INTRODUCTION

In the context of the rapid digital transformation taking place in higher education institutions, the management of graduation theses is still primarily conducted manually. This leads to many drawbacks such as inefficiencies in processes, difficulties in tracking progress, lack of transparency in supervision and evaluation, and fragmented data. Coordination among stakeholders-including students, supervising lecturers, reviewers, department heads, and faculty heads-often faces obstacles due to the absence of a centralized, interactive, and automated system. Therefore, developing a web-based graduation thesis management system is necessary to improve efficiency, increase transparency, support real-time collaboration among parties, and simplify processes such as topic registration, submission of outlines, evaluation, and defense organization.

This system aims to build a secure, user-friendly, and scalable web application integrated with role-based access control to support all stakeholders, including Faculty Heads, Department Heads, Supervising Lecturers, Reviewers, and Students. The goal is to streamline the entire thesis lifecycle-from topic registration, lecturer assignment, outline management, to evaluation, grading, and defense committee organization. The system is designed to meet evolving academic needs, ensuring flexibility and efficiency in management.

The system focuses on lecturers and students at the Faculty of Information Technology, University of Transport and Communications, aiming to automate administrative processes such as topic registration, assignment of supervising lecturers and reviewers, outline management, evaluation, grading, and defense organization. The project only concentrates on managing administrative workflows and does not assess the academic content of the theses, ensuring clarity and focus on management objectives.

CHAPTER 1 OVERVIEW OF PROJECT

1.1 Introduction of the project

Currently, the faculty and departments at the university do not have a comprehensive and unified system for managing graduation projects. The management process is mostly manual, relying on paperwork and direct communication, which leads to difficulties in tracking, assigning, and evaluating projects. This manual approach is time-consuming, prone to errors, and lacks transparency. Therefore, the development of a project management system is essential to improve efficiency and facilitate lecturers, students, and administrative staff throughout the entire process—from registering research topics, assigning supervisors, managing proposals, to evaluating and grading projects.

This project aims to develop a software system that automates and standardizes the graduation project management process, including:

- Defining detailed roles and permissions for Department Heads, Program Chairs, Supervisors, Reviewers, and Students.
- Allowing lecturers to register and manage research directions.
- Enabling Department Heads to assign supervisors to students based on preferences and workload limits.
- Allowing students to create and edit project proposals following provided templates, while supervisors can review and request revisions.
- Managing proposals at the program and department levels, including locking editing rights once proposals are approved.
- Supporting communication and collaboration between supervisors and students.
- Allowing Department Heads to assign reviewers suitable for the research direction.
- Enabling supervisors and reviewers to comment and grade students through digital forms.
- Creating defense committees consisting of lecturers from multiple programs to ensure diversity in research directions.
- Managing grades, revision requests, and generating reports efficiently.

The system will enhance management efficiency, reduce administrative workload, increase transparency, and foster better collaboration among all parties involved in the graduation project process.

1.2 Survey of current solution

The existing thesis management system presents several challenges:

- Manual topic registration causing time inefficiencies.
- Difficulties in tracking student progress in real-time.
- Suboptimal assignment of thesis supervisors.
- Non-standardized processes for thesis reviewing and grading.
- Inefficient document storage and management.

1.3 Overview of used tools of technologies

1.3.1 NextJS

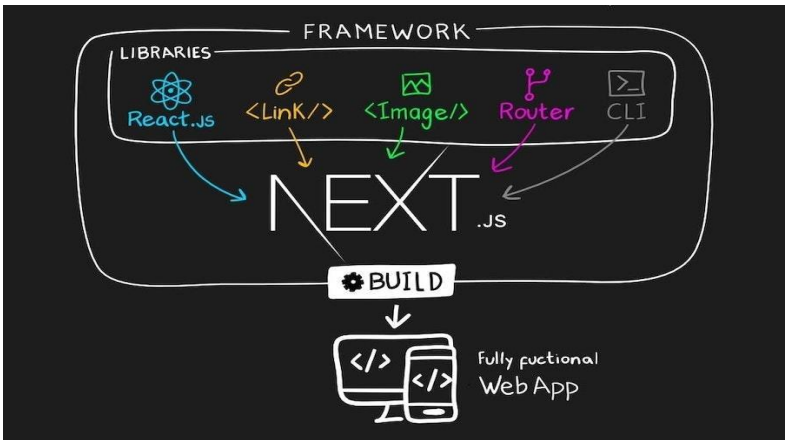

Figure 1.1. NextJS: Shaping the Future of Web Development [8]

NextJS is a robust and versatile React-based framework that excels at building highly performant, SEO-friendly, and scalable web applications. It is particularly well-suited for complex projects like a thesis management system, where responsiveness, reliability, maintainability, flexibility, and user experience are all paramount.

NextJS supports both SSR and SSG out of the box, providing significant advantages in terms of performance and user experience. SSR allows pages to be rendered on the server before being sent to the client, which results in faster initial load times and improved SEO — essential for features such as browsing thesis topics, viewing defense schedules, or accessing student profiles where content freshness and accessibility are critical. Meanwhile, SSG pre-builds static pages at build time, reducing server overhead and improving scalability, especially for pages that do not change frequently. This dual capability ensures the system can dynamically balance between real-time data needs and efficient delivery.

NextJS integrates seamlessly with powerful UI libraries such as MUI (Material-UI) and Tailwind CSS, enabling the creation of polished, intuitive, and accessible interfaces. These technologies support responsive design principles, ensuring the thesis management platform looks and functions flawlessly across a wide range of devices — from desktops and laptops to tablets and smartphones. This cross-device compatibility is crucial for catering to diverse users including students, faculty members, and administrative staff, enhancing overall user satisfaction and engagement.

The NextJS ecosystem is enriched by complementary tools that streamline development and enhance application robustness. Zustand offers an efficient and minimalistic state management solution, reducing boilerplate code and improving developer productivity. React Query simplifies asynchronous data fetching and caching, providing a seamless and reactive user experience even when dealing with complex data workflows like thesis submissions and review processes. Moreover, deploying the application via Vercel, the creators of NextJS, offers optimized hosting with features like automatic scaling, global CDN, and zero-configuration deployment, allowing rapid iterations and quick releases aligned with Agile Scrum methodologies.

NextJS benefits from a large and active developer community along with detailed official documentation. This wealth of resources mitigates technical risks during development by providing best practices, troubleshooting guides, and continuous updates. Consequently, the thesis management system development team can confidently deliver a high-quality product within constrained timelines, ensuring stability, maintainability, and future scalability.

In summary, NextJS not only provides the foundational technologies necessary for building a modern, efficient thesis management system but also fosters a productive development environment that supports quick adaptation, robust performance, and excellent user experience across all facets of the application.

1.3.2 NestJS

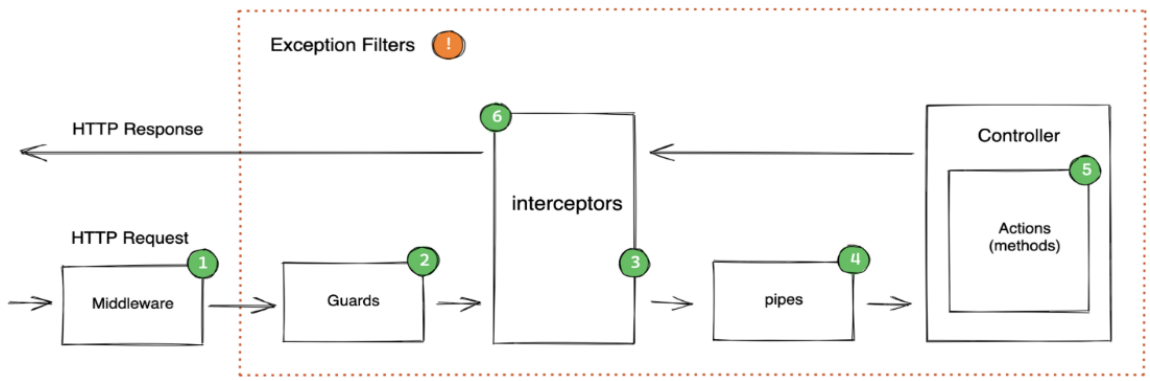

Figure 1.2. NestJS request lifecycle [9]

NestJS is a cutting-edge Node.js framework built with TypeScript, engineered to facilitate the development of highly efficient, well-structured, and scalable backend applications. Its robust design and advanced features make it an excellent choice for managing the complex and dynamic requirements of a thesis management system.

NestJS adopts a modular architecture that breaks down application logic into discrete, cohesive modules comprising controllers, services, and providers. This modularity allows developers to organize code around specific features — such as thesis topic registration, advisor-student assignments, and outline approvals — making the codebase more maintainable and scalable. Modules can be independently developed, tested, and reused across the system, greatly enhancing developer productivity and ensuring a clean separation of concerns.

Security is paramount in academic management systems, which handle sensitive data like student profiles, grades, and defense schedules. NestJS comes equipped with powerful security mechanisms including middleware, guards, and interceptors. It supports role-based access control (RBAC) to enforce fine-grained permissions, and seamless integration with JSON Web Token (JWT) authentication to ensure secure user sessions and API access. These features collectively protect the integrity and confidentiality of critical academic information, complying with best security practices.

NestJS integrates effortlessly with Prisma, a modern and performant Object-Relational Mapping (ORM) tool. Prisma simplifies database operations by providing a type-safe and declarative query API, reducing boilerplate code and minimizing runtime errors. This integration accelerates the development of reliable RESTful APIs for core system functionalities, including user management, thesis progress tracking, and dynamic reporting. It also improves maintainability by aligning database schema changes with application logic through migrations and schema validations.

Beyond core backend functionalities, NestJS streamlines development workflows by offering built-in support for critical features such as authentication, centralized logging, exception handling, and validation. Its extensible architecture empowers teams to integrate advanced capabilities in the future, such as real-time notifications for defense schedules, AI-powered recommendation systems for advisor assignments, or automated plagiarism detection modules. This flexibility ensures the thesis management system can evolve seamlessly alongside academic needs and technological advancements.

NestJS boasts an active community and a rich ecosystem of plugins, tools, and tutorials. This broad support network provides valuable resources for troubleshooting, optimization, and continuous learning, reducing development risks and enhancing project delivery timelines.

In summary, NestJS offers a powerful, secure, and developer-friendly backend framework perfectly aligned with the demands of a sophisticated thesis management system. Its modular structure, security-first design, and smooth database integration create a solid foundation for building maintainable and scalable academic applications, while its extensibility ensures long-term adaptability.

1.3.3 TypeScript and VS Code

TypeScript, a powerful superset of JavaScript, plays a crucial role in enhancing code reliability, maintainability, and scalability across both the frontend (NextJS) and backend (NestJS) of the thesis management system. By using TypeScript consistently throughout the entire stack, the development team achieves a high level of code uniformity and consistency, which is essential for complex projects that require seamless integration between different layers of the application. This uniformity not only reduces the likelihood of bugs but also makes the codebase easier to understand, maintain, and extend over time.

Complementing TypeScript is Visual Studio Code, a lightweight yet highly extensible and easy-to-integrate code editor that offers a streamlined user experience without compromising on powerful features. Its extensibility allows developers to tailor the environment to their workflow, integrating seamlessly with TypeScript and other development tools. Features such as IntelliSense provide intelligent code completions, real-time error checking, and quick navigation, significantly accelerating development. Visual Studio Code's built-in debugging capabilities also make identifying and fixing issues more efficient, which is particularly valuable during Agile sprints.

TypeScript's static typing system serves as a safety net by catching potential errors early, ensuring smoother interaction between frontend and backend components and minimizing debugging efforts. Strong typing encourages clearer documentation and self-explanatory code, easing onboarding and future maintenance. The synergy between TypeScript and Visual Studio Code results in a highly productive development environment, fostering better collaboration and reducing integration friction. This combination lowers development costs by minimizing bugs and technical debt while supporting scalable, maintainable growth, ultimately leading to a robust and reliable thesis management system.

1.3.4 PostgreSQL and Prisma

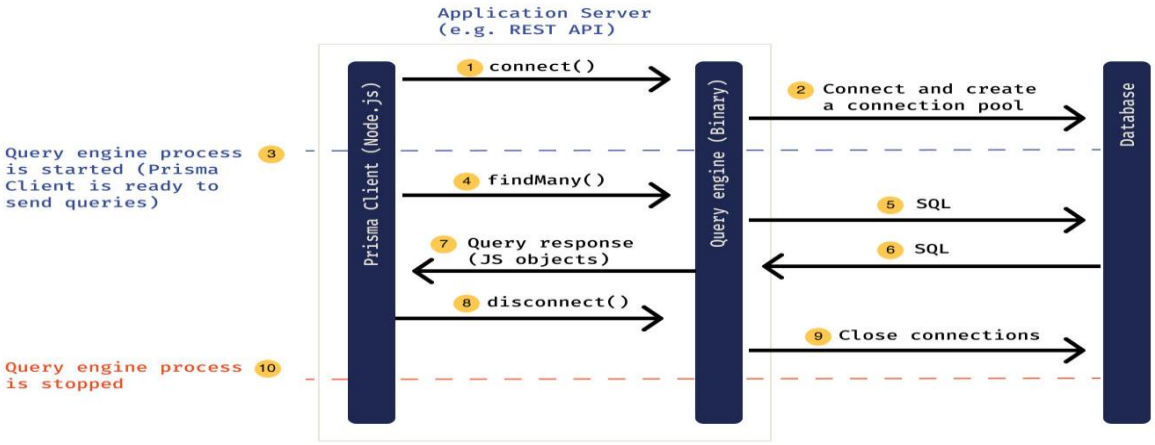

Figure 1.3. The Prisma data layer architecture? [10]

PostgreSQL and **Prisma** together create a robust, reliable, and scalable data layer for the thesis management system. This combination ensures high performance and flexible data handling, supporting long-term development needs effectively.

- **PostgreSQL**: An open-source relational database, PostgreSQL excels in handling complex queries and large datasets, making it ideal for storing information about users, thesis topics, outlines, grades, and defense schedules. Its support for foreign keys and constraints ensures data integrity, accommodating growth as user numbers increase.
- **Prisma**: A modern ORM, Prisma provides an intuitive, type-safe API that simplifies database operations like creating, updating, or querying data. It reduces boilerplate code and errors, while ensuring synchronization between TypeScript code and database schemas.
- **Scalability**: This combination allows the system to handle increasing data volumes, such as when scaling to university-wide use, and supports easy database migrations for future updates.
- **Developer Productivity:** By integrating Prisma with PostgreSQL, developers benefit from automated type generation and seamless database workflows, which speed up development time and reduce the likelihood of runtime errors, resulting in faster feature delivery and easier maintenance.
- **Security:** PostgreSQL offers robust security features, including authentication, authorization, and data encryption, while Prisma helps enforce secure database access patterns through its generated queries, ensuring that sensitive thesis and user data are well protected.

1.3.5 Conclusion

This technology stack facilitates rapid development and seamless integration with external services, which is essential for adapting to the ever-changing needs of academic institutions. The modular architecture of NestJS, combined with the component-based structure of NextJS, enables development teams to work concurrently on different parts of the system without conflict, accelerating feature delivery and boosting overall productivity.

Moreover, employing TypeScript consistently across the entire codebase introduces strong typing and robust tooling support, significantly enhancing code quality and maintainability. Early detection of errors and enforced coding standards help reduce technical debt, while simplifying onboarding for new developers, fostering smoother collaboration and more efficient teamwork.

Together, these technologies establish a solid, scalable foundation that prioritizes security, flexibility, and adaptability. The system is well-equipped to handle growing user demands, integrate new functionalities, and evolve alongside shifting academic requirements. Ultimately, this cohesive stack ensures a reliable, efficient, and future-proof thesis management platform.

In addition, this combination promotes long-term sustainability and simplified maintenance. Clear separation of concerns and well-defined interfaces between the frontend, backend, and database layers allow updates and improvements to be implemented with minimal disruption. Automated testing coupled with strong typing minimizes the risk of bugs during development and deployment, resulting in a stable and resilient system. This thoughtful design not only supports current needs but also paves the way for future innovations, making it an ideal solution for the dynamic environment of academia.

CHAPTER 2 REQUIREMENTS ANALYSIS AND SYSTEM DESIGN

2.1 Business analysis

2.1.1 Business requirements analysis

Research Direction Registration Process:

- Lecturers register primary research directions they can supervise.
- The system enforces maximum student capacity per research direction.
- Department and faculty administrators review and approve research directions.
- Students can view the list of approved research directions.

Supervisor Assignment Process

- Faculty administrators assign supervisors prioritizing lecturer and student preferences.
- Assignments respect maximum supervisory loads and lecturers' expertise areas.

Thesis Outline Management Process

- Students prepare and submit thesis outlines using standardized templates.
- Supervisors review outlines and request modifications when needed.
- Department administrators approve finalized outlines.
- Once approved, outlines are locked from further student edits.

Review and Grading Process

- Faculty administrators assign independent reviewers aligned with thesis topics.
- Reviewers provide evaluations and assign scores.
- The system calculates average scores and generates reports.

Thesis Defense Process

- Defense committees composed of 3–5 faculty members are formed.
- Defense schedules are planned and managed.
- Grades and revision requests are recorded and managed.
- Final defense results and reports are generated.

2.1.2 System requirements analysis

User Management

- User registration and authentication.
- Role-based access control.
- Personal information management.
- Management of faculty and student profiles.

Research Direction Management

- Registration and approval of research directions.
- Management of student capacity per direction.
- Search and filter functionality for research directions.

Thesis Topic Management

- Topic registration and supervisor assignment.
- Progress tracking and status monitoring.

Thesis Outline Management

- Creation, editing, and document upload.
- Outline approval workflow and locking mechanism.

Defense Management

- Formation of defense committees.
- Scheduling defenses.
- Grading and reporting functions.

Non-Functional Requirements

- Data security and confidentiality.
- System performance and responsiveness.
- Scalability for increasing user base and data volume.
- Maintainability and ease of updates.
- User-friendly and accessible interface design.
- Reliability and availability to minimize downtime.
- Compliance with relevant legal and institutional regulations.
- Efficient resource utilization for cost-effectiveness.
- Robust error handling and recovery mechanisms.
- Support for multi-device and cross-platform access.

2.1.3 Role-permission matrix

This matrix establishes and defines the permissions assigned to each account involved in the thesis defense process. It helps build a clear and secure access framework, ensuring that every user role has the appropriate rights to perform their tasks efficiently throughout the entire workflow.

Table 2.1. Role-Permission Matrix

Permission / Role	**Dean**	**Department Head / Secretary**	**Supervising Lecturer**	**Reviewing Lecturer**	**Student**
Register research topics			✓		
Assign supervisors to students	✓				
Create and upload project proposals					✓
View/edit proposals of supervised students			✓		
Request proposal revisions to supervised students			✓		
Upload finalized/agreed proposal			✓		
Prevent student from editing proposal after final	✓		✓		
Review proposals within department		✓			
Export proposals report (topic, student, supervisor)		✓			
Request proposal revisions at department level		✓			
Allow student to edit proposal after department review		✓			✓
Lock proposal editing rights after finalization	✓				

Permission / Role	**Dean**	**Department Head / Secretary**	**Supervising Lecturer**	**Reviewing Lecturer**	**Student**
Export finalized proposal reports	✓				
Assign reviewers to supervisors	✓				
Evaluate and grade students			✓	✓	
Export evaluation and grading reports			✓	✓	
Create thesis defense committees	✓				
Ensure students are not in committees with their supervisors	✓				
Input scores and manage revision requests		✓			
Export committee score sheets		✓			
Send revision requests to students		✓			
Direct interaction between supervisors and students			✓		✓

2.1 System design

2.2.1 Use case diagram

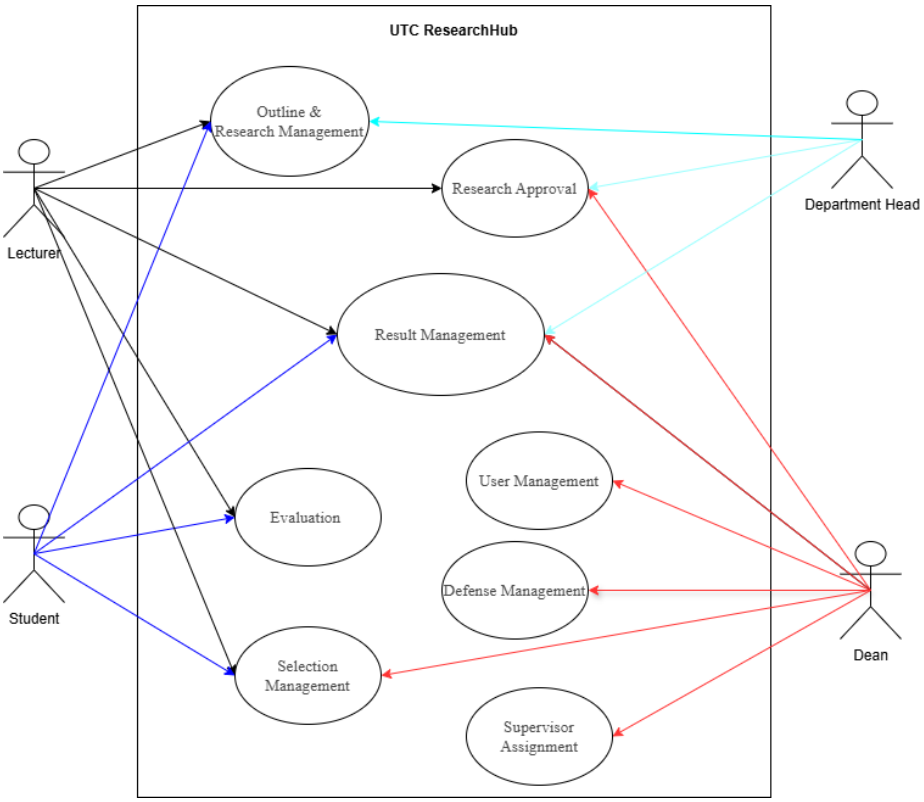

Figure 2.1. Use case diagram

2.2.2 Business function diagram

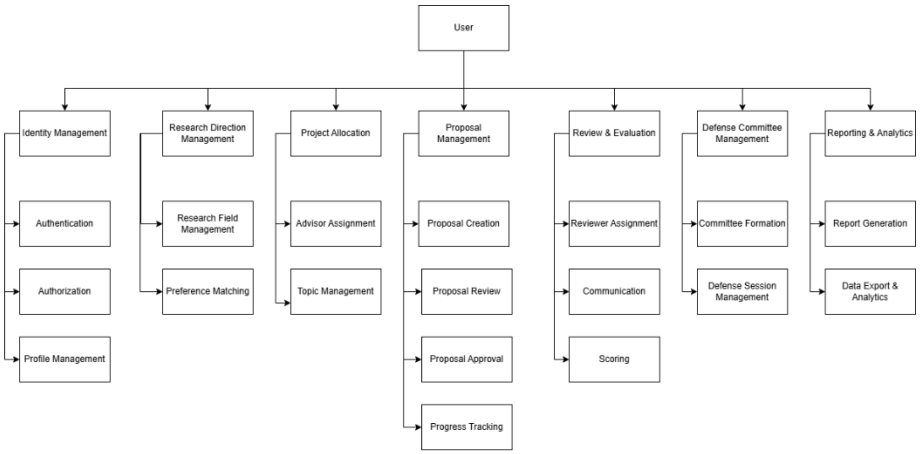

Figure 2.2. Business Function Diagram

2.2.3 System architecture diagram

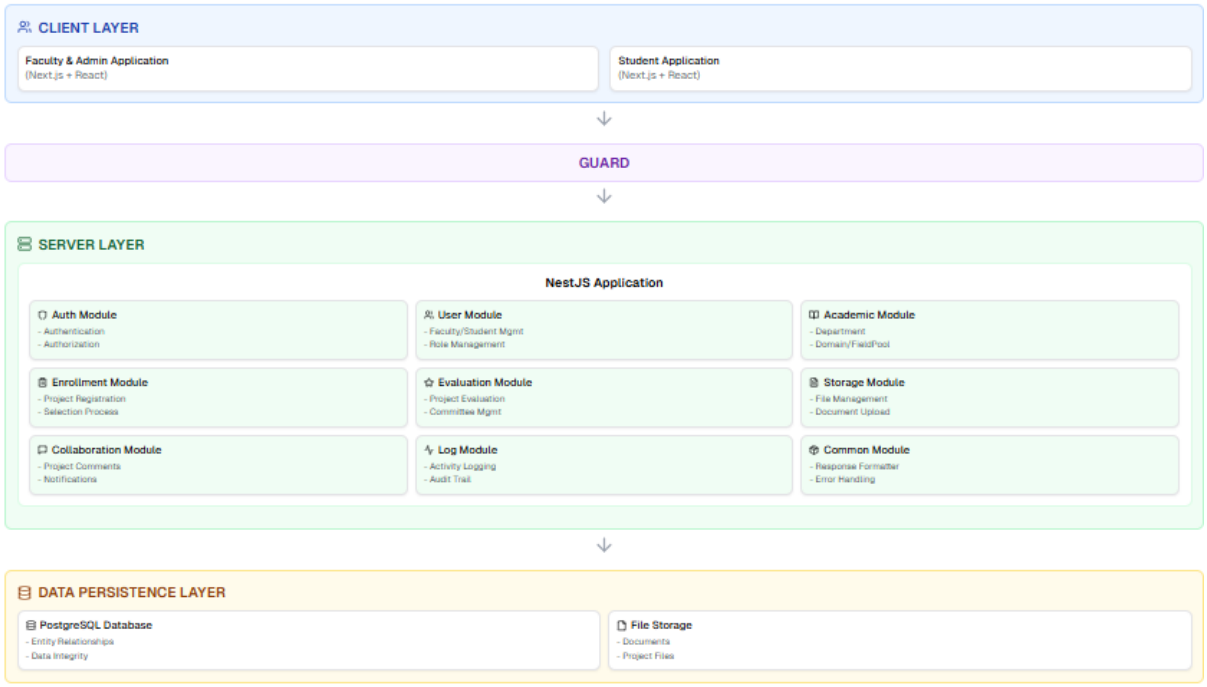

Figure 2.3. System Architecture Diagram

The system is designed with two distinct frontend applications to serve different user groups: one dedicated to students and another tailored for faculty members. Each frontend provides a customized user interface and functionality aligned with the specific needs and workflows of its user base.

Both frontend applications interact with a single, centralized backend server responsible for managing the core business logic, data processing, and communication with the database. The backend handles key operations such as user authentication and authorization, project management workflows, evaluation processes, notification delivery, and file storage management.

This architectural separation enables a more focused and intuitive user experience for students and faculty, while the unified backend ensures consistency, easier maintenance, and scalability. The design supports future expansion and integration without affecting the user interfaces.

2.2.3 Sequence diagram

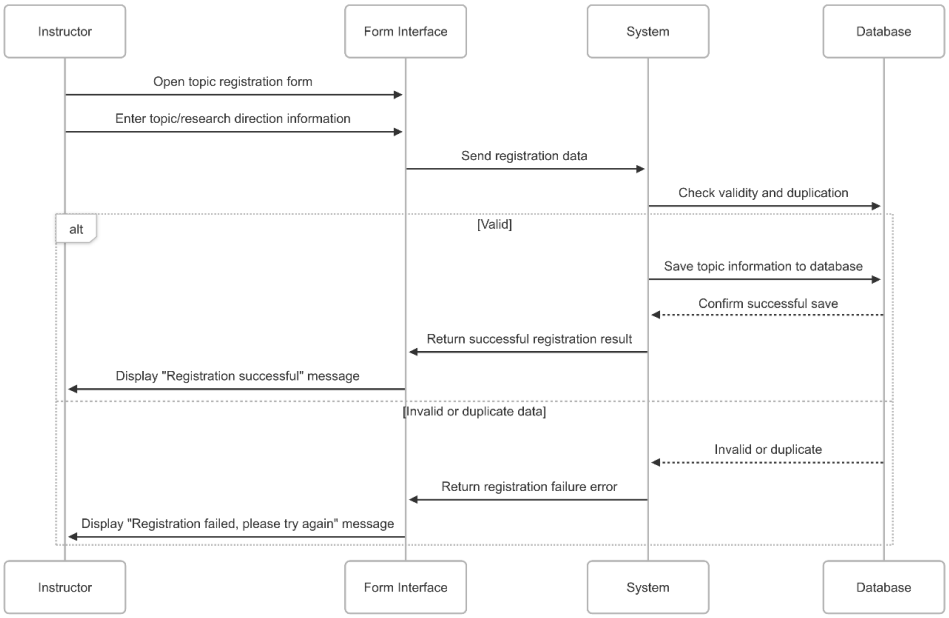

Figure 2.4. Topic Registration Process SD

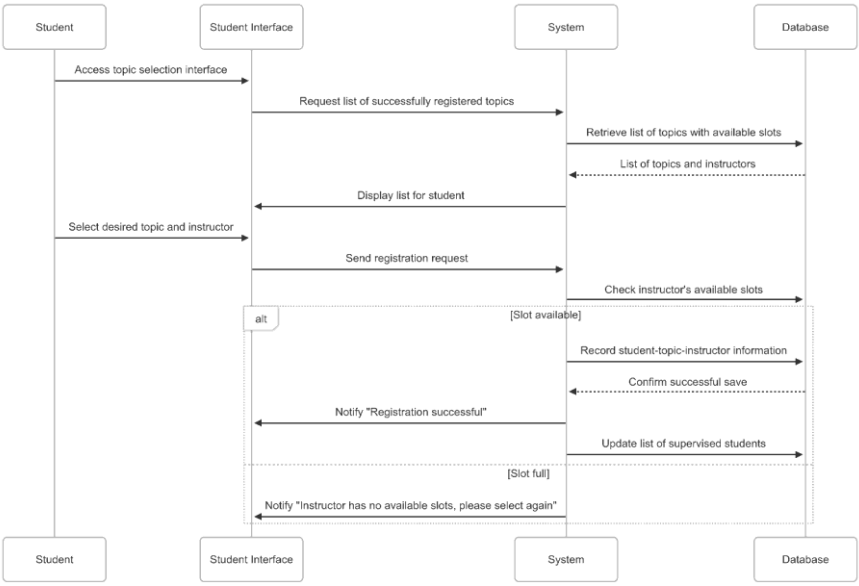

Figure 2.5. Student Topic Selection SD

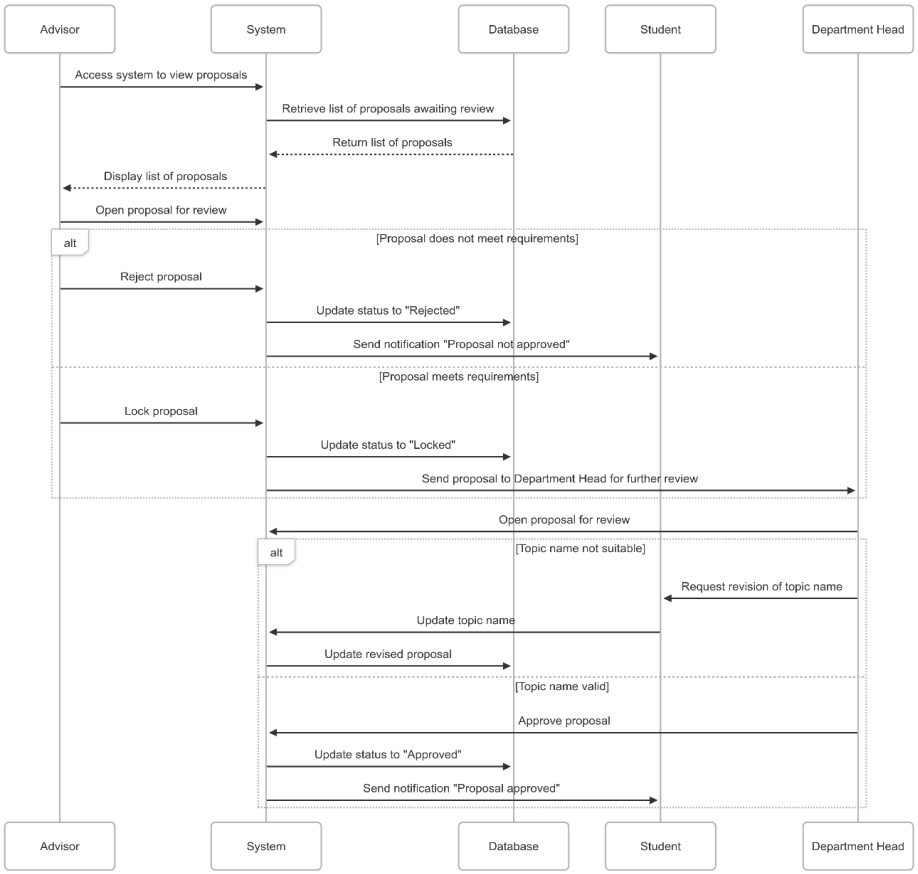

Figure 2.6. Proposal Review and Approval Process SD

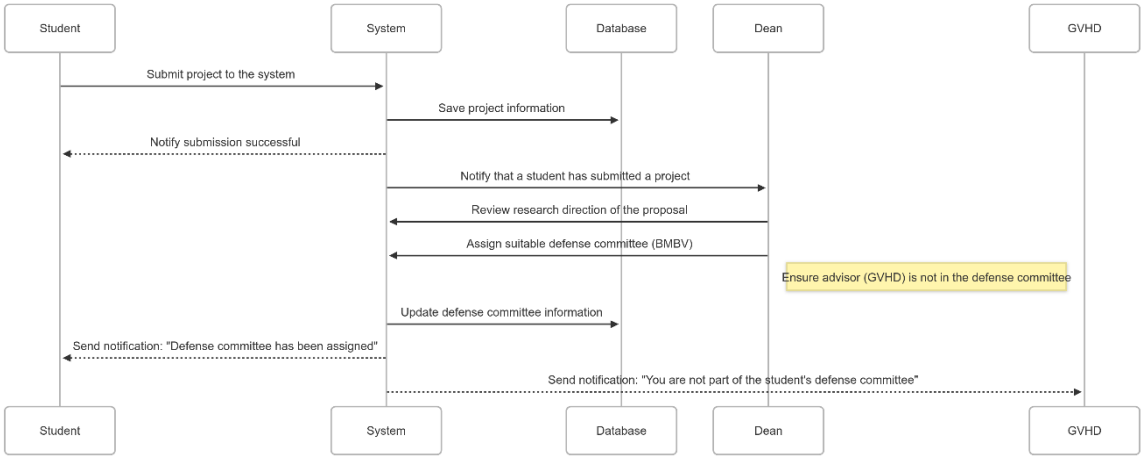

Figure 2.7. Submission Process SD

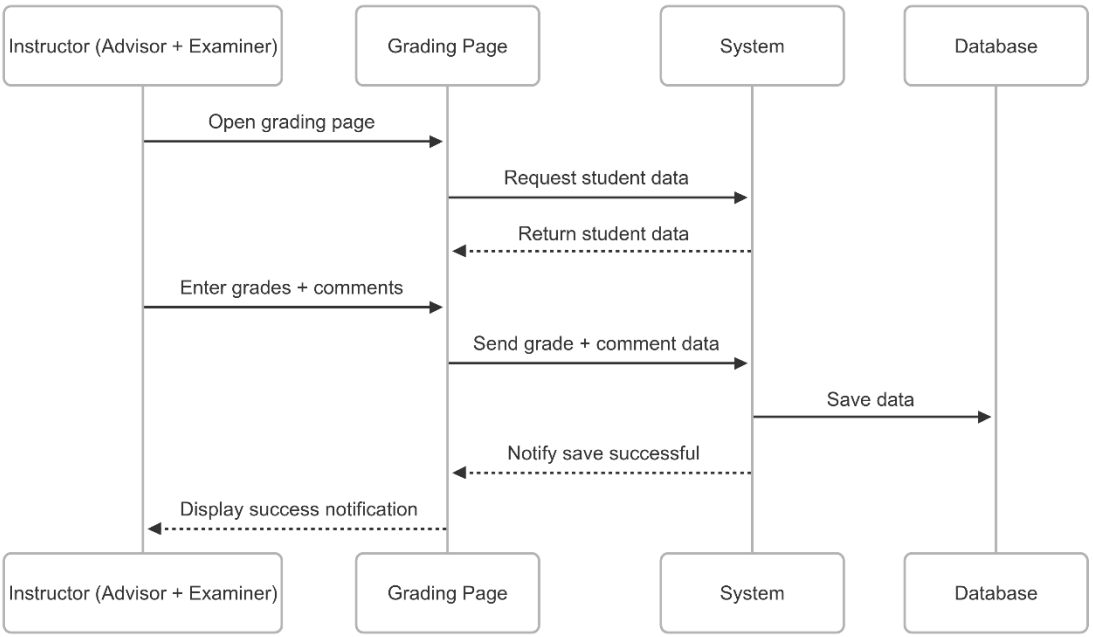

Figure 2.8. Grading SD

2.3 Database Design

2.3.1 Database table relations

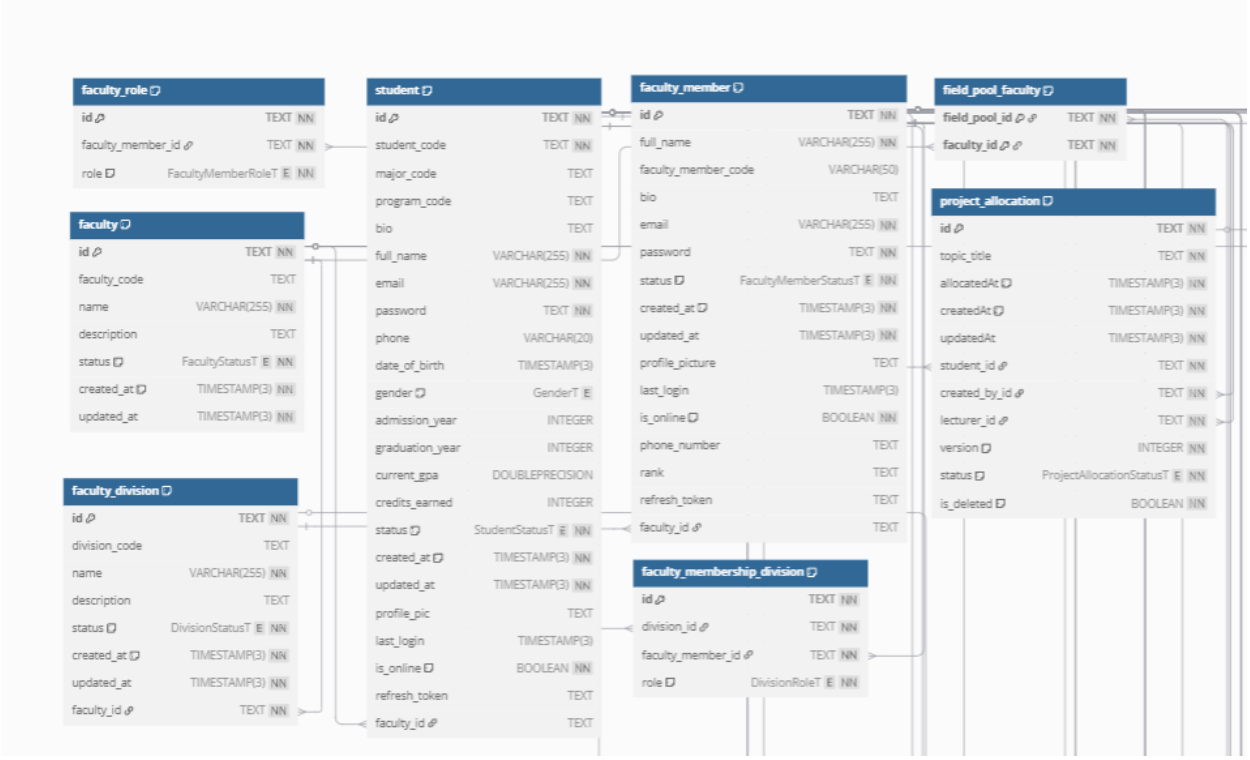

Figure 2.9. User management ERD

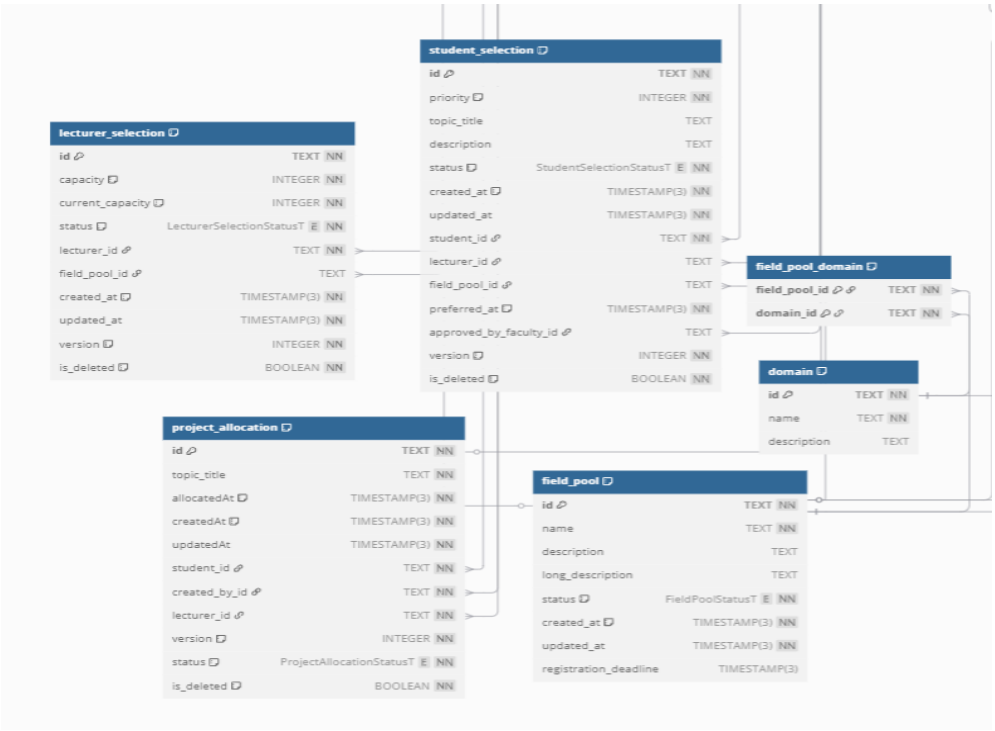

Figure 2.10. Selection management and supervisor assignment ERD

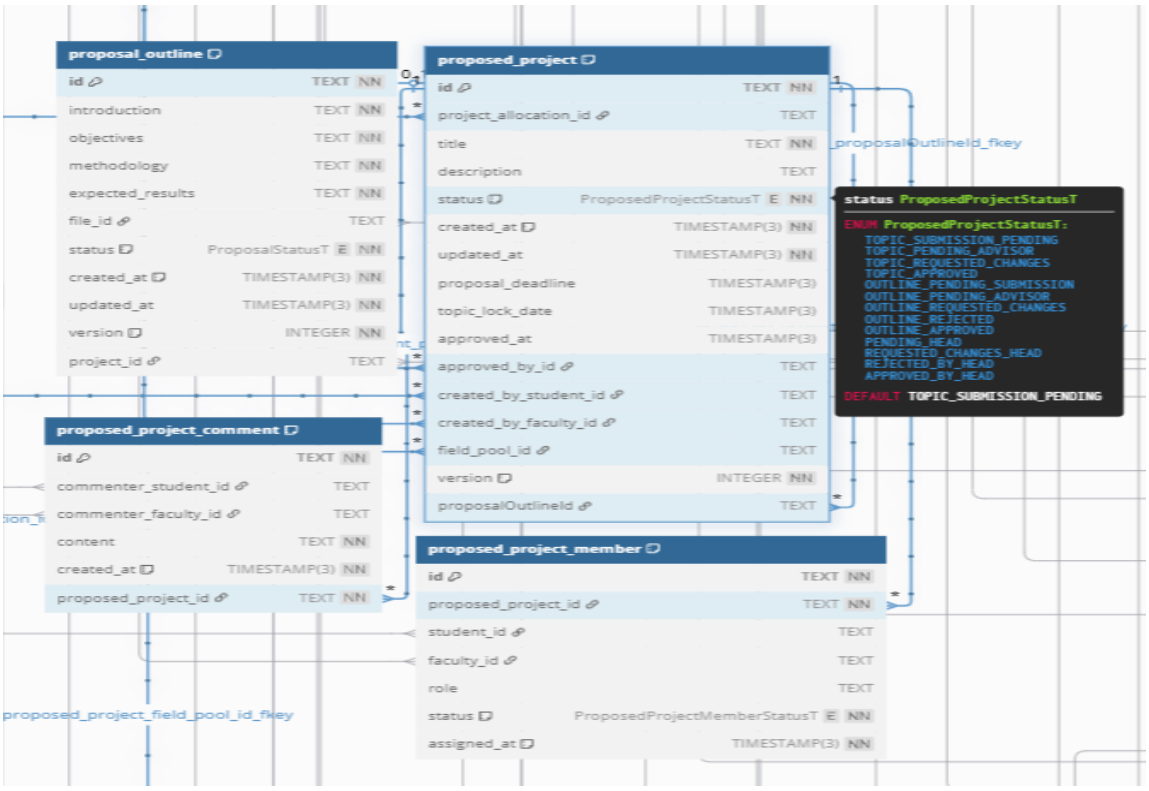

Figure 2.11. Result Approval ERD

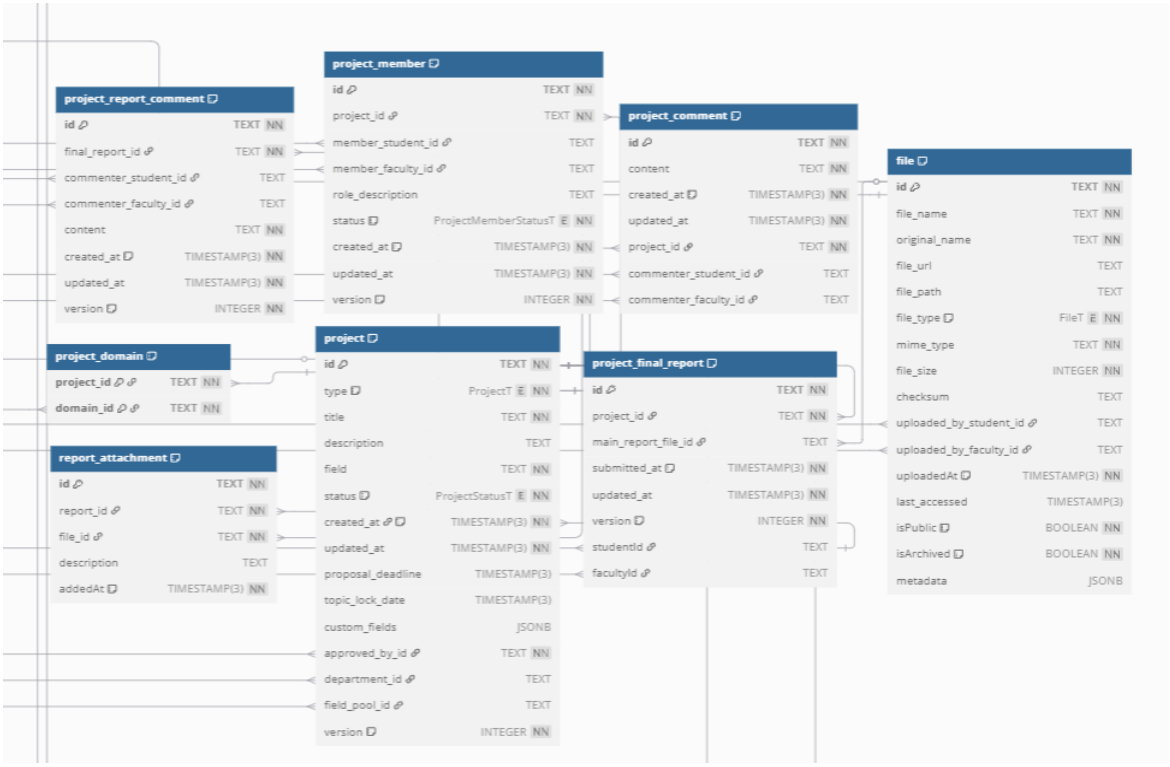

Figure 2.12. Research and result management ERD

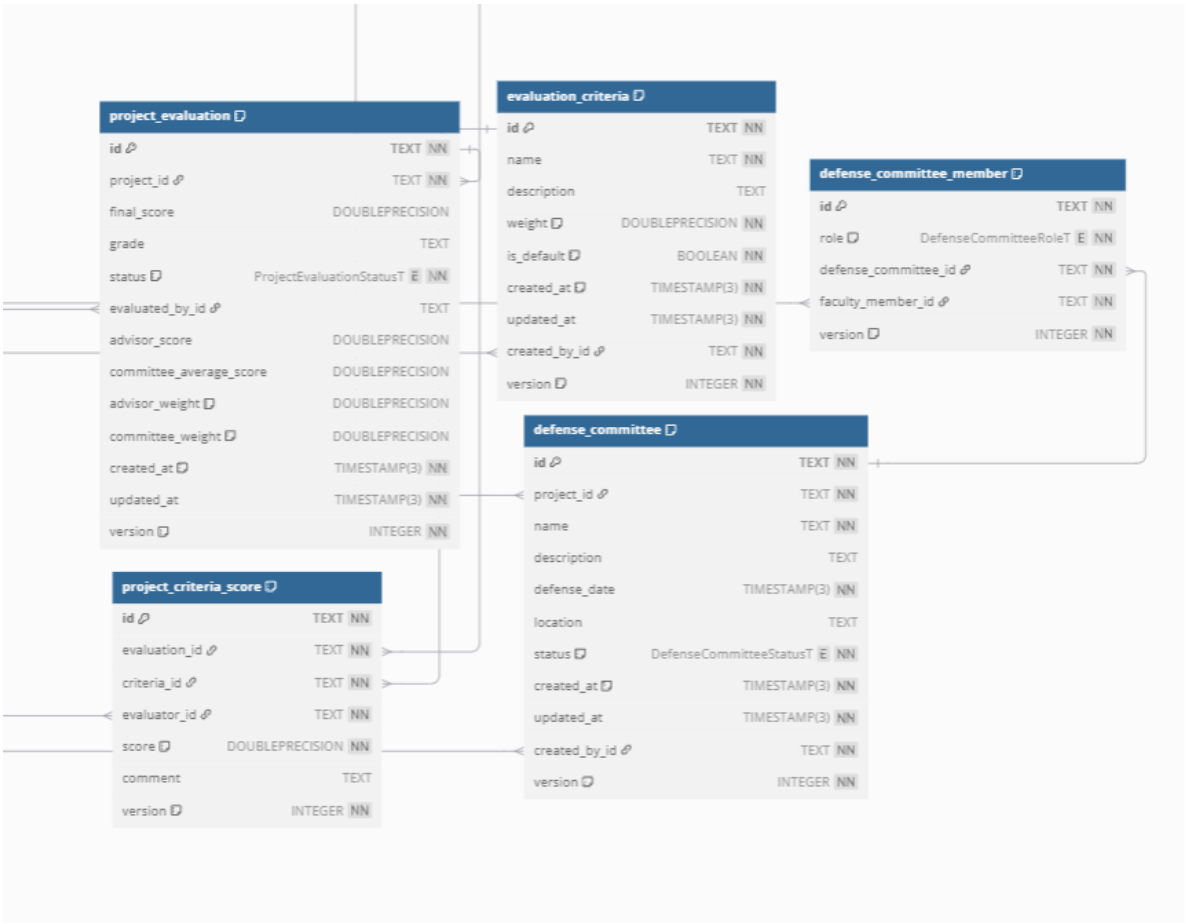

Figure 2.13. Defense and Evaluation management ERD

2.3.2 Database description

Table 2.2. Student Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier for the student
student_code	TEXT	Unique	Student code
major_code	TEXT		Code for the major
program_code	TEXT		Code for the academic program
bio	TEXT		Short biography
full_name	VARCHAR(255)		Full name
email	VARCHAR(255)	Unique	Email address
password	TEXT		Login password
phone	VARCHAR(20)		Phone number
date_of_birth	TIMESTAMP(3)		Date of birth
gender	GenderT (ENUM)		Gender (Male, Female, Other)
admission_year	INTEGER		Year of admission
graduation_year	INTEGER		Year of graduation
current_gpa	DOUBLE PRECISION		Current GPA
credits_earned	INTEGER		Number of credits earned
status	StudentStatusT (ENUM)	Not Null, Default 'ACTIVE'	Student status (Active, Inactive, etc.)
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Record last update timestamp
profile_pic	TEXT		Profile picture URL or path
last_login	TIMESTAMP(3)		Last login timestamp
is_online	BOOLEAN	Not Null, Default false	Online status

refresh_token	TEXT		Token for session refresh
faculty_id	TEXT	Foreign Key	Faculty identifier (links to faculty table)

Table 2.3. Faculty Member Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier for faculty member
full_name	VARCHAR(255)		Full name
faculty_member_code	VARCHAR(50)	Unique	Faculty member code
bio	TEXT		Biography
email	VARCHAR(255)	Unique	Email address
password	TEXT		Login password
status	FacultyMemberStatusT (ENUM)	Not Null, Default 'ACTIVE'	Employment status (Active, Retired, etc.)
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Record last update timestamp
profile_picture	TEXT		Profile picture URL or path
last_login	TIMESTAMP(3)		Last login timestamp
is_online	BOOLEAN	Not Null, Default false	Online status
phone_number	TEXT		Phone number
rank	TEXT		Academic rank or title
refresh_token	TEXT		Token for session refresh
faculty_id	TEXT	Foreign Key	Faculty identifier (links to faculty table)

Table 2.4. Faculty Member Role Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier for role record
faculty_member_id	TEXT	Foreign Key	Faculty member identifier (links to faculty_member)
role	FacultyMemberRoleT (ENUM)	Not Null	Role assigned (Admin, Dean, Lecturer)

Table 2.5. Faculty Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier for faculty
faculty_code	TEXT		Code for faculty
name	VARCHAR(255)	Unique	Faculty name
description	TEXT		Description of faculty
status	FacultyStatusT (ENUM)	Not Null, Default 'ACTIVE'	Faculty status (Active, Inactive)
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Record last update timestamp

Table 2.5. Division Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier for division
division_code	TEXT		Division code

name	VARCHAR(255)	Not Null	Division name
description	TEXT		Description of division
status	DivisionStatusT (ENUM)	Not Null, Default 'ACTIVE'	Status (Active, Inactive)
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Record last update timestamp
faculty_id	TEXT	Foreign Key	Faculty identifier linked to this division

Table 2.6. Faculty Membership Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
division_id	TEXT	Foreign Key	Division identifier (links to faculty_division)
faculty_member_id	TEXT	Foreign Key	Faculty member identifier (links to faculty_member)
role	DivisionRoleT (ENUM)	Not Null	Role in division (Lecturer, Head)

Table 2.7. FieldPool Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier for field pool
name	TEXT	Unique	Name of field pool
description	TEXT		Short description

long_description	TEXT		Detailed description
status	FieldPoolStatusT (ENUM)	Not Null, Default 'CLOSED'	Status (Open, Closed, Hidden)
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Record last update timestamp
registration_deadline	TIMESTAMP(3)		Deadline for registration

Table 2.8. FieldPool Faculty Table

Column	Data Type	Key	Description
field_pool_id	TEXT	Primary Key (Composite)	Field pool identifier
faculty_id	TEXT	Primary Key (Composite)	Faculty identifier

Table 2.9. Domain Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
name	TEXT	Unique	Domain name
description	TEXT		Description of domain

Table 2.10. Lecturer Selection Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier

capacity	INTEGER	Not Null, Default 1	Maximum number of students lecturer can accept
current_capacity	INTEGER	Not Null, Default 0	Current accepted students count
status	LecturerSelectionStatusT (ENUM)	Not Null, Default 'PENDING'	Selection status (Pending, Approved, Rejected)
lecturer_id	TEXT	Foreign Key	Lecturer identifier
field_pool_id	TEXT	Foreign Key	Field pool identifier
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Record creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Record last update timestamp
version	INTEGER	Not Null, Default 1	Version number
is_deleted	BOOLEAN	Not Null, Default false	Logical deletion flag

Table 2.11. Student Selection Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
priority	INTEGER	Not Null, Default 1	Priority number (1 is highest)
topic_title	TEXT		Title of the topic

description	TEXT		Topic description
status	StudentSelectionStatusT (ENUM)	Not Null, Default 'PENDING'	Selection status (Pending, Approved, Rejected)
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_ TIMESTAM P	Creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Last update timestamp
student_id	TEXT	Foreign Key	Student identifier
lecturer_id	TEXT	Foreign Key	Lecturer identifier
field_pool_id	TEXT	Foreign Key	Field pool identifier
preferred_at	TIMESTAMP(3)	Not Null, Default CURRENT_ TIMESTAM P	Timestamp when preferred
approved_by _faculty_me mber_id	TEXT	Foreign Key	Faculty member who approved
version	INTEGER	Not Null, Default 1	Version number
is_deleted	BOOLEAN	Not Null, Default false	Logical deletion flag

Table 2.12. Allocation Table

Column	**Data Type**	**Key**	**Descriptio n**
id	TEXT	Primary Key	Unique identifier
topic_title	TEXT	Not Null	Title of allocated project

allocatedAt	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Allocation timestamp
createdAt	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp
updatedAt	TIMESTAMP(3)	Not Null	Last update timestamp
student_id	TEXT	Foreign Key	Student identifier
created_by_id	TEXT	Foreign Key	Faculty member who created allocation
lecturer_id	TEXT	Foreign Key	Lecturer identifier
version	INTEGER	Not Null, Default 1	Version number
status	ProjectAllocationStatusT (ENUM)	Not Null, Default 'PENDING'	Allocation status
is_deleted	BOOLEAN	Not Null, Default false	Logical deletion flag

Table 2.13. Proposal Project Table

Column	**Data Type**	**Key**	**Description**
id	TEXT	Primary Key	Unique identifier
project_allocation_id	TEXT	Foreign Key	Project allocation identifier
title	TEXT	Not Null	Project title

description	TEXT		Project description
status	ProposedProjectStatusT (ENUM)	Not Null, Default 'TOPIC_SUBMISSION_PENDING'	Project status
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Last update timestamp
proposal_deadline	TIMESTAMP(3)		Deadline for proposal submission
topic_lock_date	TIMESTAMP(3)		Lock date for topic
approved_at	TIMESTAMP(3)		Approval timestamp
approved_by_id	TEXT	Foreign Key	Faculty member who approved
created_by_student_id	TEXT	Foreign Key	Student creator
created_by_faculty_id	TEXT	Foreign Key	Faculty creator
field_pool_id	TEXT	Foreign Key	Field pool identifier
version	INTEGER	Not Null, Default 1	Version number

proposal_outline_id	TEXT	Foreign Key	Proposal outline identifier

Table 2.14. Proposal Project Member

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
proposed_project_id	TEXT	Foreign Key	Proposed project identifier
student_id	TEXT	Foreign Key	Student member identifier
faculty_member_id	TEXT	Foreign Key	Faculty member identifier
role	TEXT		Role in project
status	ProposedProjectMemberStatusT (ENUM)	Not Null, Default 'ACTIVE'	Member status
assigned_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Assignment timestamp

Table 2.15. Proposal Comment Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
commenter_student_id	TEXT	Foreign Key	Student who commented

commenter_faculty_id	TEXT	Foreign Key	Faculty member who commented
content	TEXT	Not Null	Comment content
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp
proposed_project_id	TEXT	Foreign Key	Proposed project identifier

Table 2.15. Outline Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
introduction	TEXT	Not Null	Introduction section
objectives	TEXT	Not Null	Objectives section
methodology	TEXT	Not Null	Methodology section
expected_results	TEXT	Not Null	Expected results section
file_id	TEXT	Foreign Key	Related file identifier
status	ProposalOutlineStatusT (ENUM)	Not Null, Default 'DRAFT'	Outline status
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Last update timestamp

version	INTEGER	Not Null, Default 1	Version number
project_id	TEXT	Foreign Key	Linked project identifier

Table 2.16. Project Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
type	ProjectT (ENUM)	Not Null	Project type (Graduated, Research, etc.)
title	TEXT	Not Null	Project title
description	TEXT		Project description
field	TEXT	Not Null	Field or domain of project
status	ProjectStatusT (ENUM)	Not Null, Default 'IN_PROGRESS'	Project status
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Last update timestamp
proposal_deadline	TIMESTAMP(3)		Proposal submission deadline
topic_lock_date	TIMESTAMP(3)		Topic lock date
custom_fields	JSONB		Additional custom fields
approved_by_id	TEXT	Foreign Key	Faculty member who approved

field_pool_id	TEXT	Foreign Key	Field pool identifier
version	INTEGER	Not Null, Default 1	Version number
division_id	TEXT	Foreign Key	Division identifier

Table 2.17. Project Member

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
project_id	TEXT	Foreign Key	Project identifier
student_id	TEXT	Foreign Key	Student member identifier
faculty_member_id	TEXT	Foreign Key	Faculty member identifier
role	TEXT		Role in project
status	ProposedProjectMemberStatusT (ENUM)	Not Null, Default 'ACTIVE'	Status of member
assigned_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Assignment timestamp

Table 2.18. Project Comment Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
content	TEXT	Not Null	Comment content

created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Last update timestamp
project_id	TEXT	Foreign Key	Project identifier
commenter_student_id	TEXT	Foreign Key	Student commenter identifier
commenter_faculty_member_id	TEXT	Foreign Key	Faculty member commenter identifier

Table 2.19. Project Domain Table

Column	Data Type	Key	Description
project_id	TEXT	Primary Key (Composite)	Project identifier
domain_id	TEXT	Primary Key (Composite)	Domain identifier

Table 2.20. Project Report Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
project_id	TEXT	Foreign Key	Project identifier
main_report_file_id	TEXT	Foreign Key	Main report file identifier
submitted_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Submission timestamp
updated_at	TIMESTAMP(3)	Not Null	Last update timestamp

version	INTEGER	Not Null, Default 1	Version number
student_id	TEXT	Foreign Key	Student identifier
faculty_id	TEXT	Foreign Key	Faculty identifier
faculty_member_id	TEXT	Foreign Key	Faculty member identifier

Table 2.21. Project Report Attachment Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
report_id	TEXT	Foreign Key	Final report identifier
file_id	TEXT	Foreign Key	File identifier
description	TEXT		Description of attachment
addedAt	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Timestamp when added

Table 2.22. Project Report Comment Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
final_report_id	TEXT	Foreign Key	Final report identifier
commenter_student_id	TEXT	Foreign Key	Student commenter
commenter_faculty_member_id	TEXT	Foreign Key	Faculty member

			commenter
content	TEXT	Not Null	Comment content
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Last update timestamp
version	INTEGER	Not Null, Default 1	Version number

Table 2.23. Defense Commit Table

Column	**Data Type**	**Key**	**Description**
id	TEXT	Primary Key	Unique identifier
project_id	TEXT	Foreign Key	Project identifier
name	TEXT	Not Null	Name of the defense committee
description	TEXT		Description
defense_date	TIMESTAMP(3)	Not Null	Date of defense
location	TEXT		Defense location
status	DefenseCommitteeStatusT (ENUM)	Not Null, Default 'PREPARING'	Committee status
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Last update timestamp

created_by_id	TEXT	Foreign Key	Creator faculty member identifier
version	INTEGER	Not Null, Default 1	Version number

Table 2.24. Defense Commit Member Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
role	DefenseCommitteeRoleT (ENUM)	Not Null	Role in committee (Chairman, Secretary, etc.)
defense_committee_id	TEXT	Foreign Key	Defense committee identifier
faculty_member_id	TEXT	Foreign Key	Faculty member identifier
version	INTEGER	Not Null, Default 1	Version number

Table 2.25. Project Evaluation Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
project_id	TEXT	Foreign Key	Project identifier
final_score	DOUBLE PRECISION		Final evaluation score
status	ProjectEvaluationStatusT (ENUM)	Not Null, Default 'PENDING'	Evaluation status

advisor_weight	DOUBLE PRECISION		Weight assigned to advisor's score
committee_weight	DOUBLE PRECISION		Weight assigned to committee's score
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp
updated_at	TIMESTAMP(3)	Not Null	Last update timestamp
version	INTEGER	Not Null, Default 1	Version number

Table 2.26. Project Evaluation Score Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
evaluation_id	TEXT	Foreign Key	Evaluation identifier
evaluator_id	TEXT	Foreign Key	Evaluator (faculty member) identifier
role	EvaluatorRole (ENUM)	Not Null, Default 'COMMITTEE'	Role of evaluator (Advisor, Committee)
score	DOUBLE PRECISION	Not Null	Score given by evaluator
comment	TEXT		Comment from evaluator
created_at	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Creation timestamp

updated_at	TIMESTAMP(3)	Not Null	Last update timestamp
version	INTEGER	Not Null, Default 1	Version number

Table 2.27. File Table

Column	Data Type	Key	Description
id	TEXT	Primary Key	Unique identifier
file_name	TEXT	Not Null	Stored file name
original_name	TEXT	Not Null	Original uploaded file name
file_url	TEXT		URL to access the file
file_path	TEXT		File system path
file_type	FileT (ENUM)	Not Null	File type (PDF, Word, Image, etc.)
mime_type	TEXT	Not Null	MIME type of the file
file_size	INTEGER	Not Null	File size in bytes
checksum	TEXT		Checksum for file integrity
uploaded_by_student_id	TEXT	Foreign Key	Uploader student identifier
uploaded_by_faculty_id	TEXT	Foreign Key	Uploader faculty

			member identifier
uploadedAt	TIMESTAMP(3)	Not Null, Default CURRENT_TIMESTAMP	Upload timestamp
last_accessed	TIMESTAMP(3)		Last accessed timestamp
isPublic	BOOLEAN	Not Null, Default false	Whether file is public
isArchived	BOOLEAN	Not Null, Default false	Whether file is archived
metadata	JSONB		Additional metadata

2.3.3 Conclusion

The database uses ENUM types to standardize statuses and roles, ensuring consistent data. Foreign keys maintain strong relationships between tables. Versioning helps track data changes, and soft deletion preserves historical data without removing it. Security features protect sensitive information and control user access.

To facilitate easier management and future feature expansions, projects that are not yet approved and those that have been approved are separated into two distinct tables. This separation allows for clearer workflows and simpler implementation of new business rules.

Overall, this design covers the full academic research process, enabling smooth collaboration and management between students and faculty. This structure provides a reliable, scalable foundation for a university research project management system.

CHAPTER 3 IMPLEMENTATION AND TESTING

3.1 Environment, tools, and directory structure

Development Tools:

- **Code Editor: Visual Studio Code**, a versatile and efficient code editor, was used for writing and debugging code.

- **Version Control: GitLab** was utilized for version control and collaboration, enabling team members to manage code repositories and track changes effectively throughout the project.

- **Database Management: PGAdmin 4** was used to manage the PostgreSQL database, allowing for easy administration, query execution, and data visualization.
- **API Testing:** Swagger was used for testing and documenting the APIs, enabling easy interaction with the API endpoints and ensuring that they function as expected.

3.2 Main modules overview

3.2.1 Authentication

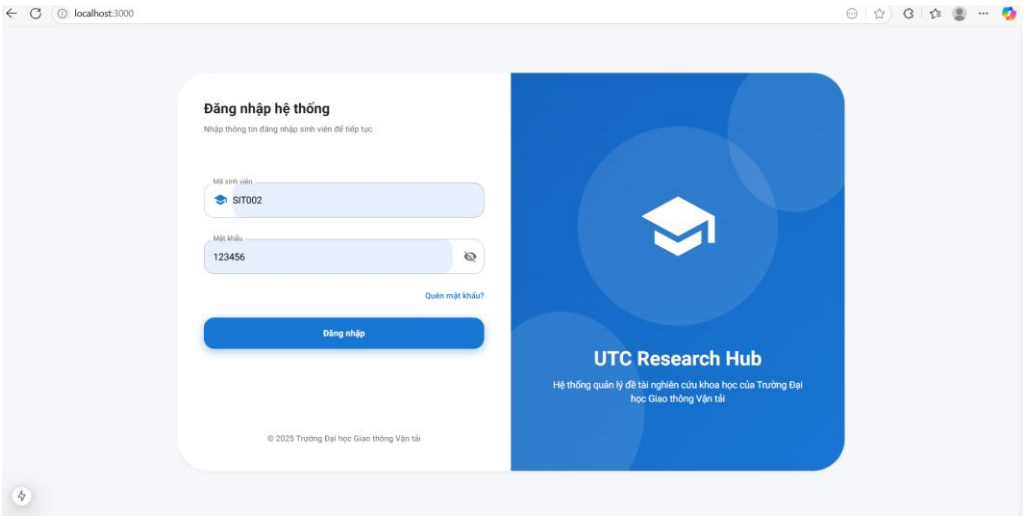

Figure 3.1. Login Page (Student)

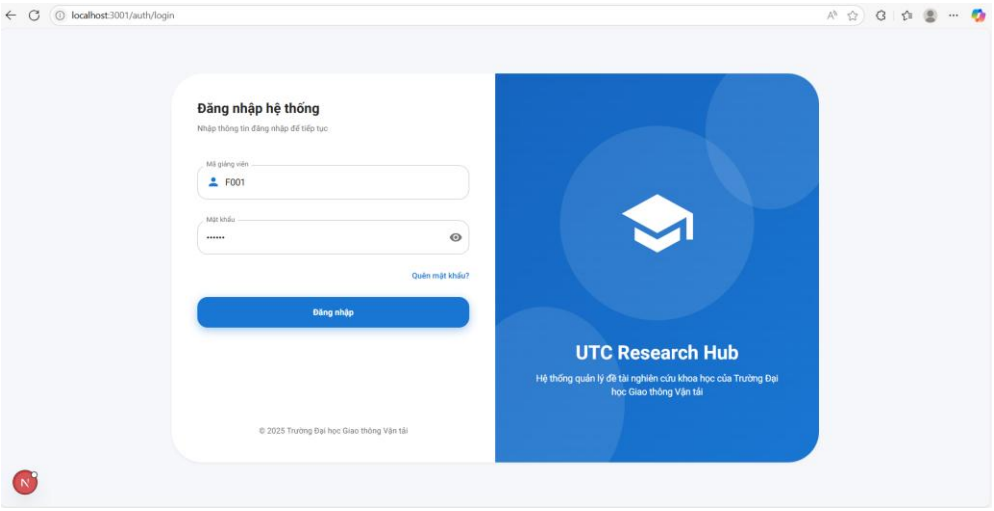

Figure 3.2. Login Page (Faculty)

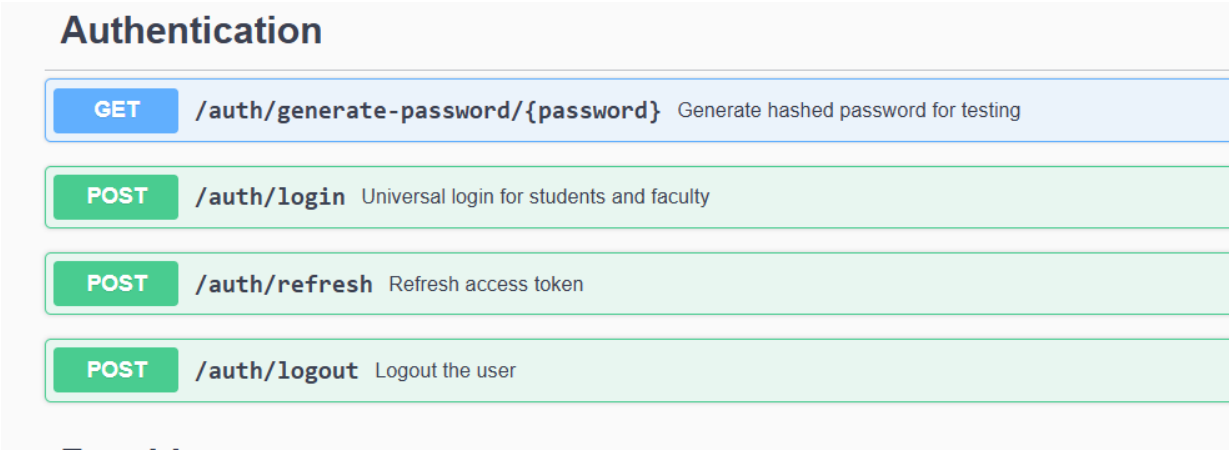

Figure 3.3. Login API

3.2.2 Selection

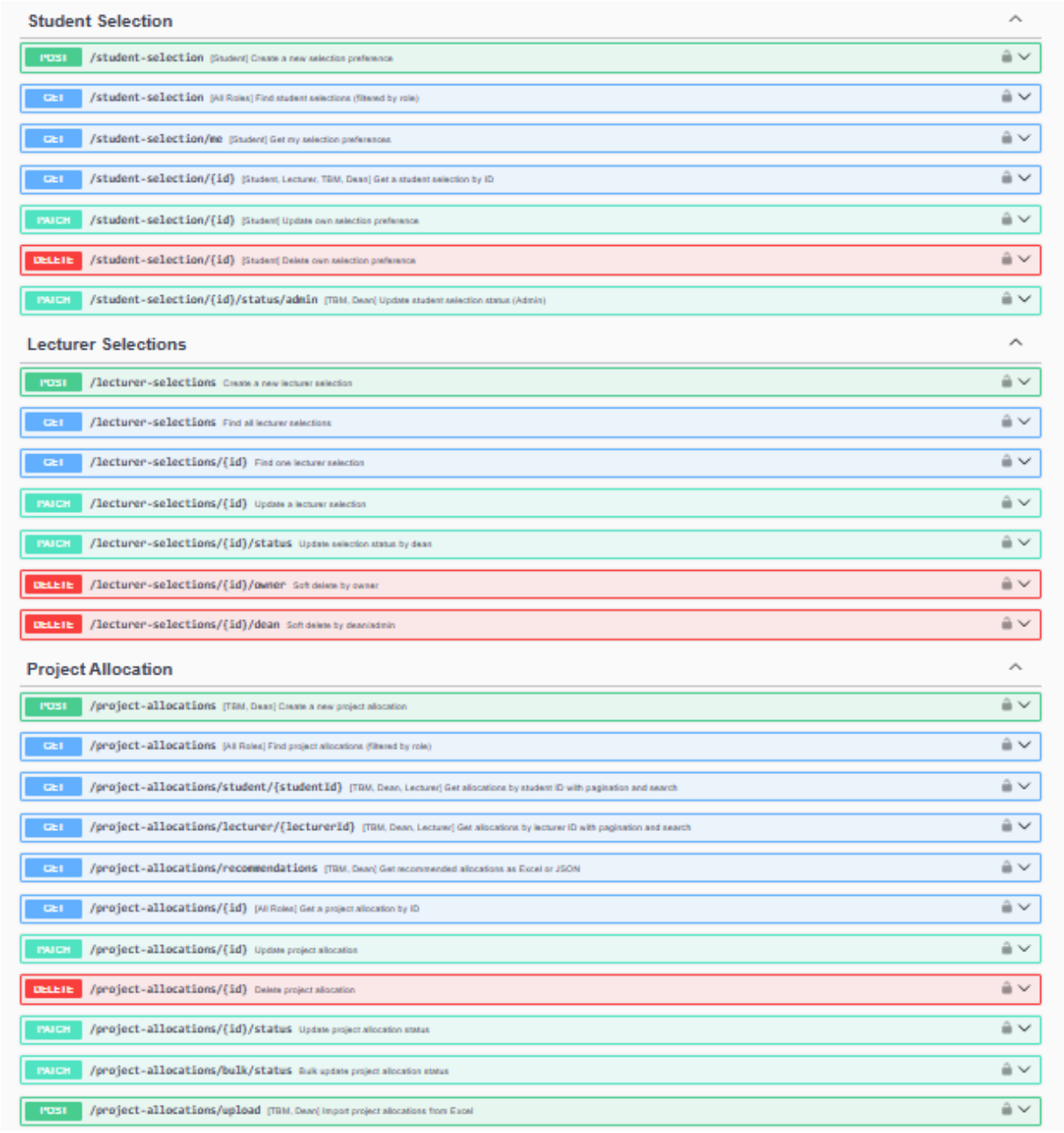

Figure 3.4. Preferences API

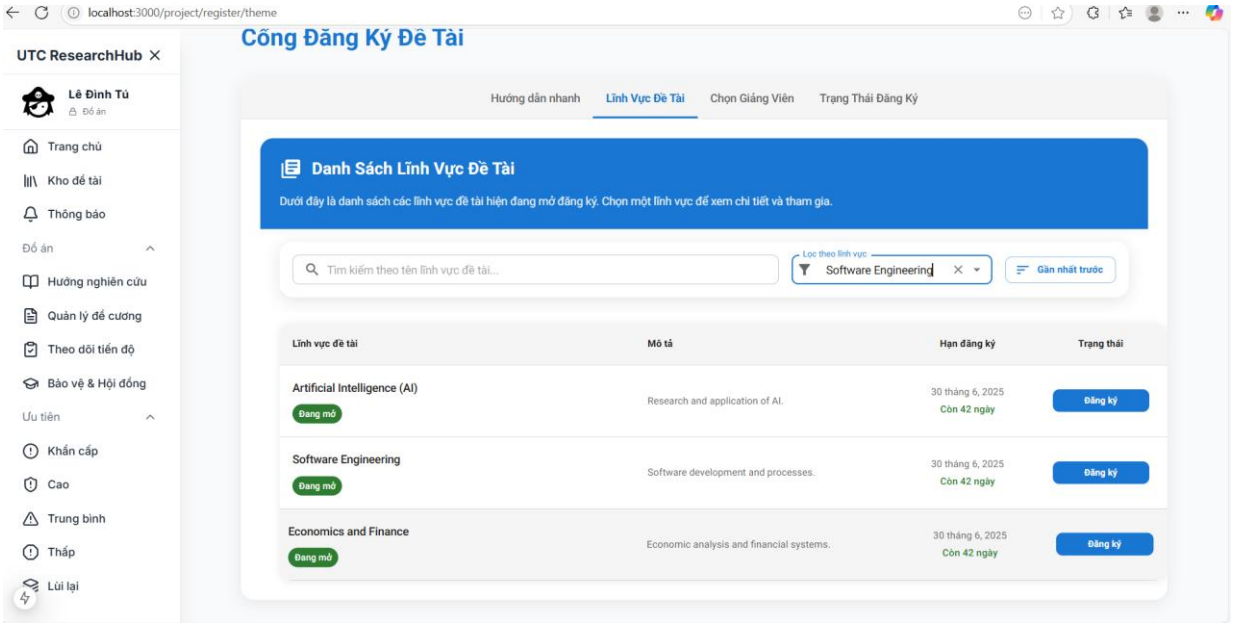

Figure 3.5. Field Preferences Page (Student)

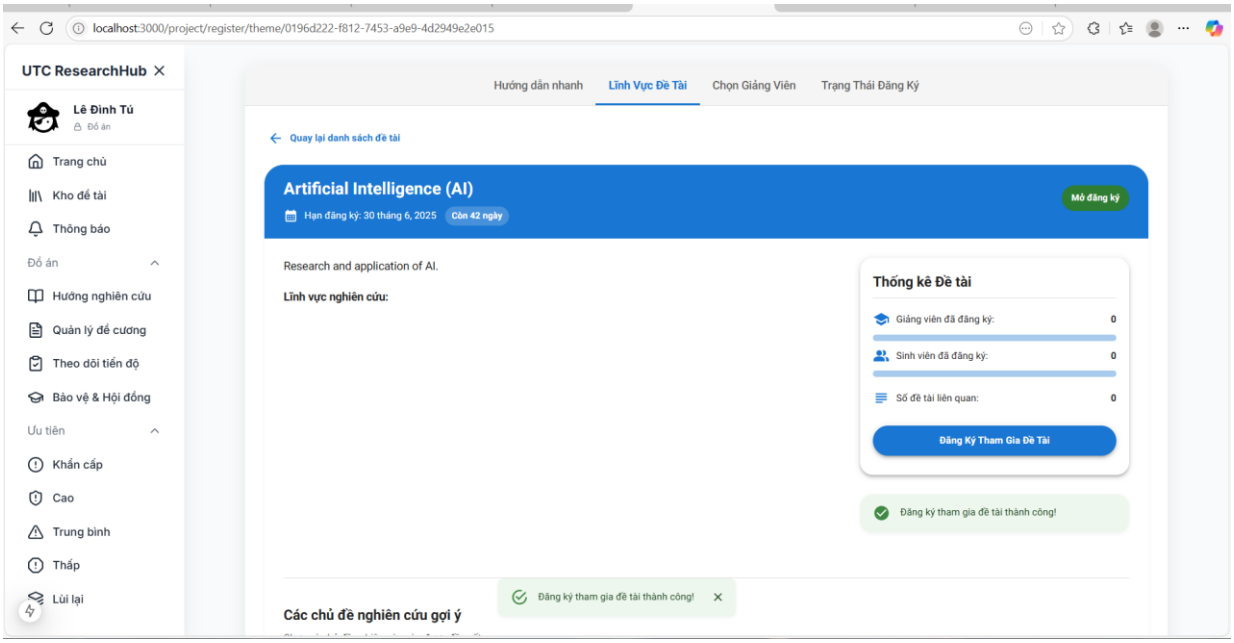

Figure 3.6. Preferences Detail Page (Student)

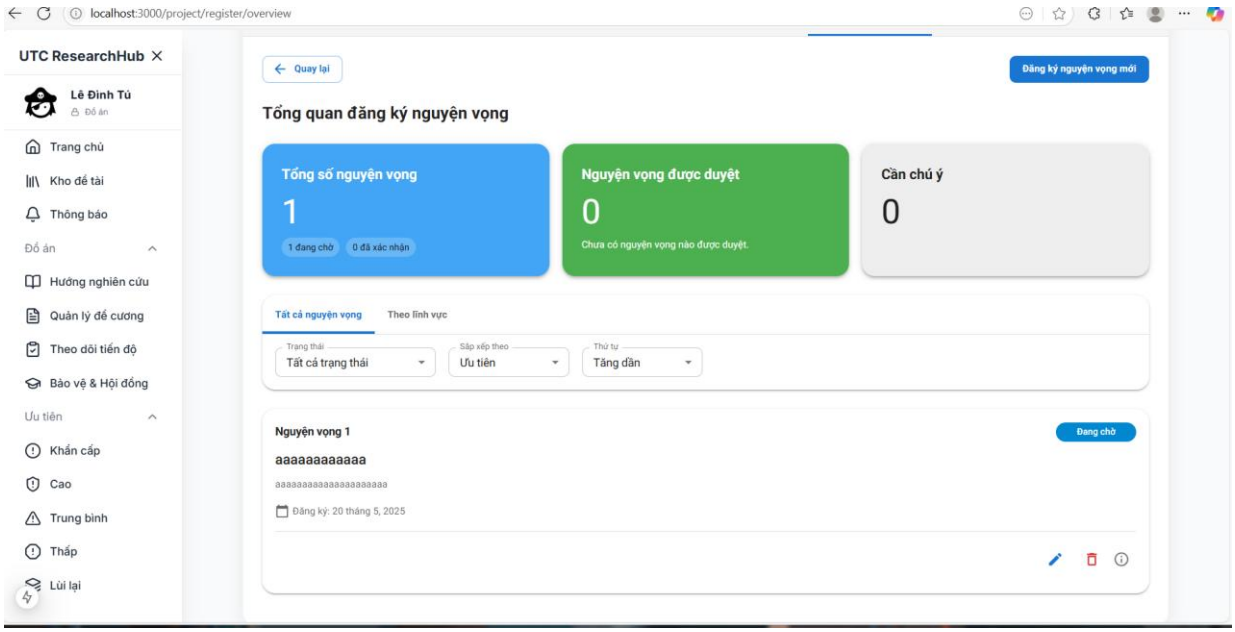

Figure 3.7. Preferences Overview Page (Student)

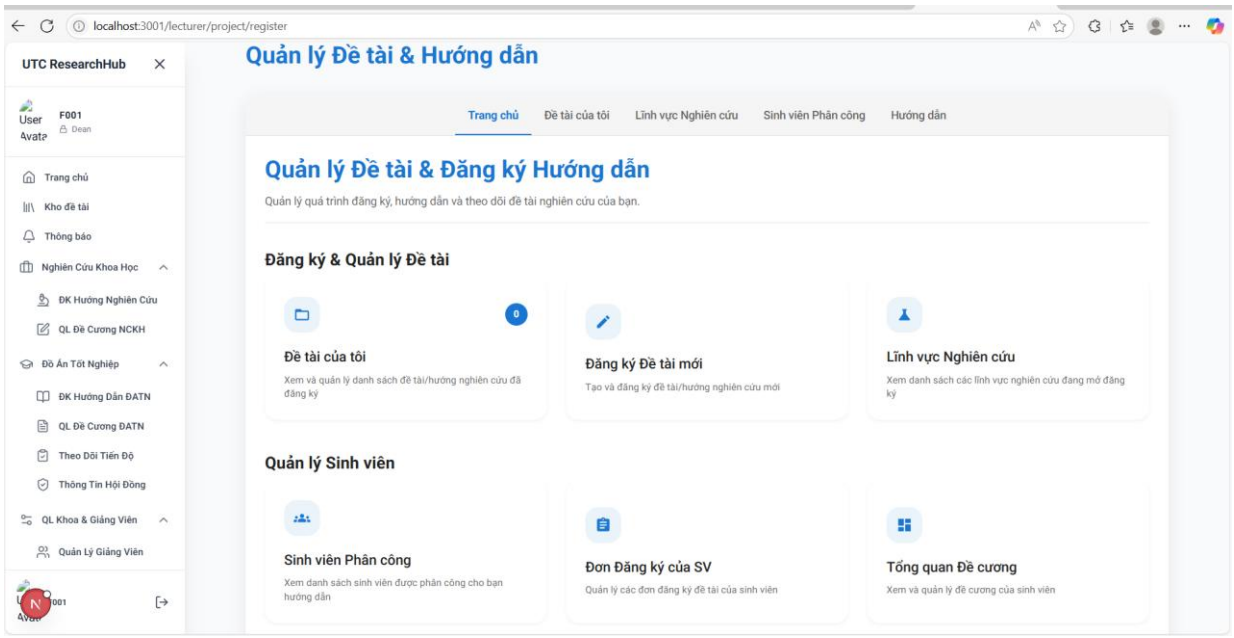

Figure 3.8. Preference Overview Page (Teacher)

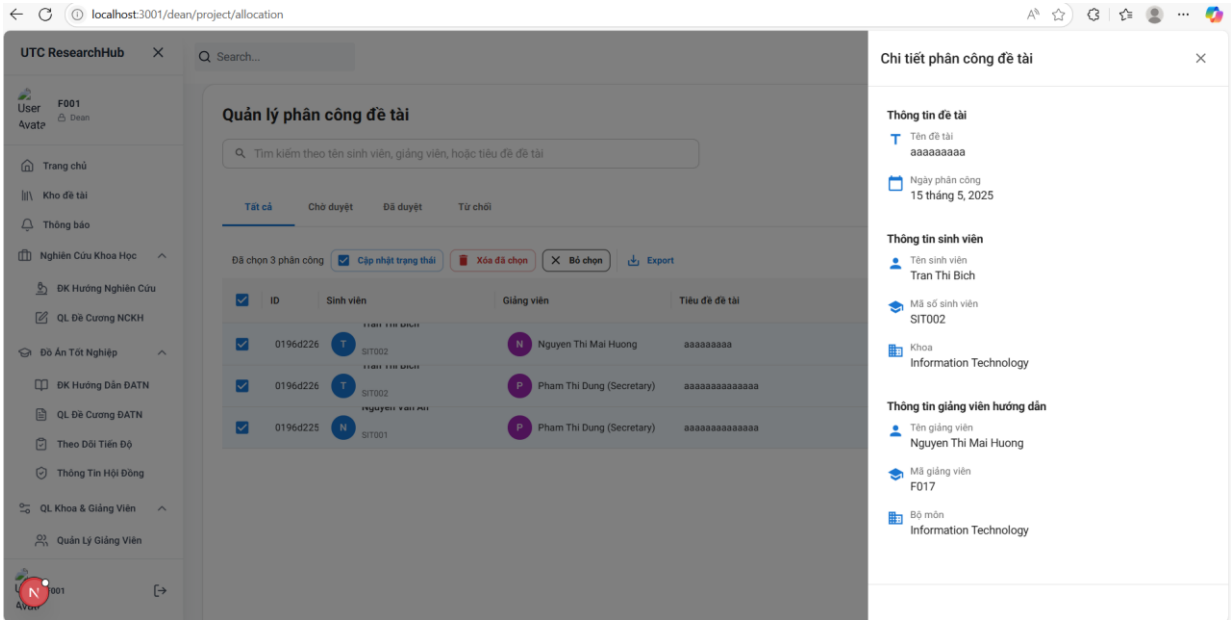

Figure 3.9. Preferences Management Page (Dean)

3.2.3 Proposal

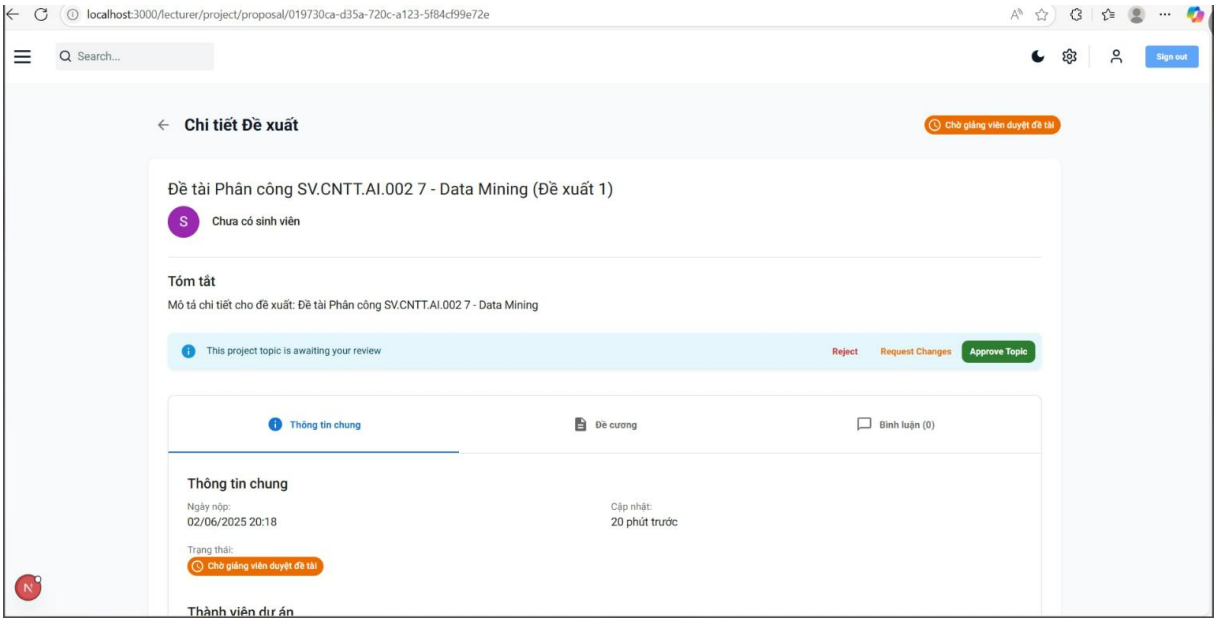

Figure 3.10. Proposal Review Page (Lecturer)

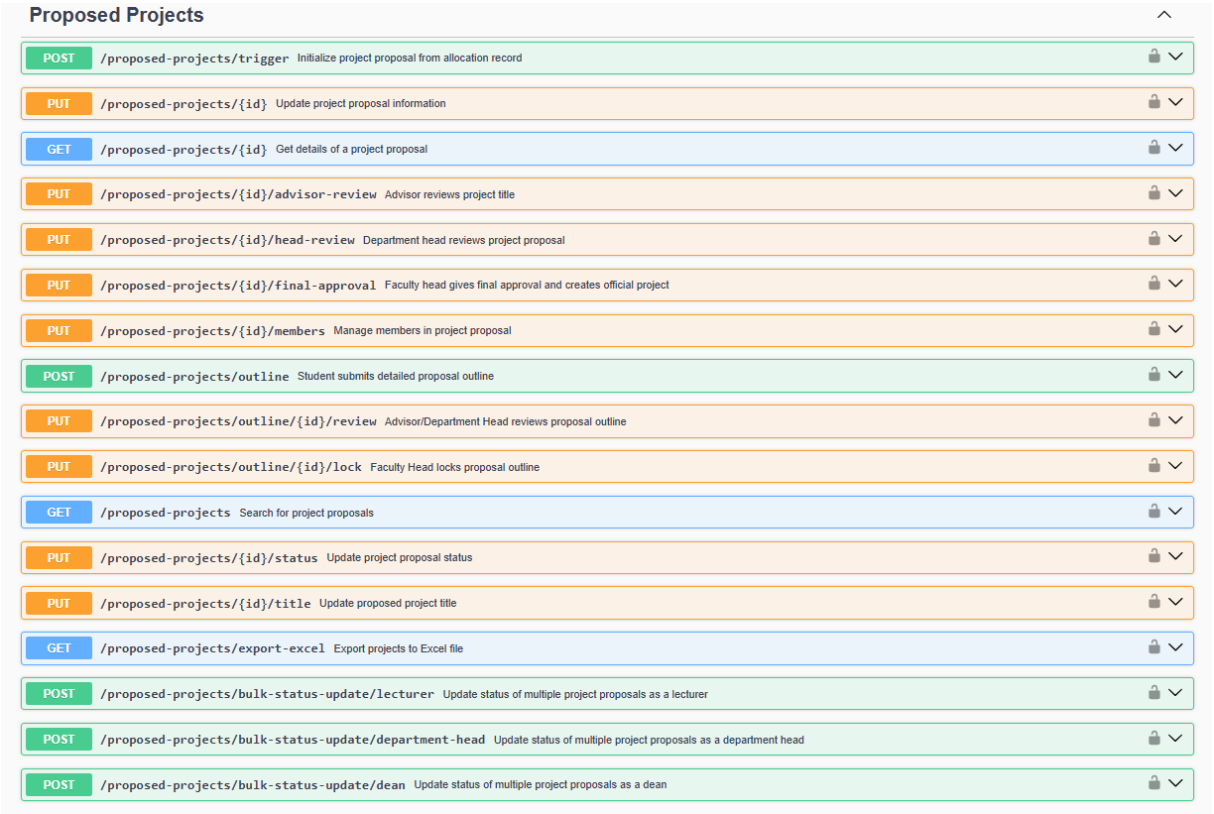

Figure 3.11. Proposal API

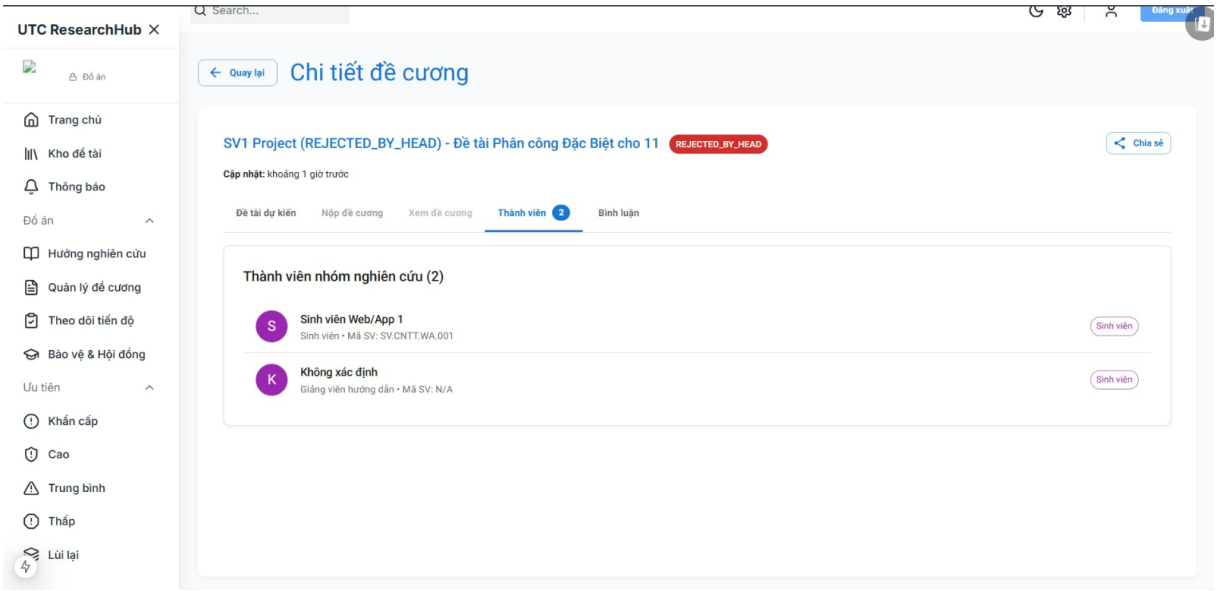

Figure 3.12. Proposal Page (Student)

3.2.4 Result

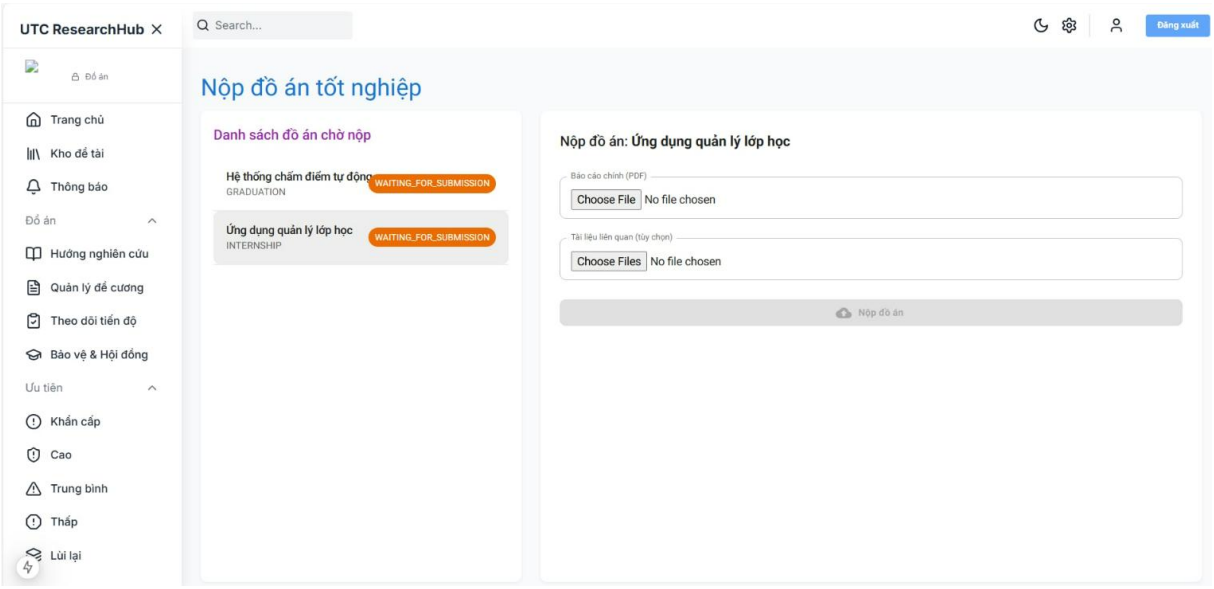

Figure 3.13. Submit Project Page (Student)

3.2.5 Evaluation

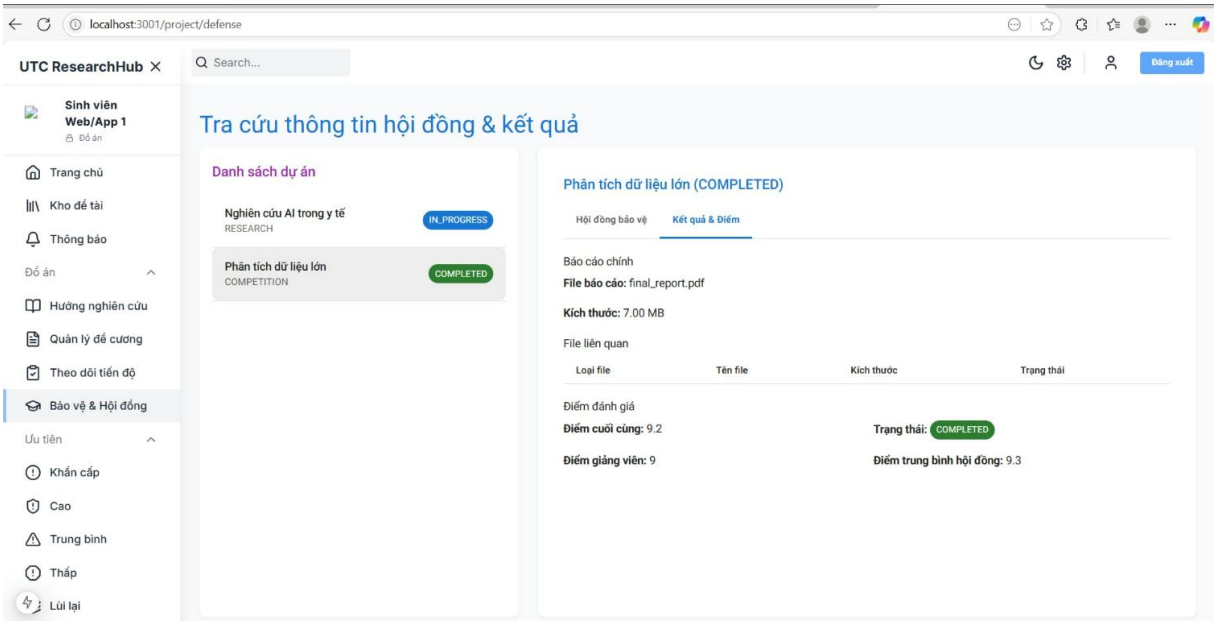

Figure 3.14. Evaluation Page (Student)

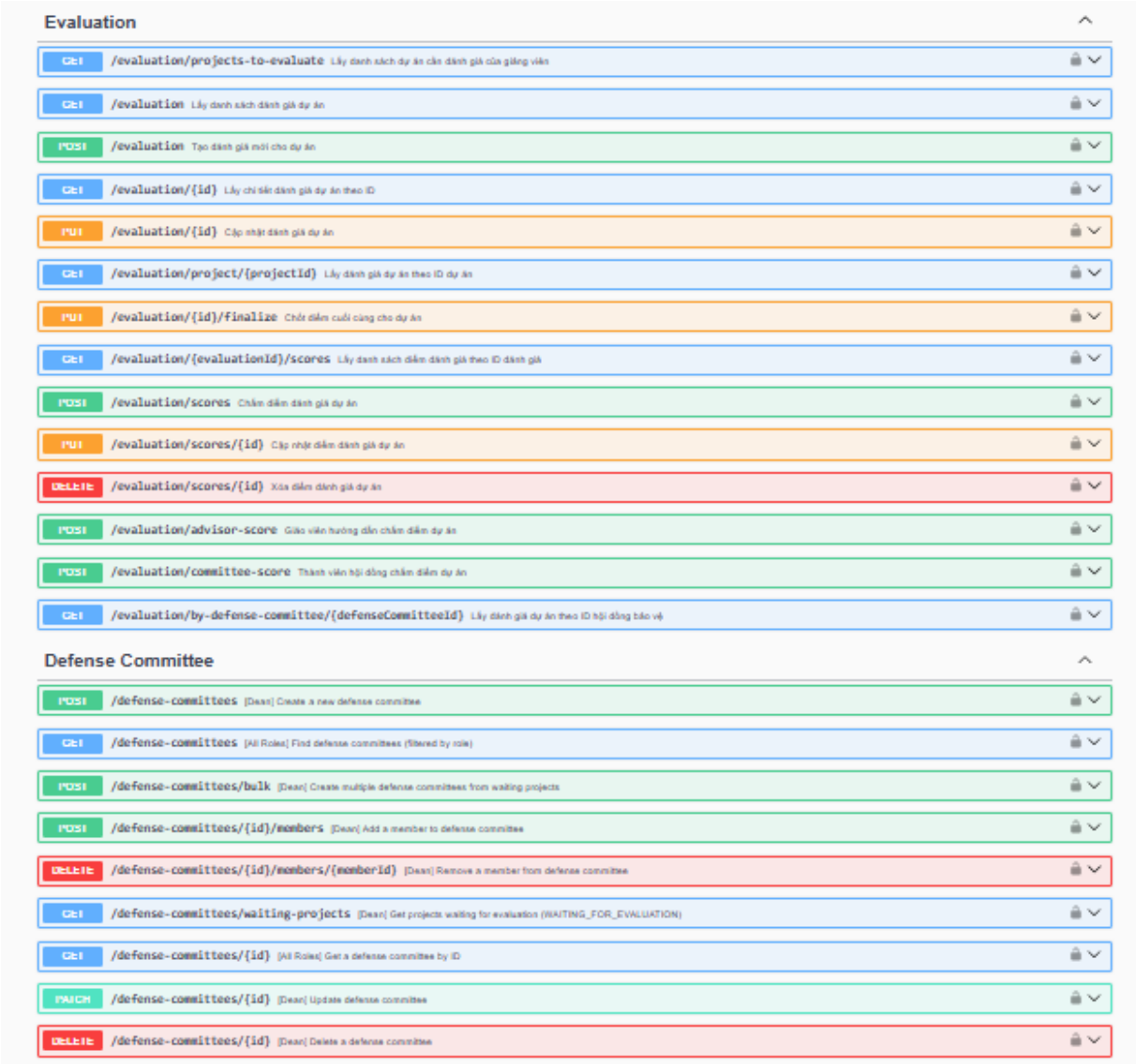

Figure 3.15. Evaluation API

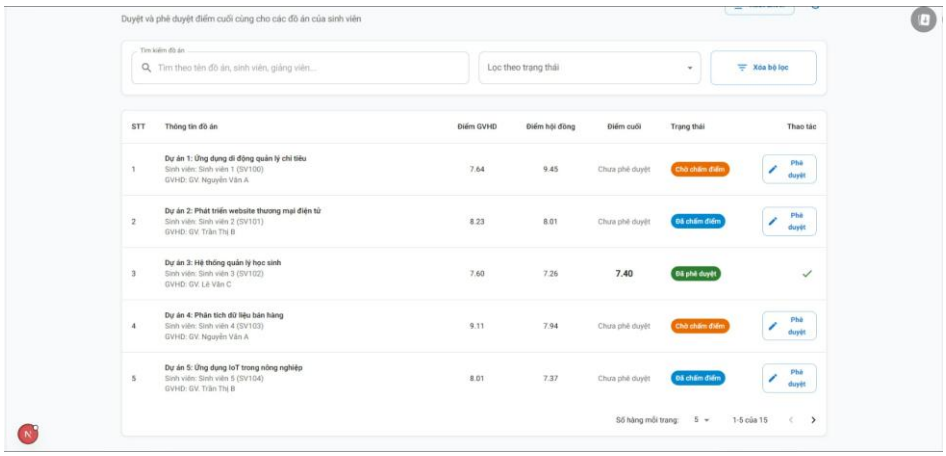

Figure 3.16. Evaluation Page (Department Head)

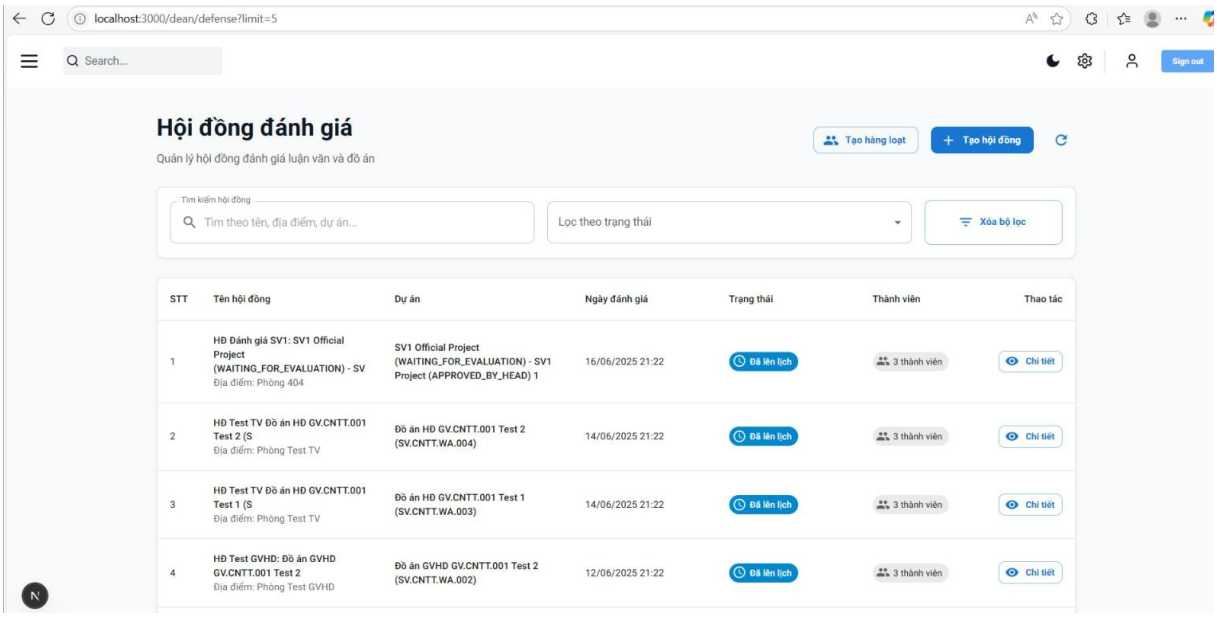

Figure 3.17. Defense Commit Management Page (Dean)

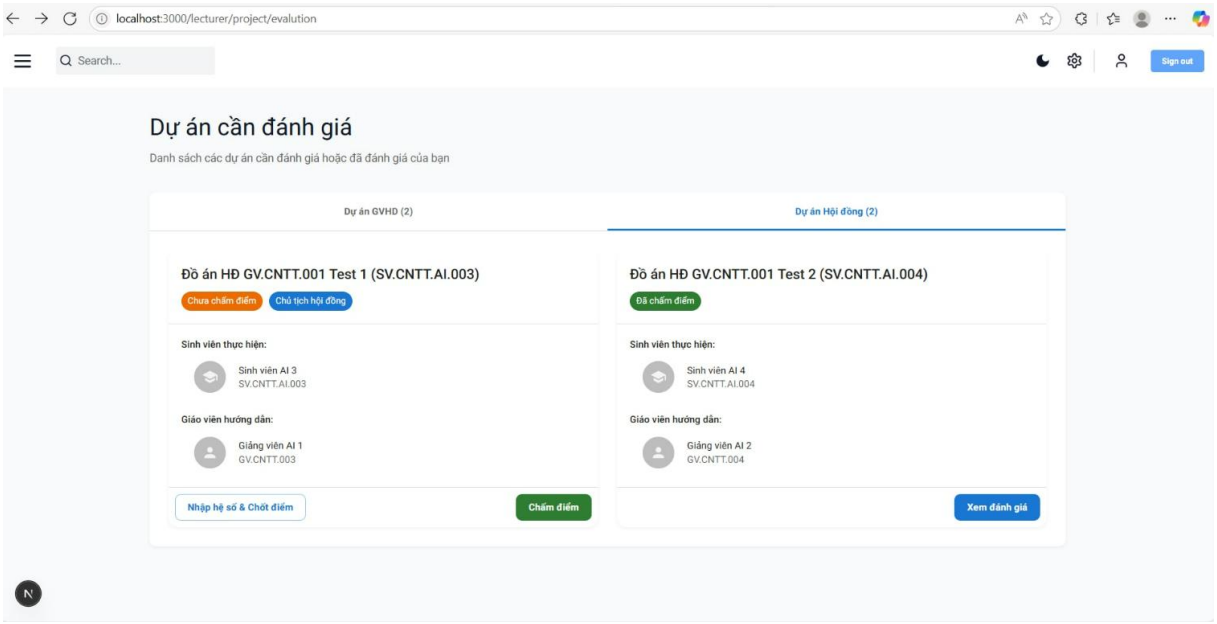

Figure 3.18. Evaluation Page (Defense Member)

3.2.6 Dashboard

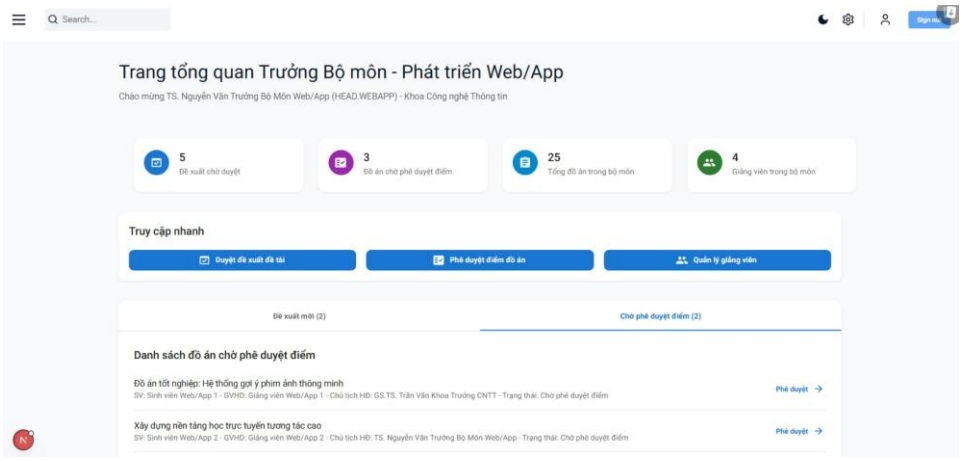

Figure 3.19. Dashboard (Department Head)

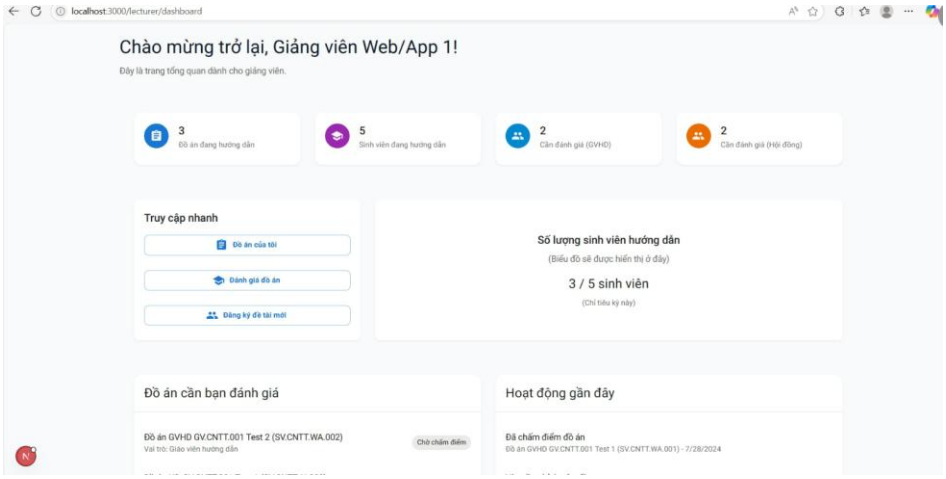

Figure 3.20. Dashboard (Lecturer)

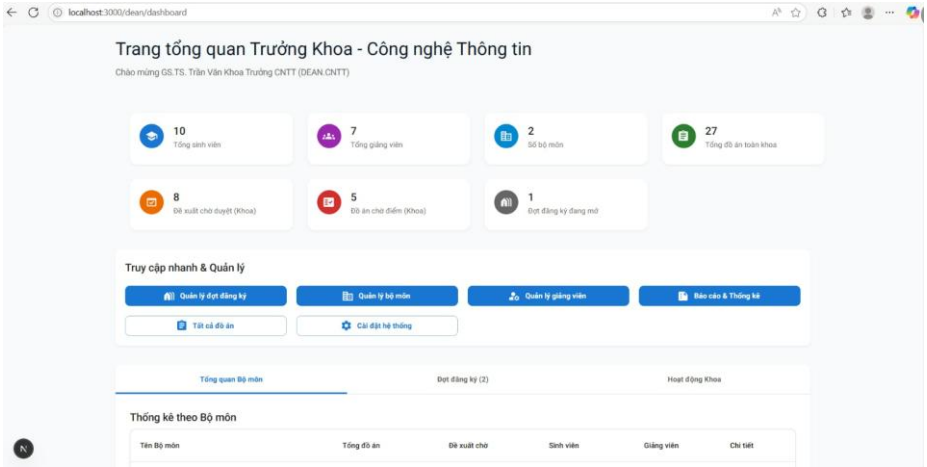

Figure 3.21. Dashboard (Dean)

CONCLUSION

This thesis management system, developed through this project, effectively addresses the critical needs involved in managing the thesis lifecycle for both students and faculty members in an academic environment. It supports a comprehensive range of core functionalities, including research direction registration, supervisor assignment, thesis outline management, feedback collection, grading, and defense committee coordination. By streamlining these processes, the system facilitates smooth, efficient workflows for topic registration, proposal submission and approval, assignment of supervisors and reviewers, and defense scheduling and management.

A key achievement in the project is the full-stack integration of TypeScript across both the frontend and backend layers. Utilizing NextJS for the user interface and NestJS for the server-side logic has significantly enhanced code quality, maintainability, and scalability. This technological choice reduces development time, minimizes potential errors, and lays a solid foundation for future expansion and feature additions. On the data management front, the employment of PostgreSQL as the relational database and Prisma as the ORM tool has ensured reliable data persistence and seamless database interactions. These technologies provide a robust backbone for complex data relationships inherent in academic project management.

Extensive testing using tools like Swagger and Postman has verified the stability, accuracy, and reliability of the system's key features. Functional testing across core workflows confirmed that the system meets design expectations and operates correctly under various scenarios.

However, the project also encountered several challenges and limitations. Time constraints restricted the scope of development, preventing the inclusion of advanced features such as real-time notifications and intelligent recommendation engines. Moreover, user adoption proved to be a gradual process, as many users were accustomed to traditional manual methods and required time to adjust to the new digital workflows. Additionally, although the system is currently capable of supporting the existing user base, further optimization is necessary to ensure scalability and maintain performance as the number of users and volume of data increase over time.

Looking forward, there are several promising directions for future development that could substantially enhance the system's functionality and user experience. Integrating artificial intelligence could revolutionize the way users interact with the system—for example, by providing automatic topic suggestions tailored to individual students' interests and academic backgrounds, evaluating thesis proposals to assist faculty in decision-making, and offering personalized recommendations throughout the research process. Implementing real-time notifications would significantly improve communication among students, supervisors, and administrative staff by alerting users promptly about outline approvals, feedback submissions, defense scheduling, and other critical events.

To expand accessibility, developing a mobile application would empower users to manage their tasks, submit documents, and track progress anytime and anywhere, accommodating the modern need for flexibility and remote access. Additionally, incorporating advanced reporting and analytics capabilities would provide

administrators with valuable insights into student progress, faculty workload, research trends, and performance metrics—supporting data-driven decision-making and continuous improvement in academic program management.

In summary, the thesis management system developed in this project successfully delivers a structured, automated, and collaborative platform that addresses essential academic workflows. By improving efficiency and streamlining communication between students and faculty, it enhances the overall research experience. The technological stack chosen establishes a scalable and maintainable framework ready for future innovations.

Continued development, including the integration of AI technologies, real-time communication features, mobile support, and enhanced analytics, combined with ongoing user feedback and iterative testing, will ensure that the system evolves to meet the dynamic and growing demands of academic institutions. This project lays a strong foundation upon which future enhancements can build to create a comprehensive, user-friendly, and intelligent research management environment.

REFERENCES

[1] NextJS Documentation. (n.d.). Retrieved April 2025, from https://nextjs.org/docs

[2] NestJS Documentation. (n.d.). Retrieved April 2025, from https://docs.nestjs.com/

[3] Prisma Documentation. (n.d.). Retrieved April 2025, from https://www.prisma.io/docs/

[4] PostgreSQL Global Development Group. (2023). PostgreSQL Documentation. Retrieved April 2025, from https://www.postgresql.org/docs/

[5] Jsonwebtoken. (n.d.). jsonwebtoken. Retrieved April 2025, from https://github.com/auth0/node-jsonwebtoken

[6] Zustand Documentation. (n.d.). Retrieved April 2025, from https://docs.pmnd.rs/zustand/getting-started/introduction

[7] Material-UI. (n.d.). Material-UI: React components for faster and easier web development. Retrieved April 2025, from https://mui.com/

[8] AppsDevPro. (n.d.). React vs NextJS: A Complete Comparison. Retrieved June 2025, from https://www.appsdevpro.com/blog/react-vs-next-js/

[9] Gilad, Y. (n.d.). NestJS Request Lifecycle [Slide presentation]. Retrieved June 2025, from https://slides.com/yariv-gilad/nest-js-request-lifecycle

[10] NearForm. (2023). Prisma ORM. Retrieved June 2025, from https://nearform.com/digital-community/prisma-orm/

[11] Zod. (n.d.). Zod: TypeScript-first schema declaration and validation. Retrieved June 2025, from https://github.com/colinhacks/zod

[12] ExcelJS. (n.d.). ExcelJS: Excel spreadsheet library for Node.js. Retrieved June 2025, from https://github.com/exceljs/exceljs

[13] bcrypt. (n.d.). bcrypt: A library to help hash passwords. Retrieved June 2025, from https://github.com/kelektiv/node.bcrypt.js

[14] UUID. (n.d.). uuid: Simple, fast generation of RFC4122 UUIDs. Retrieved June 2025, from https://github.com/uuidjs/uuid