Software Design Description for Uni-Link: University Collaboration & Academic Platform

Youssef Mohamed, Mohamed Wael, Youssef Ahmed, Omar Ehab , Ali Mohamed
Supervised by: Dr. Mohamed El Hosseiny

December 23, 2025

Table 1: Document version history

Version	Date	Reason for Change
1.0	20-Dec-2025	SDD Final version.
1.1	22-Dec-2025	Final review and formatting updates.

GitHub: https://github.com/youssefmohussein/Uni-Link

Contents

Abstract

Uni-Link is a web-based university collaboration platform that connects students, professors, and administrators in a unified environment. The system supports academic content sharing, project submission and evaluation, real-time chat rooms, and professional profiles that aggregate CVs, skills, and projects. The backend adopts a modular MVC architecture reinforced by object-oriented design and several design patterns, including Strategy for role-based access control and post interactions, Mediator for coordinating posts and chat rooms, Command for project evaluation actions, Observer for notifications, Facade for profile aggregation, and Chain of Responsibility for chat message validation. These patterns promote separation of concerns, extensibility, and testability, enabling the system to scale as requirements evolve. This Software Design Description documents the architecture, key components, data model, and interactions that ensure Uni-Link satisfies the functional and non-functional requirements defined in the Software Requirements Specification.

1 Introduction

1.1 Purpose

This Software Design Description (SDD) specifies the architecture and detailed design of the Uni-Link platform, a university collaboration and academic management system. It serves as the primary reference for developers, testers, and reviewers to understand how the system is structured, how major components collaborate, and how quality attributes such as scalability, maintainability, and security are achieved. The document is aligned with the Software Requirements Specification (SRS) and will be used to verify that the implementation conforms to the approved design.

1.2 Scope

The Uni-Link platform enables students, professors, and administrators to collaborate around academic content, project submission and evaluation, and real-time communication. The scope of this SDD covers the backend design (controllers, services, mediators, strategies, commands, repositories, and data model) and its interaction with the frontend and external services. Operational details such as deployment, hardware sizing, and production monitoring are outside the scope of this document and may be addressed in separate documents.

1.3 Overview

The remainder of this document is organized as follows. Section 2 provides a system overview, including scope, high-level objectives, and project timeline. Section 3 presents the design viewpoints, covering the context diagram, use case model, logical structure, patterns usage, algorithms, interfaces, and main interactions. Section 4 describes the data design and database schema. Section 5 focuses on the human interface design and key user interface elements. Section 6 provides a requirements matrix that traces design elements to SRS requirements, and Section 7 includes appendices and supporting material.

1.4 Intended audience

This SDD is intended for software engineering instructors and teaching assistants who will evaluate the project, as well as the development team members responsible for implementing and maintaining the system. Security-sensitive implementation details (such as actual encryption keys or deployment credentials) are not included in this document. However, the design emphasizes secure authentication, role-based authorization, and proper handling of user data to protect student and staff privacy.

1.5 Reference Material

The primary reference for this SDD is the Uni-Link Software Requirements Specification (SRS). Additional references include the project testing plan, course guidelines for software design documents, and standard software engineering literature on MVC architecture, object-oriented design, and design patterns. The System Usability Scale (SUS) [1] is used as a reference for usability-related objectives.

1.6 Definitions and Acronyms

Term	Definition
Software Design Document (SDD)	Primary document for communicating software design information.
Design Entity	A design element that is structurally and functionally distinct from others.
Design Rationale	The reasoning, options, trade-offs, and decisions that led to the chosen design.
MVC	Model–View–Controller pattern separating presentation, business logic, and data access.
OOP	Object-Oriented Programming using classes, inheritance, and polymorphism.
CRUD	Create, Read, Update, Delete operations on persistent data.
JWT	JSON Web Token, a compact format for stateless authentication.
RBAC	Role-Based Access Control based on roles (Student, Professor, Admin).
UML	Unified Modeling Language for use cases, classes, sequences, and states.
SUS	System Usability Scale, a standard usability questionnaire.

2 System Overview

The Uni-Link system is a web-based university collaboration and academic management platform. It connects students, professors, and administrators through a unified portal that supports project

submission and evaluation, academic content sharing, real-time collaboration via chat rooms, and professional profiles that aggregate CVs, skills, and projects. The backend is implemented using an MVC-style architecture and object-oriented design with multiple design patterns to keep controllers thin, centralize business logic, and separate concerns between services, mediators, strategies, commands, repositories, and the database layer.

2.1 System Scope

The scope of Uni-Link includes the following major capabilities:

- User registration, login, and secure authentication using hashed passwords and token-based sessions.
- Role-based access control (RBAC) for Students, Professors, and Admins, controlling permissions on posts, projects, chat rooms, and administration features.
- Project management, including project submission by students, review and grading by professors, and status tracking (Pending, Approved, Rejected).
- Posts and comments with media attachments and interactions (like, save, share), filtered by faculty or interest.
- Real-time collaboration through password-protected chat rooms with role-specific permissions, message validation, and user mentions.
- Notification delivery for key events such as post interactions, project reviews, chat mentions, and profile-related updates.
- Profile management that aggregates user information, CVs, skills, and projects into a unified academic profile.

The system assumes integration with existing university infrastructure for identity verification and official grading export, which are outside the current scope.

2.2 System objectives

The main objectives of Uni-Link are:

- To provide a centralized collaboration platform that supports at least the three core roles (Student, Professor, Admin) with clearly separated permissions and workflows.
- To enable students to submit and track projects, and professors to review and grade them, with all project status changes recorded and auditable.
- To support reliable real-time communication such that messages are processed through validation, permission, mention detection, and persistence steps with a failure rate below a predefined threshold during testing.

- To maintain professional user profiles by allowing each student to upload a CV and at least one project and skill set, and making these profiles consistently retrievable through a unified facade.
- To implement a modular backend using at least five distinct design patterns (Strategy, Mediator, Command, Observer, Facade, Factory, Repository, Chain of Responsibility) to improve maintainability and unit-test coverage of critical services.
- To achieve a usability level corresponding to a System Usability Scale (SUS) score of at least 68 [1] during user evaluation sessions.

2.3 System Timeline

The project timeline between the SRS and the Technical Phase is divided into the following high-level stages:

- **Detailed design and modeling:** preparation of UML diagrams, database schema, and this Software Design Description.
- **Backend implementation:** development of core domains (users, posts, projects, chat, notifications), services, design pattern components, and repositories.
- **Frontend integration:** connecting the React-based user interface with backend APIs and implementing key user flows.
- **Testing and validation:** unit testing of services, integration testing of main use cases, and usability evaluation.
- **Deployment preparation and demo:** preparing deployment scripts or configuration and delivering the final demonstration.

A detailed Gantt chart showing the planned start and end dates, dependencies, and milestones is provided in the project plan section of this document.

3 Design viewpoints

3.1 Context viewpoint

The Uni-Link platform operates as a central web system that connects three primary human actors—Students, Professors, and Admins—with the backend services and persistent data store. Students use the system to submit projects, publish posts, comment and interact with content, join chat rooms, and manage their academic profiles. Professors review and grade projects, moderate academic posts, participate in discussions, and provide feedback through comments and chat. Admins manage users, faculties, and majors, monitor platform activity, and enforce global policies such as account status and content visibility.

From an environmental perspective, Uni-Link exposes a web interface (frontend) that communicates with a backend API and a database. The backend may also integrate with external

university services (e.g., identity providers or e-mail/notification gateways) to support authentication and outbound notifications. Figure 1 shows the high-level context of Uni-Link, its main actors, and surrounding systems.

Design concerns: System services and users.

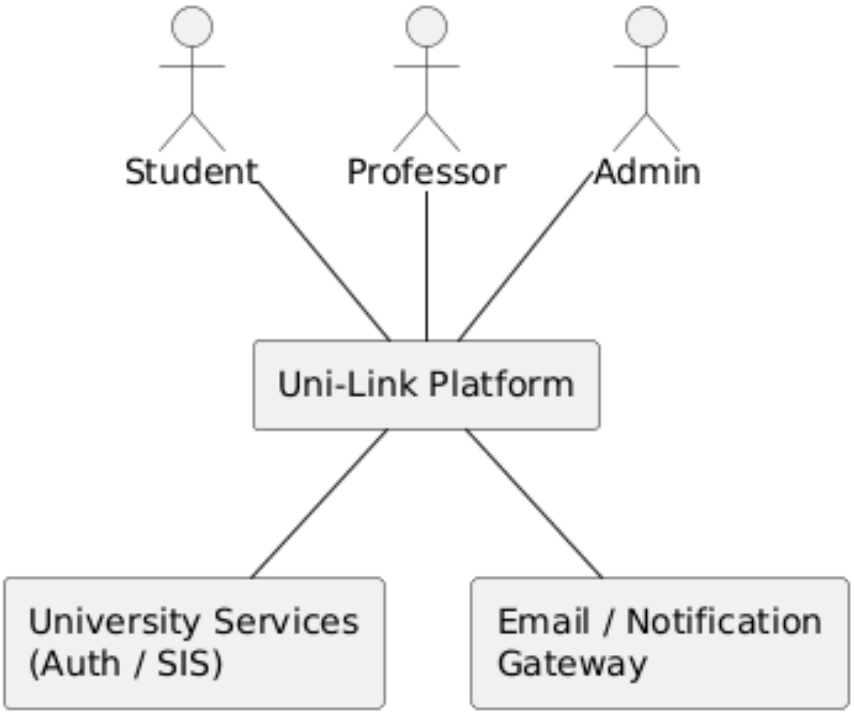

Figure 1: Context Diagram for the Uni-Link Platform

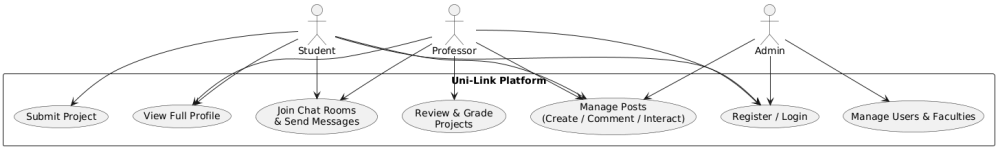

Figure 2: Main use case diagram for the Uni-Link platform

3.2 Composition viewpoint

The Uni-Link backend is composed of several cooperating layers and components that follow an MVC-style architecture. At a high level, HTTP requests are handled by controllers, which delegate business logic to services. Services coordinate domain objects and pattern-based components (mediators, strategies, commands, observers, and chains of responsibility) and rely on repositories to access the database. A facade aggregates profile-related data, and specialized mediators coordinate complex interactions such as posts and chat messages.

The main compositional elements are:

- **Controllers:** Expose RESTful endpoints for authentication, posts, projects, chat rooms, notifications, and admin operations.
- **Services:** Implement business rules (e.g., project submission, grading, post interactions, chat message processing) while keeping controllers thin.
- **Pattern Components:** Mediators, strategies, commands, observers, facades, and handlers encapsulate cross-cutting or variable behavior.
- **Repositories:** Provide an abstraction over the database for users, posts, projects, and chat messages.
- **Database:** Stores persistent entities such as users, posts, comments, projects, skills, chat rooms, and notifications.

3.2.1 Design Rationale

This composition was selected to achieve high cohesion and low coupling. Controllers are kept thin by delegating all non-trivial logic to services, which simplifies testing and allows the same services to be reused by different entry points. Pattern-based components encapsulate variable behavior (e.g., role-based access strategies, project commands, notification observers, and chat message handlers), which supports extension without modifying existing code. Repositories isolate data access concerns and make it possible to change the database technology with minimal impact on the rest of the system. Overall, this structure supports the project goals of maintainability, scalability, and clean alignment with the MVC architecture required by the course.

3.3 Logical viewpoint

From a logical perspective, Uni-Link is organized around a set of core domain classes that represent users, posts, projects, chat rooms, notifications, and supporting value types such as skills and CVs. The `User` class is an abstract base type with three concrete subclasses: `Student`, `Professor`, and `Admin`. Posts and comments form the basis of academic discussions, while projects and reviews represent formal project submissions and evaluations. Chat rooms and chat messages support real-time collaboration with mentions and role-specific permissions.

Figure 3 shows a simplified UML class diagram for the main domain entities and their relationships.

Tables 2 and 3 illustrate the detailed description format used for the main classes. The same structure can be replicated for other entities such as `Post`, `Comment`, `ChatRoom`, and `ChatMessage`.

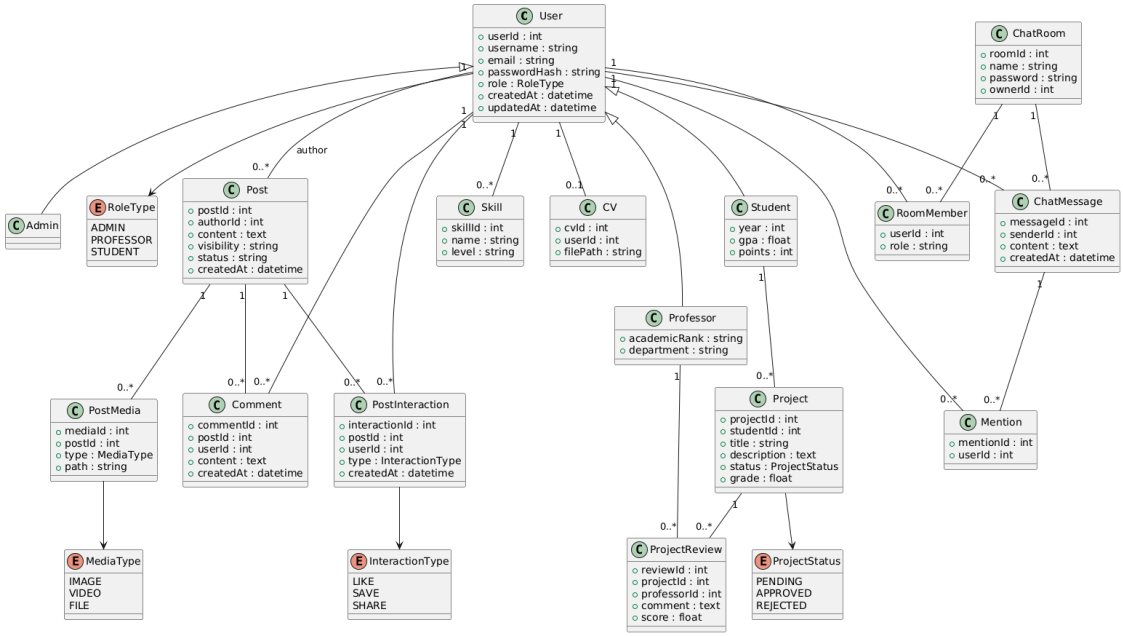

Figure 3: Simplified class diagram for Uni-Link core domain

Table 2: `User` class description

Abstract or Concrete:	Abstract
Superclasses	None
Subclasses	`Student`, `Professor`, `Admin`
Purpose	Represents a generic authenticated user of the system and encapsulates common identity, authentication, and profile data shared by all roles.
Collaborations	Collaborates with `UserRepository` for persistence, `ProfileFacade` for profile aggregation, and access-control components for role-based authorization.
Attributes	`userId:int`, `username:string`, `email:string`, `passwordHash:string`, `role:RoleType`, `createdAt:datetime`, `updatedAt:datetime`.
Operations	`login(credentials)`, `logout()`, `updateProfile(data)`, `changePassword(oldPwd, newPwd)`.

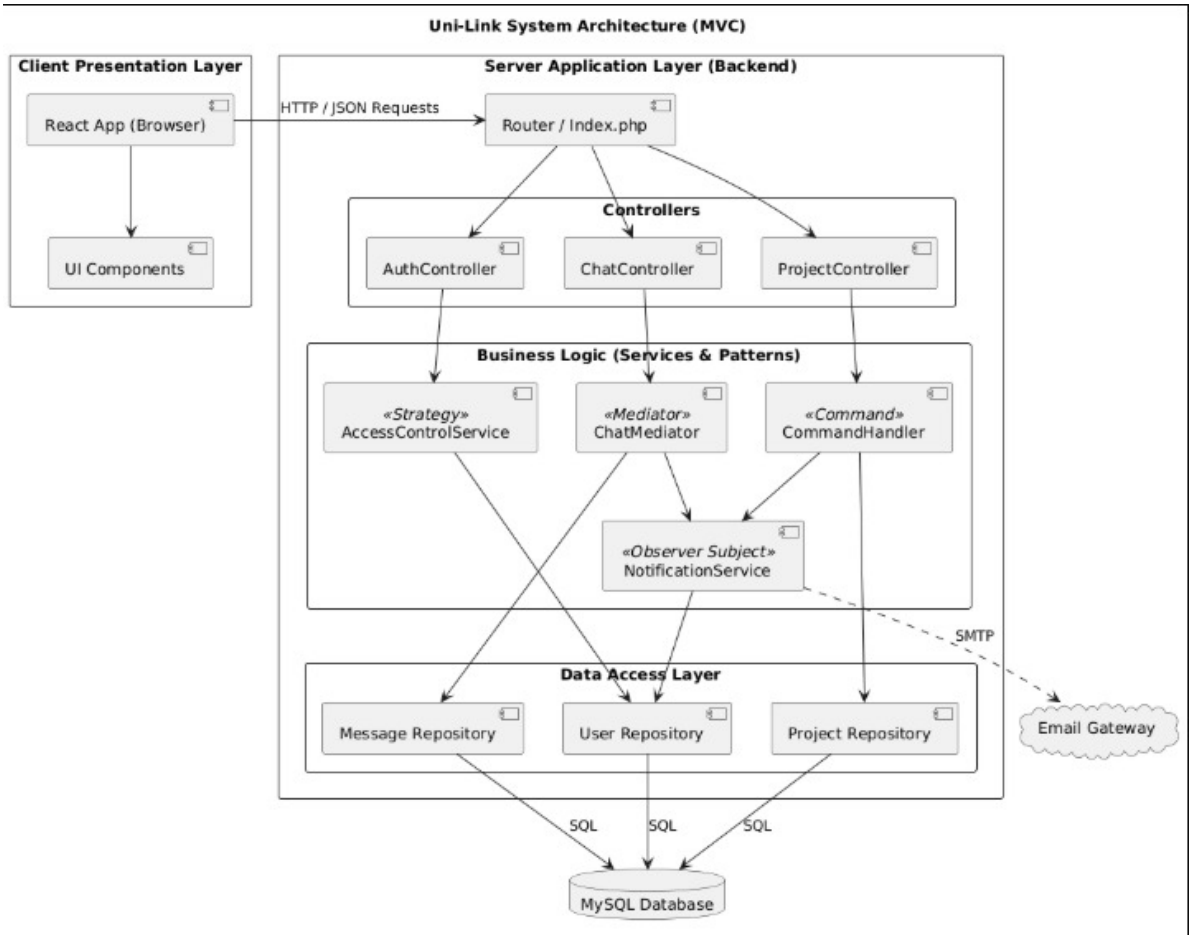

Figure 4: High-level architecture diagram for Uni-Link

Table 3: `Project` class description

Abstract or Concrete:	Concrete
Superclasses	None
Subclasses	None
Purpose	Represents a student project submitted for academic evaluation, including metadata, status, and grading information.
Collaborations	Collaborates with `Student` (owner), `Professor` (reviewer), `ProjectReview`, `ProjectRepository`, and project status/command strategies.
Attributes	`projectId:int`, `studentId:int`, `title:string`, `description:text`, `status:ProjectStatus`, `grade:float`.
Operations	`submit()`, `update()`, and operations invoked by commands such as approve, reject, and grade.

3.4 Patterns use viewpoint

The Uni-Link backend relies on several object-oriented design patterns to keep the architecture flexible, testable, and easy to extend. Each major concern is encapsulated in a dedicated pattern so that new behavior can be added with minimal changes to existing code.

Key patterns and their roles:

- **Strategy:** Used for role-based access control and post interactions. `AccessStrategy` has concrete strategies `AdminAccessStrategy`, `ProfessorAccessStrategy`, and `StudentAccessStrategy`, created by `AccessStrategyFactory` and used by `AccessControlService`. `InteractionStrategy` has `LikeInteraction`, `SaveInteraction`, and `ShareInteraction`, selected by `InteractionContext`. `ProjectStatusStrategy` encapsulates project status transitions.
- **Mediator:** `PostMediator` and `ChatMediator` coordinate interactions between posts, notifications, and chat components so that controllers and services do not depend on many collaborators directly.
- **Command:** `ProjectCommand` defines professor actions on projects. Concrete commands are `ApproveProjectCommand`, `RejectProjectCommand`, and `GradeProjectCommand`. This allows new actions without changing core project logic.
- **Observer:** `NotificationService` maintains a list of `NotificationObserver` objects such as `PostNotification`, `ProjectNotification`, and `ChatNotification`, and notifies them when events occur.
- **Facade:** `ProfileFacade` aggregates data from `User`, `Skill`, and `CV` to provide a single `getFullProfile(userId)` operation for building profiles.
- **Chain of Responsibility:** `MessageHandler` defines the chain used by `ValidationHandler`, `PermissionHandler`, `MentionHandler`, and `PersistenceHandler` when processing chat messages.
- **Repository:** `UserRepository`, `PostRepository`, `ProjectRepository`, and `ChatRepository` provide a clean abstraction over the database.

Figure 5 illustrates the main pattern families used in Uni-Link.

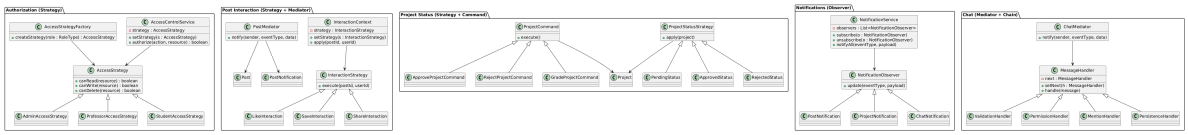

Figure 5: Key design patterns used in the Uni-Link backend

3.5 Algorithm viewpoint

The algorithm viewpoint focuses on how key use cases are realized by concrete processing steps. Uni-Link contains several important workflows, especially for chat message processing and role-based authorization.

Chat message processing (Chain of Responsibility):

- A user sends a message from the UI and the frontend calls the chat API.
- `ChatController` forwards the request to `ChatService`.
- `ChatService` sends the message to the head of the `MessageHandler` chain.
- **ValidationHandler** checks that the message is not empty and respects basic length and content rules. If invalid, the chain stops and an error is returned.
- **PermissionHandler** verifies that the sender belongs to the room and has permission to post.
- **MentionHandler** scans the text for `@username` patterns and creates the corresponding `Mention` records.
- **PersistenceHandler** stores the `ChatMessage` via `ChatRepository` and then triggers `ChatMediator` and `NotificationService` to distribute the message and notifications.

Role-based authorization (Strategy):

- A controller calls `AccessControlService.authorize(action, resource)` whenever an operation requires authorization.
- `AccessControlService` asks `AccessStrategyFactory` to create the proper `AccessStrategy` based on the current user role.
- The selected strategy checks whether the requested read, write, or delete operation is allowed on the given resource type such as a post, project, or chat room.
- The controller continues only if authorization succeeds; otherwise it returns an access denied response.

3.6 Interaction viewpoint

The interaction viewpoint describes dynamic behaviour and message flows between components during important use cases. Sequence diagrams show how controllers, services, pattern components, and repositories collaborate.

Student submits a project:

- The student fills the project form in the UI and submits it.
- The frontend sends a request to `ProjectController`.
- `ProjectController` calls `ProjectService.submitProject()` with the request data.

- `ProjectService` creates a `Project` entity, sets its status to `PENDING`, and saves it using `ProjectRepository`.
- `ProjectService` notifies `NotificationService`, which triggers `ProjectNotification` observers so professors can be informed.

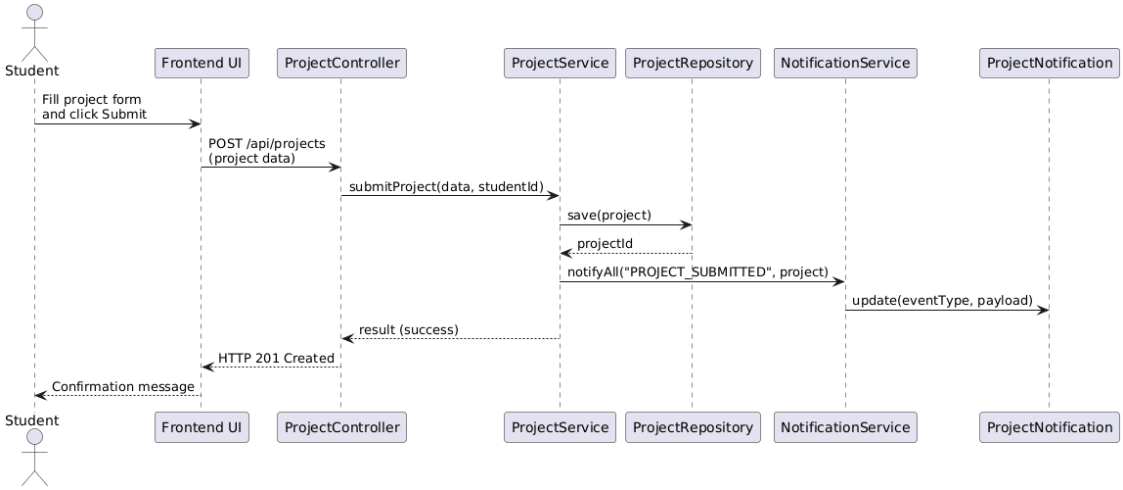

Figure 6: Sequence diagram for project submission

User sends a chat message:

- The user types a message in a chat room and presses send.
- The frontend calls `ChatController`, which delegates to `ChatService`.
- `ChatService` runs the message through the handler chain (validation, permission, mentions, persistence).
- After the message is stored, `ChatMediator` broadcasts it to connected clients and `NotificationService` creates `ChatNotification` entries for mentioned users.

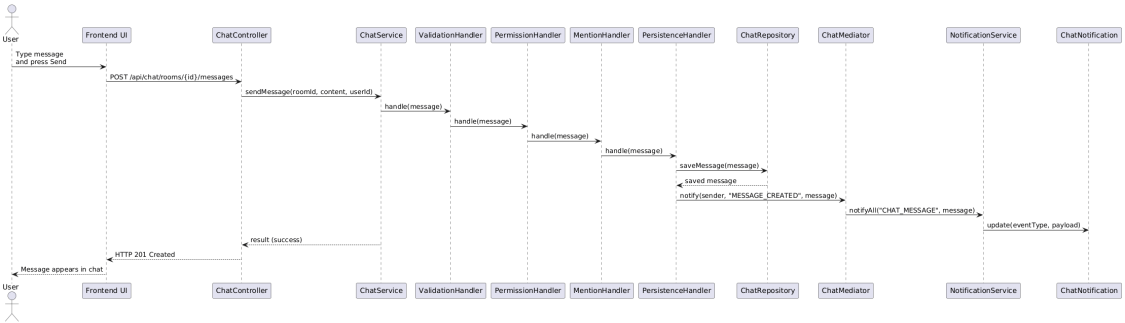

Figure 7: Sequence diagram for chat message processing

3.7 Interface viewpoint

The interface viewpoint describes the externally visible interfaces of Uni-Link, mainly the RESTful HTTP APIs consumed by the frontend and by any external systems.

The backend exposes JSON-based endpoints grouped by domain:

- **Authentication API:** registration, login, logout, and token refresh.
- **User and profile API:** retrieve and update user information, upload CVs, and manage skills using `ProfileFacade`.
- **Posts API:** create, edit, delete, and view posts; manage comments; and apply interactions (like, save, share).
- **Projects API:** allow students to submit or update projects and professors to approve, reject, or grade projects using command-based operations.
- **Chat API:** create chat rooms, join or leave rooms, and send messages that pass through the message handler pipeline.
- **Admin API:** manage users, faculties, majors, and access dashboard metrics.

Table 4 summarizes some representative endpoints.

Table 4: Representative Uni-Link REST API endpoints

Method	Path	Role(s)	Description
POST	`/api/auth/login`	All	Authenticate a user and return an access token.
POST	`/api/projects`	Student	Submit a new project for evaluation.
POST	`/api/projects/{id}/grade`	Professor	Grade an existing project using a project command.
POST	`/api/chat/rooms/{id}/messages`	Student, Professor, Admin	Send a chat message to a room; processed by the handler chain.
POST	`/api/posts/{id}/interactions`	Student, Professor	Apply an interaction (like, save, share) to a post using a strategy.
GET	`/api/profile/{id}`	All	Retrieve the full aggregated profile for a user via the profile facade.

4 Data Design

4.1 Data Description

The Uni-Link platform stores all persistent information in a relational database. The schema is organized around a small set of core entities: users and their profiles, posts and comments, projects and reviews, chat rooms and messages, and notifications. Each entity has a primary key (usually an integer identifier) and foreign keys that capture relationships between entities.

At a high level:

- **Users and roles:** The `Users` table stores common attributes for all users (students, professors, admins), including login credentials and role. Additional academic attributes are stored in role-specific tables such as `Students` and `Professors`.
- **Profiles:** The `CVs` and `Skills` tables extend the user profile. A user can have one CV file and multiple skills.
- **Posts and interactions:** `Posts` represent shared content; `Comments`, `PostMedia`, and `PostInteractions` (like, save, share) reference posts and users.
- **Projects and reviews:** `Projects` capture student submissions with a status and grade. `ProjectReviews` record professor feedback and scores.
- **Chat:** `ChatRooms` group users into conversations. `RoomMembers` links users to rooms. `ChatMessages` and `Mentions` store the messages and referenced users.
- **Notifications:** `Notifications` stores notifications generated by the observer pattern for posts, projects, and chat events.

Figure 8 shows the main database entities and their relationships.

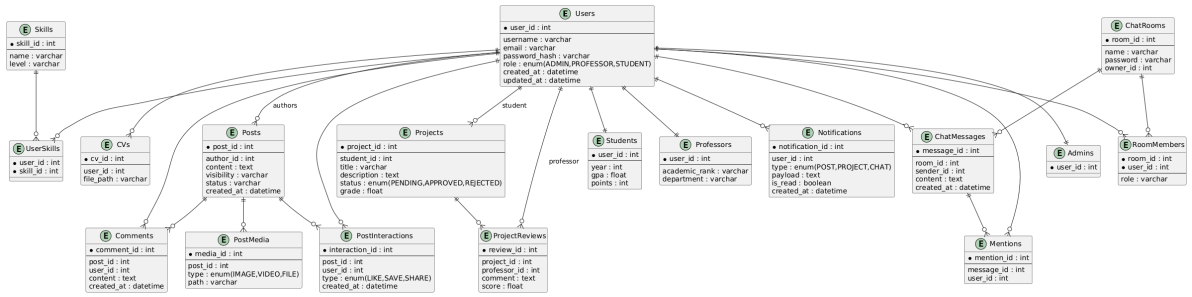

Figure 8: ER schema diagram for the Uni-Link database

4.2 Dataset Description

The current version of Uni-Link does not rely on an external dataset such as open data or machine learning corpora. All stored data is generated by users of the system during normal operation (registration, posting, project submission, chatting, and profile management).

For testing and demonstration purposes, synthetic seed data can be inserted into the database, including example users for each role, sample posts and comments, demo projects with reviews, and a few chat rooms with messages. This seed data is used only for local development and does not affect the logical data design described in this section.

4.3 Database design description

The Uni-Link backend uses a relational database to persist all application data. The schema is normalized around core entities such as users, posts, projects, chat rooms, and notifications, with foreign keys enforcing the relationships shown in Figure 8. Each table uses an integer primary key (or composite key for junction tables) and stores timestamps for auditing where appropriate.

Main design decisions:

- A single `Users` table holds common attributes for all roles. Role-specific academic attributes are stored in `Students`, `Professors`, and `Admins`, linked by a one-to-one relationship on `user_id`.
- Many-to-many relationships, such as users and skills, are modeled with junction tables (e.g., `UserSkills`) to keep the design flexible and extensible.
- Content-related entities such as `Posts`, `Comments`, `PostMedia`, and `PostInteractions` capture user-generated content, media files, and interactions while preserving authorship information through foreign keys to `Users`.
- `Projects` and `ProjectReviews` record project submissions and professor feedback, including status and grading information aligned with the project status strategy in the design.
- Real-time communication is stored using `ChatRooms`, `RoomMembers`, `ChatMessages`, and `Mentions`, enabling reconstruction of conversations and mentioned users.
- `Notifications` stores user-targeted notifications produced by the observer pattern, with a flag indicating whether each notification has been read.

The schema is designed to support typical CRUD workloads and frequent read operations by adding indexes on foreign keys such as `author_id`, `student_id`, `room_id`, and `user_id`, which improves query performance for common use cases like loading feeds, project lists, chat histories, and notifications.

5 Human Interface Design

5.1 User Interface

From the user's perspective, Uni-Link is a web portal that can be accessed through a standard browser. After authentication, the interface adapts to the user role (Student, Professor, or Admin) while keeping a consistent layout: a top navigation bar, a side menu for main modules, and a central content area for pages and forms.

Common functionality for all roles:

- Login and logout using email/username and password.
- View and edit personal profile, including basic information, CV upload, and skills.
- Access a home dashboard summarizing recent posts, notifications, and active projects or rooms.
- Open the notifications panel to see new post interactions, project updates, and chat mentions.

Student interface:

- Browse and filter posts by faculty or topic, create new posts with text and media, comment, and apply interactions (like, save, share).
- Submit new projects through a form that includes title, description, and attached files, and track the status and grades of previous submissions.
- Join available chat rooms, see the list of members, and participate in real-time conversations with message history and mentions.

Professor interface:

- View project submissions assigned to them, open project details, and perform actions such as approve, reject, or grade with feedback comments.
- Moderate academic posts by editing or removing inappropriate content and replying with academic comments.
- Participate in chat rooms with students, answer questions, and receive notifications about project-related events.

Admin interface:

- Manage users (create, enable, disable accounts) and assign or update roles.
- Manage faculties and majors, which are used to categorize posts and projects.
- Access an administrative dashboard that shows aggregated statistics about users, posts, projects, and system activity.

The UI design focuses on clarity and consistency so that users can quickly locate modules, understand available actions, and complete their tasks with minimal navigation steps.

5.2 Screen Images

This subsection presents representative screenshots of the Uni-Link user interface to illustrate the main workflows for each role. All screens follow a consistent layout with a top navigation bar, side menu, and central content area.


Uni-Link System Site Map
Uni-Link Platform (Root)
Public Area
Student Dashboard (Home)
Landing Page
Login Page
Registration Page
Forgot Password
News Feed / Posts
Create New Post
Post Comments View
Project Management
All Projects List
Create Project Wizard
Single Project View
Project Details & Description
Task Board / Timeline
Member List
Project Settings (Admin only)
Communication (Chat)
Chat List (Recent)
Chat Room
Message History
Active Participants
User Profile
View Profile
Edit Settings
Notifications
Activity Log

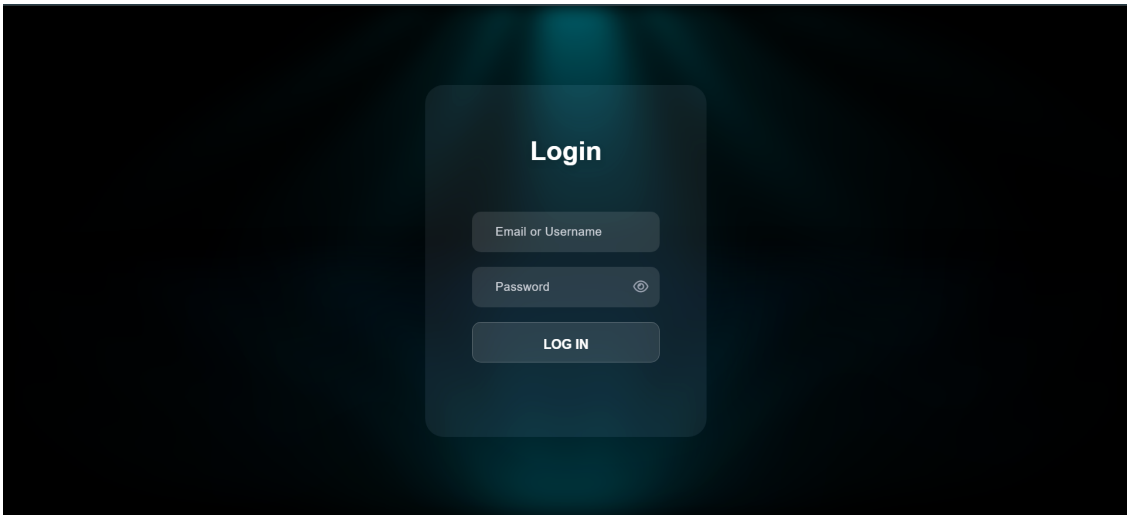

Figure 10: Login screen for all Uni-Link users

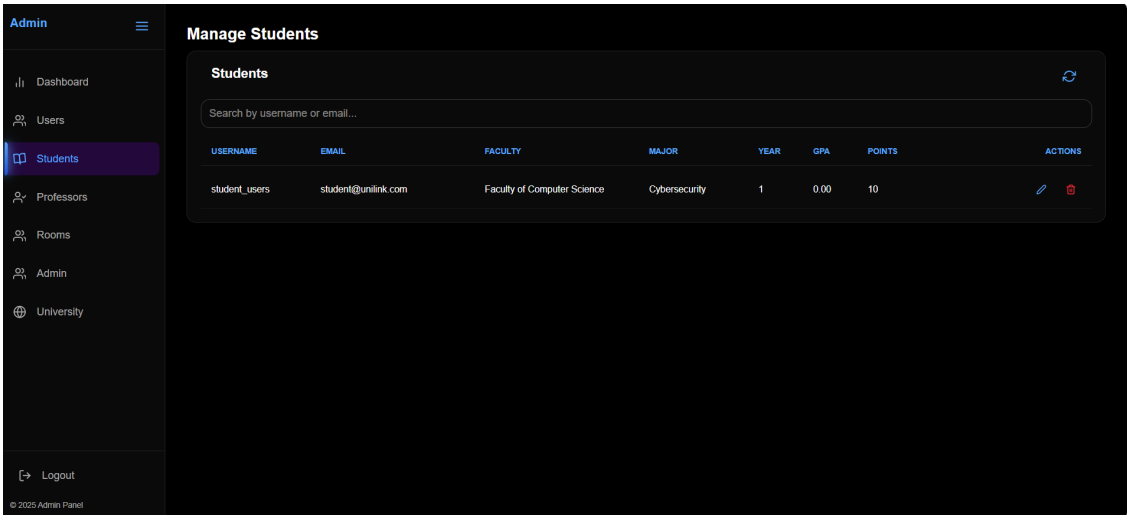

Figure 11: Student dashboard showing recent posts, projects, and notifications

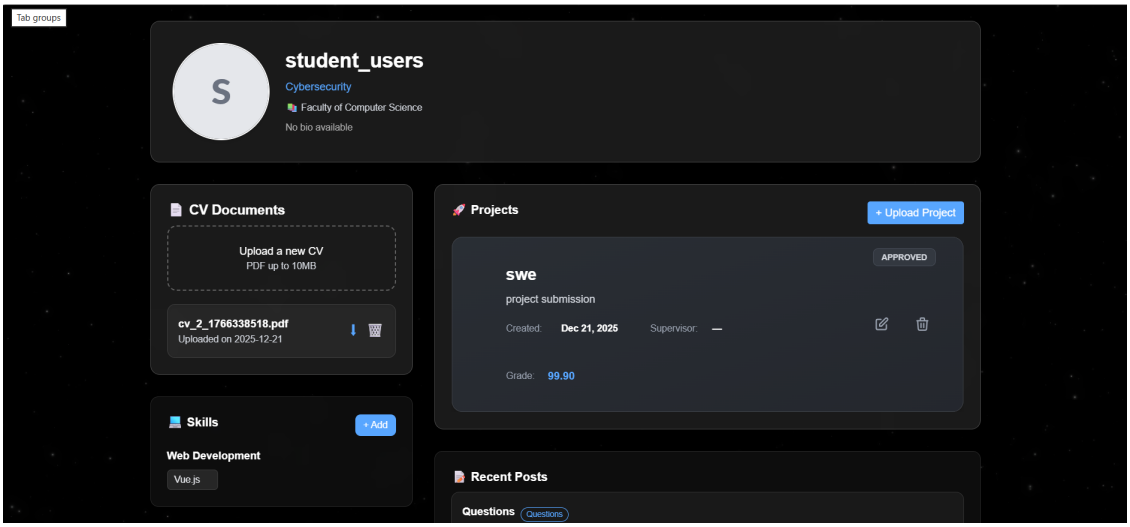

Figure 12: Project submission form used by students

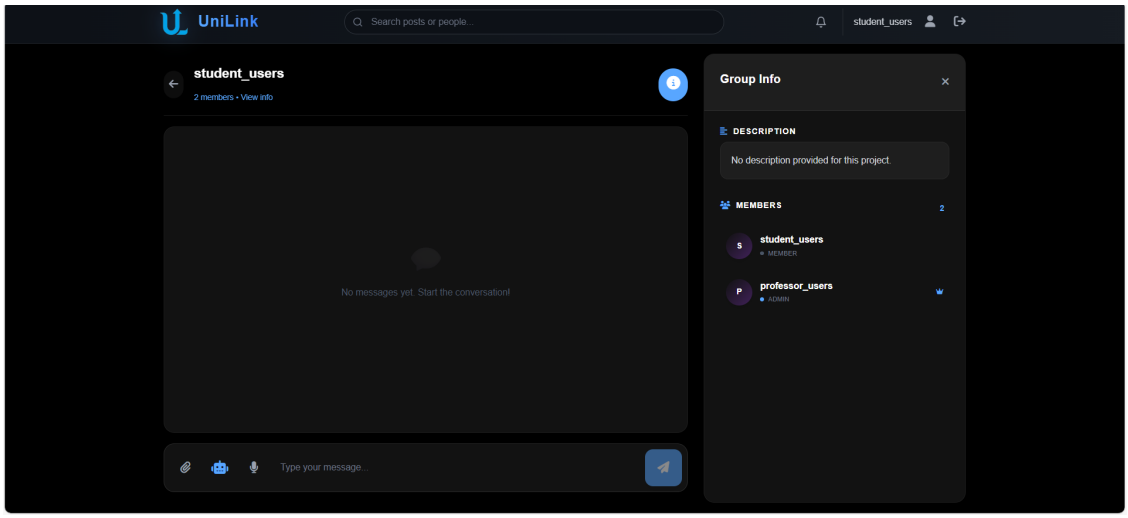

Figure 13: Chat room interface with message history and mentions

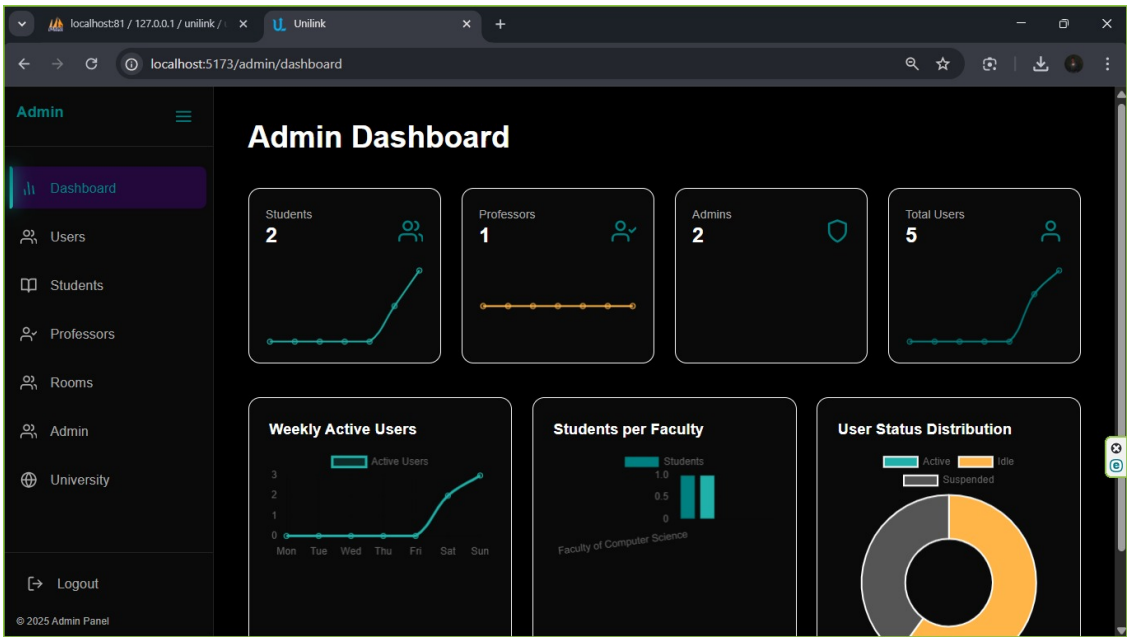

Figure 14: Admin dashboard for managing users, faculties, and system statistics

5.3 Screen Objects and Actions

This subsection summarizes the main screen objects in Uni-Link and the actions associated with them from the user's perspective.

Common objects

- **Navigation bar:** links to Home, Posts, Projects, Chat, Profile, and Logout; allows switching between main modules.
- **Side menu:** lists role-specific shortcuts (e.g., "My Projects", "Pending Reviews", "Admin Dashboard").
- **Notifications icon:** opens a panel showing recent notifications; actions: mark as read, open related item.
- **Profile avatar:** opens profile details; actions: view profile, edit profile, upload CV, manage skills.

Login screen

- **Email/username field:** enter login identifier.
- **Password field:** enter password (masked).
- **Login button:** submit credentials and authenticate the user.

Posts and feed screens

- **Post card:** shows author, content, media, and timestamp; actions: open details, like, save, share, comment.

- **New post form:** text area and media upload; actions: type content, attach images/files, publish post.
- **Comment box:** input under each post; actions: write comment, submit, delete own comment.
- **Filter controls:** dropdowns for faculty or category; actions: change filters to update visible posts.

Project screens

- **Project list table:** rows for each project; actions: open details, view status and grade.
- **Project submission form:** fields for title, description, attachments; actions: create or update a project.
- **Review panel (professor):** fields for comment and score; actions: approve, reject, or grade project.

Chat screens

- **Room list:** shows available chat rooms; actions: create room, join room, leave room.
- **Message area:** list of chat messages; actions: scroll history, see mentions and timestamps.
- **Message input box:** text field with send button; actions: type message, use `@username` mentions, send message.

Admin screens

- **User management table:** shows users with role and status; actions: enable/disable account, change role, search users.
- **Faculty/major management forms:** actions: create, edit, or delete faculties and majors.
- **Dashboard widgets:** cards and charts with aggregated statistics; actions: filter by date or faculty.

6 Requirements Matrix

7 APPENDICES

7.1 Github

This appendix presents screenshots from the GitHub repository to show the actual project structure and evidence of version control.

- Overview of the main repository page (name, description, latest commits).
- Folder structure highlighting the `backend` and `unilink` (frontend) directories.

Table 5: Requirements Matrix

Req. ID	Req. Description	Main Classes / Components	Test ID	Status
FR01	User shall be able to log in and log out securely.	User, AuthController, AuthService	TC01	Implemented
FR02	System shall enforce role-based access control (Student, Professor, Admin).	AccessControlService, AccessStrategy, AccessStrategyFactory	TC02	Implemented
FR03	Users shall create posts, comment, and interact (like, save, share).	Post, Comment, PostInteraction, PostController	TC03	Implemented
FR04	Students shall submit projects; professors shall review and grade them.	Project, ProjectReview, ProjectService, ProjectCommand	TC04	Implemented
FR05	Users shall send messages in chat rooms.	ChatRoom, ChatMessage, ChatService, MessageHandler	TC05	Implemented
FR06	System shall send notifications for key events (posts, projects, chat).	NotificationService, PostNotification, ProjectNotification, ChatNotification	TC06	Implemented
FR07	System shall display an aggregated user profile.	ProfileFacade, User, Skill, CV	TC07	Implemented
FR08	Admin shall manage users and faculties.	Admin, AdminController	TC08	Implemented

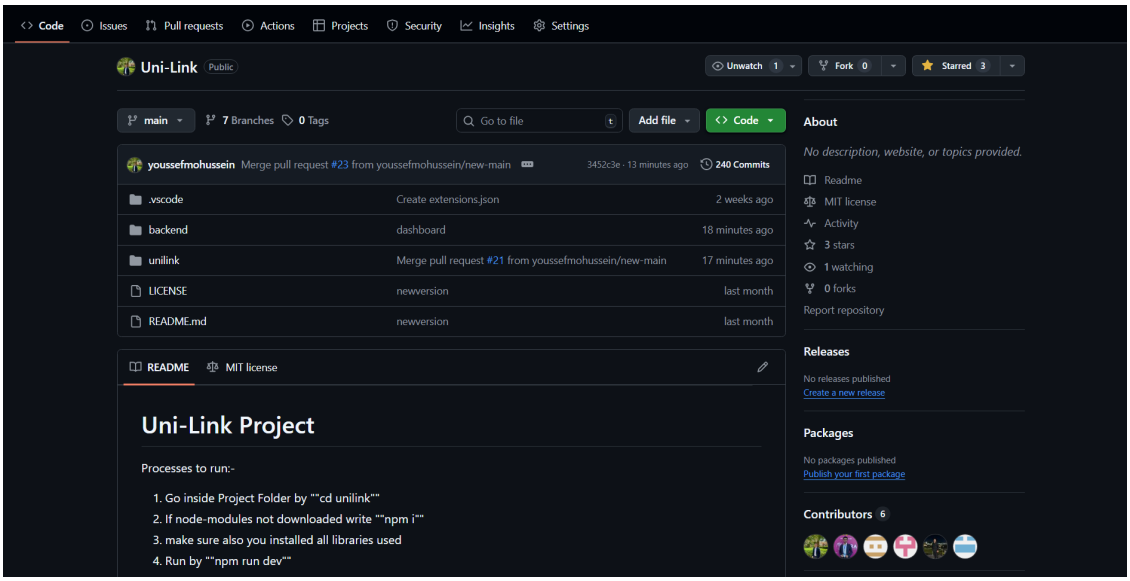

Figure 15: GitHub repository overview for the Uni-Link project

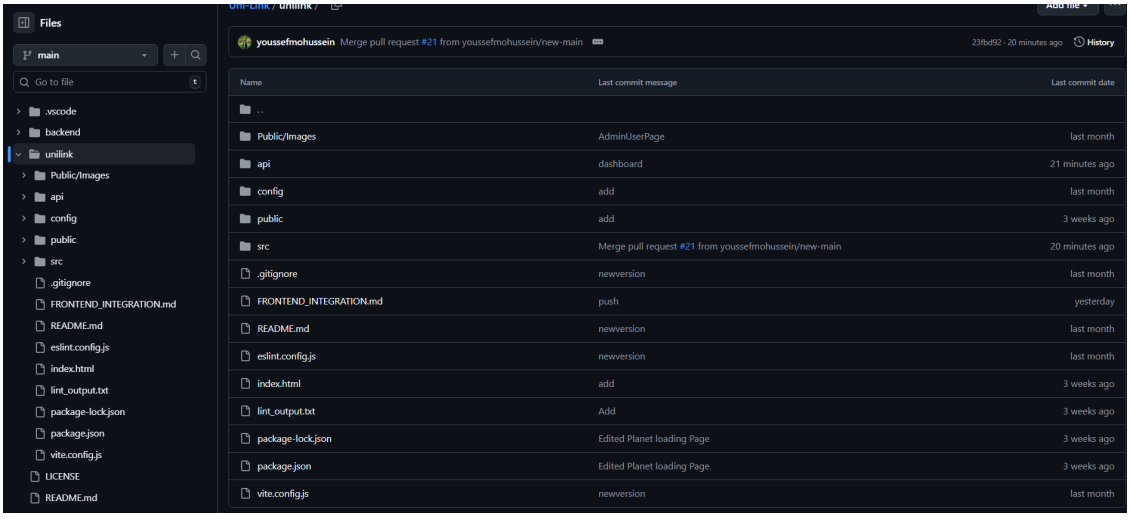

Figure 16: Repository structure showing backend and frontend folders

8 Project Plan

This section summarizes the project timeline, main phases, and milestones for the Uni-Link system. The project runs from 1 October to 20 December and is divided into several iterative phases that align with course deliverables.

8.1 Timeline and phases

Table 6: Project phases and schedule

Phase	Main Activities	Planned Dates
Requirements and analysis	Finalize problem statement, identify actors and main use cases, write SRS and initial high-level diagrams.	1 Oct – 15 Oct
Detailed design (SDD)	Design architecture, UML diagrams (class, sequence, state, context, use case), database schema, and design pattern usage.	16 Oct – 31 Oct
Implementation (backend and frontend)	Implement core domains (users, posts, projects, chat, notifications), APIs, and React screens; integrate design patterns.	1 Nov – 30 Nov
Testing and refinement	Unit and integration tests, bug fixing, UI polishing, performance tuning, and SUS-based usability evaluation.	1 Dec – 10 Dec
Deployment and final presentation	Prepare deployment environment or demo setup, finalize documentation (SDD), and conduct final presentation.	11 Dec – 20 Dec

8.2 Milestones

Key milestones within this timeline are:

- **M1 – SRS completed:** Requirements and initial use cases approved (around 15 Oct).
- **M2 – SDD completed:** Architecture, UML models, and database design finalized (around 31 Oct).
- **M3 – First working prototype:** Core features (authentication, posts, basic projects, basic chat) implemented (around 30 Nov).
- **M4 – Testing completed:** Main test cases executed and critical bugs fixed (around 10 Dec).

- **M5 – Final demo and submission:** System deployed or prepared for demo, all documents submitted (by 20 Dec).

Figure 17 shows the project timeline as a Gantt chart.

Figure 17: Gantt chart for Uni-Link project phases

array

9 Task Distribution

This section shows the responsibilities assigned to each team member. The system design follows the IEEE 1016 standard [2], and usability was measured using the SUS scale [1]. The team utilized tools like Figma [3] for prototyping and GitHub [4] for version control.

References

[1] IEEE Computer Society. (2009). *IEEE Standard for Information Technology—Systems Design—Software Design Descriptions (SDD)*. IEEE Std 1016-2009.

[2] John Brooke. (1996). "SUS: A Quick and Dirty Usability Scale." *Usability Evaluation in Industry*.

[3] Miro. *Online Collaborative Whiteboard Platform for System Design and Task Planning*. https://miro.com/

[4] Figma. *Interface Design and Prototyping Tool*. https://www.figma.com/

[5] Meta Platforms, Inc. *React Official Documentation*. https://react.dev/

[6] React Bits. *Reusable React Components and UI Patterns*. https://www.reactbits.dev/

[7] Groq Inc. *Groq API Documentation: High-Performance LPU-Based AI Inference*. https://console.groq.com/docs

[8] GitHub Inc. *GitHub Documentation: Version Control and Collaboration Workflows*. https://docs.github.com/

Table 7: Team members and main responsibilities

Member	**Role**	**Main Tasks**
Youssef Mohamed	Backend, Database, Frontend	• Full backend development, including implementation of design patterns, middleware, services, and system configuration • Database design, including UML diagrams and database schema definition
Youssef Ahmed	Backend, Database, Frontend	• Database maintenance, including adding and modifying tables to support new system features • Development of the backend notification system
Mohamed Wael	Frontend	• Frontend development and implementation of user interface components
Omar Ehab	Frontend	• Frontend development and implementation of user interface components
Ali Mohamed	Frontend and documentation	• Frontend development • Preparation and maintenance of project documentation