UTM

UNIVERSITI TEKNOLOGI MALAYSIA

SECP3106 APPLICATION DEVELOPMENT (WBL)

SEMESTER 2 2024/2025

INDIVIDUAL TEST:

FYP1 SYSTEM

SOFTWARE DESIGN DOCUMENTATION (SDD)

SECTION: 02

LECTURER: DR. AHMAD NAJMI BIN AMERHAIDER NUAR

NAME : KOH SU XUAN

MATRICS NO. : A22EC0060

Table of Contents

1.0 System Architectural Design

1.1 Architecture Style and Rationale

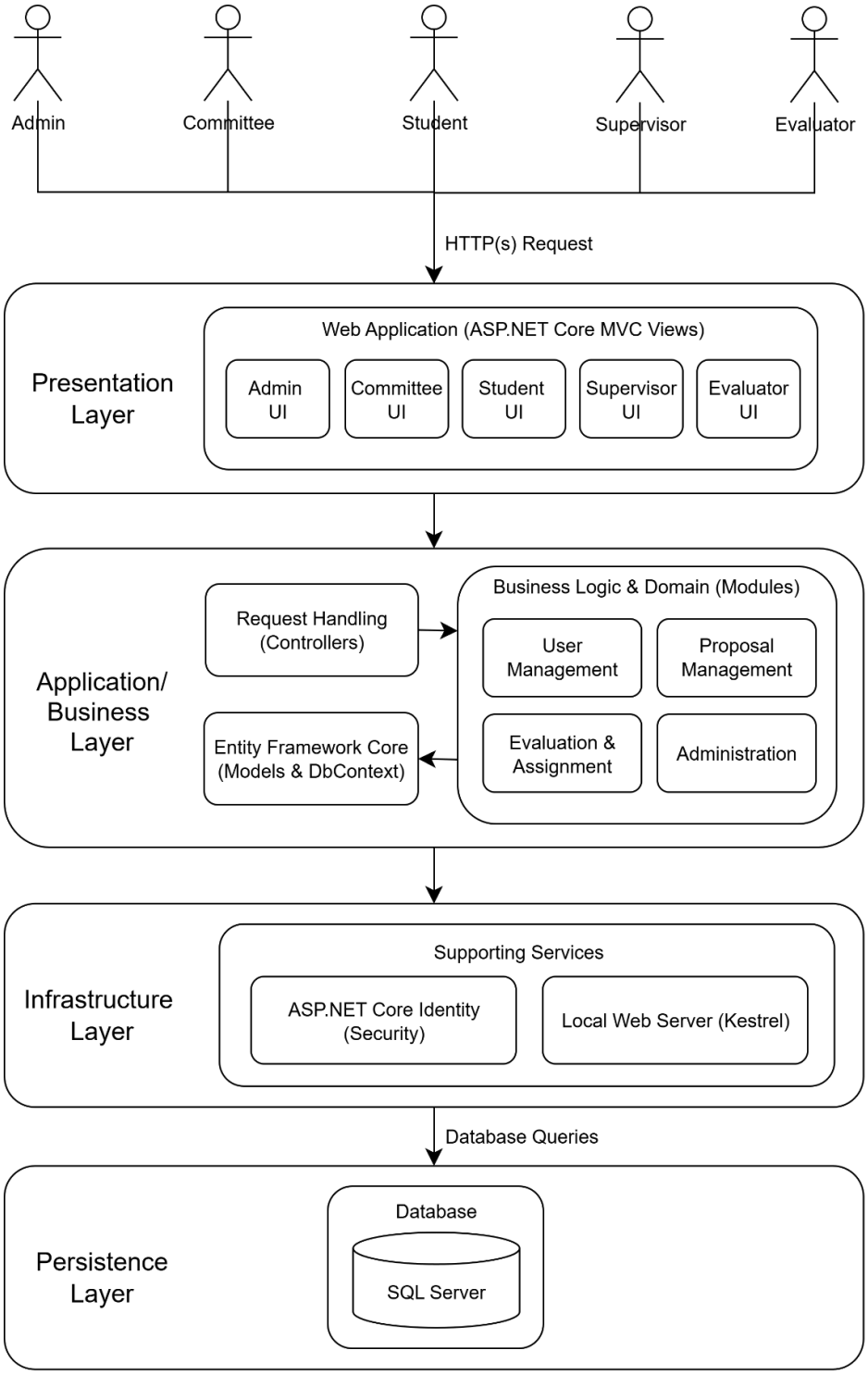

Figure 1.1.1: Architecture Diagram of <FYP1 System>

The system architecture for the FYP1 System is structured as a multi-layered design that ensures a clear separation of concerns, scalability and maintainability, as illustrated in Figure 1.1.1. The core architectural style is a classic Layered Architecture deeply integrated with the Model-View-Controller (MVC) pattern. This approach is ideal for developing a robust, self-contained web application that serves various user roles from a single, unified codebase.

This architecture divides the application into four distinct layers, which are Presentation, Application/Business, Infrastructure and Persistence.

The Presentation Layer is the client-facing layer responsible for all user interaction. It consists of a single Web Application built with ASP.NET Core MVC Views. This layer dynamically renders role-specific User Interfaces (UIs) on the server, ensuring that users such as Admins, Students and Supervisors only interact with the system components relevant to their permissions. This server-rendered approach provides a tightly coupled and secure user experience.

The Application/Business Layer, built on the ASP.NET Core framework, serves as the central hub of the system. It processes all business logic and orchestrates data flow. This layer is further subdivided into three main components.

- **Request Handling (Controllers)**
  This acts as the entry point for all HTTP requests from the Presentation Layer, routing them to the appropriate business logic. It represents the "Controller" in the MVC pattern.
- **Business Logic & Domain (Modules)**
  This is the core of the application, organized into distinct modules such as User Management, Proposal Management, Evaluation & Assignment and Administration. This modular design enhances code organization and allows for easier maintenance.
- **Entity Framework Core (Models & DbContext)**
  This component represents the "Model" in MVC. It encapsulates the data

structures and contains the logic for interacting with the database via the Object-Relational Mapper (ORM), ensuring data integrity.

The Infrastructure Layer contains all supporting technologies required for the system to operate. For this project, it consists of ASP.NET Core Identity, which provides critical security services for authentication and authorization and the Local Web Server (Kestrel), which hosts the application during development.

Finally, the Persistence Layer is responsible for the physical storage of data. The system utilizes SQL Server as its relational database, which is accessed via Entity Framework Core through structured Database Queries.

Adopting this layered architecture provides several key advantages for this web application. It enforces a strong separation of concerns, allowing different parts of the system to be developed and tested independently. The modular design of the business layer promotes code reusability and maintainability. This design effectively meets the immediate requirements of a multi-user academic platform while being well-prepared for future functional expansion.

1.2 Component Model

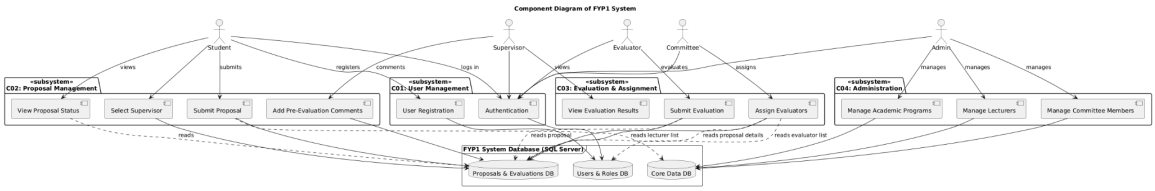

Figure 1.2.1: Component Diagram of <FYP1 System>

Figure 1.2.1 illustrates the component model, which represents the high-level architecture of the FYP1 System. The system is logically divided into four main subsystems, each consisting of components that handle specific functionalities and interact with the central database. The system is designed to serve five distinct user roles, who are Admin, Student, Supervisor, Evaluator and Committee, with access to components based on their defined permissions.

The User Management subsystem is the foundational security component, responsible for user identity and access control. It includes the Authentication component, which all roles use to log in and the User Registration component, which is used exclusively by Students. Both components interact directly with the Users & Roles DB to verify credentials and create user accounts.

The Proposal Management subsystem handles the initial phase of the FYP workflow. Students interact with its components to Select Supervisor, Submit Proposal and View Proposal Status. Supervisors engage with this subsystem to Add Pre-Evaluation Comments. These components primarily interact with the Proposals & Evaluations DB for storing and retrieving proposal data, while also reading from the Core Data DB to fetch lists of available supervisors.

The Evaluation & Assignment subsystem manages the formal review process. Committee members use the Assign Evaluators component to allocate proposals for review. This component reads proposal details and evaluator lists from the databases. Evaluators use the Submit Evaluation component to provide their formal assessment, writing the results to the Proposals & Evaluations DB. Supervisors and Students can then view the outcomes via the View Evaluation Results component.

The Administration subsystem provides top-level system configuration, accessible only to the Admin role. It consists of three distinct management components: Manage Academic Programs, Manage Lecturers and Manage Committee Members. All three components write directly to the Core Data DB, which houses the foundational information that the rest of the system depends on.

Each component in the model is linked by clear, labeled relationships that show the interaction between user roles, system functionality and data storage. Dotted lines indicate dependency or read-only access, while solid lines represent direct actions or data manipulation. This structured model enhances clarity in the system design and helps ensure maintainability and strict role-based control.

2.0 Detailed Description of Components

2.1 Complete Package Diagram

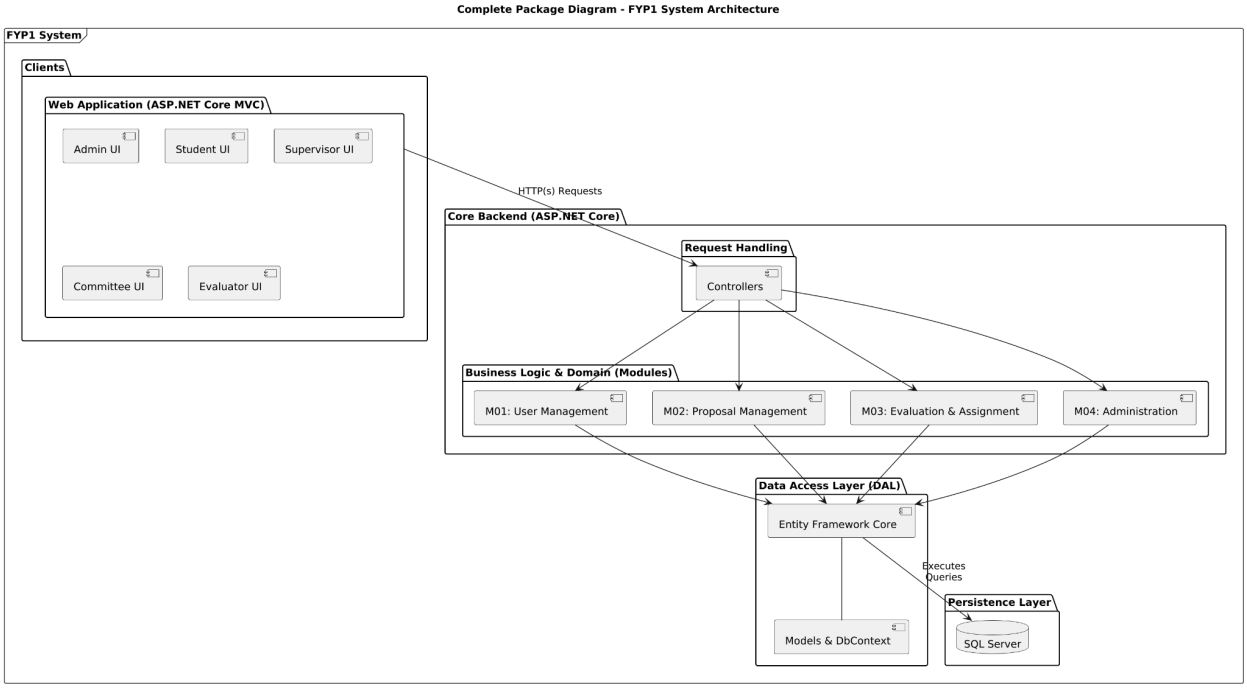

Figure 2.1.1: Complete Package Diagram for <FYP1 System>

Figure 2.1.1 shows the complete package diagram for FYP1 System. Overall, it consists of 4 subsystems. The details of each subsystem are covered in the following section.

2.2 Detailed Description

P001	<User Management Module> Subsystem	
SD01	Login/Logout	It allows users (students, lecturers, admins) to authenticate into the system using credentials and securely logout to end their session.
SD02	Register Student	It allows new students to self-register with validation and automatic role assignment.
SD03	Manage Lecturer	It allows admins to perform full CRUD operations on lecturer accounts including creation, editing, deletion and program assignments.
SD04	Assign Committee Role	It allows admins to assign or remove committee roles from lecturers to control system permissions and committee membership.

P002	<Proposal Management Module> Subsystem	
SD05	Create Proposal	It allows students to create new FYP proposals with project type selection, detailed information and required document uploads.
SD06	Select Supervisor	It allows students to select a supervisor from eligible lecturers matching their project domain with committee approval workflow.
SD07	Edit Proposal	It allows students to modify their proposal details and documents before supervisor approval is finalized.
SD08	Delete Proposal	It allows students to delete their proposals when not yet approved, with confirmation and file cleanup.

P003	<Evaluation & Assignment Module> Subsystem	
SD09	Assign Domain	It allows committee members to assign research or development domains to lecturers to enable supervisor and evaluator roles.

SD10	Approve Supervisor Selection	It allows committee members to review and approve or reject student supervisor selections with proper workflow management.
SD11	Assign Evaluators	It allows committee members to assign two different evaluators to approved proposals with validation and role management.
SD12	Supervise Proposals	It allows supervisors to view, filter and provide comments on their assigned student proposals with document access.
SD13	Evaluate Proposals	It allows evaluators to review assigned proposals and provide evaluation status with detailed feedback.

P004	<Administration Module> Subsystem	
SD14	View System Dashboard	It allows admins to view comprehensive system statistics and overviews with key performance indicators and navigation interfaces.
SD15	Manage Academic Program	It allows admins to perform full CRUD operations on academic programs.

2.2.1 P001: <User Management Module> Subsystem

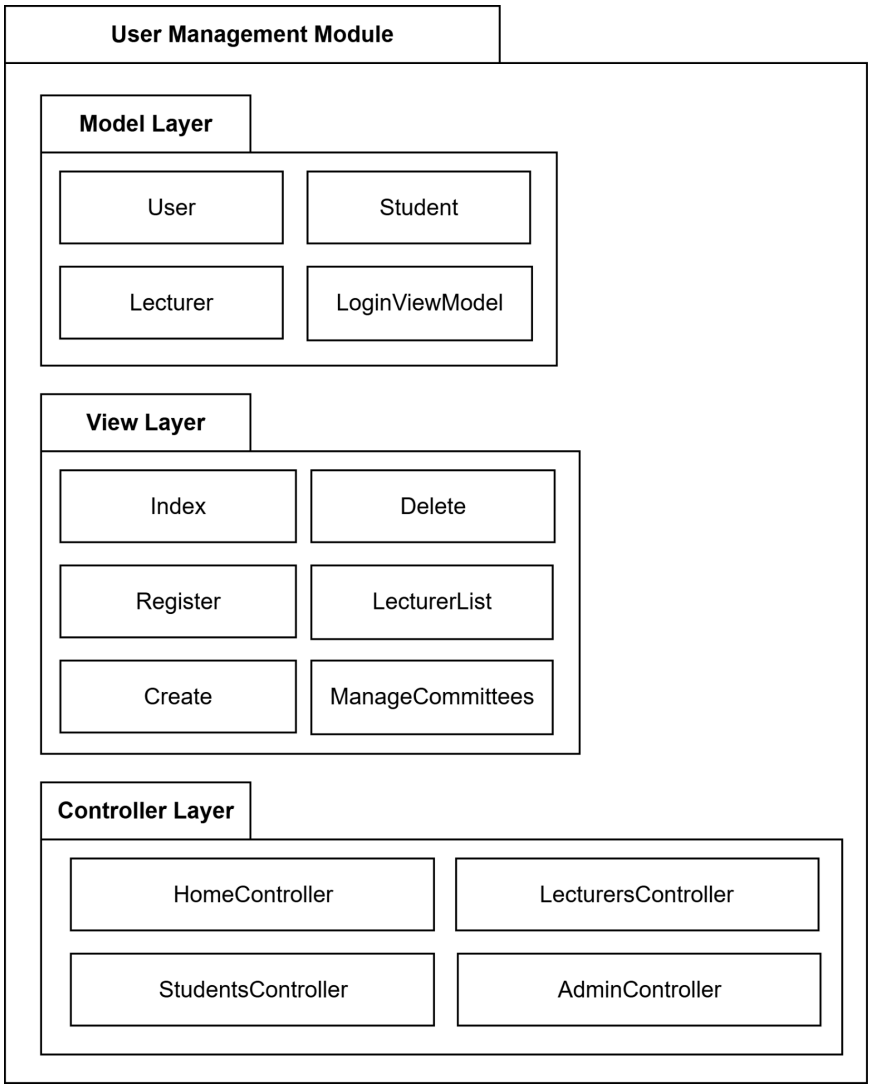

Figure 2.2.1.1: Package Diagram for <User Management Module> Subsystem

2.2.1.1 Class Diagram

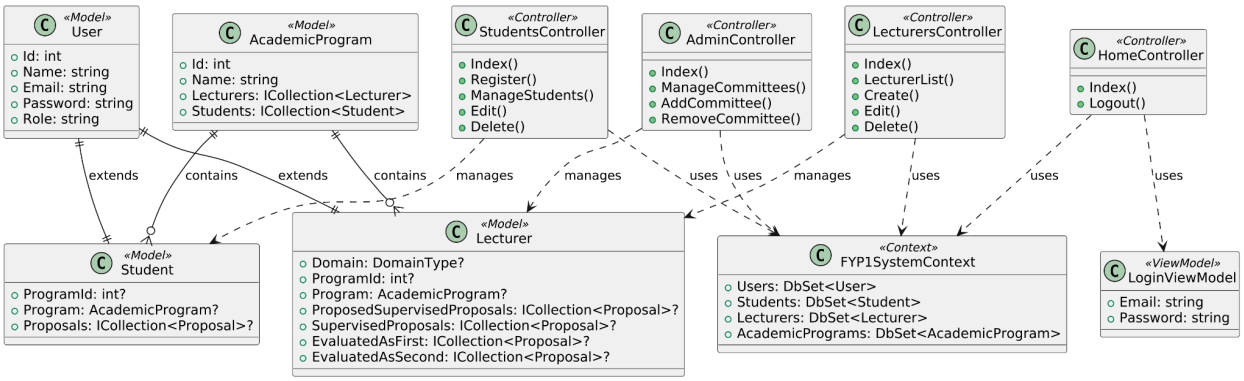

Figure 2.2.1.1.1: Class Diagram for <User Management Module> Subsystem

Entity Name	HomeController
Method Name	Index()
Input	Email, Password
Output	Authenticated user session and redirection to appropriate dashboard or error message
Algorithm	1. Start. 2. System receives email and password from the login form. 3. Validate input fields to ensure they are not empty. 4. Query the Users database table to find user with matching email and password. 4.1 If authentication is successful: 4.1.1 Create a secure session for the user. 4.1.2 Parse user roles from Role field. 4.1.3 Redirect user based on primary role: - Admin → Admin dashboard - Lecturer → Lecturers dashboard - Student → Students dashboard. 4.2 If authentication fails: 4.2.1 Return user to login page. 4.2.2 Display "Invalid email or password" error message. 5. End.

Entity Name	HomeController

Method Name	Logout()
Input	None
Output	Cleared session and redirection to login page
Algorithm	1. Start. 2. Clear all session data. 3. Redirect user to login page. 4. End.

Entity Name	StudentsController
Method Name	Index()
Input	None
Output	Student dashboard with personal information and proposal status
Algorithm	1. Start. 2. Check if user has "Student" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Get UserId from session. 4. Query Students table with Include for Program and Proposals. 5. If student not found, return NotFound. 6. Get first proposal from student's proposals collection. 7. Set ViewBag.Proposal for view usage. 8. Return student dashboard view with student data. 9. End.

Entity Name	StudentsController
Method Name	Register()
Input	Name, Email, Password, ProgramId
Output	Student registration form or successful registration redirect
Algorithm	1. Start. 2. If GET request: 2.1 Query AcademicPrograms table to get all programs. 2.2 Create SelectList for programs dropdown. 2.3 Set ViewBag.Programs with SelectList. 2.4 Return registration view.

	3. If POST request: 3.1 Validate ModelState for required fields. 3.2 Check if email already exists in Students table. 3.3 If email exists, add ModelError and return view. 3.4 If email is unique: 3.4.1 Set student.Role = "Student". 3.4.2 Add student to Students DbSet. 3.4.3 Save changes to database. 3.4.4 Redirect to Home Index. 4. End.

Entity Name	StudentsController
Method Name	ManageStudents()
Input	None
Output	List of students in committee member's program
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Lecturers Index. 3. Get UserId from session. 4. Find committee lecturer in Lecturers table. 5. If lecturer not found or ProgramId is null: 5.1 Redirect to Lecturers Index. 6. Query Students table where ProgramId matches committee's program. 7. Include Program navigation property. 8. Return ManageStudents view with filtered students list. 9. End.

Entity Name	StudentsController
Method Name	Edit()
Input	Student ID and Student object
Output	Student edit form or updated student record
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Lecturers Index.

	3. If GET request: 3.1 Query Students table to find student by ID. 3.2 If student not found, return NotFound. 3.3 Query AcademicPrograms for dropdown. 3.4 Create SelectList with current ProgramId selected. 3.5 Return Edit view with student data. 4. If POST request: 4.1 Validate that ID matches student.Id. 4.2 Validate ModelState. 4.3 If valid, update student and save changes. 4.4 Redirect to Index. 5. End.

Entity Name	StudentsController
Method Name	Delete()
Input	Student ID
Output	Student deletion confirmation or completed deletion
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Lecturers Index. 3. If GET request: 3.1 Query Students table with Include Program. 3.2 Find student by ID. 3.3 If student not found, return NotFound. 3.4 Return Delete confirmation view with student data. 4. If POST request (DeleteConfirmed): 4.1 Find student by ID in Students table. 4.2 If student found, remove from DbSet and save changes. 4.3 Redirect to Index. 5. End.

Entity Name	LecturersController
Method Name	Index()
Input	None
Output	Lecturer dashboard with supervised proposals and pending approvals

Algorithm	1. Start. 2. Check if user has "Lecturer" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Get UserId from session. 4. Query Lecturers table with Include Program and SupervisedProposals. 5. Include Student navigation in SupervisedProposals. 6. Find lecturer by UserId. 7. If lecturer not found, return NotFound. 8. Query Proposals for pending supervisor approvals. 9. Include Student and Supervisor in pending proposals query. 10. Filter by SupervisorSelectionPendingApproval status. 11. Set ViewBag.ProposalsPendingApproval. 12. Return lecturer dashboard view. 13. End.

Entity Name	LecturersController
Method Name	LecturerList()
Input	None
Output	Complete list of lecturers for admin management
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Query Lecturers table with Include Program. 4. Get all lecturers. 5. Return LecturerList view with lecturers data. 6. End.

Entity Name	LecturersController
Method Name	Create()
Input	Lecturer object
Output	Lecturer creation form or created lecturer record
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index.

	3. If GET request: 3.1 Query AcademicPrograms table for dropdown. 3.2 Create SelectList for programs. 3.3 Set ViewBag.Programs. 3.4 Return Create view. 4. If POST request: 4.1 Validate ModelState. 4.2 If valid: 4.2.1 Set lecturer.Role = "Lecturer". 4.2.2 Add lecturer to Lecturers DbSet. 4.2.3 Save changes to database. 4.2.4 Redirect to LecturerList. 4.3 If invalid, reload dropdown and return view with errors. 5. End.

Entity Name	LecturersController
Method Name	Edit()
Input	Lecturer ID and Lecturer object
Output	Lecturer edit form or updated lecturer record
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. If GET request: 3.1 Query Lecturers table to find lecturer by ID. 3.2 If lecturer not found, return NotFound. 3.3 Query AcademicPrograms for dropdown. 3.4 Create SelectList with current ProgramId selected. 3.5 Return Edit view with lecturer data. 4. If POST request: 4.1 Validate that ID matches lecturer.Id. 4.2 Validate ModelState. 4.3 If valid: 4.3.1 Set lecturer.Role = "Lecturer". 4.3.2 Update lecturer in context. 4.3.3 Save changes to database. 4.3.4 Redirect to LecturerList. 4.4 If invalid, reload dropdown and return view with errors. 5. End.

Entity Name	LecturersController
Method Name	Delete()
Input	Lecturer ID
Output	Lecturer deletion confirmation or completed deletion
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. If GET request: 3.1 Query Lecturers table with Include Program. 3.2 Find lecturer by ID. 3.3 If lecturer not found, return NotFound. 3.4 Return Delete confirmation view with lecturer data. 4. If POST request (DeleteConfirmed): 4.1 Find lecturer by ID in Lecturers table. 4.2 If lecturer found, remove from DbSet and save changes. 4.3 Redirect to LecturerList. 5. End.

Entity Name	AdminController
Method Name	Index()
Input	None
Output	Admin dashboard with system statistics
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Count total records in AcademicPrograms table. 4. Count total records in Lecturers table. 5. Query all lecturers and filter by Committee role. 6. Count lecturers with Committee role in their Role field. 7. Set ViewBag values: 7.1 ViewBag.TotalPrograms 7.2 ViewBag.TotalLecturers 7.3 ViewBag.TotalCommittees. 8. Return admin dashboard view.

	9. End.

Entity Name	AdminController
Method Name	ManageCommittees()
Input	None
Output	Committee management interface with lecturer list
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Query Lecturers table with Include Program. 4. Get all lecturers. 5. Return ManageCommittees view with lecturers list. 6. End.

Entity Name	AdminController
Method Name	AddCommittee()
Input	lecturerId
Output	Updated lecturer with Committee role added
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Find lecturer by lecturerId in Lecturers table. 4. If lecturer found: 4.1 Call lecturer.AddRole("Committee") method. 4.2 Update lecturer in context. 4.3 Save changes to database. 5. Redirect to ManageCommittees. 6. End.

Entity Name	AdminController
Method Name	RemoveCommittee()
Input	lecturerId

Output	Updated lecturer with Committee role removed
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Find lecturer by lecturerId in Lecturers table. 4. If lecturer found: 4.1 Call lecturer.RemoveRole("Committee") method. 4.2 Update lecturer in context. 4.3 Save changes to database. 5. Redirect to ManageCommittees. 6. End.

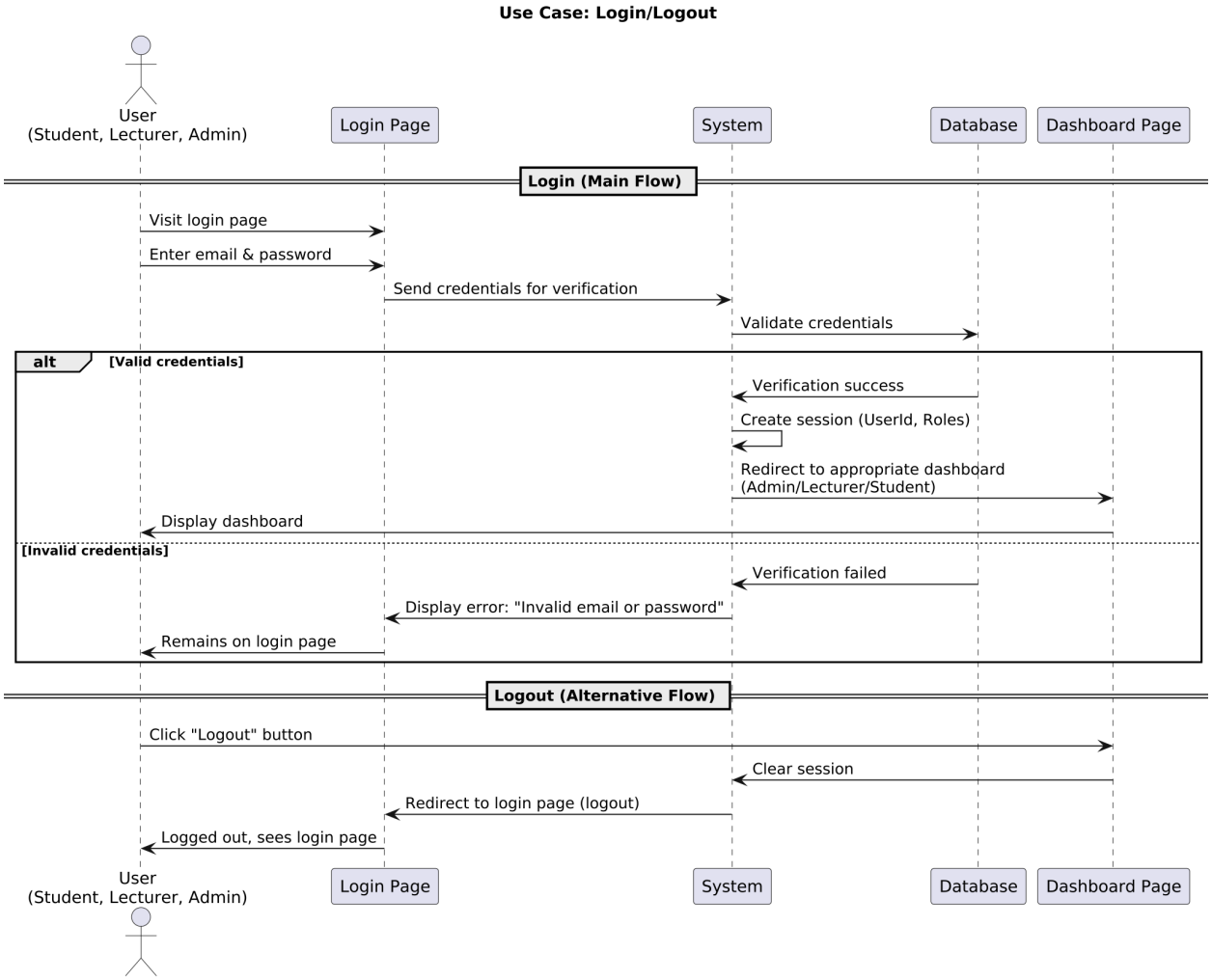

Figure 2.2.1.1.2: Sequence Diagram for <Login/Logout>

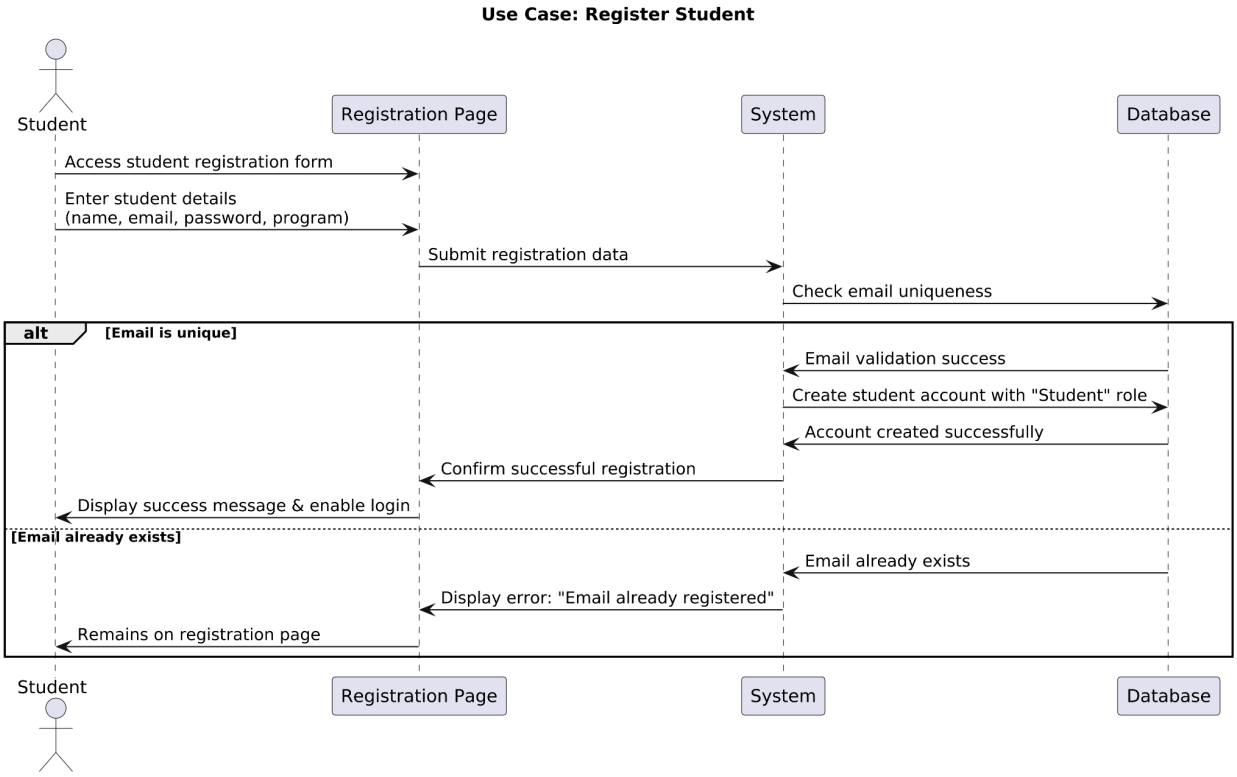

Figure 2.2.1.1.3: Sequence Diagram for <Register Student>

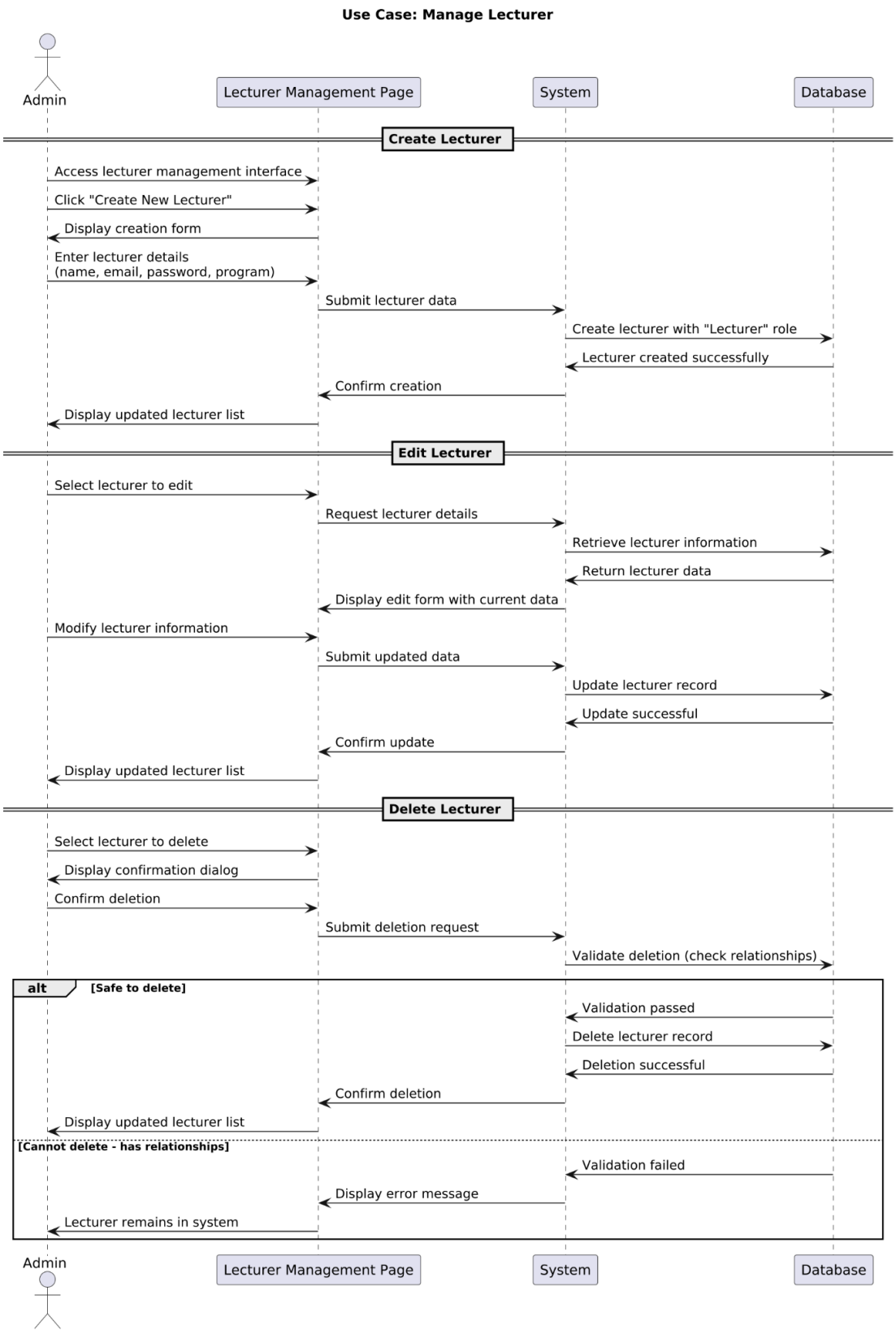

Figure 2.2.1.1.4: Sequence Diagram for <Manage Lecturer>

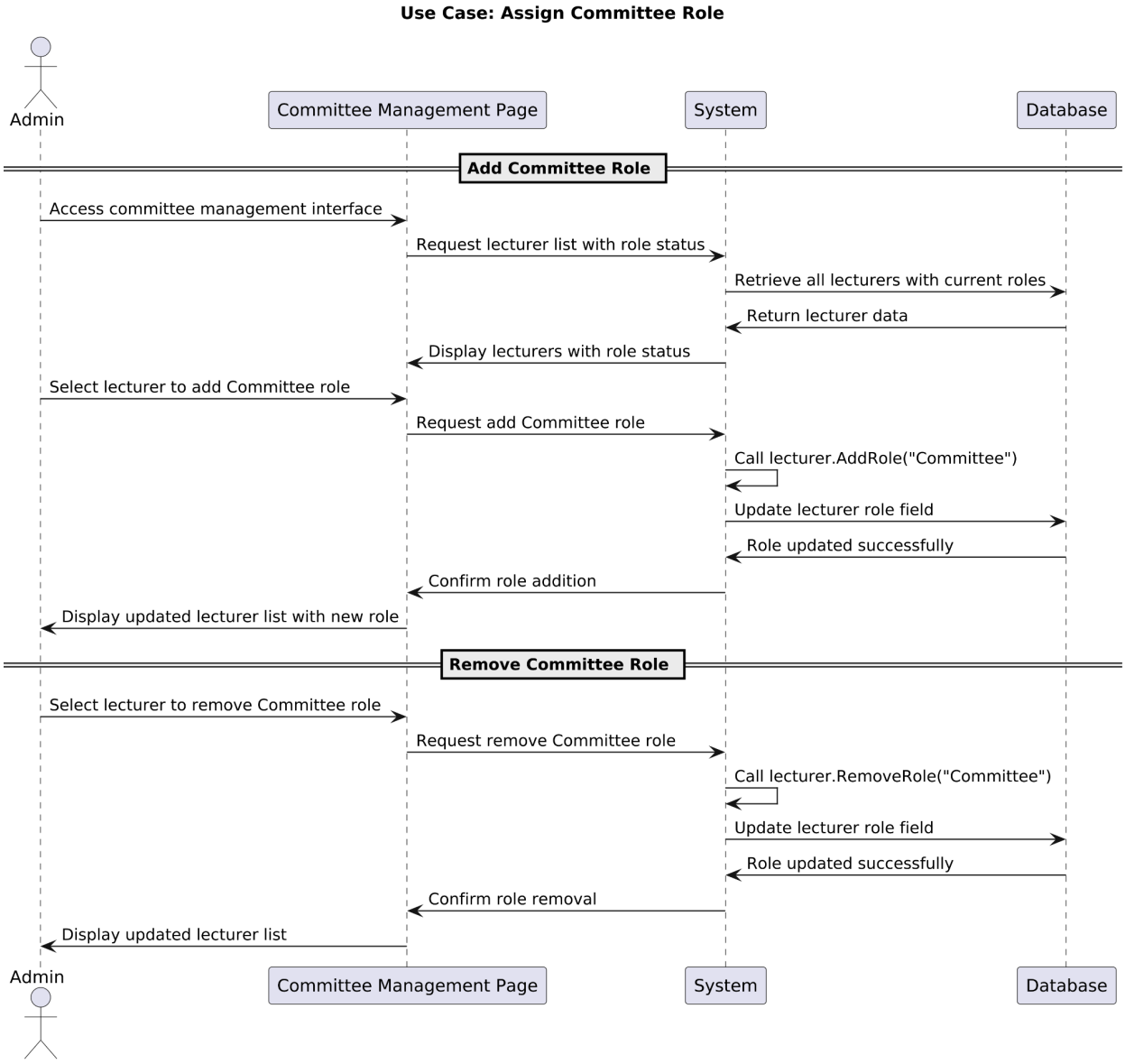

Figure 2.2.1.1.5: Sequence Diagram for <Assign Committee Role>

2.2.2 P002: <Proposal Management Module> Subsystem

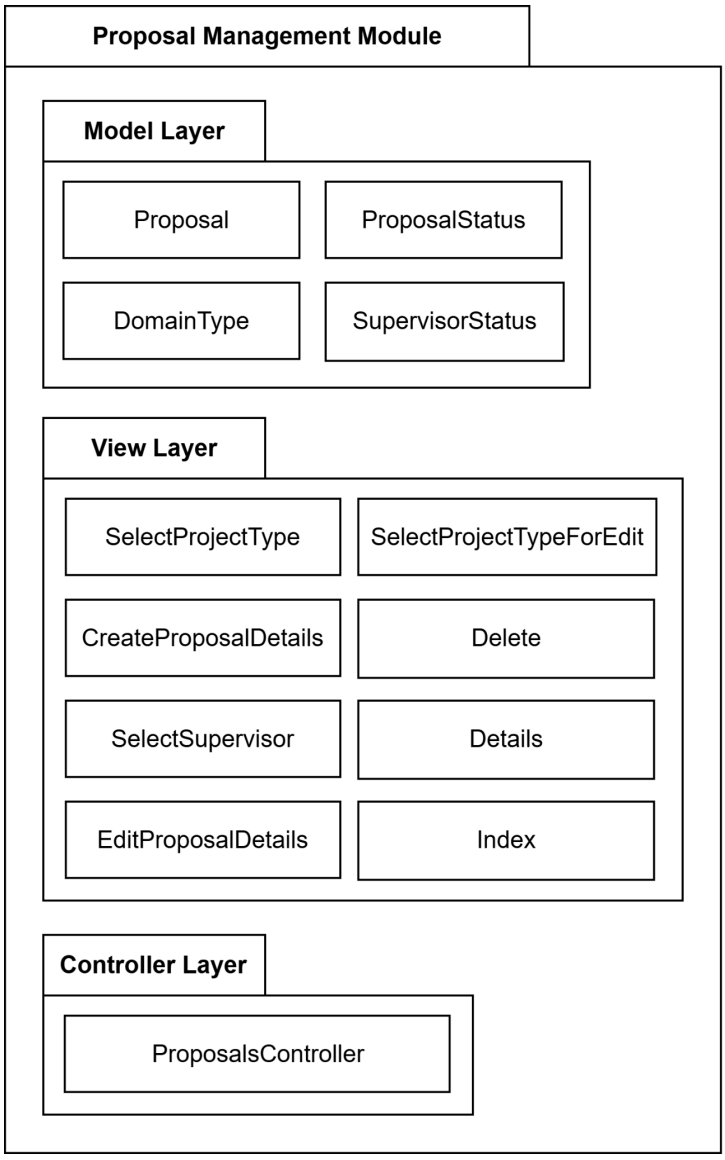

Figure 2.2.2.1: Package Diagram for <Proposal Management Module> Subsystem

2.2.2.1 Class Diagram

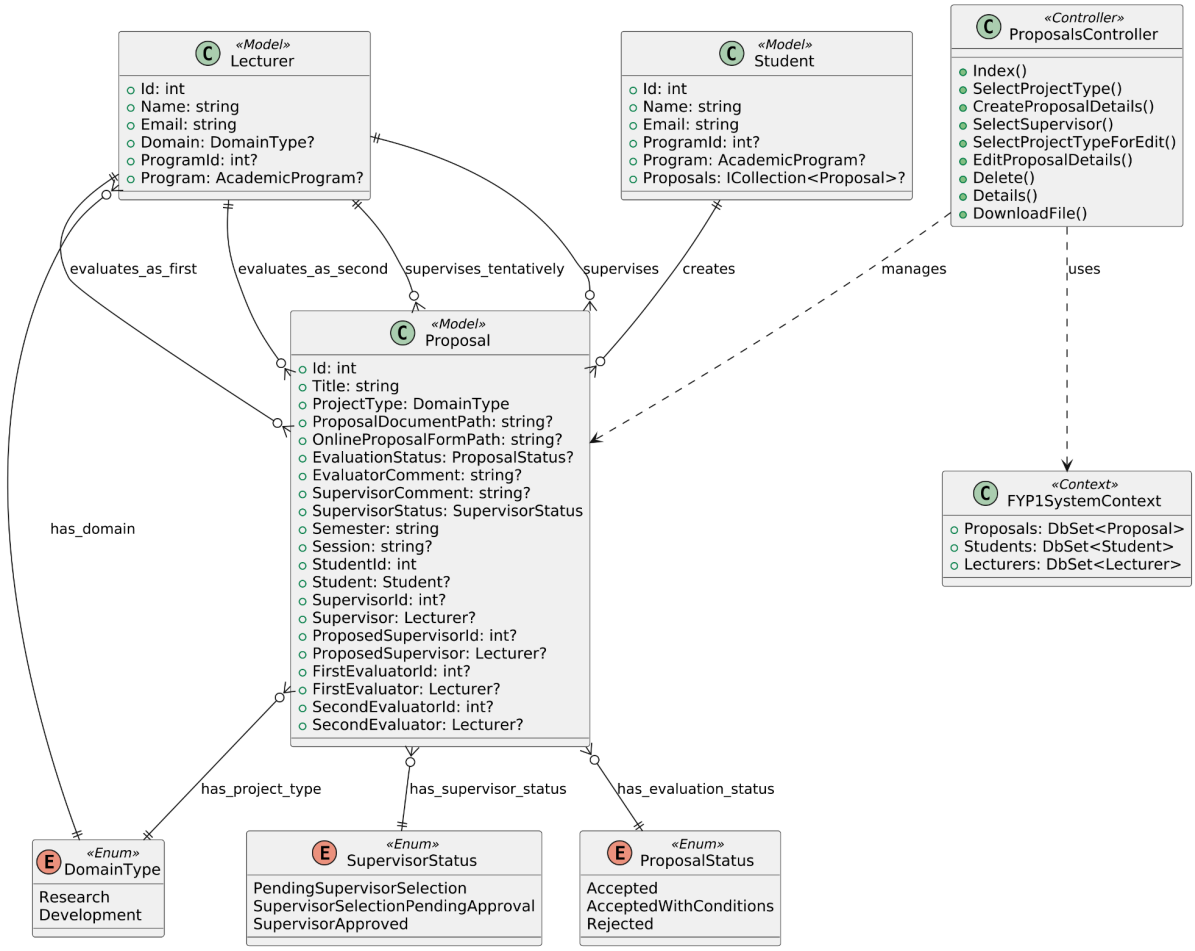

Figure 2.2.2.1.1: Class Diagram for <Proposal Management Module> Subsystem

Entity Name	ProposalsController
Method Name	Index()
Input	None
Output	List of all proposals with related entities
Algorithm	1. Start. 2. Query Proposals table with Include for Student, Supervisor, FirstEvaluator and SecondEvaluator. 3. Get all proposals from database. 4. Return Index view with proposals list. 5. End.

Entity Name	ProposalsController
Method Name	SelectProjectType()
Input	None
Output	Project type selection view for students
Algorithm	1. Start. 2. Check if user has "Student" role authorization. 2.1 If not authorized, redirect to Students Index. 3. Return SelectProjectType view with DomainType options (Research/Development). 4. End.

Entity Name	ProposalsController
Method Name	CreateProposalDetails()
Input	DomainType, Proposal object and IFormFiles
Output	Proposal creation form or created proposal with file uploads
Algorithm	1. Start. 2. Check if user has "Student" role authorization. 2.1 If not authorized, redirect to Students Index. 3. Get UserId from session. 4. If GET request: 4.1 Create new Proposal object with ProjectType and StudentId. 4.2 Set SupervisorStatus to PendingSupervisorSelection. 4.3 Return CreateProposalDetails view. 5. If POST request: 5.1 Validate required files (OnlineProposalFormFile, ProposalDocumentFile). 5.2 If files missing, add ModelState errors and return view. 5.3 If ModelState valid: 5.3.1 Generate unique filenames with GUID prefix. 5.3.2 Save OnlineProposalFormFile to uploads directory. 5.3.3 Save ProposalDocumentFile to uploads directory. 5.3.4 Set file paths in proposal object. 5.3.5 Add proposal to Proposals DbSet.

	5.3.6 Save changes to database. 5.3.7 Redirect to Students Index. 6. End.

Entity Name	ProposalsController
Method Name	SelectSupervisor()
Input	proposalId and supervisorId
Output	Supervisor selection form or updated proposal with selected supervisor
Algorithm	1. Start. 2. Check if user has "Student" role authorization. 2.1 If not authorized, redirect to Students Index. 3. If GET request: 3.1 Query Proposals table with Include Student. 3.2 Find proposal by proposalId. 3.3 Validate proposal belongs to current user. 3.4 Check if supervisor already approved (return BadRequest if so). 3.5 Validate SupervisorStatus allows selection. 3.6 Query Lecturers where Domain matches proposal ProjectType. 3.7 Create SelectList for supervisors dropdown. 3.8 Return SelectSupervisor view. 4. If POST request: 4.1 Find proposal by proposalId and validate ownership. 4.2 Find supervisor by supervisorId. 4.3 Validate supervisor domain matches proposal ProjectType. 4.4 If validation fails, reload supervisors and return view with error. 4.5 If supervisor selection changed: 4.5.1 Set proposal.ProposedSupervisorId. 4.5.2 Set SupervisorStatus to SupervisorSelectionPendingApproval. 4.5.3 Save changes to database. 4.5.4 Set success message in TempData. 4.6 Redirect to Students Index. 5. End.

Entity Name	ProposalsController
Method Name	SelectProjectTypeForEdit()
Input	Proposal ID
Output	Project type selection view for editing existing proposal
Algorithm	1. Start. 2. Check if user has "Student" role authorization. 2.1 If not authorized, redirect to Students Index. 3. Query Proposals table to find proposal by ID. 4. If proposal not found, return NotFound. 5. Return SelectProjectTypeForEdit view with proposal data. 6. End.

Entity Name	ProposalsController
Method Name	EditProposalDetails()
Input	Proposal ID, DomainType, Proposal object and IFormFiles
Output	Proposal edit form or updated proposal with modified details
Algorithm	1. Start. 2. Check if user has "Student" role authorization. 2.1 If not authorized, redirect to Students Index. 3. If GET request: 3.1 Find proposal by ID. 3.2 If proposal not found, return NotFound. 3.3 Set proposal.ProjectType to provided projectType. 3.4 Validate proposal belongs to current user. 3.5 Query Lecturers where Domain matches ProjectType. 3.6 Create SelectList for lecturers dropdown. 3.7 Set ViewBag values and return EditProposalDetails view. 4. If POST request: 4.1 Validate ID matches proposal.Id. 4.2 Validate proposal belongs to current user. 4.3 If ModelState invalid, reload lecturers dropdown and return view. 4.4 Get existing proposal from database (AsNoTracking). 4.5 Handle file uploads: 4.5.1 If ProposalDocumentFile provided, save new file and update path.

	4.5.2 If OnlineProposalFormFile provided, save new file and update path. 4.5.3 If no new files, keep existing file paths. 4.6 Update proposal in context and save changes. 4.7 Redirect to Students Index. 5. End.

Entity Name	ProposalsController
Method Name	Delete()
Input	Proposal ID
Output	Proposal deletion confirmation or completed deletion
Algorithm	1. Start. 2. Check if user has "Student" role authorization. 2.1 If not authorized, redirect to Students Index. 3. Get UserId from session. 4. If GET request: 4.1 Query Proposals table with Include Student and Supervisor. 4.2 Find proposal by ID. 4.3 If proposal not found, return NotFound. 4.4 Validate proposal belongs to current user. 4.5 Return Delete confirmation view with proposal data. 5. If POST request (DeleteConfirmed): 5.1 Find proposal by ID. 5.2 If proposal found and belongs to current user: 5.2.1 Remove proposal from Proposals DbSet. 5.2.2 Save changes to database. 5.3 Redirect to Students Index. 6. End.

Entity Name	ProposalsController
Method Name	Details()
Input	Proposal ID
Output	Detailed view of proposal with all related information

Algorithm	1. Start. 2. Check if user has "Student" or "Supervisor" role authorization. 2.1 If not authorized, redirect to Students Index. 3. Query Proposals table with Include for Student, Supervisor, FirstEvaluator and SecondEvaluator. 4. Find proposal by ID. 5. If proposal not found, return NotFound. 6. Return Details view with complete proposal information. 7. End.

Entity Name	ProposalsController
Method Name	DownloadFile()
Input	File path string
Output	Physical file download or NotFound result
Algorithm	1. Start. 2. Validate path parameter is not null or empty. 2.1 If invalid, return NotFound. 3. Remove leading slash from path if present. 4. Combine path with wwwroot directory to get full file path. 5. Check if physical file exists at the path. 5.1 If file doesn't exist, return NotFound. 6. Set content type to "application/pdf". 7. Return PhysicalFile result with file path, content type, and filename. 8. End.

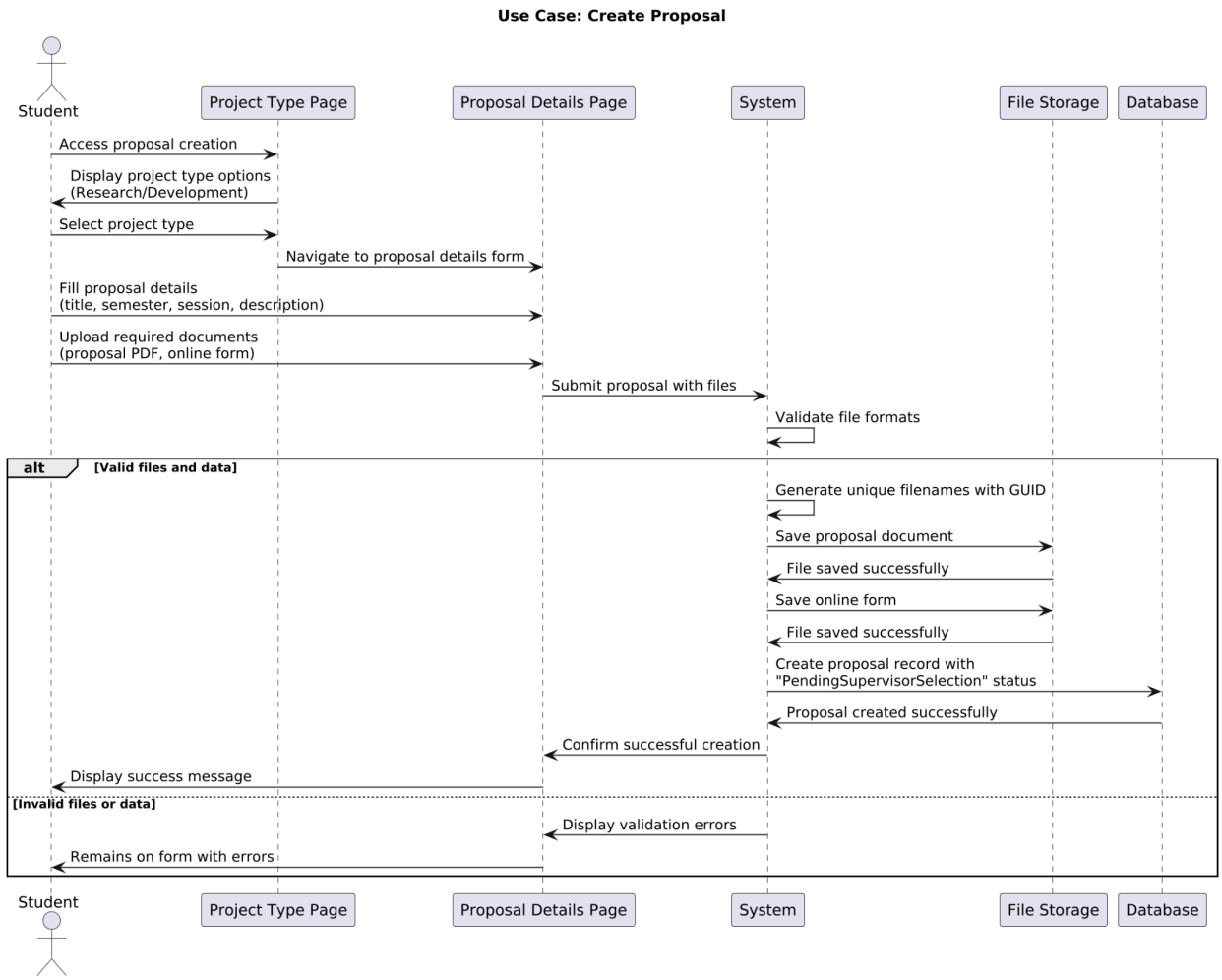

Figure 2.2.2.1.2: Sequence Diagram for <Create Proposal>

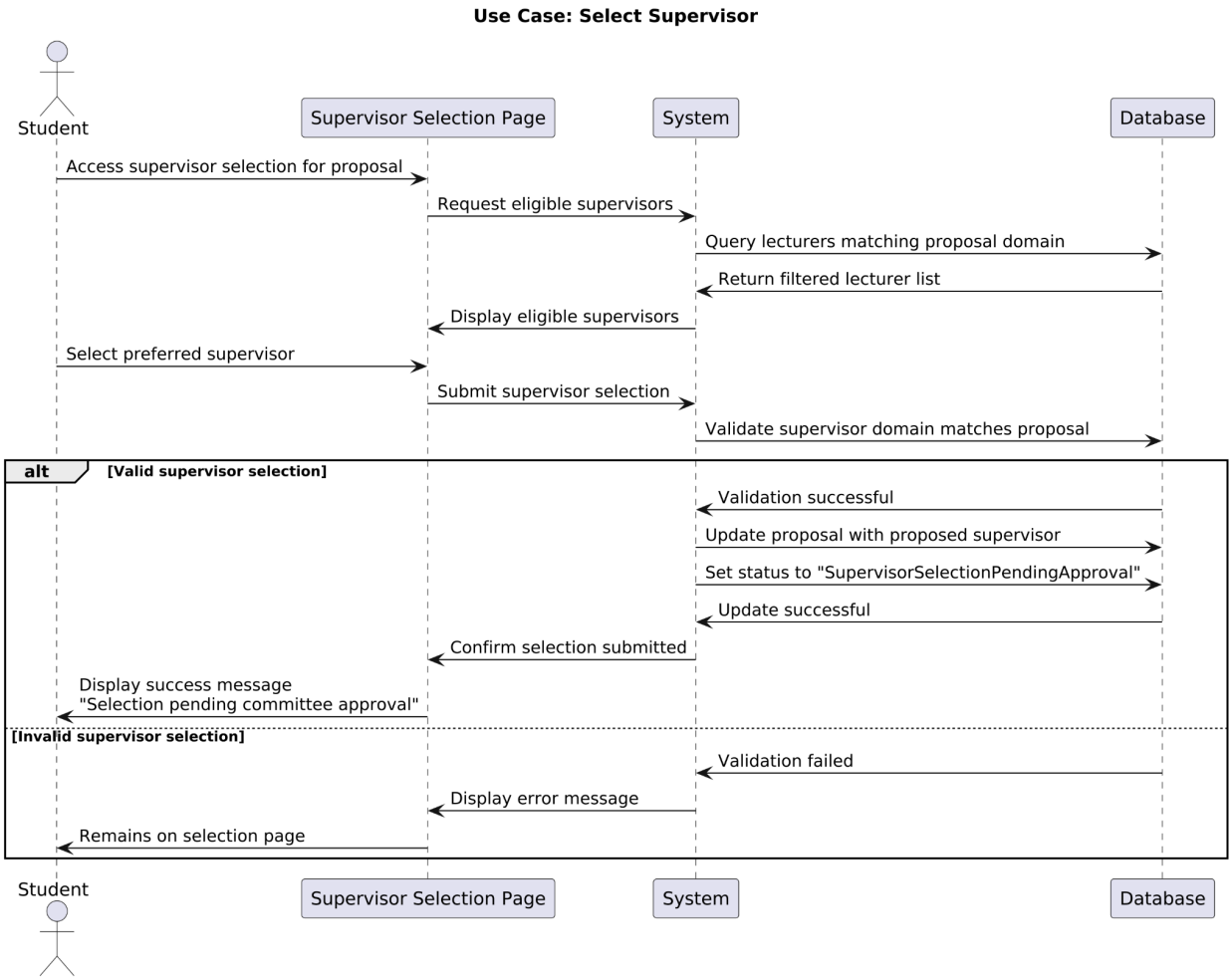

Figure 2.2.2.1.3: Sequence Diagram for <Select Supervisor>

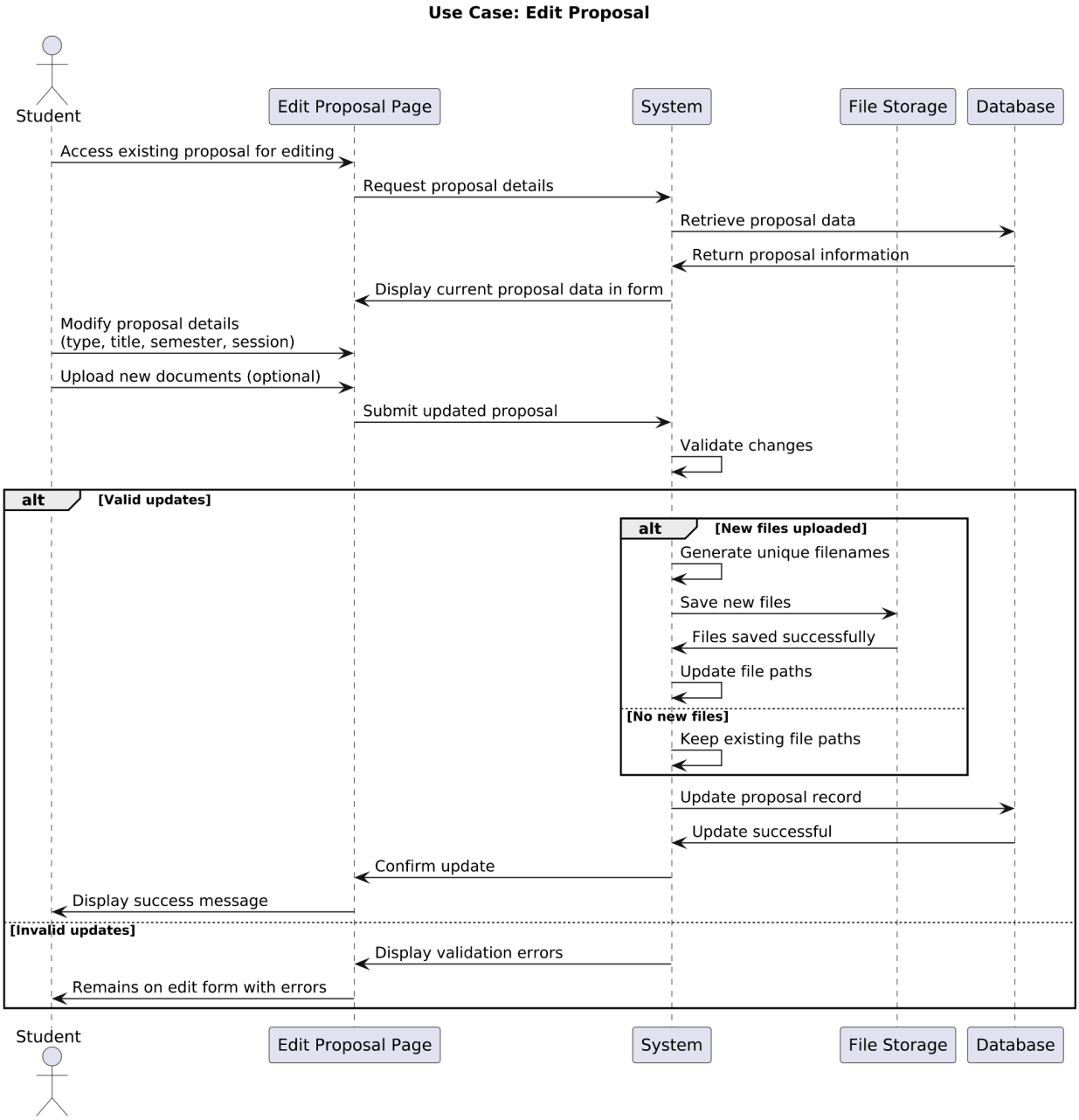

Figure 2.2.2.1.4: Sequence Diagram for <Edit Proposal>

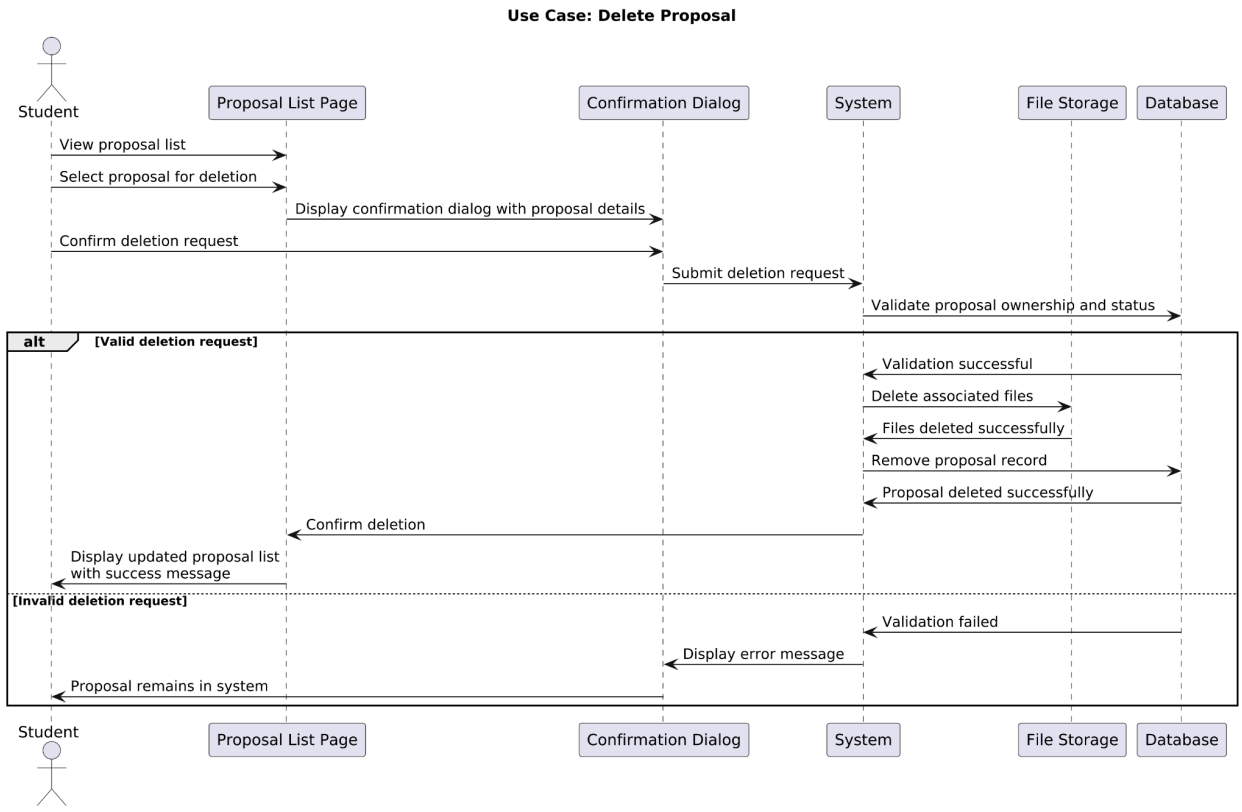

Figure 2.2.2.1.5: Sequence Diagram for <Delete Proposal>

2.2.3 P003: <Evaluation & Assignment Module> Subsystem

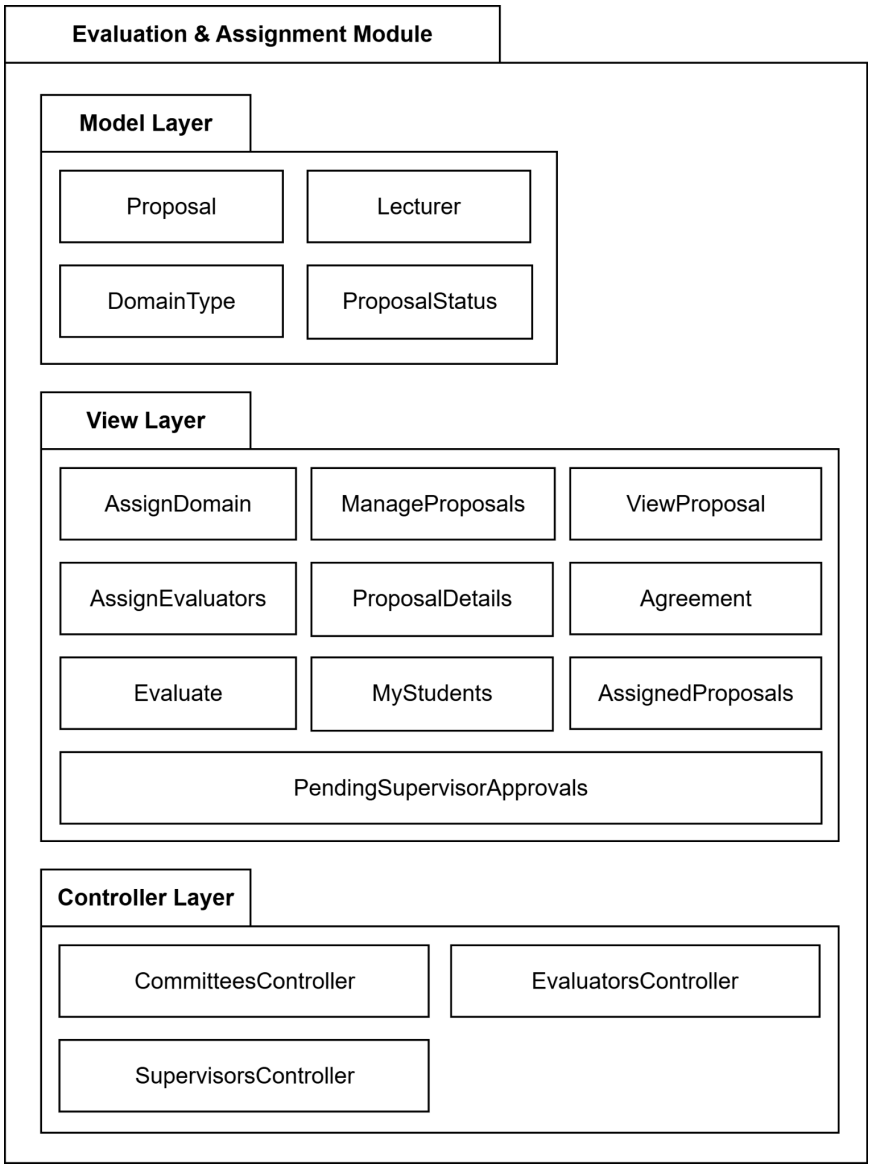

Figure 2.2.3.1: Package Diagram for <Evaluation & Assignment Module> Subsystem

2.2.3.1 Class Diagram

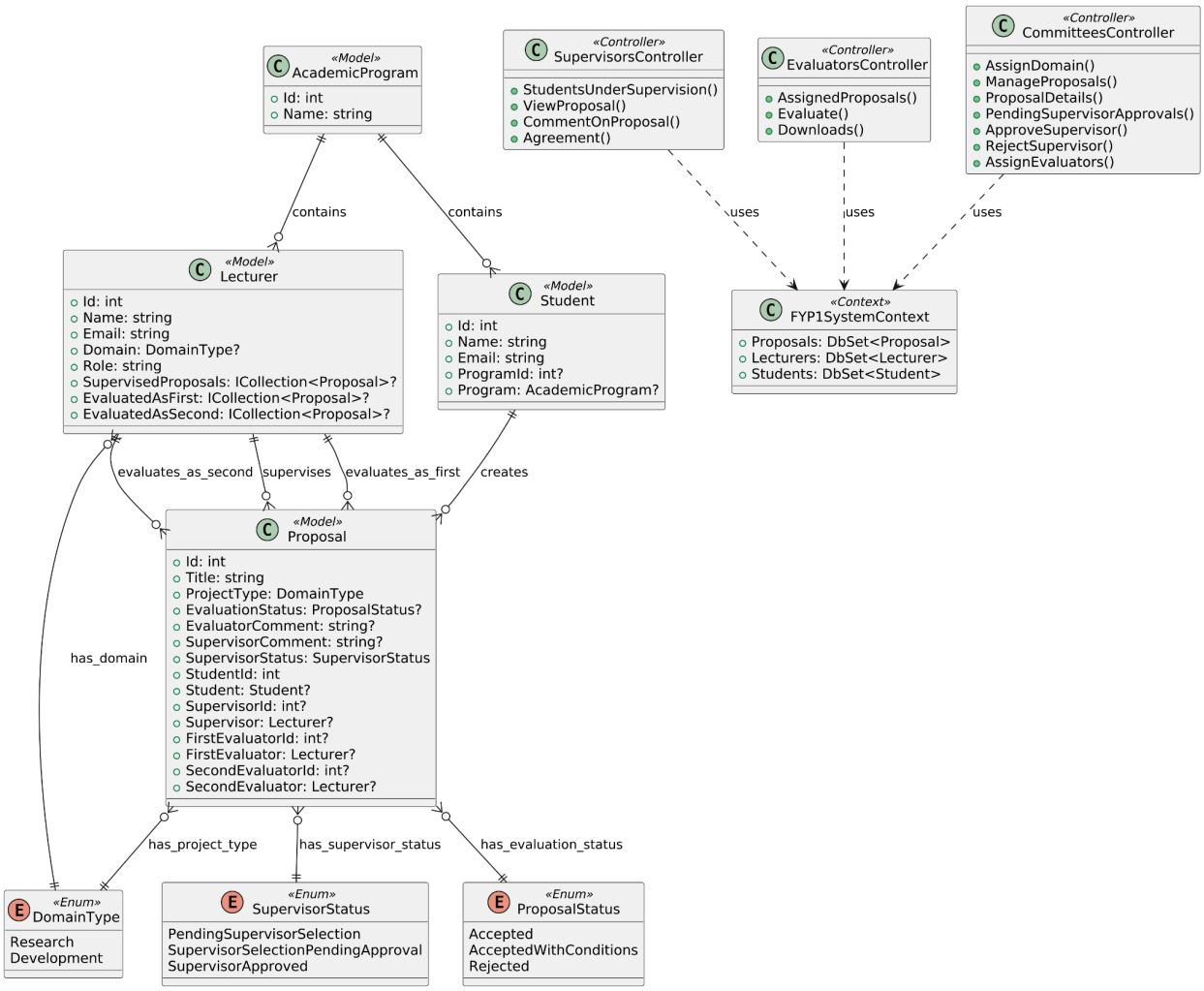

Figure 2.2.3.1.1: Class Diagram for <Evaluation & Assignment Module> Subsystem

Entity Name	SupervisorsController
Method Name	StudentsUnderSupervision()
Input	semester, session
Output	List of proposals supervised by current lecturer with filtering options
Algorithm	1. Start. 2. Get supervisorId from session. 3. If supervisorId is null, redirect to Home Index. 4. Create query for Proposals table with Include Student and Supervisor.

	5. Filter proposals where SupervisorId equals current supervisorId. 6. If semester parameter provided: 6.1 Add filter where Proposal.Semester equals semester. 7. If session parameter provided: 7.1 Add filter where Proposal.Session equals session. 8. Execute query to get filtered proposals list. 9. Set ViewBag values: 9.1 ViewBag.SupervisorId = supervisorId 9.2 ViewBag.Semester = semester 9.3 ViewBag.Session = session 10. Return StudentsUnderSupervision view with proposals list. 11. End.

Entity Name	SupervisorsController
Method Name	ViewProposal()
Input	Proposal ID
Output	Detailed view of specific proposal with all related information
Algorithm	1. Start. 2. Query Proposals table with Include for Student, Supervisor, FirstEvaluator and SecondEvaluator. 3. Find proposal by provided ID using FirstOrDefaultAsync. 4. If proposal not found, return NotFound result. 5. Return ViewProposal view with complete proposal data. 6. End.

Entity Name	SupervisorsController
Method Name	CommentOnProposal()
Input	Proposal ID, supervisor comment string
Output	Updated proposal with supervisor comment
Algorithm	1. Start. 2. Find proposal by ID in Proposals table. 3. If proposal not found, return NotFound result. 4. Set proposal.SupervisorComment to provided supervisorComment. 5. Save changes to database asynchronously.

	6. Redirect to StudentsUnderSupervision action. 7. End.

Entity Name	SupervisorsController
Method Name	Agreement()
Input	Proposal ID
Output	Supervisor-student agreement view for specific proposal
Algorithm	1. Start. 2. Query Proposals table with Include Student and Supervisor. 3. Find proposal by proposalId using FirstOrDefaultAsync. 4. If proposal not found, return NotFound result. 5. Return Agreement view with proposal data. 6. End.

Entity Name	EvaluatorsController
Method Name	AssignedProposals()
Input	None
Output	List of proposals assigned to current evaluator
Algorithm	1. Start. 2. Check if user has "Evaluator" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Get evaluatorId from session. 4. If evaluatorId is null, redirect to Home Index. 5. Query Proposals table with Include Student and Supervisor. 6. Filter proposals where FirstEvaluatorId equals evaluatorId OR SecondEvaluatorId equals evaluatorId. 7. Execute query to get assigned proposals list. 8. Return AssignedProposals view with proposals data. 9. End.

Entity Name	EvaluatorsController
Method Name	Evaluate()
Input	proposalId, updated Proposal object

Output	Evaluation form or updated proposal with evaluation results
Algorithm	1. Start. 2. Check if user has "Evaluator" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Get evaluatorId from session. 4. If evaluatorId is null, redirect to Home Index. 5. If GET request: 5.1 Query Proposals table with Include Student and Supervisor. 5.2 Find proposal where ID matches proposalId AND (FirstEvaluatorId equals evaluatorId OR SecondEvaluatorId equals evaluatorId). 5.3 If proposal not found, return NotFound. 5.4 Return Evaluate view with proposal data. 6. If POST request: 6.1 Find proposal where ID matches AND evaluator is assigned to it. 6.2 If proposal not found, return NotFound. 6.3 Update proposal.EvaluationStatus from updatedProposal. 6.4 Update proposal.EvaluatorComment from updatedProposal. 6.5 Save changes to database asynchronously. 6.6 Redirect to AssignedProposals. 7. End.

Entity Name	EvaluatorsController
Method Name	Downloads()
Input	File path string
Output	Physical file download or NotFound result
Algorithm	1. Start. 2. Validate path parameter is not null or empty. 2.1 If invalid, return NotFound. 3. Combine path with wwwroot directory to get full file path. 4. Check if physical file exists at the constructed path. 4.1 If file doesn't exist, return NotFound. 5. Set content type to "application/pdf". 6. Return PhysicalFile result with file path, content type, and filename. 7. End.

Entity Name	CommitteesController
Method Name	AssignDomain()
Input	lecturerId, DomainType
Output	Domain assignment interface or updated lecturer with assigned domain
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Lecturers Index. 3. If GET request: 3.1 Get actualUserId from session. 3.2 If userId is null, redirect to Lecturers Index. 3.3 Find committee lecturer in Lecturers table. 3.4 If lecturer not found or ProgramId is null, redirect to Lecturers Index. 3.5 Query Lecturers table with Include Program where ProgramId matches committee's program. 3.6 Return AssignDomain view with filtered lecturers list. 4. If POST request: 4.1 Find lecturer by lecturerId in Lecturers table. 4.2 If lecturer found: 4.2.1 Set lecturer.Domain to provided domain. 4.2.2 Update lecturer in context. 4.2.3 Save changes to database asynchronously. 4.3 Redirect to AssignDomain (GET). 5. End.

Entity Name	CommitteesController
Method Name	ManageProposals()
Input	semester, session
Output	List of proposals with filtering options for committee management
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Lecturers Index. 3. Create query for Proposals table with Include Student and Supervisor. 4. If semester parameter provided and not empty:

	4.1 Add filter where Proposal.Semester equals semester. 5. If session parameter provided and not empty: 5.1 Add filter where Proposal.Session equals session. 6. Execute query to get filtered proposals list. 7. Set ViewBag values: 7.1 ViewBag.Semester = semester. 7.2 ViewBag.Session = session. 8. Return ManageProposals view with proposals data. 9. End.

Entity Name	CommitteesController
Method Name	ProposalDetails()
Input	Proposal ID
Output	Detailed proposal view for committee review
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Query Proposals table with Include Student, Supervisor, FirstEvaluator, and SecondEvaluator. 4. Find proposal by ID using FirstOrDefaultAsync. 5. If proposal not found, return NotFound result. 6. Return ProposalDetails view with complete proposal information. 7. End.

Entity Name	CommitteesController
Method Name	PendingSupervisorApprovals()
Input	None
Output	List of proposals pending supervisor approval in committee's program
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Get committeeUserId from session. 4. Find committee lecturer in Lecturers table.

	5. If committee not found or ProgramId is null, redirect to Home Index. 6. Query Proposals table with Include Student and ProposedSupervisor. 7. Filter proposals where: 7.1 SupervisorStatus equals SupervisorSelectionPendingApproval 7.2 Student.ProgramId equals committee.ProgramId. 8. Execute query to get pending approvals list. 9. Return PendingSupervisorApprovals view with proposals data. 10. End.

Entity Name	CommitteesController
Method Name	ApproveSupervisor()
Input	Proposal ID
Output	Approved supervisor assignment and updated proposal status
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Find proposal by proposalId in Proposals table. 4. If proposal not found, return NotFound. 5. Validate SupervisorStatus is SupervisorSelectionPendingApproval. 5.1 If not pending approval, return BadRequest with error message. 6. Update proposal: 6.1 Set proposal.SupervisorId = proposal.ProposedSupervisorId. 6.2 Set proposal.SupervisorStatus = SupervisorApproved. 6.3 Set proposal.ProposedSupervisorId = null. 6.4 Update proposal in context. 7. Find lecturer by SupervisorId. 8. If lecturer found and doesn't have "Supervisor" role: 8.1 Add "Supervisor" role to lecturer using AddRole method. 8.2 Update lecturer in context. 9. Save changes to database asynchronously. 10. Redirect to PendingSupervisorApprovals. 11. End.

Entity Name	CommitteesController
Method Name	RejectSupervisor()
Input	Proposal ID
Output	Rejected supervisor selection and reset proposal status
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Find proposal by proposalId in Proposals table. 4. If proposal not found, return NotFound. 5. Validate SupervisorStatus is SupervisorSelectionPendingApproval. 5.1 If not pending approval, return BadRequest with error message. 6. Update proposal: 6.1 Set proposal.ProposedSupervisorId = null. 6.2 Set proposal.SupervisorStatus = PendingSupervisorSelection. 6.3 Update proposal in context. 7. Save changes to database asynchronously. 8. Redirect to PendingSupervisorApprovals. 9. End.

Entity Name	CommitteesController
Method Name	AssignEvaluators()
Input	proposalId, FirstEvaluatorId, SecondEvaluatorId
Output	Evaluator assignment form or updated proposal with assigned evaluators
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Home Index. 3. If GET request: 3.1 Query Proposals table with Include Student and Supervisor. 3.2 Find proposal by proposalId. 3.3 If proposal not found, return NotFound.

	3.4 Query Lecturers where Domain equals proposal.ProjectType AND Id not equals proposal.SupervisorId. 3.5 Create SelectList for eligible lecturers (Id, Name). 3.6 Set ViewBag.Lecturers with SelectList. 3.7 Return AssignEvaluators view with proposal data. 4. If POST request: 4.1 Query Proposals table with Include Supervisor. 4.2 Find proposal by ID. 4.3 If proposal not found, return NotFound. 4.4 Validate evaluator assignments: 4.4.1 If FirstEvaluatorId equals SecondEvaluatorId, add ModelState error. 4.4.2 If FirstEvaluatorId or SecondEvaluatorId equals SupervisorId, add ModelState error. 4.5 If ModelState invalid: 4.5.1 Reload eligible lecturers for dropdown. 4.5.2 Return view with validation errors. 4.6 If ModelState valid: 4.6.1 Set proposal.FirstEvaluatorId = FirstEvaluatorId. 4.6.2 Set proposal.SecondEvaluatorId = SecondEvaluatorId. 4.6.3 Find FirstEvaluator and SecondEvaluator lecturers. 4.6.4 Add "Evaluator" role to both lecturers if they don't have it. 4.6.5 Update lecturers in context. 4.6.6 Save changes to database asynchronously. 4.6.7 Redirect to ManageProposals. 5. End.

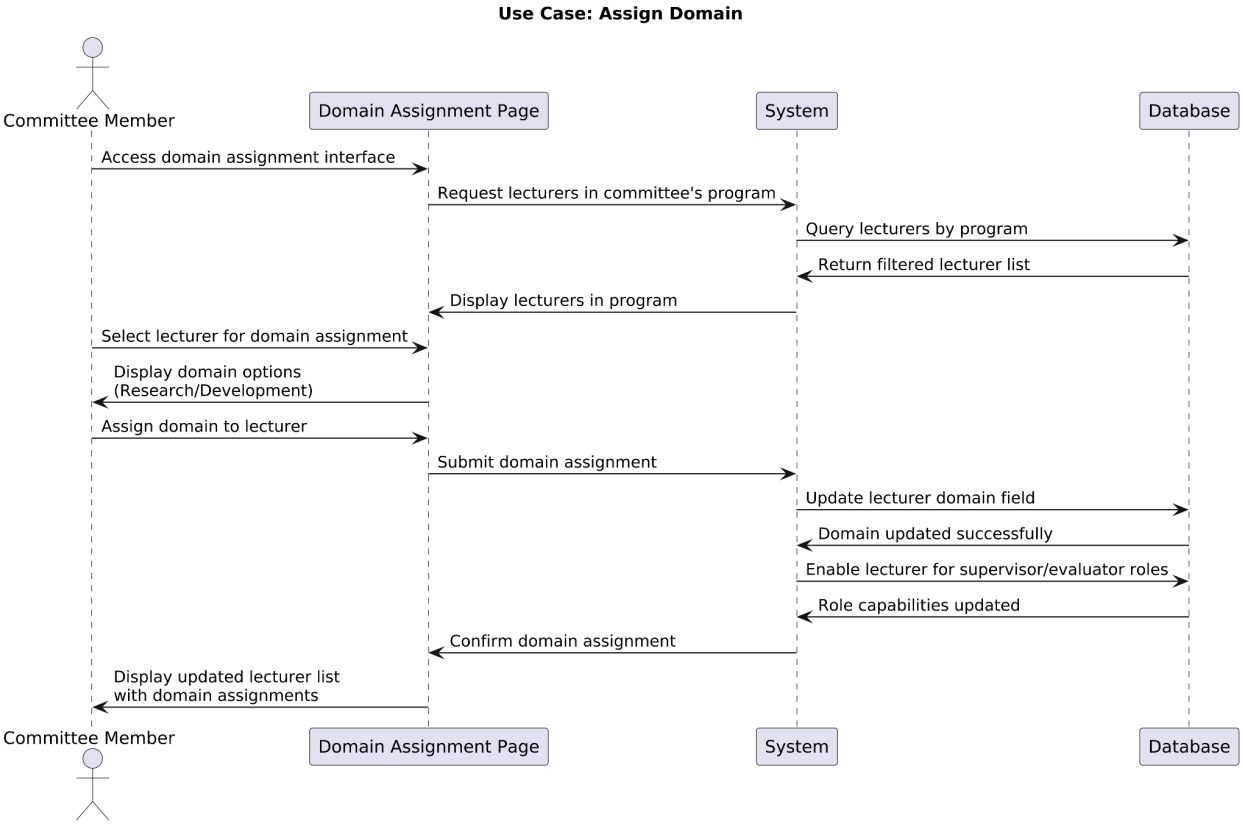

Figure 2.2.3.1.2: Sequence Diagram for <Assign Domain>

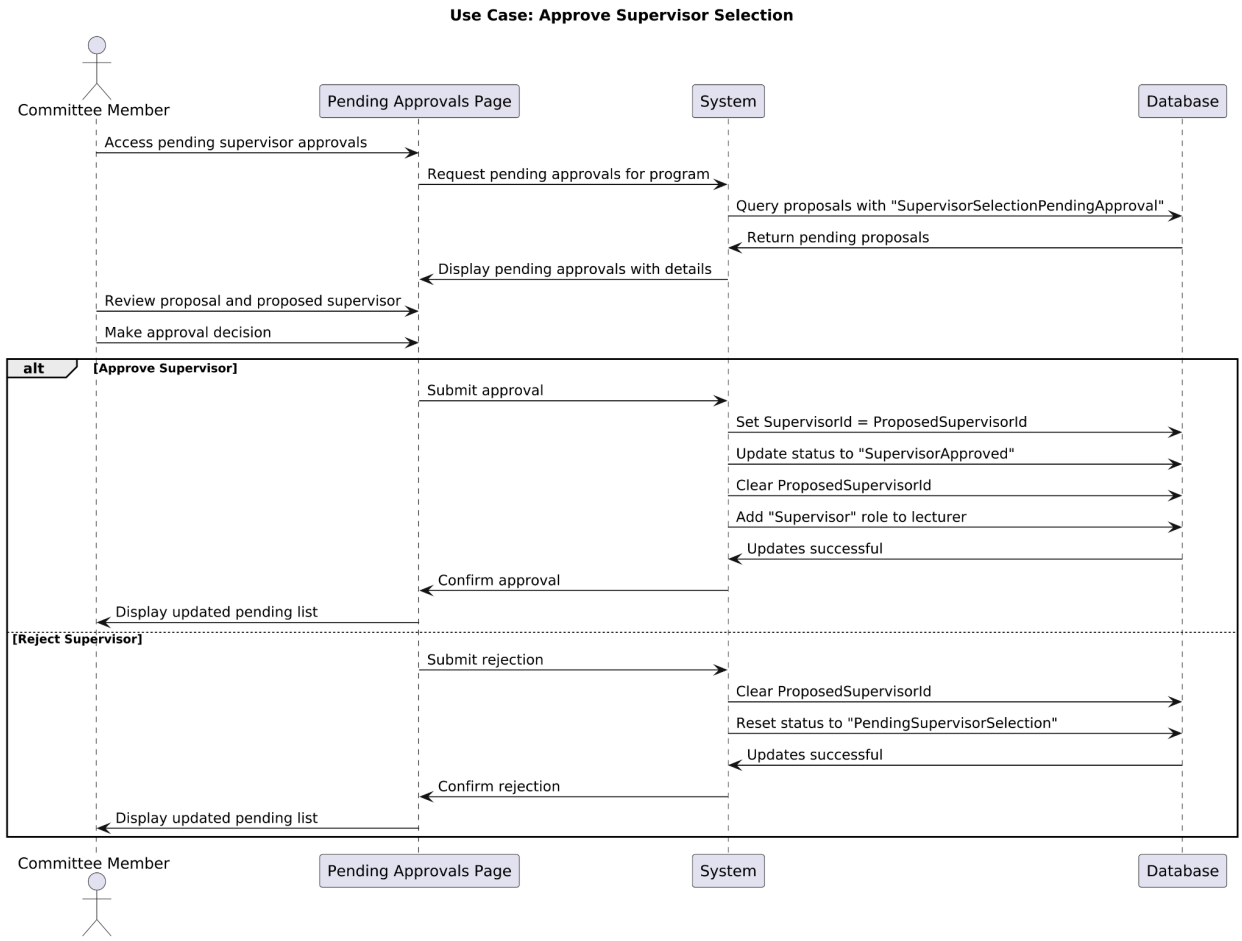

Figure 2.2.3.1.3: Sequence Diagram for <Approve Supervisor Selection>

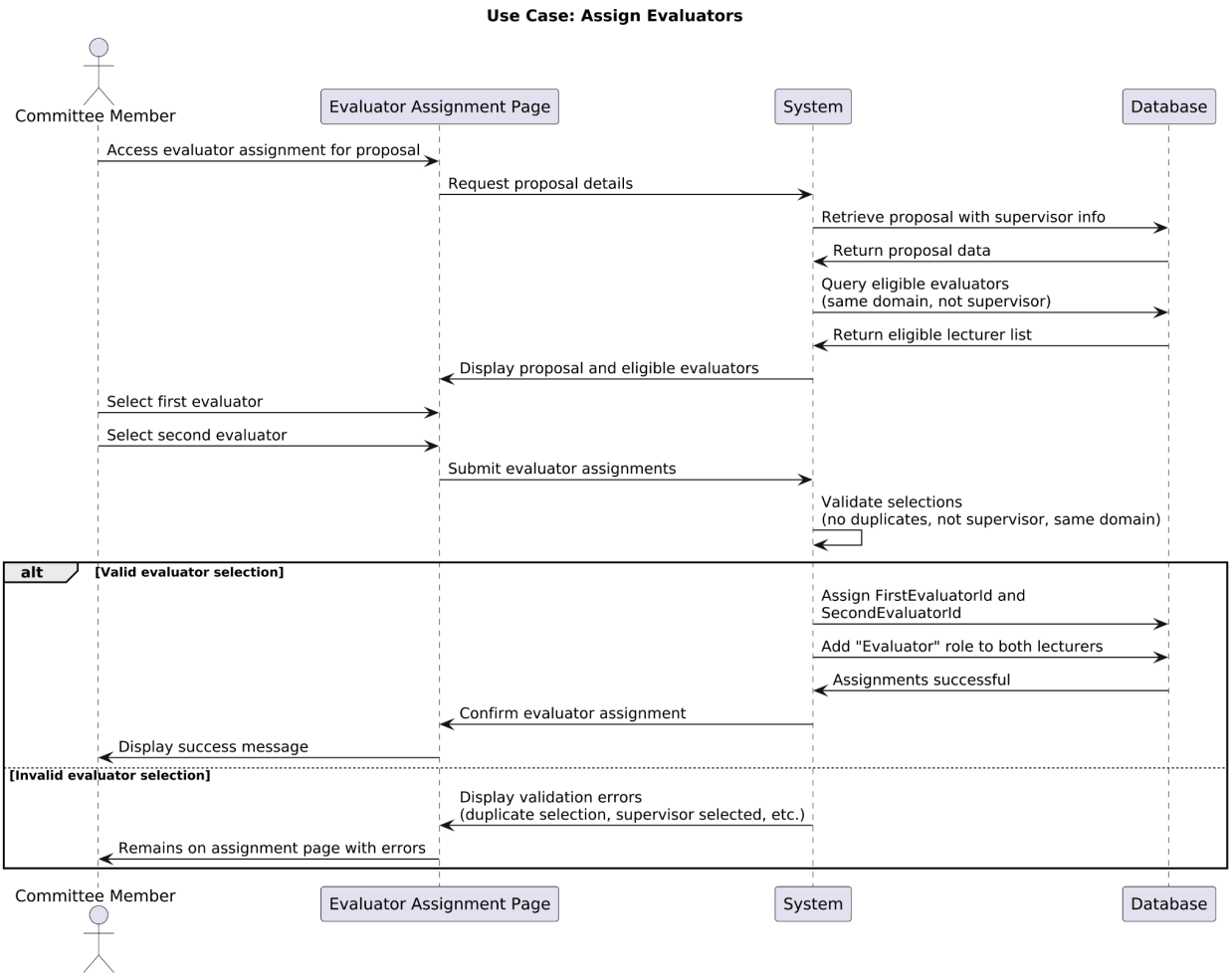

Figure 2.2.3.1.4: Sequence Diagram for <Assign Evaluators>

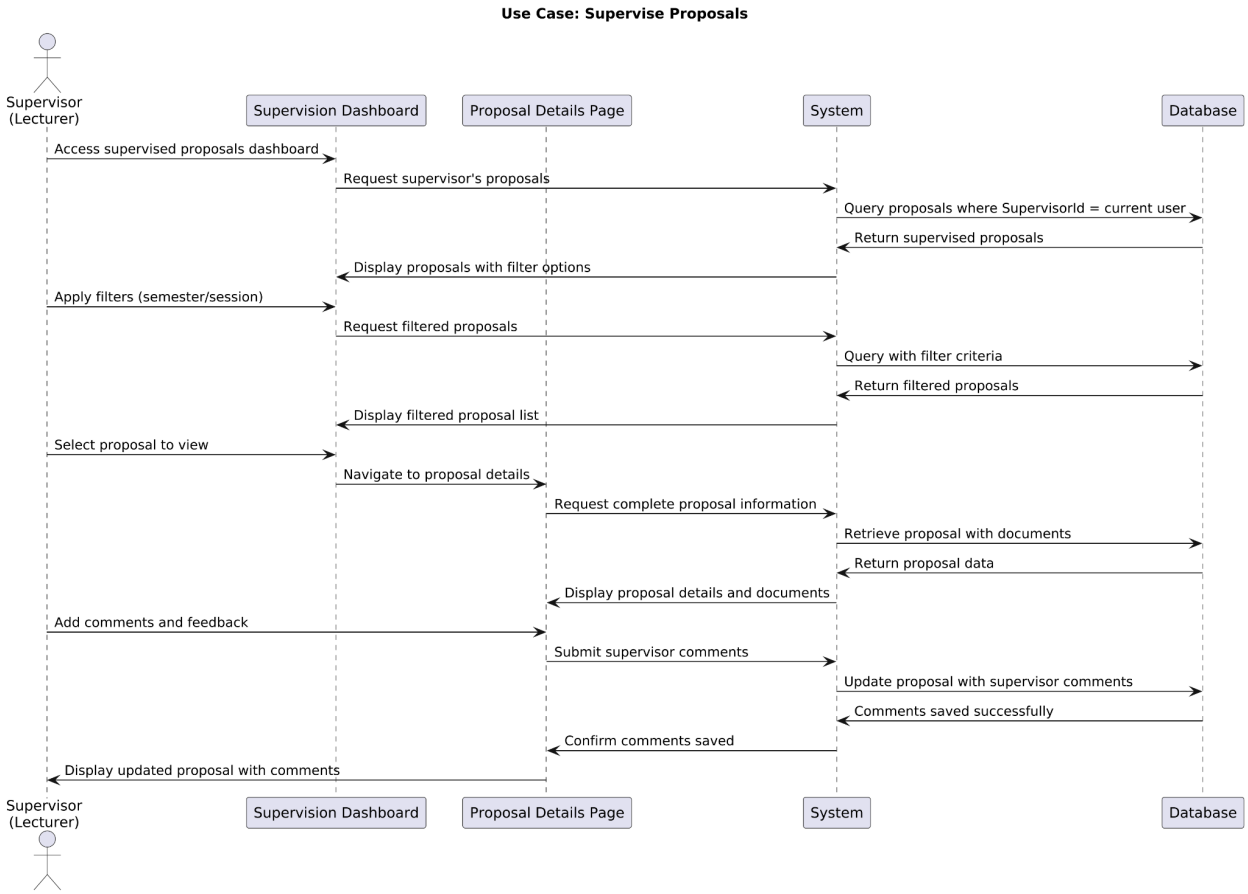

Figure 2.2.3.1.5: Sequence Diagram for <Supervise Proposals>

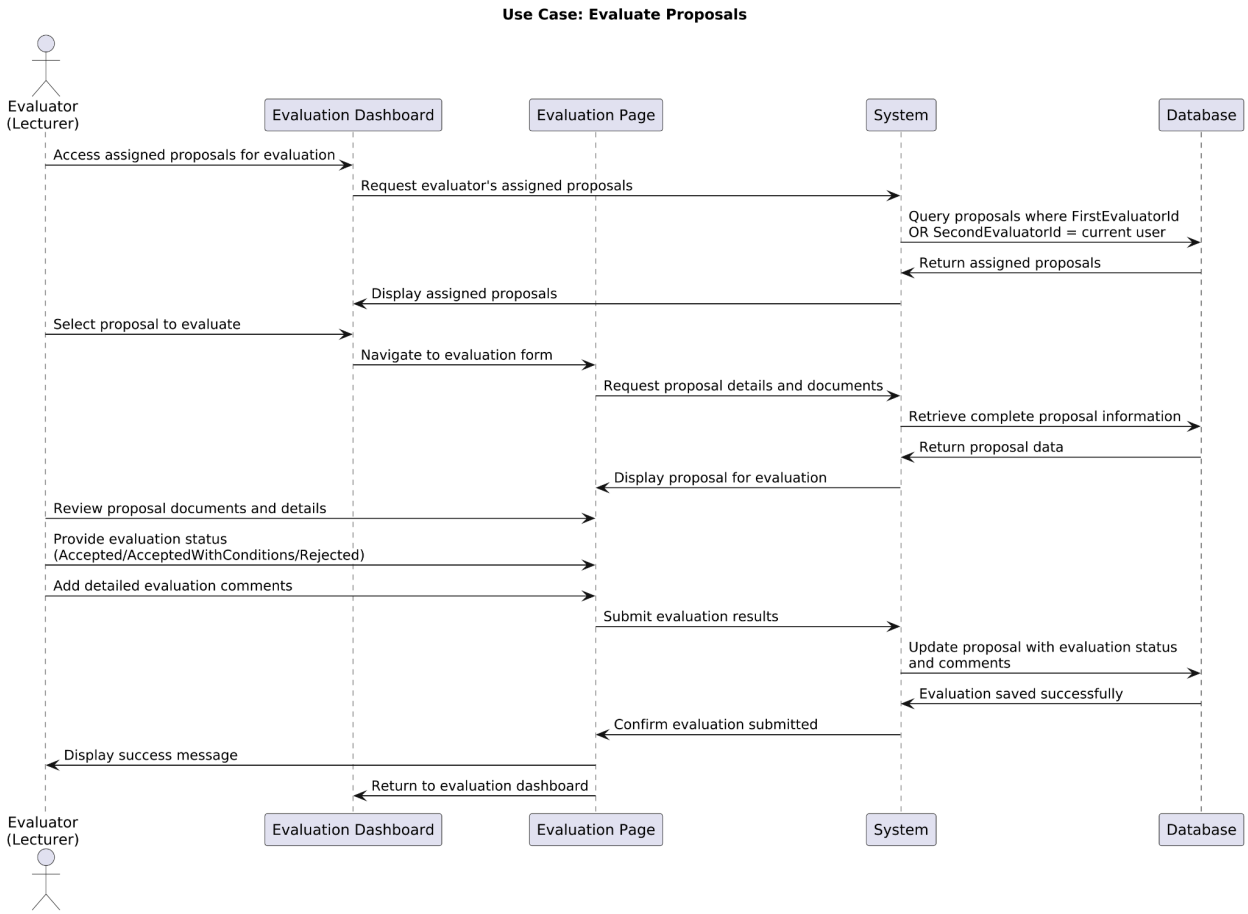

Figure 2.2.3.1.6: Sequence Diagram for <Evaluate Proposals>

2.2.4 P004: <Administration Module> Subsystem

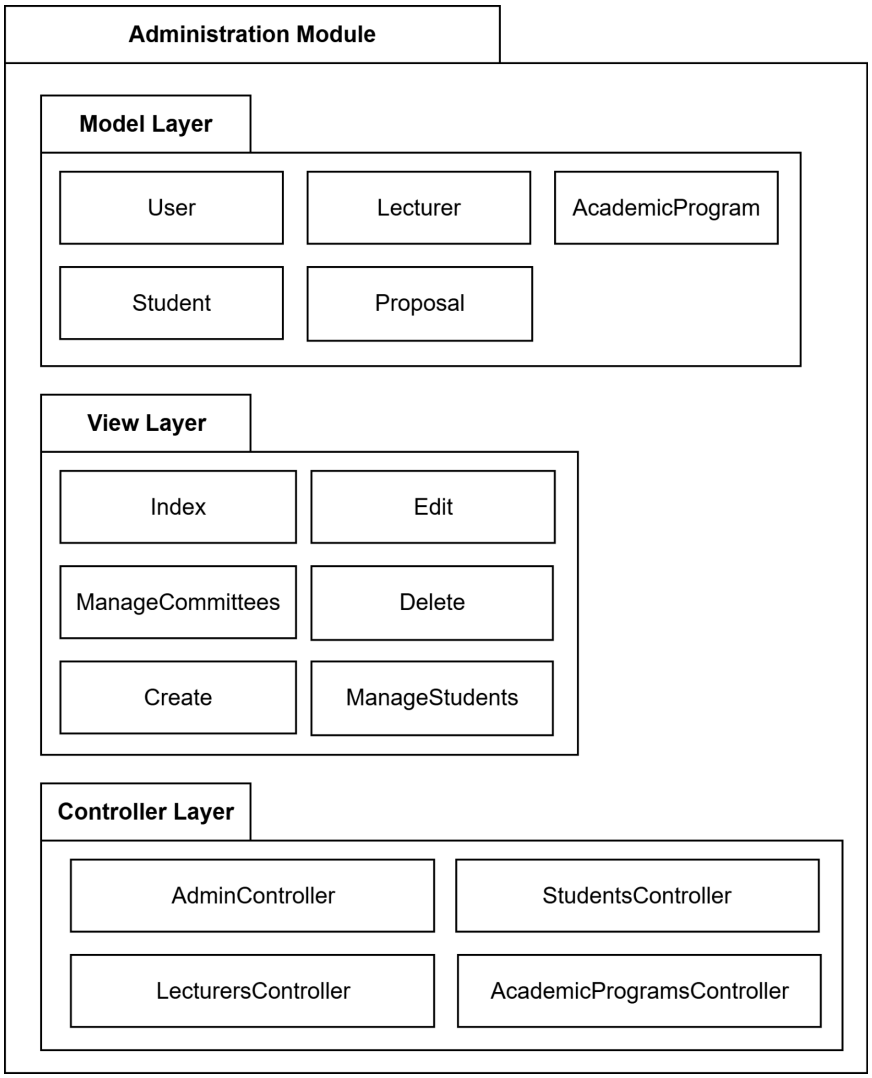

Figure 2.2.4.1: Package Diagram for <Administration Module> Subsystem

2.2.4.1 Class Diagram

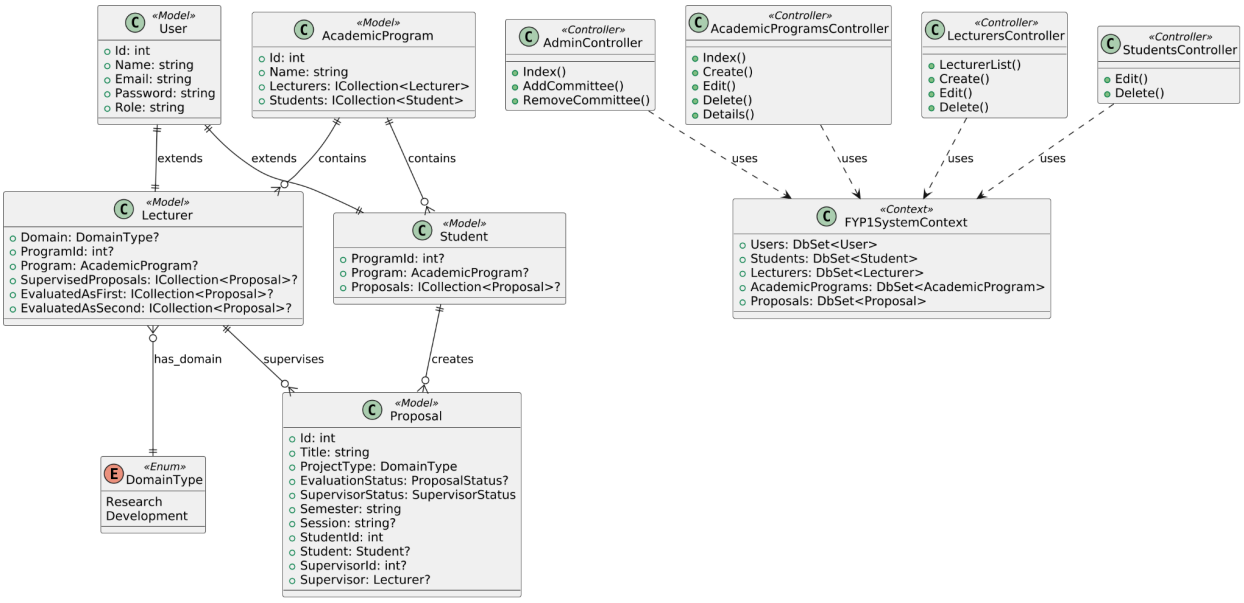

Figure 2.2.4.1.1: Class Diagram for <Administration Module> Subsystem

Entity Name	AdminController
Method Name	Index()
Input	None
Output	Admin dashboard with system statistics and overview
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Count total records in AcademicPrograms table. 4. Count total records in Lecturers table. 5. Query all lecturers from Lecturers table. 6. Filter lecturers by checking Role field for "Committee" role. 7. Count lecturers with Committee role. 8. Set ViewBag values: 8.1 ViewBag.TotalPrograms = program count 8.2 ViewBag.TotalLecturers = lecturer count 8.3 ViewBag.TotalCommittees = committee count. 9. Return admin dashboard view with statistics. 10. End.

Entity Name	AdminController
Method Name	AddCommittee()
Input	lecturerId
Output	Updated lecturer with Committee role added and redirect
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Find lecturer by lecturerId in Lecturers table. 4. If lecturer found: 4.1 Call lecturer.AddRole("Committee") method to add Committee role. 4.2 Update lecturer entity in context. 4.3 Save changes to database asynchronously. 5. Redirect to ManageCommittees action. 6. End.

Entity Name	AdminController
Method Name	RemoveCommittee()
Input	lecturerId
Output	Updated lecturer with Committee role removed and redirect
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Find lecturer by lecturerId in Lecturers table. 4. If lecturer found: 4.1 Call lecturer.RemoveRole("Committee") method to remove Committee role. 4.2 Update lecturer entity in context. 4.3 Save changes to database asynchronously. 5. Redirect to ManageCommittees action. 6. End.

Entity Name	AcademicProgramsController
Method Name	Index()

Input	None
Output	List of all academic programs
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Query AcademicPrograms table to get all programs. 4. Execute query asynchronously to retrieve programs list. 5. Return Index view with academic programs data. 6. End.

Entity Name	AcademicProgramsController
Method Name	Create()
Input	AcademicProgram object
Output	Academic program creation form or created program record
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. If GET request: 3.1 Return Create view with empty AcademicProgram form. 4. If POST request: 4.1 Validate ModelState for required fields. 4.2 If ModelState is valid: 4.2.1 Add academic program to AcademicPrograms DbSet. 4.2.2 Save changes to database asynchronously. 4.2.3 Redirect to Index action. 4.3 If ModelState invalid: 4.3.1 Return Create view with validation errors. 5. End.

Entity Name	AcademicProgramsController
Method Name	Edit()
Input	Program ID, AcademicProgram object
Output	Academic program edit form or updated program record

Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. If GET request: 3.1 Find academic program by ID in AcademicPrograms table. 3.2 If program not found, return NotFound. 3.3 Return Edit view with program data. 4. If POST request: 4.1 Validate that ID matches program.Id. 4.1.1 If not matching, return NotFound. 4.2 Validate ModelState for required fields. 4.3 If ModelState is valid: 4.3.1 Update academic program in context. 4.3.2 Save changes to database asynchronously. 4.3.3 Redirect to Index action. 4.4 If ModelState invalid: 4.4.1 Return Edit view with validation errors. 5. End.

Entity Name	AcademicProgramsController
Method Name	Delete()
Input	Program ID
Output	Academic program deletion confirmation or completed deletion
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. If GET request: 3.1 Find academic program by ID in AcademicPrograms table. 3.2 If program not found, return NotFound. 3.3 Return Delete confirmation view with program data. 4. If POST request (DeleteConfirmed): 4.1 Find academic program by ID in AcademicPrograms table. 4.2 If program found: 4.2.1 Remove program from AcademicPrograms DbSet. 4.2.2 Save changes to database asynchronously. 4.3 Redirect to Index action. 5. End.

Entity Name	AcademicProgramsController
Method Name	Details()
Input	Program ID
Output	Detailed view of academic program with related information
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Query AcademicPrograms table with Include for Lecturers and Students. 4. Find academic program by ID using FirstOrDefaultAsync. 5. If program not found, return NotFound. 6. Return Details view with complete program information including associated lecturers and students. 7. End.

Entity Name	LecturersController
Method Name	LecturerList()
Input	None
Output	Complete list of lecturers for admin management
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. Query Lecturers table with Include Program navigation property. 4. Execute query asynchronously to get all lecturers. 5. Return LecturerList view with lecturers data including their programs. 6. End.

Entity Name	LecturersController
Method Name	Create()
Input	Lecturer object

Output	Lecturer creation form or created lecturer record
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. If GET request: 3.1 Query AcademicPrograms table for dropdown options. 3.2 Create SelectList for programs (Id, Name). 3.3 Set ViewBag.Programs with SelectList. 3.4 Return Create view with empty lecturer form. 4. If POST request: 4.1 Validate ModelState for required fields. 4.2 If ModelState is valid: 4.2.1 Set lecturer.Role = "Lecturer". 4.2.2 Add lecturer to Lecturers DbSet. 4.2.3 Save changes to database asynchronously. 4.2.4 Redirect to LecturerList action. 4.3 If ModelState invalid: 4.3.1 Reload programs dropdown with selected value. 4.3.2 Return Create view with validation errors. 5. End.

Entity Name	LecturersController
Method Name	Edit()
Input	Lecturer ID, Lecturer object
Output	Lecturer edit form or updated lecturer record
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. If GET request: 3.1 Find lecturer by ID in Lecturers table. 3.2 If lecturer not found, return NotFound. 3.3 Query AcademicPrograms for dropdown options. 3.4 Create SelectList with current ProgramId selected. 3.5 Set ViewBag.Programs with SelectList. 3.6 Return Edit view with lecturer data. 4. If POST request: 4.1 Validate that ID matches lecturer.Id. 4.1.1 If not matching, return NotFound. 4.2 Validate ModelState for required fields.

	4.3 If ModelState is valid: 4.3.1 Set lecturer.Role = "Lecturer". 4.3.2 Update lecturer in context. 4.3.3 Save changes to database asynchronously. 4.3.4 Redirect to LecturerList action. 4.4 If ModelState invalid: 4.4.1 Reload programs dropdown with selected value. 4.4.2 Return Edit view with validation errors. 5. End.

Entity Name	LecturersController
Method Name	Delete()
Input	Lecturer ID
Output	Lecturer deletion confirmation or completed deletion
Algorithm	1. Start. 2. Check if user has "Admin" role authorization. 2.1 If not authorized, redirect to Home Index. 3. If GET request: 3.1 Query Lecturers table with Include Program navigation property. 3.2 Find lecturer by ID using FirstOrDefaultAsync. 3.3 If lecturer not found, return NotFound. 3.4 Return Delete confirmation view with lecturer data. 4. If POST request (DeleteConfirmed): 4.1 Find lecturer by ID in Lecturers table. 4.2 If lecturer found: 4.2.1 Remove lecturer from Lecturers DbSet. 4.2.2 Save changes to database asynchronously. 4.3 Redirect to LecturerList action. 5. End.

Entity Name	StudentsController
Method Name	Edit()
Input	Student ID, Student object
Output	Student edit form or updated student record

Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Lecturers Index. 3. If GET request: 3.1 Find student by ID in Students table. 3.2 If student not found, return NotFound. 3.3 Query AcademicPrograms for dropdown options. 3.4 Create SelectList with current ProgramId selected. 3.5 Set ViewBag.Programs with SelectList. 3.6 Return Edit view with student data. 4. If POST request: 4.1 Validate that ID matches student.Id. 4.1.1 If not matching, return NotFound. 4.2 Validate ModelState for required fields. 4.3 If ModelState is valid: 4.3.1 Set student.Role = "Student". 4.3.2 Update student in context. 4.3.3 Save changes to database asynchronously. 4.3.4 Redirect to Index action. 4.4 If ModelState invalid: 4.4.1 Reload programs dropdown with selected value. 4.4.2 Return Edit view with validation errors. 5. End.

Entity Name	StudentsController
Method Name	Delete()
Input	Student ID
Output	Student deletion confirmation or completed deletion
Algorithm	1. Start. 2. Check if user has "Committee" role authorization. 2.1 If not authorized, redirect to Lecturers Index. 3. If GET request: 3.1 Query Students table with Include Program navigation property. 3.2 Find student by ID using FirstOrDefaultAsync. 3.3 If student not found, return NotFound. 3.4 Return Delete confirmation view with student data. 4. If POST request (DeleteConfirmed): 4.1 Find student by ID in Students table. 4.2 If student found:

	4.2.1 Remove student from Students DbSet. 4.2.2 Save changes to database asynchronously. 4.3 Redirect to Index action. 5. End.

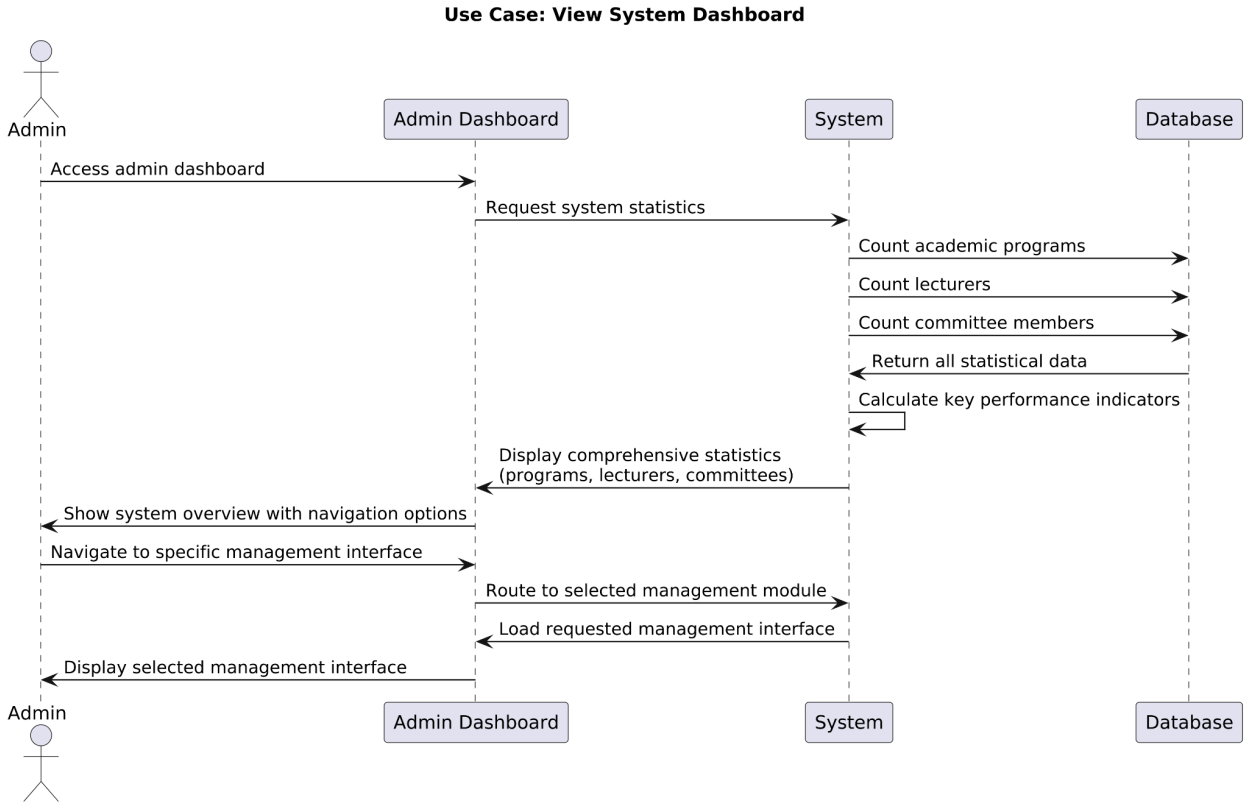

Figure 2.2.4.1.2: Sequence Diagram for <View System Dashboard>

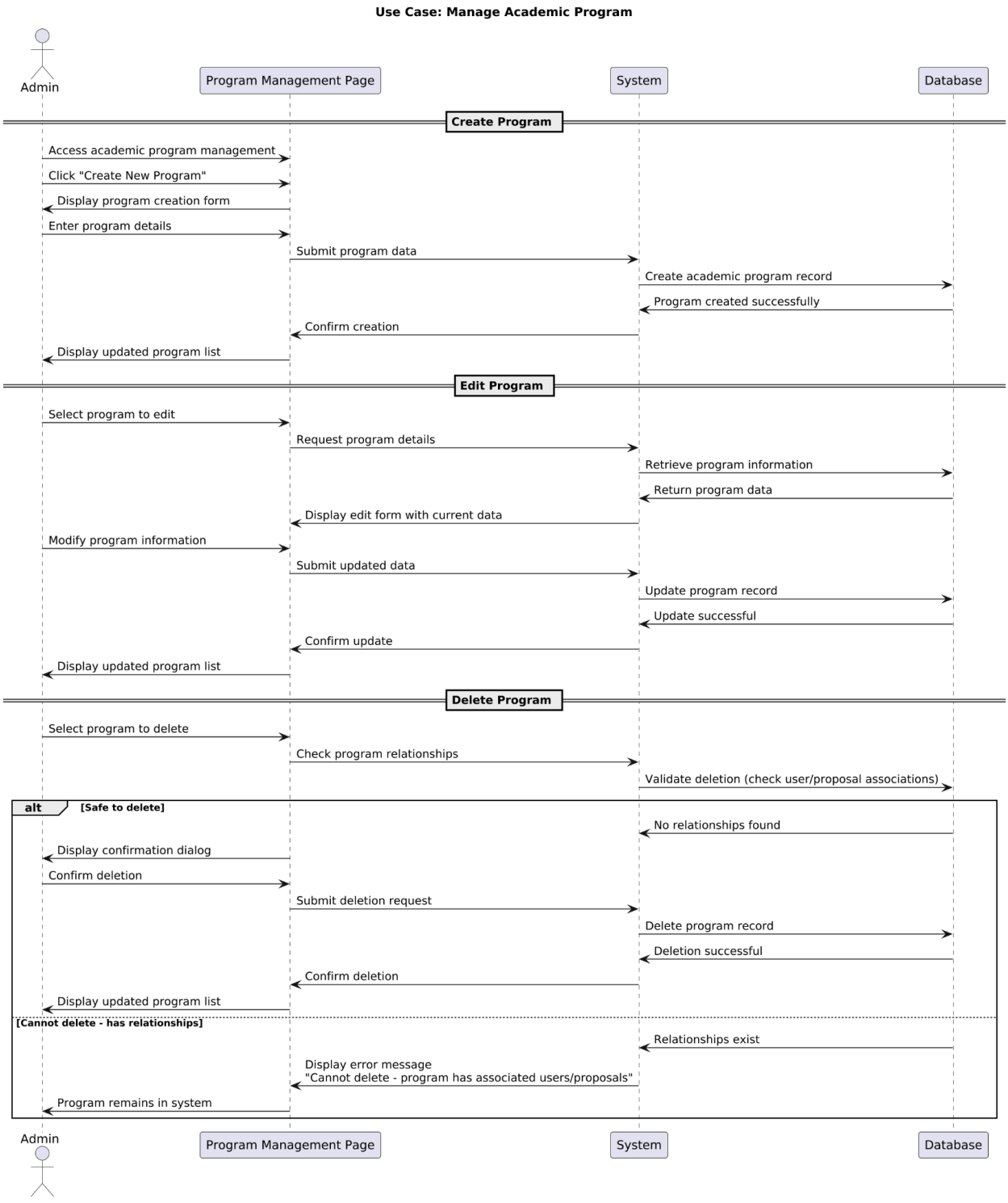

Figure 2.2.4.1.3: Sequence Diagram for <Manage Academic Program>

3.0 Data Design

3.1 Data Description

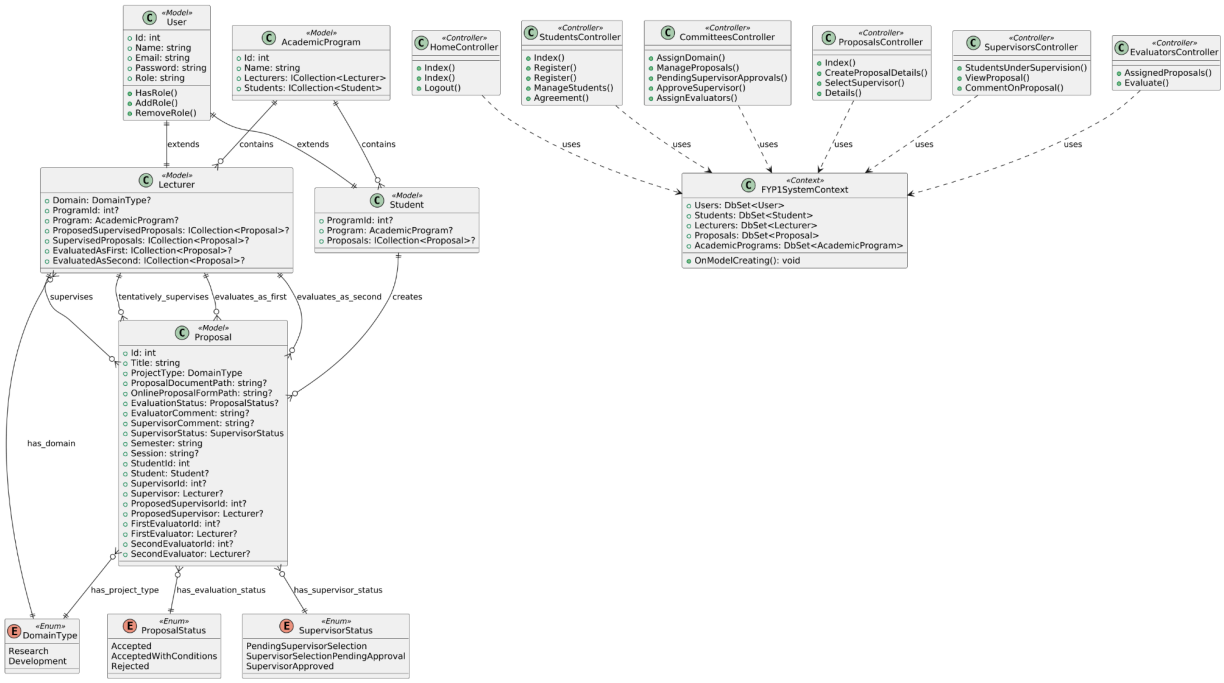

Figure 3.1.1: Complete Class Diagram for <FYP1 System>

Table 3.1.1: Description of Entities in the Database

No.	Entity Name	Description
1.	Users	Table that stores information about all system users and serves as the foundational entity for identity management. It contains core authentication details like email and passwords and most importantly, the assigned role which governs access control across the application.
2.	Lecturers	Table that stores information about all lecturers, extending the base user record with details pertinent to academic staff. This includes their specific Domain (Research/Development), which is crucial for the Committee when assigning qualified evaluators to proposals.
3.	Students	Table that stores information about all students. It links each student to their respective AcademicProgram and establishes them as the

		primary owner of Proposals, making it central to tracking their submission lifecycle and progress.
4.	AcademicPrograms	Table that stores information about all academic programs available within the system, such as “Data Engineering” and “Software Engineering” in this case study.
5.	Proposals	Table that stores information about all proposals. It captures every detail of a proposal's journey, linking the student to their supervisor and evaluators and tracking the workflow via its status, from submission to final approval.

3.2 Data Dictionary

3.2.1 Entity: <Users>

Attribute Name	Data Type	Description
Id	int	The primary key, uniquely identifying each user record.
Name	nvarchar(100)	The full name of the user.
Email	nvarchar(MAX)	The user's unique email address, used for login.
Password	nvarchar(100)	The password for user authentication.
Role	nvarchar(MAX)	A string defining the user's role (e.g., "Admin", "Student", "Supervisor", "Evaluator", "Committee").

3.2.2 Entity: <Lecturers>

Attribute Name	Data Type	Description
Id	int	The primary key, which is also a foreign key to the Users table, linking the lecturer to their user account.
Domain	int	An integer representing the lecturer's primary domain. Nullable.
ProgramId	int	A foreign key linking to the AcademicPrograms table, associating the lecturer with a specific program. Nullable.

3.2.3 Entity: <Students>

Attribute Name	Data Type	Description
Id	int	The primary key, which is also a foreign key to the Users table, linking the student to their user account.
ProgramId	int	A foreign key linking to the

		AcademicPrograms table, associating the student with their enrolled program. Nullable.

3.2.4 Entity: <AcademicPrograms>

Attribute Name	Data Type	Description
Id	int	The primary key, uniquely identifying each academic program.
Name	nvarchar(MAX)	The full name of the academic program, such as “Data Engineering” and “Software Engineering” in this case study.

3.2.5 Entity: <Proposals>

Attribute Name	Data Type	Description
Id	int	The primary key, uniquely identifying each proposal.
Title	nvarchar(200)	The title of the project proposal.
ProjectType	int	An integer representing the project's domain.
ProposalDocumentPath	nvarchar(MAX)	The file path for the uploaded proposal PDF document. Nullable.
OnlineProposalFormPath	nvarchar(MAX)	The file path for the online proposal form data. Nullable.
EvaluationStatus	int	An integer representing the outcome from the evaluators. Nullable.
EvaluatorComment	nvarchar(1000)	Stores detailed feedback and comments from the assigned evaluators. Nullable.
SupervisorComment	nvarchar(MAX)	Stores preliminary comments and feedback from the supervisor. Nullable.
SupervisorStatus	int	An integer representing the status of supervisor selection.
Semester	nvarchar(MAX)	The academic semester in which the proposal was submitted.

Session	nvarchar(MAX)	The academic session in which the proposal was submitted. Nullable.
StudentId	int	Foreign key linking to the Students table, identifying the owner of the proposal.
SupervisorId	int	Foreign key linking to the Lecturers table, identifying the officially approved supervisor. Nullable.
ProposedSupervisorId	int	Foreign key linking to the Lecturers table, identifying the student's preferred supervisor before approval. Nullable.
FirstEvaluatorId	int	Foreign key linking to the Lecturers table, identifying the first assigned evaluator. Nullable.
SecondEvaluatorId	int	Foreign key linking to the Lecturers table, identifying the second assigned evaluator. Nullable.