Chapter 1 - Introduction

The final year project is the cornerstone of any undergraduate academic program, serving as the final evaluation for the award of a degree. Its importance is further highlighted by the fact that it represents the crystallization of knowledge of a graduate throughout their academic journey so far and hence serves as a tangible demonstration of their skills to industry recruiters or for the pursuit of higher graduate studies.

At the University of Mauritius (UoM), final year projects are traditionally proposed either by the faculty staff or by the students themselves, and are then allocated accordingly, to a supervisor-student pair. This supervision process is structured as a feedback loop whereby students are expected to iterate over their ‘Dissertation’ - a comprehensive document describing the project’s aims, literature review, methodology and its results – based on their supervisor’s feedback.

Therefore, it is imperative that every aspect of the supervision process, from the meetings conducted for communication between the supervisor and students to the feedback given remain as high as possible, for they shape not only the student’s academic success but also the metrics that determine the credibility and reputation of an institution.

1.1. Problem Statement

According to a recent paper on project supervision (Stynes, P. and Pathak, P., 2022), the traditional supervisory process for final year projects is not scalable. While the research focuses mainly on the postgraduate/master’s level, its underlying principles – the supervisor’s finite time and attention, and one-on-one mentorship – are directly applicable for undergraduate programs as well. This lack of scalability leads to significant administrative burdens for academics and degrades the quality of the supervisory feedback, particularly within institutions with high enrollment numbers. As a result of these operational inefficiencies, the quality of final year projects may suffer, affecting the institution’s overall academic output

and its credibility. This can be an emerging problem within the University of Mauritius' (UOM) Faculty of Information and Communications Technology (FOICDT), whereby the total enrollment has increased from 916 in 2020 to 1339 in 2022 (University of Mauritius, 2025), representing an increase in 46.2%. This rapid growth places a significant strain on the academic staff, who must adhere to manual and inefficient processes.

58	**E700**	Post-Doctoral Fellowship	PT	2		
59	**FOICDT**					
60	**E311**	BSc (Hons) Information Systems	FT	4		961
61	**E311M**	BSc (Hons) Information Systems	FT	1		
62	**IC311**	BSc (Hons) Information Systems	FT	142		
63	**IC318**	BSc (Hons) Computer Science	FT	219		
64	**E318**	BSc (Hons) Computer Science	FT	3		
65	**E318M**	BSc (Hons) Computer Science	FT	9		
66	**IC318M**	BSc (Hons) Computer Science	FT	26		
67	**IC319**	BSc (Hons) Applied Computing	FT	136		
68	**E319**	BSc (Hons) Applied Computing	FT	10		
	E319M	BSc (Hons) Applied Computing	FT	8		

Figure 1: Sum of Total Enrollment for UoM's FOICDT 2019-2020

2	FACULTY	TOTAL		
3		T	M	F
4	Faculty of Engineering	996	635	361
5	Centre for Innovative and LifeLong Learning	954	669	285
6	Faculty of Law and Management	2833	891	1942
7	Faculty of Information Communication and Digital Technologies	1339	836	503
8	Mahatma Gandhi Institute	477	125	352
9	Faculty of Social Sciences and Humanities	1351	311	1040
10	Mauritius Institute of Education	13	11	2
11	Faculty of Agriculture	359	107	252

Figure 2: Sum of Total Enrollment for UoM's FOICDT 2021-2022

Supervisors who are already encumbered from managing multiple students, have seen their workload increase further. Much of it comprises of menial tasks such as managing communications via email, progress tracking, rescheduling meetings due to conflicting schedules and even progress log report generation. Most notably, however, the process of manually verifying a re-submitted student's work against prior feedback remains a time-consuming and arduous task. This difficulty scales with the number of projects that the supervisor needs to oversee, primarily because they need to remember or cross-reference previous material in order to perform this task correctly.

Thus, a greater portion of the workload is dedicated to handling administrative overhead as opposed to their core responsibilities such as teaching, supervising and research, impacting the quantity and quality of feedback provided. Evidently, these affect the students as well since providing insufficient or vague feedback makes it more difficult to identify and resolve underlying issues, triggering a loss in motivation for both parties. Consequently, the probability that the project meets academic standards reduces significantly.

1.2. Proposed Solution

To address these inefficiencies and subsequently the challenges of scalability on the institutional level, the project proposes a web-based platform, Project Management System (PMS), designed with the goal of automating unproductive and repetitive tasks and streamlining existing processes.

It will thus comprise of the following aims and objectives:

1.2.1. Project & Task Management

Consolidating projects and their corresponding tasks and deliverables on one platform, made accessible by an intuitive interface, replaces the need to piece together status updates from scattered emails and other communications. This ensures that both the supervisors and students are synchronized on the current state of the project.

1.2.2. Meeting Scheduling

Centralized meeting scheduling, made accessible by a calendar scheduler, aims to greatly reduce the cognitive effort required to schedule and manage meetings. For example, the visibility of project meetings from other students with a common supervisor greatly helps both the students and the supervisor in avoiding scheduling conflicts. On the same note, the meeting scheduler employs visual distinction to distinguish between pertinent meeting events for a student from unrelated events from other students. This categorization facilitates the users in processing events more efficiently.

1.2.3. AI Feedback Analysis

The inclusion of an AI Feedback Analysis component in the software can be justified due to the high cost-benefit ratio present in manual feedback analysis. While providing new feedback remains a core responsibility of a supervisor, verifying whether the feedback has been considered in a student's re-submitted work largely remains an arduous task due to reasons mentioned earlier. Therefore, by leveraging an AI model to perform the document comparisons to ascertain whether feedback has been applied, the process can be streamlined, freeing up time for more productive supervision tasks. To mitigate potential model prediction errors, the AI Feedback Analysis component will be designed as a support tool – visibility on its process will be provided by keeping track of changes for each feedback criteria and a fallback mechanism will be included well.

1.2.4. Notifications & Reminders

To ensure that the system works as a source of truth for project dissertations, relevant updates such as tasks and meeting events are tracked via a notification & reminder sub-system. Notifications and important reminders will be kept separate from one another but will retain a high visibility in the user interface to alert users and keep them updated.

1.2.5. Progress Log Report Generation

The Automatic Progress Log Report Generation feature eliminates the administrative burden of referencing meetings and task information in order to compile a progress log report.

Chapter 2 - Background Study

This chapter will consist of a literature review of the features and limitations of each solution. This will include a section on the evaluation of their features with respect to the proposed system features and an appraisal section. The background study will then end with a section that describes the proposed system and its constituent features in detail.

2.1. Literature review

The following solutions derived from both widely available commercial software and academic research from various universities, arranged in tabular form.

Existing Solutions	**Features**	**Limitations**
A Web Based Final Year Project Supervision System (Beekhy, 2013)	- Appointment scheduling system which uses the supervisor's calendar as a basis for scheduling meetings (Google Aps' API for Calendar display). - Document management system where supervisors can set tasks for deliverable submissions/re-submissions. - Progress tracker with Gantt chart visualization to provide a complete picture of project's progress.	- No visual distinctions or logical organization of different types of meeting appointments (accepted, rejected and other appointments) - The author conflates progress reports with task deliverable submissions *(Beekhy, 2013, p. 85)*. Therefore, the system lacks an automated progress log report generation functionality.
Creatrix Campus *(Creatrix Campus, 2025:-b, Creatrix Campus, 2025:-c)*	- Centralized document repository for change tracking and version control. - Customizable review and approval workflows to streamline dissertation	- No out-of-the-box solution for automated feedback checking; training is required for developers to implement and maintain custom

	management (Low-code configurations). - Reporting analytics for evaluating dissertation and thesis milestones for both students and supervisors. - Meeting management software includes meeting scheduling with conflict avoidance and attendance tracking.	review and approval workflows. - Enterprise-level pricing may present a significant financial barrier.
Moodle as a Final Year Project Management System *(Khamaruddin et al., 2018)*	- The system re-purposes Moodle's forum activity module to provide reminders with email notifications for important tasks and deadlines. - Assignment activity module in Moodle allows supervisors to set assignments where students can submit drafts and final documents. - Progress updates are tracked via the system's course administration module.	- Assignment activity module does not provide change tracking between submissions. - The system lacks a meeting planning and automatic feedback checking system.
Final Year Project Management System for Information Technology Programmes *(Leung et al.,*	- The system has a File Sharing and Repository module to centralize document and code submissions. - Project Management module provides students with a shared workspace and a scheduler for them to create tasks and set deadlines.	- The system lacks a meeting scheduling and automatic feedback checking system. - Group-based submission may obscure work accountability for individual students.

2018)	- Basic progress tracking feature for each student.	
Google Workspace for Education *(Google, 2025)*	- Google Classroom acts as the core of the system where supervisors can make classes for project students and create assignments with deadlines. - Google Classroom integrates seamlessly with Google Calendar, which can be used as a basis for meeting scheduling. - Google Docs allows keeping track of different document versions for version control and overall change tracking using “Suggest” mode. - Google Docs integrate with Google Assignment for feedback submission.	- Lack of customization flexibility to incorporate an AI solution for feedback compliance checking. - Inadequate progress tracking capabilities to provide an overview of project’s progress.
Web-Based Final Year Project Supervision Management System (FYPSMS) (Adeniyi et al., 2024)	- Student home page for uploading project work documents and viewing progress. - Supervisor dashboard to view progress for multiple students and project inspection page for viewing, downloading documents & providing feedback.	- The system lacks a dedicated meeting scheduling system and automatic feedback checking. - There is no feature to track changes between document submissions.

2.1.1. Comparison between Existing Solutions

Solutions	**Meeting Scheduling & Reminder Notification System**	**Deliverable Submission/ Re-Submission**	**Automatic Feedback Compliance Checking**	**Progress Tracking (progress log or visual)**	**Specific Features**
A Web Based FYP Supervision System	✓	✓	✗	✗	- Meeting scheduling based on the supervisor calendar. - Gantt Chart visualization for progress tracking
Creatrix Campus	✓	✓	?	✓	- Low code platform to create review and approval workflows for deliverables. - Meeting Attendance tracking
Moodle as a FYP Management System	✓	✗	✗	✓	- Re-purposing existing open-source Moodle modules to provide specific features
FYP Management System for IT Programmes	✗	✗	✗	✗	- Architecture for streamlining group-based project supervision
Google	✓	✓	✗	✗	- Document versioning for

Workspace for Education					visible change tracking - Prevent document editing/re-submission during feedback
Web-Based FYPSMS	✗	✗	✗	✓	- Dedicated dashboards for system administrators and external examiners

2.1.2. Appraisal

The ecosystem of digital solutions aimed at addressing the inefficiencies in academic management is already mature enough to justify their widespread adoption among universities around the globe. The extent of their integration with existing IT infrastructure varies significantly from one institution to another. For instance, some institutions use standard tools like Learning Management Systems for basic communication and deliverable handling. Others, however, adopt full-fledged centralized platforms to manage the entire supervision process from smart meeting scheduling to document management with progress tracking.

The University of Mauritius currently uses Google Workspace for Education as a general education platform for a variety of courses to provide learning material, centralize assignment submissions or simply to broadcast reminders to particular cohorts. However, as Beekhy (2013) pointed out, there is no unanimously agreed upon methodology, and subsequently no centralized system supervisors use for the supervision process.

An evaluation of the project management systems listed so far shows that they tend to address areas concerning project allocation, meeting scheduling and managing deliverable submission. Most of them are adapted for the one-on-one supervision process although Leung et al. (2018) described a system that provides features dedicated for group-based

supervision. Nonetheless, one functional gap that all of these systems have in common is that they lack some form of automated feedback verification for deliverable file re-submissions. In fact, most of them typically employ a rudimentary feedback mechanism whereby feedback is provided in the form of a comment rather than structured criteria that can have different states (met, unmet).

2.2. Proposed System Features

This section will present the proposed system features based on the solutions identified in the previous context and the application's intended purpose.

2.2.1. Role-Based Access Control

Firstly, the web platform will require a way to verify users of the system and determine which set of features of the system they will be allowed to access (scope of their actions). For this purpose, the system will have to provide authentication – verify users based on their credentials and categorize them as either one of the stakeholders below:

Then, RBAC + Ownership authorization *(see appendix)* will be employed to limit the scope of the authenticated user's actions.

2.2.2. Administrator Dashboard

Since the system requires a secure way of determining the roles of users, they will be managed by an administrator (academic staff) via a dedicated dashboard inaccessible to the other users. The administrator's role will encompass user creation and role assignment.

2.2.3. Project Management

Supervisors and students will primarily interact through the project management and meeting scheduling components. The system enforces a top-down hierarchy, with supervisors having the highest privileges.

These privileges include the ability to create, update and archive projects and assign specific students to projects. Students will only have view permissions on their assigned projects.

2.2.4. Progress Log Generation

To satisfy the administrative requirements for progress reports, the system will feature a progress log report generation component that automatically compiles a report by aggregating task and meeting data for a given project.

Progress log report generation will be made available for both student and supervisor roles.

2.2.5. Deliverable Task Management

The core supervision process follows an iterative life-cycle – the supervisor sets a task that expects a deliverable, the student submits/re-submits a deliverable, the supervisor reviews the deliverable submissions/re-submissions and provides feedback. Deliverable re-submission occurs until all of the feedback criteria has been met.

To facilitate this, the system is designed to allow supervisors to create, update and delete tasks for specific projects. They can also lock a given task to prevent re-submissions while providing structured feedback on the latest deliverable. Once feedback has been provided, it will be displayed in tabular format, visible to both types of users.

Students will be able to first upload a deliverable to a staging area before finalizing the submission. The concept of a staging area before submission is featured in Google Classroom’s assignment submission process. To ensure that the student has taken the feedback into consideration, re-submissions are prevented until the feedback criteria of the previous submission are either all met or overridden.

2.2.6. AI Feedback Compliance Analysis

After providing feedback, the supervisor expects the student to iterate over it and produce a version of their work aligned with the correct requirements. As highlighted in preceding sections, manually verifying whether the feedback given has been applied for each re-submission can be time-consuming and therefore needs to be automated using artificial intelligence.

The AI model will have to perform both semantic and structural comparisons between the submitted and the new deliverable to be re-submitted, track the changes and evaluate whether the feedback criteria has been met.

Since AI models are usually stochastic, the system will thus grant the permission to students to override the status of any unmet criteria. For transparency, the status of the criteria will be set to overridden.

2.2.7. Meeting Scheduling

Expanding on Bheeky's (2013) meeting scheduling module, the meeting scheduling component aims to streamline the traditional way of scheduling appointments. Supervisors will have a calendar which is shared with all students under their supervision, thus ensuring visibility by allowing students to view their supervisor's meetings.

Meeting stakeholders are categorized either as organizers or attendees, with each stakeholder having different permissions – organizers will be able to cancel their booked meeting(s) while attendees can accept and reject meeting invitations. To maintain administrative control, students will only being able to book meeting appointments with their supervisor.

Moreover, in order to enhance usability, the scheduler will assign different colors to distinguish between meeting events (accepted, rejected, organized, unrelated)

2.2.8. Notification & Reminder System

To keep the students and supervisors aware of important events, reminders will be sent as email notifications. Notifications represent project and meeting status updates that does not necessarily warrant the attention of a user but facilitates keeping track of changes.

Both notifications and reminders will be displayed in a section visible to the user when they log back into the system.

Chapter 3 - Analysis

This chapter will focus on the analysis of the system proposed in the previous chapter, evaluating the architectural considerations of the system and the technologies that will be used as the basis of their implementation.

It will first consist of a list of the architectural considerations identified and then followed by an in-depth analysis of the potential solutions/paradigms that will be used for each one of them.

3.1. List of Key Architectural Considerations

The following is a list of the most important aspects of the system to be considered:

- Application Architecture
- Server Side Web Development Framework
- Client Side Web Development Framework
- Database
- Mail Service
- Temporal Data Management Strategy
- Artificial Intelligence Model for Feedback Compliance Checking

3.2. Architectural Considerations

This section will discuss the architectural considerations identified in the previous one, dedicating three sub-sections for each consideration:

- Evaluation Criteria
- Comparison of the Features of the different Choices
- Justification for the Choice selected

3.2.1. Application Architecture

The application architecture serves as the foundation of the entire system – every subsequent considerations and design decisions will be dependent on it. Thus, a poor choice will significantly increase the cost of the evolution of the software later in the development life-cycle.

3.2.1.1. Evaluation Criteria

The following evaluation criteria have been selected and filtered from common criteria used in choosing the correct web application architecture *(Bass, Clements & Kazman, 2021)*, in order to suit the context of the solution to be developed.

- **Scalability –** It broadly refers to the ability to increase the system/component's capabilities in order to handle an increasing number of users.
- **Fault Isolation/Availability –** It is to the extent the system's features remain available to users in the event of faults.
- **Development Speed –** It describes the development effort and speed in order to provide a fully-functional system

3.2.1.2. Application Architecture Comparison

The three most common architectures for developing web applications are presented for selection:

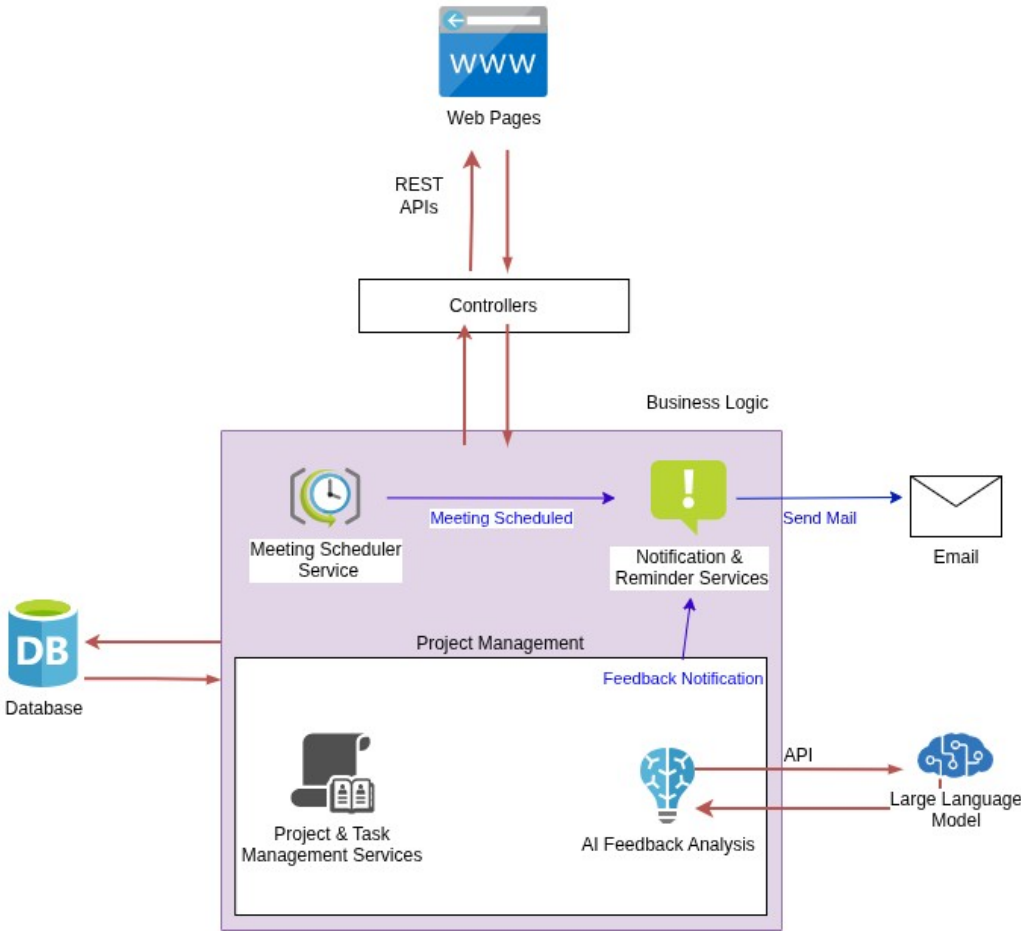
	Monolithic	**Microservices**
Description	Monolithic architecture is where all components are tightly coupled and run as a single service	Microservices is a design that structures an application as a collection of smaller, independent services (Newman, 2015)
Scalability	Lack of scalability since scaling one component requires scaling the entire application due to their tight coupling	Highly scalable since every service is decentralized
Fault Isolation/ Availability	A single failure can bring the entire application down	Faults only affect the services they occur in without impacting the rest of the application
Development Speed	Easiest and Fastest to develop and debug early since services interact directly without needing an interface to decouple them	High development complexity and slowest development speed since services are isolated and require managing their orchestration.

3.2.1.3. Application Architecture Choice

Micro-services will be chosen as the application architecture due to the following reasons:

1. Due to application requirements of handling large user traffic, it will need be to be scalable both vertically and horizontally *(see appendix).* As such, the service components needs to be loosely coupled to allow the system design to evolve to a microservices architecture later as an example of horizontal scaling.
2. While all three architectures adhere to the logical separation of concerns presented by the 3-Tier Architecture *(see appendix)*, SOA is preferable over Monolithic design since a fault in one layer (e.g presentation layer) does not affect other layers.

3. SOA is also preferable compared to microservices architecture since it has a lower development complexity and faster development and debugging speed.



3.2.2. Server-Side Web Development Framework

The programming language/framework to be used will directly affect how the components will be built, the ease of inter-component communication and how suitable will the application be for supporting the functional and non-functional requirements of the system.

3.2.2.1. Evaluation Criteria

The choice of the development framework is dependent on the following criteria commonly used to demarcate programming languages:

- **Programming Language Features –** Characteristics of the underlying language such as type safety, garbage collection for memory safety and automatic memory management, whether the language is compiled or interpreted.
- **Ecosystem -** Availability of third-party libraries for facilitating the development of custom application components, support for integration with external services such as databases and APIs and documentation quality.
- **Built-in Abstractions –** Provision of in-built features that handle tasks (such as sanitizing input to guard against XSS *(see appendix)*) without having to write the low level logic. This reduces development time and human error for critical code sections.
- **Response Latency & Concurrency –** If the framework offers a web server, response latency is the speed it can process requests. On the other hand, concurrency is the ability of the server to handle multiple simultaneous user requests.

3.2.2.2. Development Framework Comparison

The three most suitable web development frameworks have been identified and listed below for comparison:

	.NET Core (C#)	**Laravel (PHP)**	**Next.js (Javascript/Typescript)**
Programming	Its underlying language,	PHP is an interpreted	Javascript, officially known

Language Features	C# is a strongly-typed, memory-safe and compiled high level language with support for a wide variety of operations and features.	language with optional type safety, built specifically for web development. It has libraries and language features to support tasks such as session management & input validation.	as ECMAScript, is an interpreted (JIT-compiled *(see appendix)*) language that is used for both client-side and server-side logic. Its flexible syntax allows for rapid prototyping. Type safety is provided by Typescript, which is a super set of Javascript.
Ecosystem	.NET offers an ecosystem of packages via Nuget package manager, including both community-developed and Microsoft-backed libraries.	Laravel provides an ecosystem of both official and community-developed packages through the Composer dependency manager for PHP.	Next.js makes use of the node package manager (npm) which manages the largest registry of packages and therefore, provides the largest ecosystem of readily available tools.
Built-in Abstractions	.NET has built-in middlewares for Authentication, logging and testing with built-in features against CSRF & XSS *(see appendix)* attacks. (ASP.NET Core, 2025) It also provides .NET Entity Framework ORM *(see appendix)* for database abstraction.	Laravel has out-of-the-box CSRF *(see appendix)* protection and middlewares for request JSON schema validation and logging. (Laravel Docs, 2025) It provides Eloquent ORM for database abstraction.	The built-in abstractions provided by Next.js are primarily concerned with optimizing performance for website rendering on the client.

Response Latency & Concurrency	C# is compiled into a highly optimized intermediate language and then executed, ensuring high performance. Furthermore, its multi-threaded nature allows it to efficiently handle multiple I/O requests simultaneously. As a result, it can tolerate high loads.	PHP has high overhead per request and thus a high response latency due to architectural considerations and blocking I/O operations. Laravel Octane is required to achieve lower latencies and higher concurrency support via persistent memory and asynchronous processing. (Laravel Octane, 2025)	The underlying runtime, Node.js uses a single-threaded event loop for non-blocking IO, which is efficient for concurrent network requests (Node.js Docs, 2025) Next.js improves website performance for the client by providing features such as Server-Side rendering *(see appendix)* (Next.js Docs, 2025)

3.2.2.3. Choice of Framework

.NET core will the be the most appropriate choice for the development framework based on the criteria given and the following additional justifications:

I. The web platform must be designed handle thousands of concurrent users, especially when nearing the submission deadlines, during which deliverable submissions may soar considerably. This will be a common criteria in determining the most suitable technology to use. Buljić et al. (2025) showed that .NET Core consistently outperforms other web frameworks in response latency and handling concurrent HTTP requests.

II. The ecosystem and built-in abstractions for .NET are more aligned with the requirements of the project – robust middlewares for authentication, testing and logging will reduce development time considerably while its ecosystem has official libraries for a range of tasks including LLM interactions and cron job scheduling.

Framework	Request Type	Average (ms)	Min (ms)	Max (ms)	Std. Dev.	Throughput (req/sec)	Received KB/sec	Sent KB/sec	Avg. Bytes
Django	HTTP POST	52	11	115	13.43	5.1	1.87	1.12	377.8
	HTTP GET	12	3	63	10.75	5.1	55.74	0.63	11257.3
	Total	32	3	115	23.75	10.1	57.6	1.76	5817.5
.NET	HTTP POST	1	0	37	1.84	5.1	1.25	1.13	250
	HTTP GET	1	0	5	0.58	5.1	53.65	0.63	10765
	Total	1	0	37	1.36	10.2	54.87	1.76	5507.5
Express.js	HTTP POST	12	6	40	3.34	5.1	1.41	1.12	282.8
	HTTP GET	3	1	9	1.32	5.1	55.96	0.63	11247.7
	Total	7	1	40	5.24	10.2	57.35	1.76	5765.2

Fig 2. Metrics made in Apache JMeter

3.2.3. Database

The database is a critical component in ensuring the persistence of data as per the requirements of the system. Its importance is comparable to the choice of the system architecture because a poor choice may become a bottleneck in the system's performance and scalability.

3.2.3.1. Evaluation Criteria

The database to use will be selected based on the criteria as commonly stipulated by database vendors *(Belagatti, 2025)*, curated to suit the use cases of the system:

- **Performance**
- **Security & Reliability**

- **Scalability**
- **Cost & Licensing**
- **Development Framework Integration**
- **Large Object Storage & Retrieval**

3.2.3.2. Database Comparison

	PostgreSQL	Microsoft SQL Server	MySQL
Performance	PostgreSQL offers higher performance than MySQL for frequent update and delete operations. It offers less performance than MySQL for frequent read operations. (AWS - PostgreSQL vs MySQL, 2025) Supports more advanced indexing techniques as compared to MySQL.	SQL Server offers the highest performance due to several optimization capabilities. Supports advanced indexing techniques	MySQL is less performative on for heavy update workloads than PostgreSQL. It outperforms PostgreSQL in frequent read operations (AWS - PostgreSQL vs MySQL, 2025)
Security & Reliability	ACID *(see appendix)* compliant with support for advanced indexes for reliability	Extensive feature set for ensuring security and reliability including ACID compliance	ACID compliance depends on the storage engine used
Scalability	Scales both vertically (increasing resources for a	Scales both vertically and horizontally	Scales better horizontally than vertically

	single server) and horizontally (increasing number of servers)		
Cost & Licensing	Free & Open-source	Requires licensing for large scale enterprise environments	Free & Open-source
Development Framework Integration	.NET Core support provided by Npgsql, which is an open-source data provider package.	Microsoft provides an official data provider for SQL Server and native libraries are specifically optimized for interaction with SQL Server.	.NET Core support provided by MySqlConnector, which is open-source.
Large Object Storage & Retrieval	PostgreSQL uses TOAST (PostgreSQL: Docs – TOAST, 2025) optimization to compress large objects before writing to disk. TOAST also makes retrieval slightly faster than MySQL by making records cache-friendly.	Less storage compression efficiency offered as compared to PostgreSQL.	No inherent compression techniques for large objects and has less storage efficiency than PostgreSQL.

3.2.3.3. Choice of Database

In spite of SQL Server being the most advanced of the databases listed, PostgreSQL will be selected due to SQL Server's expensive licensing costs. The decision is also based on the additional reasoning:

I. Since deliverables may often be text documents ranging from 5-150 pages long, PostgreSQL will be more suitable for handling large text objects due to its TOAST

optimization features, optimizing storage utilization. Read operations will also be slightly more efficient.

II. Despite the consensus that web applications have a higher read-write ratio, the number of write operations can significantly increase when taking into account the meeting scheduling and reminder system. Therefore, PostgreSQL will be most suited for handling this change in workload.

3.2.3.4. Object Relational Mapper (EF Core)

3.2.4. Artificial Intelligence for Feedback Compliance Checking

The selection of the AI component determines the accuracy and effectiveness of the compliance checks. The process in question – feedback compliance – is a natural language inference/textual entailment task where a model is required to determine the relationship between a premise (document text) and the hypotheses (feedback criteria).

Large Language Models are state of the art systems used to perform this task.

3.2.4.1. Evaluation Criteria

The large language models will be selected based on the criteria

- **Performance**
- **Latency / Speed**
- **Context Window**
- **Licensing & Cost**

3.2.4.2. Comparison of Large Language Models

Large Language Models	**Anthropic: Claude 3 Opus**	**OpenAI: GPT-4o**	**Gemini 2.5 Flash**
Performance	Top-tier performance on many LLM benchmarks	Top-tier performance	High performance on benchmarks
Latency / Speed	High latency	Low latency	Lowest latency
Context Window	200,000 tokens	128,000 tokens	1,000,000 tokens
Licensing & Cost	Highest cost per token	Moderate costs per token	Lowest costs per token

3.2.4.3. Choice of Large Language Model

Gemini 2.5-Flash will be chosen because it is the best fit for the most of the criteria set

3.3. Requirements Specification

This section will specify the functional & non-functional requirements of the system.

3.3.1. Functional Requirements

The functional requirements of the system will be grouped based on their categories as sub-sections

3.3.1.1. User Management

Requirement ID	Description
FR1	The system shall have as the initial user an

	administrator, who shall be able to view, create users with their credentials and assign their roles.
FR2	Users shall be able to prove their role either as a (I) student (ii) supervisor or (iii) administrator through the use of their credentials in the system.

3.3.1.2. Project Management

Requirement ID	Description
FR3	Supervisors shall be able to view, search, create, update or archive projects. Students shall only be able to view their corresponding project or a create a single project.
FR4	A supervisor shall be able to assign a student to a created project or join a project created by a student.
FR5	A project shall consist of a list of Task entries arranged in reverse chronological order. Each Task entry shall allow a user to (I) access the Task and (ii) display its corresponding deadline and status from missing/completed.
FR5	For a project, the supervisor shall be able view, create, update and delete Tasks for deliverable submissions.
FR6	Creation and modification of a Task in a project by a supervisor shall create a reminder based on the deadline of the Task

Requirement ID	Description
	and notify the student assigned to the project by email.
FR7	Supervisors and students shall be able to generate and download a progress log report of the project based on prior meetings conducted and deliverable submissions.

3.3.1.3. Deliverable Task

Requirement ID	Description
FR8	A Task shall be an interface that provides the deliverable file and displays the corresponding feedback criteria for the current deliverable version.
FR9	The system shall provide a student with a function to upload or remove a deliverable file pre-submission for a specific Task.
FR10	The system shall provide a student with a function to submit the deliverable file for the Task.
FR11	The system shall store the latest version of the submitted deliverable file and its corresponding feedback criteria.
FR12	The system shall provide the supervisor with a function to submit a list of feedback criteria for the latest deliverable submitted in a Task.
FR13	The system shall prevent a student from removing/modifiying the submitted deliverable

Requirement ID	Description
	file while the supervisor provides feedback.

3.3.1.4.

3.3.1.5. AI Feedback Compliance Checker

Requirement ID	Description
FR14	During re-submission of a deliverable, the output of the LLM during feedback submission shall be used as context with the text delta of the new submission to evaluate the feedback criteria.
FR15	The system shall display a loading state for all LLM invocations until the operation either terminates successfully or unsuccessfully.
FR16	The system shall send email notifications to the supervisor and student for deliverable submission and feedback provision respectively.

3.3.1.6. Meeting Scheduling

Requirement ID	Description
FR17	The system shall allow all booked meetings to be visible to all users.
FR18	A user (organizer) shall be allowed to book a meeting between another user (attendee), with the two parties being strictly of a supervisor-student pair.
FR19	Upon booking a meeting, a reminder shall be

Requirement ID	Description
	created for both the organizer and attendee and an email is to be sent to notify the attendee.
FR20	The attendee shall be allowed to accept, reject or postpone a meeting request, creating a reminder for both parties and notifying the organizer by email.

3.3.2. Non-Functional Requirements

Requirement ID	Description
NFR1	The system shall enforce Role-Based Access Control *(see appendix)* by only authorizing users to access resources and views based on the permission sets associated with their roles.
NFR2	The system shall provide a user-friendly interface.
NFR3	All server requests except for LLM invocations shall have a response time of less than 500ms.
NFR4	The server CPU usage shall be less than 80% for 1000 concurrent users.

Chapter 4 - Design

This section will provide the necessary specifications for the system's design, including both structural and behavioral UML diagrams, arranged in the following order:

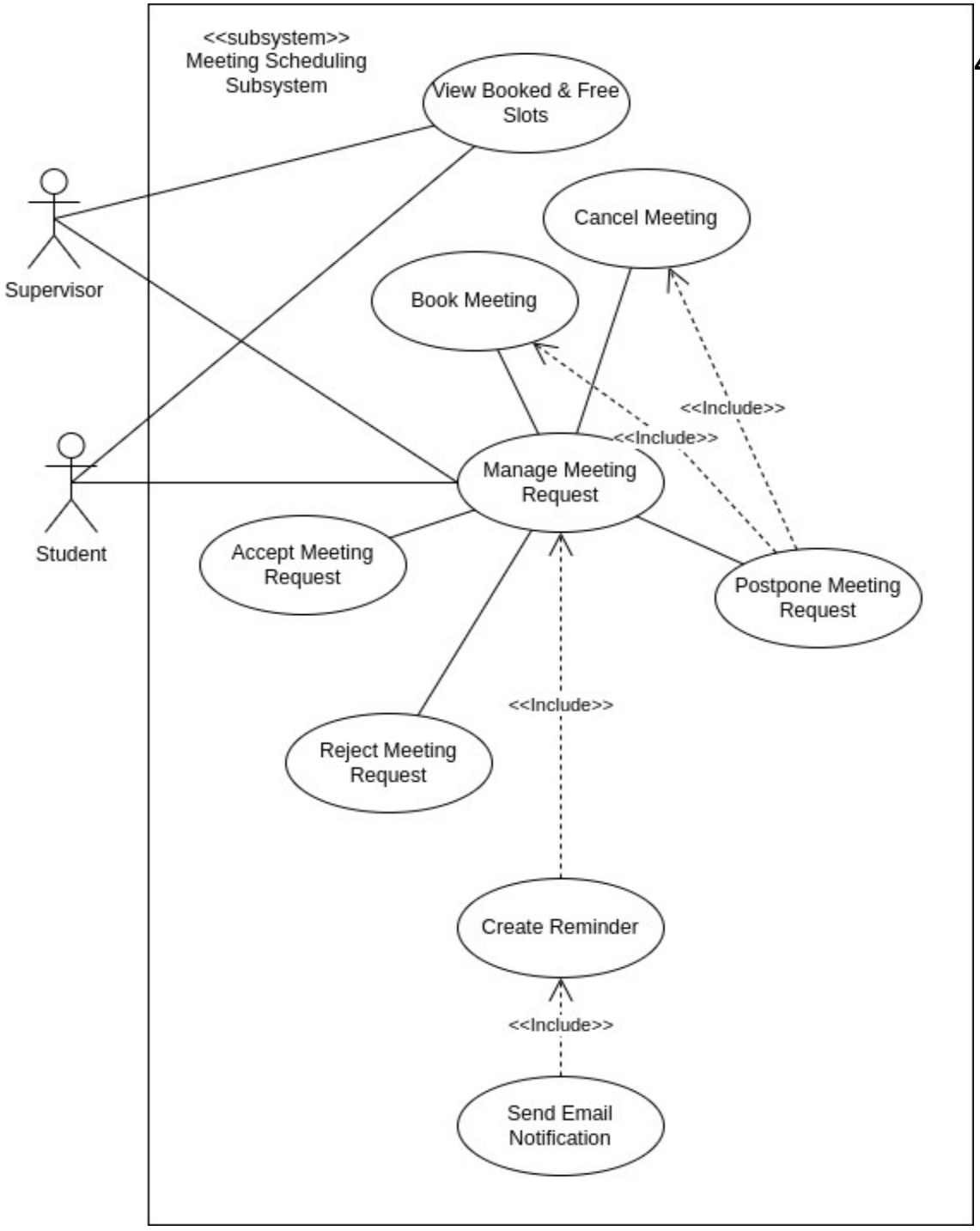
1. Use Case Diagrams
2. Class Diagrams
3. Sequence Diagrams
4. Entity Relationship Diagram

4.1. Use Case Diagrams

From the user’s perspective, the system can be divided into 4 sub-systems:

- Meeting Scheduling Subsystem
- Project Management Subsystem
- Deliverable Submission Subsystem
- Administrator Subsystem

4.1.1. Meeting Scheduling Subsystem



4.1.2.

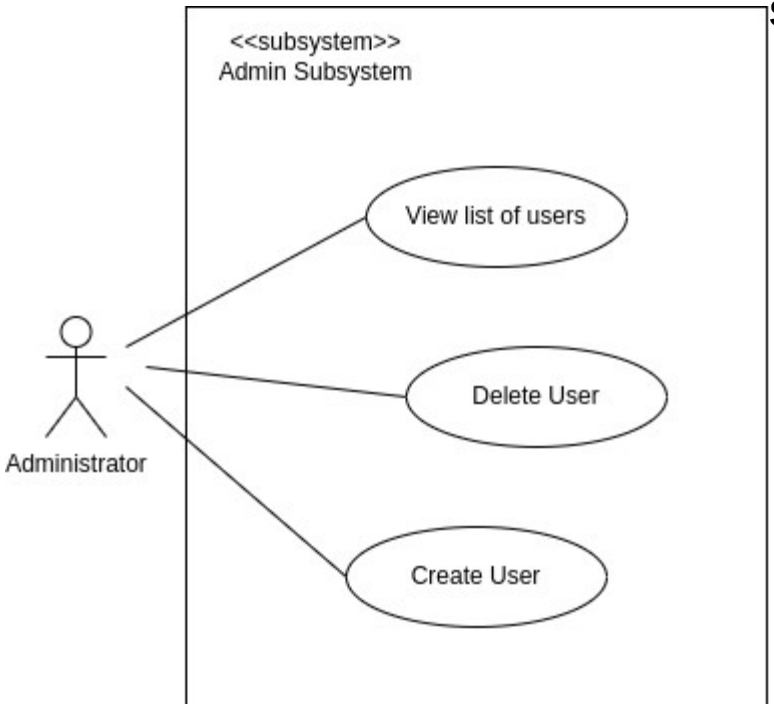
4.1.3. Project Management Subsystem

4.1.4. Deliverable Task Submission Subsystem

Figure 8: Use Case Diagram - Deliverable Task Submission Subsystem

4.1.5.

Administrator



Subsystem

4.2. Class Diagrams

For readability, the class diagram of the application will be separated into smaller class diagrams, categorized as follows:

- **Class Diagram (Domain)**– It depicts the domain model (see appendix) of the application.
- **Class Diagrams (Subsystems)** - Class Diagrams that depict the relationship (dependencies & associations) between the classes at the application level.

  They are further arranged as follows:

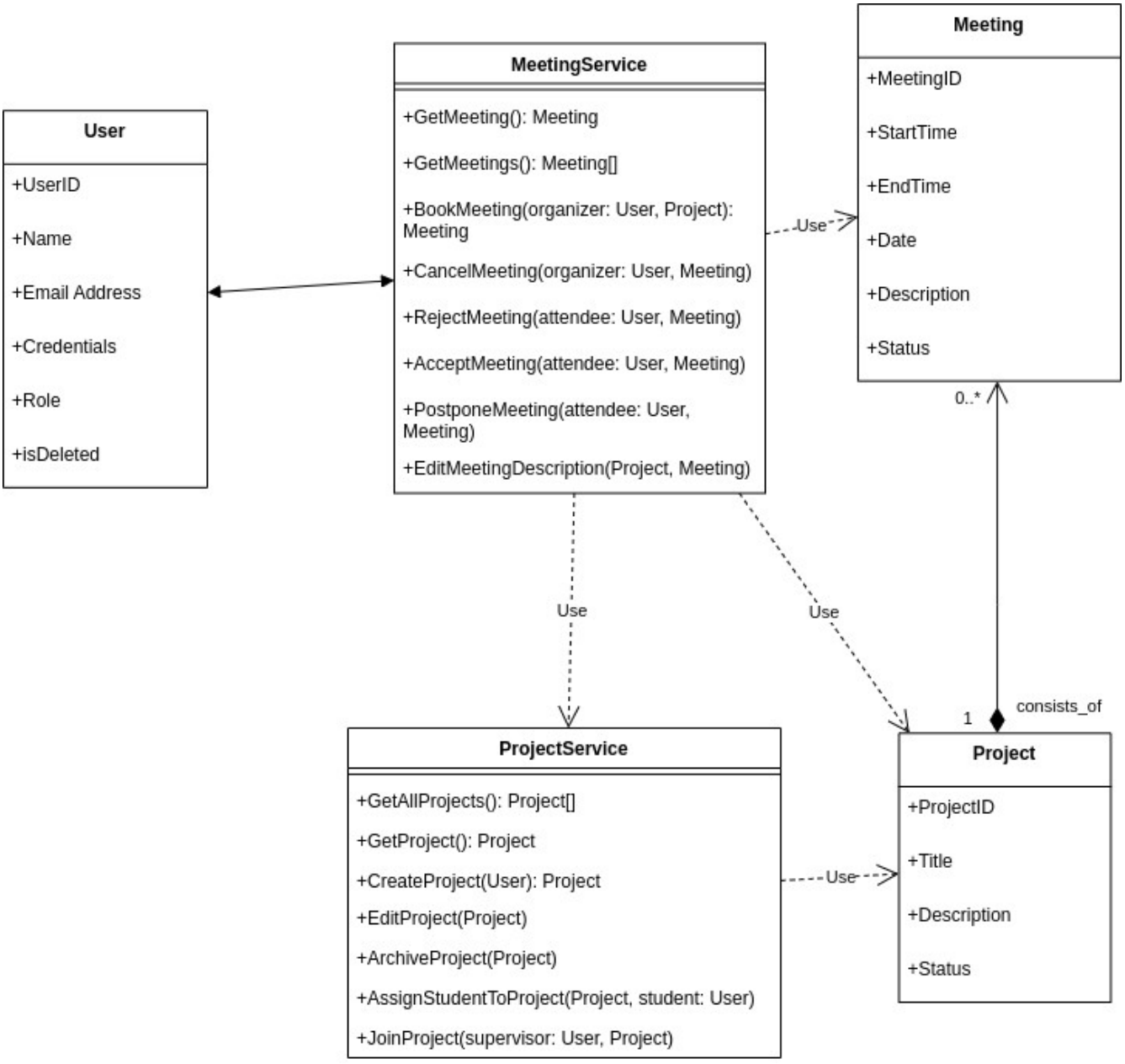
1. Meeting Scheduling Subsystem
2. Project Management Subsystem
3. Deliverable Task Submission Subsystem
4. Progress Log Subsystem
5. Notification & Reminder Subsystem
6. Administrator Subsystem

In order to reduce clutter, the method signatures for the classes will intentionally be left as incomplete, only highlighting the dependencies instead. The actual method signatures are covered in the following section on Sequence Diagrams.

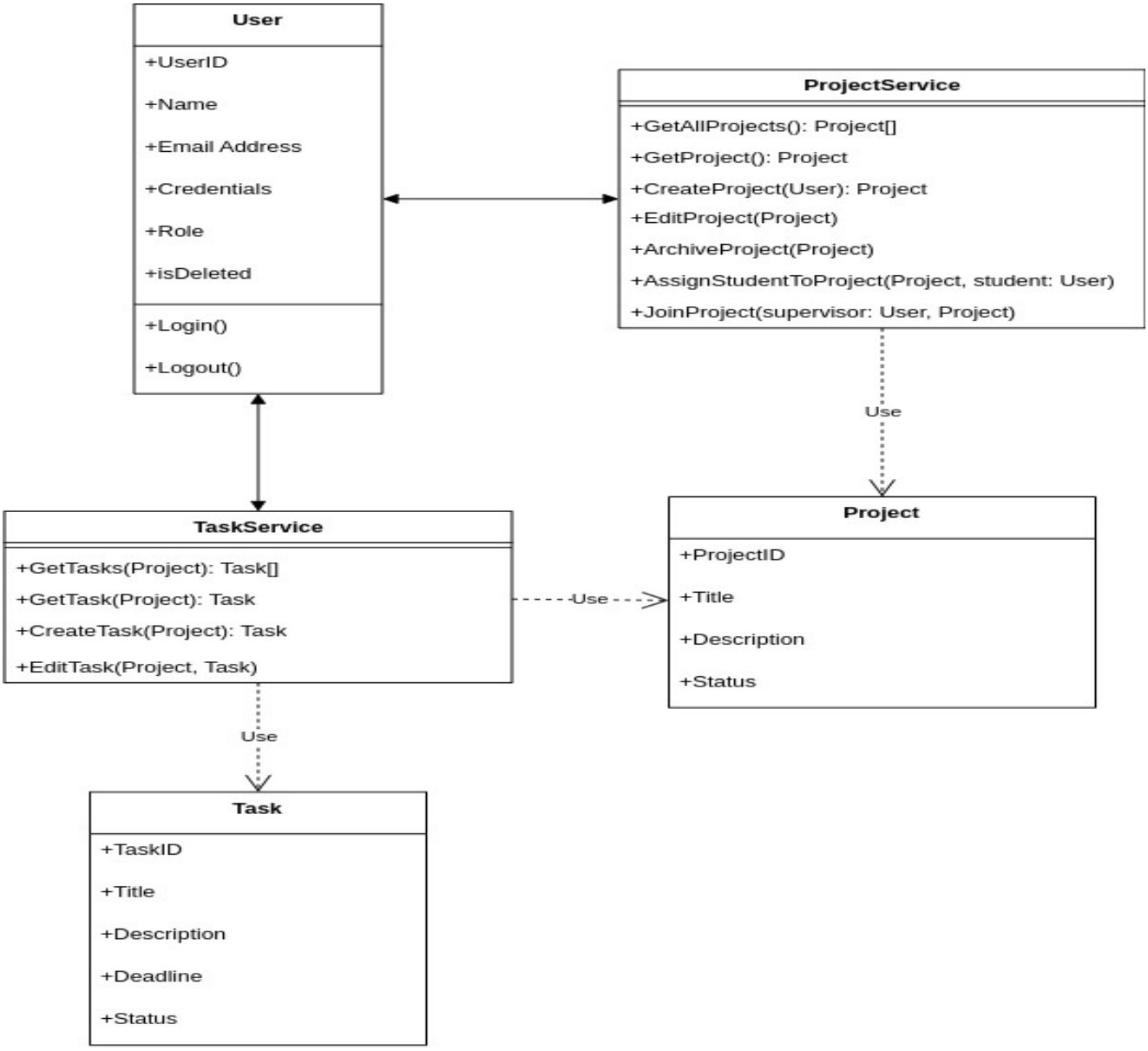
4.2.1. Class Diagram (Domain)

Figure 10: Class Diagram (Domain)

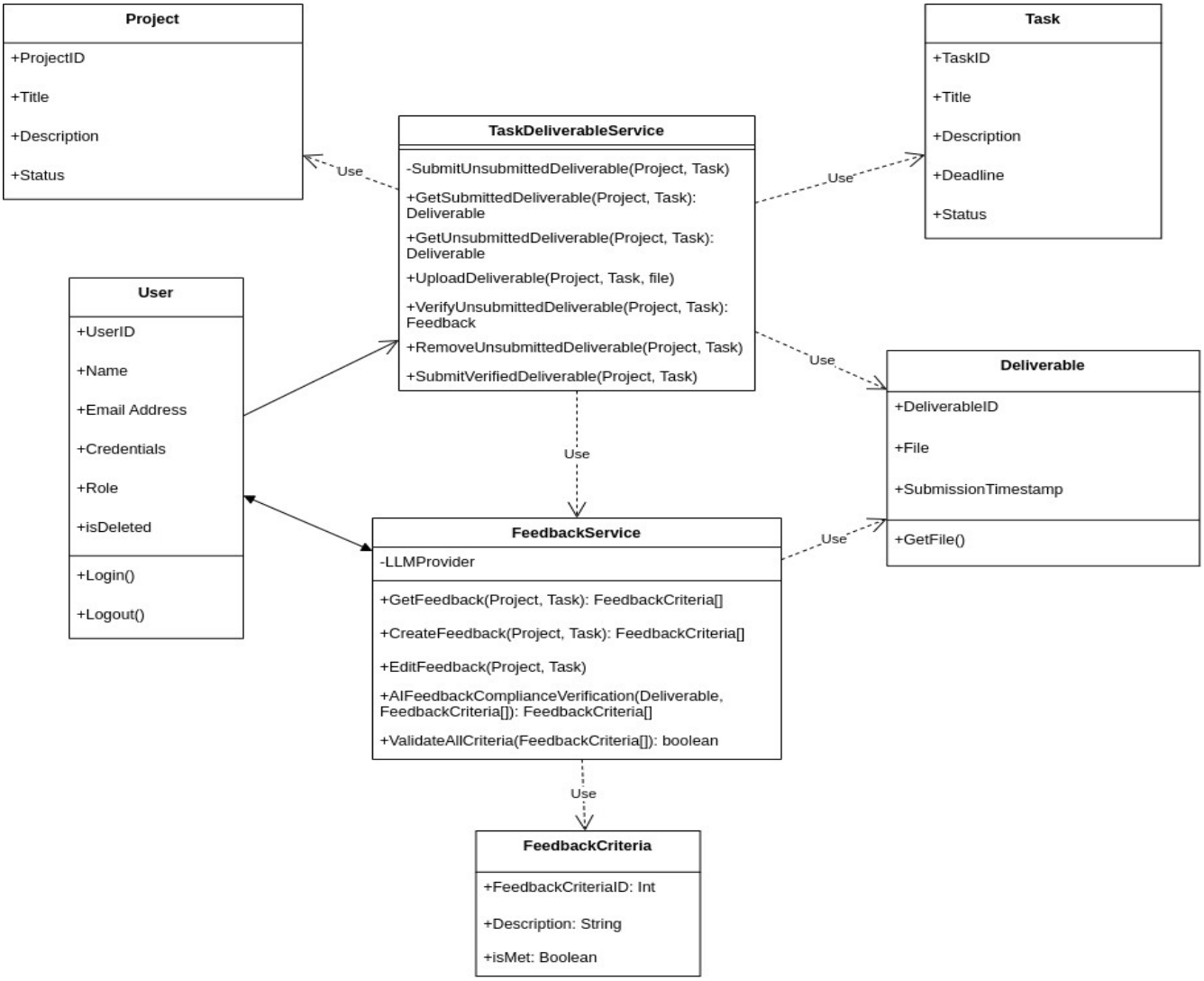
4.2.2. Meeting Scheduling Subsystem



4.2.3. Project Management Subsystem



4.2.4. Deliverable Task Submission Subsystem

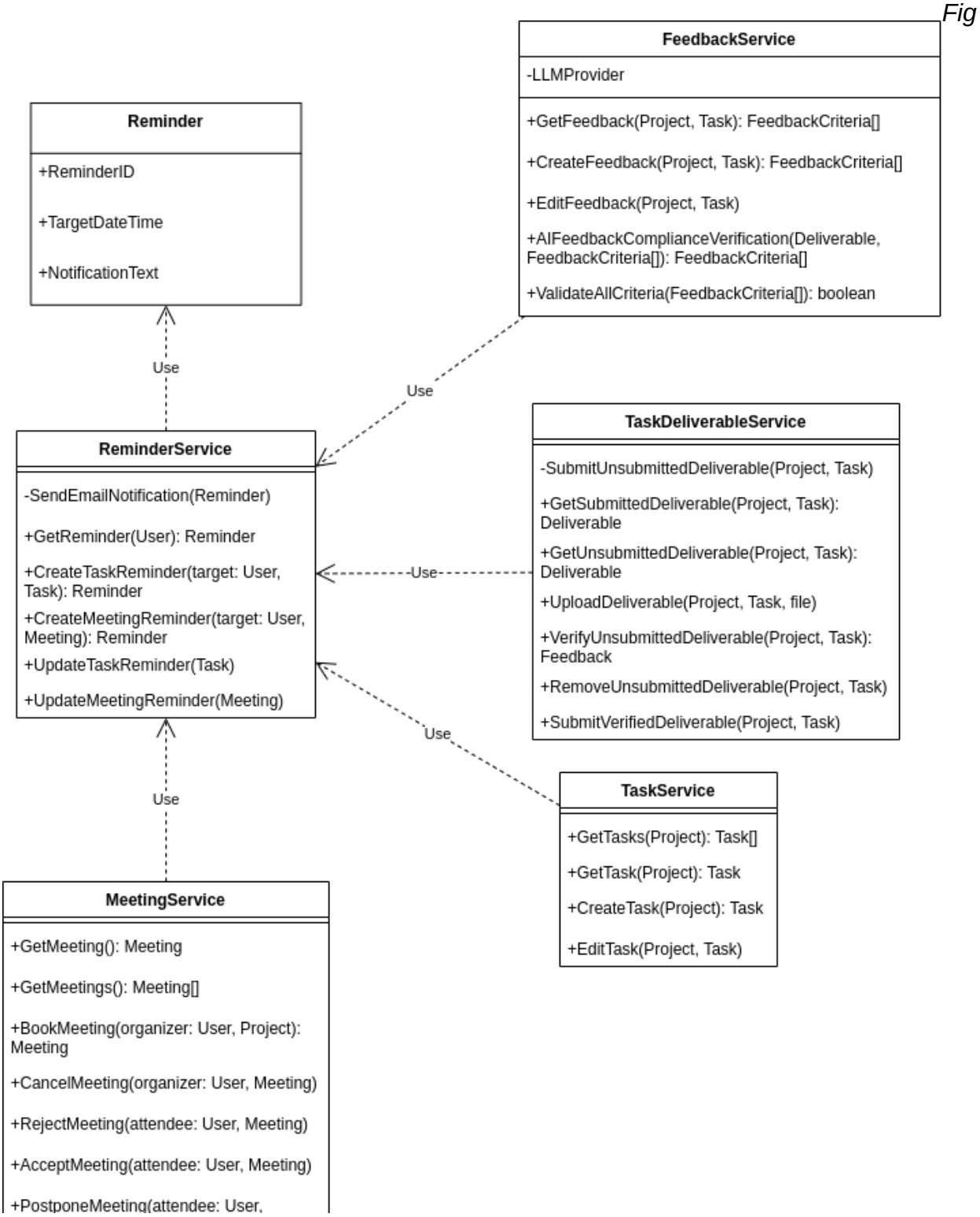


4.2.5. Progress Log Subsystem

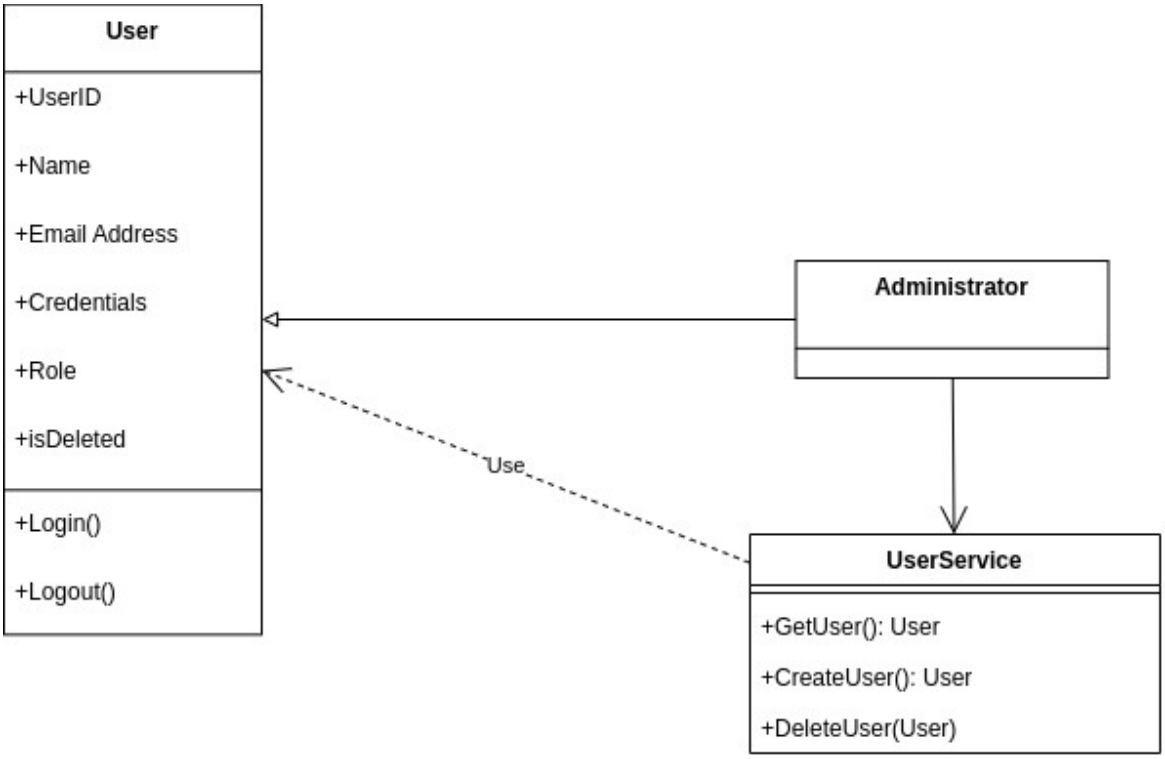
Figure 14: Class Diagram - Progress Log Subsystem

4.2.6. Notification & Reminder Subsystem

Fig



4.2.7. Administrator Subsystem



4.3. Sequence Diagrams

They aim to provide more detail to the class diagram methods shown in the previous section, highlighting further method invocations and object life-cycle management

Figure 17: Sequence Diagram: Meeting Scheduling Subsystem

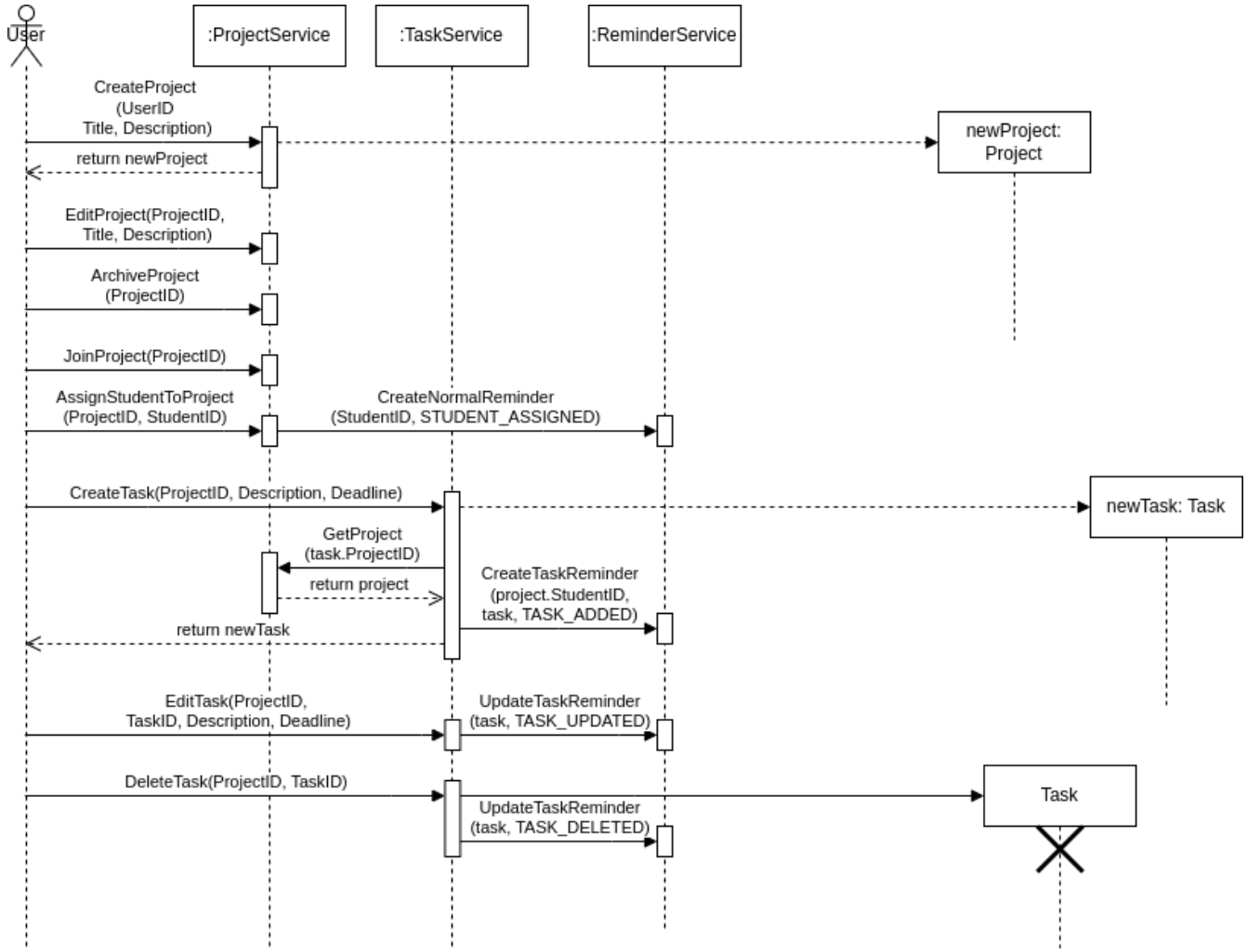
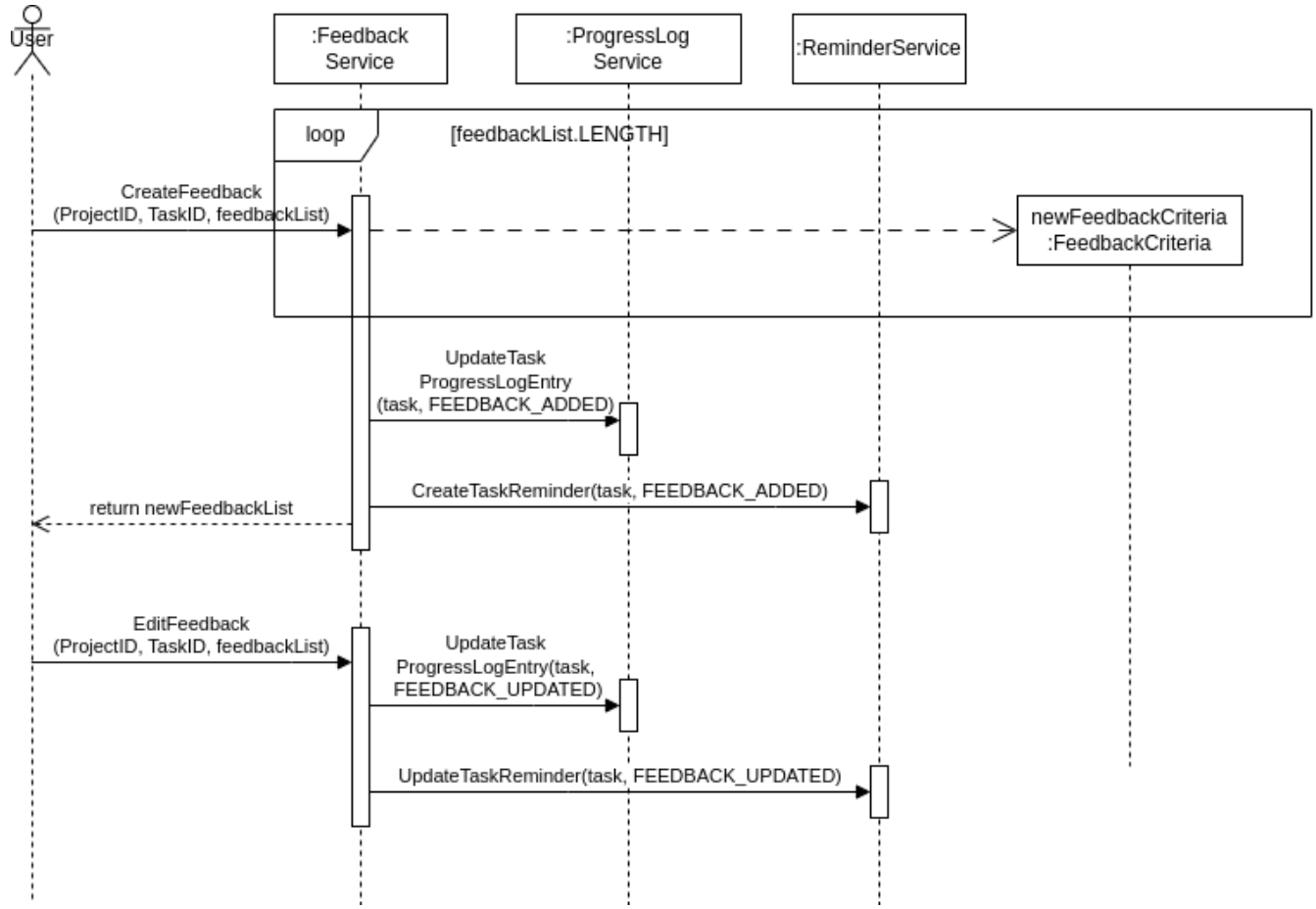

User
:ProjectService
:TaskService
:ReminderService
CreateProject
(UserID
Title, Description)
return newProject
newProject:
Project
EditProject(ProjectID,
Title, Description)
ArchiveProject
(ProjectID)
JoinProject(ProjectID)
AssignStudentToProject
(ProjectID, StudentID)
CreateNormalReminder
(StudentID, STUDENT_ASSIGNED)
CreateTask(ProjectID, Description, Deadline)
newTask: Task
GetProject
(task.ProjectID)
return project
CreateTaskReminder
(project.StudentID,
task, TASK_ADDED)
return newTask
EditTask(ProjectID,
TaskID, Description, Deadline)
UpdateTaskReminder
(task, TASK_UPDATED)
DeleteTask(ProjectID, TaskID)
Task
UpdateTaskReminder
(task, TASK_DELETED)

Figure 19: Sequence Diagram: Task Deliverable Management


User
:Feedback Service
:ProgressLog Service
:ReminderService
loop
[feedbackList.LENGTH]
CreateFeedback (ProjectID, TaskID, feedbackList)
newFeedbackCriteria :FeedbackCriteria
UpdateTask ProgressLogEntry (task, FEEDBACK_ADDED)
CreateTaskReminder(task, FEEDBACK_ADDED)
return newFeedbackList
EditFeedback (ProjectID, TaskID, feedbackList)
UpdateTask ProgressLogEntry(task, FEEDBACK_UPDATED)
UpdateTaskReminder(task, FEEDBACK_UPDATED)

4.4. Entity Relationship Diagram

The ERD depicts the database tables, their columns and their relations with one another based on the domain class diagram.

4.5. Interface Design

This section will contain the design for the interface

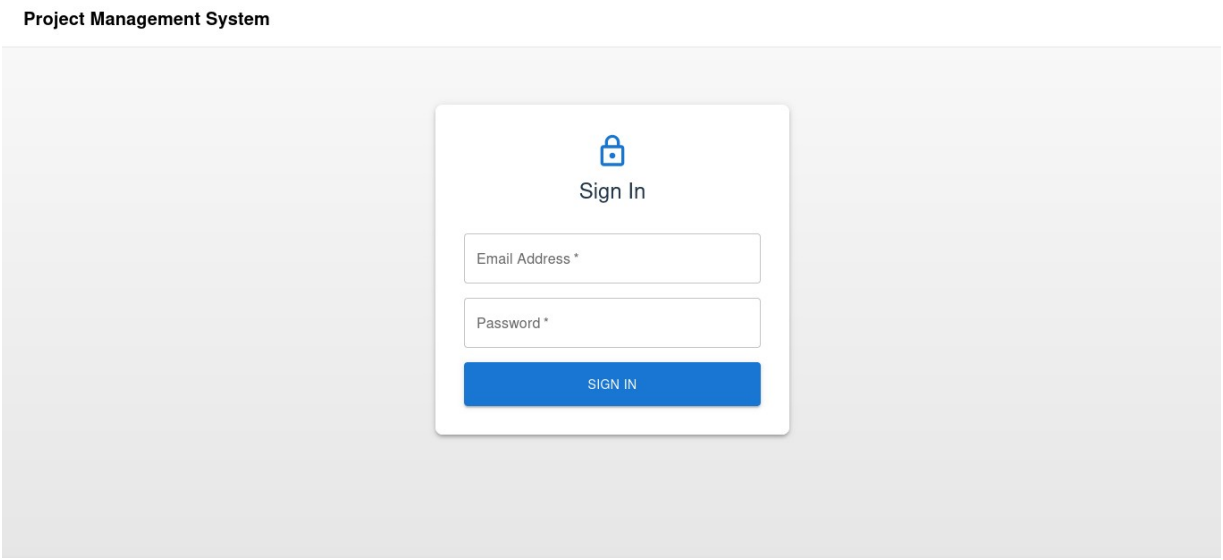


Figure 1: PMS - Login

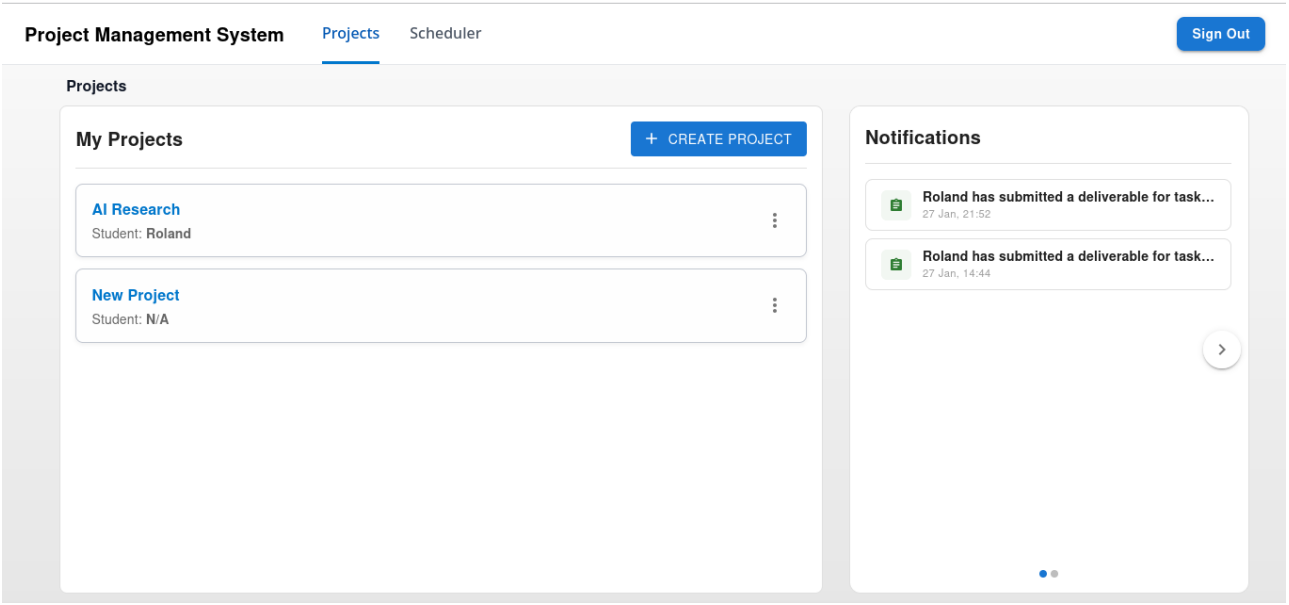


Figure 2: PMS - Projects Dashboard (Supervisor)

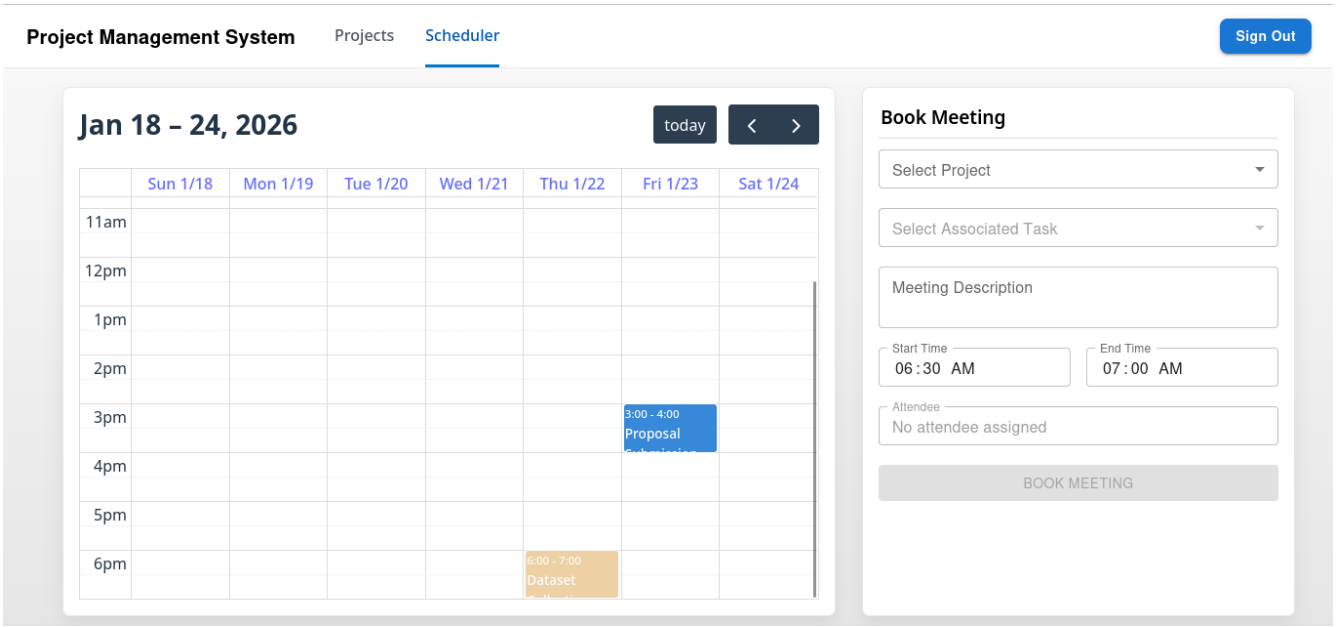


Figure 3: PMS - Meeting Scheduler

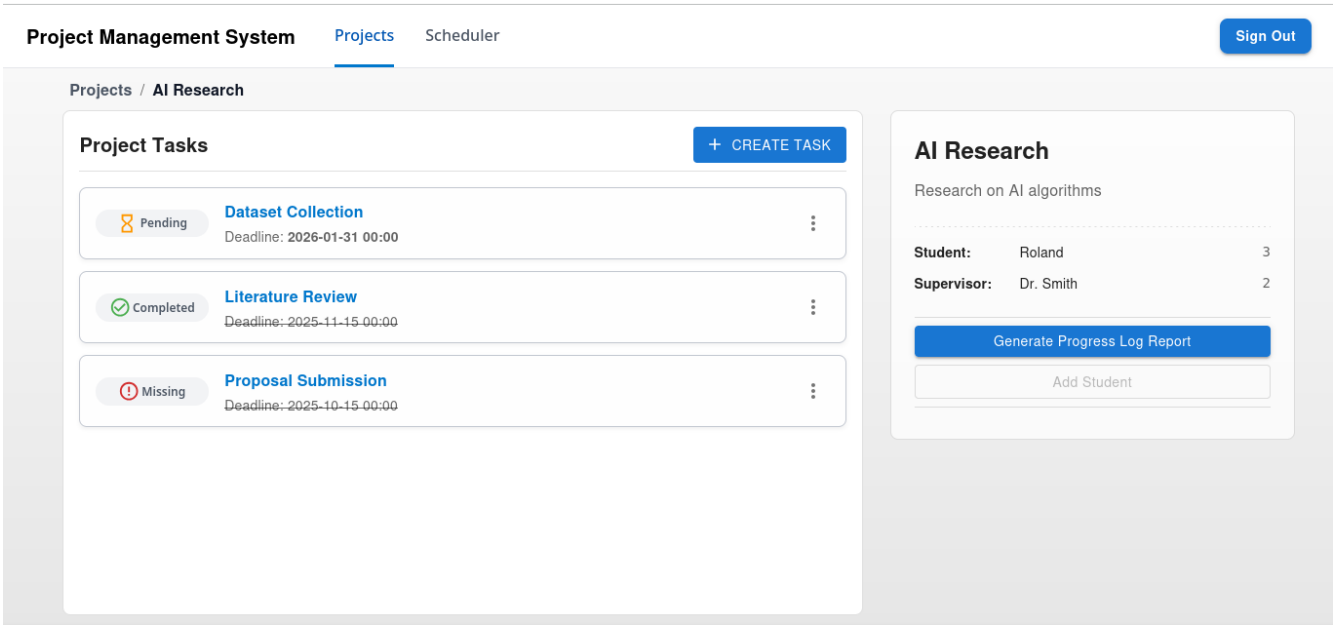


Figure 4: PMS - Tasks Dashboard (Supervisor)

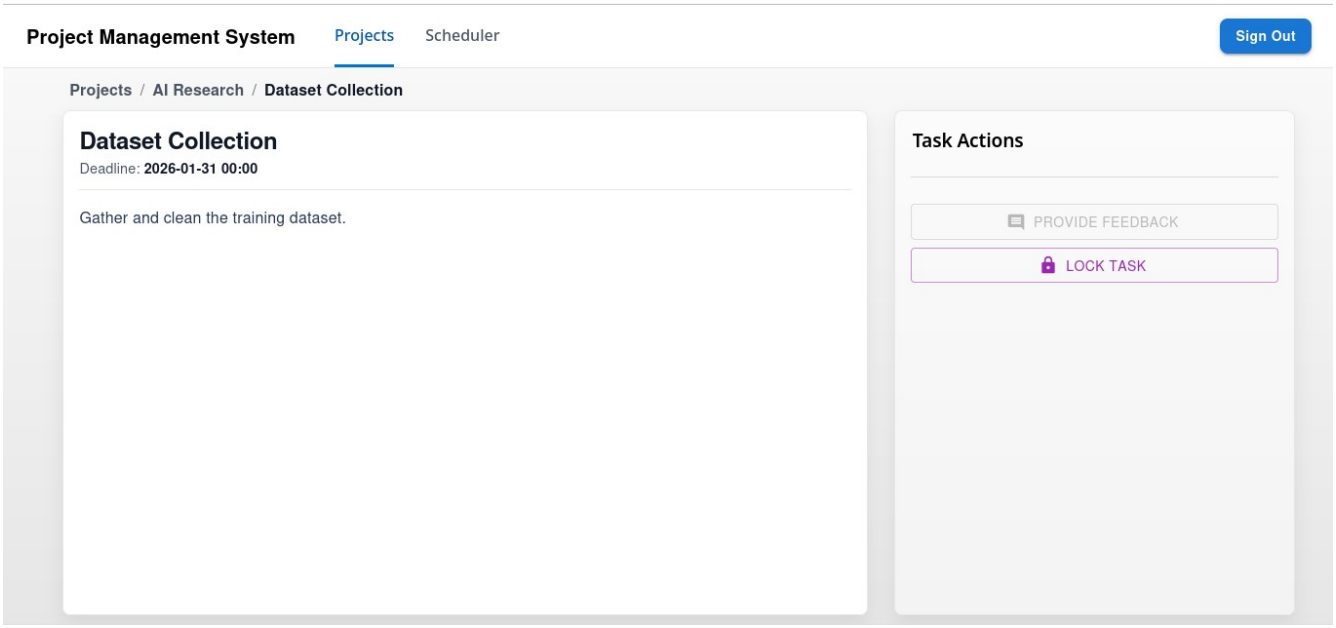


Figure 5: PMS - Task Display (Supervisor)

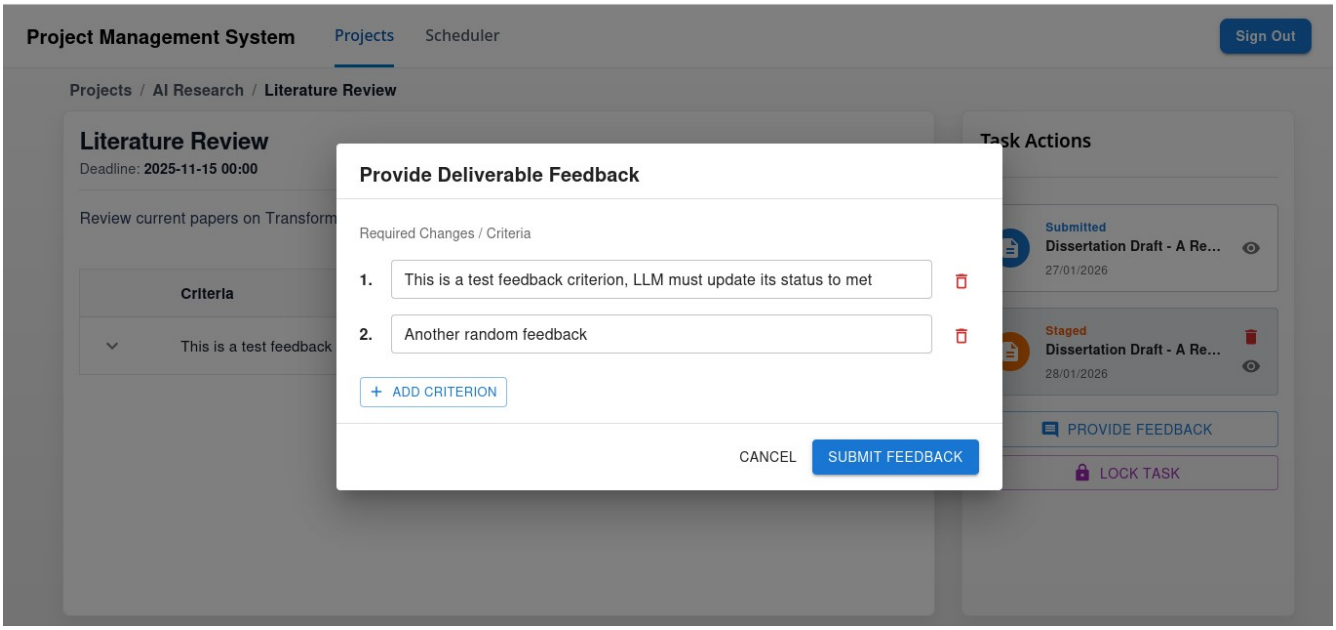


Figure 6: PMS - Task Display - Provide Feedback Modal (Supervisor)

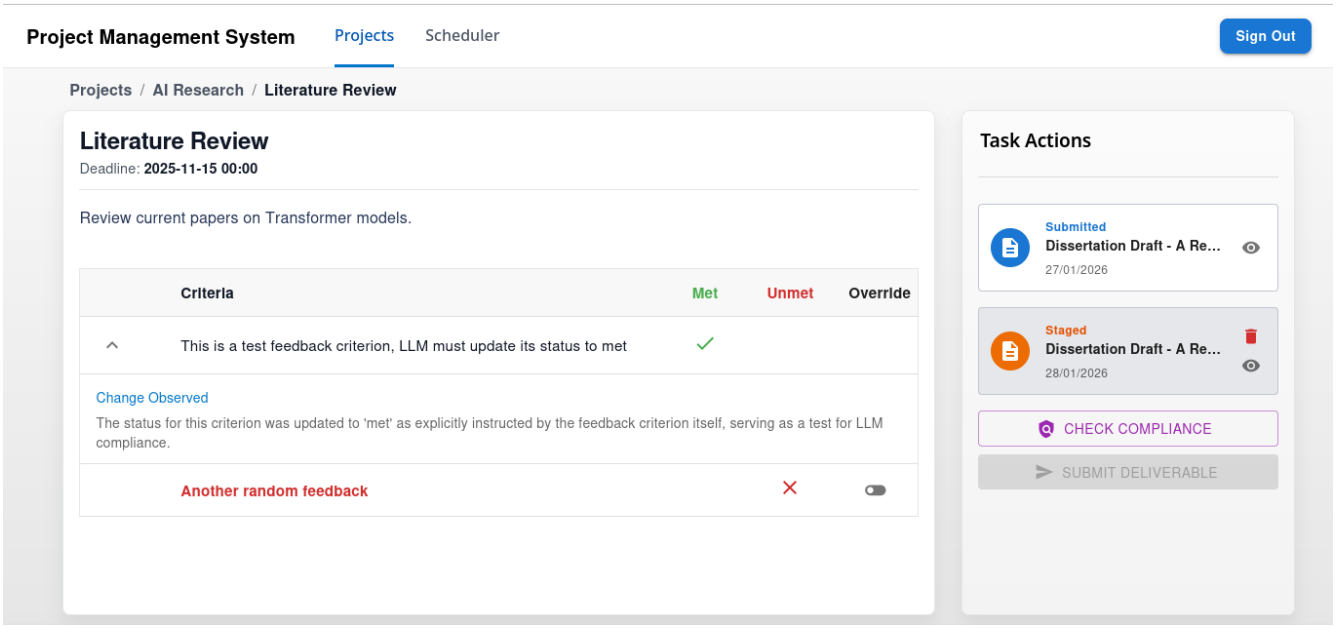


Figure 7: PMS - Task Display (Student)

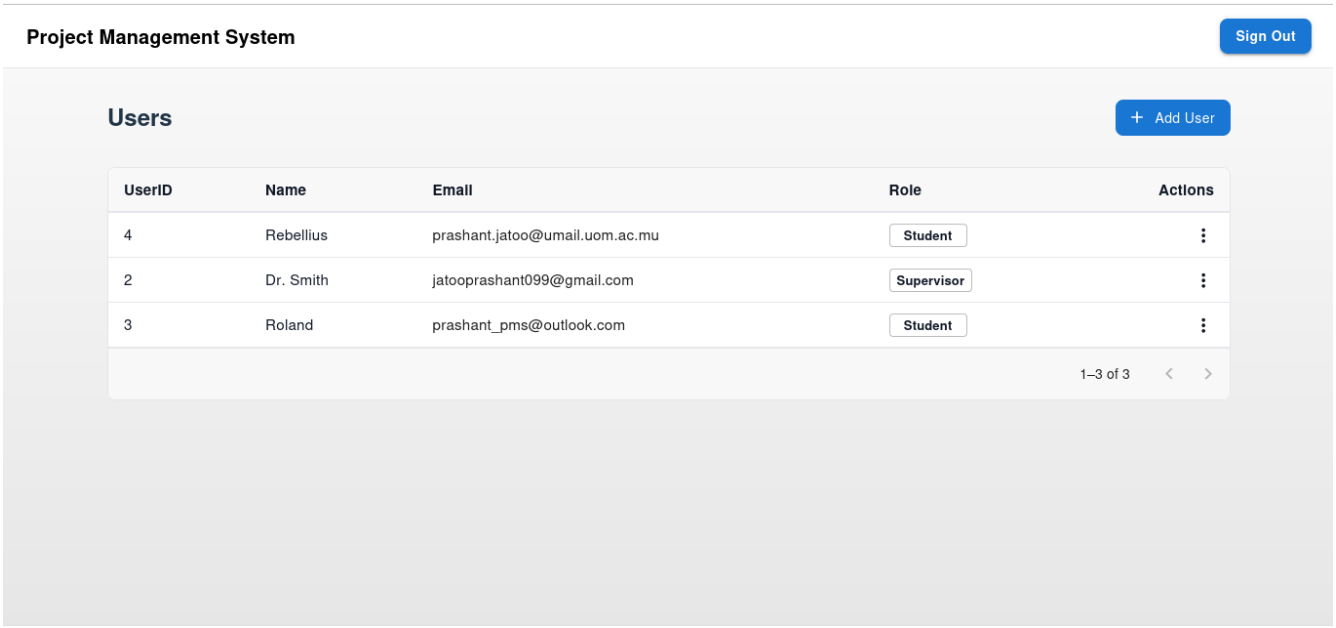


Figure 8: Admin Dashboard

Chapter 5 - Implementation

This chapter aims to provide details into the development of the code logic of the application from the design schema and architectural considerations.

The chapter will divided into 2 sections:

1. Project Structure
2. Pertinent Application Features

5.1. Project Structure

The project structure provides a holistic perspective on how the application works.

Since the application follows a micro-services architecture, the web client and application server are separate from one another.

5.1.1. Client

The client is developed as a Single-Page Application (SPA *(see appendix)*), which is enabled by the 'React Router' library.

- **components** – Stores React components that are used in the route files.
- **lib –** Contains shared utilities
- **providers** – Providers use the Context Provider API *(see appendix)* to manage global state, and make it available to downstream components.
- **routes –** Routes represent web pages that map to specific URLs.
- **main.tsx** – The entry-point for the React application. It defines the component hierarchy and includes route definitions.

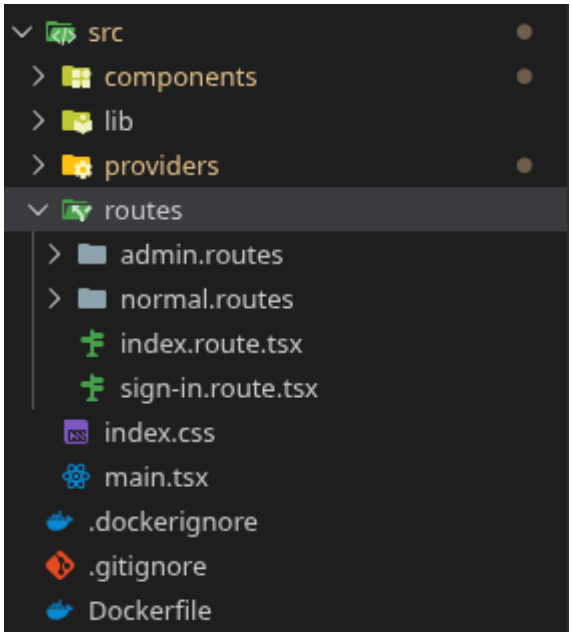


Figure 9: Project Structure - Client

5.1.2. Server

The application server contains the business logic and exposes them via REST API endpoints.

- **Controllers –** Controllers receive incoming HTTP requests and invoke services to service those requests.
- **DTOs –** Data-Transfer Objects represent contracts on the method body for HTTP requests and responses.
- **Lib –** Contains shared utilities.
- **Services –** Contains business logic which is modeled from the UML diagrams.
- **Program.cs** – The entry-point for the .NET Web API server. It contains configurations for different phases of the web server/host lifecycle.

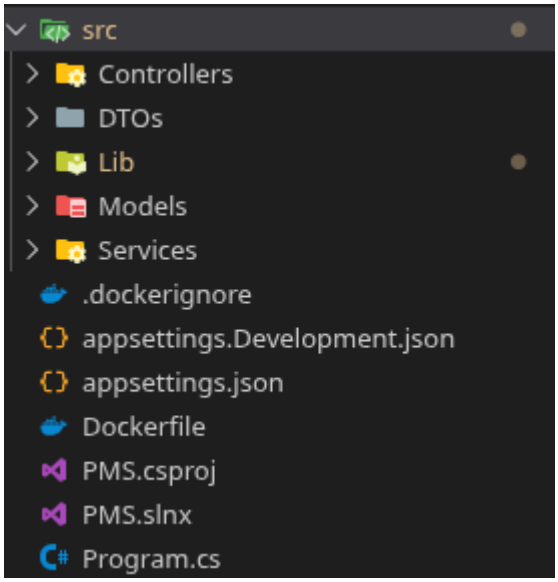


Figure 10: Project Structure - Server

5.2. Pertinent Application Features

To keep this section organized, the code has been developed in accordance with software engineering principles such as maintainability and consistency. Hence, only pertinent code snippets will be highlighted.

The items to be considered are listed below:

1. Authentication & Authorization
2. Edit Project Task
3. AI Feedback Analysis

5.2.1. Authentication & Authorization

Authentication and authorization are responsible for the application's security. Since the client and server are physically separate, they are managed primarily on the client, using access and refresh JSON web tokens to maintain integrity.

5.2.1.1. Client Side

To ensure maintainability, authentication and authorization logic is centralized to a custom provider. All actions that may change the authentication state of the application are defined as callbacks or other objects and are included in the data that the provider makes available to downstream components.

```
const authorizedAPI = useMemo(
  () =>
    ky.extend({
      prefixUrl: baseURL,
      credentials: "include",
      hooks: {
        beforeRequest: [
          (request) => {
            if (authState.token)
              request.headers.set("Authorization", `Bearer ${authState.token}`);
          },
        ],
        beforeError: [
          async (error) => {
            if (error.response?.status === 401) {
              try {
                const data = await performRefresh();

                const newAuth = { user: authState.user, isAuthenticated: true, ...data, };
                setAuthState(newAuth);
              } catch (e) {
                await signOut();
              }
            }
            return error;
          },
        ],
      },
      retry: { limit: 1 },
    }),
  [authState],
);
```

Figure 11: Auth API in auth.providers.tsx

The authorizedAPI ky instance, as shown in the figure above, is one such items provided. The ky library is a wrapper around the native fetch method for network request invocations, allowing to configurations to set when making a request, eliminating code duplication.

The useMemo React Hook *(see appendix)* prevents the authorizedAPI instance from re-initializing each time React re-renders the provider component, under the condition that authState variable does not change.

There are 3 important points to consider about the ky instance:

1) Before Request:

For making authenticated requests, including the token (access token) in the Authorization header with the `Bearer ` prefix is a standard approach. Therefore, as exemplified in the figure above, the token is inserted before every request invoked using authorizedAPI.

2) Before Error:

In session-based authentication, whenever the session expires, the user is forced to login again. Repeated login can severely reduce the usability of the system. Hence, whenever the token expires – the status code of the response is 401 (Unauthenticated), an access token refresh is attempted.

If the request succeeds, the authentication state is updated with the new token but if it fails, then the system logs the user out.

3) Refresh Token Request De-Duplication:

Multiple API requests are usually performed per route. If the token is invalid, then each of them will try to refresh the token, performing duplicate requests. The server will flag these multiple ongoing requests as a replay attack *(see appendix)* and invalidate the refresh token.

```ts
let refreshPromise: Promise<any> | null = null;
const performRefresh = async (): Promise<any> => {
  if (!refreshPromise) {
    refreshPromise = ky
      .post(`${baseURL}/api/token/refresh`, {
        credentials: "include",
      })
      .json();
  }

  try {
    const data = await refreshPromise;
    refreshPromise = null;
    return data;
  } catch (err) {
    refreshPromise = null;
    throw err;
  }
};
```

Figure 12: performRefresh in auth.providers.tsx

Hence, performRequest method maintains the state of refreshPromise across calls. The first call to performRequest method will invoke the request and set It to a non-null value. This will cause other calls to thus await the same token refresh response, de-duplicating requests to the refresh token endpoint.

Whenever the response is obtained or the request fails, the refreshPromise variable is reset. In the event of race conditions, multiple requests to the token refresh endpoint can be made, causing the token to be invalidated, and logging the user out. Since both the occurrence probability and the risk are low, they can safely be ignored.

5.2.1.2. Server Side

5.2.1.2.1. Token Generation

On the server, the tokens are generated with sub claim representing the subject identifier as defined by OpenID Connect (Sakimura et al., 2024), set to the userID and the role claim.

```
public TokenService(SymmetricSecurityKey key)
{
    creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
}
2 references
private string GenerateToken(long userID, string role, DateTime expirationDate)
{
    var claims = new List<Claim> {
        new Claim("sub", userID.ToString()),
        new Claim("role", role),
    };

    var token = new JwtSecurityToken(
        claims: claims,
        expires: expirationDate,
        signingCredentials: creds
    );

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Figure 13: Generate Token method in Token.Service.cs

To generate a token, the following must be taken into consideration:

1) Initialization

Before GenerateToken can be invoked, a secret key, obtained from the server's environment variables, uses the HMACSHA256 hashing algorithm to construct the signing credentials. This ensures that only the application server can generate and validate tokens (Microsoft, 2024) because the resulting hash from HMACSHA256 becomes unique to both the input data and the secret key provided.

2) Token Signing

After the token is created, the JwtSecurityTokenHandler class uses the signing credentials which contain the secret key, used as a symmetric key, to create its digital signature. The token with the digital signature is then serialized into a string representation and returned.

5.2.1.2.2. Token Transmission

The server distinguishes between a short-lived, access token and a long-lived, refresh token. Since the refresh token is used to request new access tokens whenever previous ones expire, it must remain secure.

```
var userAuth = await userService.Login(loginDTO.Email, loginDTO.Password);

var cookieOptions = new CookieOptions
{
    HttpOnly = true,
    Secure = !environment.IsDevelopment(),
    SameSite = SameSiteMode.Strict,
    Expires = DateTime.UtcNow.AddDays(14),
    Path = "/api/users/token"
};

Response.Cookies.Append("refreshToken", userAuth.RefreshToken.Payload, cookieOptions);
return Ok(new
{
    User = userAuth.User,
    Token = userAuth.AccessToken,
    TokenExpiry = userAuth.AccessToken.Expiry
});
```

Figure 14: Setting Refresh Token Cookie in User.Controller.cs

Therefore, it is stored as an HTTP-only cookie to prevent programmatic access on the client and has a path restriction such that the browser includes the cookie for the token refresh or logout endpoints only. The SameSite attribute set to strict, ensures that the cookie is only sent by the same origin that it is originally sent to, prevent CSRF attacks *(see appendix)*.

5.2.1.2.3. Custom Authorization Policy

In the .NET security framework, an authorization policy consists of one or more requirements, each managed by a dedicated handler (Microsoft, 2024).

While .NET provides a built-in RBAC policy, it lacks a dedicated Ownership policy that works with API route search parameters. Hence, a custom authorization requirement and its corresponding handler must be created.

```
public class OwnershipRequirement : IAuthorizationRequirement;

4 references
public class OwnershipHandler : AuthorizationHandler<OwnershipRequirement>
{
    2 references
    private readonly IHttpContextAccessor httpContextAccessor;
    1 reference
    private readonly ILogger<OwnershipHandler> logger;
    0 references
    public OwnershipHandler(IHttpContextAccessor httpContextAccessor, ILogger<OwnershipHandler> logger)
    {
        this.httpContextAccessor = httpContextAccessor;
        this.logger = logger;
    }
    0 references
    protected override Task HandleRequirementAsync(
        AuthorizationHandlerContext context,
        OwnershipRequirement requirement)
    {
        var httpContext = httpContextAccessor.HttpContext;

        var routeUserID = httpContext.GetRouteValue("userID")?.ToString();
        var tokenUserID = context.User.FindFirst("sub")?.Value;
        var role = context.User.FindFirst("role")?.Value;


        if ((!string.IsNullOrEmpty(routeUserID) && tokenUserID == routeUserID) || role == "admin")
        {
            context.Succeed(requirement);
        }

        return Task.CompletedTask;
    }
}
```

Figure 15: OwnershipRequirement And OwnershipHandler in Ownership.Lib.cs

The “OwnershipHandler” runs after the JWT has been validated and the user’s identity has been proven. It uses the “HTTPContextAccessor” object to extract the JWT and its corresponding claims. It ensures that the userID in the request URL matches the token subject, prevent users from invoking requests on behalf of other users, unless they are admin users

For example, a user with userID 2 cannot invoke HTTP request: GET /api/users/1/projects, if they are either a student or a supervisor. Administrators bypass this check entirely.

5.2.2. Edit Project Task

The “Edit Project Task” functionality represents one of the most complex operations within the system as it requires several components, both on the client and on the server, to interact together.

5.2.2.1. Client Side

On the client, in order to perform an edit operation on a given project task, its corresponding entry must be located on the task list (Fig 12) and the ellipsis button must be clicked. This opens a menu which includes the edit option. Clicking on the edit button opens a dialog populated by the task data from that entry. After the edit is performed, the save button persists the changes in the server database and causes the task list to refresh.

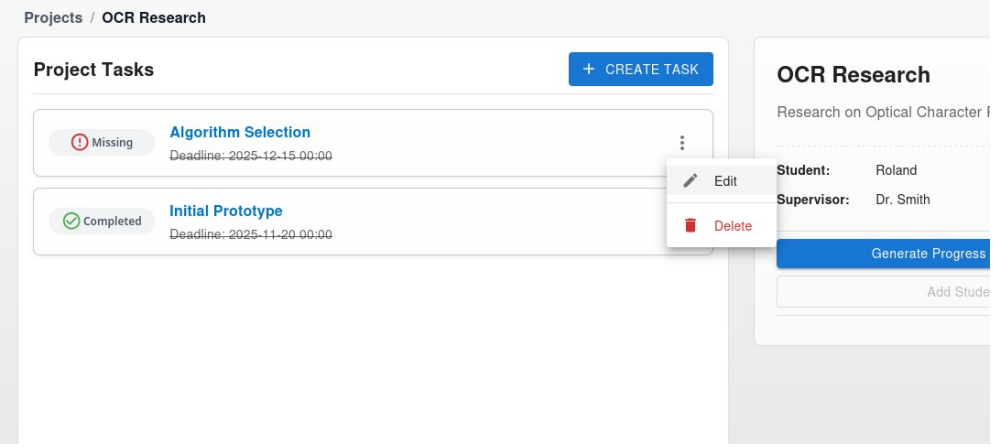


Figure 16: PMS - Project Task List

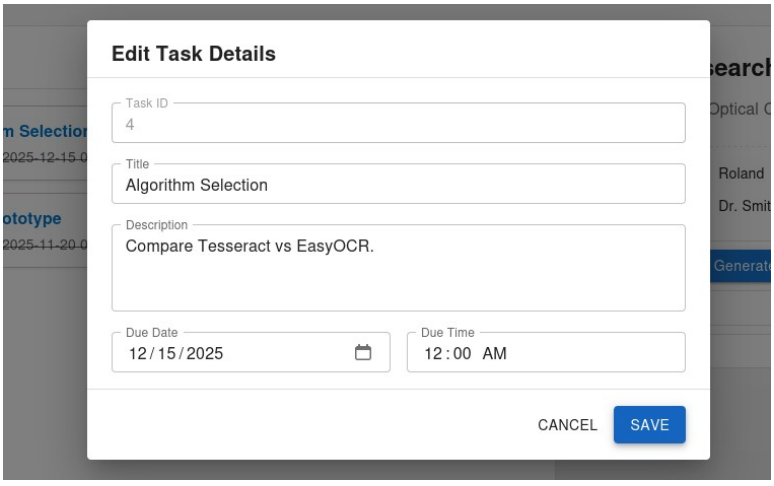


Figure 17: PMS - Project Task Modal

The following are some of the important aspects covered:

- Fetching Task Data
- Task List
- Task Modal
- Edit Task Functionality

5.2.2.1.1. Fetching Task Data

Before the user can perform actions on a task, they must first be fetched from the server.

Tanstack Query

The Tanstack Query library (Tanstack, 2025) provides utilities to synchronize the component's state with the server data.

The useQuery hook manages the internal state of the HTTP request. When the request transitions from one state to another (pending to successful response or error), React will automatically re-render the component to display the fetched data.

Pagination

Pagination is used to improve the performance of the system and reduce bandwidth usage for network requests. By including offset and limit in the search parameters of the request, only the specific subset of tasks that are displayed on the interface are fetched.

```
const { data: tasks, isLoading: tasksLoading } = useQuery({
  queryKey: [projectID, "tasks", taskListOffset, taskListLimit],
  queryFn: async (): Promise<{ items: Task[]; totalCount: number }> =>
    await authorizedAPI
      .get(`api/users/${user.userID}/projects/${projectID}/tasks`, {
        searchParams: {
          limit: taskListLimit,
          offset: taskListOffset,
        },
      })
      .json(),
  retry: 1,
  refetchInterval: 1000 * 60 * 5, // Refetch every 5 minute
});
```

Figure 18: Fetch Tasks Query in dashboard-tasks.route.tsx

5.2.2.1.2. Task List

The tasks data from the preceding section is then used to populate the list of tasks.

Component Composition

Similar to other components on the client side, the TaskList component is built on top of React components provided by the Material UI library (Material UI, 2024), and follows a component composition pattern.

In the component composition approach, the different parts of TaskList are identified and implemented instead as modular sub-components. This increases the modularity and maintainability of the components as well as provide a holistic perspective over their structure.

```
<TaskList sx={{ flexGrow: 3 }}>
  <TaskList.Header>
    {user.role == "supervisor" && (
      <TaskList.CreateTaskButton
      onClick={handleCreateTaskClick} />
    )}
  </TaskList.Header>
  <TaskList.Content
    isLoading={tasksLoading}
    projectID={projectID}
    tasks={tasks?.items ?? []}
    menuEnabled={user.role === "supervisor"}
    handleEditTaskClick={handleEditTaskClick}
    handleDeleteTaskClick={handleDeleteTaskClick}
  />
  <Pagination
    totalCount={tasks?.totalCount ?? 5}
    limit={taskListLimit}
    offset={taskListOffset}
    onPageChange={handlePageChange}
  />
</TaskList>
```

Figure 19: TaskList in dashboard-tasks.route.tsx

```
TaskList.Content = ({
  isLoading,
  projectID,
  tasks,
  menuEnabled,
  handleEditTaskClick,
  handleDeleteTaskClick,
}: {...}) => (
  <List disablePadding>
    {tasks.map((task) => (
      <TaskListEntry key={task.taskID}
                     status={task.status}>
        <TaskListEntry.Link ...  />
        {menuEnabled && (
          <TaskListEntry.MenuButton
            onEditButtonClick={() =>
              handleEditTaskClick({
                taskID: task.taskID,
                title: task.title,
                description: task.description,
                dueDate: task.dueDate,
              })
            }
            onDeleteButtonClick={() =>
              handleDeleteTaskClick({...}) }
          />
        )}
      </TaskListEntry>
    ))}
  </List>
);
```

Figure 5: TaskList.Content Composite Component

Populating TaskList.Content

The tasks data from the query is used to populate the list. The map method transforms the array of tasks into their corresponding React components, which are displayed in TaskList.Content composite component (the actual task list).

Each mapped component has a unique key for React to distinguish them from one another.

5.2.2.1.3. Task Modal

The modal as shown earlier, is designed to be maintainable and modular. This modularity is reflected in the way the modal can easily swap between different views for different purposes such as creating, editing or deleting tasks rather than creating separate components for each use case.

```tsx
<TaskModal open={taskModalState.open}>
  <TaskModal.Header mode={taskModalState.mode} />

  {modalViews[taskModalState.mode]}

  <TaskModal.Actions
    mode={taskModalState.mode}
    disabled={formDataIncomplete}
    handleCancelClick={handleCancelClick}
    handleCreateTask={handleCreateTask}
    handleEditTask={handleEditTask}
    handleDeleteTask={handleDeleteTask}
    handleAddStudent={handleAddStudent}
  />
</TaskModal>
```

Figure 20: TaskModal in dashboard-tasks.route.tsx

```tsx
const modalViews = {
  create: (
    <TaskModal.Fields>
      <TaskModal.TaskTitle
        title={taskModalData.title}
        handleTitleChange={handleTitleChange}
      />
      <TaskModal.TaskDescription
        description={taskModalData.description ?? ""}
        handleDescriptionChange={handleDescriptionChange}
      />
      <TaskModal.DueDate
        dueDate={taskModalData.dueDate}
        handleDueDateChange={handleDueDateChange}
      />
    </TaskModal.Fields>
  ),
  edit: (
    <TaskModal.Fields>
      <TaskModal.TaskID taskID={taskModalData.taskID} />
      <TaskModal.TaskTitle
        title={taskModalData.title}
        handleTitleChange={handleTitleChange}
      />
      <TaskModal.TaskDescription
        description={taskModalData.description ?? ""}
        handleDescriptionChange={handleDescriptionChange}
      />
      <TaskModal.DueDate
        dueDate={taskModalData.dueDate}
        handleDueDateChange={handleDueDateChange}
      />
    </TaskModal.Fields>
  ),
  // add-student modal view here
  delete: <TaskModal.DeleteWarning />,
};
```

Figure 21: Task Modal Views

5.2.2.1.4. Edit Task Functionality

The modal state and handler callbacks are located in the same file for accessibility.

When the edit menu button from task list entry's menu is clicked, the handleEditTaskClick callback is invoked. This populates the task entry data in the modal and changes its mode and visibility to show the edit view.

```
const [taskModalState, setTaskModalState] = useState<ModalState>({
  mode: "create",
  open: false,
});
const [taskModalData, setTaskModalData] = useState<TaskFormData>({
  taskID: 0,
  title: "",
  description: "",
  dueDate: "",
});
```

Figure 22: Task Modal - State and Data

```
const handleEditTaskClick = (taskData: TaskFormData) => {
  setTaskModalData(taskData);
  setTaskModalState((t) => ({ ...t, mode: "edit", open: true }));
};
```

Figure 23: Task List Entry Menu – Edit Button Handler

When a user edits the modal fields, the following callbacks are invoked to synchronize their internal data with the external modal data.

```
const handleTitleChange = useCallback((title: string) => {
  setTaskModalData((t) => ({ ...t, title: title }));
}, []);
const handleDescriptionChange = useCallback((description: string) => {
  setTaskModalData((t) => ({ ...t, description: description }));
}, []);
const handleDueDateChange = useCallback((dueDate: string) => {
  setTaskModalData((t) => ({ ...t, dueDate: dueDate }));
}, []);
```

Figure 24: Task Modal - Handlers

Once the save button – located inside TaskModal.Actions composite component - is clicked, the handleEditTask callback is called.

Tanstack Query library provides another utility – mutations. They are essentially used to invalidate and re-execute queries whenever a state changing request such as a POST or PUT request is performed.

```
const handleEditTask = () => {
  mutation.mutate(
    {
      method: "put",
      url: `api/users/${user.userID}/projects/${projectID}/tasks/${taskModalData.taskID}`,
      data: taskModalData,
      invalidateQueryKeys: [[projectID, "tasks", taskListOffset, taskListLimit]],
    },
    {
      onSuccess: () => {
        setTaskModalState((t) => ({ ...t, open: false }));
      },
    },
  );
};
```

Figure 25: Task Modal - Handle Task Edit

```
const queryClient = useQueryClient();
const mutation = useMutation({
  mutationFn: async ({
    method,
    url,
    data,
  }: {
    method: string;
    url: string;
    data: any;
    invalidateQueryKeys: any[][];
  }) => await authorizedAPI(url, { method: method, json: data }),
  onSuccess: (_data, variables) =>
    variables.invalidateQueryKeys.forEach((key) =>
      queryClient.invalidateQueries({
        queryKey: key,
      })
    ),
});
```

Figure 26: Task Edit Mutation in dashboard-tasks.route.tsx

The mutation above has been configured to invalidate specific queries by their associated keys, upon a successful request. For example, the key ‘[projectID, “tasks”, ...]’ is associated with the query shown earlier in the TaskList section.

The UI state for the task list will refresh, making the update to edited task visible.

5.2.2.2. Server Side

The web server's primary purpose is to listen for HTTP requests and upon receiving a request, it dispatches a thread to service it. It looks for the controller it is associated with, instantiating the controller and its dependencies before calling the appropriate method.

The following requests will be considered:

- Fetching the project tasks
- Editing the project task

5.2.2.2.1. Fetching the Project Tasks

The controllers are solely responsible for handling HTTP requests and invoking the appropriate services. In the figure below, the subset of the tasks limited by the pagination URL search query parameters alongside the total count of tasks are returned to the client.

```
[Route("api/users/{userID}/projects/{projectID}/tasks")]
[HttpGet]
[Authorize(Policy = "OwnershipRBAC")]
0 references
public async Task<IActionResult> GetProjectTasks(
    [FromRoute] long userID,
    [FromRoute] long projectID,
    [FromQuery] int limit = 5,
    [FromQuery] int offset = 0
)
{
    try
    {
        var (tasks, count) = await projectTaskService
                .GetProjectTasksWithCount(userID, projectID, limit, offset);

        return Ok(new
        {
            Items = tasks,
            TotalCount = count
        });
    }
    catch (Exception e)
    {
        return NotFound(e.Message);
    }
}
```

Figure 27: Fetch Tasks in Tasks.Controller.cs

```
public async Task<(IEnumerable<GetProjectTaskDTO> tasks, long count)>
      GetProjectTasksWithCount(long userID, long projectID, long limit = 5, long offset = 0)
  {
      var tasksQuery = dbContext.Tasks.Where(
          t => t.ProjectID == projectID &&
                  (t.Project.StudentID == userID || t.Project.SupervisorID == userID)
          );

      var count = await tasksQuery.LongCountAsync();

      var tasks = await tasksQuery
          .OrderByDescending(t => t.DueDate)
          .Skip((int)offset)
          .Take((int)limit)
          .ToListAsync();

      bool hasChanges = false;
      foreach (var task in tasks)
      {
          if (task.DueDate < DateTime.UtcNow && task.Status == "pending")
              task.Status = "missing";
          hasChanges = true;
      }

      if (hasChanges)
          await dbContext.SaveChangesAsync();

      return (tasks.Select(t => new GetProjectTaskDTO
      {
          TaskID = t.ProjectTaskID,
          Title = t.Title,
          Description = t.Description,
          AssignedDate = t.AssignedDate,
          DueDate = t.DueDate,
          Status = t.Status,
      }), count);
  }
```

Figure 28: Fetch Tasks method in ProjectTask.Service.cs

There are 4 important considerations about GetProjectsWithCount method:

Entity Framework Core ORM

Through the use of EF Core ORM, queries can be represented as declarative statements. The advantage of such abstraction over compiling and executing raw SQL is that it handles security concerns such as preventing SQL injection and automates the mapping between C# objects with database tables, reducing overall development time.

EF Core also features a Change Tracking System whereby changes made to entities in the application can be synchronized with the database using SaveChangesAsync method.

Fine-Grained Access Control

The system enforces access control on the data through checks that verify the user's identity. In the example above, the base query for fetching the tasks and total count first ensure that the tasks to be accessed are strictly shared with the authenticated student or supervisor.

Then, the separate queries to fetch the required subset of tasks based on the pagination arguments and the total count are executed.

Lazy Updates

In order to manage temporal data, a lazy update strategy was chosen instead of having a background cron service periodically updating database records due to reasons mentioned in the Analysis section.

Data Transfer Objects (DTOs)

A DTO defines the contract for an HTTP request's body. DTOs have 2 major roles in the system:

1. They serve as validations for the body of an incoming HTTP's requests.
2. They act as a bridge between the data on the client and on the server.

```
public class GetProjectTaskDTO
{
    2 references
    public long TaskID { get; set; }
    2 references
    public required string Title { get; set; }
    2 references
    public string? Description { get; set; }
    2 references
    public DateTime AssignedDate { get; set; }
    2 references
    public DateTime DueDate { get; set; }
    2 references
    public required string Status { get; set; }
    1 reference
    public bool? IsLocked { get; set; }
    1 reference
    public UserLookupDTO? AssignedBy { get; set; }
    1 reference
    public long? StagedDeliverableID { get; set; }
    1 reference
    public long? SubmittedDeliverableID { get; set; }
}
```

Figure 29: GetProjectTasksDTO

```
export type Task = {
  taskID: number;
  title: string;
  description?: string;
  status: "pending" | "completed" | "missing";
  isLocked?: boolean;
  assignedDate: string;
  dueDate: string;
  assignedBy?: User;
};
```

Figure 30: Task type in types.ts

5.2.2.2.2. Editing the Project Task

Similar to when project tasks are fetched, the controller responsible for routing project task related requests will invoke the corresponding method in the service class.

```
public async Task EditProjectTask(long userID, long projectID, long taskID, EditProjectTaskDTO dto)
   {
       var task = await dbContext.Tasks.Where(t =>
           t.ProjectTaskID == taskID && t.ProjectID == projectID && t.Project.SupervisorID == userID)
           .Include(t => t.Project)
               .ThenInclude(p => p.Student)
           .Include(t => t.Project)
               .ThenInclude(p => p.Supervisor)
           .FirstOrDefaultAsync()
           ?? throw new KeyNotFoundException("Unauthorized Access or Task Not Found.");

       bool dueDateUpdated = false;
       MimeMessage? mail = null;

       using (var transaction = await dbContext.Database.BeginTransactionAsync())
       {
           try
           {
               task.Title = dto.Title ?? task.Title;
               task.Description = dto.Description ?? task.Description;
               if (dto.DueDate != task.DueDate)
               {
                   task.DueDate = (DateTime)dto.DueDate;

                   if (task.DueDate < DateTime.UtcNow && task.Status.Equals("pending"))
                       task.Status = "missing";

                   dueDateUpdated = true;
               }
               task.IsLocked = dto.IsLocked ?? task.IsLocked;

               await dbContext.SaveChangesAsync();

               if (task.Project.StudentID != null)
               {
                   await notificationService.CreateTaskNotification(task, NotificationType.TASK_UPDATED);

                   if (dueDateUpdated)
                   {
                       await reminderService.UpdateTaskReminder(task);
                       mail = mailService.CreateTaskMail(task, MailType.TASK_UPDATED);
                   }
               }

               await transaction.CommitAsync();
           }
           catch (Exception)
           {
               await transaction.RollbackAsync();
               throw;
           }
       }

       if (mail != null)
           await mailService.SendMail(mail);
   }
```

Figure 31: Edit Project Task method in ProjectTasks.Services.cs

After the required project task is fetched from the taskID, it is then updated with the new values. To keep the student informed, a notification is created and the existing reminder associated with the task is also updated. An email is then sent to the student after the operation is successful.

Since this operation involves multiple services, the system must therefore rely on database transactions to ensure that the update is ACID compliant, mainly to preserve the consistency of the database and prevent sending emails if the operation fails.

Sending the Mail

The mail is represented by the MimeMessage class which encapsulates the content, sender and recipient of the mail. The mail is then sent using SMTP *(see appendix)*

Before it can be sent, a connection is established between the mail client and the mail server and a mail account with its corresponding credentials is used for authentication. The mail is then sent and then the connection is closed, regardless of whether the operation has been successful or not.

```
6 references
public async Task SendMail(MimeMessage message)
{
    using var client = new SmtpClient();
    try
    {
        await client.ConnectAsync("smtp.gmail.com", 587, SecureSocketOptions.StartTls);
        await client.AuthenticateAsync(mailAccount, mailPassword);
        await client.SendAsync(message);
    }
    catch (Exception e)
    {
        logger.LogError(e, "SMTP Error");
        throw;
    }
    finally
    {
        await client.DisconnectAsync(true);
    }
}
```

Figure 32: Send Mail method in Mail.Service.cs

5.3. AI Feedback Analysis

The AI Feedback Analysis component compares the newly staged deliverable with the previous deliverable submission to update the feedback criteria, tracking the relevant changes for each of them.

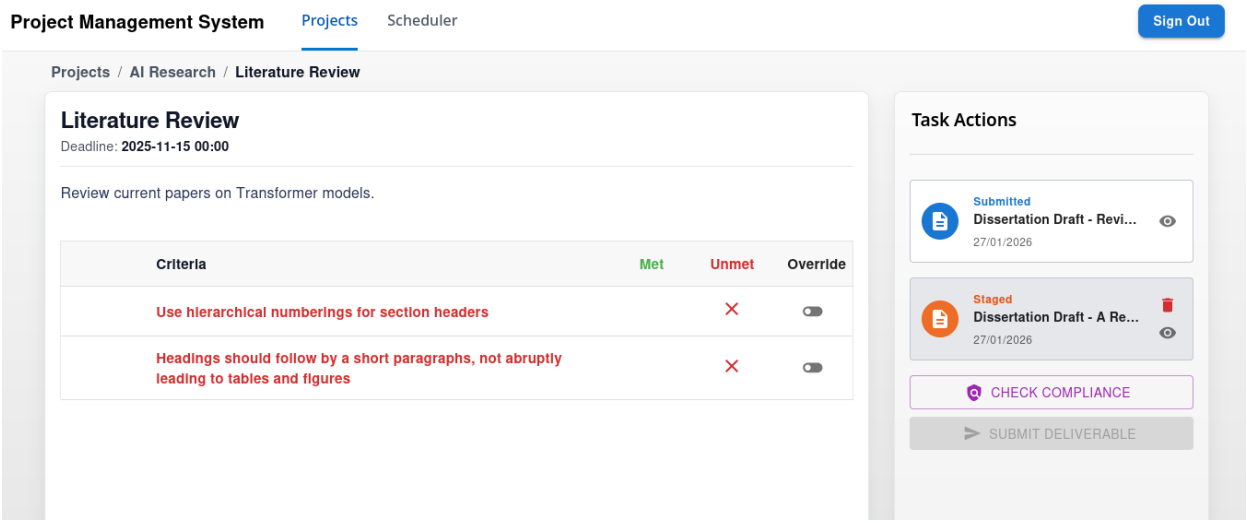
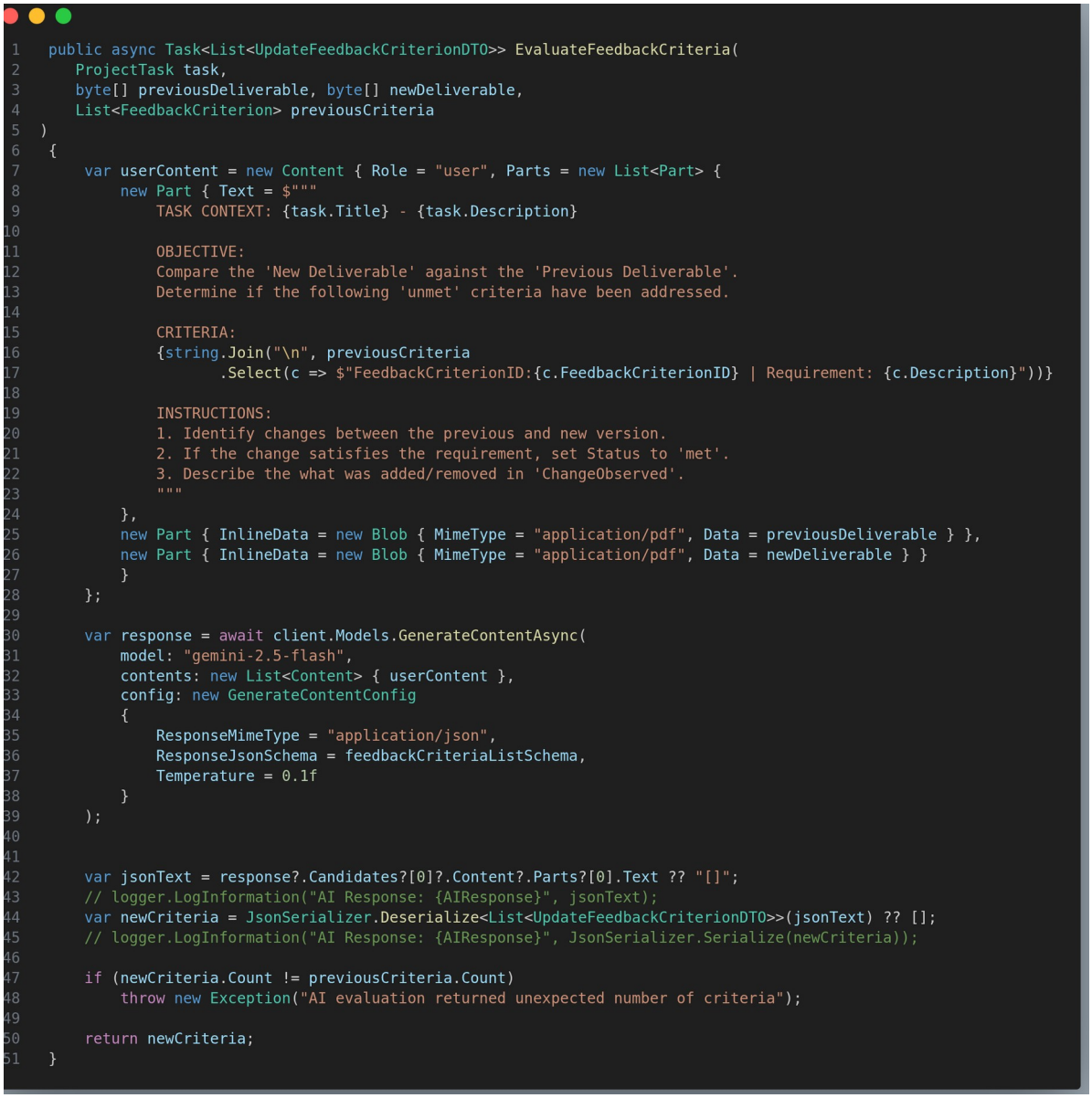


Figure 33: PMS - Task w/ Feedback Interface

```
public async Task AIFeedbackComplianceCheck(
    long userID, long projectID, long taskID
)
{
    var task = await dbContext.Tasks.Where(
                t => t.ProjectTaskID == taskID
                    && t.ProjectID == projectID
                        && (t.Project.StudentID == userID || t.Project.SupervisorID == userID)
                )
                .Include(t => t.SubmittedDeliverable)
                .Include(t => t.StagedDeliverable)
                .FirstOrDefaultAsync()
                ?? throw new UnauthorizedAccessException("Task Not Found");

    if (task.StagedDeliverable == null || task.SubmittedDeliverable == null)
        throw new Exception("Task must have both a Staged and Submitted Deliverable");

    var prevFeedbackCriteria = await dbContext.FeedbackCriterias
                .Where(c => c.DeliverableID == task.SubmittedDeliverableID)
                .ToListAsync();
    if (prevFeedbackCriteria.Count == 0)
        throw new Exception("No Feedback Found for Submitted Deliverable");

    var newFeedbackCriteria = await AIService.EvaluateFeedbackCriteria(
                task,
                previousDeliverable: task.SubmittedDeliverable.File,
                newDeliverable: task.StagedDeliverable.File,
                previousCriteria: prevFeedbackCriteria.Where(c => c.Status == "unmet").ToList()
        );

    await UpdateFeedbackCriteria(task.SubmittedDeliverable, newFeedbackCriteria);
}
```

Figure 34: AIFeedbackComplianceCheck method in Feedback.Service.cs

Upon clicking on the “Check Compliance” button, the system performs the automated feedback analysis. It firstly retrieve the relevant task from the database, performing a join on its staged and submitted deliverables. The feedback criteria is then extracted separately provided that the task currently has a submitted deliverable.



```
public async Task<List<UpdateFeedbackCriterionDTO>> EvaluateFeedbackCriteria(
   ProjectTask task,
   byte[] previousDeliverable, byte[] newDeliverable,
   List<FeedbackCriterion> previousCriteria
)
 {
     var userContent = new Content { Role = "user", Parts = new List<Part> {
         new Part { Text = $"""
             TASK CONTEXT: {task.Title} - {task.Description}

             OBJECTIVE:
             Compare the 'New Deliverable' against the 'Previous Deliverable'.
             Determine if the following 'unmet' criteria have been addressed.

             CRITERIA:
             {string.Join("\n", previousCriteria
                    .Select(c => $"FeedbackCriterionID:{c.FeedbackCriterionID} | Requirement: {c.Description}"))}

             INSTRUCTIONS:
             1. Identify changes between the previous and new version.
             2. If the change satisfies the requirement, set Status to 'met'.
             3. Describe the what was added/removed in 'ChangeObserved'.
             """
         },
         new Part { InlineData = new Blob { MimeType = "application/pdf", Data = previousDeliverable } },
         new Part { InlineData = new Blob { MimeType = "application/pdf", Data = newDeliverable } }
         }
     };

     var response = await client.Models.GenerateContentAsync(
         model: "gemini-2.5-flash",
         contents: new List<Content> { userContent },
         config: new GenerateContentConfig
         {
             ResponseMimeType = "application/json",
             ResponseJsonSchema = feedbackCriteriaListSchema,
             Temperature = 0.1f
         }
     );


     var jsonText = response?.Candidates?[0]?.Content?.Parts?[0].Text ?? "[]";
     // logger.LogInformation("AI Response: {AIResponse}", jsonText);
     var newCriteria = JsonSerializer.Deserialize<List<UpdateFeedbackCriterionDTO>>(jsonText) ?? [];
     // logger.LogInformation("AI Response: {AIResponse}", JsonSerializer.Serialize(newCriteria));

     if (newCriteria.Count != previousCriteria.Count)
         throw new Exception("AI evaluation returned unexpected number of criteria");

     return newCriteria;
 }
```

Figure 35: EvaluateFeedbackCriteria method in AI.Service.cs

The data retrieved from the database is then passed as parameters to the “EvaluateFeedbackCriteria” method. This method then performs the comparative analysis on the two documents alongside the feedback provided, using the Gemini Multi-modal LLM.

This process can be divided into the following phases:

Payload Construction

The payload to be sent to Gemini LLM will include the prompt with the task objectives, the unmet feedback criteria and the two deliverable files. The PDF document files are attached as raw binary data as “InlineData” blobs.

There are a few advantages of sending them as raw binary data as opposed to text format:

- This preserves the model’s ability to consider figures and document layout as opposed to extracting the document text which is a lossy process.
- Gemini can leverage its internal document processing tools and multi-modal nature to reduce the input tokens that are computed by the model.

Model Invocation with Schema Constraints

The model is invoked with the payload and some configurations.

The most important configuration is enforcing a JSON schema constraint on the model’s output. By specifying a schema, the model zeroes out probabilities for other tokens that do not align with the given schema during processing, ensuring the validity of the final response.

```
protected static readonly Schema feedbackCriteriaListSchema = new Schema {
    Type = Type.ARRAY, Items = new Schema {
        Type = Type.OBJECT,
        Properties = new Dictionary<string, Schema> {
            {"FeedbackCriterionID", new Schema{Type=Type.NUMBER}},
            {"Status", new Schema{Type=Type.STRING, Enum=new List<string>{"met", "unmet"}}},
            {"ChangeObserved", new Schema
                { Type=Type.STRING, Description="What specific change was made between documents?" }},
        },
        Required = new List<string>(["FeedbackCriterionID", "Status", "ChangeObserved"])
    }
};
```

Figure 36: AI Response Schema in AI.Service.cs

The second configuration is the temperature of the AI model, which determines the variety of the model’s response. Restricting the temperature to a low value causes the model to only prioritize the most likely token probabilities during processing.

This reduces hallucinations and ensures that the same response is likely to be obtained when the same payload is sent multiple times.

Once the AI JSON response is obtained, it is then parsed and extracted in an “UpdateFeedbackCriteriaDTO” object. This Data Transfer Object is then returned to the “AIFeedbackComplianceCheck” method as the updated criteria data.

Bass, L., Clements, P. & Kazman, R. (2021). *Software Architecture in Practice* (4th ed.). Addison-Wesley Professional.

https://learn.microsoft.com/en-us/nuget/what-is-nuget

https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-9.0

https://laravel.com/docs/12.x/csrf

Buljić, I., Kadusic, E., Cvijanovic, T., Hadzajlic, N. and Zivic, N. (2025) Comparative Performance Analysis of Leading Backend Frameworks for Developers. In: (Title of Conference/Proceedings/Book - *Inferred*) **2025 International Scientific Conference on Information Technology and Data Related Fields (INFOTEH)**. (Date/Month of Conference - *Inferred*) Tuzla, Bosnia and Herzegovina: INFOTEH, pp. 1–5. doi: 10.1109/INFOTEH64129.2025.10959250.

Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed.). Apress. (For Git storage and object model).

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). *Database Systems: The Complete Book* (2nd ed.). Pearson Prentice Hall. (For database storage and data structure management).

https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/

https://www.postgresql.org/docs/current/storage-toast.html

https://assets.bytebytego.com/diagrams/0333-what-s-the-difference-between-session-based-authentication-and-jwts.png

Mehta, M. R., Lee, S., & Shah, J. R. (2006). Service-Oriented Architecture: Concepts and Implementation. *Proceedings of the Information Systems Educators Conference (ISECON) 2006*, 23(35).

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. Sebastopol, CA: O'Reilly Media.

Gupta, M., Sharma, S., Rathi, M., & Singh, A. (2025). Secure API Gateway with Rate Limiting and JWT Authentication. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 13(4), pp. 3559-3562.

Belagatti, P. (2025) *How to Choose the Right SQL Database*. SingleStore. Available at: https://www.singlestore.com/blog/how-to-choose-the-right-sql-database/ (Accessed: 19 November 2025).

Sakimura, N., Bradley, J., Jones, M., de Medeiros, B. and Mortimore, C. (2014) *OpenID Connect Core 1.0 incorporating errata set 1*. Available at: https://openid.net/specs/openid-connect-core-1_0.html (Accessed: 24 January 2026).

Microsoft (2024) *HMACSHA256 Class (System.Security.Cryptography)*. Available at: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.hmacsha256 (Accessed: 24 January 2026)

Microsoft (2024) *Policy-based authorization in ASP.NET Core*. Available at: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/policies (Accessed: 25 January 2026).

tanstack.com. (2025.). *Overview | TanStack Query Docs*. [online] Available at: https://tanstack.com/query/latest/docs/framework/react/overview.

Material UI (2024). *Overview - Material UI*. [online] mui.com. Available at: https://mui.com/material-ui/getting-started/.

Appendix

A **domain model** is a conceptual model of the domain that incorporates both behavior and data. (Fowler, Martin: Analysis Patterns, Reusable object models, Addison-Wesley Longman, 1997. ISBN .)