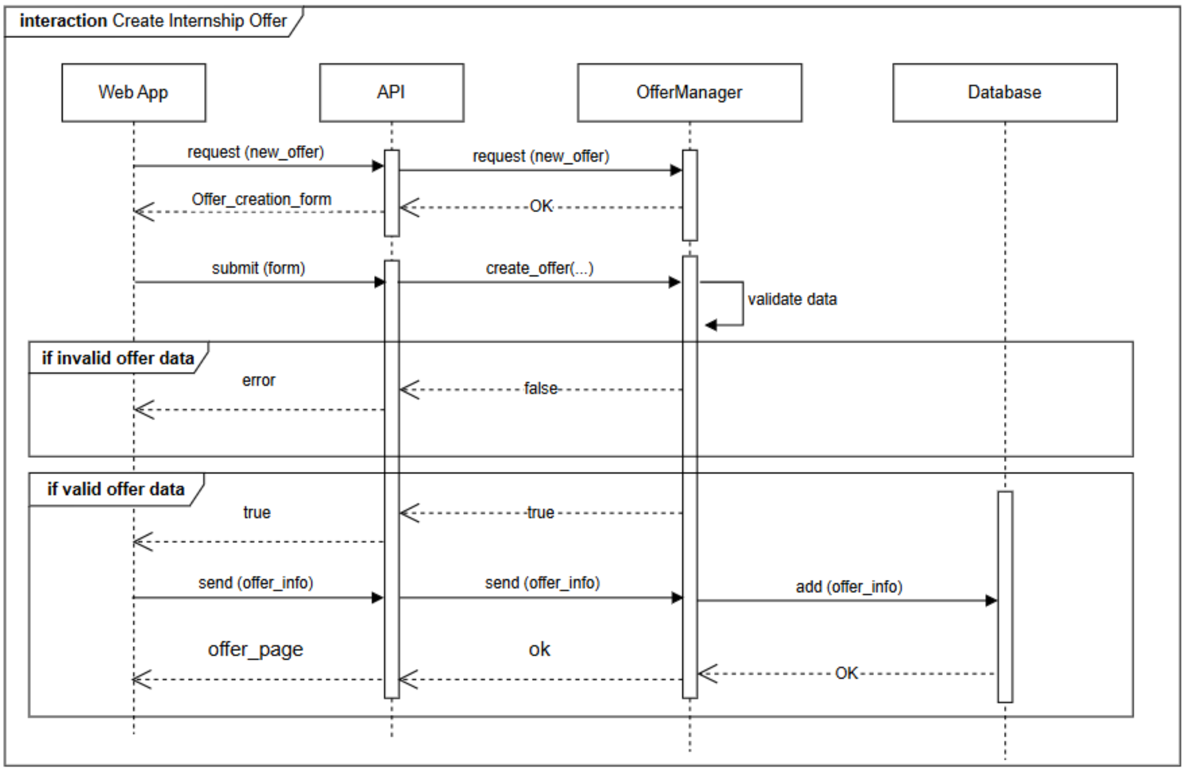


Figure 2.7: Create Internship Offer

[UC4] Modify Internship offer

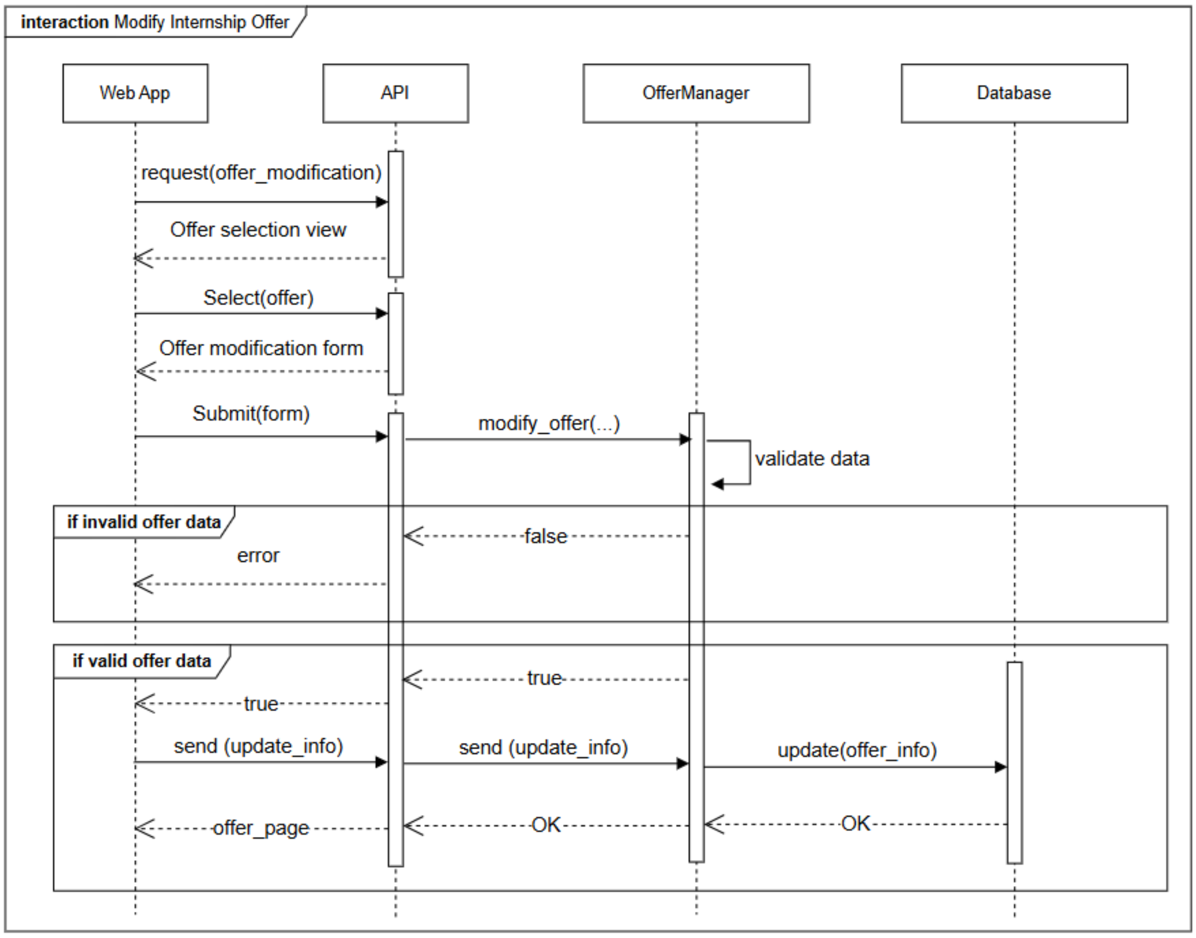


Figure 2.8: Modify Internship offer

[UC5] Delete Internship offer

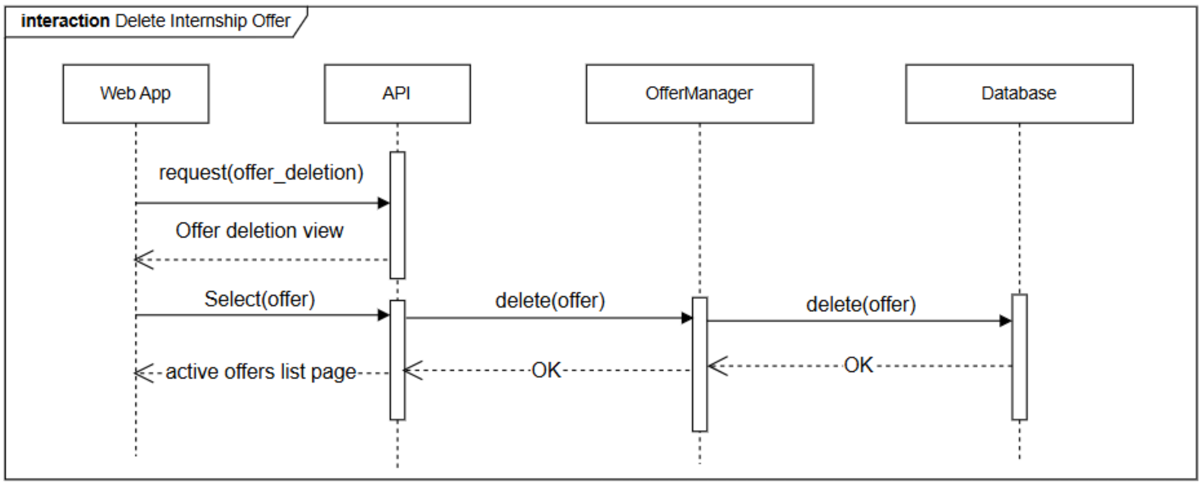


Figure 2.9: Delete Internship offer

[UC6] Browse and Filter Internship Offers

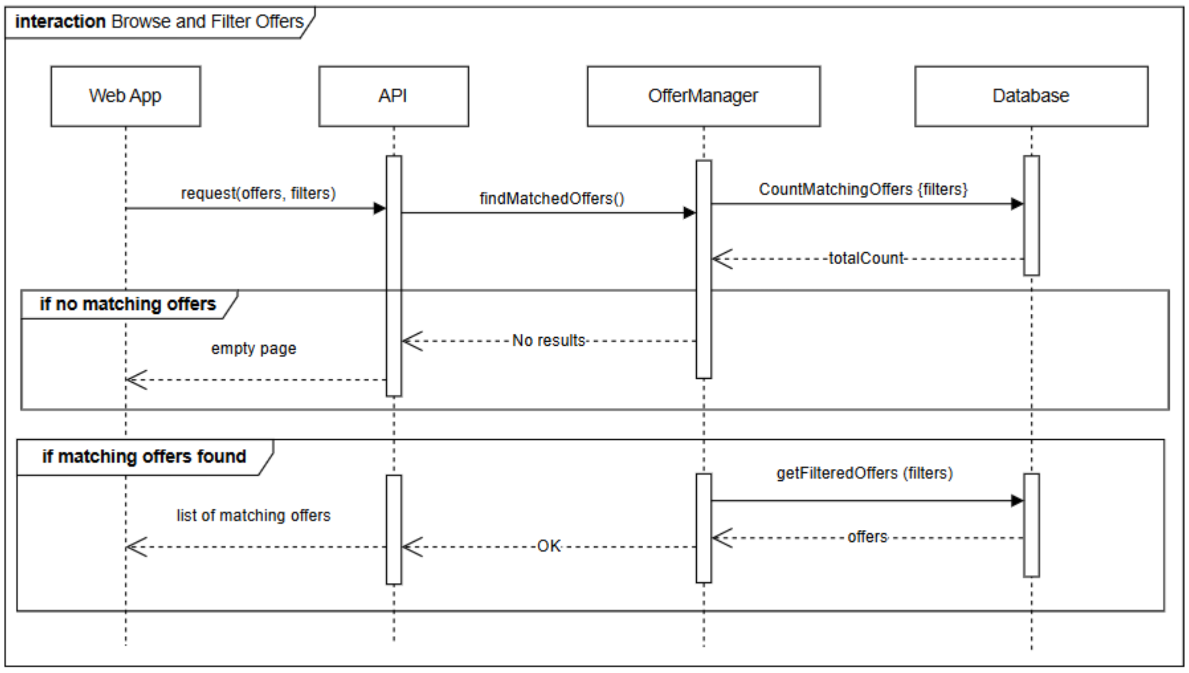


Figure 2.10: Browse and Filter Internship Offers

[UC7] Upload/Update CV

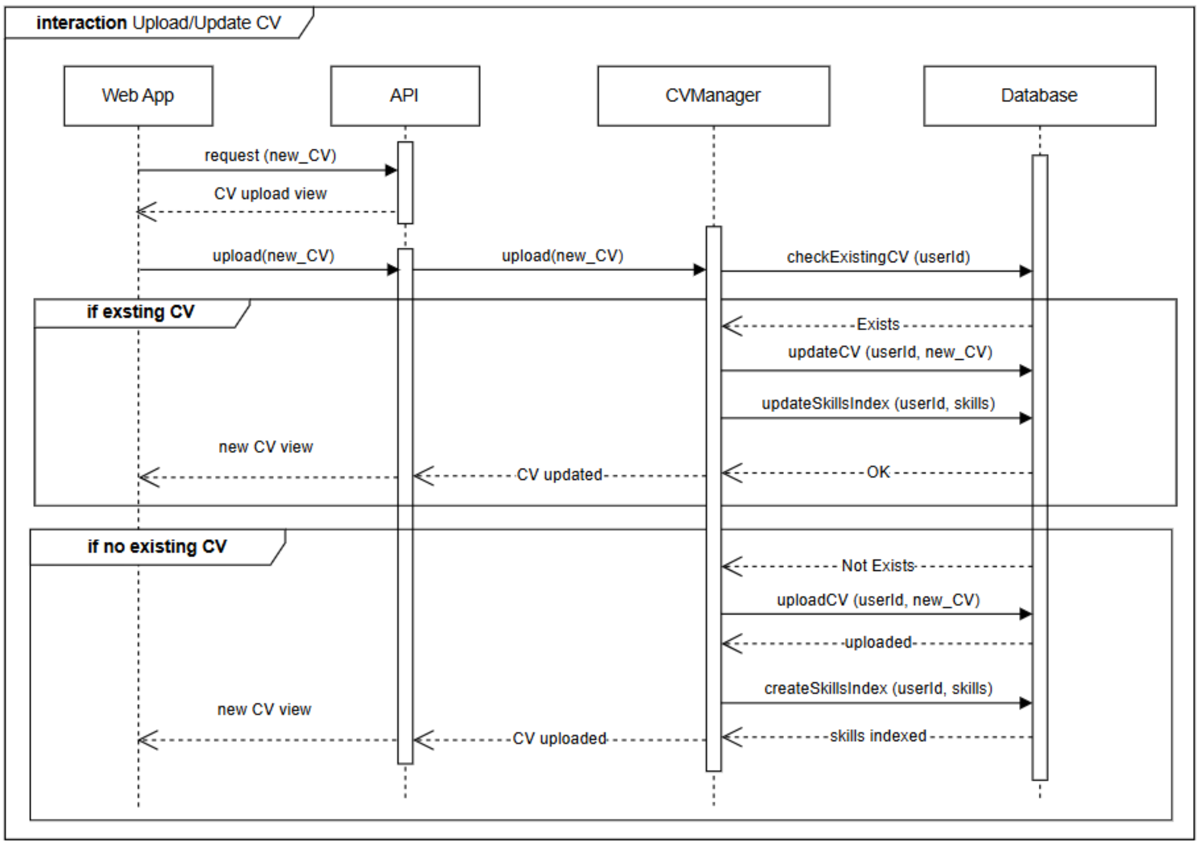


Figure 2.11: Upload/Update CV

[UC8] Apply for Internship Offer

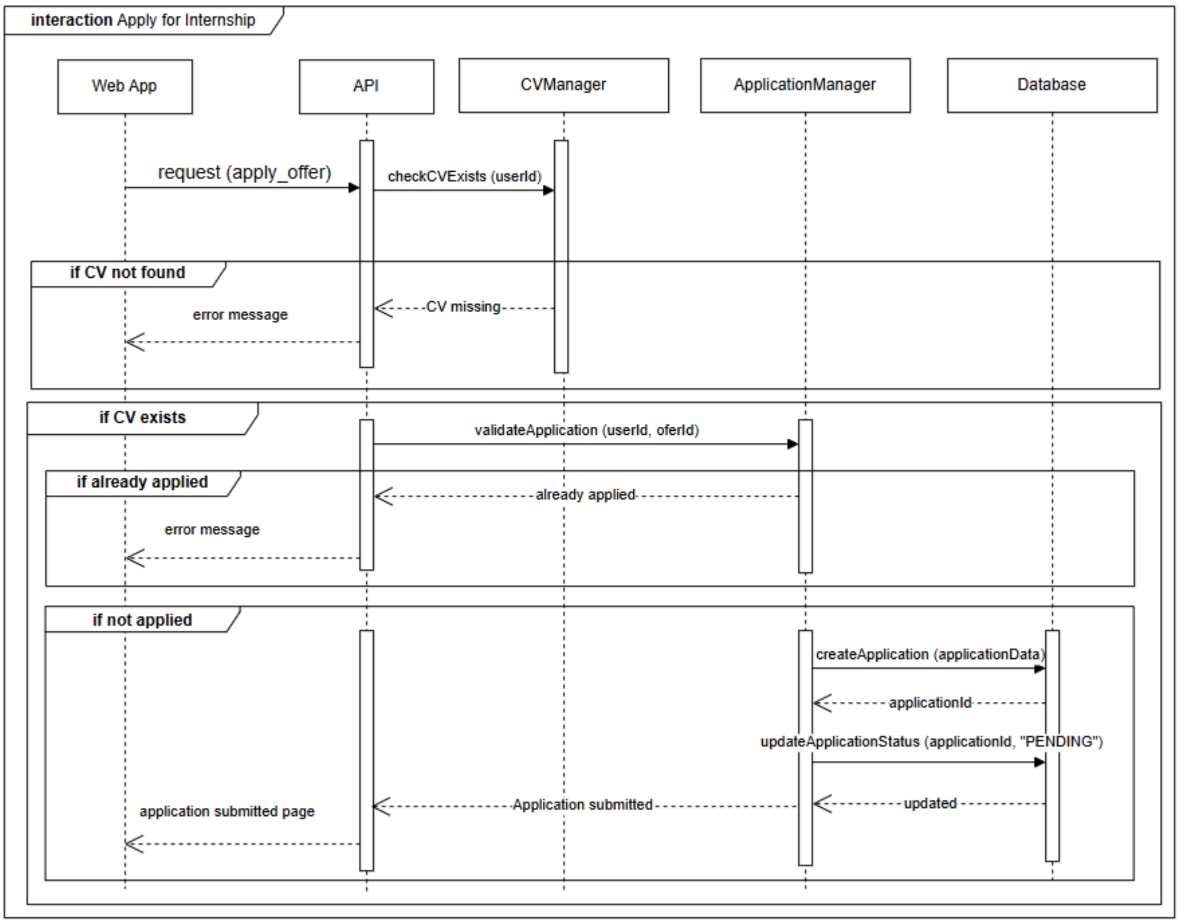


Figure 2.12: Apply for Internship Offer

[UC9] Initiate Interviews

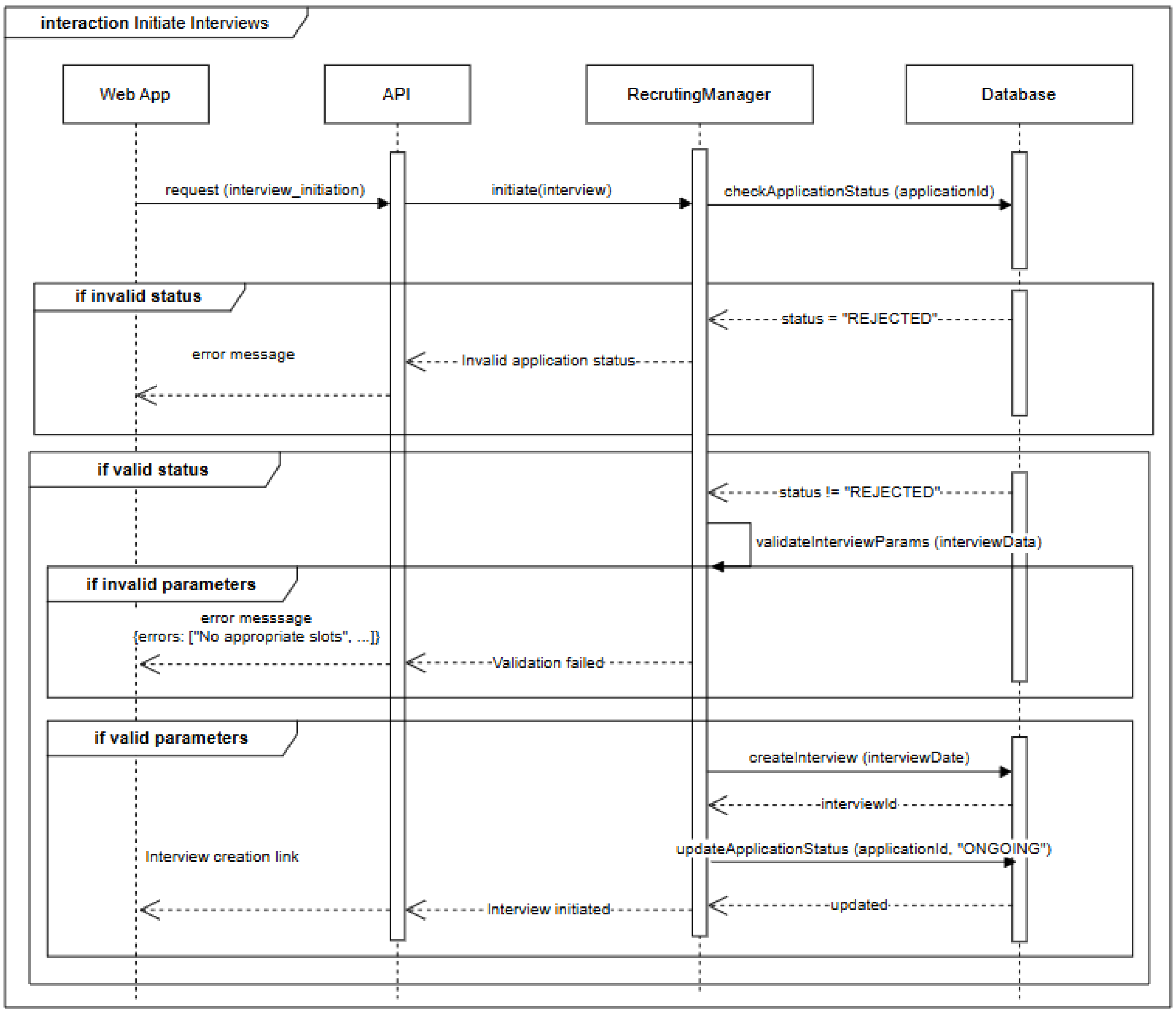

interaction Initiate Interviews
Web App
API
RecrutingManager
Database
request (interview_initiation)
initiate(interview)
checkApplicationStatus (applicationId)
if invalid status
status = "REJECTED"
error message
Invalid application status
if valid status
status != "REJECTED"
validateInterviewParams (interviewData)
if invalid parameters
error messsage
{errors: ["No appropriate slots", ...]}
Validation failed
if valid parameters
createInterview (interviewDate)
interviewId
Interview creation link
updateApplicationStatus (applicationId, "ONGOING")
Interview initiated
updated

Figure 2.13: Initiate Interviews

[UC10] Assign Tests

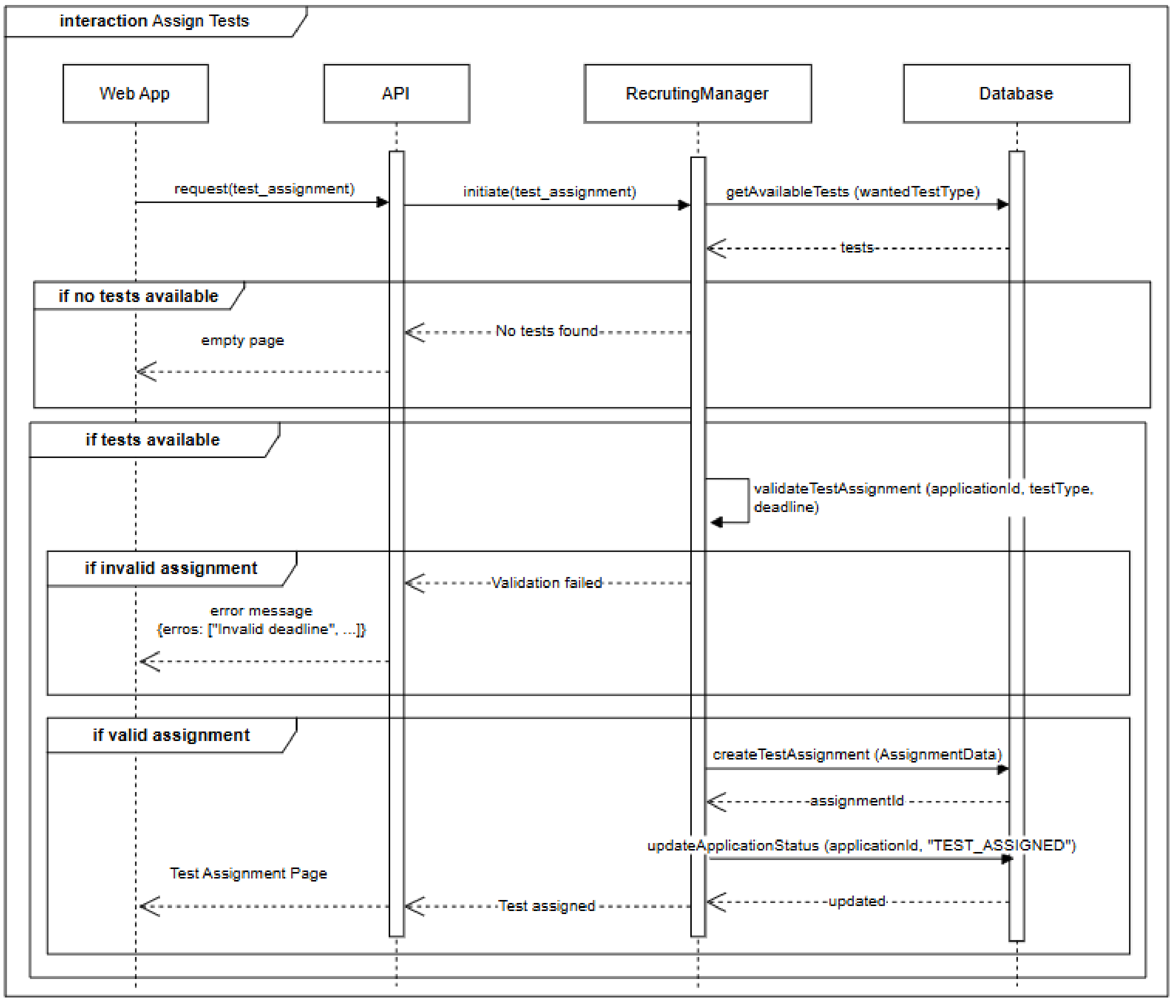


Figure 2.14: Assign Tests

[UC11] Evaluate and Score Candidates

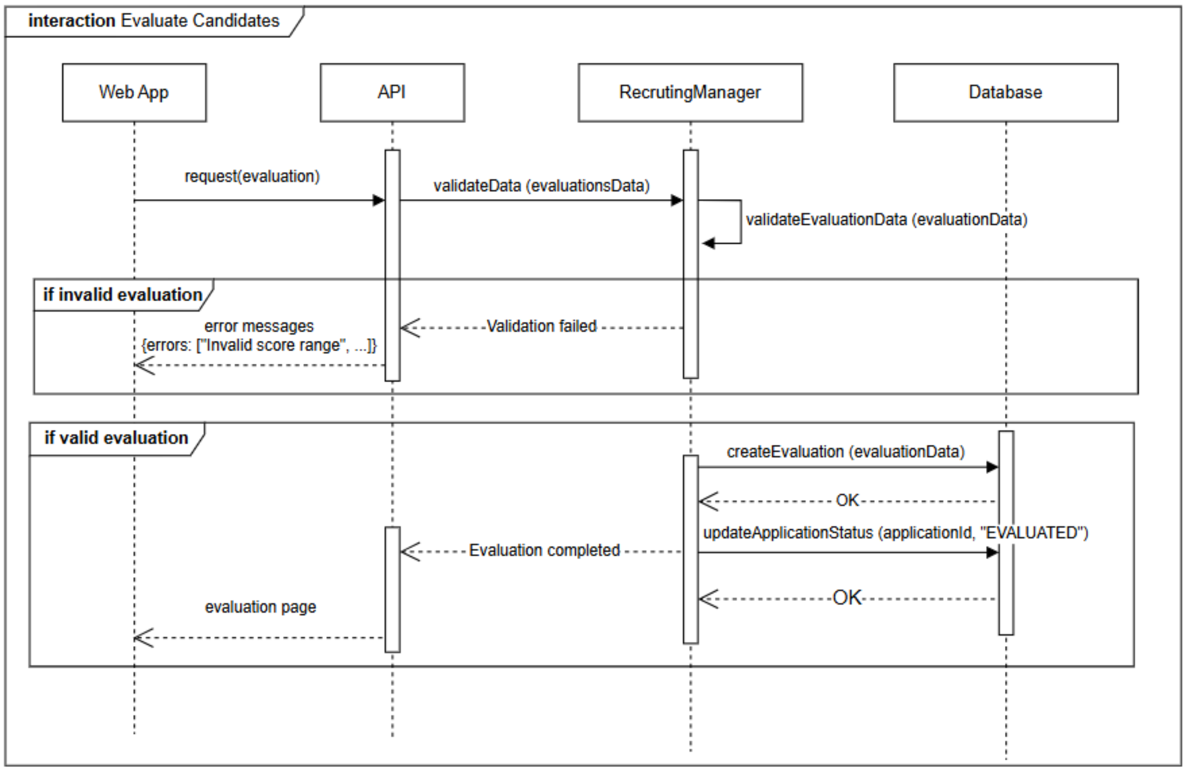

interaction Evaluate Candidates
Web App
API
RecrutingManager
Database
request(evaluation)
validateData (evaluationsData)
validateEvaluationData (evaluationData)
if invalid evaluation
error messages
{errors: ["Invalid score range", ...]}
Validation failed
if valid evaluation
createEvaluation (evaluationData)
OK
updateApplicationStatus (applicationId, "EVALUATED")
Evaluation completed
OK
evaluation page

Figure 2.15: Evaluate and Score Candidates

[UC12] Provide Feedback

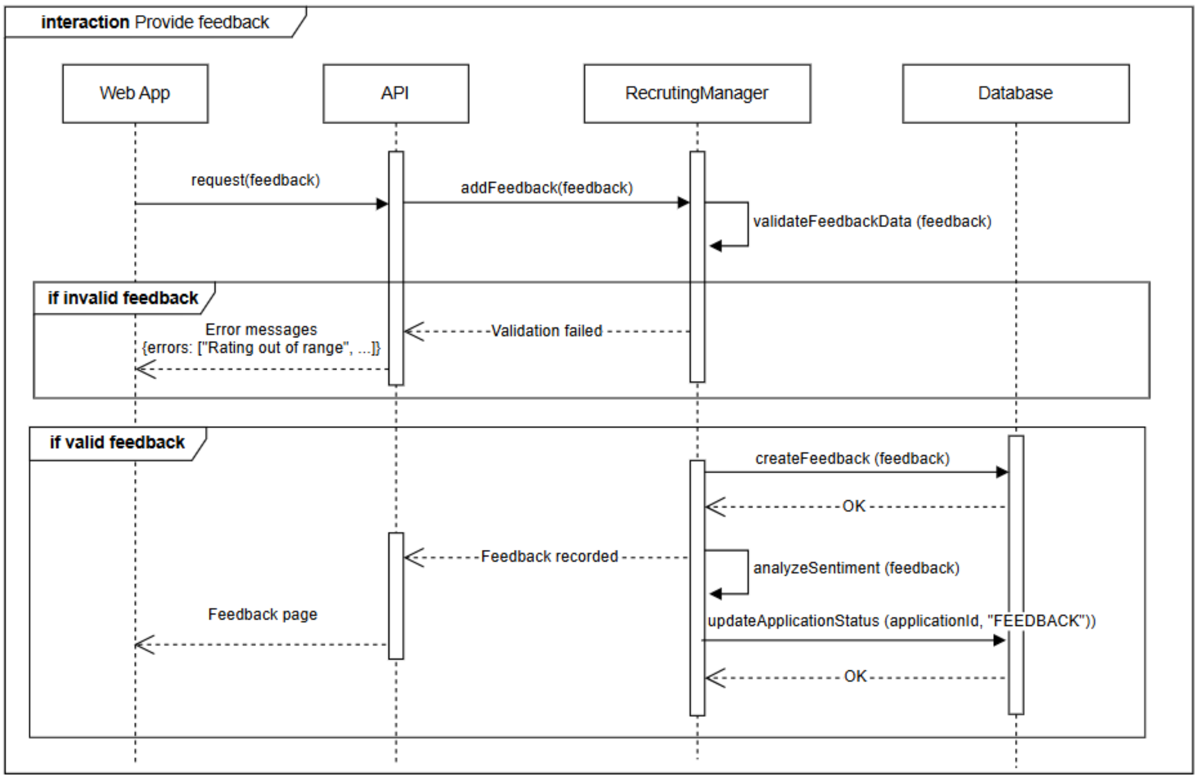

interaction Provide feedback
Web App
API
RecrutingManager
Database
request(feedback)
addFeedback(feedback)
validateFeedbackData (feedback)
if invalid feedback
Error messages
{errors: ["Rating out of range", ...]}
Validation failed
if valid feedback
createFeedback (feedback)
OK
Feedback recorded
analyzeSentiment (feedback)
Feedback page
updateApplicationStatus (applicationId, "FEEDBACK"))
OK

Figure 2.16: Provide Feedback

[UC13] Generate Suggestions

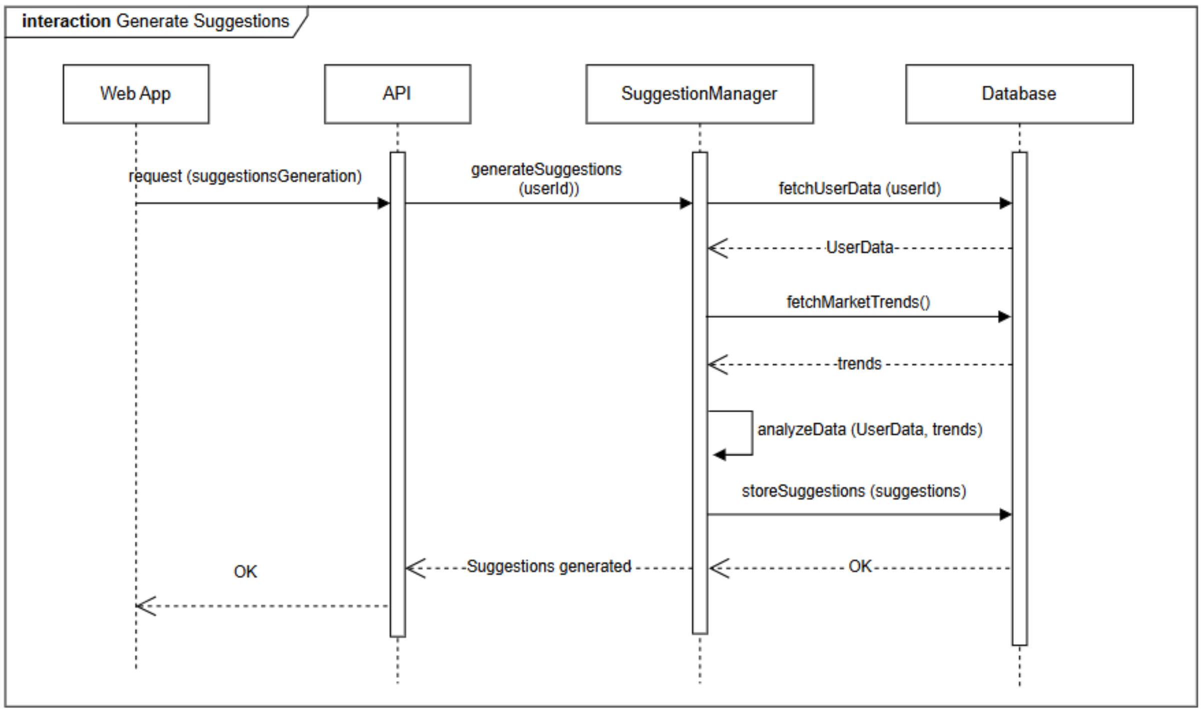

interaction Generate Suggestions
Web App
API
SuggestionManager
Database
request (suggestionsGeneration)
generateSuggestions (userId))
fetchUserData (userId)
UserData
fetchMarketTrends()
trends
analyzeData (UserData, trends)
storeSuggestions (suggestions)
Suggestions generated
OK
OK

Figure 2.17: Generate Suggestions

[UC14] Receive Suggestions

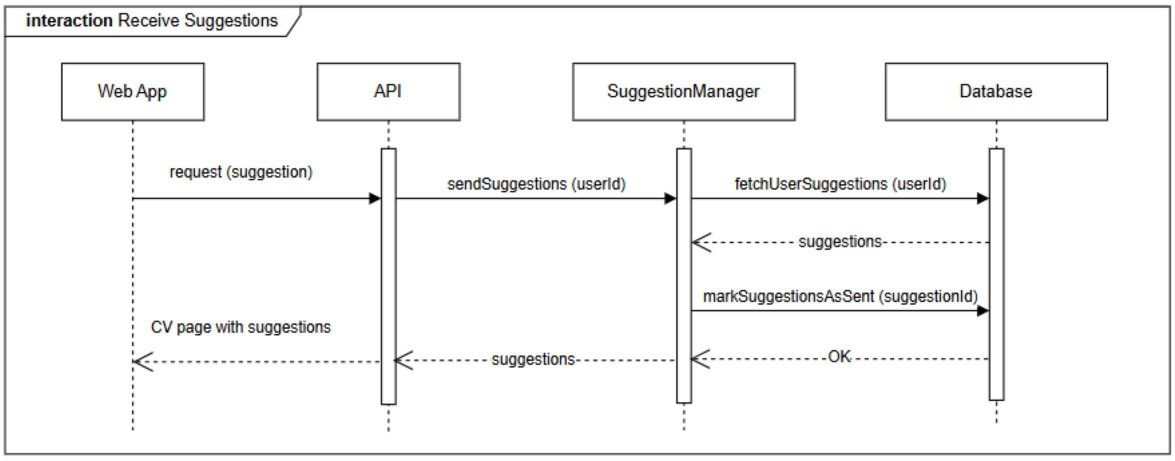


Figure 2.18: Receive Suggestions

[UC15] Generate Recommendations

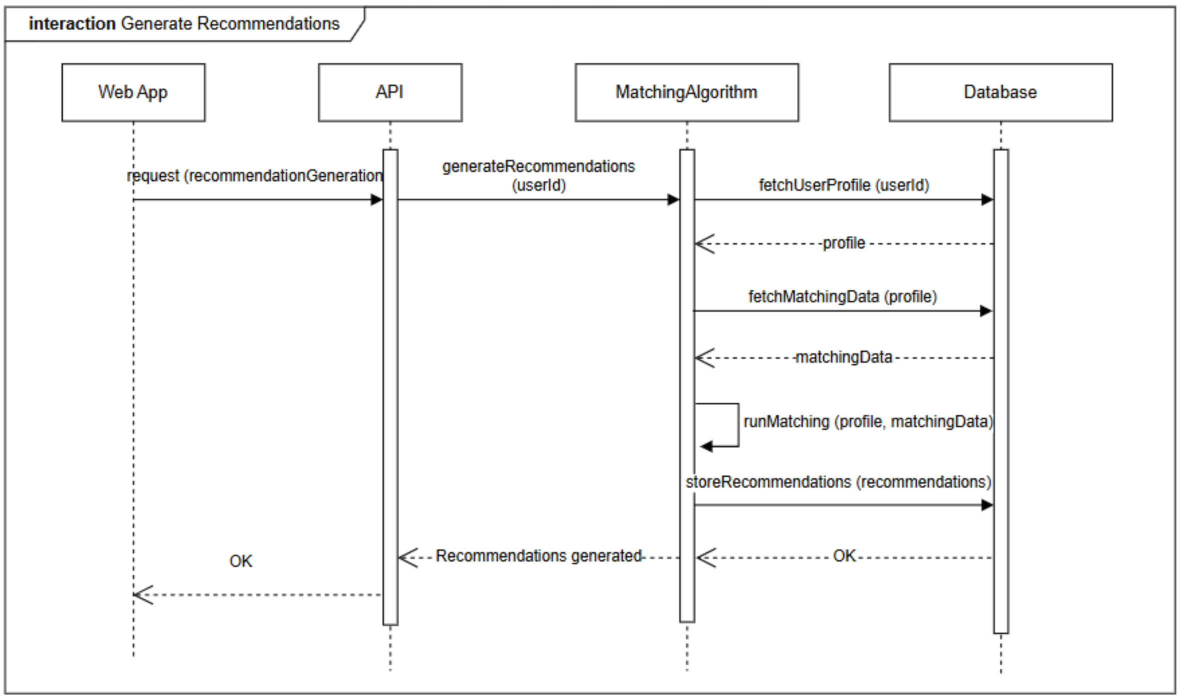

interaction Generate Recommendations
Web App
API
MatchingAlgorithm
Database
request (recommendationGeneration
generateRecommendations (userId)
fetchUserProfile (userId)
profile
fetchMatchingData (profile)
matchingData
runMatching (profile, matchingData)
storeRecommendations (recommendations)
OK
Recommendations generated
OK

Figure 2.19: Generate Recommendations

[UC16] Receive Recommendations

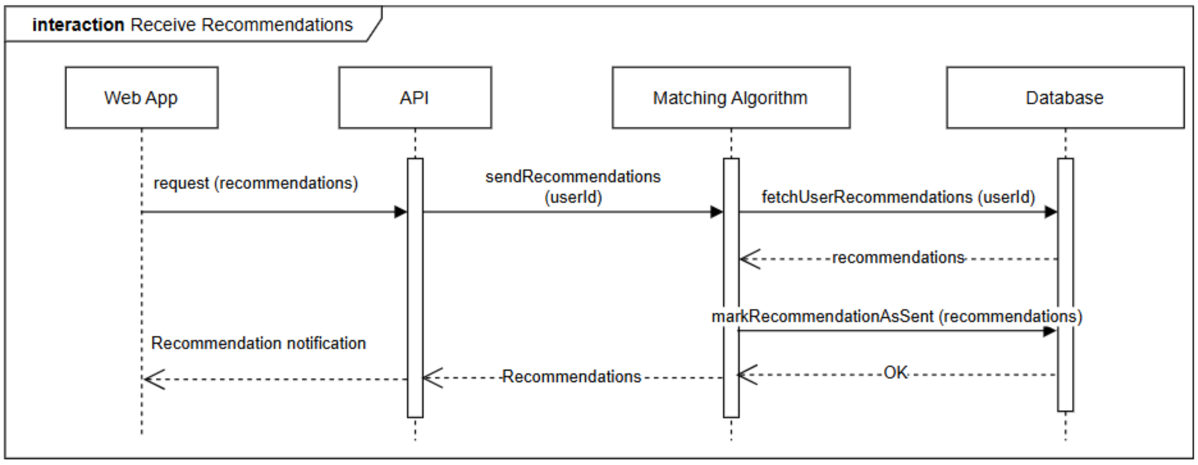

interaction Receive Recommendations
Web App
API
Matching Algorithm
Database
request (recommendations)
sendRecommendations (userId)
fetchUserRecommendations (userId)
recommendations
markRecommendationAsSent (recommendations)
Recommendation notification
Recommendations
OK

Figure 2.20: Receive Recommendations

[UC17] Submit Complaint

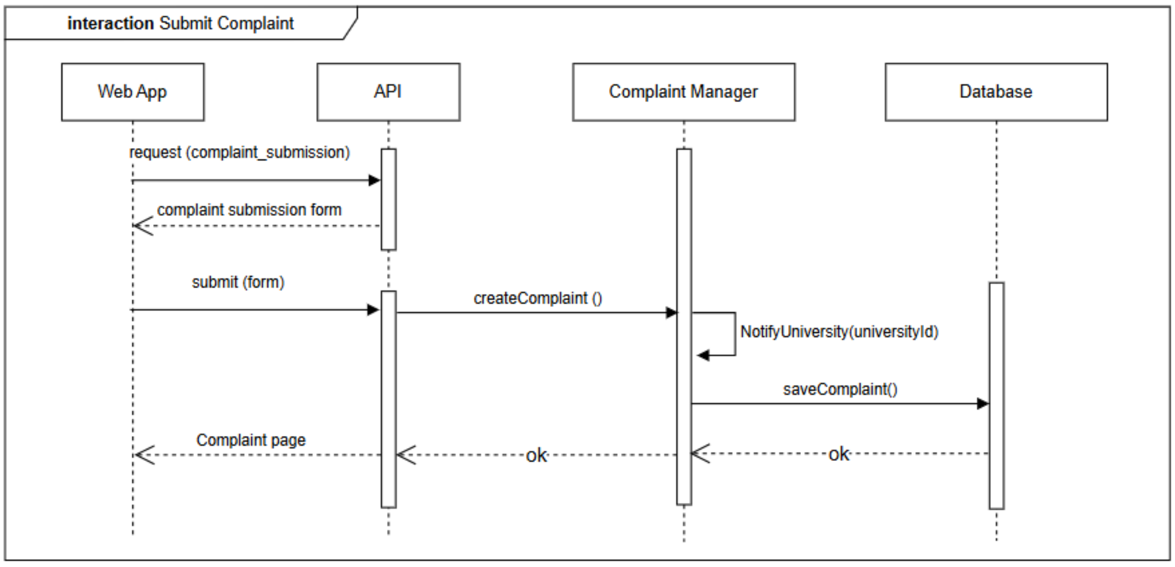

interaction Submit Complaint
Web App
API
Complaint Manager
Database
request (complaint_submission)
complaint submission form
submit (form)
createComplaint ()
NotifyUniversity(universityId)
saveComplaint()
Complaint page
ok
ok

Figure 2.21: Submit Complaint

[UC18] Monitor and Resolve Complaint

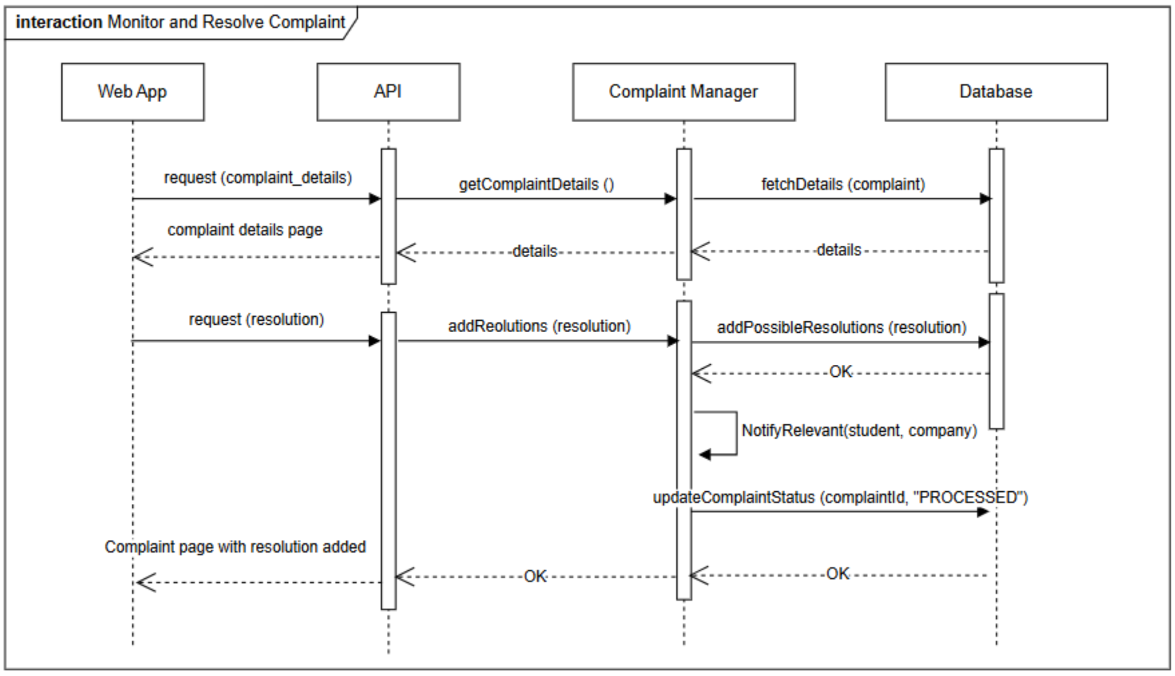

interaction Monitor and Resolve Complaint
Web App
API
Complaint Manager
Database
request (complaint_details)
getComplaintDetails ()
fetchDetails (complaint)
complaint details page
details
details
request (resolution)
addReolutions (resolution)
addPossibleResolutions (resolution)
OK
NotifyRelevant(student, company)
updateComplaintStatus (complaintId, "PROCESSED")
Complaint page with resolution added
OK
OK

Figure 2.22: Monitor and Resolve Complaint

[UC19] Interview scheduling

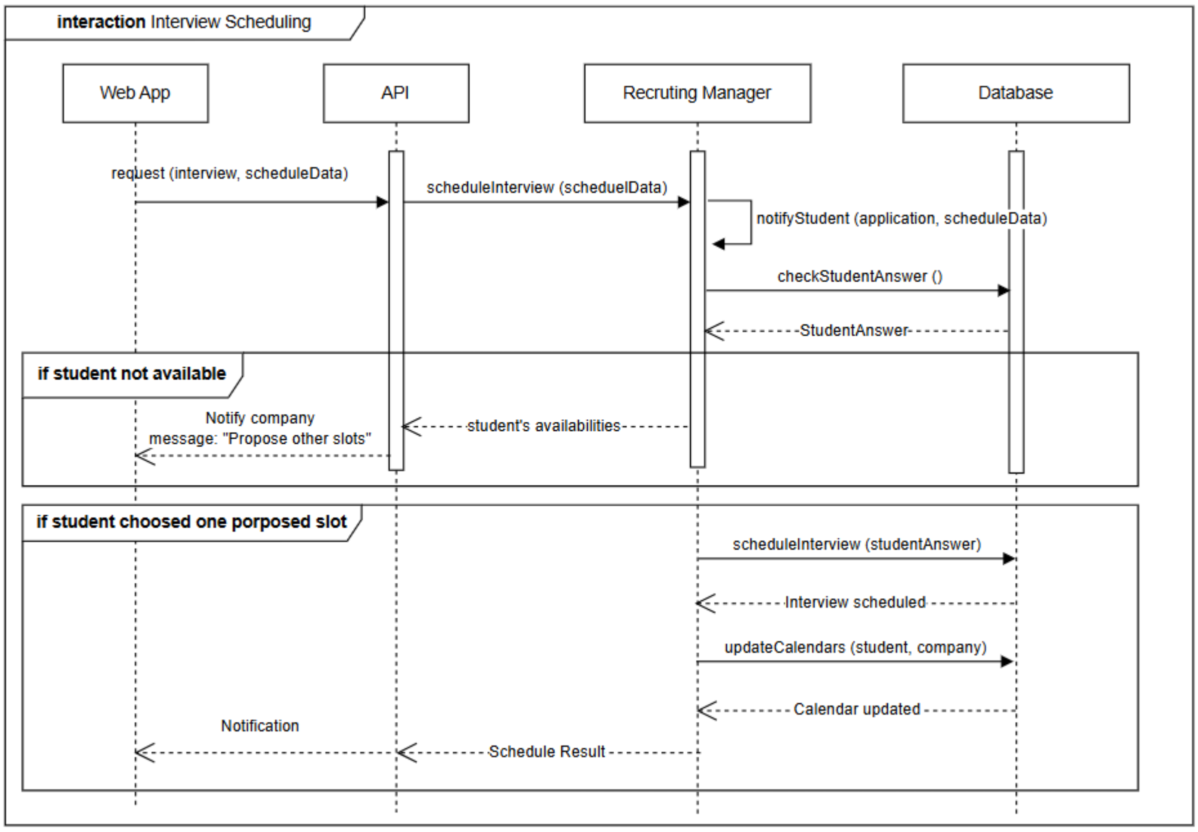

interaction Interview Scheduling
Web App
API
Recruting Manager
Database
request (interview, scheduleData)
scheduleInterview (scheduelData)
notifyStudent (application, scheduleData)
checkStudentAnswer ()
StudentAnswer
if student not available
student's availabilities
Notify company
message: "Propose other slots"
if student choosed one porposed slot
scheduleInterview (studentAnswer)
Interview scheduled
updateCalendars (student, company)
Calendar updated
Notification
Schedule Result

Figure 2.23: Interview scheduling

[UC20] Test taking

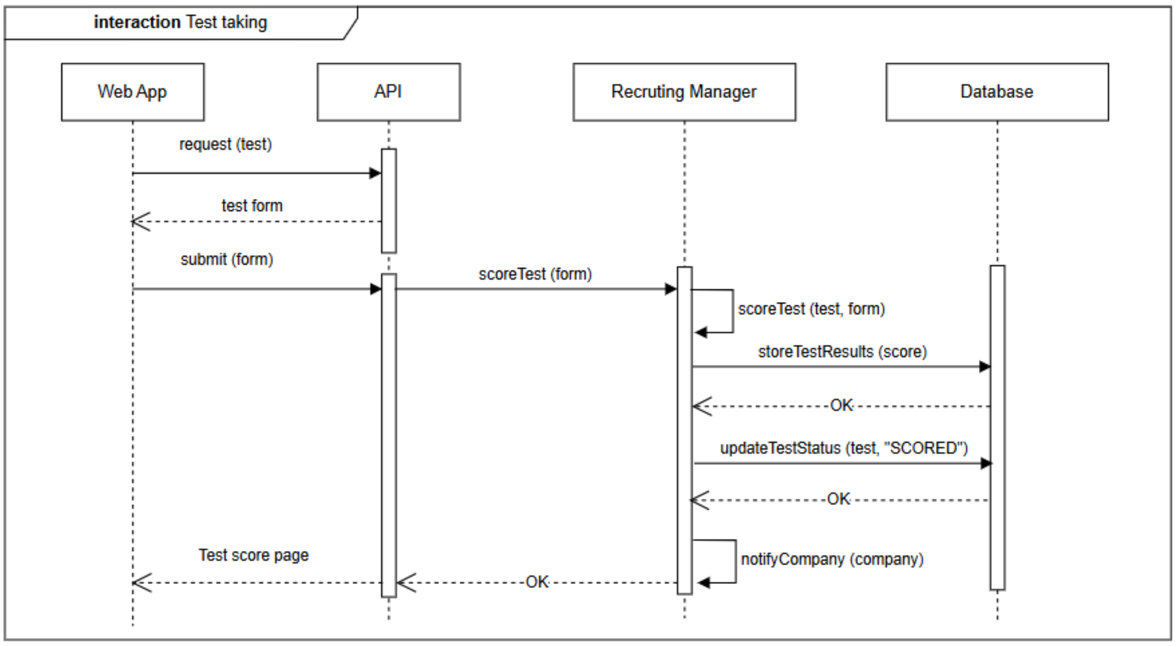


Figure 2.24: Test taking

2.5. Component interfaces

Web App

- request(registration)
- press student sign in button()
- insertRegistration(email)
- request(login)
- press login button()
- insertLogin(email, password)
- request(new_offer)
- submit(form)
- send(offer_info)
- request(offer_modification)
- request(offer_deletion)
- request(offers, filters)

- request(new_CV)
- upload(new_CV)
- request(apply_offer)
- request(inrerview_initiation)
- request(test_assignment)
- request(evaluation)
- request(feedback)
- request(suggestionGeneration)
- request(suggestions)
- request(recommendationsGeneration)
- request(recommendations)
- request(complaint_submission)
- request(complaint_details)
- request(interview, scheduleData) (test)

API

- register(email)
- login(email, password)
- request(new_offer)
- create_offer(...)
- modify_offer(...)
- deleter(offer)
- findMatchedOffers()
- upload(new_CV)
- initiate(interview)
- initiate(test_assignment)
- addFeedback(feedback)

- generateSuggestions(suggestions)
- generateRecommendations(recommendations)
- sendSuggestions(user)
- sendRecommendations(user)
- getComplaintDetails(complaint)
- scoreTest(form)

AuthenticationManager

- checkEmailExists(email)
- create(user_type)
- checkPasswordCorrect(password)
- login()
- checkCVExisting(userId)
- validateApplication (userId, offerId)

CVManager

- checkExistingCV(userId)
- updateCV(userId, new_CV)
- updateSkillsIndex (userId, skills)
- uplaodCV(userId, new_CV)
- createSkillsIndex(userId, skills)

ApplicationManager

- createApplication(applicationData)
- updateApplicationStatus(applicationId, status)

OfferManage

- add(offer_info)
- update(offer_info)
- deleter(offer)

- CountMatchingOffers(filters)
- getFilteredOffers(filters)

MatchingAlgorithm

- fetchUserProfile(user)
- fetchMachingDate(profile)
- runMatching(profile, MatchingData)
- storeRecommendations(recommendation)
- fetchUserRecommendations(user)
- markRecommendationsAsSent(recommendation)

ComplaintManager

- createComplaint(complaint_form)
- NotifyUniversity(university)
- saveComplaint(complaint)
- fetchDetails(complaint)
- addPossibleResolutions(resolutions)
- NotifyRelevant(student, company)
- updateComplaintStatus(complaint, status)

SuggestionManager

- fetchUserData(user)
- fetchMarketTrends()
- analyzeData(userData, trends)
- storeSuggestions(suggestion)
- fetchUserSuggestions(user)
- markSuggestionsAsSent(suggestion)

RecruitingManager

- checkApplicationStatus(application)

- validInterviewParams(interview)
- createInterview(interview)
- updateApplicationStatus (application, status)
- getAvailableTests(wantedType)
- validateTestAssignment(application, wantedTestType, deadline)
- createTestAssignment (test_assignment)
- validateEvaluationData(evaluation)
- createEvaluation(evaluation)
- validateFeedbackData(feedback)
- createFeedback(feedback)
- analyzeSentiment(feedback)
- scheduleInterview(scheduleData)
- notifyStudent(application, scheduleData)
- checkStudentAnswer()
- scheduleInterview(studentAnswer)
- updateCalendars(student, company)
- scoreTest(test,form)
- storeTestResults(score, form)
- notifyCompany(company)

2.6. Selected architectural styles and patterns

2.6.1. 3-tier Architecture

The S&C platform implements a three-tiered architectural style, selected for its robust separation of concerns through three distinct layers:

- **The web server:** this is the presentation tier and provides the user interface. This is usually a web page or a web site. The content can be static or dynamic, and uses technologies such as HTML, CSS and Javascript.

- **The application server:** this is the middle tier, implementing core business logic and processing.
- **The database server:** this is the backend tier of a web application. It runs on DBMS, typically MySQL.

2.6.2. Model View Controller Pattern

For the S&C platform, the MVC pattern provides an optimal structure for web application development through clear separation of responsibilities:

The Model manages application data and business rules, ensuring data integrity and handling business operations. The View transforms data into user-friendly interfaces, allowing multiple representations of the same information depending on user needs. The Controller orchestrates the interaction between users and the system, processing inputs and coordinating responses between Model and View components. This separation enhances maintainability and facilitates independent development of components.

2.6.3. Facade Pattern

To streamline request handling, the platform employs the Facade pattern, which provides a simplified interface to the complex underlying system. This architectural choice offers several advantages: it reduces system complexity from the client's perspective, improves API usability, and facilitates future system modifications by decoupling components. The pattern serves as an excellent foundation for evolving the system toward a more modular architecture.

2.7. Other design decisions

The platform's architecture incorporates several key design decisions to ensure optimal performance and reliability

2.7.1. Availability

For availability, the system implements load balancing mechanisms to distribute traffic efficiently and maintain high availability. Database replication strategies are employed to eliminate single points of failure and ensure continuous service.

2.7.2. Data Storage

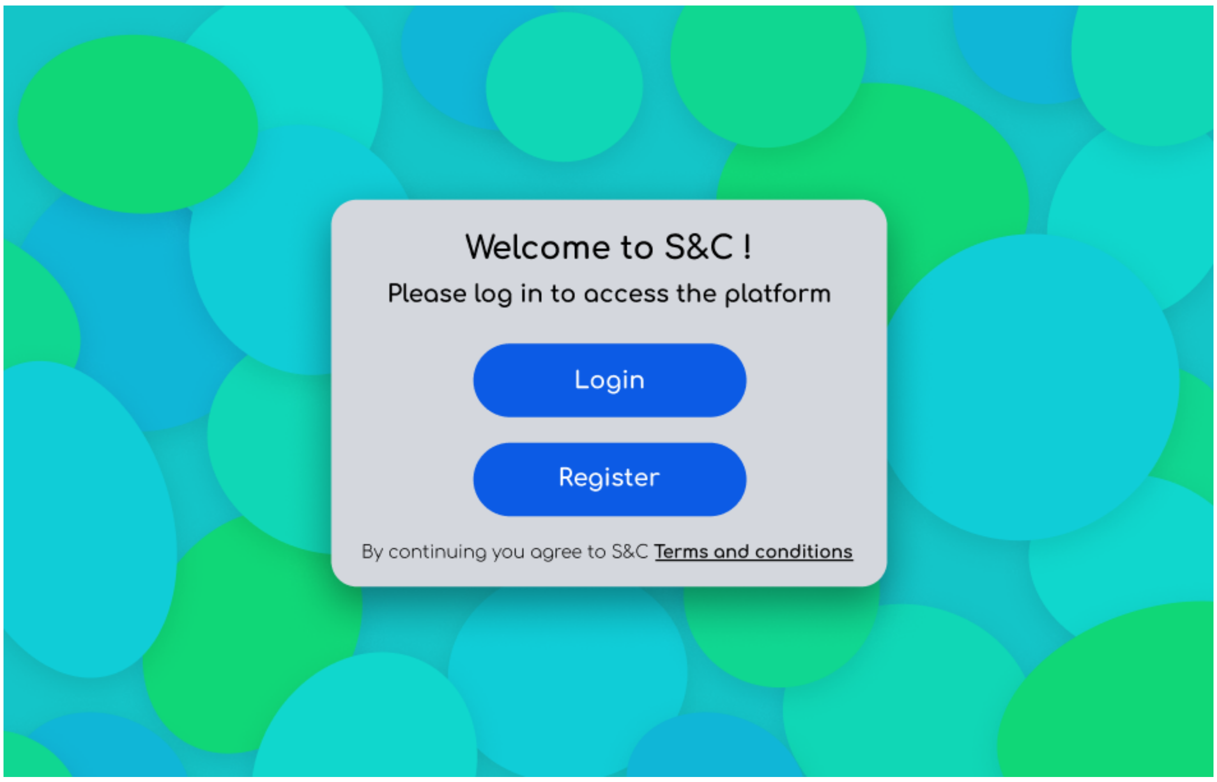
Regarding data management, the system utilizes a sophisticated database structure for storing user information, complemented by optimized data structures to enhance query performance and response times.

2.7.3. Security

Security is maintained through a comprehensive firewall system that filters and protects against various cyber threats, forming a crucial component of the platform's defense strategy.

3 | User Interface Design

The platform should provide intuitive and accessible user interfaces tailored to the three main user groups : Students, Companies and Universities


Welcome to S&C !

Please log in to access the platform

By continuing you agree to S&C Terms and conditions

Figure 3.1: Login Interface

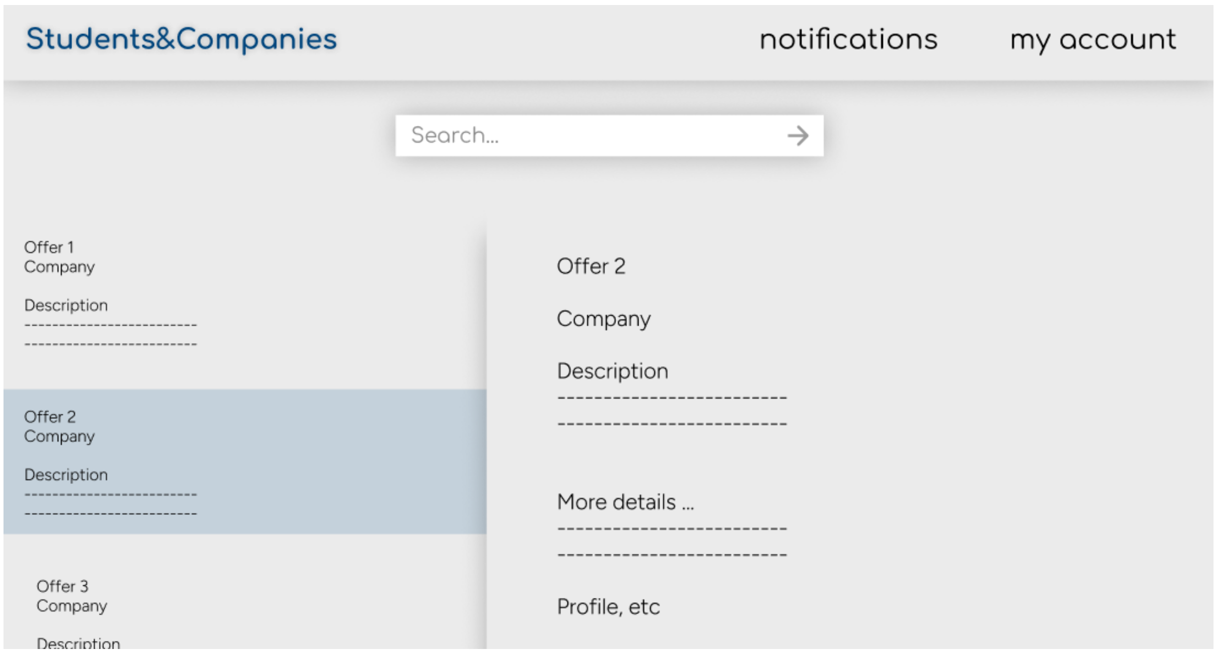


Figure 3.2: Browsing Interface

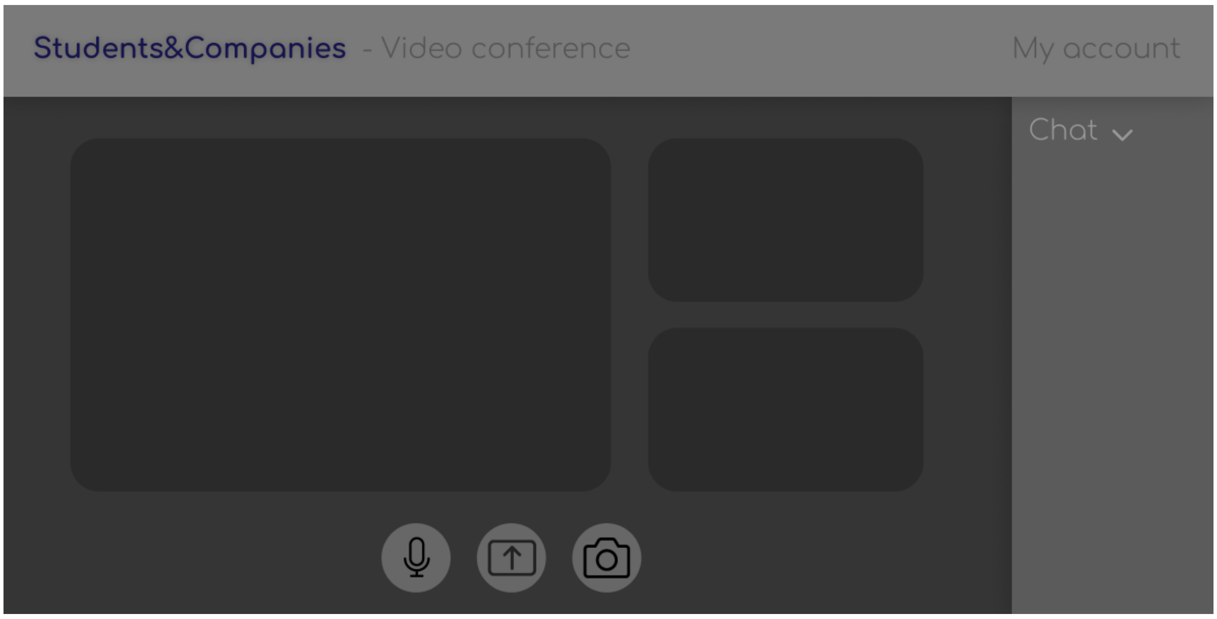


Figure 3.3: Interview Interface

Student Interface:

- Allows students to create accounts, upload CVs, search for offers by keywords and browse internship offers
- Displays personalized suggestions to improve CVs and provides recommendations

for matching internships

- Offers a dashboard to track applications, manage interviews and communicate with Companies
- Includes a complaints section for reporting issues

Company Interface:

- Enables companies to create and publish detailed internship offers
- Offers tools for creating assessment tests and managing candidate evaluations, interviews and recruitment processes
- Displays suggestions to improve internship descriptions and recommendations for suitable candidates
- Includes a complaints section for reporting issues

University Interface:

- Provides tools to monitor ongoing internships and address complaints from both students and companies
- Displays internship statuses, including student progress and company compliance

4 | Requirements Traceability

In this section it is explained how the requirements defined in the RASD, map the design components explained in this document.

Requirements	• **[R1]** The system allows users to register by providing their personal information (Full Name, etc.), a valid email address, and a password. • **[R2]** The system allows registered users to log in.
Components	• WebApp • API • AuthenticationManager • Database: Stores user data.

Requirements	• **[R3]** The system allows companies to create internship offers. • **[R4]** The system allows companies to manage the status of their offers. • **[R5]** The system allows students to browse and filter internship offers.
Components	• WebApp • API • OfferManager • Database

Requirements	• **[R6]** The system notifies students of matching offers.
Components	• WebApp • API • MatchingAlgorithm • Database

Requirements	• **[R7]** The system allows students to upload and manage their CVs. • **[R8]** The system allows students to edit or delete their CVs.
Components	• WebApp • API • CVManager • Database

Requirements	• **[R9]** The system allows students to apply for internships. • **[R10]** The system allows companies to review applications.
Components	• WebApp • API • ApplicationManager • Database

Requirements	• **[R11]** The system allows companies to manage the recruitment process. • **[R12]** The system provides monitoring tools for tracking recruitment progress. • **[R13]** The system enables companies to evaluate candidates during recruitment. • **[R14]** The system notifies candidates of their recruitment status.
Components	• WebApp • API • RecrutingManager • Database

Requirements	• **[R15]** The system analyzes user activities to provide personalized suggestions. • **[R16]** The system provides CV improvement suggestions for students. • **[R17]** The system provides offer improvement suggestions for companies.
Components	• WebApp • API • SuggestionManager • Database

Requirements	• **[R18]** The system allows students to provide feedback on their application and recruitment experiences. • **[R19]** The system allows companies to provide feedback on candidates they evaluated and recruitment experiences. • **[R20]** The system uses a matching algorithm to analyze student CVs, internship offers, and user feedback to generate recommendations. • **[R21]** The system sends recommendations to both students and companies when a match is found.
Components	• WebApp • API • MatchingAlgorithm • Database

Requirements	• **[R22]** The system allows students and companies to submit complaints about any issues related to the internship or recruitment process. • **[R23]** The system allows universities to monitor and resolve complaints submitted by students or companies. • **[R24]** The system notifies universities of unresolved issues or complaints requiring attention.
Components	• WebApp • API • ComplaintManager • MonitoringManager • Database

Requirements	• **[R25]** The system notifies students of application deadlines, interview schedules, test results, and application status updates in real time. • **[R26]** The system notifies companies of updates on candidate applications and recruitment stages.
Components	• WebApp • API • ApplicationManager • RecrutingManager • Database

5 | Implementation,Integration and Test Plan

5.1. Overview

This chapter explains how the described platform will be implemented and tested. The purpose of testing is to identify and fix most of the bugs in the code developed by the team. Additionally, a detailed description of how the different components of the code are integrated will be provided in section 5.3. Meanwhile, section 5.2 will present the key implementation strategies used to develop the project.

5.2. Implementation Plan

Implementation and Integration Strategy

This section outlines our hybrid implementation approach, which combines elements of multiple development methodologies to maximize efficiency and visibility.

Our implementation begins with establishing a foundational framework using the top-down approach. This creates a robust architectural skeleton upon which we can build. As development progresses, individual features are implemented as complete functional threads, allowing stakeholders to see tangible progress throughout the development cycle.

The hybrid strategy enables parallel development streams where different teams can work independently on distinct components. Once these components pass validation, they are integrated into the main system architecture. This approach reduces integration complexity while maintaining development speed and quality assurance.

This methodology provides several advantages: it allows for early demonstration of functionality, enables efficient resource allocation across teams, and ensures systematic growth of the system while maintaining architectural integrity. The reduced need for development stubs and drivers simplifies the testing process, though it requires careful coordination during integration phases.

5.2.1. Features identification

Following are the most important features to implement:

[F1] Sign-up and Sign-in
These are quite easy to test and require the same methodology. It is important to distinguish the three types of users, company, university, and student, because they have different levels of access.

[F2] CV upload
This feature allows students to upload and manage their CVs on the platform. It needs to handle file uploads, validate CV formats, and store CV data securely. Testing should verify proper file handling, storage, and retrieval of CV information, as well as the CV update functionality.

[F3] Offer creation
Companies can create, modify, and delete internship offers through this feature. Testing should verify all CRUD operations for offers, including validation of required fields (title, description, required skills, etc.), proper storage in the database, and correct display to students.

[F4] Application creation
This feature enables students to apply for internship offers by submitting their applications. Testing should verify the application submission process, including CV attachment, proper status tracking, and notification delivery to both students and companies.

[F5] Recruiting process
This complex feature involves managing the entire recruitment workflow, including interview scheduling, test assignment, and candidate evaluation. Testing should verify each step of the process:

- Interview scheduling and management
- Test assignment and completion
- Evaluation and scoring functionality
- Status updates and notifications

- Integration between different recruitment stages

[F6] Complaint filing
This feature allows students and companies to submit complaints about issues during the internship or recruitment process. Testing should verify:

- Complaint submission functionality
- Proper routing to universities
- Status tracking
- Communication channels between parties
- Resolution documentation

[F7] Feedback providing
This feature enables users to provide feedback on their experiences. Testing should verify:

- Feedback submission for both students and companies
- Proper storage and retrieval of feedback
- Integration with the recommendation system
- Analysis of feedback for platform improvement

[F8] Recommendation
This feature uses the matching algorithm to generate personalized recommendations. Testing should verify:

- Accuracy of the matching algorithm
- Proper delivery of recommendations to both parties
- Status tracking of recommendations
- Integration with the application process
- Prevention of conflicts between recommendations and existing applications

[F9] Suggestion
This feature provides personalized suggestions for improving CVs and offers. Testing should verify:

- Generation of relevant suggestions based on user data

- Proper delivery of suggestions to users
- Tracking of suggestion implementation
- Integration with the platform's analytics system
- Impact of suggestions on matching

5.3. Component Integration and Testing

This section details the integration testing approach for each major feature of the S&C platform. For each feature, we present an integration test diagram showing the relationship between test drivers, components under test, and test stubs, followed by a description of the integration testing strategy.

The sign-up and sign-in integration testing focuses on verifying the interaction between the authentication component and the database layer. The test driver simulates various user registration and login scenarios, while the database stub provides predefined responses. Key integration test cases include:

- Verification of data flow between UI and authentication manager
- Validation of user credentials against database records
- Session management and token handling
- Error handling across component boundaries

[F1] Sign-up and Sign-in

The Driver simulates user authentication requests (both sign-up and sign-in attempts). The AuthenticationManager processes these requests, applying validation rules and authentication logic. The Model stub simulates database responses for user verification and storage. Integration testing focuses on verifying the correct flow of authentication data between these components.

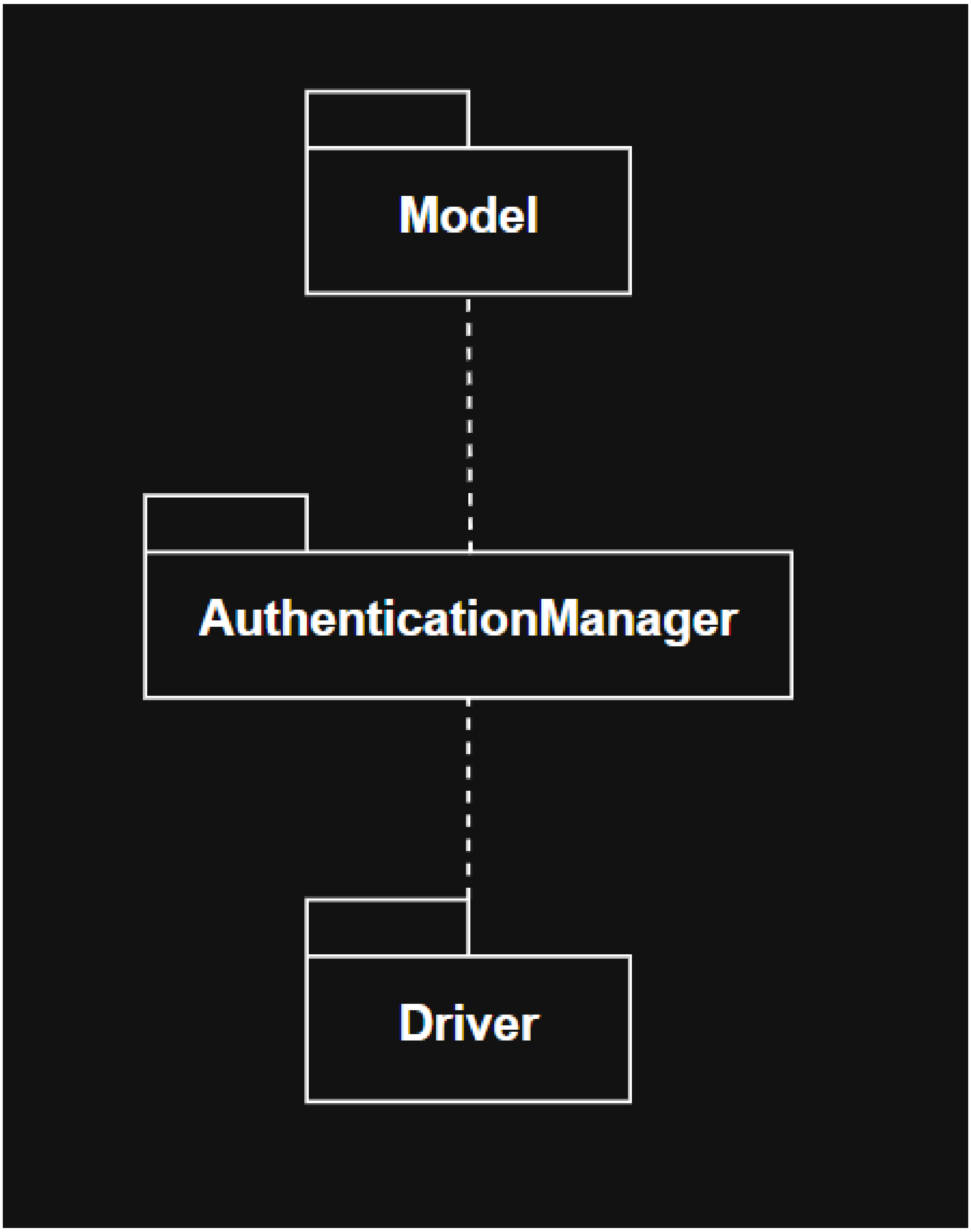


Figure 5.1: Sign-up and Sign-in

[F2] CV upload

The Driver generates CV upload requests with test files and metadata. The CVManager handles file processing and storage operations. The Model stub simulates the storage system and database responses. Integration testing ensures proper handling of file uploads and storage confirmations across these components.

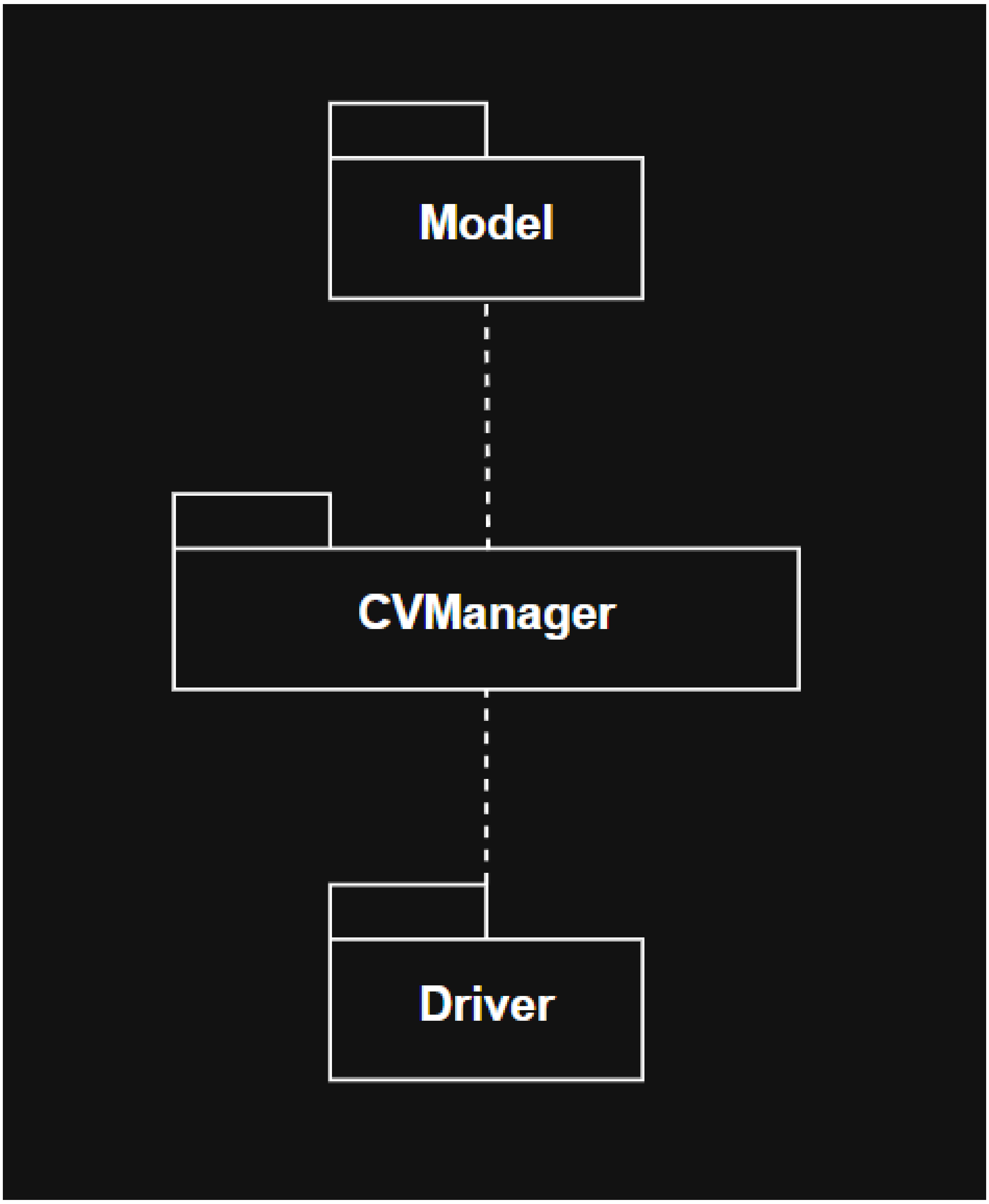


Figure 5.2: CV upload

[F3] Offer creation

The Driver simulates company requests for creating, modifying, and deleting internship offers. The OfferManager processes these requests, handling offer validation and management operations. The Model stub simulates the database operations for offer storage and retrieval. Integration testing verifies the proper handling of offer data across these components, including creation, modification, and deletion operations.

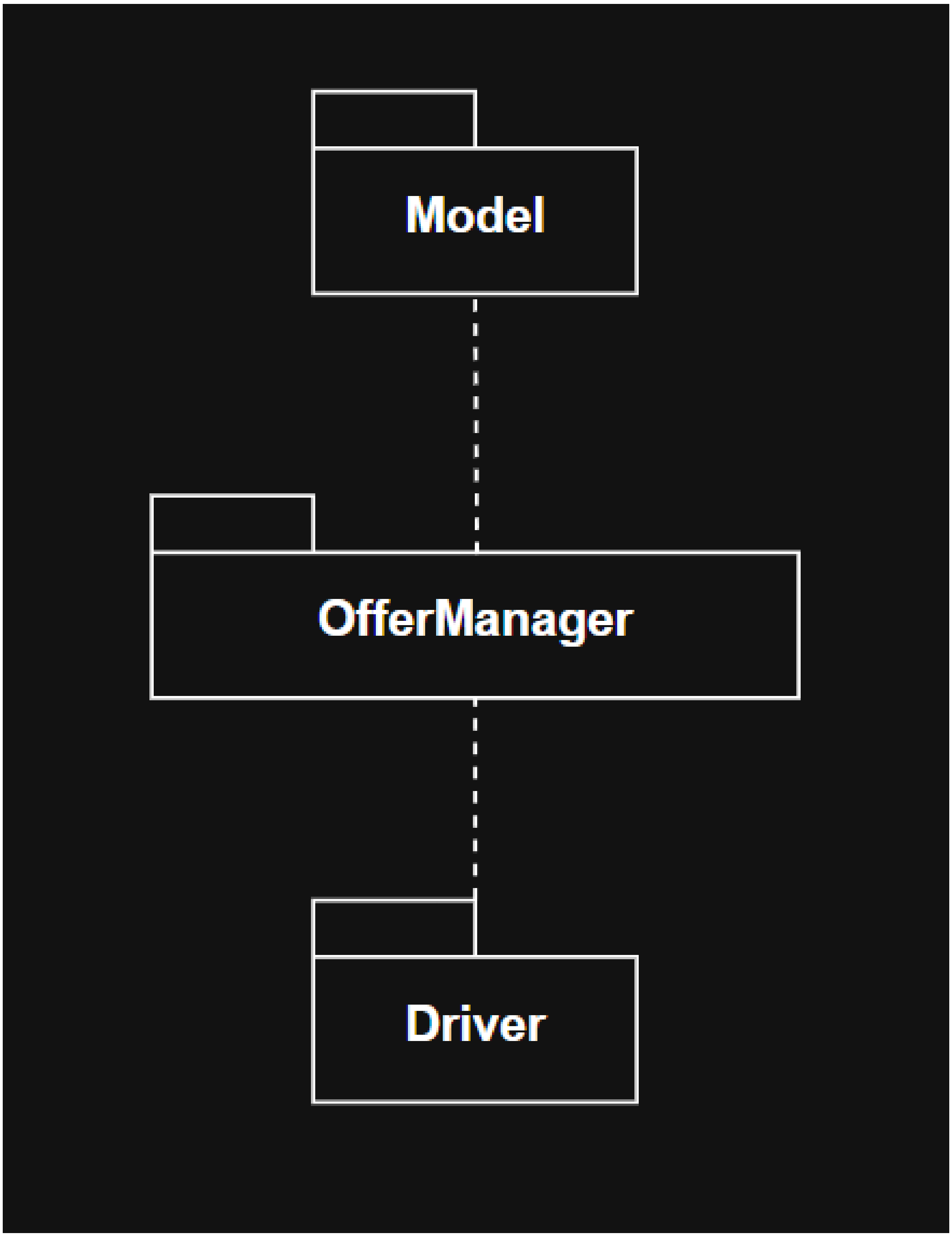


Figure 5.3: Offer creation

[F4] Application creation

The Driver simulates student application requests. The ApplicationManager processes these requests and coordinates with CVManager to verify CV existence and OfferManager to validate offer availability. The Model stub simulates database operations for storing application data. Integration testing verifies the proper flow of application data across components, including CV and offer validation, and ensures correct storage of application status.

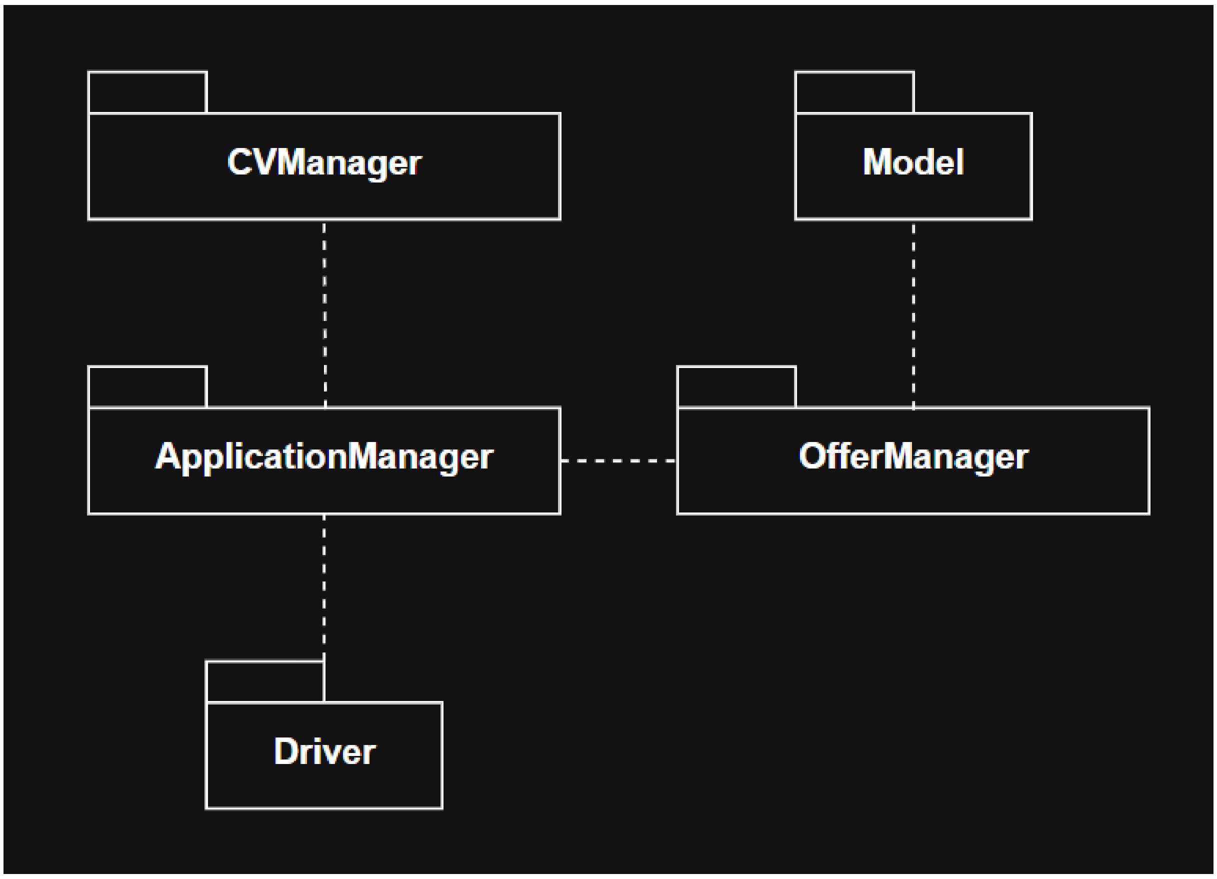


Figure 5.4: Application creation

[F5] Recruiting process

The Driver simulates various recruiting actions including interview scheduling, test assignments, and candidate evaluations. The RecruitingManager handles all recruitment operations internally (interviews, tests) and coordinates with ApplicationManager to track application status. The Model stub simulates database operations for storing all recruitment-related data. Integration testing verifies the proper handling of the entire recruitment workflow, including interview scheduling, test management, status updates, and data persistence.

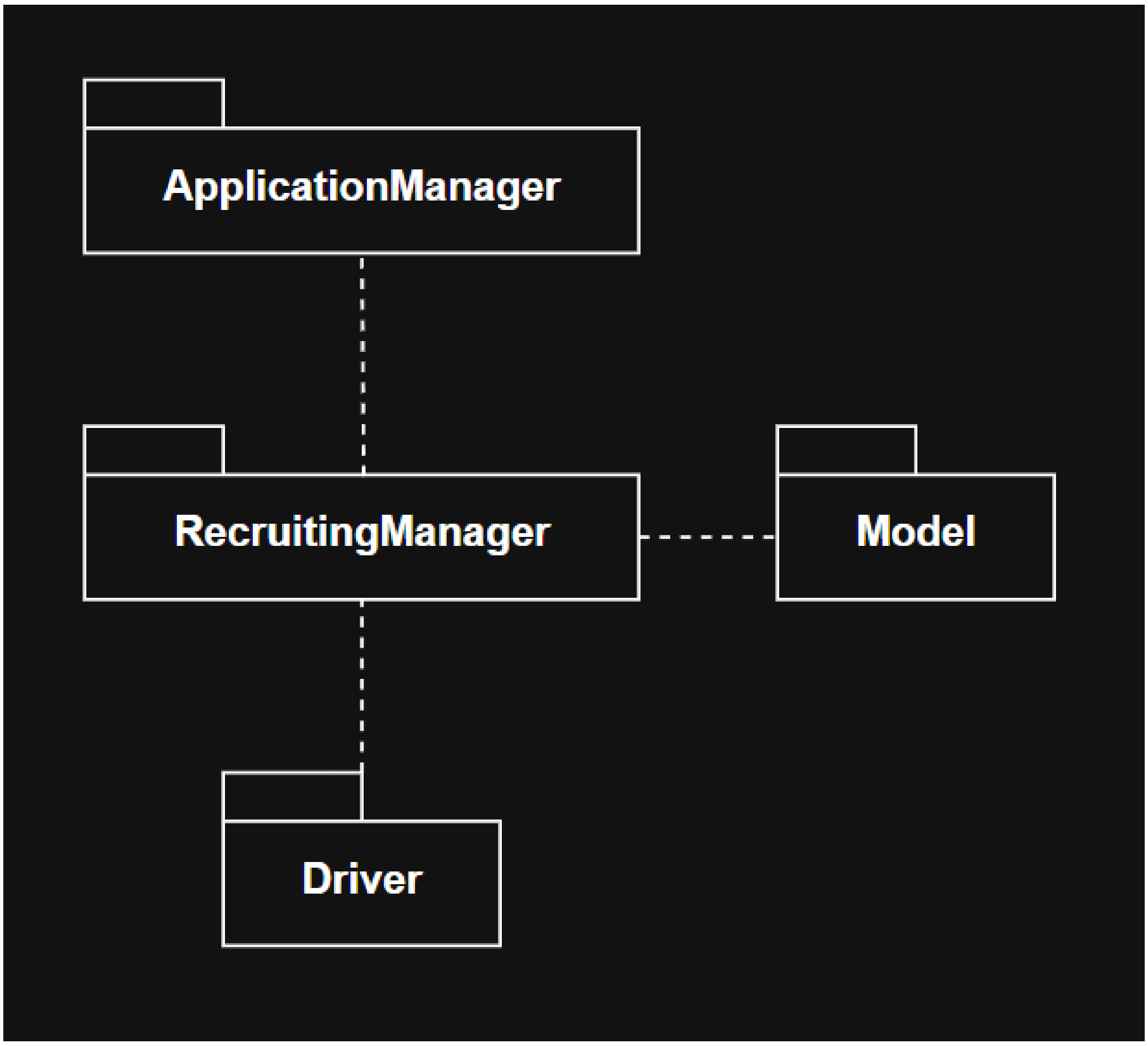


Figure 5.5: Recruiting process

[F6] Complaint filing

The Driver simulates complaint submissions from students and companies. The ComplaintManager processes these complaints and coordinates with MonitoringManager to ensure proper university oversight. The Model stub simulates database operations for storing complaint data and tracking resolution status. Integration testing verifies the proper handling of complaints across components, including submission, university notification, status tracking, and resolution documentation.

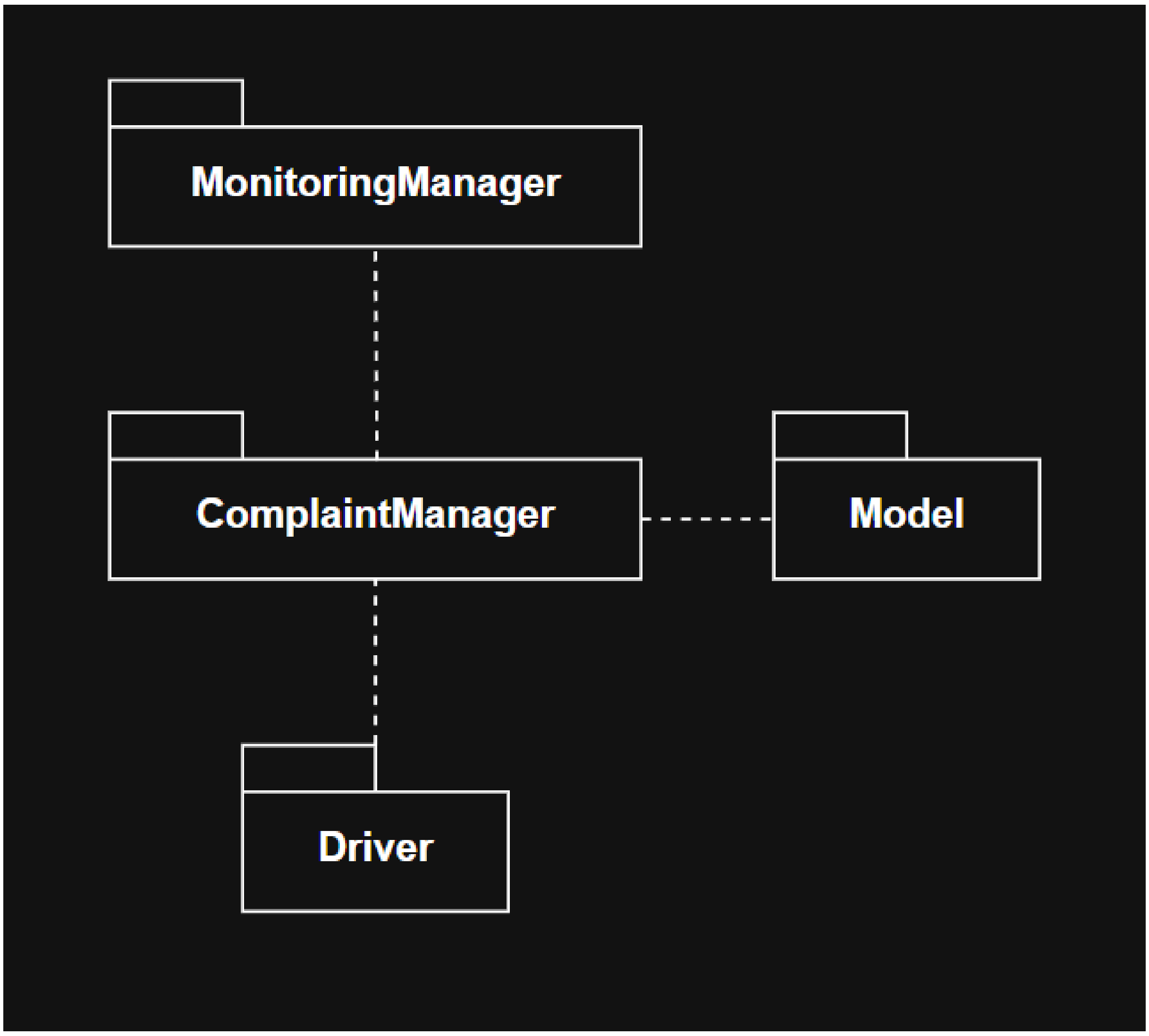


Figure 5.6: Complaint filing

[F7] Feedback providing

The Driver simulates feedback submissions from both students and companies about their recruitment experiences. The RecruitingManager processes this feedback as part of the overall recruitment process, since feedback is tied to specific recruitment interactions. The Model stub simulates database operations for storing feedback data. Integration testing verifies the proper handling of feedback submission, storage, and association with the relevant recruitment processes.

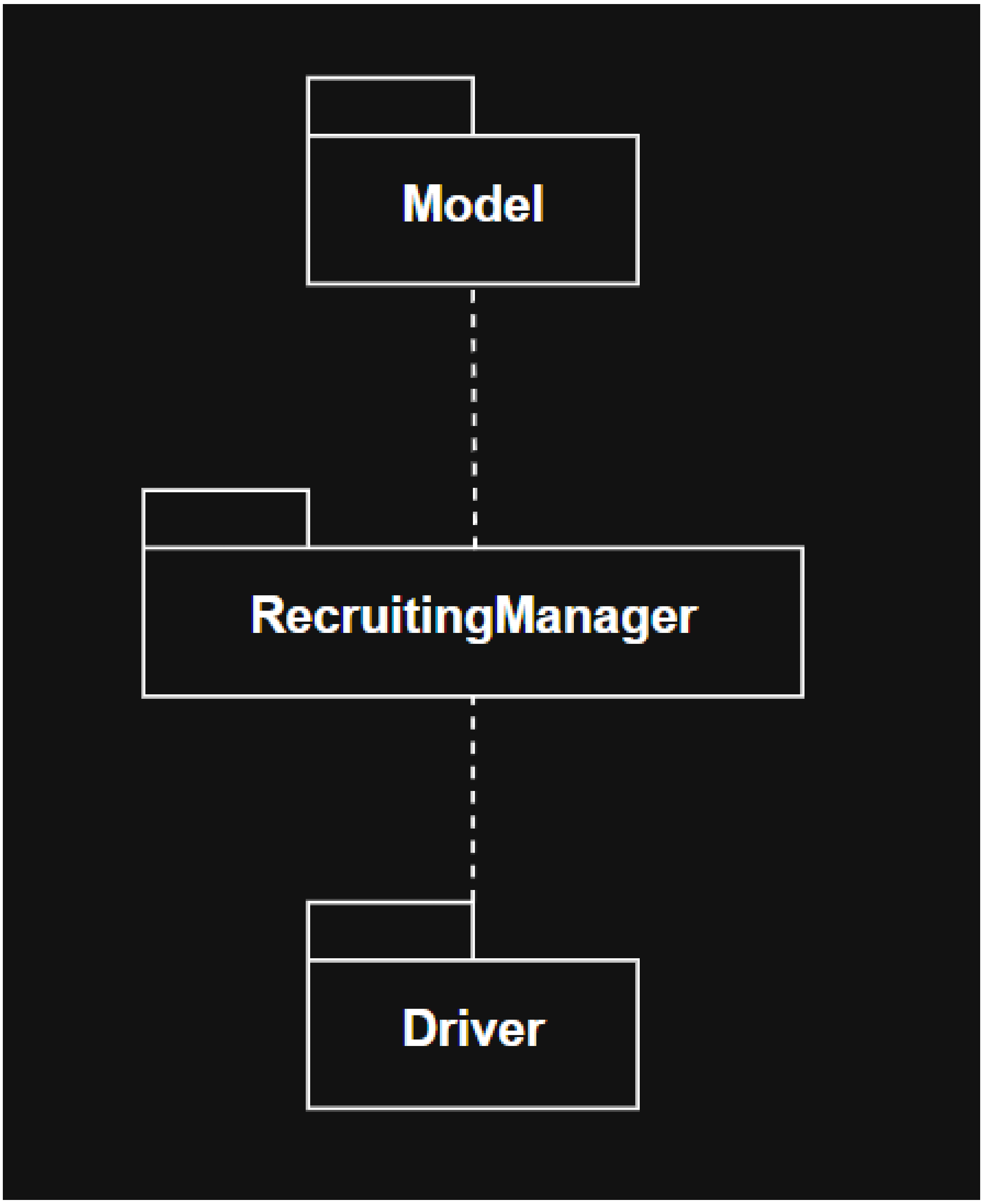


Figure 5.7: Feedback providing

[F8] Recommendation

The Driver simulates the recommendation generation process. The MatchingAlgorithm analyzes student CVs through CVManager and internship offers through OfferManager to generate personalized recommendations. The Model stub simulates database operations for storing and retrieving matching data and recommendations. Integration testing verifies the proper analysis of CVs and offers, generation of relevant recommendations, and correct storage of recommendation data.

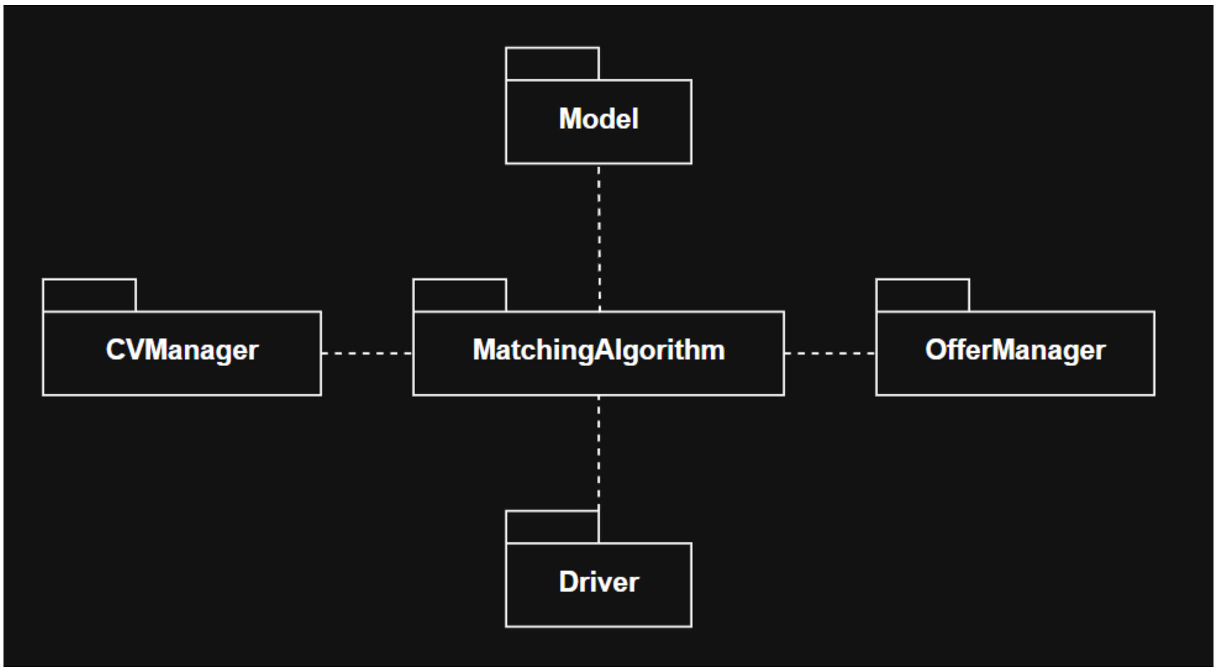


Figure 5.8: Recommendation

[F9] Suggestion

The Driver simulates the suggestion generation process. The SuggestionManager analyzes student CVs through CVManager and company offers through OfferManager to generate improvement suggestions for both. The Model stub simulates database operations for storing and retrieving suggestion data. Integration testing verifies the proper analysis of CVs and offers, generation of relevant suggestions for improvement, and correct storage of suggestion data.

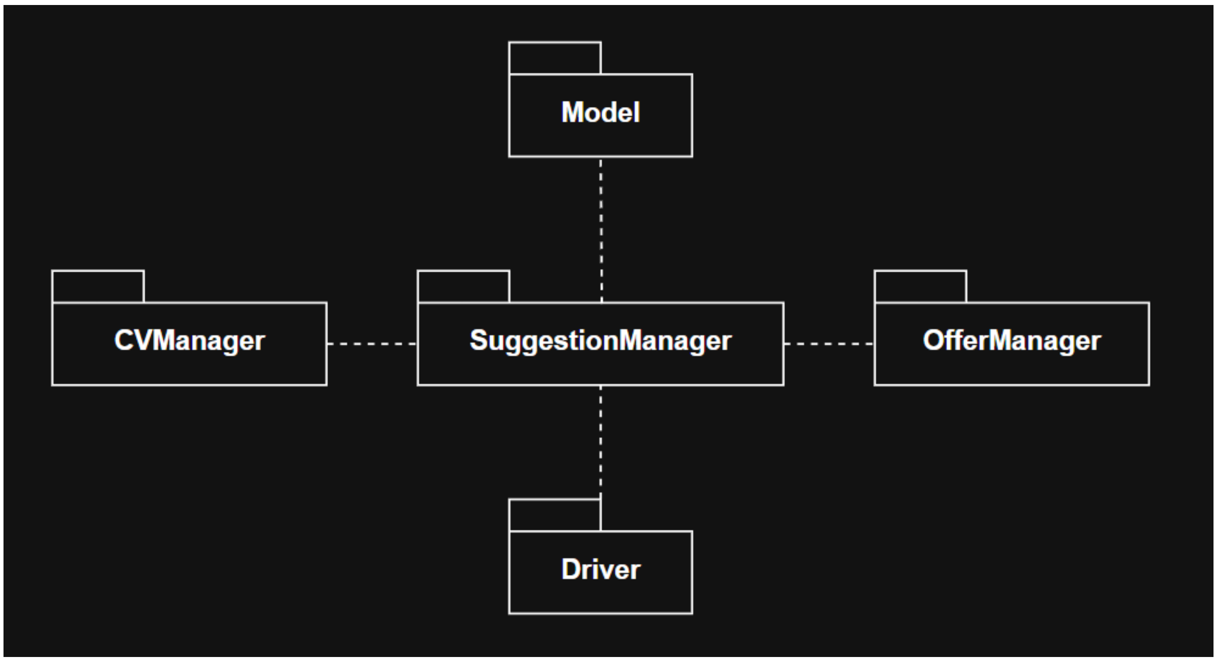


Figure 5.9: Suggestion

5.4. System testing

To make sure the S&C Platform works well, we need to check that all features made by the development team meet the requirements. During the development, each part of the platform should be tested carefully to make sure it works correctly. Sometimes, some parts can't be tested alone, so we will use stubs and drivers as temporary solutions until everything is ready.

Once each part works, we'll put it all together to form the full platform. Then, a complete test will be done to check if the platform meets the required goals listed in the RASD document. This testing will include developers, universities, students, and companies to make sure it is checked from all points of view.

Here are the types of tests we will do on the S&C Platform:

- **Functional Testing**: This test checks if the platform's features work as they should. For example:
  - Make sure students can sign up, log in, and look at internship offers.
  - Check that companies can make offers and manage applications.
  - Ensure universities can track complaints and manage internships.

- **Performance Testing**: This test checks if the platform works well without problems like slow loading or using too many resources. It helps find any issues that could slow down the platform. For this test:
  - We will simulate how students, companies, and universities normally use the platform.
  - Measure the platform's performance and see if we can make it better.
- **Usability Testing**: This test checks if the S&C Platform is easy to use. Real users like students, companies, and universities will be observed as they use the platform. Their feedback will help improve the platform's design and user experience.
- **Load Testing**: This test checks if the platform can handle many users at once without problems like crashes or slowdowns. It will check:
  - Can the platform handle many students logging in at the same time?
  - Can it manage many companies searching for offers or submitting applications?
  - Can universities handle many complaints or internships at once?

  The platform will be tested with more users over time to make sure it works well even under heavy use.
- **Stress Testing**: This test checks how the platform handles difficult situations, like when resources (such as memory) are low or used up. It will test:
  - What happens when the system has too many users at once?
  - Can the platform handle sudden spikes, like many students applying for internships at the same time?
  - Does the platform recover well without losing any data or causing major problems?

5.5. Additional specifications on testings

During the development of the S&C Platform, it is important to regularly get feedback from users and stakeholders. This should happen every time a new feature is added or changed. This helps make sure the platform meets the needs of the users.

In the alpha testing phase, it's important to check how satisfied the testers are with the platform. To get good feedback, it can be helpful to include experts, like psychologists, who can create the right questions to ask the testers. The alpha test helps find any problems or bugs before moving on to beta testing.

Beta testing is when real users—students, companies, and universities—use the platform in a real environment. This is important to find any issues that weren't noticed in earlier tests. It's the final check before the platform is fully released.

Even after the platform is released, testing is still important. Developers need to keep track of any issues that come up by looking at logs from the system. This helps them fix problems and improve the platform.

6 Effort Spent

- YE Yumeng

Chapter	Effort (in hours)
1	1
2	15
3	0
4	9
5	2

- Léo APPOURCHAUX

Chapter	Effort (in hours)
1	1
2	15
3	2
4	3
5	6