SOFTWARE DESIGN DOCUMENT (SDD)

PRESENTED BY:	PRESENTED TO:	DATE:
PathSync Sdn. Bhd.	Faculty of Computing UMPSA	22/12/2024

TEAM MEMBERS

Chief Executive Officer	Chief Business Analyst	Chief Developer	Software Quality Assurance
KEH BAN NING CB23137	CHONG MANZIE CB23115	AZMI WAN NURHALIZA CB22004	NUR ALYA SYAKIRAH BINTI NASARUDIN CB22141

Submitted to the Software Design Workshop
(BCS2343) Bachelor of Computer Science (Software Engineering)
with Honours
Faculty of Computing, Universiti Malaysia Pahang Al-Sultan Abdullah (UMPSA)

DOCUMENT APPROVAL

	Name	**Date**
Authenticated by: [illegible] ______________ Project Leader	KEH BAN NING	22/12/2024
Approved by: ______________ Project Manager		
Approved by: ______________ Client		

Software : Google Drive, Microsoft Word, Draw.io

Archiving Place : SDD Document & GDrive Link System

Copies Available : 1

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

1. SDD - Software Design Document
2. FYP – Final Year Project
3. PSYNC – PathSync Sdn. Bhd
4. REQ - Requirement
5. ID - Identification number
6. MVC – Model View Controller
7. PK – Primary Key
8. FK – Foreign Key

1. INTRODUCTION

The Final Year Project (FYP) is a critical component of the undergraduate curriculum for students in the Faculty of Computing requiring them to apply their accumulated knowledge and skills to solve real-world problems. To ensure the success of these projects, students must secure a supervisor who can guide them throughout the process. However, the current method of selecting and assigning supervisors often proves cumbersome with challenges such as managing lecturer quotas and scheduling appointments.

To address these issues, the "FYP Supervisor Hunting" system is proposed. This system is designed to streamline and digitize the pre-FYP phase, providing an efficient platform for students, lecturers, and the FYP Coordinator. By leveraging technology, the application facilitates smoother communication, simplifies appointment scheduling and ensures fair and transparent supervisor allocations while keeping all stakeholders informed through timely updates and notifications.

This report outlines the objectives, requirements, and detailed module descriptions of the proposed system, emphasizing its role in enhancing the management and user experience of the FYP supervisor selection process.

1.1 PURPOSE

The purpose of this Software Design Document (SDD) is to provide a detailed reference for the development of the *FYP Supervisor Hunting* system. The system is designed to streamline the process of managing Final Year Project (FYP) supervision for the Faculty of Computing. This document establishes a clear understanding between stakeholders by outlining the system's functional requirements, quality attributes, and design decisions. The system aims to address key challenges, such as managing user information, scheduling appointments, applying for project topics or titles, and handling supervision quotas. By automating these tasks, the system reduces inefficiencies, prevents scheduling conflicts, and ensures accurate supervision management.

The Software Design Document (SDD) also provides necessary information to provide description of the details for the software and system to be built. The developers team members are also using this document as their future reference.

1.2 SYSTEM IDENTIFICATION

ID Type	System ID
Identification Number	FYP-01-2024-PSYNC
System Title	FYP Supervisor Hunting System
System Abbreviation	FYP Supervisor Hunting System FYP Final Year Project
Version Number	01 01 Version 1
Release Number	2024 2024 Year 2024
Developer / Company Name	PSYNC PSYNC PathSync Sdn. Bhd.

Table 1.2.1 System Identification Table

ID Type	Document ID
Identification Number	SDD_FYP_2024
Document Abbreviation	SDD FYP Software Design Document
System Abbreviation	FYP Supervisor Hunting System FYP Final Year Project
Release Number	2024 2024 Year 2024

Table 1.2.2 Document Identification Table

ID Type	Requirement ID
Identification Number	SDD_REQ_001_01
Document Abbreviation	SDD SDD Software Design Document

Requirement Abbreviation	REQ REQ Requirement
Use Case Number	01 001 Use Case
Requirement Number	01 01 Requirement

Table 1.2.3 Requirement Identification Table

1.3 SYSTEM OVERVIEW

The **Final Year Project Supervisor Hunting System** is a web-based platform developed to streamline and organize the management of Final Year Project (FYP) courses, which are compulsory for undergraduate students in the Faculty of Computing. The system addresses the challenge of supervisor selection for FYP1 and FYP2 courses, undertaken in the final two semesters.

Students can explore a list of available lecturers, check their supervision quotas, and submit project proposals. Each lecturer's quota is predefined by the faculty to ensure a balanced workload. Once a student submits a proposal, supervisors can review, approve, or reject the request based on their interest and remaining availability. The system also includes features for faculty management to monitor the supervisor-student assignment process, manage quotas, and generate reports on project progress, student requests, and supervision data. By integrating these functionalities, the system enhances efficiency, transparency, and resource optimization, ensuring a smoother and more organized FYP process for both students and faculty members.

Module 1 : Manage User

The Manage User module allows FYP coordinators to manage user registration and management of students and teachers efficiently. Coding registration can be done by creating the code into controller and next is to generate usernames and temporary passwords, which will be provided to users on the first login. For security reasons, users are required to change their temporary password the first time they log in. This module also allows FYP coordinators to create and print reports based on registered users' programs and assign instructors to students' study groups. The system streamlines user onboarding, ensures secure access, and offers powerful reporting and management capabilities.

Module 2 : Manage Appointment

The Manage Appointment Module facilitates efficient scheduling between students and lecturers by enabling students to search for prospective supervisors, and request appointments by selecting a preferred date and time. Lecturers can respond to these requests by approving or rejecting. Both students and lecturers can cancel appointments before the scheduled date, ensuring flexibility and effective communication.

Module 3 : Manage Topic/Title

The Apply Topic/Title module provides an interactive platform for students to apply for their Final Year Project (FYP) topics from their prospective supervisors. Supervisors can post proposed FYP topics on the system, and students can view these topics either by specific lecturers or as a complete list. Students can apply for a topic of their choice or propose their own topics to the respective supervisor. Once a student applies, the system updates the status of the selected topic, ensuring no duplication of applications.

Supervisors have the capability to review, accept, or reject applications with detailed remarks to guide the students. The system automatically notifies students about the supervisor's decision, keeping them informed throughout the process. Additionally, the supervisor's quota is updated in real-time whenever an application is approved, ensuring accurate management of available slots. This module not only streamlines the topic application process but also facilitates better communication between students and supervisors, ensuring a transparent and efficient workflow for managing FYP topics.

Module 4 : Manage Timeframe and Quota

The Manage Timeframe and Quota module is essential for organizing and streamlining the supervisor hunting process in the "FYP Supervisor Hunting" system. It enables the FYP Coordinator to set and update the start and end dates for the supervisor hunting phase and manage the supervision quotas for each lecturer. These quotas, which are accessible to students and lecturers, can be updated throughout the semester with all changes promptly communicated through automated notifications. The system ensures transparency by providing real-time quota information and sending reminders about impending deadlines. Additionally, the FYP Coordinator can generate and print detailed reports on lecturer quotas, supporting effective record-keeping and analysis. This module enhances efficiency and transparency in managing the supervisor assignment process.

1.4 REFERENCES

1. Datensen. (2023, July 24). *One-to-many relationships | ER diagram | Luna Modeler*. Datensen. https://www.datensen.com/blog/er-diagram/one-to-many-relationships/

2. Sheldon, R. (2023, September 12). *model-view-controller (MVC)*. WhatIs. https://www.techtarget.com/whatis/definition/model-view-controller-MVC#:~:text=In%20programming%2C%20model%2Dview%2D,a%20specific%20set%20of%20tasks.

3. GeeksforGeeks. (2023, November 6). *Entity in DBMS*. GeeksforGeeks. https://www.geeksforgeeks.org/entity-in-dbms/

4. Fujiwara, L. (2023, December 22). ViewModels : A Simple Example - Android Developers - Medium. *Medium*. https://medium.com/androiddevelopers/viewmodels-a-simple-example-ed5ac416317e

5. Rozali, M. A. Z. B. (2021). FYP Supervisor Allocation Management System [Thesis]. In *Bachelor of Information Technology (Hons)*. Universiti Teknologi PETRONAS.

2. DATA DESIGN

2.1 ENTITY RELATIONSHIP DIAGRAM (ERD)

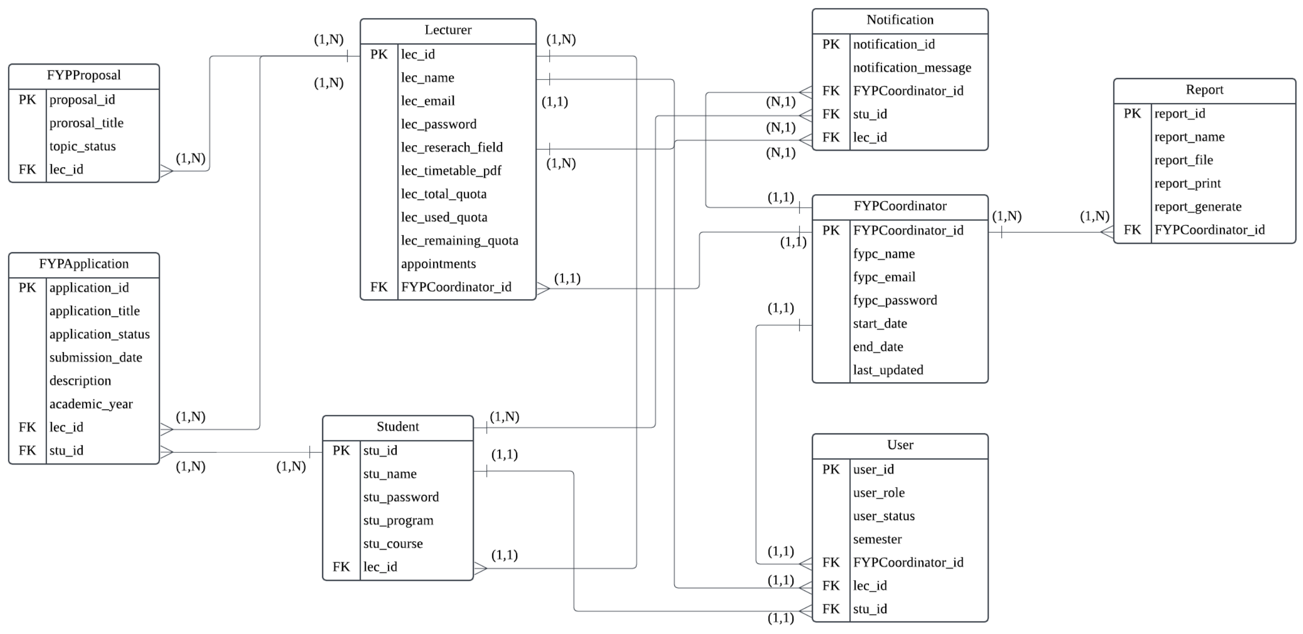


Figure 2.1 Entity Relationship Diagram (ERD) for FYP Supervisor Hunting System

2.2 DATA DICTIONARY

2.2.1 User

Field Name	**Description**	**Data Type**	**Constraint**
user_id	Unique ID for the user	INT	PK
user_role	Role of the user (FYP Coordinator, Student , Lecturer)	VARCHAR(50)	NOT NULL
user_status	Status of the user	VARCHAR(50)	NOT NULL
semester	Current semester	INT	NOT NULL
FYPCoordinator_id	Linked FYP Coordinator ID	INT	FK
lec_id	Linked Lecturer ID	INT	FK
stu_id	Linked Student ID	INT	FK

Table 2.2.1 Data Dictionary for User

2.2.2 Lecturer

Field Name	**Description**	**Data Type**	**Constraint**
lec_id	Lecturer's unique identifier	VARCHAR(10)	PK
lec_name	Lecturer's full name	VARCHAR(50)	NOT NULL
lec_email	Lecturer's email address	VARCHAR(50)	NOT NULL, UNIQUE
lec_password	Lecturer's password for system login	VARCHAR(20)	NOT NULL
lec_research_field	Lecturer's area of research expertise	VARCHAR(50)	NULL
lec_timetable_pdf	Timetable uploaded by the lecturer (file link)	VARCHAR(100)	NULL
lec_total_quota	Total number of students lecturer can supervise	INT	DEFAULT 0
lec_used_quota	Number of students currently supervised	INT	DEFAULT 0
lec_remaining_quota	Remaining available quota	INT	GENERATED AS (lec_total_quota - lec_used_quota) STORED
appointments	Number of appointments booked	INT	DEFAULT 0
FYPCoordinator_id	Foreign key referencing FYP Coordinator	VARCHAR(10)	FK

Table 2.2.2 Data Dictionary for Lecturer

2.2.3 Student

Field Name	**Description**	**Data Type**	**Constraint**
stu_id	Student's unique identifier	VARCHAR(10)	PK
stu_name	Student's full name	VARCHAR(50)	NOT NULL
stu_password	Student's password for system login	VARCHAR(20)	NOT NULL
stu_program	Program enrolled by student	VARCHAR(30)	NOT NULL
stu_course	Current course title	VARCHAR(50)	NULL
lec_id	Foreign key referencing assigned lecturer	VARCHAR(10)	FK

Table 2.2.3 Data Dictionary for Student

2.2.4 FYP Coordinator

Field Name	**Description**	**Data Type**	**Constraint**
FYPCoordinator_id	Unique ID for FYP Coordinator	INT	PK
fypc_name	Name of the FYP Coordinator	VARCHAR(100)	NOT NULL
fypc_email	Email of the FYP Coordinator	VARCHAR(100)	NOT NULL, UNIQUE
fypc_password	Password for the FYP Coordinator	VARCHAR(255)	NOT NULL
start_date	Start date of supervisor hunting phase	DATE	NOT NULL
end_date	End date of supervisor hunting phase	DATE	NOT NULL
last_updated	Timestamp of the last update	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP

Table 2.2.4 Data Dictionary for FYP Coordinator

2.2.5 FYP Proposal

Field Name	**Description**	**Data Type**	**Constraint**
proposal_id	Unique identifier for FYP proposal	VARCHAR(10)	PK
proposal_title	Title of the FYP proposal	VARCHAR(100)	NOT NULL
topic_status	Current status of the proposal topic	VARCHAR(20)	DEFAULT ‘PENDING’

lec_id	Foreign key referencing supervising lecturer	VARCHAR(10)	FK

Table 2.2.5 Data Dictionary for FYP Proposal

2.2.6 FYP Application

Field Name	Description	Data Type	Constraint
application_id	Unique identifier for FYP application	VARCHAR(10)	PK
application_title	Title of the application submitted	VARCHAR(100)	NOT NULL
application_status	Current status of the application	VARCHAR(20)	DEFAULT 'PENDING'
submission_date	Date when the application was submitted	DATE	NOT NULL
description	Brief description of the application	VARCHAR(200)	NULL
academic_year	Academic year of the application	VARCHAR(9)	NOT NULL
lec_id	Foreign key referencing supervising lecturer	VARCHAR(10)	FK
stu_id	Foreign key referencing submitting student	VARCHAR(10)	FK

Table 2.2.6 Data Dictionary for FYP Application

2.2.7 Notification

Field Name	Description	Data Type	Constraint
notification_id	Unique ID for the notification	INT	PK
notification_message	Message content of the notification	VARCHAR(255)	NOT NULL
FYPCoordinator_id	FYP Coordinator linked to notification	INT	FK
stu_id	Student linked to notification	INT	FK
lec_id	Lecturer linked to notification	INT	FK

Table 2.2.7 Data Dictionary for Notification

2.2.8 Report

Field Name	Description	Data Type	Constraint

report_id	Unique ID for the report	INT	PK
report_name	Name of the report	VARCHAR(100)	NOT NULL
report_file	File associated with the report	VARCHAR(255)	NOT NULL
report_print	Indicates if the report is printed	BOOLEAN	DEFAULT FALSE
report_generate	Indicates if the report is generated	BOOLEAN	DEFAULT FALSE
FYPCoordinator_id	Linked FYP Coordinator ID	INT	FK

Table 2.2.8 Data Dictionary for Report

3. GENERAL ARCHITECTURE

We chose the MVC (Model-View-Controller) design pattern for this project because it promotes separation of concerns, which helps improve maintainability and scalability. By dividing the application into three distinct components—Model, View, and Controller—each part focuses on its specific responsibility, allowing for cleaner code and easier updates. The Model handles data and business logic, the View manages the user interface, and the Controller processes user input and coordinates between the Model and View.

However, we also decided to divide the system by modules to better align with the project's structure, where distinct functionalities such as Manage Appointment, Manage Users, Apply Topic/Title, and Manage Time Frame are key. This approach allows us to focus on individual features, ensuring that each module has its own Model, View, and Controller components, thus maintaining modularity and flexibility. Dividing by modules makes it easier to manage complex functionality and scale the project in the future, while still adhering to the principles of MVC within each module.

3.1 MODULE 1: MANAGE USER (CHONG MANZIE CB23115)

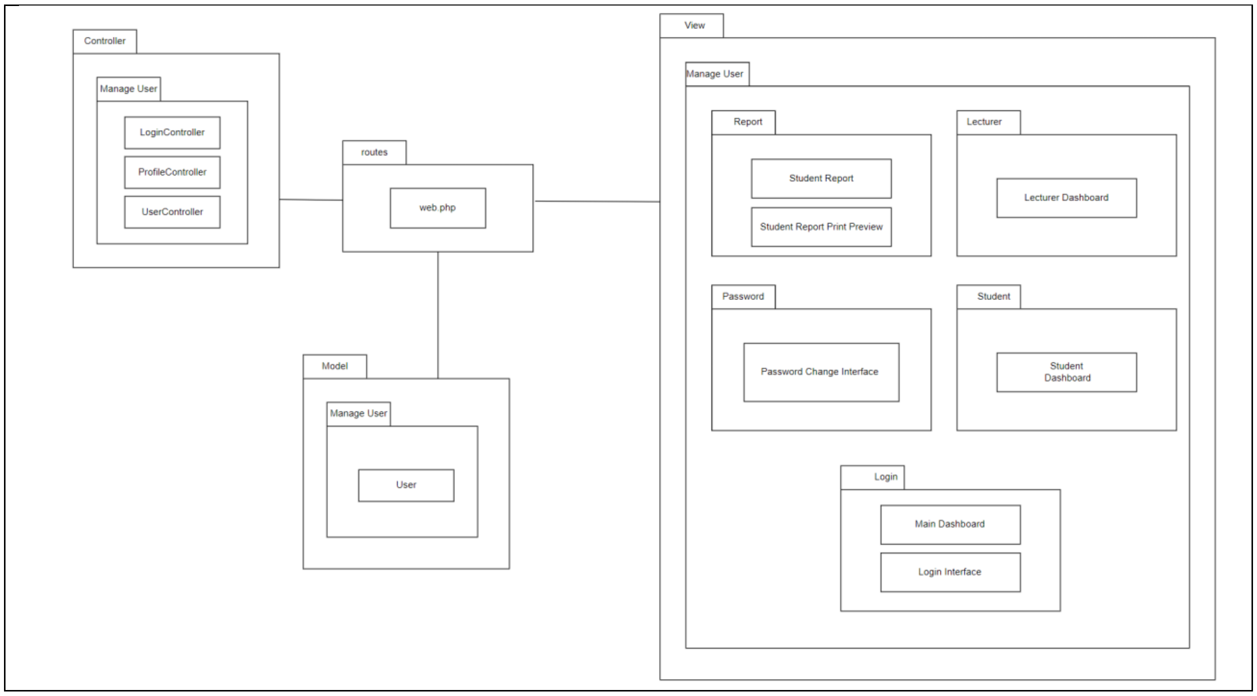


Figure 3.1 MVC for Module 1: Manage User

3.1.1 Package Model [SDD-REQ-101]

Class Name	Description
User	This User model in Laravel manages users with authentication, notifications, and secure attributes. It uses mass-assignment for key fields, hides sensitive data, and auto-casts attributes like email_verified_at and password. The boot method ensures a unique users_id is generated for new users, streamlining user management with security and customization.

3.1.2 Package View [SDD-REQ-102]

Class Name	Description
Student Report	This **Student Report** page generates a student's details, including ID, name, email, courses, and report date, using Laravel's Blade syntax. It features a simple layout with a header and a link to navigate back to the student dashboard.
Student Report Print Preview	This **Print Student Report** page provides a print-friendly preview of a student's details, including ID, name, email, courses, and report date, styled with a clean and centered layout. It features a "Print Report" button that triggers the browser's print functionality for easy hardcopy generation.
Password Change Interface	This **Password Change** page allows users to update their passwords securely. It includes fields for the current password, new password, and confirmation of the new password, with validation error messages displayed if needed. Success or error alerts are shown based on the operation outcome, and a button redirects users back to their respective dashboards (lecturer or student). The page is styled using Bootstrap for a clean, responsive design.
Lecturer Dashboard	The **Lecturer Dashboard** page displays a list of lecturers with their ID, name, email, and research fields. It includes buttons for adding research fields like AI, Software, and Data Science, which trigger success alerts when clicked. Users can also log out or navigate to a password change page. The page is designed to manage and interact with lecturer data in a simple, functional layout.
Student Dashboard	The **Student Dashboard** page provides a user-friendly interface for managing student details, generating reports, and changing passwords. It displays a list of students with their ID, name, email, and course list, along with buttons to generate or print reports. The page also includes options for changing the password and logging out. The layout is styled with Bootstrap for responsiveness, featuring a clean header, action buttons, and a footer for branding.

Main Dashboard	The **Main Dashboard** introduces the platform for managing Final Year Project (FYP) activities. It features a central jumbotron with a welcoming message, describing the platform's purpose. Users can navigate to the Student or Lecturer dashboards, and there is a login button for authentication. The page is styled with Bootstrap for a clean, responsive design.
Login Interface	The **Login Interface** allows users to authenticate by entering their User ID and password. It includes a clean, centered login form with Bootstrap styling. Error messages are displayed if login fails, and validation errors are shown for any incorrect input. The form is structured with a table layout and includes a "Login" button for submission.

3.1.3 Package Controller [SDD-REQ-103]

Class Name	Description
LoginController	The **LoginController** handles user authentication by validating the login credentials (User ID and password). It checks if the user exists in the database and verifies the password using hash comparison. Upon successful authentication, the user is stored in the session and redirected to either the lecturer or student dashboard based on their role. If authentication fails, an error message is returned.
ProfileController	The **ProfileController** handles the password change functionality for users. It provides two main actions: showing the password change form and updating the password. The showChangePasswordForm method returns the view for password change, while the updatePassword method validates the current and new passwords, checks the correctness of the current password, updates the password if valid, and redirects the user to their respective dashboard with a success message. If the current password is incorrect, an error message is shown.
UserController	The **UserController** handles various user-related functionalities, including inserting dummy user data into the database, displaying student and lecturer dashboards, and

	generating or printing reports for students. It provides methods for adding users with hashed passwords, fetching and displaying student and lecturer data on their respective dashboards, and generating student reports in both HTML and printable formats. The controller ensures that only valid users are fetched and reports are generated based on student data, returning views that display the necessary information.

3.2 MODULE 2: MANAGE APPOINTMENT (KEH BAN NING CB23137)

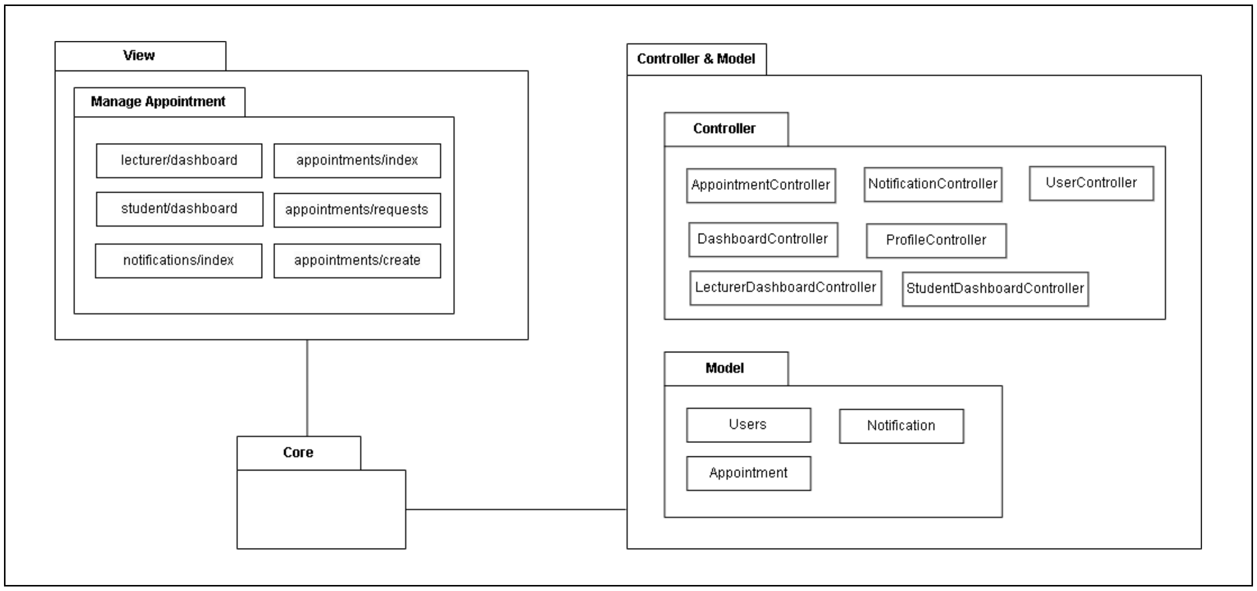


Figure 3.2 MVC for Module 2: Manage Appointment

3.2.1 MODEL

3.2.1.1 Package Model [SDD-REQ-201]

Class Name	Description
Users	This model stores user details, including name, email, password, role, and program, for lecturers, students, and FYP Coordinators.
Appointment	This model holds information about appointments between students and lecturers, such as student ID, lecturer ID, appointment date, time, and status.
Notification	This model stores notification messages related to appointments.

3.2.2 VIEW

3.2.2.1 Package View [SDD-REQ-202]

Class Name	Description
lecturer/dashboard	A dashboard interface for lecturers to approve or decline student appointment requests, showing the requested appointment details such as date, time, status, and the student's name.
student/dashboard	A dashboard interface for students to schedule new appointments with lecturers by selecting the desired date and time, as well as to view the status of their appointments.
notifications/index	An interface that displays a list of notifications for the user, showing the message and the time since creation. It includes options to mark notifications as read and delete them. Unread notifications are highlighted, and appropriate buttons are provided for each action.
appointments/index	• Coordinator: A coordinator manages and oversees the appointments, ensuring students and lecturers are matched correctly. They can view both student and lecturer details and monitor the status of appointments. • Lecturer: A lecturer handles appointment requests from students. They can view student details, approve or reject appointments, and update the status of each appointment.

	• Student: A student can request appointments with lecturers and view their own appointments.
appointments/requests	An interface that displays a list of pending appointment requests. If no pending requests exist, a message is shown. If there are pending appointments, they are displayed in a table with columns for the row number, student name, appointment date, time, status, and action buttons. The action buttons allow the coordinator or lecturer to accept or reject each appointment by updating the appointment's status.
appointments/create	An interface that allows students to request an appointment by selecting a lecturer, date, and time. The form includes a hidden field for the student's ID and submits the data to store the appointment request. A submit button is provided to complete the request.

3.2.3 CONTROLLER

3.2.3.1 Package Controller [SDD-REQ-203]

Class Name	Description
AppointmentController	The controller handles appointment-related actions: displaying appointments based on user roles, storing new requests, fetching pending requests for lecturers, showing appointment details, updating status, and deleting appointments.

DashboardController	The controller handles the dashboard data, displaying the total number of users, appointments, and pending appointments.
LecturerDashboardController	The controller handles the lecturer's dashboard, displaying the lecturer's appointments, pending requests, and the total number of students with approved appointments.
StudentDashboardController	The controller handles the student's dashboard, displaying the student's appointments, current supervisor (if approved), and the count of pending requests.
NotificationController	The controller handles managing notifications for a specific user, including displaying notifications, marking them as read, and deleting them.
ProfileController	The controller handles displaying the profile page for the authenticated user.
UserController	The controller handles managing users, including creating, updating, displaying, and deleting user records, as well as validating input data for user creation and updates.

3.3 MODULE 3: APPLY TOPIC/ TITLE (AZMI WAN NURHALIZA CB22004)

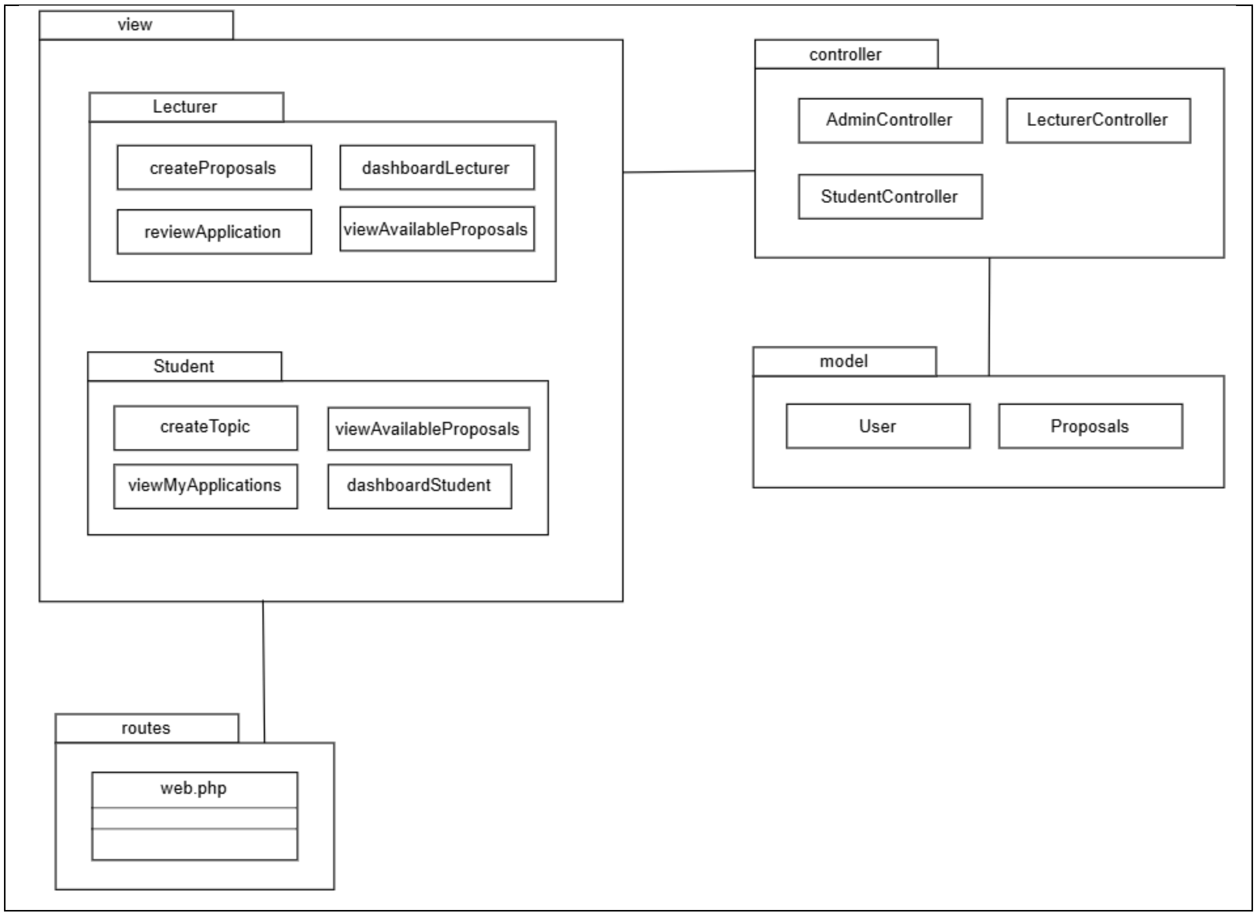


Figure 3.3 MVC for Module 3: Apply Topic/ Title

3.3.1 MODEL

3.3.3.1 Package Model [SDD-REQ-301]

Class Name	Description
Proposals	Manages all data related to Final Year Project (FYP) proposals. This includes proposal details such as title, status, quota, and associated lecturer or student information. It also handles status updates for proposals (e.g., approved, rejected).
User	Manages user data, including student, lecturer, and admin credentials, roles, and related interactions within the system. It ensures user authentication and authorization for role-based functionalities.

3.3.2 VIEW

3.3.2.1 Package View [SDD-REQ-302]

Class Name	Description
CreateTopic	Allows students to submit their own FYP proposals by providing necessary details, including title, description, and selecting a lecturer.
ViewAvailableProposals	Displays a list of available FYP proposals for students to browse and apply for. Lecturers can also view their submitted proposals.
ViewMyApplications	Provides students with a detailed view of their submitted proposals, including the application status (e.g., approved, pending, or rejected).
CreateProposals	Allows lecturers to create and submit new FYP proposals for students to apply for. This includes managing quotas and uploading schedules.
ReviewApplication	Enables lecturers to view and review applications submitted by students. Lecturers can approve or reject applications and update the status accordingly.
DashboardLecturer	Displays the dashboard interface specific to lecturers, showing options for proposal management.
DashboardStudent	Displays the dashboard interface specific to students, showing options for viewing and managing their proposals.

3.3.2 CONTROLLER

3.3.2.1 Package Controller [SDD-REQ-303]

Class Name	Description
AdminController	Manages admin specific operations, including viewing, creating, editing, and deleting user accounts.
LecturerController	Handles lecturer functionalities such as creating proposals, viewing available proposals, and reviewing applications submitted by students.
StudentController	Manages student actions, including viewing available proposals, submitting proposals, and viewing the status of submitted proposals.

3.4 MODULE 4: MANAGE TIME FRAME AND QUOTA (NUR ALYA SYAKIRAH BINTI NASARUDIN CB22141)

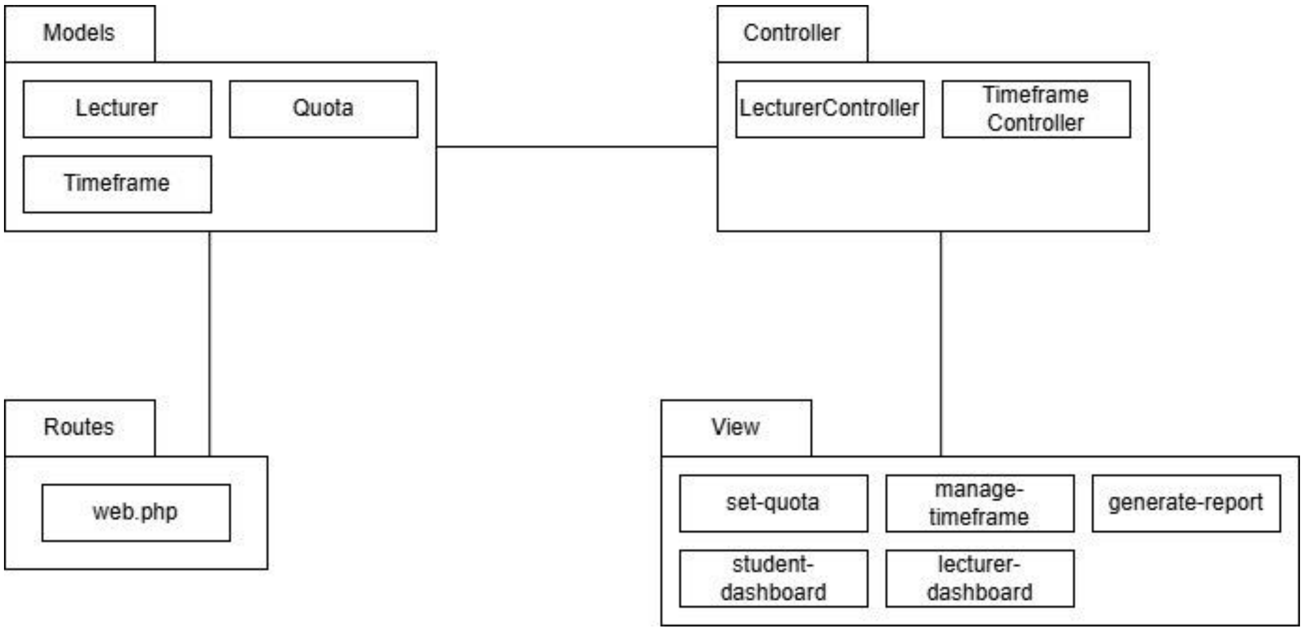


Figure 3.4 MVC for Module 4: Manage Time Frame and Quota

3.4.1 MODEL

3.4.1.1 Package Model [SDD-REQ-401]

Class Name	Description
Lecturer	The Lecturer model represents the lecturers in the system. It contains information about each lecturer, such as their name, assigned quotas, and the current quota usage. The model provides a calculated attribute remaining_slots, which computes the remaining available slots by subtracting the current quota from the maximum quota. This model is used for managing and displaying lecturer-related data, as well as updating quota information.
Quota	The Quota model represents the quotas for each lecturer, particularly for each semester. It stores the total quota, used quota, and remaining quota for a given lecturer. The

	model defines the relationship with the Lecturer model, allowing easy access to the associated lecturer for each quota record. It uses the `fillable` property to define which fields are mass assignable.
Timeframe	The Timeframe model represents the start and end dates for the supervisor hunting timeframe. It stores the timeframe in the database, and these dates are essential for managing the deadlines. The model defines the fillable property for the start and end dates and stores the data in the timeframes table. This model is used to manage the timeframe data that is essential for the system's operations.

3.4.2 CONTROLLER

3.4.2.1 Package Controller [SDD-REQ-402]

Class Name	Description
LecturerController	The LecturerController handles actions related to lecturers in the system. It is responsible for managing and updating the lecturer's quota, generating reports, and interacting with the Lecturer and Quota models. The controller allows the system to display and manage lecturer information based on various criteria, including semester selection. It also facilitates the storage of quota updates and allows for the display of lecturer dashboards with relevant quota information.
TimeframeController	The TimeframeController manages the functionality related to setting and saving the supervisor hunting timeframe. It handles the display of the "Manage Timeframe" page and the saving of start and end dates for the timeframe. The controller stores the saved dates in the session, ensuring that they are available for subsequent requests. Additionally, it controls the visibility of a popup notification after saving the timeframe. This class provides the necessary logic for the timeframe form and the related session handling.

3.4.3 VIEW

3.4.3.1 Package View [SDD-REQ-403]

Class Name	Description
set-quota	The set-quota view is responsible for displaying and managing the quotas assigned to lecturers. This page allows the user to select a specific semester, view the lecturers associated with that semester, and update their quotas. The page includes a form where the user can input new quota values for each lecturer and save these changes. The view is also used to display a list of lecturers and their current quotas, providing an overview of the quota distribution across lecturers.
manage-timeframe	The manage-timeframe view is used to manage the supervisor hunting timeframe. This page allows the user to set the start and end dates for the timeframe. Once the dates are entered and the form is submitted, the data is saved to the session, and a success message is displayed.
generate-report	The generate-report view is used to generate reports about the lecturers and their quotas for a selected semester. The page allows the user to choose a specific semester and then displays relevant data for the lecturers in that semester. It provides insights into the total quota, used quota, and remaining quota for each lecturer. The view might include options to export or download the generated report in different formats.
lecturer-dashboard	The lecturer-dashboard view provides a detailed dashboard for a specific lecturer. This page displays the lecturer’s personal information, including their name, assigned semester, and quota data (total, used, remaining). The dashboard allows users to view their quotas for different semesters, providing an overview of the quota management. The page can also display notifications.
student-dashboard	The student-dashboard view provides a personalized dashboard for students, displaying a list of lecturers and their quotas based on the selected semester. This view allows students to see which lecturers are available and their remaining slots for

	supervisor hunting. It also includes functionality for students to select a specific semester and view the available lecturers, helping them make informed decisions when selecting a supervisor.

3.5 PACKAGE RELATIONSHIP

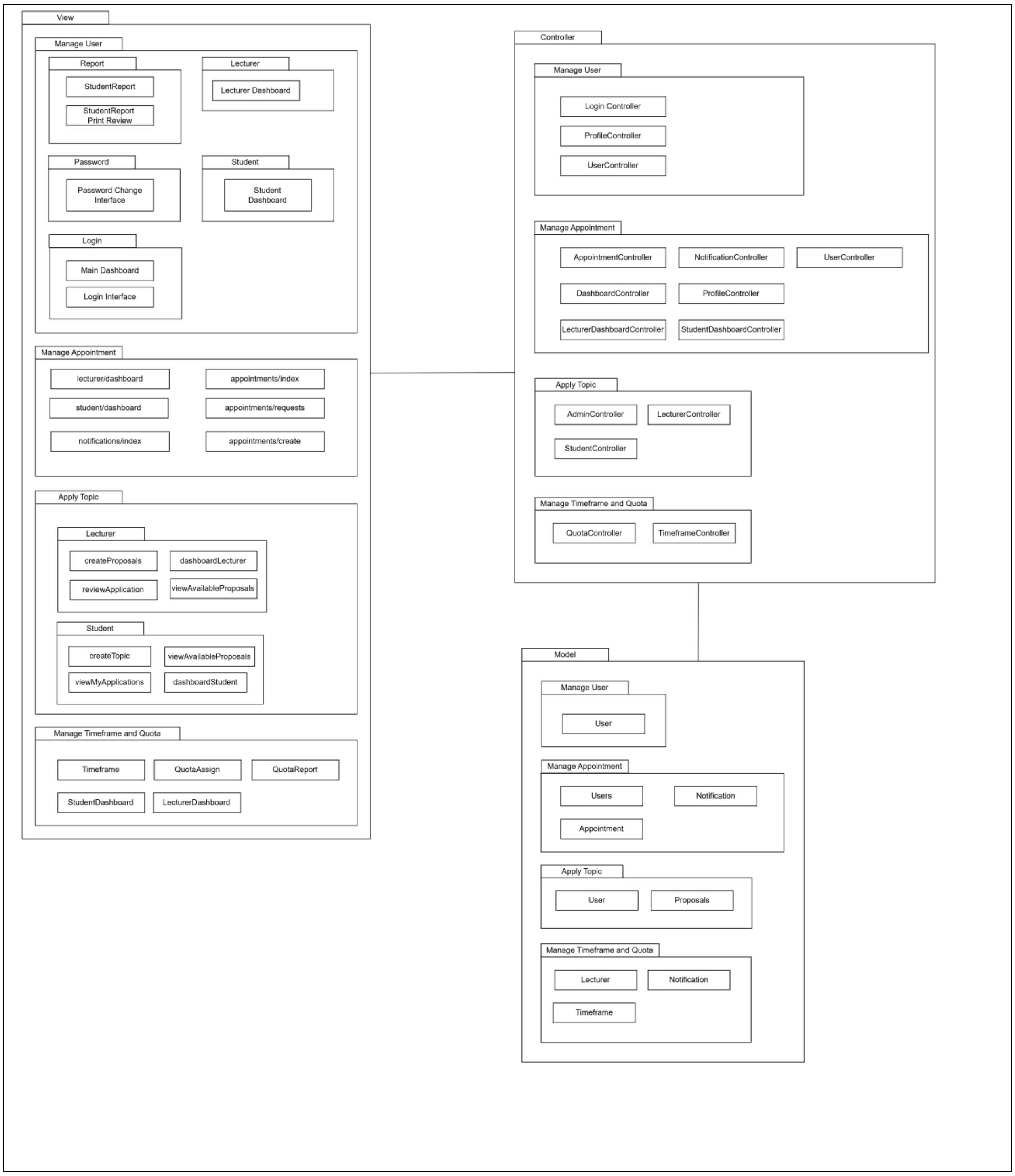
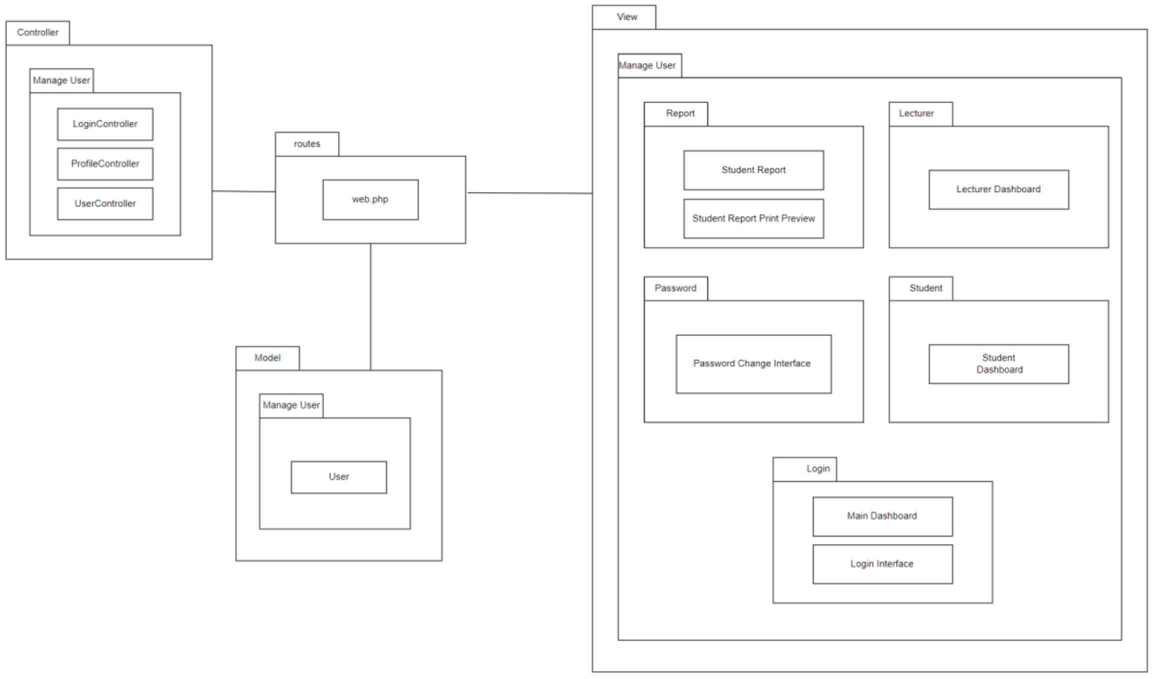


Figure 3.5 Model View Controller (MVC) Package Relationship for FYP Supervisor Hunting System

4. DETAIL DESIGN

4.1 MODULE 1: MANAGE USER (CHONG MANZIE CB23115)



4.1.1 User [SDD-REQ-101]

Class Type	Eloquent Model, extends Authenticatable	
Responsibility	Represents the User entity in the application and interacts with the corresponding database table. It handles authentication and user-related operations such as generating a unique user ID, managing user attributes, and hashing passwords.	
Attributes	Attributes Name	Attributes Type
	users_id	String
	name	String
	email	String
	role	String
	password	String(hashed)
	course_list	String
Methods	Method Name	Description

	1. boot() 2. casts()	1. This method is part of the model lifecycle and is used to handle model events. It ensures that before a user is created, if the users_id is empty, a random 8-character unique ID is generated for the user. 2. Defines the attributes that should be cast to specific data types.
Algorithm	1. boot() Check if users_id is empty. If true, generate a unique users_id using Str::random(8) (e.g., cb12345). 2. casts() Cast email_verified_at to datetime. Cast password to hashed.	

4.1.2 LoginController [SDD-REQ-102]

Class Type	Controller	
Responsibility	Manages the authentication process of users, verifies user credentials, and redirects them to the appropriate dashboard based on their role (lecturer or student).	
Attributes	Attributes Name	Attributes Type
	users_id role password	String String String

Methods	Method Name	Description
	1. authenticate(Request $request)	1. This method handles the authentication process. It validates the incoming request (checking that the users_id and password are provided), attempts to find the user by users_id, and then checks if the password matches. If successful, the user is stored in the session and redirected to the appropriate dashboard. If authentication fails, an error message is returned.
Algorithm	1. authenticate(Request $request) **Step 1**: Validate the request, ensuring both users_id and password are present. **Step 2**: Find the user by the provided users_id in the database. **Step 3**: Check if the user exists and if the provided password matches the stored (hashed) password. **Step 4**: If authentication is successful: • Store the authenticated user in the session. • Redirect the user to the appropriate dashboard based on their role (lecturer or student). **Step 5**: If authentication fails: • Redirect back with an error message indicating invalid credentials.	

4.1.3 ProfileController [SDD-REQ-103]

Class Type	Controller	
Responsibility	This class handles user-related profile functionality, specifically for displaying the password change form and updating the user's password in the database.	
Attributes	Attributes Name	Attributes Type
	users_id role password	String String String(hashed)
Methods	Method Name	Description
	1. showChangePasswordForm 2. updatePassword	1. Displays the form for changing the user's password. 2. Validates and updates the user's password if current password matches.
Algorithm	1. showChangePasswordForm a. Simply return the change-password view to the user. 2. updatePassword a. Validate Input: Ensure current_password and new_password are provided, and the new password matches its confirmation. b.Retrieve User: Fetch the user using users_id from the session. c.Verify Password: Check if the provided current_password matches the stored hashed password. d.Update Password: If valid, hash the new_password and save it. e.Redirect: Based on the user's role, redirect to the appropriate dashboard with a success message. f.Handle Errors: If validation fails or the password is incorrect, redirect back with an error message.	

4.1.4 UserController [SDD-REQ-104]

Class Type	Controller	
Responsibility	Handle user-related operations such as creating users, displaying dashboards, generating reports, and managing user data.	
Attributes	Attributes Name	Attributes Type
	-	-
Methods	Method Name	Description
	1. insertUsers() 2. showStudentDashboard() 3. showLecturerDashboard() 4. generateReport($id) 5. printReport($id)	1. Inserts dummy user data into the database, including hashing passwords. 2. Fetches and displays all users with the role "student". 3. Fetches and displays all users with the role "lecturer". 4. Generates a detailed report for a student identified by their users_id. 5. Displays a print-friendly version of the student's report.
Algorithm	1. **Insert Users**: • Create an array of user data. • Hash each user's password. • Insert the data into the users table. 2. **Show Dashboard**: • Query the users table for users based on their role (student/lecturer). • Pass the results to the respective view. 3.**Generate Report**: • Find a student by users_id and verify their role. • Prepare a report with details such as name, email, and course list.	

	• Pass the report data to the view. **4. Print Report**: • Fetch the same student details as in generateReport. • Pass the data to a print-friendly view for display.

4.1.5 Student Report [SDD-REQ-105]

Class Type	View, HTML Template (Blade Template in Laravel context).	
Responsibility	Display a student report using dynamic data passed from a Laravel controller.	
Attributes	Attributes Name	Attributes Type
	Student ID Name Email Courses Report Date	String String String String String(DateTime format)
Methods	Method Name	Description
	1.Display Report Data 2. Navigation	1. Dynamically populates placeholders like {{ $reportData['Student ID'] }} with data passed from the controller. 2. Provides a link (<a>) for returning to the student dashboard.
Algorithm	Receive a $reportData array from the Laravel controller. Populate the report fields dynamically using Blade syntax ({{ $reportData['key'] }}). Render a back-navigation link using the url helper. Display the final formatted report in the browser.	

4.1.6 Student Report Print Preview [SDD-REQ-106]

Class Type	View, HTML Template (Blade Template in Laravel context).

Responsibility	The template is responsible for displaying the student report in a printable format. It dynamically renders student details and includes a print button that triggers the browser's print functionality.	
Attributes	Attributes Name	Attributes Type
	Title (<title>) Student ID Name Email Courses Report Date Print Button Container CSS Styles	String String String String String String(DateTime) Button(Anchor) Div Inline CSS
Methods	Method Name	Description
	1. window.print()	1. A JavaScript method triggered by the print button to open the browser's print dialog.
Algorithm	1. **Receive Data**: Retrieve the student report data ($reportData) from the controller. 2. **Render Data**: Populate the report with: o Student ID o Name o Email o List of courses o Report generation date 3. **Style the View**: Use inline CSS for styling the page layout, including the container and print button. 4. **Print Functionality**: Provide a print button (<a> with href="javascript:window.print();") that invokes the browser's native print dialog when clicked. 5. **Display and Interaction**: o Display the report in a user-friendly, centered container. o Allow users to preview the content and click the print button to generate a hard copy.	

4.1.7 Password Change Interface [SDD-REQ-107]

Class Type	View, HTML Template (Blade Template in Laravel context).	
Responsibility	This template renders the change password interface for users, allowing them to update their password securely. It includes input fields for current and new passwords and validates the inputs using Laravel's session messages and error handling.	
Attributes	Attributes Name	Attributes Type
	Current Password New Password New Password Confirmation CSRF Token Success Message Error Message Error for current password Error for new password Error for password confirmation Dashboard Link	String String String Token Session Session String String String Button
Methods	Method Name	Description
	1.session('success') 2.session('error') 3. @csrf 4. @error('field_name') 5. session('user_role')	1. Retrieves the success message from the session if available. 2. Retrieves the error message from the session if available. 3. Includes a CSRF token for securing the form submission against cross-site request forgery. 4. Displays validation errors associated with the specified field (current_password, new_password, etc.).

		5. Dynamically determines the user's role (e.g., lecturer or student) to generate the correct dashboard redirection URL.
Algorithm	**Retrieve Session Messages**: • Check for success or error messages in the session (session('success') and session('error')). • Display appropriate alerts using Bootstrap classes. **Render Form**: • Create a form with fields for: o Current password o New password o New password confirmation • Use the @csrf directive to include a CSRF token for security. **Validate and Display Errors**: • Use @error to display validation errors for each input field. **Submit the Form**: • Direct the form submission to the /change-password route using the POST method. **Provide Dashboard Navigation**: • Add a dynamic button that redirects users back to their respective dashboards (lecturer-dashboard or student-dashboard) based on the session's user_role. **Style and Usability**: • Use Bootstrap classes to ensure a responsive and user-friendly design.	

4.1.8 Lecturer Dashboard [SDD-REQ-108]

Class Type	View, HTML Template (Blade Template in Laravel context).

Responsibility	The template serves as the lecturer dashboard, displaying a list of lecturers with their details and providing functionality for lecturers to manage research fields, change passwords, or log out.	
Attributes	Attributes Name	Attributes Type
	Lecturer list Lecturer id Name Email Research fields Change password button Logout link	Array of Objects String String String Array of Objects Button Link
Methods	Method Name	Description
	1.addResearchField(field) 2. @foreach Blade Directive 3. url('/logout') 4. route('change.password')	1. JavaScript function triggered when a research field button is clicked. Displays a success message. 2. Iterates over the lecturers array and dynamically generates table rows for each lecturer. 3. Generates a URL for the logout route. 4. Generates a route for the change password page.
Algorithm	**Display Lecturer List**: • Fetch the lecturers data from the controller and iterate over it using the @foreach directive. • For each lecturer, display their users_id, name, and email in a table row. **Add Research Fields**: • Include buttons for predefined research fields such as "AI", "Software", and "Data Science". • Each button triggers the addResearchField(field) JavaScript function, which displays a success message.	

	Change Password Functionality: • Add a button that links to the change password page using the route('change.password') helper. **Logout Functionality**: • Provide a link to the logout route using url('/logout'). **Interactivity and Usability**: • Use JavaScript for real-time interaction (e.g., showing success messages when buttons are clicked). • Ensure the interface is clean and user-friendly with logical navigation options.

4.1.9 Student Dashboard [SDD-REQ-109]

Class Type	View, HTML Template (Blade Template in Laravel context).	
Responsibility	The template provides a user interface for managing student information, generating and printing reports, and updating the user's password. It lists all students and their details, offering actionable buttons for reports and password management.	
Attributes	Attributes Name	Attributes Type
	Student list	Array of Objects
	Student id	String
	Name	String
	Email	String
	Course List	String(csv)
	Generate Report Button	Button
	Print Report Button	Button
	Change password button	Button
	Logout button	Button
Methods	Method Name	Description

	1. @foreach Blade Directive 2. route('generate.report', id) 3. route('print.report', id) 4. route('change.password')	1. Iterates over the students array to display each student's details dynamically. 2. Generates a URL for the route to generate a report for the student identified by id. 3. Generates a URL for the route to print a report for the student identified by id. 4. Generates a URL for the route to change the user's password.
Algorithm	• Display Lecturer List: • Fetch the lecturers data from the controller and iterate over it using the @foreach directive. • For each lecturer, display their users_id, name, and email in a table row. • Add Research Fields: • Include buttons for predefined research fields such as "AI", "Software", and "Data Science". • Each button triggers the addResearchField(field) JavaScript function, which displays a success message. • Change Password Functionality: • Add a button that links to the change password page using the route('change.password') helper. • Logout Functionality: • Provide a link to the logout route using url('/logout'). • Interactivity and Usability: • Use JavaScript for real-time interaction (e.g., showing success messages when buttons are clicked). • Ensure the interface is clean and user-friendly with logical navigation options.	

4.1.10 Main Dashboard [SDD-REQ-110]

Class Type	View, HTML Template (Blade Template in Laravel context).	
Responsibility	This page serves as the welcome screen for the UMPsa FYP System, offering users links to the Student and Lecturer Dashboards and a Login button. It introduces the platform and provides navigation options to different sections of the system.	
Attributes	Attributes Name	Attributes Type
	Title Welcome message Student dashboard link Lecturer dashboard link Login button	String String Button Button Button
Methods	Method Name	Description
	1. route('student.dashboard') 2. route('lecturer.dashboard') 3. route('login')	1. Generates the URL for the student dashboard route, directing users to the student dashboard page. 2. Generates the URL for the lecturer dashboard route, directing users to the lecturer dashboard page. 3. Generates the URL for the login page, allowing users to log in and access restricted areas of the system
Algorithm	**Page Setup**: • Set up the page with an HTML structure, including a container class for centering content. **Page Header**: • Display a large heading (<h1>) welcoming the user to the platform.	

	• Add a subheading (<p>) that describes the purpose of the system, mentioning the management of Final Year Projects (FYP). **Introduction**: • Add a horizontal rule (<hr>) for separation and a short paragraph encouraging users to explore the system. **Links to Dashboards**: • Provide two buttons that link to the student and lecturer dashboards, generated using the route helper in Laravel to dynamically generate the URLs. **Login Button**: • Include a login button that links to the login page, generated using route('login'), for user authentication. **Styling**: • Use Bootstrap for responsive design and styling of the page, ensuring that it is user-friendly and visually appealing. • The btn-lg classes are used to make the buttons large for better visibility. **User Interaction**: • The buttons allow users to navigate to relevant areas of the platform: either the student dashboard, lecturer dashboard, or login page. **Responsive Design**: • Utilize Bootstrap's built-in responsive design to ensure the page looks good on various screen sizes.

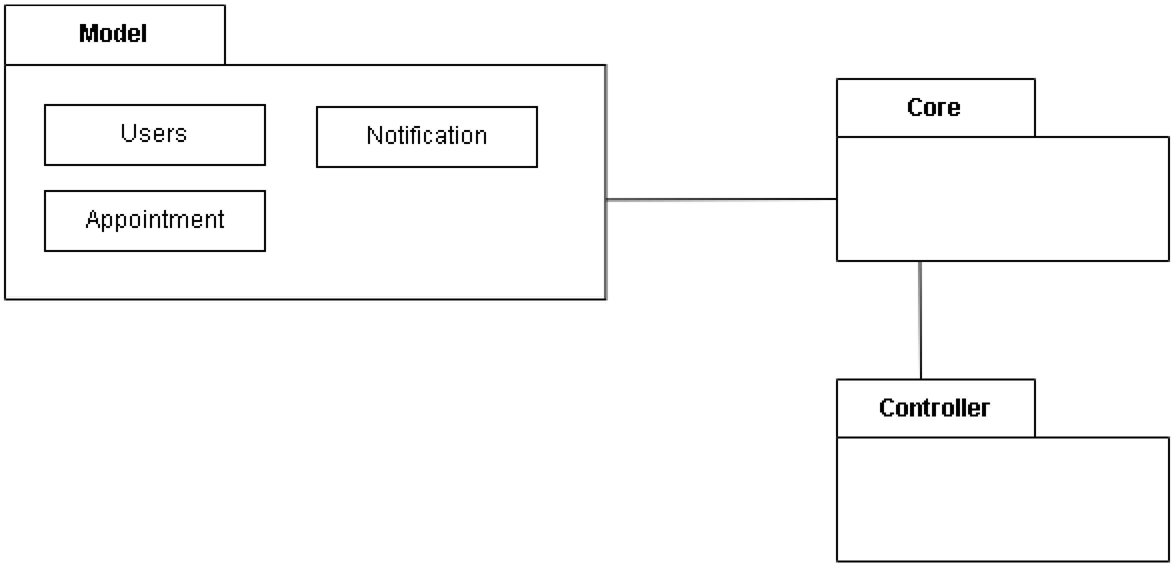
4.1.11 Login Interface [SDD-REQ-111]

Class Type	View, HTML Template (Blade Template in Laravel context).

Responsibility	This page allows users to log into the platform using their user ID and password. It displays error messages if the login fails or if validation issues occur.	
Attributes	Attributes Name	Attributes Type
	Title Login form User id input Password input Login button Error messages Validation errors	String Form Input Input Button Alert Alert
Methods	Method Name	Description
	1. route('login.submit') 2. session('error') 3. @foreach ($errors->all() as $error)	1. Generates the URL for submitting the login form (handling login logic, usually via a controller). 2. Retrieves and displays any session error messages (e.g., invalid login attempts). 3. Iterates through any validation errors and displays them in an unordered list.
Algorithm	1. **Page Setup**: o Load the page with an HTML structure that includes Bootstrap for styling and responsive design. o Center the login form using the .login-container class. 2. **Session Error Handling**: o Check if there is any session error (session('error')). If present, display it in a red alert box at the top of the page. 3. **Form Creation**: o Create a login form with two fields: one for the user ID (users_id) and one for the password (password). o Both input fields are marked with the required attribute, making them mandatory. o Use a table layout to organize the input fields and labels, which is styled to ensure proper alignment. 4. **Error Validation**:	

	◦ If any form validation errors occur (e.g., missing or invalid fields), they are displayed in a red alert box below the form using the @if ($errors->any()) directive. 5. **Login Button**: ◦ The form contains a submit button labeled Login, which triggers the form submission to the specified route (route('login.submit')). 6. **Bootstrap Integration**: ◦ Use Bootstrap classes (btn, form-control, alert, etc.) for styling the form, buttons, and error messages. ◦ The page layout adjusts responsively for different screen sizes. 7. **Form Submission**: ◦ Upon successful submission, the user is redirected to the appropriate page (usually handled by a controller based on login credentials). If the login is unsuccessful, the error messages are shown on the page. 8. **Security**: ◦ CSRF protection is automatically included with the @csrf directive, ensuring secure form submission. ◦ Password field uses the type="password" attribute to mask user input.

4.2 MODULE 2: MANAGE APPOINTMENT (KEH BAN NING CB23137)



4.2.1 User [SDD-REQ-201]

Class Type	Model Class	
Responsibility	This model contains the users information.	
Attributes	Attributes Name	Attributes Type
	id	bigint(20)
	name	varchar(255)
	email	varchar(255)
	password	varchar(255)
	role	enum(‘student’,’lecturer’,’coordinator’)
	program	varchar(255)
Methods	Method Name	Description
	studentAppointments()	Defines a relationship where the user (student) has many appointments linked by the student_id.
	lecturerAppointments()	Defines a relationship where the user (lecturer) has many appointments linked by the lecturer_id.
	isStudent()	Checks if the user’s role is "student."
	isLecturer()	Checks if the user’s role is "lecturer."

Algorithm	1. studentAppointments() 3. Returns the appointments where this user is the student (using student_id). 2. lecturerAppointments() 1. Returns the appointments where this user is the lecturer (using lecturer_id). 3. isStudent() 1. Compares the user's role attribute to 'student'. 2. Returns true if the role is 'student', otherwise false. 4. isLecturer() 1. Compares the user's role attribute to 'lecturer'. 2. Returns true if the role is 'lecturer', otherwise false.

4.2.2 Appointment [SDD-REQ-202]

Class Type	Model Class	
Responsibility	This model contains the appointment information.	
Attributes	Attributes Name	Attributes Type

	id
	student_id
	lecturer_id
	date
	time
	status
		bigint(20)
		bigint(20)
		bigint(20)
		date
		time
		enum(‘pending’,’approved’,’rejected’)

Methods	Method Name	Description
	student()	Defines a relationship where the appointment belongs to a student.
	lecturer()	Defines a relationship where the appointment belongs to a lecturer.

Algorithm	1. student() 1. Establishes a relationship between the Appointment and User models, where the student_id in the appointment table refers to the id of a User. 2. lecturer() 1. Establishes a relationship between the Appointment and User models, where the lecturer_id in the appointment table refers to the id of a User.

4.2.3 Notification [SDD-REQ-203]

Class Type	Model Class	
Responsibility	This model contains the notification information.	
Attributes	Attributes Name	Attributes Type

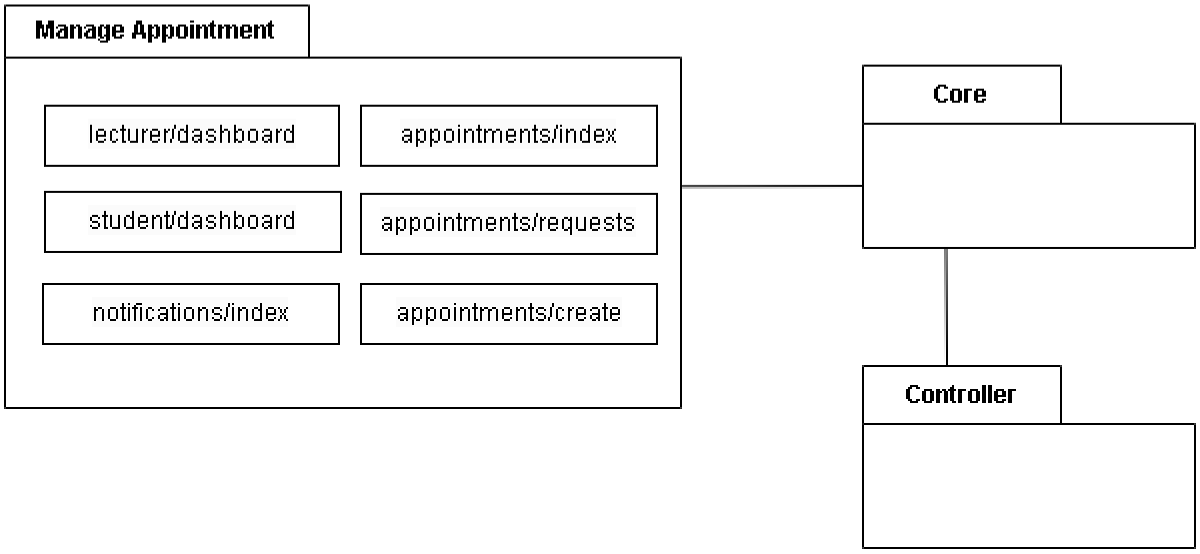
	id
	type
	notifiable_type
	notifiable_id
	data
	read_at
		char(36)
		varchar(255)
		varchar(255)
		bigint(20)
		text
		timestamp

Methods	Method Name	Description

	user()
	
	isRead()
		Defines a relationship where the notification belongs to a user.
		Checks if the notification has been read.

Algorithm	1. user() 1. Establishes a relationship between the Notification and User models, where the user_id in the notification table refers to the id of a User. 2. isRead()	

	1. Returns true if read_at is not null, meaning the notification has been marked as read. 2. Returns false if read_at is null, indicating the notification is unread.



4.2.4 lecturer/dashboard [SDD-REQ-204]

Class Type	Boudary Class	
Responsibility	A dashboard interface for lecturers to approve or decline student appointment requests, showing the requested appointment details such as date, time, status, and the student's name.	
Attributes	Attributes Name	Attributes Type
	student_id lecturer_id date time status id name email password role	unsignedBigInteger unsignedBigInteger date time enum('pending', 'approved', 'rejected') bigIncrements string string string enum('student', 'lecturer', 'admin')
Methods	Method Name	Description
	construct	Initializes middleware for authentication and role restriction.

	index	Fetches and displays the lecturer's appointments and relevant statistics on the dashboard.
Algorithm	1. Retrieve the current authenticated user using Auth::user(). 2. Fetch the appointments for the lecturer using Appointment::where('lecturer_id', $user->id), paginate, and display. 3. Count the pending appointment requests using Appointment::where('lecturer_id', $user->id)->where('status', 'pending')->count(). 4. Count the total distinct students who have approved appointments with the lecturer. 5. Return the view with the required data: appointments, pending requests, and total students.	

4.2.5 student/dashboard [SDD-REQ-205]

Class Type	Boudary Class	
Responsibility	A dashboard interface for students to schedule new appointments with lecturers by selecting the desired date and time, as well as to view the status of their appointments.	
Attributes	Attributes Name	Attributes Type
	student_id lecturer_id date time status id name email password	unsignedBigInteger unsignedBigInteger date time enum('pending', 'approved', 'rejected') bigIncrements string string string enum('student', 'lecturer', 'admin')

	role	
Methods	Method Name	Description
	index	Displays the student's dashboard with information on current supervisor, pending requests, and a list of appointments.
Algorithm	1. Retrieve the current authenticated user using Auth::user(). 2. Fetch the current supervisor of the student using Appointment::where('student_id', $user->id)->where('status', 'approved')->with('lecturer')->first(). 3. Count the number of pending requests for the student using Appointment::where('student_id', $user->id)->where('status', 'pending')->count(). 4. Fetch the list of appointments for the student using Appointment::where('student_id', $user->id)->with('lecturer')->latest()->paginate(10). 5. Return the view with the required data: current supervisor, pending requests, and appointments.	

4.2.6 notifications/index [[SDD-REQ-206]

Class Type	Boudary Class	
Responsibility	An interface that displays a list of notifications for the user, showing the message and the time since creation. It includes options to mark notifications as read and delete them. Unread notifications are highlighted, and appropriate buttons are provided for each action.	
Attributes	Attributes Name	Attributes Type
	Id user_id message read_at	bigIncrements unsignedBigInteger text timestamp
Methods	Method Name	Description

	Index markAsRead destroy	Displays the list of notifications for the authenticated user. Marks a notification as read by updating the read_at timestamp Deletes the specified notification.
Algorithm	Algorithm for index: 1. Retrieve the current authenticated user using $request->user(). 2. Fetch all notifications for the user from the Notification model using Notification::where('user_id', $user->id)->latest()->get(). 3. Return the notifications.index view with the notifications data. Algorithm for markAsRead: 1. Find the notification by ID using $notification->update(['read_at' => now()]). 2. Return a JSON response indicating the notification has been marked as read. Algorithm for destroy: 1. Delete the notification using $notification->delete(). 2. Return a JSON response confirming the notification was deleted.	

4.2.7 appointments/index [SDD-REQ-207]

Class Type	Boudary Class
Responsibility	• Coordinator: A coordinator manages and oversees the appointments, ensuring students and lecturers are matched correctly. They can view both student and lecturer details and monitor the status of appointments. • Lecturer: A lecturer handles appointment requests from students. They can view student details, approve or reject appointments, and update the status of each appointment.

	• Student: A student can request appointments with lecturers and view their own appointments.	
Attributes	Attributes Name	Attributes Type
	student_id lecturer_id date time status role	unsignedBigInteger unsignedBigInteger date time enum('pending', 'approved', 'rejected') enum('student', 'lecturer', 'admin')
Methods	Method Name	Description
	index	Displays a list of appointments based on the role of the logged-in user (coordinator, lecturer, or student).
Algorithm	Determine the role of the logged-in user (coordinator, lecturer, or student). Fetch appointments based on the role: • For coordinator, show all appointments. • For lecturer, show appointments where the lecturer is involved. • For student, show appointments involving the logged-in student. Display the appointments in a table with the option to update or cancel based on the user's role.	

4.2.8 appointments/requests [SDD-REQ-208]

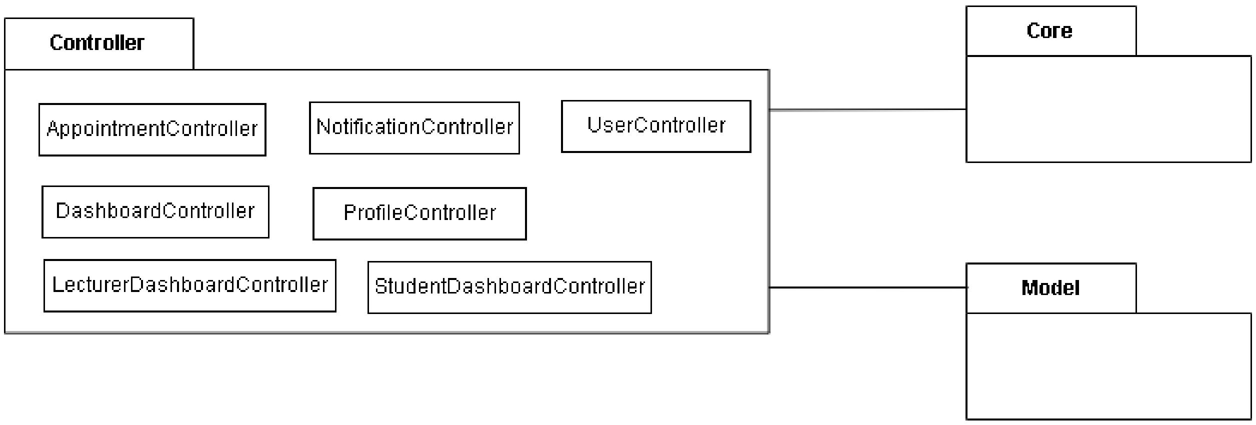
Class Type	Boudary Class
Responsibility	An interface that displays a list of pending appointment requests. If no pending requests exist, a message is shown. If there are pending appointments, they are displayed in a table with columns for the row number, student name, appointment date, time, status, and action buttons. The action buttons allow the coordinator or lecturer to accept or reject each appointment by updating the appointment's status.

Attributes	Attributes Name	Attributes Type
	student_id lecturer_id date time status role	unsignedBigInteger unsignedBigInteger date time enum('pending', 'approved', 'rejected') enum('student', 'lecturer', 'admin')
Methods	Method Name	Description
	update-status	Allows lecturer to update the status of an appointment.
Algorithm	1. Capture the new status (approved or rejected) based on the button clicked. 2. Update the status of the corresponding appointment in the database. 3. Redirect back to the appointments page with a success message indicating the action (acceptance or rejection) was completed.	

4.2.9 appointments/create [SDD-REQ-209]

Class Type	Boudary Class	
Responsibility	An interface that allows students to request an appointment by selecting a lecturer, date, and time. The form includes a hidden field for the student's ID and submits the data to store the appointment request. A submit button is provided to complete the request.	
Attributes	Attributes Name	Attributes Type
	student_id lecturer_id date time status	unsignedBigInteger unsignedBigInteger date time enum('pending', 'approved', 'rejected') bigint

	id name	varchar
Methods	Method Name	Description
	store	Handles the storing of a new appointment request made by the student.
Algorithm	1.Validate the input data (lecturer selection, date, time). 2. Create a new appointment entry in the database with the lecturer_id, student_id, date, and time. 3. Set the status of the appointment to pending by default. 4. Redirect the student to the appointments page with a success message confirming the request was made.	



4.2.10 AppointmentController [SDD-REQ-210]

Class Type	Boudary Class	
Responsibility	The controller handles appointment-related actions: displaying appointments based on user roles, storing new requests, fetching pending requests for lecturers, showing appointment details, updating status, and deleting appointments.	
Attributes	Attributes Name	Attributes Type
	student_id	unsignedBigInteger
	lecturer_id	unsignedBigInteger
	date	date
	time	time
	status	enum('pending', 'approved', 'rejected')
		bigint
	id	enum('student', 'lecturer', 'coordinator')
	role	
Methods	Method Name	Description
	index()	Displays a listing of appointments based on the authenticated user's role.
		Stores a new appointment request after validating the data.
	store(Request $request)	Fetches and displays pending appointment requests for a lecturer.
	requests()	

	show(Appointment $appointment)
	
	create()
	
	updateStatus(Request $request, Appointment $appointment)
	
	destroy(Appointment $appointment)
		Displays the details of a specific appointment.
		
		Displays a form for creating a new appointment, populated with lecturers.
		Updates the status of an appointment (approved or rejected).
		
		Deletes a specified appointment from the database.

Algorithm	1. index() 1. Get the authenticated user. 2. Initialize an appointment query. 3. Filter appointments based on the user's role (student, lecturer, coordinator). 4. Return the relevant appointments to the appointments.index view. 2. store(Request $request) 1. Validate the input fields (student_id, lecturer_id, date, time). 2. If validation passes, create a new appointment. 3. Redirect to the appointments.create page with a success message. 4. If an error occurs, redirect back with an error message. 3. requests() 1. Ensure the authenticated user is a lecturer. 2. Fetch all pending appointments for the lecturer using a relationship method (lecturerAppointments()). 3. Return the pending appointments to the appointments.requests view.	

	4. show(Appointment $appointment) 1. Load the related student and lecturer for the given appointment. 2. Return the appointment details as a JSON response. 5. create() 1. Fetch all users with the lecturer role. 2. Pass the lecturers to the appointments.create view for the user to select. 6. updateStatus(Request $request, Appointment $appointment) 1. Validate the incoming status field to ensure it is one of pending, approved, or rejected. 2. Update the status of the given appointment. 3. Redirect to the appointments.index route with a success message. 7. destroy(Appointment $appointment) 1. Delete the given appointment. 2. Redirect back with a success message.

4.2.11 DashboardController [SDD-REQ-211]

Class Type	Boudary Class	
Responsibility	The controller handles the dashboard data, displaying the total number of users, appointments, and pending appointments.	
Attributes	Attributes Name	Attributes Type

	id
	name
	email
	password
	role
		bigint(20)
		varchar(255)
		varchar(255)
		varchar(255)
		enum('student','lecturer','coordinator')

	program
	id
	student_id
	lecturer_id
	date
	time
	status
		varchar(255)
		bigint(20)
		bigint(20)
		bigint(20)
		date
		time
		enum('pending','approved','rejected')

Methods	Method Name	Description
	index()	Retrieves counts of users, appointments, and pending appointments, and passes them to the dashboard view.
Algorithm	1. Count the total number of users using User::count(). 2. Count the total number of appointments using Appointment::count(). 3. Count the number of pending appointments using Appointment::where('status', 'pending')->count(). 4. Return the dashboard view with the counts (usersCount, appointmentsCount, pendingAppointments).	

4.2.12 LecturerDashboardController [SDD-REQ-212]

Class Type	Boudary Class	
Responsibility	The controller handles the lecturer's dashboard, displaying the lecturer's appointments, pending requests, and the total number of students with approved appointments.	
Attributes	Attributes Name	Attributes Type

	id
	name
	email
	password
	role
	program
		bigint(20)
		varchar(255)
		varchar(255)
		varchar(255)
		enum('student','lecturer','coordinator')
		varchar(255)

	id
	student_id
	lecturer_id
	date
	time
	status
		bigint(20)
		bigint(20)
		bigint(20)
		date
		time
		enum(‘pending’,’approved’,’rejected’)

Methods	Method Name	Description
	construct()	Initializes middleware to ensure the user is authenticated and has the 'lecturer' role.
	index()	Fetches the lecturer's appointment data, pending requests, and total number of distinct students with approved appointments.
Algorithm	1. construct() 1. Apply the auth middleware to ensure the user is logged in. 2. Apply the role:lecturer middleware to restrict access to users with the 'lecturer' role. 3. index() 1. Get the authenticated user. 2. Fetch the lecturer's appointments, ordered by the most recent, and paginate the results (10 per page). 3. Count the number of pending requests for the lecturer. 4. Count the number of distinct students with approved appointments. 5. Return the data to the view lecturer.dashboard.	

4.2.13 StudentDashboardController [SDD-REQ-213]

Class Type	Boudary Class

Responsibility	The controller handles the student's dashboard, displaying the student's appointments, current supervisor (if approved), and the count of pending requests.	
Attributes	Attributes Name	Attributes Type

	id
	name
	email
	password
	role
	program
	id
	student_id
	lecturer_id
	date
	time
	status
		bigint(20)
		varchar(255)
		varchar(255)
		varchar(255)
		enum(‘student’,’lecturer’,’coordinator’)
		varchar(255)
		bigint(20)
		bigint(20)
		bigint(20)
		date
		time
		enum(‘pending’,’approved’,’rejected’)

Methods	Method Name	Description

	construct()
	index()
		Initializes middleware to ensure the user is authenticated and has the 'student' role.
		Fetches the student's appointments, current supervisor, and pending requests.

Algorithm	1. construct() 1. Apply the auth middleware to ensure the user is logged in. 2. Apply the role:student middleware to restrict access to users with the 'student' role. 2. index() 1. Get the authenticated user. 2. Fetch the student's appointments, ordered by the most recent, and paginate the results (10 per page).	

	3. Retrieve the first approved appointment (current supervisor) for the student. 4. Count the number of pending requests for the student. 5. Return the data to the view student.dashboard.

4.2.14 NotificationController [SDD-REQ-214]

Class Type	Boudary Class	
Responsibility	The controller handles managing notifications for a specific user, including displaying notifications, marking them as read, and deleting them.	
Attributes	Attributes Name	Attributes Type

	id
	type
	notifiable_type
	notifiable_id
	data
	read_at
		char(36)
		varchar(255)
		varchar(255)
		bigint(20)
		text
		timestamp

Methods	Method Name	Description
	index()	Retrieves a list of notifications for a specific user, ordered by the latest.
	markAsRead()	Marks a notification as read by updating the read_at timestamp.
	destroy()	Deletes a specific notification.
Algorithm	1. index() 1. Get the authenticated user from the request. 2. Fetch all notifications for the user, ordered by the most recent. 3. Return the notifications as a JSON response. 2. markAsRead() 1. Update the read_at column of the notification to the current timestamp. 2. Return a JSON response indicating the notification has been marked as read.	

	3. destroy() 1. Delete the given notification. 2. Return a JSON response confirming the deletion.

4.2.15 ProfileController [SDD-REQ-215]

Class Type	Boudary Class	
Responsibility	The controller handles displaying the profile page for the authenticated user.	
Attributes	Attributes Name	Attributes Type

	id
	name
	email
	password
	role
	program
		bigint(20)
		varchar(255)
		varchar(255)
		varchar(255)
		enum('student','lecturer','coordinator')
		varchar(255)

Methods	Method Name	Description
	index()	Fetches the authenticated user and passes it to the view for displaying the profile.
Algorithm	1. Retrieve the authenticated user using Auth::user(). 2. Return the profile.index view, passing the user data to it for display.	

4.2.16 UserController [SDD-REQ-216]

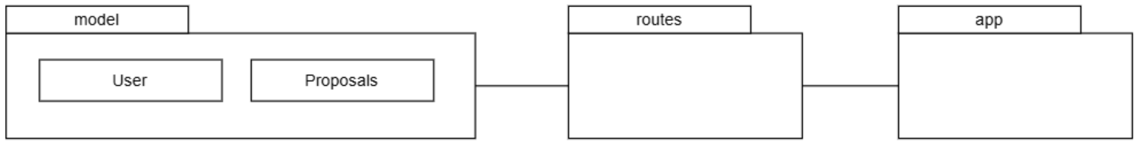
Class Type	Boudary Class	
Responsibility	The controller handles managing users, including creating, updating, displaying, and deleting user records, as well as validating input data for user creation and updates.	
Attributes	Attributes Name	Attributes Type

	id
	name
	email
	password
	role
	program
		bigint(20)
		varchar(255)
		varchar(255)
		varchar(255)
		enum('student','lecturer','coordinator')
		varchar(255)

Methods	Method Name	Description
	index()	Fetches all users from the users table and passes them to the view for listing.
	store(Request $request)	Validates the request and stores a new user in the database.
	show(User $user)	Returns the specified user as a JSON response.
	update(Request $request, User $user)	Validates and updates the specified user in the database.
	edit(User $user)	Returns a view for editing the specified user.
	create()	Returns a view for creating a new user.
	destroy(User $user)	Deletes the specified user from the database.
Algorithm	1. index() 1. Retrieve all users using User::all(). 2. Return the users.index view with the users data. 2. store(Request $request) 1. Validate the input request for name, email, password, and role. 2. Create a new user with the validated data. 3. Redirect to the users.index route with a success message. 3. show(User $user) 1. Fetch the user based on the route parameter. 2. Return the user data as a JSON response. 4. update(Request $request, User $user) 1. Validate the input request for name, email, password, and role. 2. Update the user with the validated data. 3. Redirect to the users.index route with a success message.	

	5. edit(User $user) 1. Fetch the user based on the route parameter. 2. Return the users.edit view with the user data. 6. create() 1.Return the users.create view for creating a new user. 7. destroy(User $user) 1. Delete the user based on the route parameter. 2. Return a JSON response with a success message.

4.3 MODULE 3: APPLY TOPIC/TITLE (AZMI WAN NURHALIZA CB22004)



4.3.1 User [SDD-REQ-301]

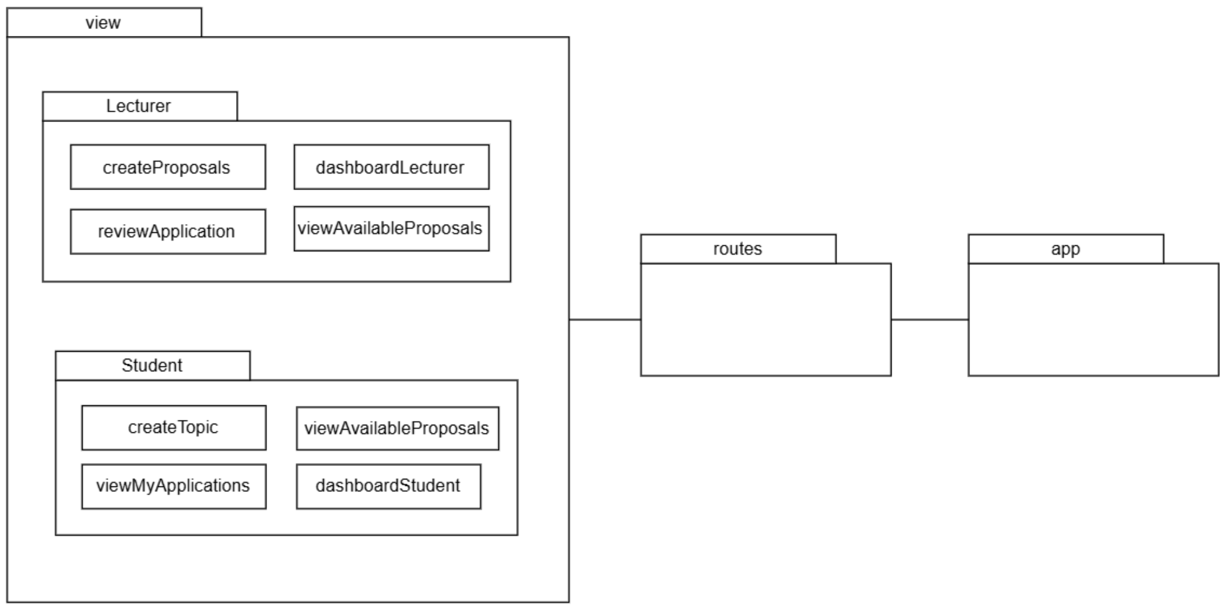
Class Type	Model Class	
Responsibility	Represents the users table in the database. Manages user data, including authentication, roles, and interactions with proposals.	
Attributes	Attributes Name	Attributes Type

	id
	name
	email
	password
	role
	email_verified_at
		bigint(20)
		varchar(255)
		varchar(255)
		varchar(255)
		enum(‘student’,’lecturer’,’admin’)
		datetime (nullable)

Methods	Method Name	Description
	proposals()	Defines a one-to-many relationship with the Proposals model. Associates a user with their proposals.
Algorithm	1. Define the User model extending Laravel's Authenticatable class. 2. Implement the proposals() method to establish the relationship with the Proposals model. 3. Ensure the role attribute supports role-based access control in the application.	

4.3.2 Proposals [SDD-REQ-302]

Class Type	Model Class	
Responsibility	Represents the proposals table in the database. Manages FYP proposal data, including title, status, lecturer, and student details.	
Attributes	Attributes Name	Attributes Type
	id	bigint(20)
	proposal_id	varchar(255)
	title	varchar(255)
	id_users	bigint(20) (foreign key)
	quota	int(11)
	status	enum('approved', 'pending', 'rejected', '-')
	status_available	Boolean
	jadwal	varchar(255) (file path)
	name_student	varchar(255)
	program	varchar(255)
	fyp	varchar(255)
	description	Text
	semester	varchar(255)
	year_academy	varchar(255)
Methods	Method Name	Description
	user()	Defines a many-to-one relationship with the User model, linking proposals to their owners.
Algorithm	1. Define the Proposals model extending Laravel's Model class. 2. Implement the user() method to establish the many-to-one relationship with the User model. 3. Ensure data validations are in place for required attributes such as title, quota, and status.	



4.3.3 Lecturer : createProposals [SDD-REQ-303]

Class Type	View Class	
Responsibility	This view enables lecturers to create new FYP proposals. Lecturers can define the proposal details such as title, quota, and schedule file, which will be made available for students to apply for.	
Attributes	Attributes Name	Attributes Type

	title
	quota
	schedule_file
		varchar(255)
		int(11)
		varchar(255)

Methods	Method Name	Description
	createProposals()	Displays a form for lecturers to input and submit new FYP proposals.
Algorithm	1. Display the proposal creation form with fields for title, quota, and schedule_file. 2. When the lecturer submits the form: • Validate the inputs to ensure all fields are completed, and the schedule file is a valid PDF. 3. If validation passes:	

	• Save the proposal details in the proposals table, associating it with the lecturer's ID. • Mark the proposal as "Available" (status_available = true). • Redirect the lecturer to their dashboard with a success message. 4. If validation fails, display error messages for invalid fields and remain on the creation page.

4.3.4 Lecturer : dashboardLecturer [SDD-REQ-304]

Class Type	View Class	
Responsibility	This view serves as the main interface for lecturers, summarizing their created proposals, reviewing pending applications, and accessing proposal management tools.	
Attributes	Attributes Name	Attributes Type

	lecturer_name
	proposal_count
	pending_applications
		varchar(255)
		int(11)
		int(11)

Methods	Method Name	Description
	dashboardLecturer()	Displays a summary of the lecturer's activity and options for managing proposals and applications.
Algorithm	1. Retrieve the lecturer's proposals and applications from the database. 2. Display the total number of proposals, pending applications, and other relevant metrics. 3. Provide navigation options for creating proposals, viewing available proposals, and reviewing applications.	

4.3.5 Lecturer : reviewApplication [SDD-REQ-305]

Class Type	View Class	
Responsibility	This view allows lecturers to review student applications for their FYP proposals. Lecturers can approve or reject applications and update their status accordingly.	
Attributes	Attributes Name	Attributes Type

	student_name
	proposal_title
	application_status
	
	submission_date
		varchar(255)
		varchar(255)
		enum('pending', 'approved', 'rejected')
		datetime

Methods	Method Name	Description
	reviewApplication()	Displays the list of student applications and allows lecturers to approve or reject them.
Algorithm	1. Retrieve all applications for the lecturer's proposals from the database, filtering by pending status. 2. Display the list of applications with details such as student_name, proposal_title, and submission_date. 3. Provide buttons or options for approving or rejecting each application. 4. On approval, update the application's status to approved. 5. On rejection, update the application's status to rejected. 6. Refresh the list and display a success message.	

4.3.6 Lecturer : viewAvailableProposals [SDD-REQ-306]

Class Type	View Class	
Responsibility	Displays all the FYP proposals created by the lecturer and their current status. Lecturers can manage their proposals through this interface.	
Attributes	Attributes Name	Attributes Type

	proposal_title
	quota
	status_available
		varchar(255)
		int(11)
		boolean

	application_count	int(11)
Methods	Method Name	Description
	viewAvailableProposals()	Displays all the FYP proposals created by the lecturer, along with their current status.
Algorithm	1. Retrieve all proposals associated with the lecturer’s ID from the database. 2. Display each proposal’s title, quota, status_available, and application_count. 3. Provide options to edit or close proposals if necessary. 4. If a proposal is closed (status_available = false), display it as "Not Available" to indicate no further applications can be submitted.	

4.3.7 Student : createTopic [SDD-REQ-307]

Class Type	View Class	
Responsibility	This view allows students to submit their own FYP proposals by providing details such as title, description, and selecting a lecturer.	
Attributes	Attributes Name	Attributes Type
	name_student	varchar(255)
	program	varchar(255)
	fyp_title	varchar(255)
	description	text
	lecturer_id	bigint(20)
Methods	Method Name	Description
	createTopic()	createTopic()
Algorithm	1. Display the proposal submission form with fields for fyp_title, description, program, and lecturer_id. 2. Validate the inputs to ensure all required fields are filled correctly. 3. On form submission: • Save the data into the proposals table, associating it with the student's id. • Mark the proposal's status as pending for lecturer review. 4. Redirect the student to their applications page with a success message.	

4.3.8 Student : viewAvailableProposals [SDD-REQ-308]

Class Type	View Class	
Responsibility	Displays a list of available FYP proposals for students to browse and apply for.	
Attributes	Attributes Name	Attributes Type

	proposal_title
	lecturer_name
	quota
	status_available
		varchar(255)
		varchar(255)
		int(11)
		boolean

Methods	Method Name	Description
	viewAvailableProposals()	Displays all available proposals that students can apply for.
Algorithm	1. Retrieve all proposals from the database where status_available = true. 2. Display the proposals with details such as proposal_title, lecturer_name, and quota. 3. Provide an "Apply" button for each proposal: • On click, reduce the proposal's quota by 1. • Create a new application record in the database.	

4.3.9 Student : viewMyApplications [SDD-REQ-309]

Class Type	View Class	
Responsibility	Provides students with a summary of their submitted applications, including their current status (e.g., approved, pending, or rejected).	
Attributes	Attributes Name	Attributes Type

	application_id
	proposal_title
	application_status
	
	submission_date
		bigint(20)
		varchar(255)
		enum('approved', 'rejected', 'pending')
		datetime

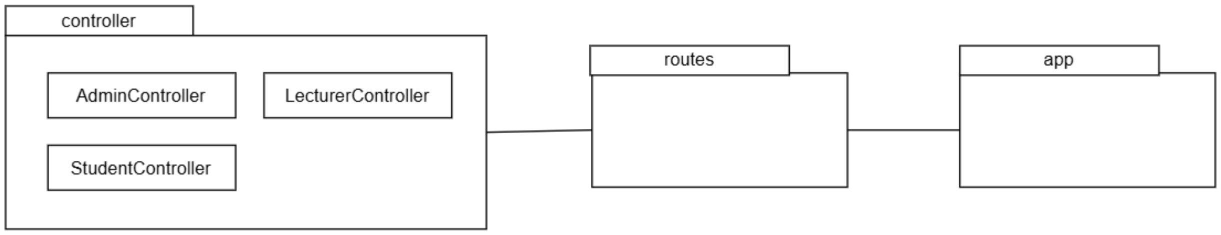
Methods	Method Name	Description
	viewMyApplications()	Displays a summary of all proposals submitted by the student.
Algorithm	1. Retrieve all applications submitted by the logged-in student. 2. Display details for each application, including proposal_title, status, and submission_date. 3. Highlight applications based on their status (e.g., green for approved, red for rejected).	

4.3.10 Student : dashboardStudent [SDD-REQ-310]

Class Type	View Class	
Responsibility	Provides a central interface for students to access their submitted applications, browse proposals, and manage their activities.	
Attributes	Attributes Name	Attributes Type

	student_name
	application_count
		varchar(255)
		int(11)

Methods	Method Name	Description
	dashboardStudent()	Displays the student's main dashboard, summarizing their activities.
Algorithm	1. Retrieve the student's applications and available proposals from the database. 2. Display a summary of submitted applications and their statuses. 3. Provide navigation links to view available proposals or submit a new proposal.	



4.3.11 AdminController [SDD-REQ-311]

Class Type	Controller Class	
Responsibility	Handles administrative functionalities such as managing user accounts, viewing user data, and performing CRUD operations.	
Attributes	Attributes Name	Attributes Type

	user_id
	role
		bigint(20)
		enum('student', 'lecturer', 'admin')

Methods	Method Name	Description

	index()
	
	destroy($id)
	
	showDashboard()
		Retrieves a list of all users and displays them for management purposes.
		Deletes a user account based on their ID.
		Displays the admin dashboard with user and system management tools.

Algorithm	1. For index: • Retrieve all users from the users table. • Display them in a list format for admin management. 2. For destroy: • Find the user by id and delete their record. • Redirect back to the user list with a success message. 3. For showDashboard: • Display a summary of user data and management options for the admin.	

4.3.12 LecturerController [SDD-REQ-312]

Class Type	Controller Class	
Responsibility	Manages lecturer specific functionalities, including creating proposals, reviewing applications, and viewing available proposals.	
Attributes	Attributes Name	Attributes Type

	proposal_id
	title
	status
		varchar(255)
		varchar(255)
		enum('approved', 'rejected', 'pending')

Methods	Method Name	Description
	create()	Displays the proposal creation form for lecturers.
	store(Request $request)	Validates and stores the proposal data in the database.
	reviewApplication()	Retrieves and displays student applications for lecturer review.
	approve($id)	Approves a student's application and updates the lecturer quota.
	reject($id)	Rejects a student's application and restores the lecturer quota.
Algorithm	1. For create: • Display the form for proposal creation. 2. For store: • Validate the proposal's inputs (title, quota, etc.). • Save the proposal details in the proposals table. • Redirect to the lecturer dashboard with a success message. 3. For reviewApplication: • Retrieve all applications for the lecturer's proposals. • Display them with options to approve or reject. 4. For approve and reject:	

	• Update the application's status in the database (approved or rejected). • Adjust the lecturer quota accordingly.

4.3.13 StudentController [SDD-REQ-313]

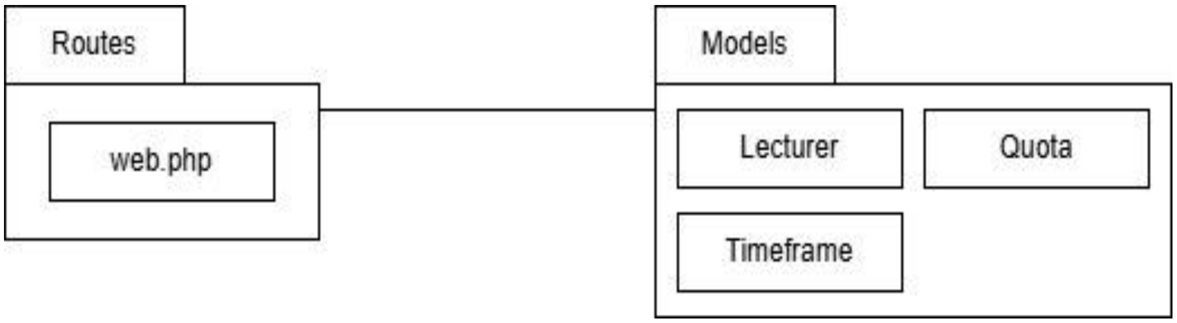
Class Type	Controller Class	
Responsibility	Manages student specific functionalities such as browsing available proposals, submitting applications, and tracking their applications.	
Attributes	Attributes Name	Attributes Type

	student_id
	application_status
		bigint(20)
		enum('pending', 'approved', 'rejected')

Methods	Method Name	Description
	index()	Displays all available proposals for students to apply for.
	apply($id)	Handles the process of applying for a selected proposal.
	viewMyApplications()	Displays the student's submitted applications and their statuses.
	store(Request $request)	Validates and stores a student-proposed FYP topic in the database.
Algorithm	1. For index: • Retrieve all proposals with status_available = true. • Display them for students to browse and apply. 2. For apply: • Validate the student's eligibility to apply for the proposal. • Decrease the proposal's quota by 1 and mark the application as pending. • Save the application record in the database. 3. For viewMyApplications:	

	• Retrieve all applications submitted by the logged-in student. • Display their statuses (pending, approved, or rejected).

4.4 MODULE 4: MANAGE TIMEFRAME AND QUOTA (NUR ALYA SYAKIRAH BINTI NASARUDIN CB22141)



4.4.1 Lecturer [SDD-REQ-401]

Class Type	Model Class	
Responsibility	This model contains the lecturer's information, including quota assignments and usage.	
Attributes	Attributes Name	Attributes Type

	Id
	name
	quota
	used_quota
	remaining_quota
		bigint(20)
		varchar(255)
		int(11)
		int(11)
		int(11)

Methods	Method Name	Description
	getRemainingSlots()	Returns the remaining quota slots available for the lecturer.
	getUsedQuota()	Retrieves the lecturer's used quota.
	getRemainingQuota()	Calculates and retrieves the lecturer's remaining quota.
Algorithm	1. getRemainingSlots() Calculates remaining_quota as quota - used_quota.	

	2. getUsedQuota() Fetches the value of used_quota from the database. 3. getRemainingQuota() Calculates remaining_quota by subtracting used_quota from quota and returns the result.	

4.4.2Quota [SDD-REQ-402]

Class Type	Model Class	
Responsibility	This model represents the quota assignments for each lecturer within specific semesters, including the total quota, used quota, and remaining quota.	
Attributes	Attributes Name	Attributes Type

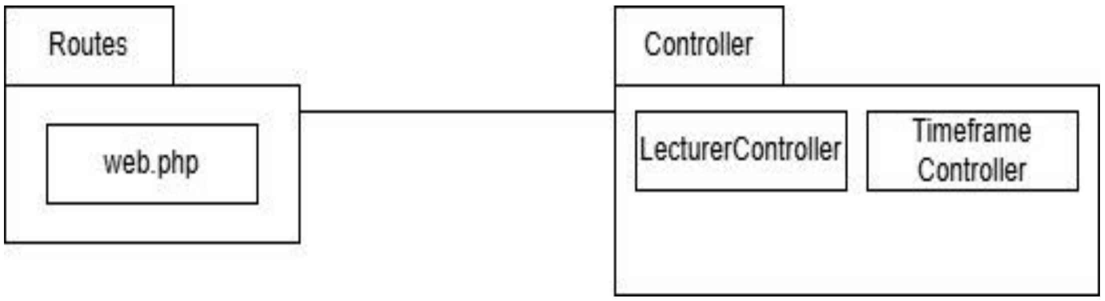
	id
	lecturer_id
	semester
	total_quota
	used_quota
	remaining_quota
		bigint(20)
		bigint(20)
		varchar(255)
		int(11)
		int(11)
		int(11)

Methods	Method Name	Description
	lecturer()	Defines the relationship between a quota and a lecturer.
Algorithm	1. lecturer() Defines a one-to-many relationship where each quota is associated with a lecturer.	

4.4.3 Timeframe [SDD-REQ-403]

Class Type	Model Class

Responsibility	This model stores the start and end dates for the supervisor hunting timeframe. It is used to manage the periods within which supervisor assignments are allowed..	
Attributes	Attributes Name	Attributes Type
	id start_date end_date	bigint(20) date date
Methods	Method Name	Description
	getFormattedDates()	Returns the start and end dates in a human-readable format.
Algorithm	1. getFormattedDates() Retrieves the start_date and end_date attributes from the Timeframe model.	



4.4.4 LecturerController [SDD-REQ-404]

Class Type	Controller Class	
Responsibility	The LecturerController is responsible for managing lecturer data, including setting quotas, displaying reports, and retrieving the list of lecturers based on semester input. It handles the logic related to quotas, including fetching and updating the quota values for lecturers	
Attributes	Attributes Name	Attributes Type
	lecturers	collection

	semester	string
	quotaData	quota
	notifications	boolean
	success	boolean
Methods	Method Name	Description
	index()	Displays the `set-quota` page, filtering lecturers by selected semester.
	store()	Handles the storage of updated quotas for lecturers..
	showQuotaPage()	Displays the `set-quota` page with all lecturers, irrespective of semester.
	generateReport()	Generates and displays a report of lecturers based on the selected semester.
		Fetches the list of lecturers based on the semester and returns them as JSON.
	getLecturers()	Displays quota information specifically formatted for students.
	showStudentDashboard()	Displays the student dashboard, filtered by semester if provided.
	setQuota()	Placeholder for handling quota setting logic .
	showDashboard()	Displays a detailed view of a specific lecturer's dashboard, including quota details.
Algorithm	1. index() Retrieves the selected semester from the request.	

	2. store() Loops through the lecturers input from the form, where each lecturer's ID and updated quota are provided. 3. showQuotaPage() Fetches all lecturers and returns the view with the full list. 4. generateReport() Filters lecturers based on the semester and generates the corresponding report. 5. getLecturers() Fetches lecturers based on the selected semester and sends them as JSON. 6. showStudentDashboard() Filters lecturers based on the semester and returns the view with the relevant data. 7. setQuota() Placeholder for setting quota logic 8. showDashboard() Fetches quota data for the selected semester and prepares it for display

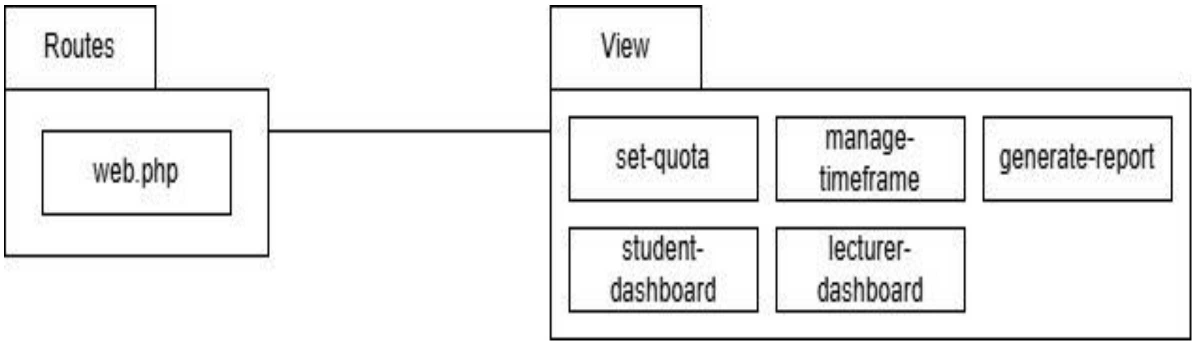
4.4.5 TimeframeController [SDD-REQ-405]

Class Type	Controller Class

Responsibility	The TimeframeController is responsible for managing the timeframes related to the supervisor hunting process, including displaying and saving start and end dates. It handles the logic related to timeframe management.	
Attributes	Attributes Name	Attributes Type
	start_date end_date show_popup success	string string boolean boolean
Methods	Method Name	Description

	showTimeframe()
	
	saveTimeframe()
		Displays the manage-timeframe page where timeframes can be set.
		
		Saves the selected start and end dates in session and redirects with a success message.

Algorithm	1. showTimeframe() Returns the view for managing timeframes (manage-timeframe). 2. saveTimeframe() Saves the dates in the session to persist them temporarily.	



4.4.6 set-quota [SDD-REQ-406]

Class Type	View Class	
Responsibility	Displays a list of lecturers for the selected semester and allows users to update quotas.	
Attributes	Attributes Name	Attributes Type
	lecturers semester	array of objects string
Methods	Method Name	Description

	index()
	
	store()
		Retrieves lecturers based on the selected semester and passes them to the view.
		
		Updates the lecturers' quotas based on user input and refreshes the view.

Algorithm	1. index() Fetches lecturers based on semester. 2. store() Loops through lecturers and updates their quotas.	

4.4.7 manage-timeframe [SDD-REQ-407]

Class Type	View Class	
Responsibility	Provides an interface to set or view the start and end dates of a timeframe.	
Attributes	Attributes Name	Attributes Type
	start_date	string

	end_date success show_popup	string boolean boolean
Methods	Method Name	Description

	showTimeframe()
	
	saveTimeframe()
		Displays the view for managing the timeframe.
		
		Saves the entered start and end dates and updates the session.

Algorithm	1. showTimeframe() Simply loads the manage-timeframe.blade.php view. 2. saveTimeframe() Retrieves start and end dates from the request.	

4.4.8 generate-report [SDD-REQ-408]

Class Type	View Class	
Responsibility	Displays lecturer information and reports for a selected semester.	
Attributes	Attributes Name	Attributes Type
	lecturer semester	array of objects string
Methods	Method Name	Description
	generateReport()	Fetches and filters lecturers based on the semester and generates a report.
Algorithm	1. generateReport() Accepts a semester from the request and filters lecturers based on the semester.	

4.4.9 lecturer-dashboard [SDD-REQ-409]

Class Type	View Class	
Responsibility	Displays information specific to a lecturer, including quota data and notifications.	
Attributes	Attributes Name	Attributes Type
	lecturer semesters quotaData notifications	object array object boolean
Methods	Method Name	Description
	showDashboard()	Fetches lecturer details, quota data, and notifications for the dashboard.
Algorithm	1. showDashboard() Accepts lecturer name and semester and fetches lecturer details.	

4.4.10 student-dashboard [SDD-REQ-410]

Class Type	View Class	
Responsibility	Displays available lecturers and their details for students, filtered by semester.	
Attributes	Attributes Name	Attributes Type
	lecturers semester	array of objects string
Methods	Method Name	Description
	showStudentDashboard()	Displays lecturers and their details filtered by the selected semester.
Algorithm	1. showStudentDashboard() Accepts semester from the request and filters lecturers based on semester.	

5. REQUIREMENT TRACEBILITY

5.1 Use Case [SRS-REQ-100] - [SRS-REQ-400]

Requirement ID	Description	Design ID
SRS_REQ_101	This model allows for seamless interaction with the database, such as fetching, updating, and managing user records, and often includes additional features like accessors, mutators, and relationships to other models for extended functionality.	SDD_REQ_101
SRS_REQ_102	FYP Coordinator manages the user authentication process by receiving the login form data (user ID and password), validating the input, and attempting to authenticate the user. If authentication is successful, it redirects the user to their respective dashboard; if not, it returns an error message and prompts the user to try again. The controller typically leverages Laravel's built-in Auth system to simplify the login and session management processes. It also handles any necessary validation of the login credentials before attempting to authenticate.	SDD_REQ_102
SRS_REQ_103	FYP Coordinator manages user profile details, including displaying and updating information like name, email, and password. It also handles profile picture uploads and ensures secure validation of user inputs during updates. Key functions include displaying the profile, updating	SDD_REQ_103

	details, changing passwords, and uploading profile pictures.	
SRS_REQ_104	It includes functionality to create, update, view, and delete user records. The controller ensures proper handling of user registration, login, and profile management. It may also manage user roles, permissions, and authentication. The UserController serves as an interface between the user and the backend logic, ensuring data is properly validated and processed before interacting with the database.	SDD_REQ_104
SRS_REQ_105	The **generateReport** function retrieves a student's details by their ID, checks for their existence, and displays their information (e.g., ID, name, email, courses) in an HTML view for review. The **printReport** function works similarly but provides a print-friendly version of the report for physical documentation. Both functions ensure error handling for invalid or missing student records.	SDD_REQ_105 SDD_REQ_106
SRS_REQ_106	Users can update their password by entering their current password, a new password, and confirming the new password. It includes real-time error validation for incorrect inputs and displays success messages upon successful password updates. Designed with Bootstrap for a clean and formal	SDD_REQ_107

	look, it also features a button to return to the user's dashboard.	
SRS_REQ_107	It displays a list of lecturers, their details, and may allow interaction with specific lecturer-related functions such as assigning or viewing research fields. The dashboard could include buttons for adding research fields, updating personal information, and navigating to other relevant sections, like managing students or viewing reports. The LecturerDashboard ensures that the lecturer's data is organized and accessible for efficient management.	SDD_REQ_108
SRS_REQ_108	Students manage their information and perform various actions related to their academic progress. It typically includes features like displaying student details (e.g., ID, name, email), course lists, and options to generate or print reports. Students may have access to functionalities such as updating their profile, viewing grades, and managing their course registrations. The dashboard is user-friendly and provides a centralized place for students to interact with their academic data and perform necessary actions such as report generation.	SDD_REQ_109
SRS_REQ_109	These handle navigation to different sections of the application based on user roles (e.g., redirecting to student or lecturer dashboards), displaying key	SDD_REQ_110
		SDD_REQ _111

	information, and allowing access to core features (like report generation or profile updates). The login function validates the user's credentials, checks for errors, and redirects the user to the appropriate dashboard if successful. It also displays error messages for failed login attempts and handles form submissions securely.	
SRS_REQ_201	The lecturer should be able to upload their timetable for the respective semester.	SDD_REQ_201
		SDD_REQ_202
		SDD_REQ_212
SRS_REQ_202	The student should be able to select their prospective supervisor from the registered lecturer in the system.	SDD_REQ_201
		SDD_REQ_202
		SDD_REQ_213
SRS_REQ_203	The student should be able to view the respective lecturer's timetable.	SDD_REQ_201
		SDD_REQ_202
		SDD_REQ_213
SRS_REQ_204	The student should be able to apply for an appointment with the respective lecturer by inputting the date and time from the calendar.	SDD_REQ_201
		SDD_REQ_202
		SDD_REQ_209
		SDD_REQ_213
SRS_REQ_205	The lecturer should be able to respond (approve or reject) to the requested appointment.	SDD_REQ_202
		SDD_REQ_210
		SDD_REQ_204
		SDD_REQ_212
SRS_REQ_206	If the lecturer does not respond to an appointment request within 24 hours of the requested date, the system will automatically send a reminder.	SDD_REQ_203
		SDD_REQ_214
SRS_REQ_207		SDD-REQ-205

	The student should be able to view their appointment request status from the appointment page (Pending, Approved, or Rejected).	SDD_REQ_209
		SDD_REQ_207
		SDD_REQ_213
SRS_REQ_208	The system will send the reminder to the respective lecturer and student one day before their appointment date.	SDD_REQ_203
		SDD_REQ_214
		SDD_REQ_213
SRS_REQ_209	Students cannot book overlapping appointments with different lecturers.	SDD_REQ_202
		SDD_REQ_210
SRS_REQ_210	Both students and lecturers can cancel an appointment before the scheduled date.	SDD_REQ_202
		SDD_REQ_210
		SDD_REQ_213
SRS_REQ_211	The system prevents appointments from being scheduled outside working hours or overlapping lecturer-blocked slots	SDD_REQ_202
		SDD_REQ_210
SRS_REQ_301	The system should allow students to apply for a Final Year Project (FYP) proposal.	SDD_REQ_300
SRS_REQ_302	The system must allow students to view the status of their application (e.g., Pending, Approved, Rejected).	SDD_REQ_301
SRS_REQ_303	The system should send a notification to students when a lecturer approves or rejects their application.	SDD_REQ_302, SDD_REQ_306
SRS_REQ_304	The system must allow lecturers to post new FYP proposals.	SDD_REQ_303, SDD_REQ_308
SRS_REQ_305	The system should enable lecturers to review and approve/reject student applications.	SDD_REQ_304, SDD_REQ_305, SDD_REQ_312
SRS_REQ_306	Students should be able to view all available FYP proposals from lecturers.	SDD_REQ_305

SRS_REQ_307	The system should update the proposal status and notify students when a lecturer approves or rejects an application.	SDD_REQ_306
SRS_REQ_308	The system must ensure that students cannot apply for more than one proposal from the same lecturer.	SDD_REQ_307
SRS_REQ_309	Lecturers should be able to manage their available slots for proposals (e.g., adjusting quotas).	SDD_REQ_308
SRS_REQ_310	Students must be able to track the status of their applications through a submission tracking page.	SDD_REQ_309, SDD_REQ_313
SRS_REQ_400	Enable the FYP Coordinator to manage timeframe and quotas for supervisor allocation.	SDD_REQ_402, SDD_REQ_403, SDD_REQ_405, SDD_REQ_406 SDD_REQ_407
SRS_REQ_401	Validate the entered dates to ensure the end date is not earlier than the start date	SDD_REQ_403
SRS_REQ_402	Allow the coordinator to cancel saving the timeframe changes.	SDD_REQ_403, SDD_REQ_405
SRS_REQ_403	Provide functionality for generating and printing reports on lecturer quotas.	SDD_REQ_404, SDD_REQ_408
SRS_REQ_404	Require users (students and lecturers) to select a semester before quota data is displayed.	SDD_REQ_409, SDD_REQ_410

SRS_REQ_405	Restrict access to quota data until the FYP Coordinator has updated the information.	SDD_REQ_402, SDD_REQ_404
SRS_REQ_407	Provide lecturers the view of the dashboard	SDD_REQ_409