POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Students & Companies

DD Document
Software Engineering 2 Project

Authors: **Paone A. Pasqual M. Restelli M.**

Advisor: Prof. Camilli Matteo
Academic Year: 2024-25

Contents

1 Introduction

1.1. Purpose

This document represents the Design Document (DD). The purpose of the DD is to provide overall guidance to the architecture of the software product discussed in the RASD.

It contains the description of the architecture chosen to design the Students&Companies. Furthermore, it describes how the UI has to look like, and the map from the requirements defined in the RASD and the architectural components defined in this document. It also includes a set of design characteristics required for the implementations by introducing constraints and quality attributes, and it gives a presentation of the implementation, integration, and testing plan.

DD is addressed to the software development team who will have to implement the described system and it has the purpose to guide them through the development process.

1.2. Scope

Students & Companies (S&C) is a platform linking students with internship opportunities and corporations. Companies can advertise roles, get candidate recommendations, and send invitations. Students can search for roles actively or receive notifications for matches. Mutual interest brings to the selection processes, composed of interviews, and possibly to an internship being carried out. During internships, S&C gathers feedback, manages complaints, and involves universities when needed to resolve issues.

The platform is implemented using the 3-tier design pattern, a client-server software pattern that separates applications into three logical layers, each with distinct responsibilities. More implementation choices are covered in later sections of this document.

1.3. Definitions, Acronymous, Abbreviations

Abbreviation	Description
RASD	Requirements Analysis & Specification Document
DD	Design Document
G*	Goal
D*	Domain assumption
R*	Functional requirement
S&C	Students & Companies
STs	Students
COMs	Companies
UNs	Universities
UML	Unified Modelling Language
UI	User Interface

Table 1.1: List of Definitions, Acronymous, and Abbreviations

1.4. Revision History

- Version 1.0 (07/01/2025)

1.5. Reference Documents

The document is based on the following materials:

- IEEE Standard Documentation For DD
- 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

This document is composed of six chapters:

1. **Introduction:** This chapter describes the scope and purpose of this DD. It also includes the structure of the document and a set of definitions, acronyms, and abbreviations used.
2. **Architectural Design:** This chapter presents the architectural design choices. It includes an overview of the designed architecture, all the components, the interfaces, and the technologies used for designing the application. It also details the main functions of the interfaces and the processes in which they are utilized (Runtime view and component interfaces). Finally, it explains the design pattern chosen and recommended for system development.
3. **User Interface Design:** This chapter illustrates how the UI should look. It complements and expands upon the User Interfaces subsection in the RASD.
4. **Requirements Traceability.** This chapter explains how the requirements defined in the RASD are mapped to the design elements defined in this document.
5. **Implementation, Integration, and Test Plan.** This chapter outlines the planned order for implementing the system's subcomponents, integrating them, and testing the integration.
6. **Effort Spent.** This chapter provides information on the number of hours each group member has worked on this document.

2 | Architectural Design

2.1. Overview

We chose to design the Students&Companies (S&C) web application using a three-tier architecture with remote presentation. These three tier correspond to a client tier, application tier and data tier. The client (presentation) tier serves the needs of both students, companies and universities using the S&C platform. The application layer is the one that processes user request, performs statistical analysis and interacts with the internal data tier. The data tier is responsible for data management and storage. It handles all database operations, including data retrieval, updates, and management. To better improve and ensure load distribution and scalability, it was chosen to implement a load balancer between the client and application layer, in particular inside the web server.

A 3-tier architecture is suitable for a career service application as it makes the system easier to manage and scale. It ensures reliability through load balancing, which helps handle high traffic during peak usage. This architecture supports advanced features like job matching and analytics, integrates well with cloud services, and provides flexibility for future updates, such as adding mobile apps or AI tools. Its structure also improves security by isolating sensitive data in the database layer.

The S&C architecture specializes in three different groups of components each with its own characteristics and responsibilities:

1. **Presentation layer:** The Presentation (client) Tier is the application's front end, managing the user interface and interactions for data input and display. Its goal is to display data, collect inputs and provide a seamless user experience.
2. **Application layer:** It handles tasks like user authentication, recommendation matching algorithms, managing user accounts, handling notifications, and communication between the user interface and the database.
3. **Data layer**: Stores and manages critical data such as user profiles, student's CVs, project descriptions, and application history. It ensures data security and efficient

retrieval for real-time use in the application.

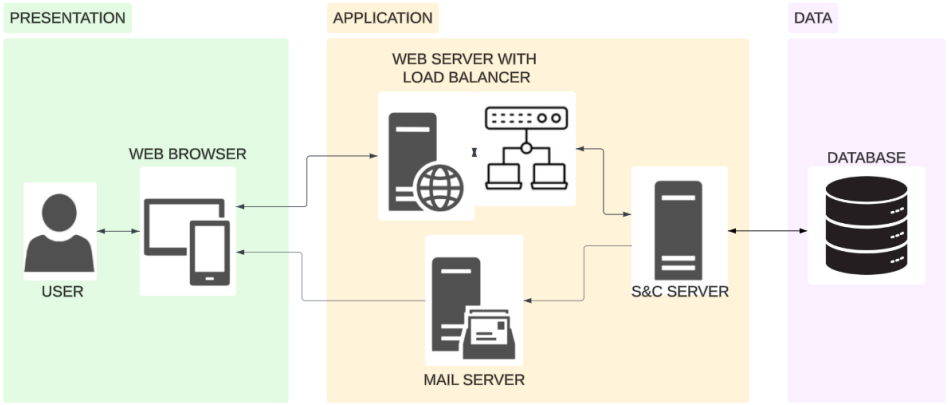

Figure 2.1: High level architecture

2.2. Component view

2.2.1. Component diagram

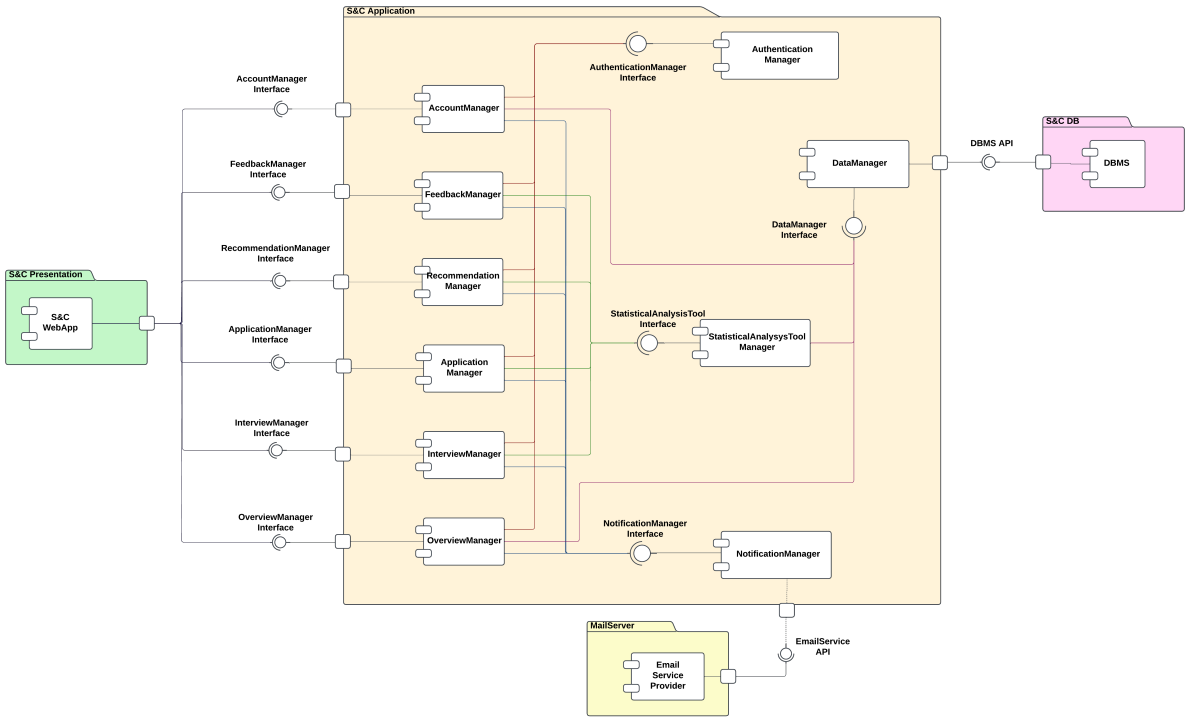

Figure 2.2: Component Diagram

Figure 2.2 presents a detailed diagram illustrating the components that constitute our application. Below is an explanation of each component and its purpose.

- **WebApp:** The WebApp serves as the primary interface for all users: students, companies, and universities, allowing them to interact with the application via their web browser. User requests are processed and sent to the appropriate backend components through the defined interfaces.
- **Account Manager:** The Account Manager is a backend module responsible for managing all aspects of account registration, login, and user data. It handles the creation of different user profiles and ensures the validity of the submitted data. The Account Manager communicates with the Statistical Analysis Tool Manager to notify it of any updates to user data and works with the Notification Manager to send notifications to users as needed. To save the new user's data, the Account Manager interacts with the Data Manager to store the information in the database. However, before performing any database operations, the Account Manager first requests permission from the Authentication Manager to ensure that the user is authenticated and authorized to perform the action.
- **Feedback Manager:**The Feedback Manager is responsible for gathering user feedback from both students and companies. It proactively requests feedback when needed through the Notification Manager, ensuring timely user engagement. This module collaborates with the Statistical Analysis Tool by providing updated feedback data, enabling continuous improvement of the recommendation system and analysis capabilities.
- **Recommendation Manager:** The Recommendation Manager is the module that is responsible for requesting and receiving internship matches between students and projects from the Statistical Analysis Tool. After obtaining the matches, it communicates the results to the Notification Manager, which then notifies the involved users about the recommended opportunities.
- **Application Manager:** The Application Manager handles student applications in three scenarios: when a student proactively applies for an internship they are interested in, when a student either accepts a recommendation, and when responds to an invitation from a company that has previously accepted the system's recommendation. After processing the applications, it communicates with the Notification Manager to notify both students and companies about the submitted applications. Additionally, it interacts with the Statistical Analysis Tool to provide data for further analysis and improvement of the recommendation system.

- **Interview Manager:** The Interview Manager oversees the entire interview and final selection process. It communicates with the Notification Manager to keep both students and companies informed about new developments throughout the process. Additionally, it interacts with the Statistical Analysis Tool to provide data, such as information on who was selected for the job, helping to refine the system and adjust parameters for future job recommendations.
- **Overview Manager:** The Overview Manager is responsible for monitoring ongoing internships. It communicates with the Notification Manager to keep universities, students, and companies involved in the project updated on progress. Additionally, it interacts with the Authentication Manager to obtain permission for accessing data, and then works with the Data Manager to insert and register the relevant updates.
- **StatisticalAnalysisTool Manager:** The Statistical Analysis Tool Manager is responsible for generating matches between students and internship projects based on student CVs and project descriptions. It also provides suggestions to students for improving their CVs and to companies for enhancing their project descriptions. These tasks are performed using a recommendation system powered by AI and machine learning. The tool is continuously refined through feedback from both students and companies about its previous recommendations. It communicates with the Data Manager to access the necessary data, with prior authentication provided by the Authentication Server.
- **Notification Manager:** The Notification Manager is responsible for sending notifications to all users across various channels. It interacts with the email service API to send email notifications and also delivers real-time updates directly within the web app. When delivering notifications through the app, it follows the chain of requests, processing them in reverse order to ensure that all relevant parties receive the appropriate updates.
- **Authentication Manager:** The Authentication Manager is responsible for managing access rights to data, ensuring that users can perform read or write operations based on their role and permissions.
- **Data Manager:** The Data Manager is responsible for managing and storing all data within the system. It handles data retrieval, updates, and ensures that information is accurately recorded and stored. The module interacts with other components to provide the necessary data for operations while maintaining data integrity and security.

- **Email Service Provider:** The Email Service Provider is an external module that interacts with the Notification Manager to send email notifications to users. It handles the delivery of emails for various events, such as application updates, interview schedules, and system alerts.
- **DBMS:** The DBMS (Database Management System) is responsible for managing and organizing the system's database. It handles data storage, retrieval, and manipulation, ensuring data integrity, security, and efficient access. The DBMS interacts with other modules to support operations such as user authentication, data analysis, and application management, storing and managing all relevant information within the system.

2.3. Deployment view

2.4. Runtime view

All the sequence diagrams described in the Requirements Analysis and Specification Document (RASD) will be elaborated in greater detail. In this expanded version, we will identify and specify the individual components of the application that are utilized to fulfill each specific functionality.

Here below we will include an explanation of how each application component contributes to the implementation of the corresponding functionality. By doing so, we hope to provide a clearer understanding of the interaction between various components and their roles in achieving the described features and requirements.

Spiegazione su di cosa si occupa ogni component e poi tutti i sequence diagram..

UC1: Student creates an account WebApp, AccountManager, Authentication manager, DataManager, DBMS, NotificationManager, EmailServiceProvider

UC2: Company creates an account WebApp, AccountManager, Authentication manager, DataManager, DBMS, NotificationManager, EmailServiceProvider

UC3: Student searches and applies for an internship WebApp, AccountManager, Authentication manager, ApplicationManager, StatisticalAnalysisToolManager, DataManager, DBMS, NotificationManager, EmailServiceProvider

UC4: Student recieves an internship recommendation WebApp, AccountManager, Authentication manager, RecommendationManager, ApplicationManager, StatisticalAnalysisToolManager, DataManager, DBMS, NotificationManager, EmailServiceProvider

UC5: Company recieves a student's recommendation WebApp, AccountManager, Authentication manager, RecommendationManager, ApplicationManager, StatisticalAnalysisToolManager, DataManager, DBMS, NotificationManager, EmailServiceProvider

UC6: Interview WebApp, AccountManager, Authentication manager, ApplicationManager, StatisticalAnalysisToolManager, InterviewManager, DataManager, DBMS, NotificationManager, EmailServiceProvider

UC7: User leaves feedback WebApp, AccountManager, Authentication manager, FeedbacknManager, StatisticalAnalysisToolManager, DataManager, DBMS, NotificationManager, EmailServiceProvider

UC8: Student recieves suggestion on CV WebApp, AccountManager, Authentication manager, StatisticalAnalysisToolManager, DataManager, DBMS, NotificationManager, EmailServiceProvider

UC9: Company recieves suggestion on project description WebApp, AccountManager, Authentication manager, StatisticalAnalysisToolManager, DataManager, DBMS, NotificationManager, EmailServiceProvider

UC10: User makes a complaint WebApp, AccountManager, Authentication manager, OverviewManager, StatisticalAnalysisToolManager, DataManager, DBMS, NotificationManager, EmailServiceProvider

2.5. Component interfaces

The Component Interfaces here described are the ones exposed in the components. Only the most relevant method parameters and methods are shown.

- **Account Manager Interface**
  - **registerUserAsStudent(string Email, string Name, data BirthDate, string LocationBirth, string Password) : boolean AccountCreated**

    This method takes all the data in input and contacts the DataManager to register a new StudentUser in the system.
  - **registerUserAsCompany(string Email, string CompanyName, string LocationAddress, string Password) : boolean AccountCreated**

    This method takes all the data in input and contacts the DataManager to register a new CompanyUser in the system.

- **registerUserAsUniversity(string Email, string UniversityName, string LocationAddress, string Password) : boolean AccountCreated** This method takes all the data in input and contacts the DataManager to register a new UniversityUser in the system.
- **loginUser(string Email, string Password) : boolean LoginSuccessful**

  This method takes the user's email and password, check whetere they are registered and if they are it logs them into their account
- **insertCv(CV Cv): boolean Inserted**

  This method allows a StudentUser to upload their Cv on their profile.
- **deleteCv(): boolean Deleated**

  This method allows a StudentUser to delete their Cv from their profile.
- **insertProjectDescription(string ProjectDescription): booleand Inserted** This method allows a CompanyUser to upload a new project description on their profile.
- **deleteProjectDescription(string ProjectName): booleand Deleted**

  This method allows a CompanyUser to delete a project description from their profile.
- **modifyAttribute(string AttributeName, string NewValue): boolean Modified**

  This method allows users to modify any data present on their profile.

- **Authentication Manager Interface:**
  - **authenticateUser(string Email, string Password) : boolean IsAuthenticated** This method verifies the provided email and password against the stored credentials. If the credentials are valid, it returns 'true'; otherwise, it returns 'false'.
  - **authorizeAction(string Email, string Action) : boolean IsAuthorized** This method determines whether the specified user has the necessary permissions to perform the given action. It returns 'true' if the action is allowed, otherwise 'false'.
- **Feedback Manager Interface**

- **requestFeedback(string UserType, string FeedbackType): boolean Requested**

  This method sends a request for feedback to the specified user type (e.g., student or company) using the Notification Manager. The FeedbackType determines the nature of the feedback requested.

- **submitFeedback(string UserId, string FeedbackContent): boolean Submitted**

  This method allows users to submit their feedback. The content is processed and forwarded to the Statistical Analysis Tool for further use.

- **getFeedbackHistory(string UserId): list FeedbackList**

  This method retrieves the history of feedback provided by a specific user for review or analysis purposes.

- **updateStatisticalAnalysisTool(string FeedbackData): boolean Updated**

  This method sends the collected feedback to the Statistical Analysis Tool, enabling continuous refinement of the recommendation system.

- **notifyFeedbackUpdate(string UserId, string FeedbackStatus): boolean Notified**

  This method communicates with the Notification Manager to inform users about the status or impact of their feedback submissions.

- **Application Manager Interface:**
  - **submitApplication(string UserId, string InternshipId): boolean Submitted**

    This method allows a student to submit an application for a specified internship. It processes the application and records it in the system.

  - **acceptRecommendationStudent(string UserId, string RecommendationId): boolean Accepted**

    This method processes a student's acceptance of a recommendation provided by the system and creates an application for the corresponding internship.

  - **acceptRecommendationCompany(string UserId, string RecommendationId): boolean Accepted**

This method processes a company's acceptance of a recommendation provided by the system and creates an invitation for the Student for the corresponding internship.

- **respondToInvitation(string UserId, string InvitationId): boolean Responded**

  This method allows a student to respond to an invitation from a company and generates an application if the invitation is accepted.

- **notifyApplicationUpdate(string UserId, string CompanyId, string ApplicationStatus): boolean Notified**

  This method communicates with the Notification Manager to inform both the student and the company about updates or changes to the application status.

- **sendDataToStatisticalAnalysisTool(string ApplicationData): boolean Sent**

  This method sends processed application data to the Statistical Analysis Tool to assist in refining the recommendation system and improving future matches.

- **Interview Manager Interface:**
  - **createAssessmentForum(string CompanyId, string UserId, string AssessmentDetails): boolean Created**

    This method allows a company to create a forum for conducting a prior assessment with a student. The forum includes assessment details and discussion tools for evaluations.

  - **resolveAssessment(string ForumId, string ResolutionDetails): boolean Resolved**

    This method finalizes the outcome of a prior assessment conducted in the forum and records the resolution in the system.

  - **evaluateAssessment(string ForumId, string EvaluationFeedback): boolean Evaluated**

    This method records feedback and evaluations on the assessment process, providing insights for both the student and the company.

  - **scheduleInterview(string UserId, string CompanyId, datetime InterviewDate): boolean Scheduled**

This method schedules an interview between a student and a company and records the details in the system.

- **updateInterviewStatus(string InterviewId, string Status): boolean Updated**

  This method allows updating the status of an interview (e.g., scheduled, completed, canceled) and ensures the system reflects the current progress.

- **finalizeSelection(string CompanyId, string UserId): boolean Finalized**

  This method records the final selection decision made by the company and updates the relevant data.

- **notifyInterviewUpdate(string UserId, string CompanyId, string NotificationMessage): boolean Notified**

  This method communicates with the Notification Manager to keep both the student and the company updated about any changes or developments in the interview process.

- **sendSelectionDataToStatisticalAnalysisTool(string SelectionData): boolean Sent**

  This method forwards data on the final selection process to the Statistical Analysis Tool to refine future recommendations and adjust job parameters.

- **Overview Manager Interface:**
  - **trackInternshipProgress(string InternshipId): map ProgressData**

    This method retrieves the current progress data for a specific internship, including details about the student, company, and related activities.

  - **updateInternshipStatus(string InternshipId, string StatusUpdate): boolean Updated**

    This method updates the status of an ongoing internship (e.g., in progress, paused, completed) and records the changes in the system.

  - **logInternshipUpdate(string InternshipId, string UpdateDetails): boolean Logged**

    This method registers detailed updates regarding the internship in the database. It communicates with the Authentication Manager for permission and the Data

Manager to store the data.

- **uploadComplaint(string InternshipId, string UserId, string ComplaintDetails): boolean Uploaded**

  This method allows students or companies to file a complaint about the internship. The complaint is logged in the system for review and resolution by the university.

- **resolveComplaint(string InternshipId, string ComplaintId, string ResolutionDetails): boolean Resolved**

  This method allows universities to resolve complaints raised during the internship. Resolutions are documented and communicated back through the app.

- **reportComplaintOutcome(string InternshipId, string ComplaintId): boolean Reported**

  This method updates the system with the outcome of externally resolved complaints, ensuring all data remains centralized and accessible.

- **manageFeedback(string InternshipId, string FeedbackContent): boolean Managed**

  This method processes feedback submitted by students or companies about the internship and stores it in the system for future reference.

- **contactParticipant(string InternshipId, string ParticipantId, string MessageContent): boolean Contacted**

  This method enables the university to directly contact a student or company involved in the internship by sending a message through the app.

- **notifyProgressUpdate(string UserId, string NotificationMessage): boolean Notified**

  This method interacts with the Notification Manager to inform universities, students, and companies about any new updates or changes in the internship.

- **sendProgressDataToStatisticalAnalysisTool(string ProgressData): boolean Sent**

  This method provides ongoing internship data to the Statistical Analysis Tool, enabling analysis and improvement of internship processes.

- **StatisticalAnalysisTool Manager Interface:**

- **generateMatches(string StudentId, string ProjectId): list Matches**

  This method generates a list of potential matches between students and internship projects based on student CVs and project descriptions using the recommendation system.

- **suggestCvImprovements(string StudentId, CV Cv): list Suggestions**

  This method analyzes a student's CV and provides personalized suggestions for improvements to enhance their profile and compatibility with internship opportunities.

- **suggestProjectImprovements(string CompanyId, string ProjectDescription): list Suggestions**

  This method evaluates a company's project description and provides recommendations for making it more appealing and clear to potential candidates.

- **refineRecommendations(string FeedbackData): boolean Refined**

  This method processes feedback received from students and companies about previous recommendations to update and improve the recommendation algorithm.

- **analyzeSelectionData(string SelectionData): boolean Analyzed**

  This method evaluates data on selected candidates to refine parameters and optimize future recommendations.

- **fetchData(string DataRequest): map Data**

  This method interacts with the Data Manager to retrieve necessary data for analysis, ensuring access permissions are obtained through the Authentication Server.

- **updateRecommendationParameters(string Parameters): boolean Updated**

  This method adjusts the recommendation system's parameters based on new data or trends observed during analysis.

- **notifyFeedbackProcessed(string UserId, string NotificationMessage): boolean Notified**

  This method interacts with the Notification Manager to inform users that their feedback has been processed and incorporated into system improvements.

- **Notification Manager Interface:**
  - **sendAccountNotification(userEmail: String, notificationType: String, message: String): String message**

    This method is used to send notifications related to account activities, such as registration confirmations, password updates, or account deactivations. It's typically called by the Account Manager. The method takes the userEmail (the ID of the user to notify), a notificationType (like "Registration"), and a message containing the notification content. The message is both sent by email through the notification manager and The Account Manager receives the formatted message through the backward chain and passes it along to the WebApp, where it's displayed to the user in the notification page.
  - **sendFeedbackNotification(userEmail: String, feedbackType: String, message: String): String message**

    This method is used to send notifications requesting or acknowledging user feedback, such as a prompt to review an internship or provide suggestions for improvement. It's typically called by the Feedback Manager. The method takes the userEmail (the email of the user to notify), a feedbackType (like "InternshipFeedback"), and a message containing the notification content. The message is sent by email through the Notification Manager, and the Feedback Manager receives the formatted message through the backward chain and passes it along to the WebApp, where it is displayed to the user to encourage engagement.
  - **sendRecommendationNotification(userEmail: String, recommendationDetails: String): String message**

    This method is used to send notifications about internship recommendations to students or companies, informing them about relevant matches. It's typically called by the Recommendation Manager. The method takes the userEmail (the email of the user to notify) and recommendationDetails (like "You have been recommended for the Software Engineer position at TechCorp"). The message is sent by email through the Notification Manager, and the Recommendation Manager receives the formatted message through the backward chain and passes it along to the WebApp, where the user is notified about the recommendation on their dashboard.
  - **sendApplicationNotification(userEmail: String, applicationStatus: String,**

message: String): String message

This method is used to notify users about updates related to their internship applications, such as submission confirmations, acceptance, or rejections. It's typically called by the Application Manager. The method takes the userEmail (the email of the user to notify), an applicationStatus (like "Submitted" or "Accepted"), and a message detailing the update. The message is sent by email through the Notification Manager, and the Application Manager receives the formatted message through the backward chain and passes it along to the WebApp, where the user sees the application update on the notification page.

- **sendInterviewNotification(userEmail: String, interviewDetails: String): String message**

  This method is used to notify users about interview-related updates, such as scheduling, rescheduling, or cancellations. It's typically called by the Interview Manager. The method takes the userEmail (the email of the user to notify) and interviewDetails (like "Your interview is scheduled for July 5th at 3:00 PM"). The message is sent by email through the Notification Manager, and the Interview Manager receives the formatted message through the backward chain and passes it along to the WebApp, where the user sees the interview details on their notification page.

- **sendOverviewNotification(userEmail: String, progressUpdate: String): String message**

  This method is used to notify users about updates on the progress of ongoing internships, ensuring students, universities, and companies are informed about important milestones. It's typically called by the Overview Manager. The method takes the userEmail (the email of the user to notify) and a progressUpdate (like "Your student has reached 80% of the internship goals"). The message is sent by email through the Notification Manager, and the Overview Manager receives the formatted message through the backward chain and passes it along to the WebApp, where it's displayed on the notification page for relevant users.

- **sendGeneralNotification(userEmail: string, toUserIds: List[String], message: String): String message**

  This method is used for sending generic notifications that don't fit into other specific categories. It supports notifications across various channels and can

target multiple users simultaneously. It's called by any module requiring an ad hoc notification. The method takes a list of toUserEmails (the emails of the users to notify), a channel (like "email" or "app"), and a message containing the content to be sent. The message is sent through the specified channel by the Notification Manager, and the calling module receives the formatted message through the backward chain and passes it along to the WebApp, where it's displayed in the user's notification area.

- **Data Manager:**
  - **saveRecord(string TableName, map<string, string> Data) : boolean RecordSaved**

    This method saves a new record in the specified table. It returns 'true' if the record is successfully stored.
  - **fetchRecord(string TableName, string Condition) : map<string, string> FetchedRecord**

    This method retrieves a single record from the specified table that matches the provided condition. Returns the record as a key-value map.
  - **fetchAllRecords(string TableName, string Condition) : list<map<string, string» RecordsList**

    This method retrieves all records from the specified table that match the provided condition. Returns the data as a list of key-value maps.
  - **updateRecord(string TableName, map<string, string> Updates, string Condition) : boolean RecordUpdated**

    This method updates records in the specified table based on the condition provided. Returns 'true' if the update is successful.
  - **deleteRecord(string TableName, string Condition) : boolean RecordDeleted**

    This method deletes records from the specified table that match the provided condition. Returns 'true' if the deletion is successful.
  - **provideDataAccess(string ModuleName, string DataRequest) : map<string, string> AccessedData** This method grants secure data access to a specific module, returning the requested data while enforcing access control policies.
- **Email Service Provider:**

- **sendEmail(string SenderEmail, string RecipientEmail, string Subject, string Body) : boolean EmailSent, string EmailId**

  This method sends an email to the specified recipient with the provided subject and body content. Returns ‘true‘ if the email is successfully sent.

- **sendTemplatedEmail(string SenderEmail, string RecipientEmail, string TemplateName, map<string, string> TemplateData) : boolean EmailSent, string EmailId**

  This method sends an email using a predefined template. It takes the template name and a mapping of dynamic data to populate the template fields.

- **trackEmailDelivery(string EmailId) : string DeliveryStatus**

  This method checks the delivery status of a previously sent email using its unique email ID. It returns statuses such as "Delivered," "Failed," or "Pending."

- **configureEmailSettings(string SMTPServer, int Port, string SenderEmail, string SenderPassword) : boolean SettingsConfigured**

  This method configures the SMTP settings for sending emails, including the server address, port, sender email, and authentication details.

- **validateEmailAddress(string EmailAddress) : boolean IsValid**

  This method checks whether the given email address is in a valid format and can be used for sending emails.

- **resendFailedEmails(string[] FailedEmailIds) : boolean Resent** This method retries sending emails that previously failed to be delivered. Returns ‘true‘ if all retries are successful.

- **DBMS:**

  - **postData(string TableName, map<string, string> Data) : boolean IsStored**

    This method inserts the provided data into the specified table within the database. Returns ‘true‘ if the operation is successful.

  - **getData(string Query) : list<map<string, string» RetrievedData**

    This method executes the provided query to fetch data from the database. It returns the results as a list of records.

- **updateData(string TableName, map<string, string> Updates, string Condition) : boolean IsUpdated**

  This method updates records in the specified table based on the condition provided. Returns ‘true‘ if the update is successful.

- **deleteData(string TableName, string Condition) : boolean IsDeleted**

  This method removes records from the specified table that match the provided condition. Returns ‘true‘ if the deletion is successful.

- **manageAccessControl(string UserId, string[] Permissions) : boolean AccessManaged**

  This method updates access control policies for a specific user by assigning or revoking permissions.

2.6. Selected architectural styles and patterns

2.7. Other design decisions

3 | User Interface Design

This section presents screenshots of the Students and Companies app, highlighting its interface and key features, giving a clear view of how users interact with the platform.

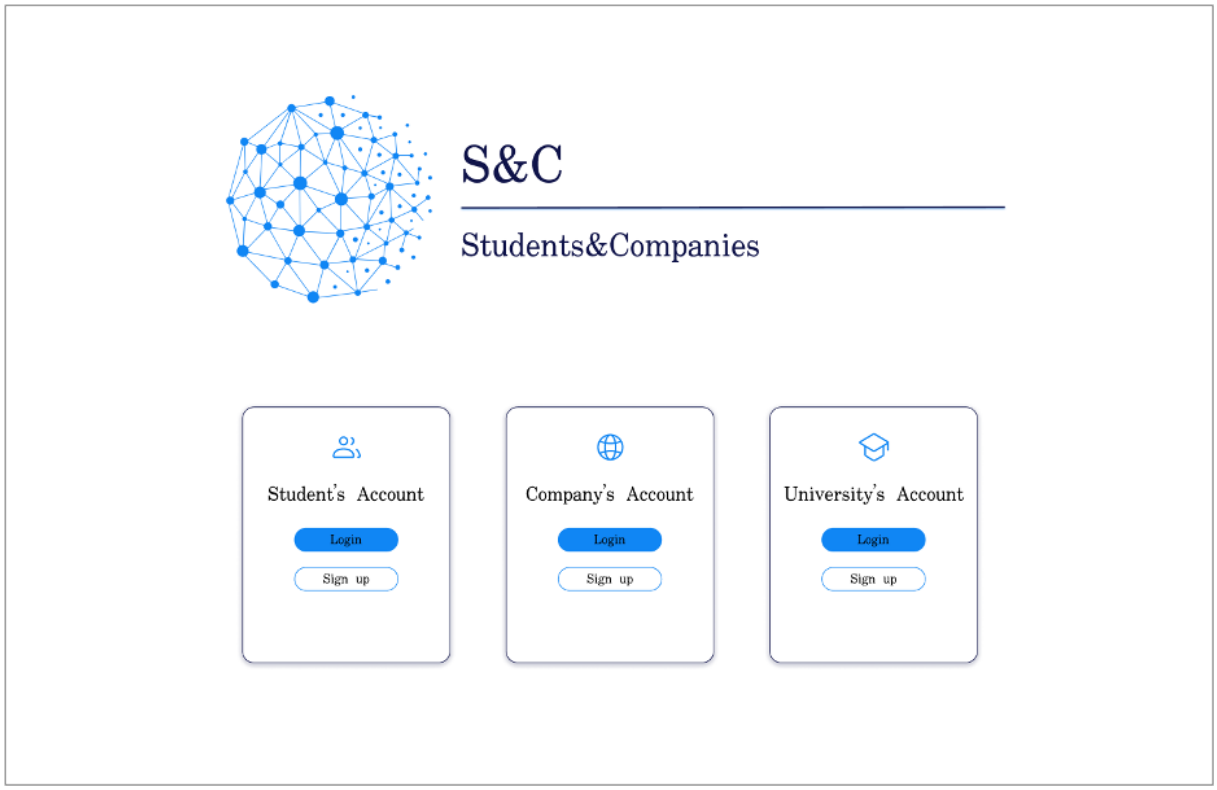

Figure 3.1: Login Page

The Login Page allows users to choose whether to create a new account or log into an existing one. Users can select their role (student, company, or university) to access the appropriate features of the app.

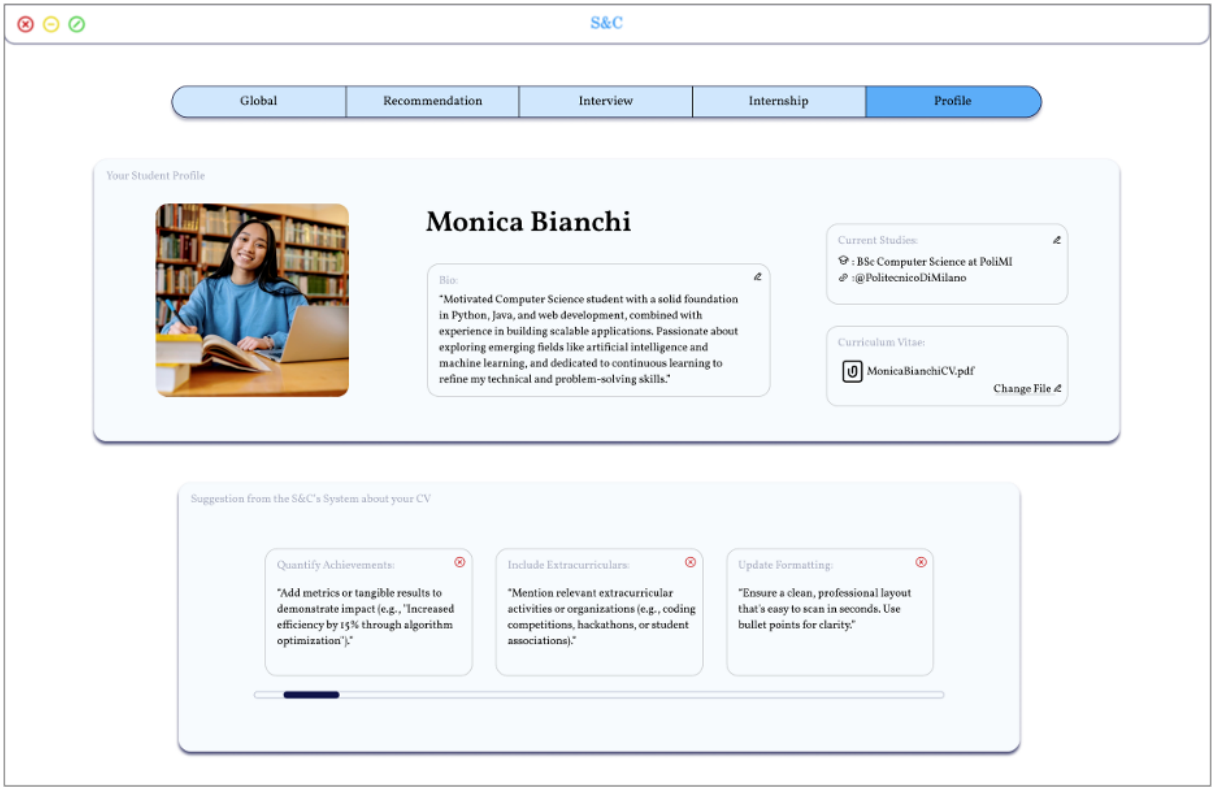

Figure 3.2: Student's Profile Page

The Student's Profile Page displays the student's photo, bio, and education information, with options to view and update this information. It also allows the student to upload, delete, or re-upload their CV. Below the profile details, the page shows personalized suggestions from the Students and Companies system, offering hints to improve their CV.

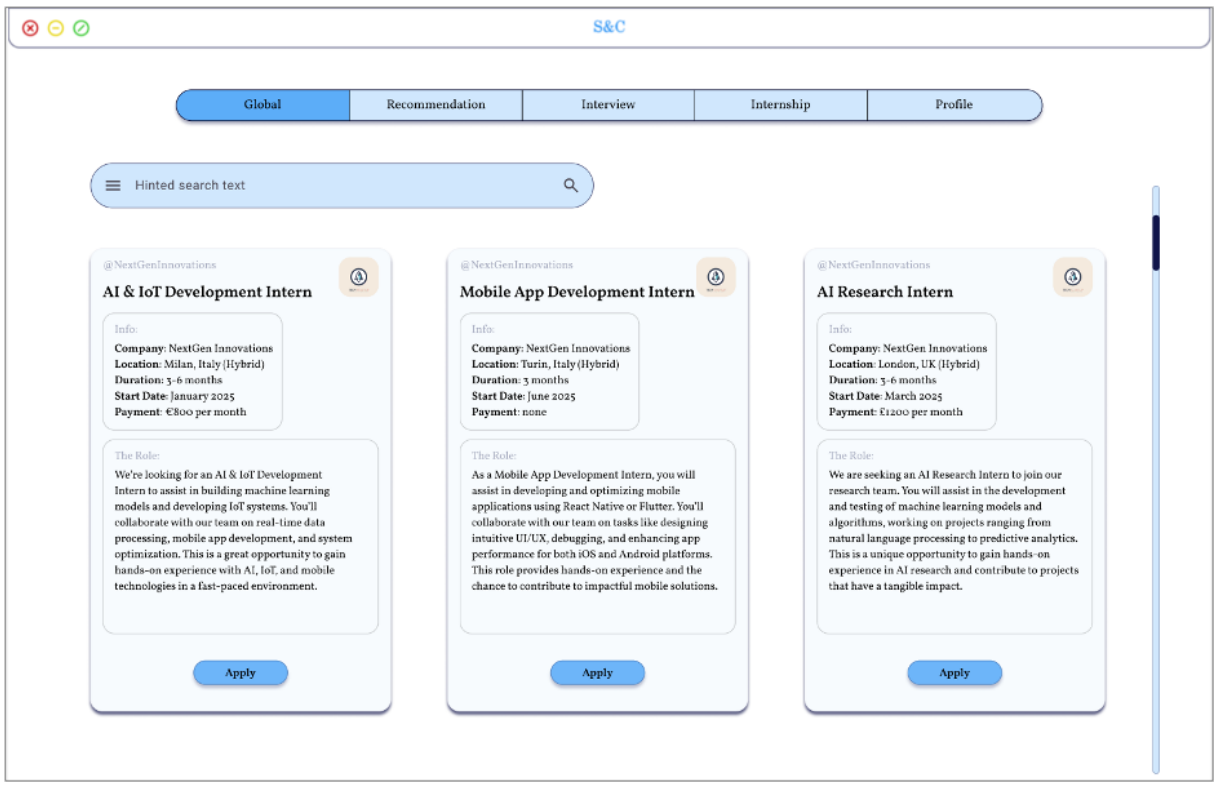

Figure 3.3: Student's Global Searching Page

The Student's Global Searching Page allows students to browse all available internship project descriptions. They can proactively search for internships that match their interests and skills, using keywords or specific parameters. Students also have the option to apply directly to internships that suit them best.

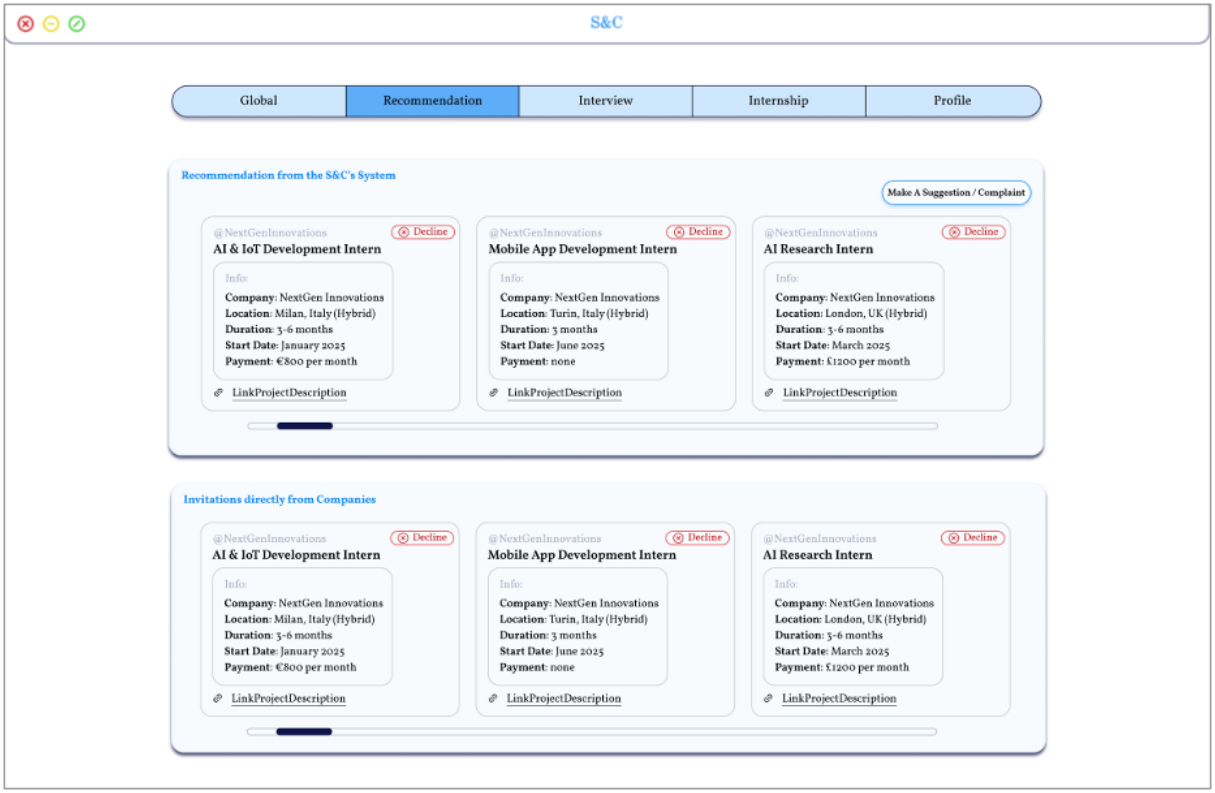

Figure 3.4: Student's Recommendations Page

The Student's Recommendations Page displays internship opportunities suggested by the Students and Companies system, tailored to the student's profile. The student can accepts the recommendation applying for the position. Below these recommendations, the student can also view invitations sent directly from companies. In both cases, the student can apply directly to the internships.

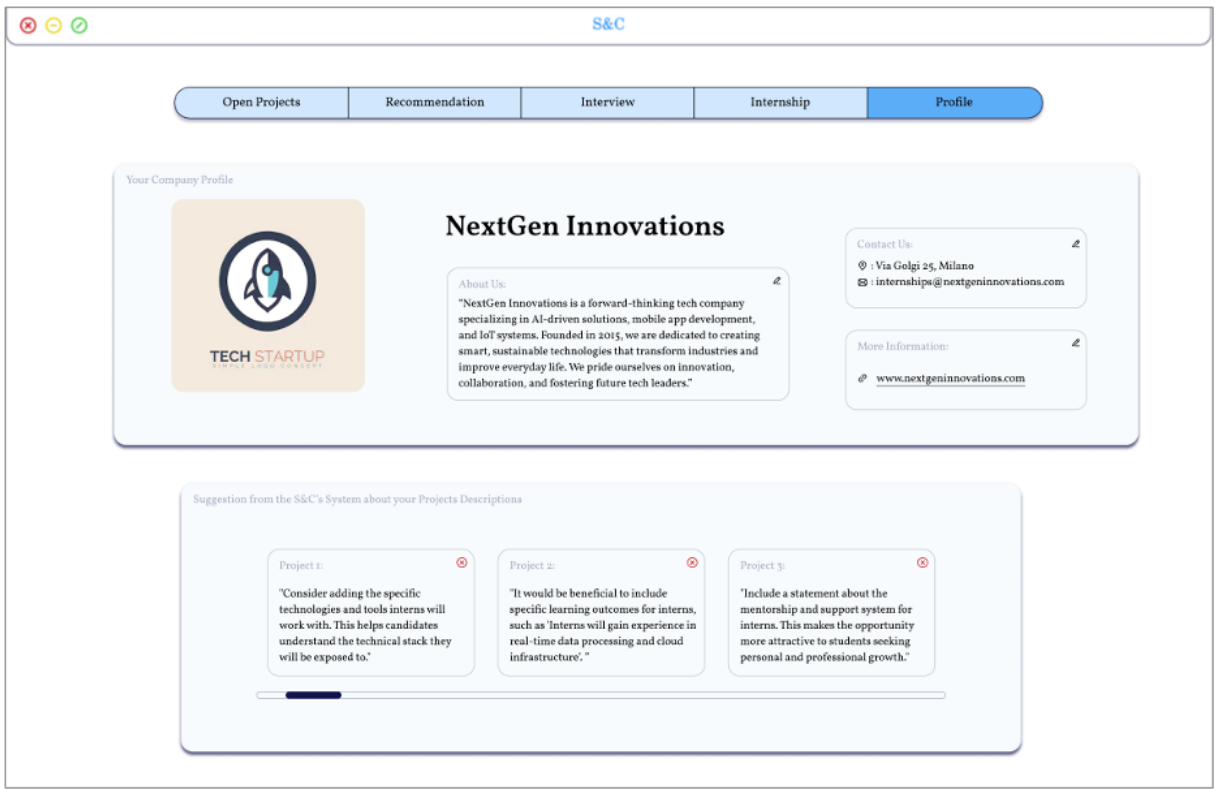

Figure 3.5: Company's Profile Page

The Company's Profile Page displays the company's bio and all the information they've provided about themselves, with the option to edit and update this content. Below, the page shows suggestions from the Students and Companies system on how to improve the descriptions of each posted internship project, making them more appealing and informative.

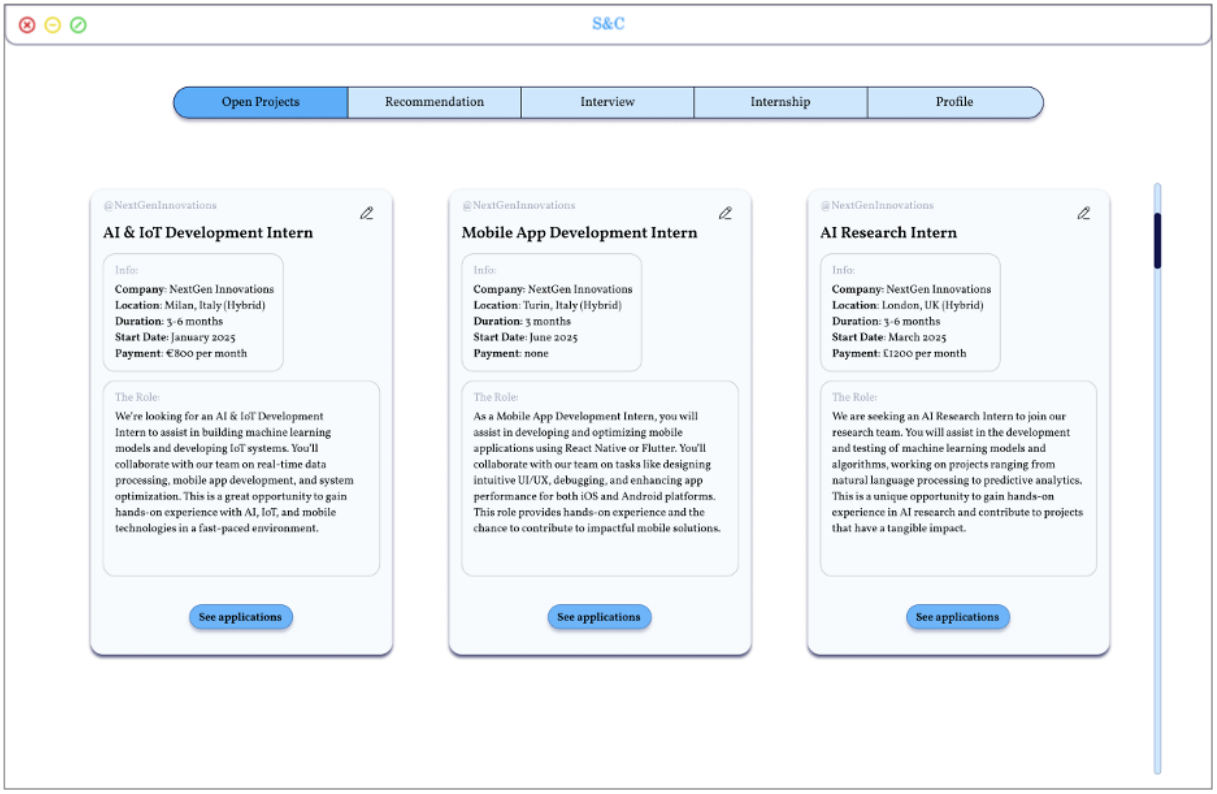

Figure 3.6: Company's open projects Page

The Company's Open Projects Page displays all the internship project descriptions that the company has published. It offers the option to modify these descriptions and provides direct links to view the applications received for each internship.

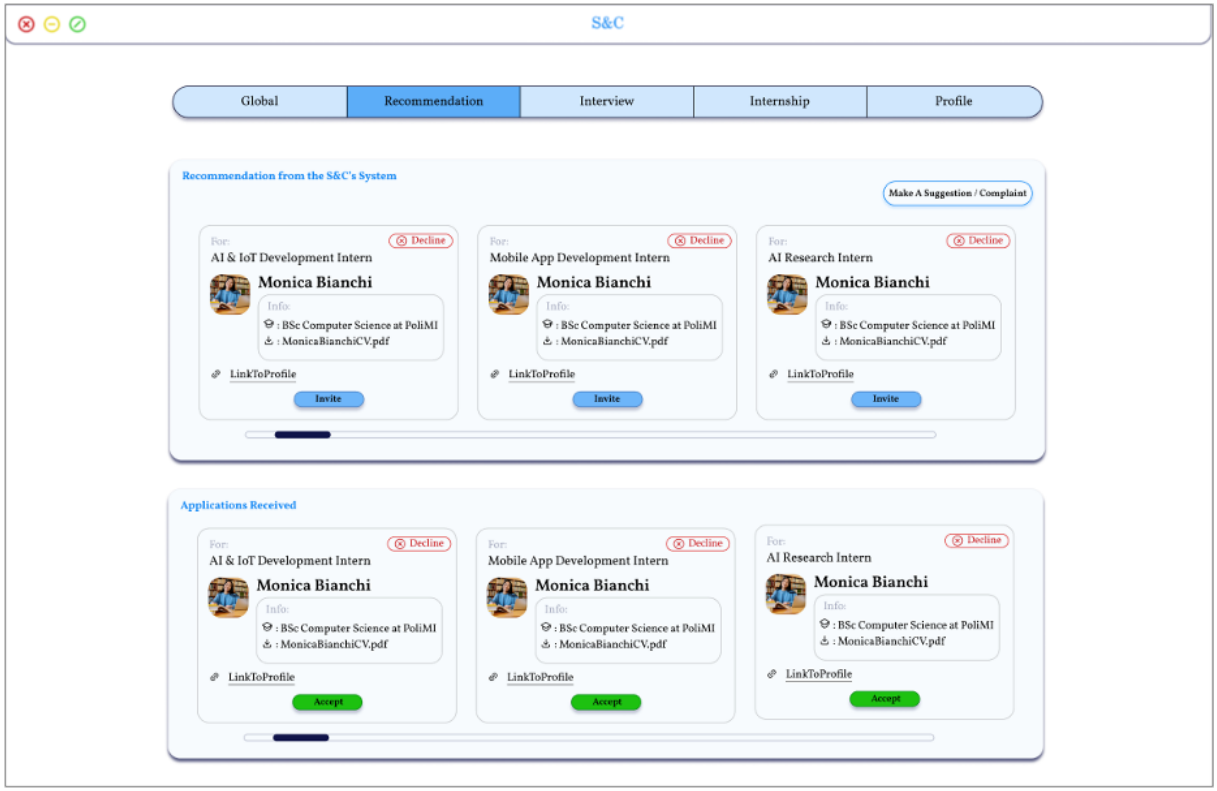

Figure 3.7: Company's Recommendations Page

The Company's Recommendations Page displays a list of recommended candidates from the system, providing a preview of their profiles with a direct link to view more details. The company can accept recommendations and invite students to apply. Below the recommendations, the page also shows all the applications the company has received from students.

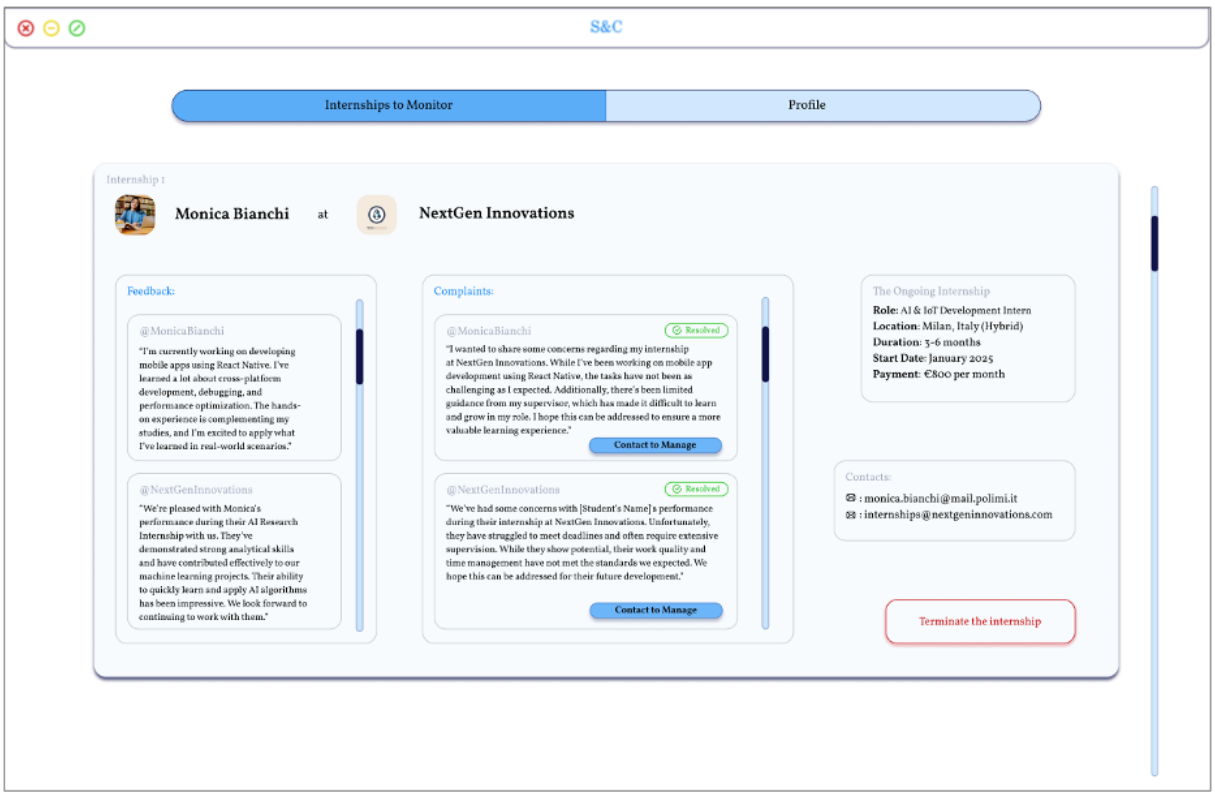

Figure 3.8: University's ongoing internship Monitoring Page

The University's Ongoing Internship Monitoring Page provides a comprehensive overview of all ongoing internships, including details about the student, company, role, and relevant contact information. The page features sections for both feedback and complaints, which can be submitted by either the student or the company. Complaints can be managed within the app or resolved externally, with the option to report the resolution back on the platform.

4 | Requirements Traceability

5 | Implementation, Integration, and Testing

This chapter explains how the platform described will be implemented and tested. The primary aim of testing is to identify and address the majority of the bugs in the code produced by the development team. Additionally, a detailed description of the integration of components within the code is provided in Section 5.2. Section 5.1 presents an overview of the most important implementation strategies used in the project.

5.1. Implementation Plan

This section describes the strategies for implementing, integrating, and testing the various components of the system. The implementation approach combines the advantages of both bottom-up and thread-based strategies.

A thread-based strategy is functional because it makes progress visible to users and stakeholders. It allows for fewer drivers than expected but introduces greater complexity during integration.

The top-down methodology will be employed by first designing a basic sketch of the system. Then, more complex functionalities will be added incrementally as validated thread units. This strategy enables different teams to work in parallel, accomplishing tasks independently. Once validated, these units will be integrated into the overall software architecture.

5.1.1. Features Identification

The implementation process begins with identifying the key features required for the system. These features align with the goals specified in the RASD document and correspond to the primary system components outlined in Figure 2.2. The key features include:

write features

These components collectively form the core features and capabilities of the platform.

5.2. Component Integration and Testing

Integrating the components of the system is a critical phase that ensures all modules work together seamlessly. The integration process follows these steps:

1. **Unit Validation:** Each component is tested individually to ensure it performs as intended. Stubs and drivers are employed to simulate missing parts during testing.
2. **Thread Integration:** Components validated in isolation are integrated incrementally. Threads representing functional workflows (e.g., user registration, internship posting) are built and tested sequentially.
3. **System Assembly:** Once all threads are validated, they are integrated into the overall system. Compatibility and interactions between threads are thoroughly tested to ensure no conflicts arise.
4. **Final Testing:** After the entire system is assembled, it undergoes a full round of testing, including functional, performance, usability, load, and stress testing, to verify its readiness for deployment.

This systematic approach to integration and testing minimizes errors and ensures that the platform meets its specified requirements.

5.3. System Testing

The CKB platform must be thoroughly tested to ensure that the implemented functionalities align with expectations. During development, each component will be tested individually. However, not all components can be tested in isolation from the entire architecture; therefore, stubs and drivers will be used to replace missing components as needed. Once a thread unit is tested and validated, it will be integrated into the software architecture. When the entire architecture is complete, comprehensive testing will be conducted.

The primary objective of system testing is to verify that the platform meets the functional and non-functional requirements specified in the RASD document. In this process, participation from all stakeholders, including developers and end-users, is essential. The testing steps include:

- **Functional Testing:** This test verifies whether the functional requirements are fulfilled. The most effective way to conduct functional testing is to execute the

software as described in the RASD use cases and ensure they are satisfied.

- **Performance Testing:** The goal of this test is to identify bottlenecks that may affect response time, utilization, and throughput. It can also detect inefficient algorithms, hardware/network issues, and optimization opportunities. This test involves loading the system with the expected workload, measuring performance, and comparing results to identify areas for improvement.
- **Usability Testing:** This method evaluates the usability of a website, application, or other digital product by observing real users as they attempt to complete tasks.
- **Load Testing:** Load testing is conducted to identify potential issues such as memory leaks, buffer overflows, or memory mismanagement. This test determines the upper limits of the system's components by increasing the workload until the system can no longer sustain it for a pre-established duration.
- **Stress Testing:** This test ensures that the system can recover gracefully after failure. For stress testing, system resources may be increased or decreased to evaluate the system's resilience.

5.3.1. Extra:

Continuous feedback from users and stakeholders is crucial throughout the development process. Feedback should be solicited each time a new feature is implemented. During alpha testing, it is important to assess user satisfaction. Psychologists or other experts may be consulted to design appropriate questionnaires for the testers involved. Alpha testing helps identify malfunctions before beta testing.

Beta testing, conducted in a production environment, allows real users to uncover any bugs or issues before the general release. Even after release, it is vital to collect logs from the application to assist developers in debugging and improving the system.

6 Effort Spent

Member of group	Effort spent	
Arianna Paone	Introduction	1*h*
	Architectural Design	8*h*
	Requirements Traceability	0*h*
	User Interface Design	0*h*
	Implementation, Integration, and Testing	0*h*
	Homework	3*h*
	Total	**3h**
Matteo Pasqual	Introduction	2*h*
	Architectural Design	8*h*
	Requirements Traceability	0*h*
	User Interface Design	0*h*
	Implementation, Integration, and Testing	3*h*
	Homework	3*h*
	Total	**5h**
Matilde Restelli	Introduction	1*h*
	Architectural Design	8*h*
	Requirements Traceability	0*h*
	User Interface Design	5*h*
	Implementation, Integration, and Testing	0*h*
	Homework	3*h*
	Total	**3h**

Table 6.1: Effort spent by each member of the group.