Enrollment of Tuition Center Application

By

Ong Yi Xin

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY
KUALA LUMPUR

ACADEMIC YEAR
2025/26

Tuition Center App: Subject, Class, Other Service, Payment, Chat

By

Ong Yi Xin

Supervisor: Ms. Yeoh Kar Peng

A project report submitted to the
Faculty of Computing and Information Technology
in partial fulfillment of the requirement for the
Bachelor of Information Technology (Honours)

Faculty of Computing and Information Technology
Tunku Abdul Rahman University of Management and Technology
Kuala Lumpur

Copyright by Tunku Abdul Rahman University of Management and Technology.
All rights reserved. No part of this project documentation may be reproduced, stored in retrieval system, or transmitted in any form or by any means without prior permission of Tunku Abdul Rahman University of Management and Technology.

Declaration

The project submitted herewith is a result of my own efforts in totality and in every aspect of the project works. All information that has been obtained from other sources had been fully acknowledged. I understand that any plagiarism, cheating or collusion or any sorts constitutes a breach of TAR University rules and regulations and would be subjected to disciplinary actions.

_____

Ong Yi Xin

Bachelor of Information Technology (Honours) in Software Systems Development

ID: 24WMR09097

Abstract

EaseTuition is a smart tuition management system designed to address two common **challenges** faced by tuition centers: **manual scheduling** and **inefficient payment processes**. Manual scheduling can lead to scheduling confusion and miscommunication, while outdated payment systems often result in delays and errors. This project aims to address these issues by introducing an integrated platform that automates core management functions and improves overall operational efficiency.

The system integrates **modules** for **subject** and **class management**, **timetable generation**, **other service processing**, **payment automation** and **centralized personal messaging**. The core highlight is implementing an **AI-powered timetable evaluation engine** by comprehensively considering students lifestyle, academic performance, workload and structural factors to identify optimal timetable solutions. EaseTuition was developed using **Python** for backend control handling, **React Native** to build the UI, **MySQl** in Xampp as a localhost database and **Stripe** for implementing the online payment.

The project used **incremental modeling** as a development methodology to allow for systematic functional expansion, continuous validation and flexible optimization throughout implementation. Testing included **cross-module integration testing** to ensure seamless data flow such as verifying how subjects map to classes, how class details integrate into class schedule scheduling and how service, enrollment and payment modules collaborate to generate precise invoices. Test results confirm the system maintains stable operation during interconnected operations.

In conclusion, EaseTuition delivers with the main goal of providing tuition centers with an efficient, automated, and transparent management platform. The operational reliability can be improved through the system by reducing manual operations, minimizing human error and enhancing communication efficiency. It also delivers a more professional and convenient user experience for administrators, tutors, parents and students.

Acknowledgement

I would like to express my sincere gratitude to all those who have contributed directly or indirectly to the completion of this project proposal.

First and foremost, I would like to express my sincere gratitude to my project mentor, Ms. Yeoh Kar Peng, who provided clear guidance and invaluable advice that laid the foundation for my project. Her insights helped refine my idea and ensure the viability of the Tutorial Center App.

I would also like to express my sincere gratitude to my family for their unwavering support throughout the process. They provided me with a quiet and comfortable environment where I was able to focus on my final project without any distractions. Their encouragement was a source of motivation for me to move forward.

I would like to give special thanks to my team member, Sia Keng Loon, for his excellent cooperation and communication. His constructive feedback and suggestions helped me improve my part of the project and ensured a more comprehensive outcome.

Last but not least, I would like to thank my friends Jun Wei and Ming Yi for being my pillars. They were very understanding and kept encouraging me whenever I felt overwhelmed or stressed. Their motivation has helped me to overcome difficulties and persevere in my work.

To all of you who have contributed in any way, I sincerely thank you for your support and help in making this project possible. Thank you all.

Table of Contents

Chapter 1

Introduction

1.1 Objectives

1.1.1 Intelligent Scheduling and Academic Planning

EazeTuition's main objective is to design and implement an intelligent automated system to **improve the scheduling, communication and payment processes** in tuition centers. The system aims to introduce AI-driven scheduling based on student and tutor preferences, work schedules and operational constraints to reduce conflicts and waiting times, thereby eliminating inefficiencies associated with manual scheduling and improving the learning and teaching state of both students and tutors.

It also aims to **improve subject and class management** through flexible subject creation, batch import, class type differentiation and customized class packages, enabling personalized and **scalable academic planning** that is not limited by a single curriculum model.

1.1.2 Centralized and Secure Communication

Another key objective is to **centrally manage communication** between staff and parents using an integrated one-to-one chat system, replacing fragmented external messaging platforms and ensuring secure and organized communication.

1.1.3 Streamlined Payments and Service Integration

On the payment side, EazeTuition aims to **streamline fee processing** by providing a flexible online and offline payment module that automatically calculates fees based on class choices, and other services. The system also supports multiple payment methods, a deposit-based structure and instant e-receipt generation to minimize manual calculation errors and increase financial transparency.

Finally, additional service management and payment system integration **ensures accurate and flexible billing models** for services such as child care and transportation. Together, these objectives aim to reduce administrative overhead and increase financial transparency.

1.2 Problems

1.2.1 Manual Scheduling and Communication Barriers in Tuition Centers

1. **Unbalanced and Inefficient Schedules from Manual Processes**

   In many tutoring centers, scheduling is still managed manually by administrators or staff, which creates many challenges for both students and tutors. One of the main issues is the creation of **uncomfortable and unbalanced schedules**. This often leads to **insufficient breaks**, such as insufficient time for lunch or dinner, or long wait times between classes, resulting in wasted time and inefficient daily work.

2. **Time-Consuming and Error-Prone Timetable Creation**

   The process of **creating timetables manually** is also time-consuming and error-prone. Staff must constantly refer to previous schedules and manually cross-check multiple calendars, making it difficult to create optimal schedules that balance the time available to students and counselors. This lack of automation increases the likelihood of scheduling conflicts, such as overlapping class times or misaligned breaks.

   It also **increases the difficulty** of meeting special scheduling requirements without disrupting the entire schedule. There is a study that has proven that manual scheduling is a time-consuming process as manually creating a schedule often takes 12 to 15 days to complete. In contrast, an automated scheduling system can significantly reduce this time, completing the same task in just 3 to 5 days (Nsulangi et al.).

3. **Limitations in Supporting Diverse Student Needs**

   Manual systems also make it **difficult to support students in different subjects or class structures**. For example, students who wish to take multiple courses or who require intensive instruction rather than regular group instruction often cannot be accommodated. Most tutoring centers are only equipped to handle basic multi-student courses, which limits the potential for personalized education and flexible learning paths.

4. **Miscommunication and Confusion with Parents**

Additionally, **miscommunication with parents** is a common problem. Scheduling information is often shared through lengthy and cluttered WhatsApp threads, where key details can easily be missed or misunderstood. This leads to confusion, frustration, and a lack of trust between parents and centers.

1.2.2 Time-consuming and inefficient payment process

1. **Inefficiencies and Errors in Manual Payment Processing**

   Currently, the payment process in many tuition centers is still **l**argely manual and face-to-face, leading to **inefficiencies** that affect both staff and parents. One of the most common problems is **long waiting times**. Parents often have to wait in line while tutors manually verify records and calculate fees, which leads to unnecessary delays and frustration. In addition, manual tracking increases the likelihood of human error, such as miscalculations, overlooked payments, or input errors. Studies have shown that manual payment systems can have error rates as high as 15-20%, which can undermine trust and create an administrative burden (Lee & Park, 2020)

2. **Inconvenience and Lack of Flexibility for Parents**

   Another major challenge is the **lack of payment flexibility**. Without an online system, parents have to physically visit tuition centers during working hours, which is inconvenient, especially for working parents. This inconvenience makes the payment process more **stressful and time-consuming** for families. According to the U.S. Bureau of Labor Statistics (2020), more than 40% of working parents say they have difficulty taking care of basic tasks, including paying bills, due to conflicts with standard work schedules.

3. **Challenges of Managing Complex Fee Structures Manually**

   Manually dealing with **complex fee structures** adds another layer of difficulty. Many tutorial centers offer not only academic courses, but also services such as transportation and childcare. These services often involve different fee models, for example, direct payment at the time of enrollment, or a deposit model with monthly fees based on attendance. Managing these different models manually is time-consuming and error-prone, often resulting in inconsistency, confusion and disputes.

1.3 Advantages and Contributions

1.3.1 Enhanced Scheduling and Academic Flexibility

EazeTuition has significantly improved the way tuition centers manage their schedules by introducing an AI automated scheduling feature. The system automatically generates optimal timetables taking into account tutor availability, student preferences and operational constraints. As a result, the **time required** for timetable planning is significantly **reduced** and managers **no longer need to manually plan and cross-check timetables**. Issues such as overlapping classes and uneven breaks between classes can be addressed more effectively, thus enhancing the overall teaching and learning experience for both students and tutors.

In addition, the system supports flexible subject management and class type differentiation, enabling administrators to create regular and focused classes according to the specific needs of students. Batch import and batch scheduling functions further streamline data entry, enabling staff to process large amounts of academic data faster and with lower error rates. Together, these features provide a **more personalized, efficient and scalable learning environment** for students.

1.3.2 Centralized Communication and Improved Parent Engagement

The system includes a built-in one-to-one chat module that centralizes communication between staff and parents. This eliminates the reliance on external platforms such as WhatsApp, which tend to be cluttered. By linking messages directly to each student's and parent's personal profile, EazeTuition ensures that communication is organized, searchable, and secure. This **improves clarity**, **reduces misunderstandings**, and provides a transparent, reliable, and time-saving communication channel between tuition centers and parents, thus **fostering a stronger, more professional relationship**. It also ensures that important notifications, schedule changes or fee-related information is received in a timely and verifiable manner.

1.3.3 Efficient Payment Handling for Transparency and Parent Convenience

EazeTuition has transformed the payment experience for tuition center staff and parents by automating complex fee calculations and reducing reliance on manual processes. This automation not only eliminates the human error that is common in manual tracking and calculations, but also significantly reduces the amount of staff time spent on routine administrative tasks. As a result, tuition center operations have become **more efficient**, and staff can redirect their efforts to **more value-added activities**, such as academic planning or student support.

For parents, flexible billing structures such as direct debit and deposit programs **offer greater financial control and convenience**. They no longer need to travel to centers or wait in long lines during business hours, which is especially beneficial to working families. The system's ability to generate and store digital receipts instantly further builds trust, as parents have full access to their payment history and can easily verify past transactions without having to contact staff.

In addition, the integration between the service module and the payment system ensures accurate calculation of fees for services such as transportation and childcare, **avoiding misunderstandings or disputes**. All in all, EazeTuition has not only modernized the financial workflow of tuition centers, it has also improved transparency, reduced parental stress, and created a more professional and trustworthy image for the institution.

1.4 Project Plan

1.4.1 Project Scope

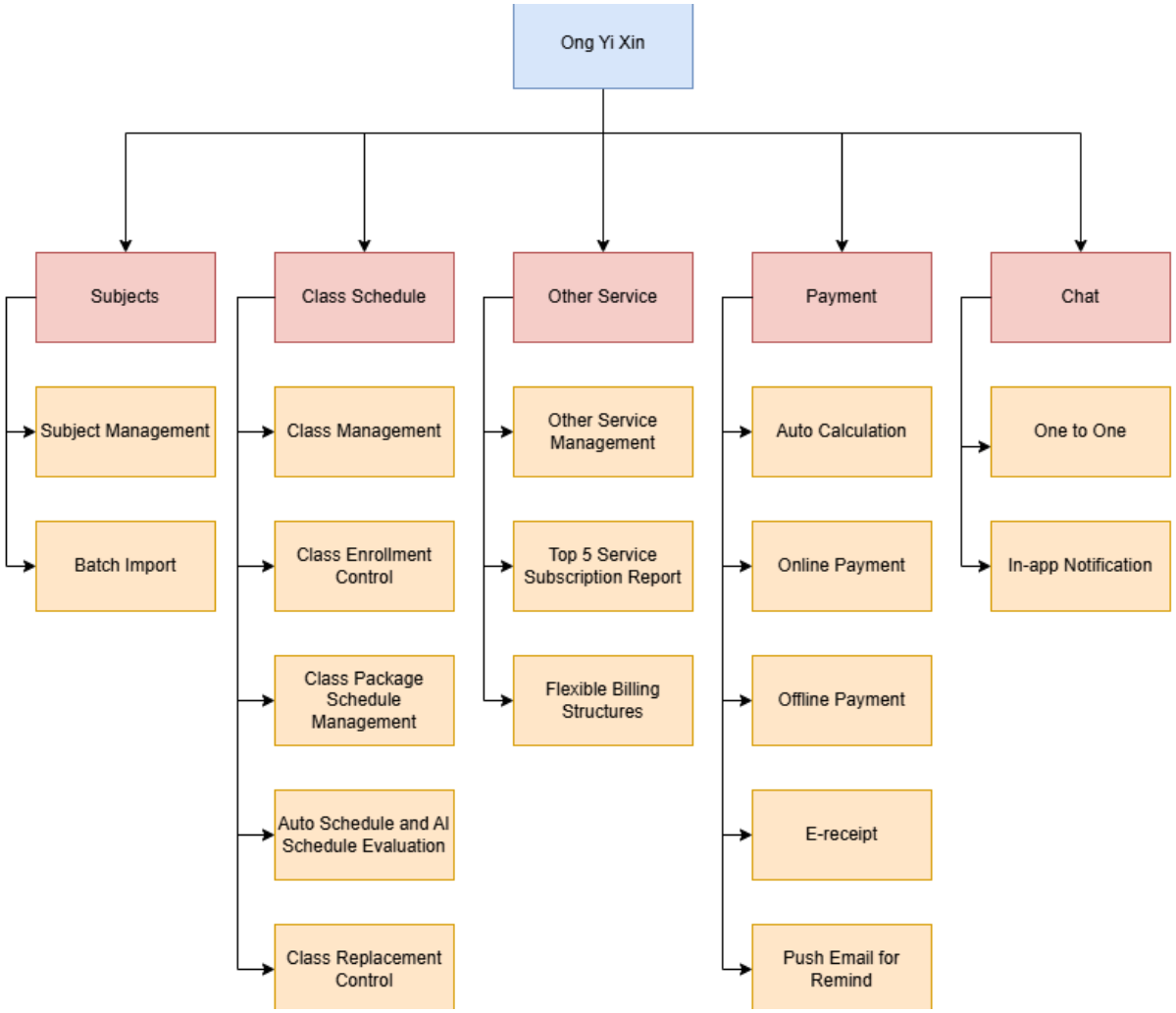


Figure 1.1 Hierarchical Chart of Project Scope

1.4.2 Functional Feature

1.0 Subject Module

1.1 Subject Management

1.1.1 The system shall allow administrators to view all created subjects with sorting, filtering, and pagination features to support efficient navigation.

1.1.2 The system shall allow administrators to create new subjects with a customizable subject name and multiple class-type fee configurations, including but not limited to “Normal” and “Focus” class fee structures.

1.1.3 The system shall validate new subject entries by performing duplicate detection based on subject name and fee type combinations, preventing administrators from creating subjects with identical definitions.

1.1.4 The system shall allow administrators to update existing subject details, including subject name and individual fee settings for each class type.

1.1.5 The system shall allow administrators to deactivate outdated or unused subjects, preventing them from being used in future class creation.

1.1.6 When a subject is deactivated, the system shall automatically cascade the deactivation to all related entities.

1.1.7 The system shall allow administrators to reactivate previously deactivated subjects when needed, restoring their availability for class creation and scheduling activities.

1.2 Batch Subject Import

1.2.1 The system shall allow administrators to upload a CSV or Excel file containing subject data.

1.2.2 The system shall validate imported data by checking duplicate subject name, duplicate subject, education level and fee structure and incomplete or invalid fields.

1.2.3 system shall store all successfully validated subjects into the database in bulk, while rejecting invalid rows with detailed error messages.

2.0 Class Schedule Module

2.1 Class Management

2.1.1 The system shall allow administrators to view created classes with support for sorting, filtering, and pagination.

2.1.2 The system shall allow administrators to create, edit, and deactivate classes.

2.1.3 The system shall allow each class to be associated with a specific subject, tutor and maximum student capacity.

2.1.4 The system shall allow administrators to configure class types (e.g., Normal, Focus) and ensure the selected subject supports the selected class type.

2.1.5 The system shall allow tutors to view their own assigned classes in a dedicated class view.

2.1.6 The system shall allow parents to view classes available for enrollment.

2.1.7 The system shall allow students and parents to view enrolled classes in a dashboard or calendar interface.

2.1.8 The system shall perform cascading deactivation. When a subject is deactivated, all related classes and class-detail schedules shall automatically be deactivated.

2.1.9 The system shall validate room usage to prevent multiple classes from using the same room at overlapping times.

2.1.10 The system shall allow administrators to toggle whether a class is “involved for scheduling”, enabling them to control whether the class should be included in automatic schedule generation.

2.2 Class Enrollment Control

2.2.1 The system shall allow parents to select available classes and enroll their children without requiring prior subject enrollment.

2.2.2 The system shall allow administrators to perform class enrollment on behalf of students.

2.2.3 The system shall enforce class capacity limits to prevent over-enrollment.

2.2.4 The system shall prevent students from enrolling in overlapping class time slots.

2.2.5 The system shall validate registration rules, including capacity, duplication, class-type restrictions and schedule conflicts in real time and display warning messages when issues occur.

2.2.6 The system shall finalize enrollment only after successful payment.

2.2.7 The system shall allow staff to view, filter and manage the list of enrolled students for each class.

2.2.8 The system shall automatically generate related payment records upon successful enrollment submission.

2.2.9 The system shall prevent enrollment into deactivated classes or classes belonging to deactivated subjects.

2.3 Rule-Based Auto Scheduling

2.3.1 The system shall automatically generate class schedules using predefined rule-based scheduling algorithms.

2.3.2 The system shall avoid schedule conflicts by ensuring no overlapping time slots for the same student, tutor or room.

2.3.3 The system shall allocate rooms based on availability and capacity constraints.

2.3.4 The system shall ensure each generated class package includes at least one class per selected subject.

2.3.5 The system shall detect conflicts dynamically and reassign class details when necessary.

2.3.6 The system shall auto-generate time slots for unassigned classes based on predefined scheduling windows such as morning, afternoon, full-day.

2.3.7 The system shall prioritize the placement of fixed-time classes before generating time slots for flexible classes.

2.3.8 The system shall re-evaluate schedules when leftover or missing subject classes are detected.

2.3.9 The system shall prevent tutor double-booking across all generated schedule sets.

2.3.10 The system shall validate room availability during the auto-generation process to avoid room conflicts.

2.4 AI Schedule Evaluation

2.4.1 The system shall compute lifestyle scores using the trained Lifestyle Model based on the student's stress level, study time and sleep duration.

2.4.2 The system shall compute performance scores using the trained Performance Model based on study–rest balance.

2.4.3 The system shall calculate a weighted recommendation score combining lifestyle and performance metrics.

2.4.4 The system shall classify schedules into ranked categories, such as "Highly Recommended," "Recommended," "Moderate," and "Not Recommended."

2.4.5 The system shall display evaluation results for each generated schedule set without modifying the rule-based schedule.

2.4.6 The system shall refresh the AI evaluation score whenever a schedule package is regenerated or updated.

2.5 Class Package Schedule Management

2.5.1 The system shall allow staff to view suggested class packages generated through the auto-scheduling module.

2.5.2 The system shall allow staff to deactivate generated schedule packages but shall not allow editing of package contents.

2.5.3 The system shall apply signature-based integrity validation to ensure schedule package data is not modified when transmitted between backend and frontend.

2.5.4 The system shall block activation of schedule packages that contain unresolved scheduling conflicts, invalid class combinations, missing subjects, duplicated class slots or room allocation conflicts.

2.5.5 The system shall allow administrators to toggle the "Open for Registration" status for schedule packages, enabling parents to enroll only when a package is explicitly set as available.

2.5.6 The system shall ensure that any class or subject deactivation automatically propagates to linked schedule packages, updating their availability accordingly.

2.5.7 The system shall automatically update the status of scheduled class entries within the timetable when a linked class, subject, tutor assignment, or room allocation changes or is deactivated.

2.6 Centralized Messaging

2.6.1 The system shall allow administrators to cancel any scheduled class within the generated timetable, ensuring the cancellation is instantly reflected in all parent-facing, student-facing and staff-facing views.

2.6.2 The system shall support creating a replacement class to compensate for a canceled lesson, enabling administrators to select a new time slot, tutor and room where applicable.

2.6.3 The system shall prevent replacement class creation if the proposed replacement time slot conflicts with existing student schedules, tutor schedules, room allocations or subject-level constraints.

2.6.4 The system shall ensure replacement classes follow the same validation rules and constraints as regular scheduled classes, including tutor availability, room usage and cross-subject enrollment conflicts.

2.6.5 The system shall maintain a complete replacement history, enabling administrators to review all canceled-and-replaced lesson chains.

3.0 Chat Module

3.1 Centralized Messaging

3.1.1 The system shall allow staff and parents to send and receive one-to-one messages.

3.1.2 The system shall store and display message history capability.

3.1.3 The system shall send in-app push notifications to alert staff and parents of new messages.

4.0 Payment Module

4.1 Online and Offline Payment

4.1.1 The system shall support online payment through Stripe, including validation, checkout redirection and webhook confirmation.

4.1.2 The system shall allow staff to record offline payments, including cash, bank transfer or partial payments.

4.1.3 The system shall validate received cash amounts and automatically compute the change amount for offline transactions.

4.1.4 The system shall prevent duplicate payments for the same billing record regardless of online or offline method.

4.2 Fee Calculation

4.2.1 The system shall calculate tuition fees based on enrolled class type such as Normal or Focus, subject fee configuration and class count.

4.2.2 The system shall support automatic application of class-based tiered discounts when students enroll in multiple tuition classes.

4.2.3 The system shall calculate service fees such as transport, childcare using the appropriate billing model such as monthly fixed fee, attendance-based fee or deposit logic.

4.2.4 The system shall dynamically recompute total payable amounts when enrollment changes occur such as class enrollment or cancellation or service modifications.

4.2.5 The system shall allow parents to preview a full fee breakdown before confirming payment.

4.3 Payment Generation & Billing Lifecycle

4.3.1 The system shall automatically generate a pending payment record upon class enrollment or service registration, regardless of whether initiated by administrators or parents.

4.3.2 The system shall generate recurring monthly payments automatically based on active enrollments and active services at the beginning of each billing cycle.

4.3.3 The system shall prevent monthly charges from being created for deactivated classes or canceled services.

4.3.4 The system shall automatically cancel pending payments when the related enrollment, class or service is canceled before payment is made.

4.3.5 The system shall automatically link billing items to the correct enrollment, service or replacement attendance record for audit accuracy.

4.4 Payment Status & Enrollment Synchronization

4.4.1 The system shall automatically update payment status such as Paid or Unpaid when receiving Stripe webhook results or offline payment confirmation.

4.4.2 The system shall confirm class enrollments only when payment status is updated to “Paid”.

4.4.3 The system shall automatically mark enrollment as inactive (canceled) if the corresponding payment is removed or fails under specific business rules.

4.4.4 The system shall automatically bar student enrollment when payments remain overdue beyond a configurable threshold.

4.4.5 The system shall automatically unbar a student's enrollment eligibility once outstanding payments are settled.

4.5 Payment Plans

4.5.1 The system shall enable direct-pay tuition fees that require full payment at enrollment.

4.5.2 The system shall support deposit-based payment plans for childcare and transport services.

4.5.3 The system shall validate whether deposits have been fully cleared before generating monthly service charges.

4.5.4 The system shall support hybrid billing models where a student may have both tuition charges and attendance-based service fees simultaneously.

4.6 Digital Receipts & Payment History

4.6.1 The system shall allow parents to view, download or print past receipts in the application.

4.6.2 The system shall clearly distinguish payment types such as enrollment fees, monthly tuition fees, service fees, deposit payments and adjustments.

5.0 Other Service Module

5.1 Service Management

5.1.1 The system shall allow administrators to view all created services with sorting, filtering, and pagination support.

5.1.2 The system shall allow staff to create new services, including transport and childcare, with customizable fee rules and availability constraints.

5.1.3 The system shall allow staff to edit service details, including naming, fee structure, billing model and deposit requirements.

5.1.4 The system shall allow administrators to deactivate services, preventing future enrollments while preserving historical records.

5.1.6 The system shall allow the configuration of multiple service billing types, including fixed monthly fees, deposit with attendance-based fees.

5.1.7 For transportation services, the system shall allow parents and staff to specify pickup and drop-off addresses during enrollment.

5.1.8 The system shall validate required fields for each service type.

5.2 Service Enrollment Management

5.2.1 The system shall allow parents to self-register their children for available services through the mobile application.

5.2.2 The system shall allow administrators to register students for services on behalf of parents.

5.2.3 The system shall validate duplicate registrations and prevent students from enrolling in the same service twice.

5.2.4 The system shall validate that the selected service is active and eligible before confirming enrollment.

5.2.5 The system shall enforce service-type–specific enrollment rules, including transport requiring valid pickup or drop-off addresses.

5.2.6 The system shall automatically deactivate service enrollments when students are barred or withdrawn.

5.3 Payment Integration

5.4.1 The system shall automatically generate pending payments when a new service enrollment is created.

5.4.2 The system shall generate monthly payments for services that operate under recurring billing models.

5.4.3 The system shall ensure accurate calculation of service-related fees using the correct billing model such as fixed monthly and deposit with attendance-based.

5.4.4 The system shall automatically cancel or adjust pending service-related payments when an enrollment is canceled or modified.

1.4.3 Non-Functional Feature

1.0 Performance

1.1 The system shall support at least 100 concurrent users without significant degradation in response time.

1.2 The system shall respond to user actions within 2 seconds on average.

2.0 Reliability

2.1 The system shall be available 97.5% of the time during operational hours.

2.2 The system shall ensure that critical data (e.g., payments, enrollments) is never lost and always recoverable.

3.0 Security

3.1 The system shall require authenticated login for all users.

3.2 The system shall encrypt sensitive data in storage and transmission.

3.3 The system shall provide role-based access control to protect restricted features and data.

4.0 Usability

4.1 The system shall provide a user-friendly interface accessible to staff, parents, and tutors.

4.2 The system shall provide help documentation and tooltips for important actions.

5.0 Maintainability

5.1 The system shall be modular and follow coding best practices to support future updates.

Organization

- The system shall be developed using Python for backend APIs and React Native (JavaScript/TypeScript) for the mobile application.
- The system shall use XAMPP with phpMyAdmin for managing the MySQL database and backend services.
- Version control shall be managed via Git and hosted on GitHub.
- UI/UX designs shall be created and iterated using Figma.

External

- The app shall support online payments processed using Stripe in Malaysian Ringgit (MYR).
- All user data handling shall comply with local privacy regulations (e.g., Malaysia PDPA) and follow best practices for security and data protection using XAMPP and phpMyAdmin.
- Future expansion may include compliance with relevant e-commerce and consumer protection laws.

1.4.4 Project Schedule

ID	Task Name	Duration	Start	Finish	Predecessors
1	**EazeTuition System Project**	**178 days**	**Tue 4/1/25**	**Fri 11/28/25**	
2	**Project I**	**96 days**	**Tue 4/1/25**	**Sat 8/9/25**	
3	**Project Planning**	**32 days**	**Tue 4/1/25**	**Wed 5/14/25**	
4	Proposal	15 days	Tue 4/1/25	Mon 4/21/25	
5	**Introduction**	**7 days**	**Tue 4/22/25**	**Wed 4/30/25**	**4**
6	Define Objective	1 day	Tue 4/22/25	Tue 4/22/25	4
7	Identify problniem and proposed solution	2 days	Wed 4/23/25	Thu 4/24/25	6
8	Identify benefits and contributions	1 day	Fri 4/25/25	Fri 4/25/25	7
9	Design projects plan	2 days	Mon 4/28/25	Tue 4/29/25	8
10	Design module hierarchy chart	1 day	Wed 4/30/25	Wed 4/30/25	9
11	**Project background and related work**	**3 days**	**Thu 5/1/25**	**Mon 5/5/25**	**10**
12	Identify target users background	1 day	Thu 5/1/25	Thu 5/1/25	10
13	Project background	2 days	Fri 5/2/25	Mon 5/5/25	12
14	**Technology research and analysis**	**5 days**	**Tue 5/6/25**	**Mon 5/12/25**	**13**
15	Literature review	2 days	Tue 5/6/25	Wed 5/7/25	13
16	Feasibility study	1 day	Thu 5/8/25	Thu 5/8/25	15
17	Hardware and API research	2 days	Fri 5/9/25	Mon 5/12/25	16
18	Project planning review	2 days	Tue 5/13/25	Wed 5/14/25	17
19	**Requirement Analysis**	**14 days**	**Thu 5/15/25**	**Tue 6/3/25**	
20	Identify methodolgy	2 days	Thu 5/15/25	Fri 5/16/25	18
21	Identiffy software process model	1 day	Mon 5/19/25	Mon 5/19/25	20
22	Requirements gathering	2 days	Tue 5/20/25	Wed 5/21/25	21
23	**Requirement analysis**	**3 days**	**Thu 5/22/25**	**Mon 5/26/25**	
24	Use case diagram	2 days	Thu 5/22/25	Fri 5/23/25	22
25	Use case description	1 day	Mon 5/26/25	Mon 5/26/25	24
26	**System requirments specification**	**2 days**	**Tue 5/27/25**	**Wed 5/28/25**	
27	Identify functional and non-functional requirements	2 days	Tue 5/27/25	Wed 5/28/25	25
28	Development platform analysis	2 days	Thu 5/29/25	Fri 5/30/25	27
29	Requirement analysis review	2 days	Mon 6/2/25	Tue 6/3/25	28
30	**Design**	**49 days**	**Wed 6/4/25**	**Fri 8/8/25**	
31	System designs	8 days	Wed 6/4/25	Fri 6/13/25	29
32	User interface design	15 days	Mon 6/16/25	Fri 7/4/25	31
33	**Data design**	**5 days**	**Mon 7/7/25**	**Fri 7/11/25**	
34	ERD diagram design	3 days	Mon 7/7/25	Wed 7/9/25	32
35	Database design	2 days	Thu 7/10/25	Fri 7/11/25	34
36	Report design	7 days	Sat 7/12/25	Mon 7/21/25	35
37	**Process design**	**4 days**	**Tue 7/22/25**	**Fri 7/25/25**	
38	Flowchart	4 days	Tue 7/22/25	Fri 7/25/25	36
39	Software architecture design	5 days	Mon 7/28/25	Fri 8/1/25	38
40	Design review	5 days	Mon 8/4/25	Fri 8/8/25	39

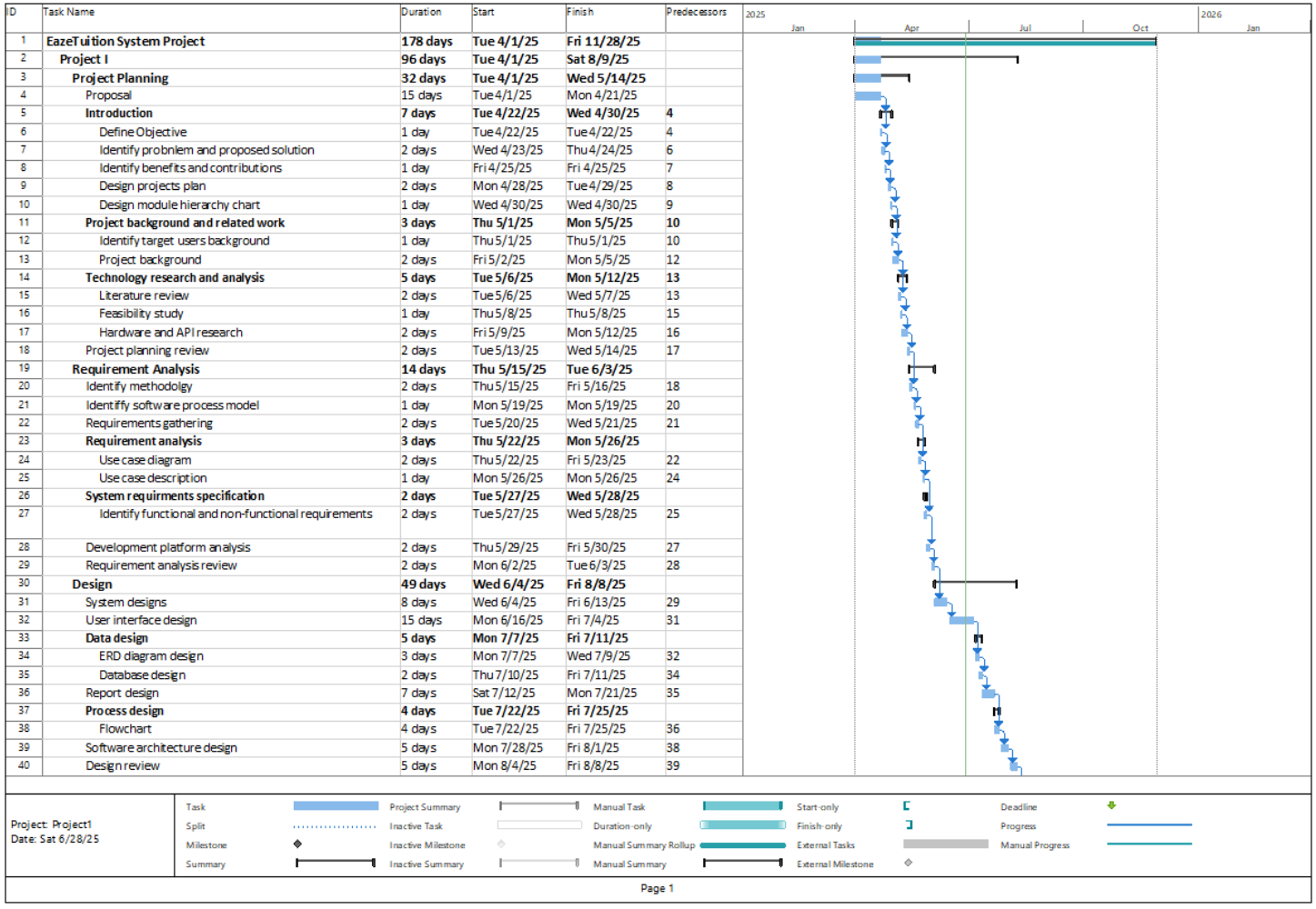
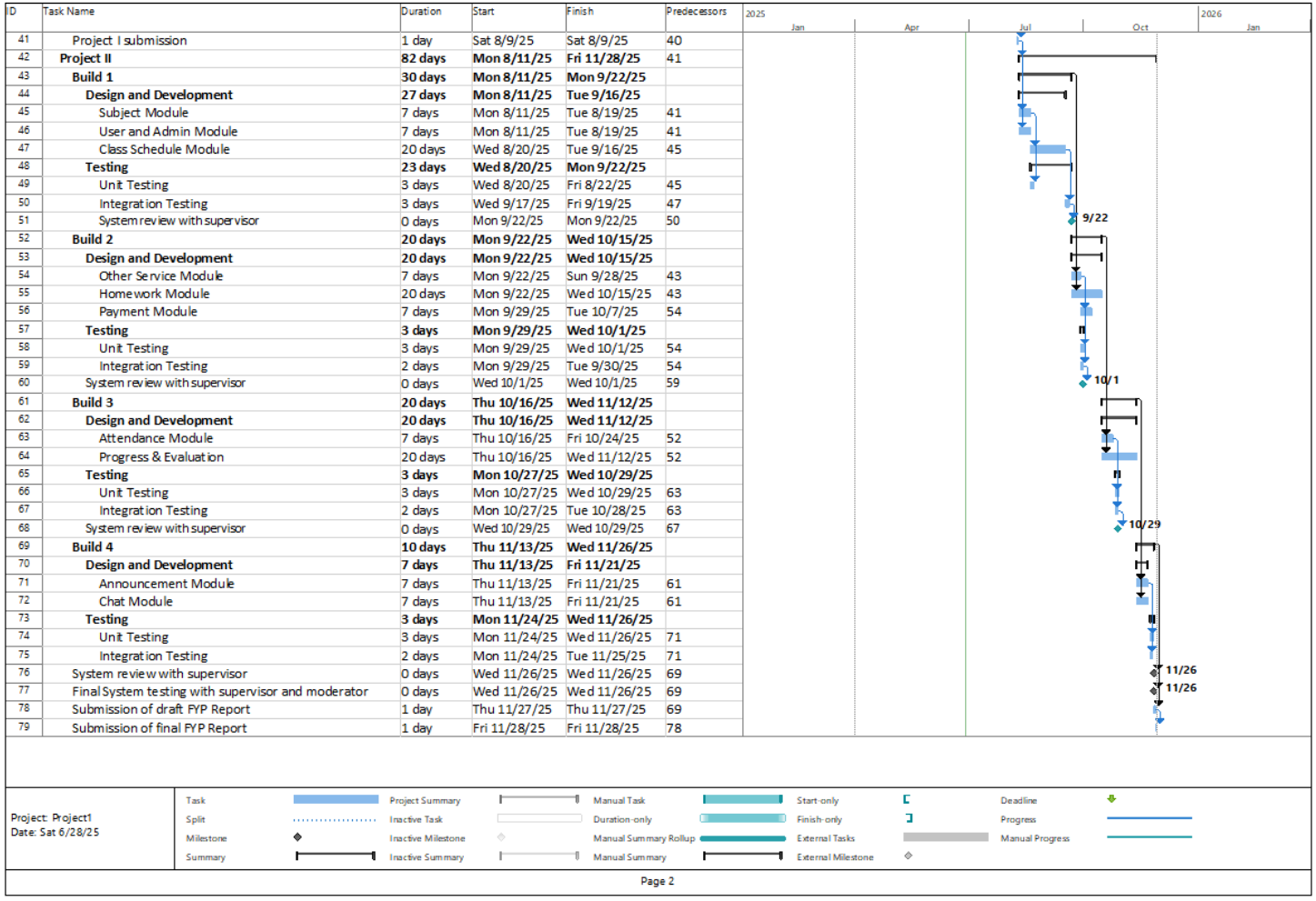


Figure 2.1 Gantt Chart of Project Schedule (Page 1)



ID	Task Name	Duration	Start	Finish	Predecessors
41	Project I submission	1 day	Sat 8/9/25	Sat 8/9/25	40
42	**Project II**	**82 days**	**Mon 8/11/25**	**Fri 11/28/25**	41
43	**Build 1**	**30 days**	**Mon 8/11/25**	**Mon 9/22/25**	
44	**Design and Development**	**27 days**	**Mon 8/11/25**	**Tue 9/16/25**	
45	Subject Module	7 days	Mon 8/11/25	Tue 8/19/25	41
46	User and Admin Module	7 days	Mon 8/11/25	Tue 8/19/25	41
47	Class Schedule Module	20 days	Wed 8/20/25	Tue 9/16/25	45
48	**Testing**	**23 days**	**Wed 8/20/25**	**Mon 9/22/25**	
49	Unit Testing	3 days	Wed 8/20/25	Fri 8/22/25	45
50	Integration Testing	3 days	Wed 9/17/25	Fri 9/19/25	47
51	System review with supervisor	0 days	Mon 9/22/25	Mon 9/22/25	50
52	**Build 2**	**20 days**	**Mon 9/22/25**	**Wed 10/15/25**	
53	**Design and Development**	**20 days**	**Mon 9/22/25**	**Wed 10/15/25**	
54	Other Service Module	7 days	Mon 9/22/25	Sun 9/28/25	43
55	Homework Module	20 days	Mon 9/22/25	Wed 10/15/25	43
56	Payment Module	7 days	Mon 9/29/25	Tue 10/7/25	54
57	**Testing**	**3 days**	**Mon 9/29/25**	**Wed 10/1/25**	
58	Unit Testing	3 days	Mon 9/29/25	Wed 10/1/25	54
59	Integration Testing	2 days	Mon 9/29/25	Tue 9/30/25	54
60	System review with supervisor	0 days	Wed 10/1/25	Wed 10/1/25	59
61	**Build 3**	**20 days**	**Thu 10/16/25**	**Wed 11/12/25**	
62	**Design and Development**	**20 days**	**Thu 10/16/25**	**Wed 11/12/25**	
63	Attendance Module	7 days	Thu 10/16/25	Fri 10/24/25	52
64	Progress & Evaluation	20 days	Thu 10/16/25	Wed 11/12/25	52
65	**Testing**	**3 days**	**Mon 10/27/25**	**Wed 10/29/25**	
66	Unit Testing	3 days	Mon 10/27/25	Wed 10/29/25	63
67	Integration Testing	2 days	Mon 10/27/25	Tue 10/28/25	63
68	System review with supervisor	0 days	Wed 10/29/25	Wed 10/29/25	67
69	**Build 4**	**10 days**	**Thu 11/13/25**	**Wed 11/26/25**	
70	**Design and Development**	**7 days**	**Thu 11/13/25**	**Fri 11/21/25**	
71	Announcement Module	7 days	Thu 11/13/25	Fri 11/21/25	61
72	Chat Module	7 days	Thu 11/13/25	Fri 11/21/25	61
73	**Testing**	**3 days**	**Mon 11/24/25**	**Wed 11/26/25**	
74	Unit Testing	3 days	Mon 11/24/25	Wed 11/26/25	71
75	Integration Testing	2 days	Mon 11/24/25	Tue 11/25/25	71
76	System review with supervisor	0 days	Wed 11/26/25	Wed 11/26/25	69
77	Final System testing with supervisor and moderator	0 days	Wed 11/26/25	Wed 11/26/25	69
78	Submission of draft FYP Report	1 day	Thu 11/27/25	Thu 11/27/25	69
79	Submission of final FYP Report	1 day	Fri 11/28/25	Fri 11/28/25	78

Figure 2.2 Gantt Chart of Project Schedule (Page 2)

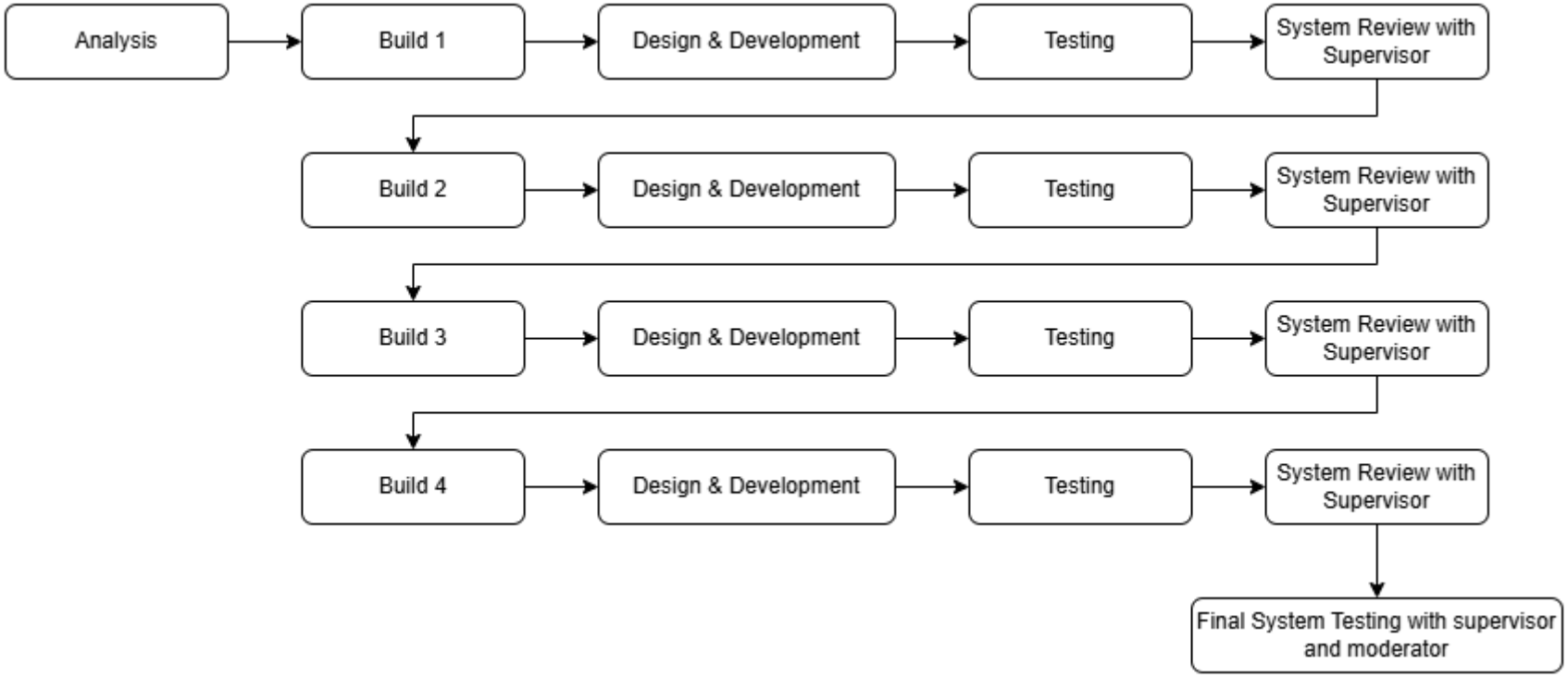


Figure 2.3 The Incremental Model

1.5 Project Team and Organization

Modules	Ong Yi Xin	Sia Keng Loon
Subject	✅	
Class Schedule	✅	
Payment	✅	
Other Service	✅	
Chat	✅	
User & Admin		✅
Announcement		✅
Progress & Evaluation		✅
Homework		✅
Attendance		✅

Table 1.5.1 Table for Module Incharge

1.6 Chapter Summary and Evaluation

This chapter outlined the objectives, identified key problems, proposed solutions, and detailed the functional requirements and project scope for the EazeTuition system.

- The **objective** focuses on automating and improving the scheduling, communication and payment processes in tuition centers by introducing AI-driven scheduling, centralized chat, and integrated payment and service modules.
- The **problems** highlight inefficient manual scheduling, fragmented communication channels, and cumbersome and error-prone manual payment processing for the current tuition center.
- The **advantages** describe how the proposed system reduces scheduling conflicts and wait times, improves parental engagement through secure chat, and streamlines fee processing to increase transparency and reduce staff workload.
- The **project scope** was broken down into functional features for five key modules: subject management, course scheduling (including AI-based functionality), chat, payments, and other service management. Each feature has clear functional requirements.

This chapter lays a comprehensive foundation for the system by linking the identified problems with the proposed solutions and translating them into detailed, actionable functional requirements. The clear division of modules and assignment of responsibilities lays the foundation for smooth execution and progress tracking of the project in subsequent phases.

Chapter 2

Literature Review

2 Literature Review

This chapter provides an **overview** of the basics of the EazeTuition project, including its **background, target users, and selected development technologies**. It provides a literature review covering **key components** such as programming languages, development tools, frameworks, databases, artificial intelligence integration and user interface design platforms. The chapter also assesses the overall feasibility of the project from a technical, economic, and operational perspective to ensure that the proposed system is practical, sustainable, and tailored to real-world needs.

2.1 Project Background

2.1.1 Target User

Target Market 1: Tuition Center Administrators & Staff

Tuition Center administrators and staff are the primary users of the app, and they rely on it to efficiently **manage their day-to-day operations**.

1. **Demographic Analysis**
    - **Ages**

      Administrators and staff of tuition centers in Malaysia are usually between the ages of **25 and 50 years old,** with a majority being female. This gender distribution aligns with broader employment trends in the education and childcare sectors, where females typically make up the majority of the workforce, particularly in teaching and administrative roles at early childhood and primary levels *(DOSM)*.

    - **Education Background**

      The majority of employees have **at least a diploma or bachelor's degree**, usually in fields such as e**ducation, business administration or early childhood management**. This educational background equips them with the necessary skills to effectively manage academic and administrative functions.

    - **Lifestyle**

      These employees usually work **full-time** in tuition centers, which range in size **from small local institutions** serving 20 to 50 students to **medium-sized**

centers that can accommodate more than 100 students. These tuition centers are mainly located in urban and semi-urban areas such as **Kuala Lumpur, Penang and Johor Bahru**. The demand for remedial education in these areas remains high due to the competitive academic environment and the importance parents place on remedial education.

Due to the need for employees to juggle **multiple responsibilities**, working hours are usually extended to six days a week, including weekends. In smaller tutoring centers, managers often take on multiple responsibilities - from **attendance tracking, scheduling coordination, billing, responding to parent inquiries to maintaining student records**. These centers often lack a dedicated IT or finance department, and therefore require employees with a wide range of skills and practical abilities.

2. **Psychographic Analysis**

    - **Work Priorities and Professional Values**

      Tutoring center administrators and staff are usually **pragmatic and task-oriented**, prioritizing the smooth running of day-to-day operations and customer satisfaction. They place a **high value on accuracy, organization and professionalism**, especially when interacting with paying parents. Their work environments often involve high levels of multitasking stress, especially during peak times, such as before and after school.

    - **Openness to Simple and Familiar Technologies**

      While they may **not be tech-savvy in a professional sense**, these employees are increasingly open to user-friendly digital tools-especially when these tools show clear benefits and require minimal training. For example, they are comfortable using WhatsApp to communicate with parents, Excel to calculate fees, and Google Calendar or whiteboards for scheduling.

    - **Core Motivations and Needs**

        - Reduce errors in fee collection (e.g., avoiding undercharges or missed payments).
        - Minimize scheduling conflicts and last-minute rescheduling.

- Keep parents informed and avoid answering the same questions over and over again.
- Save time on repetitive tasks (e.g., issuing receipts, managing spreadsheets).

- **Decision Drivers When Adopting New Tools**

  They prefer systems that are **reliable and intuitive**, and often worry that **overly complex digital platforms** may **slow them down** rather than help. Trust, time-savings, and simplicity are the three pillars that lead them to embrace new solutions. This preference is supported by findings from the Center for Digital Education, which reports that 76% of administrators and educators are actively seeking to innovate through the use of technology for educational management *("The Impact of Digital Technology on Education in 25 Stats")*.

3. **Behavior Analysis**

- **These users show a strong pattern of behavior in terms of routine tasks, usually repeating the same management cycle every month:**
  - Prepare fee invoices around the 25th-30th of each month.
  - Tracking student attendance on a daily basis.
  - Responding to parent inquiries, especially regarding course changes, payments, or make-ups.
  - Adjust schedules weekly, especially when counselors request substitutions or departures.
- **Many schools still rely on manual methods such as**
  - Handwriting receipts.
  - Collecting fees using cash or bank transfer screenshots.
  - Using WhatsApp group broadcasts to notify or inform parents of class changes.
- **These manual methods are prone to errors, such as**
  - Misplacing payment vouchers.

- Forgetting to mark a payment on a spreadsheet.
- Failing to notify all parents of class schedule changes.

Under pressure, especially during busy class transitions, staff may s**kip steps or make mistakes** that can lead to parent complaints or disruptions in the classroom.

Staff also tend to be **reactive rather than proactive** - responding to problems after they have occurred such as realizing that a student hasn't paid after several weeks have passed. An automated system that displays the status of payments and schedules in real time would help reduce the reliance on memorization and manual tracking.

- **They are more likely to adopt the system if**
  - The system would significantly improve the speed and accuracy of their daily work.
  - The system replaces existing tools they are already using (such as WhatsApp or Excel).
  - The system provides visual dashboards or summaries for a quick overview of the

Target Market 2: Parents

1. **Demographic Analysis**
   - **Ages**

     Parents who send their children to tuition centers in Malaysia are usually between **30 and 50 years old**. Parents of this age group have children in primary and secondary school, the two key stages where tuition is most needed to improve academic performance.

   - **Education Background**

     The majority of parents have at least secondary or tertiary education, with many of them holding a diploma or bachelor's degree. Their educational background enables them to value and understand the importance of academic support and structured learning. Many parents understand the

competitive academic culture in Malaysia and actively seek additional educational resources for their children.

- **Lifestyle**

  These parents typically **work full-time** in the education, healthcare, finance, civil service or private sector. They usually live in urban or suburban areas such as **Kuala Lumpur, Penang, Selangor or Johor Bahru**, where the demand for tuition services is high due to the highly competitive academic environment.

  They are busy with work, household chores and children's education on a daily basis. Most of them have **limited time** to visit tuition centers in person, and thus prefer online or mobile-first solutions for administrative tasks such as enrollment and payment. Parents also value convenience and prefer digital reminders to help them manage their children's schedules and assessments.

2. **Psychographic Analysis**

- **Parental Values and Academic Concerns**

  Parents prioritize their children's academic success, especially in subjects such as math, science, and languages, which often require remediation. They valued the accountability, transparency and regular updates of the tuition centers. Even though they do not always visit the tuition centers in person, most parents are highly attentive to their children's progress, tuition fees, and schedule updates.

- **Attitude towards technology**

  While parents are not always tech-savvy, they are generally comfortable using digital platforms. Most parents already use apps such as online banking, e-wallets (e.g. Touch 'n Go), WhatsApp and school portals. They are more likely to adopt if the platform is intuitive, accessible via mobile and does not require complex navigation.

- **Core motivations and needs**

  - Avoid time-consuming in-person enrollment and payments
  - Easily view and manage payment history and receipts

- Stay up-to-date with course schedules and assessment results
- Communicate directly with staff without searching for contacts
- Get automatic reminders to avoid missing payments or classes

- Decision Drivers When Using Apps
  - Parents are most likely to use an app if it offers the following features:
    - Real-time updates on fees, schedules and assessments
    - Digital payment options and downloadable receipts
    - Direct chat functionality to contact staff instantly
    - A centralized dashboard that displays all of your child's tuition information in one place
    - Automatic push notifications to avoid forgetting key dates

3. **Behavior Analysis**

- **Usage Patterns**
  - Parents interact with the system primarily during critical periods:
    - Enrollment period (start of semester or program)
    - Monthly billing cycle (usually 1st to 10th of the month)
    - Before exams or assessments
    - When courses are changed or canceled
    - The features they frequently access include
    - Online Payment Module
    - Course reminders
    - Attendance and progress tracking
    - Messaging with staff
- **Communication preferences**

Parents are more responsive to push notifications within the app than lengthy or cluttered WhatsApp group messages. A centralized app ensures that messages are not lost and can be accessed at any time without having to dig through chat logs.

- **Pain points of current practices**
  - Uncertainty or delays in receiving payment confirmations
  - Forgetting class dates or assessments due to lack of reminders
  - Inconsistent or unclear communication from the tuition center
  - Difficulty tracking past payments or retrieving lost receipts
- **Likelihood of adopting the system**
  - They are more likely to use the system if:
    - The system simplifies tasks they already do manually (e.g., payments, checking schedules)
    - Provides mobile-first convenience that fits their busy lifestyles
    - Provides quick access to all information related to their child's tuition costs
    - Reduces reliance on WhatsApp or paper receipts and provides structured, reliable records

Development Tools and Software Used

Program Language And Framework

- **Python**

  In this project, Python is used to build the **backend server logic** that handles key operations such as student enrollment, fee tracking, schedule coordination, and data processing. It manages API endpoints to support mobile client requests and connects to a MySQL database to perform CRUD operations. In addition, Python is used to integrate machine learning capabilities such as schedule automation and recommendation logic.

- **React Native**

  React Native has been chosen as a cross-platform framework for building Windows desktop applications and Android mobile applications from a single codebase.

  In this project, we use React Native to

  - Develop a parent interface for registering for classes and making payments
  - Build an administrative dashboard for managing class schedules and checking the status of fees
  - Implement real-time push notifications for reminders and alerts
  - Design a chat interface for direct communication between staff and parents
  - While iOS wasn't an initial priority, React Native provided us with the option of expanding it later, with minimal tweaking.

Development Tool

- **VS Studio Code**
  - Visual Studio Code was our main development environment for both frontend and backend development.
  - It allowed us to write Python code for the backend and JavaScript/TypeScript for the frontend within the same editor.
  - We also used VS Code extensions (e.g., Python, ESLint, Prettier) to improve code quality and debugging efficiency.
- **Expo Go**
  - We used Expo Go to quickly preview the React Native mobile app on real Android devices.
  - Instead of building APKs every time we made changes, we scanned a QR code to instantly test layout adjustments, navigation logic, and input validation.
  - This saved us hours during the iterative design and testing phases.

Database

- **MySQL**

  MySQL served as the main database for storing and managing all the core data in the system, such as:

  - Student and parent profiles
  - Class and subject details
  - Staff schedules
  - Fee payment history

  The structured schema allowed our team to enforce data relationships and run reliable queries for generating reports, checking payment status, and retrieving student attendance records.

- **XAMPP + phpMyAdmin**

  During the development and testing phases, we used XAMPP to host the MySQL database locally. phpMyAdmin was heavily used by two team members to

  - Quickly create and modify tables
  - Insert sample data for testing
  - Debug queries and monitor changes
  - Export backups before each test
  - This setup allows us to collaborate efficiently without in-depth knowledge of command-line SQL.

Version Control

- **Git + GitHub**

  With only two people on the development team, Git is critical to avoiding code conflicts.

  We use Git to

  - Create branches for each feature (e.g. payment system, chat module)
  - Merge changes after testing

- Maintain a clean commit history

GitHub is used for

- hosting project repositories
- tracking issues (e.g. bugs and feature requests)
- asynchronous collaboration by pulling requests and comments
- Even though we're a small team, this workflow has helped us stay organized and accountable.

AI / ML

- **Scikit-Learn**

  Scikit-Learn is a powerful **machine learning library** in Python used to build and integrate AI models into the project. It supports tasks such as classification, clustering, and prediction, and is used here to enhance features like automated scheduling or user behavior analysis.

Design Tool

- **Figma**

  Figma was used at the beginning of the project to design the application interface and user flow.

  We created

  - User interface models for parent and admin users
  - Clickable prototypes to simulate navigation before coding
  - A shared design system with uniform colors, fonts, and layout rules
  - Both team members contributed to the Figma file, which served as a blueprint for front-end development using React Native. It also helps us gather early feedback from supervisors or peers before writing actual code.

2.2 Literature Review

2.2.1 Programming Languages, Development Tools and IDE Used

Python

Python is a high-level, interpreted, general-purpose language known for its readability, simple syntax, and large ecosystem of libraries (Sanner, 1999). For example, Flask, Django for the web; Scikit-Learn, NumP for AI/ML.

Artificial Intelligence/ML Integration

Python is the leading language for machine learning and data science because of its rich ecosystem of libraries such as Scikit-learn, TensorFlow, and Pandas. These tools allow developers to build predictive models, recommender systems, and classification algorithms with minimal overhead (Walia).

EazeTuition leverages Scikit-learn to analyze user behavior such as preferred class times or attendance patterns and supports automated scheduling recommendations. This helps administrators optimize class time assignments based on historical preferences and trends, enhancing personalization and reducing manual errors.

Integration with Databases and IoT

Python has strong support for MySQL (via mysql-connector-python, SQLAlchemy), SQLite, Firebase, and real-time protocols such as MQTT. This makes it ideal for educational platforms that may extend to hardware integration such as sensors or smart displays.

EazeTuition's backend is built using Python Flask and is connected to a MySQL database for managing student profiles, course schedules, and payment records. It also supports MQTT-based communication for future extensions such as a smart attendance system or indoor environmental sensors such as room temperature, brightness, etc.

Cross-Platform Deployment

Python applications can run on Windows, macOS, and Linux with only minor modifications (Sanner, 1999). This flexibility enables teams to develop and deploy consistently across devices, servers, and operating systems.

The system's Flask backend and supporting scripts such as PDF receipt generation, report processing. allows the EazeTuition application to be hosted on Windows during development

and can be deployed to cloud Linux servers as it scales. This ensures compatibility with the machines used by the team.

VS Code Studio

Visual Studio Code (VS Code) is a lightweight open source code editor developed by Microsoft. With support for multiple programming languages and frameworks, including JavaScript, Python, Dart, and TypeScript, it is a versatile tool for full-stack and cross-platform application development (Hadley). With support for extensions, Git integration, real-time debugging, and a rich and intelligent hinting engine, VS Code has become the development environment of choice for modern software teams.

Cross-platform and full-stack projects

VS Code supports multiple programming languages and frameworks such as JavaScript, Python, Dart, HTML/CSS, SQL, etc. through its rich extended marketplace. Therefore, it is ideal for cross-platform development using both front-end and back-end in the same environment.

The EazeTuition development team uses VS Code to manage mobile front-end (React Native) and back-end APIs (Python Flask) in a unified workspace.

Integrated Git and CI/CD Workflows

VS Code has built-in Git support that enables developers to manage source code control directly from the editor (Hadley). This makes it ideal for collaborative workflows involving GitHub or GitLab, especially version control, issue tracking, and deployment integration.

The EazeTuition development team uses Git integration to efficiently manage version control and pull requests without leaving the IDE.

Lightweight IDE for Small Teams or Academic Projects

Unlike heavier IDEs such as Visual Studio or Android Studio, VS Code is lightweight, fast loading, and takes up fewer resources, making it ideal for small development teams or students working on academic software projects.

On the EazeTuition development team, VS Code ensures rapid iteration, minimizes setup overhead, and makes it easy to share files via GitHub.

Expo Go

Expo Go is a mobile application powered by the Expo framework that allows developers to preview and test React Native apps on real devices without having to build or install APK or IPA files. It works by scanning a QR code from a development server (Expo CLI) and then rendering the application on-the-fly through a live development environment.Available for both Android and iOS platforms, Expo Go greatly simplifies the testing process during mobile application development.

Rapid mobile testing without local builds

Expo Go is ideal for projects such as EazeTuition, where developers need to quickly test UI changes, logic flows, or new features without having to compile the APK each time (Farid). Since EazeTuition targets mobile-first users such as parents and students, the development team needs to frequently preview real-time functionality of the Expo Go allowing instant updates and real-time reloads directly on Android phones by scanning QR codes.

Small Teams and Resource-Limited Development

With only two developers on the EazeTuition team, using Expo Go avoids the complexity and time required to use native build tools like Android Studio or Xcode. This allows the team to focus on the logic and usability of the application rather than spending time managing build configurations.

Ensure cross-platform UI consistency

EazeTuition is designed for Android phones and Windows desktops. Expo Go allows testing on different Android devices to ensure UI consistency across screen sizes and operating system versions without having to use multiple emulators or devices.

2.2.2 Framework

React Native

React Native is an **open source cross-platform framework** developed by Meta that allows developers to build native-like mobile and desktop applications using JavaScript or TypeScript. It supports Android, iOS, and even Windows systems using a single codebase (Eisenman) through extensions such as React Native for Windows. Its modern architectures, such as Fabric and TurboModules, improve runtime performance by reducing the communication overhead between JavaScript and native modules.

Cross-platform development for small teams

React Native allows developers to build Android, iOS, and Windows apps using a single code base, reducing development and maintenance costs. This is especially beneficial for small teams with limited manpower. With only two developers on the EazeTuition team, using React Native to provide a unified app experience for both Windows and Android platforms was a favorable choice. By maintaining only one code base, the team can efficiently develop and update features across both platforms without duplicating work.

Familiarity with the JavaScript/React ecosystem

React Native is based on JavaScript and ReactJS, two technologies well-known in web development, making it easier for teams familiar with web development to transition to mobile and desktop app development (Eisenman). EazeTuition developers already had experience with JavaScript and React, so they were able to quickly adapt to React Native without having to start from scratch. This not only speeded up development, but also significantly reduced the learning curve.

Integrate with existing backends (Python, .NET, etc.)

React Native works seamlessly with REST and GraphQL APIs built on backend technologies such as Python (Flask, FastAPI) or .NET (ASP.NET Core) for full-stack flexibility (eSparkBiz). The system backend is built using Python Flask, and React Native communicates with it via REST APIs to obtain user data, course schedules, and payment records. This enables real-time updates and centralized data management.

2.2.3 Database

MySQL (via XAMPP + phpMyAdmin)

MySQL is an open source relational database management system (RDBMS) that uses structured schemas and SQL for data management. XAMPP provides a local server environment, while phpMyAdmin provides a GUI to easily manage the database.

Structured and transactional data

MySQL supports the definition of strict foreign key relationships, ensuring that enrollment records do not reference non-existent users or courses. This relational integrity ensures data consistency (Samonte & Samonte, n.d.). MySQL's relational model works well when applications need to manage interrelated entities such as parents, students, courses, registrations, scheduling, and payments.

EazeTuition leverages MySQL's relational structure to ensure consistent data relationships and eliminate duplication and errors, which is critical for generating accurate course lists, expense reports, and attendance records. For example, the Payments table is linked to the Parents table and the Invoices table to track who paid and what was paid.

Transactional Integrity and ACID Compliance

MySQL's InnoDB engine supports full ACID transactions, ensuring atomicity, consistency, isolation, and durability (Samonte & Samonte, nd). For example, if a parent's payment needs to be recorded along with a receipt entry, the two operations can occur simultaneously, thus avoiding mismatched or incomplete data records. EazeTuition uses ACID-compliant transactions to maintain reliable financial records and avoid disputes or reconciliation errors.

Stable schema that can be modified occasionally

MySQL is well suited for applications with highly stable, predefined data schemas whose table structures and relationships remain consistent over time. While it supports schema modifications through tools such as ALTER TABLE, it performs best when structural changes are rare and planned.

EazeTuition takes advantage of this feature by adopting a fixed-schema design for core modules such as students, courses, payments, and tutors. These structures are unlikely to change frequently, ensuring long-term stability and maintainability. When adjustments are needed, such as adding new fee types or updating course properties, they can be applied through phpMyAdmin in XAMPP with minimal disruption. This controlled flexibility not only supports system reliability, but also leaves room for occasional feature updates.

2.2.4 AI / ML

Random Forest Regression

Random Forest Regression is an ensemble learning technique that generates more accurate and stable predictions by constructing multiple decision trees and aggregating their outputs. Each decision tree learns from distinct subsets of features and samples, enabling the model to capture complex nonlinear patterns within structured datasets. This technique is widely applied in educational analytics due to its robustness, noise resistance and ability to model multifactorial influences such as student habits, learning load, lifestyle indicators and historical performance.

Predictive Modeling for Lifestyle and Stress Estimation

Within the EazeTuition system, the Random Forest regression model can predict student health scores based on features such as study duration, sleep duration, extracurricular workload and activity levels. This model converts these continuous lifestyle attributes into a health index, thereby reflecting underlying stress patterns.

Random forest proves particularly suitable in this scenario due to its ability to effectively capture nonlinear relationships such as how excessive study time lowers the health index, but only when both sleep and activity levels fall below specific thresholds. This enables the system to build a refined, data-driven lifestyle scoring engine for timetable evaluation.

Performance Forecasting with Multi-Factor Inputs

The system also employs a random forest algorithm to predict academic performance by analyzing variables such as attendance rates, past exam scores, parental involvement, learning motivation, physical activity and socioeconomic indicators.

Random forests can simultaneously process mixed numerical and categorical data, making them an ideal choice for modeling real-world educational datasets, where factors like parental education levels or internet access must be encoded and interpreted alongside quantitative metrics. Predicted exam scores once standardized are integrated into EazeTuition's hybrid decision model to support personalized schedule recommendations.

Timetable Feature Evaluation via Structural Metrics

Beyond lifestyle and performance predictions, EazeTuition employs random forest principles to construct a heuristic timetable evaluation model. The system converts raw timetables into structured features through a feature extraction module which includes break duration, class density, classroom and changes.

These features are analyzed through a random forest scoring mechanism to evaluate schedule quality such as penalizing overly dense schedules, frequent classroom changes or excessively long breaks. This approach enables the system to provide human-like judgments of schedule comfort while maintaining deterministic output, meeting administrative decision-making needs.

Integration with Python Data Pipelines and Backend Systems

The random forest model in EazeTuition is implemented using Scikit-learn and seamlessly integrates with Pandas for preprocessing, NumPy for numerical computation and Flask for backend deployment.

The training dataset, stored in CSV format, undergoes processing and standardization before being converted into an encoded feature matrix. This matrix is then used for training and inference within the Random Forest model. Final prediction results are transmitted via API interfaces to the HTML frontend, enabling real-time evaluation of class schedule proposals. This integrated workflow ensures reliable machine learning execution within production-ready educational management systems.

2.2.5 UI Design Tool

Figma

Figma is a cloud-based interface design and prototyping tool for designing user interfaces, wireframes and collaborative prototypes. Unlike traditional design software that requires installation, Figma runs entirely in the browser, allowing real-time collaboration between team members, much like the Google Docs of the design world. It supports vector editing, reusable components, design systems, and prototyping capabilities in a unified platform.

Collaborative Design in Small Teams

Figma is effective for small teams because it allows multiple users to work on the same design file in real time. Its cloud-based environment reduces version control issues by ensuring that all collaborators see updates instantly.

The EazeTuition development team uses Figma to allow two team members to simultaneously co-design key interfaces such as dashboards, registration forms, and course schedule screens. The team uses annotations and live editing to communicate ideas and iterate quickly without the need for additional tools.

Rapid Prototyping and User Interface Testing

Figma's prototyping tools enable developers to create interactive, clickable user interface flows without writing code. This helps stakeholders visualize the structure and interaction of the application before development begins.

The EazeTuition team used Figma to build interactive models of key user processes, such as topic registration, payment tracking, and messaging. This prototype provides a clear referee for final implementations in React Native.

Cross-platform and Remote-Friendly Workflow

As a browser-based tool, Figma is platform-agnostic and works seamlessly across devices without complicated setup or installation.

Although the EazeTuition team only uses Windows devices, Figma's cloud-based environment still plays a critical role in enabling remote collaboration. Team members access shared design files via a browser during virtual meetings, enabling real-time editing, seamless feedback exchanges, and more efficient workflows, even when working in different locations.

2.3 Feasibility Study

2.3.1 Technical Feasibility

To effectively serve students, parents and tuition center administrators accessing the digital platform primarily through **Android devices and Windows laptops**, the EazeTuition app was designed with cross-platform consistency, real-time responsiveness and data-driven decision support. **React Native** was required to be used as the **primary development framework** to enable a single codebase to support both Android (for mobile-first parents and students) and Windows (for desktop-based administrators), reducing development overhead while maintaining feature parity. **Expo Go** is required to further enhance this workflow by allowing real-time mobile preview and testing without manual APK builds, significantly accelerating the testing process for mobile devices.

The **backend** is required to be built with **Python Flask**, which provides fast, lightweight API services to ensure responsive user interactions for time-sensitive features such as attendance tracking and payment updates. For **intelligent learning analytics**, EazeTuition requires **Scikit-learn's Random Forest Regression** to **estimate lifestyle wellness** and **predict academic performance** based on factors such as study hours, sleep duration and stress level. These predictive outcomes are then **integrated** into **AI-generated class scheduling proposals**, enabling administrators to review data-driven recommendations for optimizing schedule quality and supporting more personalized timetable planning.

Critical business data, including registration, payment, and course schedules, is **securely stored** in a **MySQL database**, which was chosen for its ACID compliance and ability to ensure relational integrity. During the design phase, **Figma** was required to be used to **prototype** a user-friendly interface that ensures usability for different user groups, especially those who are less tech-savvy. To streamline development, the team requires **Visual Studio Code** as the **IDE** and **Git and GitHub** for **agile version control and collaboration**. This architecture ensures that EazeTuition is scalable, maintainable, and able to meet the real-world needs of a modern teaching center.

2.3.2 Economical Feasibility

EazeTuition is **economically** viable because it relies on **open source and free technologies** throughout its development lifecycle. By leveraging **Python** for back-end development and **React Native** for front-end development, the system **eliminates licensing costs** while enabling scalable and cross-platform functionality across Android and Windows environments.

The **framework** also **benefits** from **Python's extensive ecosystem,** including Flask for API integration and Scikit-learn for machine learning logic, both of which are powerful community-supported tools that require no capital investment.

Auxiliary development tools such as **Visual Studio Code, XAMPP, phpMyAdmin, Figma** which is a **free program** and **Expo Go** for testing and previewing React Native apps on mobile devices provide basic IDE, database, and UI/UX functionality without adding to the budgetary burden and are suitable for a small academic team.

No additional hardware expenditure is required as development is done using the team's **existing laptops and mobile devices**. Initial hosting can be handled through local servers or free-tier cloud platforms, thereby deferring infrastructure costs until growth requires expansion.

Additionally, the application integrates automation for payment processing, scheduling, and reporting, thereby reducing manual effort and operational overhead. This strategic use of free and scalable technology allows EazeTuition to be developed and deployed with minimal upfront costs, perfectly fitting the constraints of academic programs and resource-focused startups.

2.3.3 Operational Feasibility

EazeTuition directly meets the practical day-to-day needs of tuition center staff and parents and is highly operationally viable. The system **runs on multiple platforms**, ensuring easy access for users no matter which device they use. **Parents** can easily access the application on their **Android smartphones** to manage the schedule and receive updates, while **administrative staff** can operate the system during working hours on their **Windows desktops** or laptops.

The **user interface** is optimized for both **mobile and desktop environments**, making critical tasks such as real-time attendance checking, schedule management and payment monitoring simple and intuitive. The platform is **browser accessible** and compatible with widely used options such as **Chrome**, eliminating the need for complex installations and allowing users to log in from any Internet-enabled device.

Deployment is also cost-effective; EazeTuition can run on **free web hosting services or local networks**, making it particularly suitable for small to **medium-sized teaching centers** that may lack dedicated IT infrastructure. This flexibility ensures readiness to go live without incurring high installation costs.

Real-time updates support business-critical workflows, enabling staff to **instantly view** and **manage class changes, student attendance and payment status**. Integrated notifications and reminders help minimize communication delays, prevent missed classes or fees, and streamline coordination between staff and parents.

In summary, EazeTuition demonstrates clear operational feasibility through its user-friendly access, cost-effective deployment, and seamless support for daily tuition center processes.

2.3.4 Schedule Feasibility

The development of the EazeTuition application follows a structured academic timetable, ensuring that each key milestone is completed within the allocated time. The timeline was planned according to the University's Final Year Project (FYP) phase, providing ample time for development, testing, documentation and refinement.

The **proposal phase** was completed on **April 18, 2025** and included research, scope definition, and project approval. By **August 9, 2025**, the **Project I portfolio** will be completed, ensuring that all system planning, requirements analysis, and design methodologies are documented and approved.

An **initial system preview** is scheduled for **September 26, 2025** to allow for early evaluation of core functionality and user interface design. This will be followed by a **system preview** on **November 21, 2025**, at which time in-depth testing will be conducted to ensure functional stability.

Final system testing is scheduled for **November 28, 2025** and will focus on verifying the overall performance and reliability of the application.The **draft FYP report** will be submitted on **December 12, 2025** and the **final report** is scheduled for **December 19, 2025** to mark the formal completion of the project.

This schedule is realistic and fully aligned with the development objectives and academic results, confirming the feasibility of the EazeTuition project schedule.

2.4 Chapter Summary and Evaluation

This chapter provides an in-depth look at the technologies, tools, and feasibility considerations that support EazeTuition application development. The chapter began by identifying the **primary target users**, i.e., **parents and tutorial center administrators**, and gaining an understanding of their needs, technology habits, and operational workflows. Guided by this in-depth understanding, we selected appropriate technologies for backend and frontend development.

In the **literature review**, we have reviewed the **programming languages** (Python for the backend and JavaScript for the frontend via React Native), **development tools** (Visual Studio Code, Expo Go), **frameworks** (React Native), **database systems** (via XAMPP/phpMyAdmin using MySQL), **machine learning tools** (Scikit-learn), and **design tools** (Figma) were evaluated and justified. Each was selected based on its suitability for small academic teams, ease of integration, scalability, and ability to meet project goals such as real-time communication, automated scheduling, and intuitive user interface/user experience.

The **feasibility study** covered the **technical, economic, operational and schedule** aspects of the system. Technically, EazeTuition is feasible with cross-platform tool compatibility and a scalable backend infrastructure. **Economically**, the project relies on **open source and free tier** technologies and requires minimal capital investment, and thus remains viable. **Operationally**, the system can be deployed on **multiple platforms** (Android and Windows) and can be accessed through **browser-based** administration or mobile applications. In **terms of timing**, the development timeline is **organized and realistic**, with clear milestones to ensure steady progress from proposal to final report.

In summary, the assessment of tools and feasibility confirmed that the EazeTuition system is not only technically reliable and cost-effective, but also meets the actual operational needs of a modern tuition center. The chosen technology strikes a balance between functionality, usability and maintainability, providing a solid foundation for a successful implementation in the later development phases.

Chapter 3

Methodology and Requirements Analysis

3 Methodology and Requirements Analysis

This chapter describes the development methodology, requirements analysis, and development environment for the EazeTuition system. The chapter begins with an overview of the chosen software development model, followed by a description of the techniques used to gather system requirements. Functional and non-functional requirements are then defined, along with detailed use case diagrams and descriptions for each module. The chapter concludes with an overview of the software and hardware environments used in the development process.

3.1 Methodology

The EazeTuition system was developed using an incremental development model, which is particularly appropriate given the **scope, schedule, and team size of the project**. Rather than building the entire system at once, this approach develops and delivers the system through multiple iterations, with new features and improvements added with each release (Sarker et al.).

3.1.1 Reason

Small Team Considerations

One of the main reasons for choosing an incremental development model was limited development resources.With only two developers on the EazeTuition project team, attempting to build the entire system in a single phase would have entailed significant risks, such as the system being incomplete or unusable by the project deadline. By breaking development into smaller, manageable increments, we were able to deliver functional modules incrementally, ensuring steady progress while reducing complexity.

Time constraints and practicality

Another important factor was the tight development timeline. With only 5 to 6 months available to complete the system, a phased development model enabled early delivery of the core system functionalities. This allowed essential modules, such as course scheduling, registration, and student management to be implemented and tested early. As a result, feedback from stakeholders could be gathered in time to guide improvements in later stages.

Version-Based Delivery Plan

The EazeTuition system utilizes a version-based deployment strategy where the application is released in multiple phases. Each release adds new features based on the project roadmap and stakeholder feedback. The first release serves as the minimum viable product (MVP),

providing basic functionality such as user management, scheduling and registration. Later releases will include more advanced features such as online payments, transportation and childcare modules, and reporting tools. This structured versioning helps manage expectations, supports incremental enhancements, and ensures a stable foundation for future updates.

Core-first development focus

Adopting a core-first development strategy ensures that the most critical functionality is built and tested early in the process. This approach maximizes value in a shorter timeframe and reduces project risk by validating the most important features first. Deliver core modules early. For example, by enabling subject enrollment before payment functionality is in place, the system can begin to deliver real value while remaining open for iterative improvements. This also facilitates early user feedback to inform better design in future development phases.

In summary, the incremental development model provides a practical and adaptive framework for EazeTuition. It enables continuous improvement, manageable task allocation, and timely delivery of core functionality, all of which are critical for small development teams operating within limited time and labor constraints.

3.1.2 Stages

3.1.2.1 Analysis Phase

During the initial phase of the EazeTuition development process, the focus is on a thorough analysis to establish a clear understanding of the project scope, objectives, user needs, system requirements and technical feasibility. This phase lays the foundation for all subsequent incremental development cycles.

1. **Project proposal and scope validation**

   The first step was to prepare and submit a formal project proposal to the project manager. The proposal outlines the initial vision of EazeTuition, including the main problem it aims to solve - manual and inefficient tuition fee management - as well as the core modules to be developed: subject management, course scheduling, payment system, management of other services (transportation and childcare), and chat communication.

   The proposal serves as a reference to confirm that the proposed scope of the project is realistic and achievable within the development timeframe of 5 to 6 months. After review and feedback from the supervisor, the project scope was refined and finalized.

2. **Identification of objectives, issues and contributions**

Once the scope was approved, the next task was to identify the key objectives of the project. These objectives included streamlining class and subject scheduling, digitizing the registration and payment process, and providing additional services such as transportation and childcare on a centralized platform.

The development team identified key societal issues with current tutoring center workflows, such as heavy reliance on manual processes, lack of centralized scheduling, and inefficient communication, and positioned EazeTuition as a solution that could provide tangible improvements. A contribution summary was prepared detailing how each module would facilitate users and improve operational efficiency.

3. Analysis of Target Users and Project Background

To ensure that the system meets the actual needs of the users, the target users were analyzed. The main users identified were parents, tuition center staff and administrators. Parents needed an intuitive interface to enroll their children and pay fees, while administrators and staff needed tools for scheduling, managing student data, and communicating with parents.

This phase also included identifying the functional and non-functional requirements of the system. Functional requirements include features such as subject lists, class schedule creation, payment tracking, and internal messaging. Non-functional requirements focused on usability, responsiveness (mobile support) and system reliability. The technologies used have also been identified. Python (Flask) for the back-end, React Native and JavaScript (jQuery, AJAX) for the front-end, MySQL for data storage, and others.

4. Research And Feasibility Study

The research phase was designed to understand the existing solutions and technologies associated with the system. This included a literature review of current tuition center platforms to identify gaps that EazeTuition could fill. A feasibility study was then conducted to evaluate the project in terms of technical implementation, time constraints, and available resources.

Since the system will be deployed online, the hardware requirements are minimal. However, Application Programming Interfaces (APIs) were

examined to evaluate possible integrations such as chat and payment gateways (e.g., Stripe) and et al.

5. **Project Planning and Milestone Design**

   A detailed project plan was devised, dividing the work into four incremental phases, each focusing on a specific set of modules. To visualize the system architecture, module hierarchical diagrams were created to categorize functionality into core and extended components. Milestones and deliverables for each increment were clearly defined to ensure time management and development tracking. Gantt charts and task breakdown sheets were also created for internal use.

6. **Methodology Selection and Requirements Gathering**

   Once the plan was in place, the team reviewed and finalized the incremental development model as the formal methodology. The methodology fit the team's constraints and allowed for flexible iteration and regular oversight.

   Requirements were gathered through informal interviews and observations of existing remedial center workflows. Key use cases were then developed to illustrate core user interactions such as registering for a subject, scheduling a class, making a payment, and chatting with the center.

   Several use case diagrams were created, along with use case descriptions detailing participant interactions, triggers, prerequisites, and expected outcomes for all key operations.

7. **Supervisor Review and Sign-off**

   At the end of the analysis phase, all deliverables, including finalized requirements, project plan, module diagrams, and research results - are submitted to the supervisor for review. Feedback is incorporated and, upon approval, the team moves into the first incremental design and development phase.

3.1.2.2 Design Phase

After the analysis phase, the EazeTuition development team moved into the design phase. This phase focuses on translating the requirements into a concrete system design, ensuring that all components, both functional and structural, are carefully planned before actual development begins. The design phase is critical to integrating

user requirements with the technical solution and ensuring a smooth implementation at a later stage.

1. **System Design**

   The design process began with defining the overall structure and workflow of the EazeTuition system. Based on the previously approved module hierarchy and requirements, the team outlined how the different modules (e.g., subject management, course scheduling, payments, other services, and chat) would interact with each other.

   This included outlining the logical flow between functions and ensuring that the system would run smoothly in both web and mobile environments. We also paid particular attention to the modularity of the modules so that they could be extended in the future without major refactoring.

2. **User Interface (UI) Design**

   To ensure a friendly and intuitive user experience for all user groups, including parents, staff, and administrators, the development team designed a comprehensive user interface prototype using Figma.

   Each screen maps out expected user actions such as subject selection, schedule viewing, payment processing, and live chat. The goal was to achieve consistency across pages, reduce the learning curve for users, and accommodate responsive design principles for mobile devices. The user interface prototype was then used as a reference throughout the front-end development process.

3. **Entity Relationship Diagram (ERD) and Database Design**

   Once the system architecture and user interface were determined, the team designed the data model to support the system logic. Using draw.io, an Entity Relationship Diagram (ERD) was created to visualize all the necessary tables and their relationships.

   Key entities included students, subjects, classes, class scheduling, payments, service requests, and messages. Where appropriate, relationships such as one-to-many such as a subject linked to multiple classes and many-to-many such as a student enrolled in multiple subjects were carefully handled using linked tables.

XAMPP's MySQL was chosen as the database platform for local development and testing because of its simplicity and compatibility with Python backends.

4. **Report Design**

During this phase, the team also designed the report generation component, which will later be used to support administrative decision-making.

- Planned report features include:
- Unpaid Student Report to help staff track unpaid tuition
- Enrollment by Class to monitor class popularity and resource allocation
- Service Usage Summary to view the frequency of requests for childcare or transportation services

The structure of each report was pre-determined, with data sources and filtering options specified.

5. **Software Architecture Design**

A software architecture diagram was prepared to represent the technical structure and components of the EazeTuition system. The project was designed following the MVC (Model-View-Controller) pattern to promote modularity and maintainability.

The Model layer utilized SQLAlchemy, a Python SQL toolkit and Object Relational Mapper (ORM), to interact with a MySQL database running via XAMPP. This allowed for cleaner, more abstracted database queries, making the codebase easier to maintain and scale.

The View layer was divided based on user roles:

- Admin Interface: Developed as a web-based application using HTML, CSS, JavaScript (jQuery + AJAX). Admin users could manage subjects, schedules, payments, and other services through a browser-based dashboard.
- Mobile App Interface: Designed for parents and tutors, this interface was developed as a cross-platform mobile application using React Native, enabling users to view class schedules, make payments,

request transport or childcare services, and use the in-app chat feature.

The Controller logic was implemented using the Flask web framework in Python, handling routing, API endpoints, form processing, and interaction between the model and views.

6. **Design Review with Supervisor**

Before moving into the development phase, the entire design output—including UI prototypes, ERD, report structure, flowcharts, and architecture—was reviewed with the project supervisor. Feedback was received and applied, particularly regarding optimizing database relationships and refining the chat and payment workflows. Once approved, the team proceeded with implementing the first development increment.

3.1.2.3 Incremental 1: Subject, and Class Schedule Module

1. **Development Phase**

In the first increment, development focused on building the core academic and user management modules. This included:

- Subject Management Module: Allows the admin to create, edit, activate, or deactivate subjects.
- Class Schedule Module: Enables admins to create class sessions with associated subjects, tutors, and timeslots.

These modules were built using Python Flask for backend logic, SQLAlchemy for database interaction, and Jinja2 templates with HTML/CSS for admin views. The parent and tutor interactions were planned to be handled through the mobile app.

2. **Testing Phase**

Unit testing included:

- Verifying subject creation, status toggling, and subject-list filtering.
- Ensuring class schedules properly matched selected subjects, tutors, and available time slots.

Integration testing validated:

- Active subjects could be scheduled but deactivated ones were hidden.
- Admins could not create overlapping class times for the same tutor.

3. **System Review with Supervisors**

   A review was conducted with supervisors to assess the functionality and interface of the system. Based on feedback, improvements were made to the scheduling interface and validation rules for course slots were enhanced.

3.1.2.4 Incremental 2: Other Service and Payment Module

1. **Development Phase**

   This increment focuses on expanding the system to support additional tuition services and financial management:

   - Other Services Module: Allows administrators to assign transportation and childcare to students.
   - Payment Module: Administrators can generate invoices and record payment status; parents can view payment history.

   Designed to maintain consistency of service assignment and billing record data.

2. **Testing Phase**

   Unit tests covered:

   - Service assignment to individual students.
   - Accurate fee computation based on selected subjects and services.
   - Payment status updates (e.g., paid, unpaid) reflected correctly.

   Integration testing ensured that:

   - Assigned services appeared correctly in the invoice breakdown.
   - Only valid service combinations were accepted.
   - The payment history matched the student's registration and service data.

3. **System Review with Supervisors**

   During the review, improvements were recommended for invoice filtering and accessibility of service records. Improvements were made to the assignment interface to allow counselors to track assignments more clearly.

3.1.2.5 Incremental 4: Chat Module

1. **Development Phase**

   In the final increment, a chat module was developed to improve communication between parents and counselors:

- The module embeds a simple real-time messaging feature in the mobile application program interface.
- Chat messages are structured to store the sender, receiver, message body, and timestamp in a database.

The team considered integrating real-time solutions such as Firebase or Socket.IO in future enhancements, but due to time constraints, a polling-based version was initially implemented.

2. **Testing phase**

   Unit testing covered:

   - Send and retrieve messages based on user roles (counselor and parent).
   - Verify consistency of message history database.

   Integration testing confirms that

   - Only intended recipients have access to specific chat sessions.
   - Message sending order and timestamps are handled correctly.

3. **System review with supervisor**

   Supervisor reviewed information flow and emphasized improving mobile responsiveness and clarity of user interface. Suggested exploring push notification support as time permits.

3.2 Requirements Gathering Techniques

3.2.1 Observation

During the analysis phase of EazeTuition, the development team utilized direct observation as a requirements gathering technique. The observation focused on a Form 4 student (the developer's cousin) who attended the tutorial center from May to June 2025.

Subjects Enrolled and Service Availability

Enrolled Subjects and Service Provision

The main academic subjects that the student enrolled in were Additional Mathematics, Mathematics, English (Comprehension), English (Composition), Malay (Comprehension), Malay (Composition), History and Accounting.

During the observation, the developers noted that the tuition center did not offer certain non-core subjects such as Chinese, nor did it provide individualized learning pathways such as standard and focused classes.

This discovery influenced the design of the Subjects Module, which allows the administrator to define a wider range of subjects and categorize them by type such as Core, Elective, and Language, while the Schedule Module supports multiple course types per subject such as Standard Classes and Focused Classes .

Other Services for Different Age Groups

The center provides childcare for elementary students only, not middle school students. However, transportation services are available to students in both grades.

This coincides with the system's "Other Services" module, which is designed to differentiate service availability by age group, enabling administrators to configure eligible services accordingly.

Payment Process and Mode

Parents usually pay the fees in person at the center. For first time enrollees, they must ask for fee details at the counter and complete their registration on site. In subsequent years, Google forms were sent at the end of the academic year to collect subject and transportation options for the following academic year.

This highlighted inefficiencies in the fee payment and subject enrollment process, leading to the integration of the fee payment module and subject module with the self-service digital enrollment process and online fee detail display.

Chat and Communication

There is no dedicated chat system to streamline communication between parents and counselors or administrators. All communication, including absence reporting, is done through WhatsApp, which is informal and lacks record tracking.

Therefore, we designed EazeTuition's chat module to facilitate recordable communication between parents, staff, and counselors within the app to increase accountability and efficiency.

3.2.2 Online Research

To support the requirements gathering process for the EazeTuition system, a comparative analysis of three existing tuition-related systems was conducted: the CS Tuition application,

Tuition Management, and EazeTuition (an existing prototype or similar solution). This online study was designed to understand current trends, identify system gaps, and gain inspiration for functional planning related to the proposed modules.

Purpose of the Study

The purpose of the study is to explore how current systems enable core functions of tuition centers such as subject registration, course scheduling, payment, other services (transportation/childcare) and communication tools. This helps to identify the strengths and limitations of existing solutions, especially for tutorial center practices in Malaysia or Southeast Asia.

Key Findings and Comparison

Module	Cs Tuition Application	Tuition Management	EazeTuition
Subject	Yes, offering the main subject for different levels, but no additional subject.	Yes, offering the main subject for different levels, but no additional subject.	Yes, offering the main subject for different levels or some additional subjects.
Class	Yes, involving letting the parent to register a class for children, but no package to select, need to select one-by-one.	Yes, users select which online class to enroll in, but no package to select, need to select one-by-one.	Yes, provide multiple packages or self-select classes to enroll. Also include AI auto scheduling class.
Other Service	No, do not consider other services.	No, do not consider other services.	Yes, provide optional service such as transport and childcare service.
Payment	Yes, allow for various payment ways.	No, it does not include payment features, only for fees management.	Yes, provide various payment methods for paying fees.
Chat	Yes, one to one chat.	Yes, students tutor one to one chat.	Yes, parents, tutor one to one chatting.

3.2.2.1 Summary result collection

Based on a comparison of real-world observations and online research, the following key findings and requirements were identified for the EazeTuition system. These findings guided the design decisions for the system.

1. **Subject Management Flexibility**

   From observation, students often enroll in multiple core subjects, yet many centers do not offer optional subjects like Chinese or flexible class types. Online systems typically only allow static subject selection per level.

   - Requirement Identified: Enable flexible subject selection, including additional optional subjects and the ability to categorize classes as "Normal" or "Focus".

   - EazeTuition Implementation: The Subject module supports a wide range of subjects, including optional ones. Each subject can be associated with multiple class types to support different learning needs and intensity levels.

2. **Intelligent and Convenient Class Scheduling**

   Observation showed that replacement classes are handled manually via WhatsApp, causing inefficiencies. Online systems rarely offer scheduling intelligence or class grouping options.

   - Requirement Identified: Provide an automated or semi-automated scheduling system that simplifies registration and allows class replacement in a structured way.

   - EazeTuition Implementation: The Class Schedule module supports smart scheduling, including features for package-based enrollment and optional AI-driven timetable suggestions (if implemented). The module also supports leave-taking with proper replacement tracking.

3. **Incorporation of other services**

   Based on actual observations, services such as transportation and childcare are provided at specific levels. For example, transportation is provided for all and childcare for elementary school children only. However, most online systems ignore these needs.

   - **Requirement Identified:** Allow parents to select additional services, such as transportation and child care, during registration.

- **EazeTuition Implementation:** The "Other Services" module provides optional additional services (transportation and childcare) that can be customized during the subject registration process.

4. **Modernize Seamless Payment Processing**

   The current payment process is manual and requires an in-person visit. Only some systems offer online payments, while others focus only on expense tracking.

   - **Requirement Identified:** Allow users to securely and conveniently pay online, especially busy parents.
   - **EazeTuition Implementation:** The payment module integrates with Stripe to support various payment methods including Ringgit Malaysia to facilitate a streamlined cashless process.

5. **Formal Chat System for Communication**

   Observations revealed that leave and class scheduling were handled through WhatsApp, which is informal and difficult to track. Online platforms generally offer limited or no chat functionality for parents.

   - **Requirement Need:** In-app messaging functionality for structured, trackable communication between parents, tutors and centers.
   - **EazeTuition Implementation:** The chat module allows parents and tutors to communicate one-on-one within the application, ensuring that conversations related to classroom progress, excused absences, or announcements are secure and documented.

3.3 Requirement Analysis

3.3.1 Use Case Diagram

3.3.1.1 Subject Module

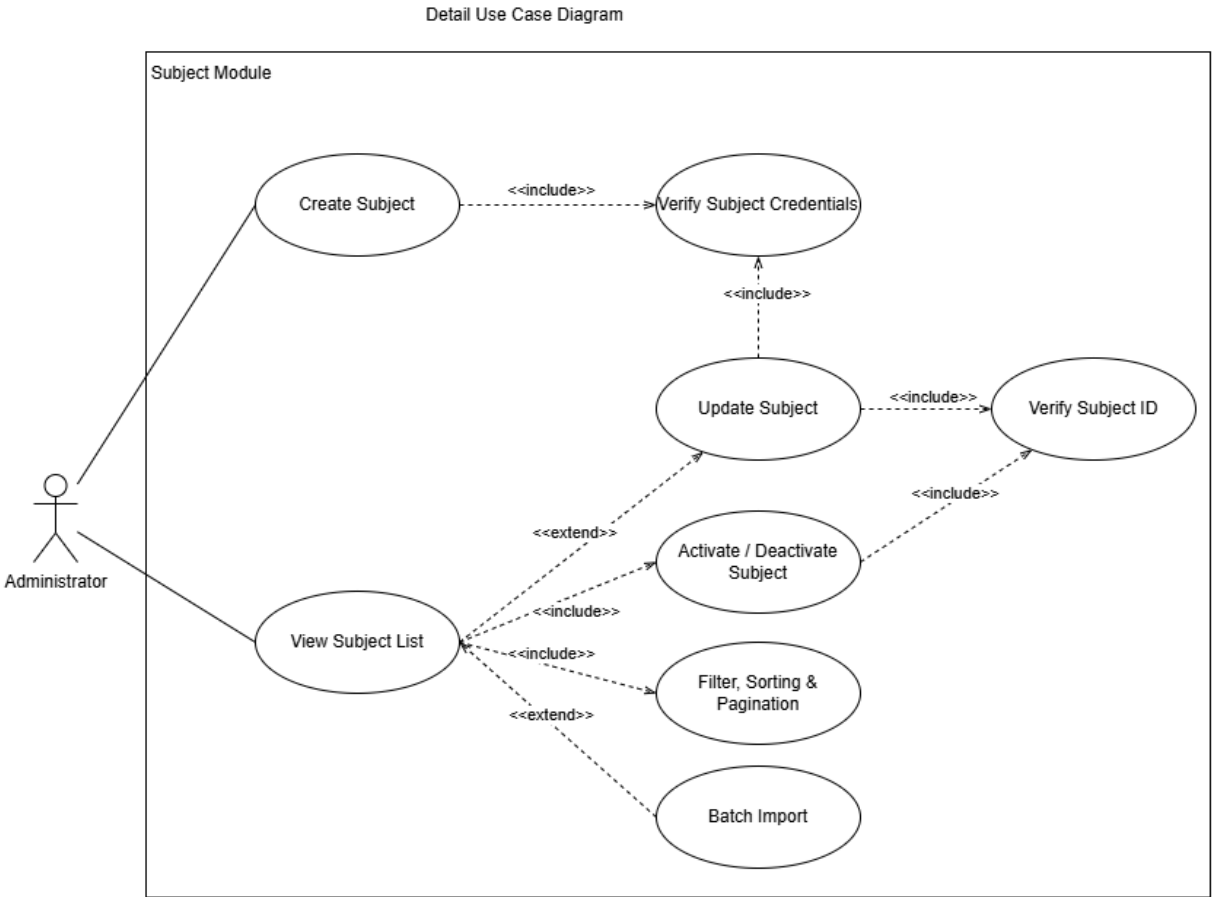


Figure 3.3.1.1 Use Case Diagram of Subject Module

3.3.1.2 Class Module

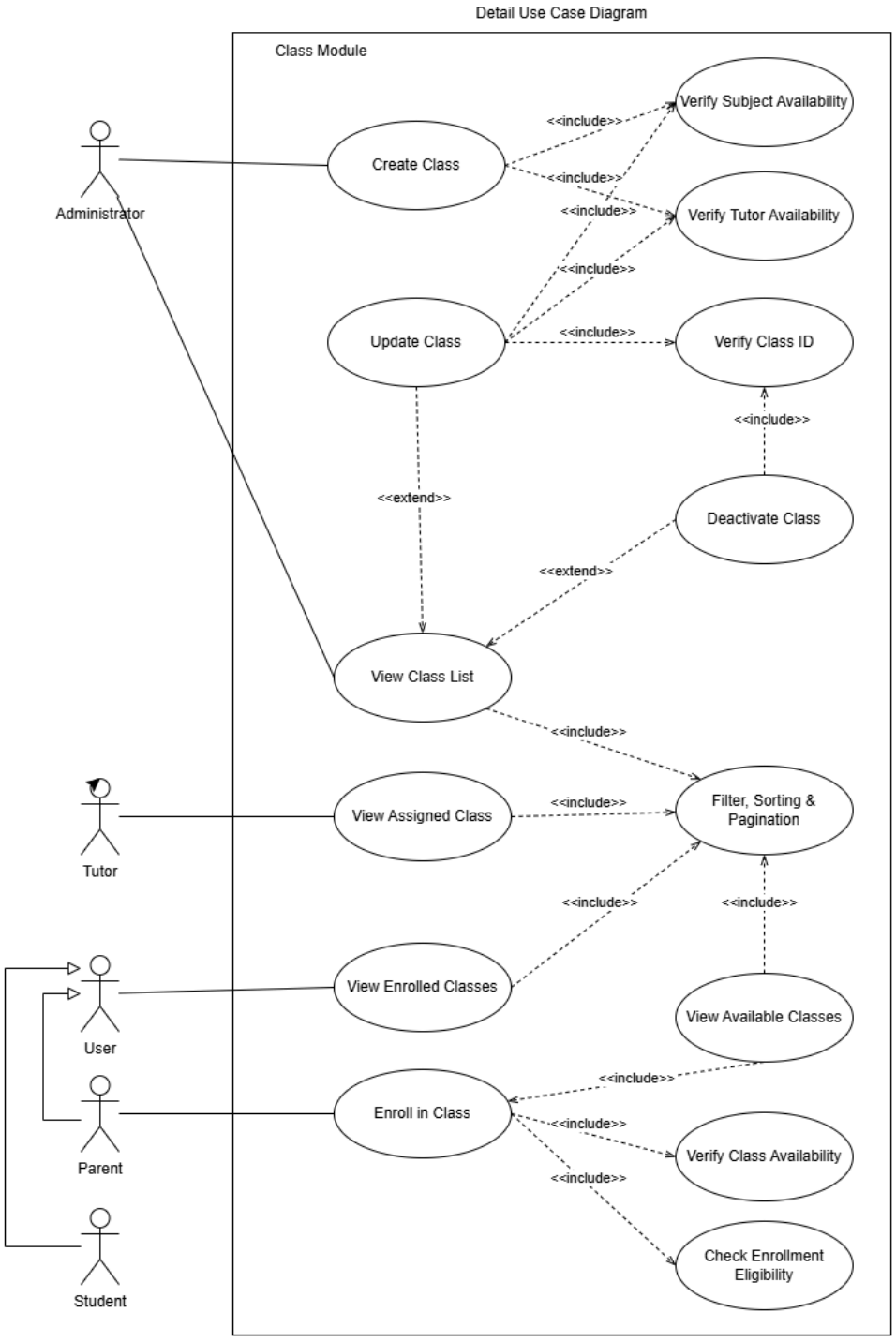


Figure 3.3.1.2 Use Case Diagram of Class Module

3.3.1.3 Class Schedule Module

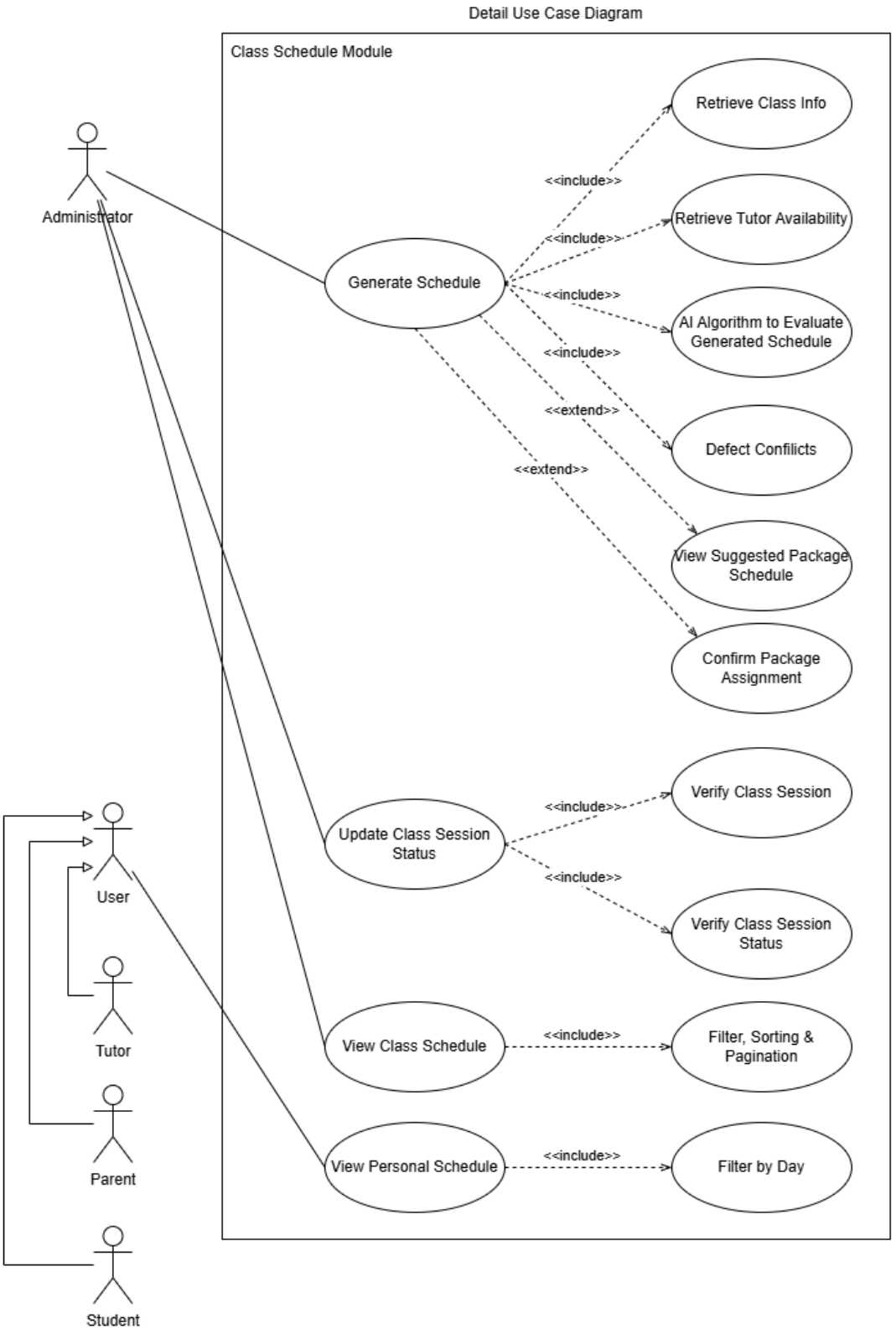


Figure 3.3.1.3 Use Case Diagram of Class Schedule Module

3.3.1.4 Payment Module

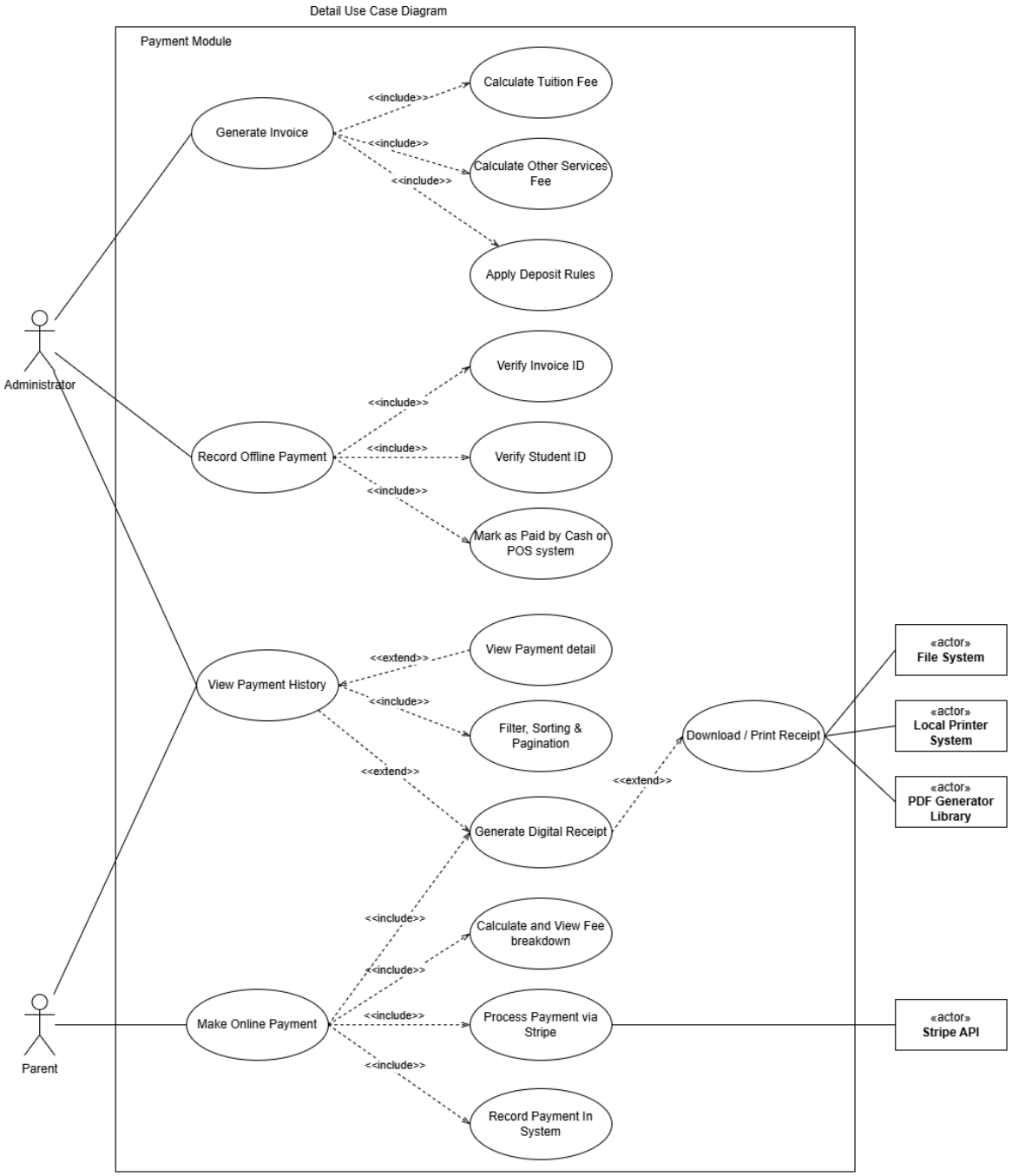


Figure 3.3.1.4 Use Case Diagram of Payment Module

3.3.1.5 Other Service Module

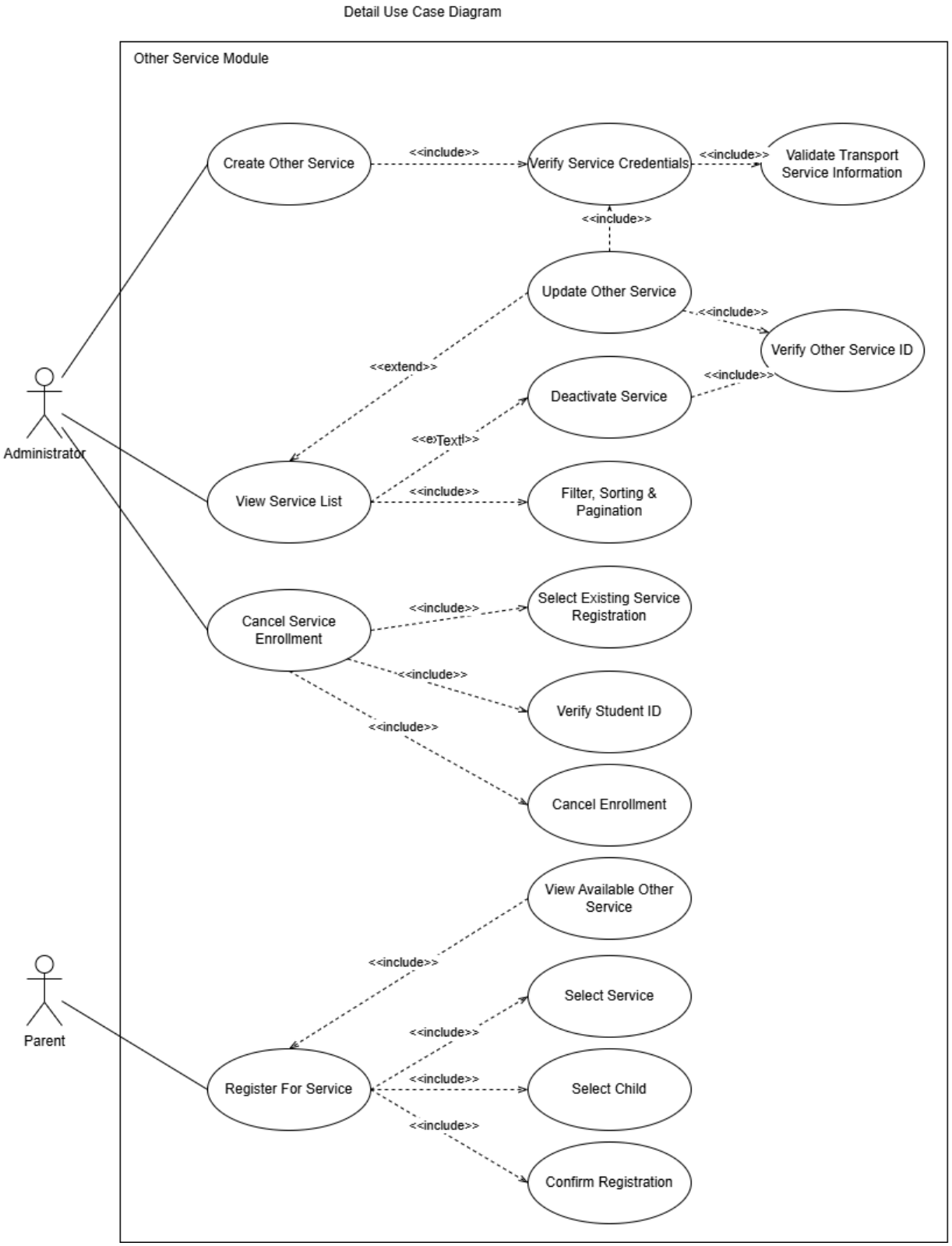
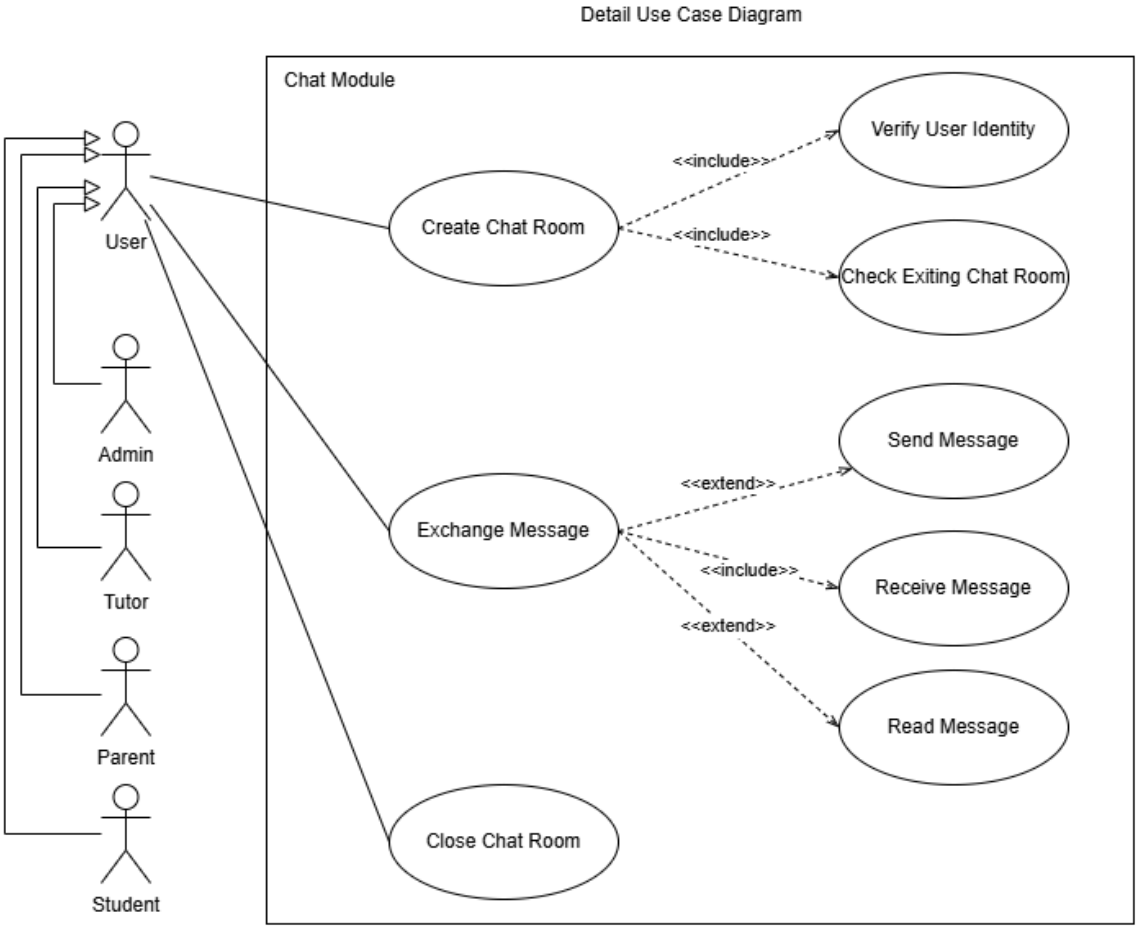


Figure 3.3.1.5 Use Case Diagram of Other Service Module

3.3.1.6 Chat Module



3.3.2 Use Case Description

3.3.2.1 Subject Module

Create Subject

Name of Use Case: Create Subject	
Brief Description: The administrator creates a new subject by filling in all necessary subject details. The system will verify the credentials before creating the subject.	
Actor: Administrator	
Precondition: The administrator must be logged into the system.	
Actor Action	**System Response**
1. Administrator clicks on the "Create Subject" button.	2. The system displays a form to input subject details.
3. Administrator fills in all the required fields and clicks "Create".	4. The system displays a confirmation popup with "Yes" and "Cancel".
5. The administrator selects one of the action buttons.	6. System redirects to the subject list.
Alternative Flows: A1. Step 2. If the subject name already exists in the same education level and also all the fees settings are the same, the system displays the message "Subject already exists." A2. Step 3. If the administrator leaves any required field empty or enters invalid data format. The system displays validation error messages "Please fill in all required fields.". A3. Step 5. If the administrator selects "Yes", the system saves the subject and redirects to the subject list with a success message. If the administrator selects "Cancel", the system stays on the page and does not save.	
Postcondition: New subject is stored in the database and displayed in the subject list.	

Table 3.3.2.1.1 Use Case Description of Create Subject

View Subject List

Name of Use Case: View Subject List
Brief Description: The administrator views the list of all existing subjects in the system. From this list, the administrator can perform filtering, sorting, and paging

actions. Additionally, the administrator may proceed to update a subject, activate/deactivate a subject, or perform a batch import.	
Actor: Administrator	
Precondition: The administrator must be logged into the system.	
Actor Action	**System Response**
1. Administrator clicks on the "Subject List" menu.	2. The system retrieves all existing subjects and displays them in a paginated table format.
	3. The system displays available actions for each subject including "Edit" and "Activate/Deactivate" buttons. The system also displays options for filtering, sorting, paging, and batch import.
4. Administrator interacts with any of the provided options.	5. The system redirects or responds accordingly based on the selected option.
Alternative Flows: A1. Step 2. If there are no subjects in the system, the subject list will be empt, and the system will display the message "No subjects found." A2. Step 4. If the administrator applies filters, the system filters the list and shows the filtered results. If the administrator clicks on "Edit" for a subject, the system extends to the Update Subject use case. If the administrator clicks on "Activate/Deactivate", the system extends to the Activate/Deactivate Subject use case. If the administrator clicks on the "Batch Import" button, the system includes the Batch Import Subject use case.	
Postcondition: The administrator is able to view and manage the list of subjects. The filtered or sorted results are updated accordingly. If further actions are taken (update, activate/deactivate, or batch import), the corresponding use cases will be executed.	

Table 3.3.2.1.2 Use Case Description of View Subject List

Update Subject

Name of Use Case: Update Subject
Brief Description: The administrator updates an existing subject's details such as name, level, or fees. The system verifies the subject's ID and credentials before updating the record.
Actor: Administrator
Precondition: The administrator must be logged into the system. The subject to be

edited must exist and in active status.	
Actor Action	**System Response**
1. The administrator clicks on the "Edit" button of a specific subject in the subject list.	2. The system retrieves the selected subject's details based on its ID and displays them in an editable form.
3. The administrator updates one or more fields and clicks the "Save" button.	4. The system verifies subject credentials and displays a confirmation popup with "Yes" and "Cancel" options.
5. The administrator selects an option.	6. System redirect user to the corresponding page or corresponding action.
Alternative Flows: A1. Step 2. If the system fails to find the subject ID, an error message "Subject not found." is displayed. The system redirects the user back to the subject list. A2. Step 4. If the updated subject name already exists in the same education level and fees settings, the system displays the message "A subject with this name and level already exists.". If any required fields are left blank or invalid, the system highlights the fields and displays an appropriate error message. A3. Step 6. If the administrator clicks "Cancel" in the confirmation popup, the system remains on the edit page and does not update the record. If the administrator clicks "Yes" in the confirmation popup, the system updates the information, shows the message "Subject updated successfully" and redirects to the subject list.	
Postcondition: The subject's information is successfully updated in the system database and reflected in the subject list.	

Table 3.3.2.1.3 Use Case Description of Update Subject

Activate / Deactivate Subject

Name of Use Case: Activate / Deactivate Subject	
Brief Description: The administrator can toggle a subject's status between "Active" and "Inactive" through the subject list. This controls whether the subject is available for class assignment and enrollment.	
Actor: Administrator	
Precondition: The administrator must be logged into the system. The subject to be activated or deactivated must exist.	
Actor Action	**System Response**
1. The administrator clicks the "Activate" or "Deactivate"	2. Verify if the subject exists.

button on a subject from the subject list.	
	3. The system displays a confirmation popup: "Are you sure you want to [Activate/Deactivate] this subject?" with "Yes" and "Cancel" options.
4. The administrator selects an option.	5. The system redirects the user to the corresponding action.
Alternative Flows: A1. If the subject record is not found, the system displays the error: "Subject not found" and redirects to the subject list. A2. Step 5. If the system fails to update the status due to a database error, it displays "Failed to update subject status. Please try again." A3. Step 5. If the administrator clicks "Yes" on the confirmation popup, the system updates the subject's status accordingly and displays a success message "The subject has successfully [Activate/Deactivated]". If the administrator clicks "Cancel" on the confirmation popup, the system does not apply any change.	
Postcondition: The subject's status is successfully updated in the database and reflected in the subject list.	

Table 3.3.2.1.4 Use Case Description of Activate / Deactivate Subject

Batch Import Subject

Name of Use Case: Batch Import Subject	
Brief Description: The administrator can upload a structured file (e.g., CSV or Excel) to import multiple subject records at once. The system will validate and store the data accordingly.	
Actor: Administrator	
Precondition: The administrator must be logged into the system. The import file follows the system's format specification.	
Actor Action	**System Response**
1. Administrator clicks the "Batch Import" button.	2. The system displays a file upload form with instructions on the required file format.
3. Administrator selects a file and clicks the "Import" button.	4. The system validates the file structure and content.
	5. The system imports all valid subject records, skips or flags

	invalid ones, and displays a pop-up summary with the "OK" button.
6. The administrator clicks the "OK" button.	7. The system redirects the user to the "Batch Import" page.
Alternative Flows: A1. Step 4. If the uploaded file format is incorrect, the system displays: "Invalid file format. Please upload a valid CSV or Excel file." A2. Step 5. If the system fails to update the status due to a database error, it displays "Failed to update subject status. Please try again." If one or more records fail validation, the system displays the invalid row number on the pop up message. A3. Step 5. If a system/database error occurs during import, the system shows: "Import failed due to a system error. Please try again later."	
Postcondition: Valid subject records are added to the database and displayed in the subject list. Invalid entries are reported and not saved.	

Table 3.3.2.1.5 Use Case Description of Batch Import Subject

3.3.2.2 Class Module

Create Class

Name of Use Case: Create Class	
Brief Description: The administrator creates a new class by selecting the subject, assigning a tutor and entering class details such as time, mode and capacity. The system verifies subject and tutor availability before saving.	
Actor: Administrator	
Precondition: The administrator must be logged into the system. At least one active subject and available tutor must exist in the system.	
Actor Action	**System Response**
1. Administrator clicks on the "Create Class" button.	2. The system displays a form to input class details, including subject, tutor, time, class type and registration status.
3. Administrator fills in the required fields and clicks "Create".	4. The system verifies the subject and the tutor is active and available and shows a confirmation popup: "Confirm to create class?" with "Yes" and "Cancel" buttons..
5. The administrator selects an	6. The system redirects the user to

option.	the corresponding action or corresponding page.
Alternative Flows: A1. Step 4. If the subject selected is inactive or unavailable, the system displays "Selected subject is not available.". If the tutor is already assigned to another class at the selected time, the system shows "Tutor is unavailable at this time." If any required field is left empty, the system highlights the field and displays "Please complete all required fields." A2. Step 6. If the administrator clicks "Yes" on the confirmation popup, the system saves the class and displays a success message .If the administrator clicks "Cancel" at confirmation, no data is saved and the system stays on the create page. A3. Step 6. If a database error occurs during saving, the system displays: "Failed to create class. Please try again later."	
Postcondition: The class is stored in the database and is visible in the class list for further actions.	

Table 3.3.2.2.1 Use Case Description of Create Class

View Class List

Name of Use Case: View Class List	
Brief Description: The administrator can view a list of all existing classes in the system. The system supports filtering, sorting, and pagination. From this list, the administrator can choose to update, activate/deactivate, or batch import classes.	
Actor: Administrator	
Precondition: The administrator must be logged into the system.	
Actor Action	**System Response**
1. Administrator clicks on the "Class List" menu.	2. The system retrieves all existing classes and displays them in a paginated table format.
	3. The system displays available actions for each class including "Edit" and "Deactivate" buttons. The system also displays options for filtering, sorting, paging, and batch import.
4. Administrator interacts with any of the provided options.	5. The system redirects or responds accordingly based on the selected option.
Alternative Flows: A1. Step 2. If there are no classes in the system, the subject list will be empty, and the system will display the message "No classes found."	

A2. Step 4. If the administrator applies filters, the system filters the list and shows the filtered results. If the administrator clicks on "Edit" for a class, the system extends to the Update Class use case. If the administrator clicks on "Deactivate", the system extends to the Deactivate Class use case.
Postcondition: The administrator successfully views and manages class records through the list.

Table 3.3.2.2.2 Use Case Description of View Class List

Update Class

Name of Use Case: Update Class	
Brief Description: The administrator updates details of an existing class such as date and time. Before saving, the system will verify the class ID, subject availability and tutor availability.	
Actor: Administrator	
Precondition: The administrator must be logged into the system. The class to be edited must exist and in active status.	
Actor Action	**System Response**
1. The administrator clicks on the "Edit" button of a specific class in the class list.	2. The system retrieves the selected class's details based on its ID and displays them in an editable form.
3. The administrator updates one or more fields and clicks the "Save" button.	4. The system verifies class credentials and displays a confirmation popup with "Yes" and "Cancel" options.
5. The administrator selects an option.	6. System redirect user to the corresponding page or corresponding action.
Alternative Flows: A1. Step 2. If the system fails to find the class ID, an error message "Class not found." is displayed. The system redirects the user back to the class list. If the subject selected is inactive or unavailable, the system displays "Selected subject is not available.". If the tutor is already assigned to another class at the selected time, the system shows "Tutor is unavailable at this time." If any required field is left empty, the system highlights the field and displays "Please complete all required fields. A2. Step 4. If the selected tutor is already assigned to another class at the same time, the system displays "Selected tutor is not available during this time." A3. Step 6. If the administrator clicks "Yes" on the confirmation popup, the system	

updates the class information and displays a success message .If the administrator clicks "Cancel" at confirmation, the system remains on the edit page and does not update the record. A4. Step 6. If a database error occurs during saving, the system displays: "Failed to update class. Please try again later."
Postcondition: The selected class is updated in the system and reflected in the class list.

Table 3.3.2.2.3 Use Case Description of Update Class

Deactivate Class

Name of Use Case: Deactivate Class	
Brief Description: The administrator can toggle the activation status of a class. An active class is visible and enrollable by parents/students, while a deactivated class is hidden and disabled from enrollment.	
Actor: Administrator	
Precondition: The administrator must be logged into the system. The class to be activated or deactivated must exist.	
Actor Action	**System Response**
1. The administrator clicks the "Deactivate" button on a class from the class list.	2. Verify if the class exists.
	3. The system displays a confirmation popup: "Are you sure you want to Deactivate this class?" with "Yes" and "Cancel" options.
4. The administrator selects an option.	5. The system redirects the user to the corresponding action.
Alternative Flows: A1. If the class record is not found, the system displays the error: "Class not found" and redirects to the class list. A2. Step 5. If the system fails to update the status due to a database error, it displays "Failed to update class status. Please try again." A3. Step 5. If the administrator clicks "Yes" on the confirmation popup, the system updates the class's status accordingly and displays a success message "The class has successfully [Activate/Deactivated]". If the administrator clicks "Cancel" on the confirmation popup, the system does not apply any change.	
Postcondition: The class status is updated in the database and reflected in the class list.	

Table 3.3.2.2.4 Use Case Description of Activate/ Deactivate Class

View Assigned Classes

Name of Use Case: View Assigned Classes	
Brief Description: The tutor can view a list of classes assigned to them. This allows the tutor to manage and track their teaching schedule efficiently.	
Actor: Tutor	
Precondition: The tutor must be logged into the system. The tutor has at least one class assigned.	
Actor Action	**System Response**
1. Tutor clicks on the "Assigned Classes" tab.	2. The system retrieves the list of classes assigned to the tutor and displays them in a list format.
3. Tutor uses search or filtering options.	4. The system filters and re-displays the assigned classes based on selected criteria.
Alternative Flows: A1. Step 2. If the tutor has no assigned classes, the system displays a message: "No classes assigned yet." If the system fails to retrieve the class list due to technical error, the system displays: "Unable to retrieve assigned classes. Please try again later."	
Postcondition: The tutor is able to view a list of their assigned classes and access all relevant details for teaching preparation.	

Table 3.3.2.2.5 Use Case Description of View Assigned Class (Tutor)

View Enrolled Classes

Name of Use Case: View Enrolled Classes	
Brief Description: The user (Parent or Student) can view a list of classes the student is currently enrolled in. This helps parents and students stay informed about the student's academic schedule.	
Actor: Parent / Student	
Precondition: The user must be logged into the system. Student must have at least one active class enrollment.	
Actor Action	**System Response**
1. Tutor clicks on the "My Classes" tab.	2. The system retrieves the list of classes the student is enrolled in and displays them in a list format.

3. The user uses search or filtering options.	4. The system filters and refreshes the list based on the user's input.
Alternative Flows: A1. Step 2. A1. If the student is not enrolled in any class, the system displays the message "No classes enrolled yet.". If the system encounters an error while fetching the data, it displays: "Unable to retrieve enrolled classes. Please try again later."	
Postcondition: The user is able to view a complete and updated list of the student's enrolled classes.	

Table 3.3.2.2.6 Use Case Description of View Enrolled Class (Student / Parent)

Enroll in Multiple Classes

Name of Use Case: Enroll in Multiple Classes	
Brief Description: The parent browses through the list of available classes and selects multiple classes to enroll their child at once. The system will validate all selections before confirming enrollment.	
Actor: Parent	
Precondition: The parent must be logged into the system. There must be at least one active class with available slots. The parent's child must be eligible to enroll.	
Actor Action	**System Response**
1. Parent clicks on the "Enroll New Class" button.	2. The system retrieves available classes to display.
3. Parent select the child and classes for the new enrollment.	
4. Parent clicks the "Checkout" button.	5. The system displays the total fees and a confirmation popup showing the list of selected classes, with "Yes" and "Cancel" options.
6. The parent selects one of the options.	7. The system redirects the user to the corresponding action or corresponding page.
Alternative Flows: A1. Step 2. If the child is already enrolled in one of the selected classes, the system skips that class and shows "Already enrolled in [Class Name]. Enrollment skipped." If a class is full, the system skips it and shows "Enrollment failed for [Class Name]. This class is already full." If there is a schedule conflict between a selected class and another enrolled class, the system shows "Schedule conflict for [Class Name]. Enrollment is not allowed."	

A2. Step 5. If the parent clicks "Yes" at the confirmation step, the system enrolls the child in each and shows a success message. If the parent clicks "Cancel" at the confirmation step, the system does not proceed with any enrollment and stays on the current page.
Postcondition: The child is successfully enrolled in all valid and eligible selected classes. The system updates the class enrollment records and reflects them in the user's view.

Table 3.3.2.2.7 Use Case Description of Enrolled in Multiple Class

3.3.2.3 Class Schedule Module

Generate Schedule

Name of Use Case: Generate Schedule	
Brief Description: The administrator triggers the system to automatically generate class schedules using a rules-based scheduling engine. The engine evaluates tutor availability, room capacity, class duration, conflict avoidance rules and subject-level constraints to form multiple valid schedule package options. The system highlights any conflicts and displays recommended schedules in grouped package form. AI lifestyle and performance scoring is used only to evaluate and rank the generated schedules—not to generate time slots.	
Actor: Administrator	
Precondition: The administrator must be logged into the system. Class and tutor data must already be available in the system.	
Actor Action	**System Response**
1. Administrator clicks the "Generate Schedule" button.	2. The system runs the rules-based scheduling algorithm to produce conflict-free schedule packages. The system then applies AI scoring (Lifestyle + Performance Model) to rank or evaluate each generated package.
	3. The system shows each package with conflict indicators, tutor assignments and time slot details.
4. Administrator checks the box(es) for one or more preferred schedule packages.	5. The system enables the "Confirm Package" button once at least one schedule is selected.
6. Administrator clicks the "Confirm Package" button.	7. The system prompts for confirmation with a popup: "Are you sure you want to

	assign the selected schedules?"
8. The administrator selects one of the options.	9. The system redirects the administrator to corresponding action or corresponding page.
Alternative Flows: A1. Step 5. If no schedule is selected and the admin tries to confirm, the system displays "Please select at least one schedule package to proceed." A2.Step 8. If the admin selects "Yes" on the confirmation popup, the system finalizes and stores the selected schedules, linking them to their respective classes and displays a success message. If the admin cancels the confirmation popup, the system remains on the same page and does not assign the schedules.	
Postcondition: The selected package schedules are officially assigned and saved in the database. Class sessions are updated accordingly.	

Table 3.3.2.3.1 Use Case Description of Generate Schedule

Update Class Session Status

Name of Use Case: Update Class Session Status	
Brief Description: The administrator updates the status of a class session. The system verifies the session details and allows status modification based on session validity.	
Actor: Administrator	
Precondition: The administrator must be logged into the system. The class schedule has been generated. The session to be updated exists in the system.	
Actor Action	**System Response**
1. Administrator navigates to the class schedule interface.	2. The system displays a calendar of upcoming and past sessions.
3. Administrator selects a session to update.	4. The system verifies the session ID and retrieves session details.
5. Administrator clicks "Cancel Class" and selects a new status.	6. The system verifies whether the session is eligible for status update and prompts for confirmation with a popup: "Are you sure you want to cancel the class of selected class session?".
7. The administrator selects one of the options.	8. The system redirects the administrator to corresponding action or corresponding page.
Alternative Flows:	

A1. Step 4. If the session ID is invalid or not found "Unable to locate the selected session. Please refresh and try again." A2. Step 6. If the session has already been marked as ongoing or completed, display message "Session already finalized. Status cannot be changed." A2.Step 8. If the admin selects "Yes" on the confirmation popup, The system updates the session record and shows a success message. If the admin selects "Cancel" on the confirmation popup, The system does not apply any changes and remains on the same screen.
Postcondition: The selected class session's status is updated in the system and reflected in all related class views.

Table 3.3.2.3.2 Use Case Description of Update Class Session Status

View Class Schedule

Name of Use Case: View Class Schedule	
Brief Description: The administrator views the schedule of all classes in calendar or list format. The system supports filtering, sorting, pagination, and printing of the displayed schedule.	
Actor: Administrator	
Precondition: The administrator must be logged into the system. Class schedule has been generated.	
Actor Action	**System Response**
1. Administrator clicks on the "View Class Schedule" tab.	2. The system retrieves and displays all scheduled classes in list/calendar format.
3. Administrator applies filter/sorting options.	4. The system refreshes the schedule list based on the selected criteria.
Alternative Flows: A1. Step 2. If no schedule data is available "No schedule found. Please generate a schedule first." A2. Step 4. If filtering returns no results "No matching records found for the selected filter."	
Postcondition: The administrator successfully views and optionally prints the class schedule.	

Table 3.3.2.3.3 Use Case Description of View Class Schedule

View Personal Schedule

Name of Use Case: View Personal Schedule	
Brief Description: Users (parents, students, tutors) view their child or own class schedules. The system displays a filtered list or calendar based on user role and associated classes. Users may also print their schedule.	
Actor: Parent / Student / Tutor	
Precondition: The user is logged into the system. The user or user's child has at least one class assigned or enrolled.	
Actor Action	**System Response**
1. The user clicks on "My Schedule" or "My Child Schedule".	2. The system retrieves and displays the user's schedule in a calendar or list view.
3. The user applies filters.	4. The system updates the displayed schedule according to the selected filter.
Alternative Flows: A1. Step 2. If the user has no schedule, the system displays "No scheduled classes available." A2. Step 4. If the filter yields no result, the system displays "No matching schedule for the selected filters."	
Postcondition: The user successfully views and optionally prints their personal schedule.	

Table 3.3.2.3.4 Use Case Description of View Personal Schedule

3.3.2.4 Payment Module

Generate Invoice

Name of Use Case: Generate Invoice	
Brief Description: The administrator generates an invoice for a student by retrieving class and service data. After calculating applicable fees and applying deposit rules, the system prompts the administrator to either proceed to payment or cancel the process.	
Actor: Administrator	
Precondition: The administrator is logged into the system. The student has at least one enrolled class or registered service.	
Actor Action	**System Response**
1. Administrator navigates to the Invoice Management section	2. The system displays a list of students eligible for invoicing.

3. Administrator selects a student and clicks "Generate Invoice".	4. The system retrieves the student's enrolled classes and selected services.
	5. The system calculates tuition fees based on enrolled subjects, other services fee and deposit rules if applicable.
	6. The system displays invoice preview with fee breakdown and two buttons: "Proceed to Payment" and "Cancel".
7. Administrator selects an action.	8. The system redirect administrator to corresponding action or corresponding page.
Alternative Flows: A1. Step 4. If the student has no active enrollment or services the system displays "No applicable fees found for this student." A2. Step 8. If the admin selects "Proceed to Payment", the system redirects to the payment interface for this invoice. If "Cancel", the system discards the process and returns to the previous page.	
Postcondition: The invoice is generated and presented to the parent for immediate action.	

Table 3.3.2.4.1 Use Case Description of Generate Invoice

Record Offline Payment

Name of Use Case: Record Offline Payment	
Brief Description: The administrator records an offline payment made at the counter by a parent or student. The system verifies the invoice and student IDs before marking the payment as completed.	
Actor: Administrator	
Precondition: The administrator is logged into the system.The parent/student has paid at the counter. The invoice must be generated and be in an unpaid status.	
Actor Action	**System Response**
1. Administrator clicks "Proceed to Payment" after generating an invoice.	2. The system displays the offline payment form, with fee breakdown carried over from invoice generation.
3. The administrator reviews the pre-filled student and payment details.	4. System validates the session data and displays confirmation.

5. Administrator confirms receipt of offline payment.	6. The system records the payment directly, bypassing invoice storage.
	7. The system displays a success message and provides options to download or print the receipt.
	8. The system generates a digital receipt, displays a success message and provides options "Print Receipt" or "Back to Payment List"..
9. The administrator selects one of the options.	10. The system redirects the administrator to corresponding action or corresponding page.
Alternative Flows: A1. At any point, if the administrator cancels the process, the system discards the payment and returns to the previous screen without saving anything. A2.Step 10. If the administrator chooses "Print Receipt", the system connects to the printer and prints the receipt. If "Back to Payment List". is selected, the system redirects to the payment list page.	
Postcondition: The payment is recorded in the system, and a receipt is generated. The invoice is not stored as a persistent record; the payment is processed in real time during the session.	

Table 3.3.2.4.2 Use Case Description of Record Offline Payment

View Payment History

Name of Use Case: View Payment History	
Brief Description: The user (administrator, parent, or student) views a list of recorded payment transactions. The system supports filtering, sorting, pagination, and provides options to view invoice details or download/print digital receipts. The data scope depends on the user's role.	
Actor: Administrator / Parent	
Precondition: The user is logged into the system. Payment transactions must exist in the system.	
Actor Action	**System Response**
1. The user navigates to the "Payment History" section from the menu.	2. The system displays a list of payment records relevant to the user.
3. The user applies filters, sorting, or pagination if desired.	4. The system updates the list according to the user's filter or

	sort criteria.
5. The user clicks on a payment record to view more details.	6. The system displays the full invoice details, including fee breakdown and payment status.
7. The user selects to download or print the receipt (if needed).	8. The system generates and displays a digital receipt with options to download or print.
Alternative Flows: A1. Step 2. If there are no payment records available. The system displays a message "No payment records found." If a parent attempts to access another user's payment history, the system denies access and displays an "Unauthorized" message. A2.Step 8. If an error occurs while attempting to download or print a receipt, the system shows an error message and suggests the user try again.	
Postcondition: The user successfully views their payment history records. If chosen, the digital receipt is successfully downloaded or printed.	

Table 3.3.2.4.3 Use Case Description of View Payment History

Make Online Payment

Name of Use Case: Make Online Payment	
Brief Description: The parent initiates an online payment for their tuition or other service fees. The system retrieves the invoice details, displays the fee breakdown, processes the payment through Stripe and generates a digital receipt.	
Actor: Parent	
Precondition: The parent is logged into the system. The parent has at least one unpaid invoice.	
Actor Action	**System Response**
1. The parent selects "Make Payment" from the dashboard.	2. The system retrieves unpaid invoice(s) and displays the associated fee breakdown for review.
3. The parent reviews the fee breakdown and clicks "Proceed to Pay".	4. The system verifies invoice validity and payment eligibility.
5. The parent provides payment details and submits the payment.	6. The system processes the payment via Stripe and successfully completes.
	7. The system records the payment, generates a digital

	receipt, and displays a success message.
8. The parent selects either "Download Receipt" or "Back to Payment History".	9. The system carries out the selected action.
Alternative Flows: A1. Step 2. If the parent cancels before submitting payment, the system halts the transaction and returns to the previous screen without saving any payment. A2.Step 6. If the payment fails (e.g., card declined or connection error), the system displays an appropriate error message and prompts the parent to retry. A3. Step 9. If the parent chooses to download the receipt, the system generates a downloadable PDF file for the payment. If the parent chooses "Back to Payment History", redirect back to the payment history list.	
Postcondition: The system has successfully recorded the payment and issued a digital receipt. The parent can download or print the receipt. No invoice is stored for future reference, the transaction is processed immediately and completely within the same session.	

Table 3.3.2.4.4 Use Case Description of Make Online Payment

3.3.2.5 Other Service Module

Create Other Service

Name of Use Case: Create Other Service	
Brief Description: The administrator creates a new other service by entering required information and validating service parameters such as address.	
Actor: Administrator	
Precondition: The administrator is logged into the system. No conflicting service exists for the same zone, time, and direction.	
Actor Action	**System Response**
1. The administrator selects the "Create New Service" button.	2. The system opens the service creation form.
3. The administrator inputs the service, service type and address if needed.	4. The system verifies the completeness of the data.
5. The administrator submits the form.	6. The system validates transport credentials.
	7. The system creates the service record and displays a success

	message.
Alternative Flows: A1. Step 4. If the selected service's service type is transport and the address is invalid, the system displays an error message and prompts for filling in. If a duplicate service already exists for the same time and direction in the same zone, the system prevents creation and displays a conflict warning.	
Postcondition: The new other service is successfully created and visible in the service list. The service is now available for parents to view and register.	

Table 3.3.2.5.1 Use Case Description of Create Other Service

View Service List

Name of Use Case: View Service List	
Brief Description: The administrator views a list of existing other services, with options to update and deactivate. The system also supports filtering, sorting, and pagination.	
Actor: Administrator	
Precondition: The administrator is logged into the system. At least one other service exists in the system.	
Actor Action	**System Response**
1. The administrator navigates to the "Service List" page.	2. The system retrieves and displays a paginated list of all services.
	3. The system displays available actions for each service including "Edit" and "Deactivate" buttons. The system also displays options for filtering, sorting, paging, and batch import.
4. The administrator selects one of the action buttons.	5. The administrator selects one of the action buttons.
	6. System redirects to the subject list.
Alternative Flows: A1. Step 2. If there are no subjects in the system, the other service list will be empty, and the system will display the message "No services found." A2. Step 4. If the administrator applies filters, the system filters the list and shows the filtered results. If the administrator clicks on "Edit" for a service, the system extends to the Update Service use case. If the administrator clicks on "Deactivate", the system extends to the Deactivate Service use case.	

Postcondition: The administrator is able to view and manage the list of services. The filtered or sorted results are updated accordingly. If further actions are taken, the corresponding use cases will be executed.

Table 3.3.2.5.2 Use Case Description of View Other Service List

Deactivate Other Service

Name of Use Case: Deactivate Other Service	
Brief Description: The administrator can deactivate an existing other service to control its availability for registration by parents. Deactivated services are not shown to parents for new registrations.	
Actor: Administrator	
Precondition: The administrator is logged into the system. At least one other service exists in the system.	
Actor Action	**System Response**
1. The administrator views the list of other services.	2. The system displays a list of services with their current status (Active/Inactive).
3. The administrator clicks the "Deactivate" button on a specific service.	4. The system prompts for confirmation: "Are you sure you want to deactivate this service?"
5. The administrator confirms the action.	6. The system updates the status of the selected service accordingly and refreshes the list with updated status.
	7. The system displays a success message indicating the new status of the service.
Alternative Flows: A1. Step 6. If the administrator cancels the confirmation dialog, the system takes no action and returns to the service list. A2. Step 6. If the system encounters a database error while updating, an error message is shown "Unable to update service status. Please try again later."	
Postcondition: The selected service is successfully activated or deactivated. Parents will only be able to register for services that are in the Active state.	

Table 3.3.2.5.3 Use Case Description of Activate / Deactivate Other Service

Cancel Service Enrollment

Name of Use Case: Cancel Service Enrollment

Brief Description: The administrator cancels a student's enrollment in an existing other service. The system verifies the selected registration and student identity before processing the cancellation.	
Actor: Administrator	
Precondition: The administrator is logged into the system. At least one other service exists in the system. The student is currently registered for a service.	
Actor Action	**System Response**
1. The administrator navigates to the "Service Enrollment List" tab.	2. The system displays all active service enrollments along with filtering, search, and selection options.
3. The administrator selects a specific enrollment to cancel.	4. The system retrieves and displays the student ID and service details for confirmation.
5. The administrator verifies the information and confirms the cancellation.	6. The system cancels the enrollment and updates the service status accordingly.
	7. The system displays a success message and returns to the service enrollment list.
Alternative Flows: A1. Step 4. If no enrollment exists for the selected student, the system shows "No enrollment found for this student." If the administrator applies filters or search on the list, the system displays the filtered results accordingly. A2. Step 6. If the administrator cancels the operation before confirmation, the system aborts the cancellation and returns to the previous screen.	
Postcondition: The selected student's enrollment in the service is successfully canceled, and the service capacity or availability is updated if applicable.	

Table 3.3.2.5.4 Use Case Description of Cancel Other Service Enrollment

Register For Service

Name of Use Case: Register For Service
Brief Description: The parent selects an available transport or childcare service and registers one of their children for it. The system verifies the selected service and child before confirming the registration.
Actor: Parent
Precondition: The parent is logged into the system. At least one eligible child is linked to the parent's account. At least one active service is available.

Actor Action	System Response
1. The parent clicks the "Enroll New Service" button.	2. The system retrieves available services and displays them in a list.
3. The parent selects a child to enroll and confirms the registration.	4. The system verifies the child and service information.
	5. The system confirms the enrollment and adds the service to the child's registered service and a success message is shown.
Alternative Flows: A1. Step 3. If the parent cancels during registration, the system aborts the process and returns to the service list. A2. Step 4. If the selected child is already enrolled in the service, the system displays an error message preventing duplicate enrollment.	
Postcondition: The selected child is successfully registered for the chosen service, and the registration is recorded in the system.	

Table 3.3.2.5.5 Use Case Description of Register for Service (Parent)

Update Other Service

Name of Use Case: Update Other Service	
Brief Description: The administrator updates details of an existing service such as name, fees and description. Before saving, the system will verify the service ID.	
Actor: Administrator	
Precondition: The administrator must be logged into the system. The service to be edited must exist and in active status.	
Actor Action	**System Response**
1. The administrator clicks on the "Edit" button of a specific service in the service list.	2. The system retrieves the selected service's details based on its ID and displays them in an editable form.
3. The administrator updates one or more fields and clicks the "Save" button.	4. The system verifies class credentials and displays a confirmation popup with "Yes" and "Cancel" options.
5. The administrator selects an option.	6. System redirect user to the corresponding page or corresponding action.

Alternative Flows: A1. Step 2. If the system fails to find the service ID, an error message "Service not found." is displayed. The system redirects the user back to the service list. If any required field is left empty, the system highlights the field and displays "Please complete all required fields. A2. Step 6. If the administrator clicks "Yes" on the confirmation popup, the system updates the service information and displays a success message .If the administrator clicks "Cancel" at confirmation, the system remains on the edit page and does not update the record. A3. Step 6. If a database error occurs during saving, the system displays: "Failed to update class. Please try again later."
Postcondition: The selected service is updated in the system and reflected in the service list.

Table 3.3.2.5.6 Use Case Description of Update Other Service

3.3.2.6 Chat Module

Create Chat Room

Name of Use Case: Create Chat Room	
Brief Description: The user creates a chat room with another user. The system verifies the user's identity and checks if a chat room already exists before creating a new one.	
Actor: User (Parent, Tutor, Student)	
Precondition: The user must be logged into the system.	
Actor Action	**System Response**
1. The user clicks on the "Chat" icon or selects another user to chat with.	2. System verifies user identity.
	3. The system checks if a chat room already exists between both users.
4. If no chat room exists, the user confirms creation.	5. The system creates a new chat room and redirects the user to the chat interface.
Alternative Flows: A1. Step 2. If the user identity verification fails, the system shows an error message "User not authorized to initiate chat." A2. Step 3. If a chat room already exists, the system skips creation and opens the existing chat room.	

A3. Step 4. If the user cancels the chat room creation, the system returns to the previous page.
Postcondition: A chat room is created or reused, and the chat interface is opened.

Table 3.3.2.6.1 Use Case Description of Create Chat Room

Exchange Message

Name of Use Case: Exchange Message	
Brief Description: A user sends and receives messages within a chat room. When a message is sent, the system delivers it to the receiver and sends a notification if the receiver is offline or not currently in the chat room. Messages can also be marked as read once viewed.	
Actor: Sender(Parent, Tutor, Student), Receiver (Parent, Tutor, Student)	
Precondition: Both sender and receiver have an existing chat room and are registered users.	
Actor Action	**System Response**
1. Sender enters an existing chat room.	2. System loads and displays previous message history
3. The sender types a message and clicks "Send".	4. The system saves and delivers the message.
5. The receiver opens the chat room.	6. The system marks the message as "Read".
7. Receiver replies with a new message.	8. The system repeats the above flow in reverse .
Alternative Flows: A1. Step 2. If the sender has no internet connection, the message is queued and displayed as "Waiting to send…". If the system fails to save the message, it shows "Message failed. Retry?". If the receiver is offline or not in the chat room, the system triggers a notification. A2. Step 3. If the receiver does not open the chat room, the message remains "Unread".	
Postcondition: Messages are stored in the database, marked as "Read" when viewed, and appropriate notifications are sent when needed.	

Table 3.3.2.6.2 Use Case Description of Exchange Message

Close Chat Room

Name of Use Case: Close Chat Room

Brief Description: The user exits or closes the chat interface. The chat room remains available for future access, and no data is deleted.	
Actor: User (Parent, Tutor, Student)	
Precondition: The user must be logged into the system and is in a chat room.	
Actor Action	**System Response**
1. User clicks the "Back" button or navigates away from chat.	2. System exits the chat interface.
	3. System retains the chat room for future use.
Alternative Flows: A1. Step 1. If the app is closed unexpectedly, the system still retains chat history upon next login. A2. Step 2. If a user receives a message after exiting, the system stores it for next access and may notify via push notification.	
Postcondition: The chat room remains intact. The user is no longer active in the chat interface.	

Table 3.3.2.6.3 Use Case Description of Close Chat Room

3.3.3 Functional and Non-functional Requirements

3.3.3.1.1 Functional Requirements

1.0 Subject Module

1.1 Subject Management

1.1.1 The system shall allow administrators to view all created subjects with sorting, filtering, and pagination features to support efficient navigation.

1.1.2 The system shall allow administrators to create new subjects with a customizable subject name and multiple class-type fee configurations, including but not limited to “Normal” and “Focus” class fee structures.

1.1.3 The system shall validate new subject entries by performing duplicate detection based on subject name and fee type combinations, preventing administrators from creating subjects with identical definitions.

1.1.4 The system shall allow administrators to update existing subject details, including subject name and individual fee settings for each class type.

1.1.5 The system shall allow administrators to deactivate outdated or unused subjects, preventing them from being used in future class creation.

1.1.6 When a subject is deactivated, the system shall automatically cascade the deactivation to all related entities.

1.1.7 The system shall allow administrators to reactivate previously deactivated subjects when needed, restoring their availability for class creation and scheduling activities.

1.2 Batch Subject Import

1.2.1 The system shall allow administrators to upload a CSV or Excel file containing subject data.

1.2.2 The system shall validate imported data by checking duplicate subject name, duplicate subject, education level and fee structure and incomplete or invalid fields.

1.2.3 system shall store all successfully validated subjects into the database in bulk, while rejecting invalid rows with detailed error messages.

2.0 Class Schedule Module

2.1 Class Management

2.1.1 The system shall allow administrators to view created classes with support for sorting, filtering, and pagination.

2.1.2 The system shall allow administrators to create, edit, and deactivate classes.

2.1.3 The system shall allow each class to be associated with a specific subject, tutor and maximum student capacity.

2.1.4 The system shall allow administrators to configure class types (e.g., Normal, Focus) and ensure the selected subject supports the selected class type.

2.1.5 The system shall allow tutors to view their own assigned classes in a dedicated class view.

2.1.6 The system shall allow parents to view classes available for enrollment.

2.1.7 The system shall allow students and parents to view enrolled classes in a dashboard or calendar interface.

2.1.8 The system shall perform cascading deactivation. When a subject is deactivated, all related classes and class-detail schedules shall automatically be deactivated.

2.1.9 The system shall validate room usage to prevent multiple classes from using the same room at overlapping times.

2.1.10 The system shall allow administrators to toggle whether a class is “involved for scheduling”, enabling them to control whether the class should be included in automatic schedule generation.

2.2 Class Enrollment Control

2.2.1 The system shall allow parents to select available classes and enroll their children without requiring prior subject enrollment.

2.2.2 The system shall allow administrators to perform class enrollment on behalf of students.

2.2.3 The system shall enforce class capacity limits to prevent over-enrollment.

2.2.4 The system shall prevent students from enrolling in overlapping class time slots.

2.2.5 The system shall validate registration rules, including capacity, duplication, class-type restrictions and schedule conflicts in real time and display warning messages when issues occur.

2.2.6 The system shall finalize enrollment only after successful payment.

2.2.7 The system shall allow staff to view, filter and manage the list of enrolled students for each class.

2.2.8 The system shall automatically generate related payment records upon successful enrollment submission.

2.2.9 The system shall prevent enrollment into deactivated classes or classes belonging to deactivated subjects.

2.3 Rule-Based Auto Scheduling

2.3.1 The system shall automatically generate class schedules using predefined rule-based scheduling algorithms.

2.3.2 The system shall avoid schedule conflicts by ensuring no overlapping time slots for the same student, tutor or room.

2.3.3 The system shall allocate rooms based on availability and capacity constraints.

2.3.4 The system shall ensure each generated class package includes at least one class per selected subject.

2.3.5 The system shall detect conflicts dynamically and reassign class details when necessary.

2.3.6 The system shall auto-generate time slots for unassigned classes based on predefined scheduling windows such as morning, afternoon, full-day.

2.3.7 The system shall prioritize the placement of fixed-time classes before generating time slots for flexible classes.

2.3.8 The system shall re-evaluate schedules when leftover or missing subject classes are detected.

2.3.9 The system shall prevent tutor double-booking across all generated schedule sets.

2.3.10 The system shall validate room availability during the auto-generation process to avoid room conflicts.

2.4 AI Schedule Evaluation

2.4.1 The system shall compute lifestyle scores using the trained Lifestyle Model based on the student's stress level, study time and sleep duration.

2.4.2 The system shall compute performance scores using the trained Performance Model based on study–rest balance.

2.4.3 The system shall calculate a weighted recommendation score combining lifestyle and performance metrics.

2.4.4 The system shall classify schedules into ranked categories, such as "Highly Recommended," "Recommended," "Moderate," and "Not Recommended."

2.4.5 The system shall display evaluation results for each generated schedule set without modifying the rule-based schedule.

2.4.6 The system shall refresh the AI evaluation score whenever a schedule package is regenerated or updated.

2.5 Class Package Schedule Management

2.5.1 The system shall allow staff to view suggested class packages generated through the auto-scheduling module.

2.5.2 The system shall allow staff to deactivate generated schedule packages but shall not allow editing of package contents.

2.5.3 The system shall apply signature-based integrity validation to ensure schedule package data is not modified when transmitted between backend and frontend.

2.5.4 The system shall block activation of schedule packages that contain unresolved scheduling conflicts, invalid class combinations, missing subjects, duplicated class slots or room allocation conflicts.

2.5.5 The system shall allow administrators to toggle the “Open for Registration” status for schedule packages, enabling parents to enroll only when a package is explicitly set as available.

2.5.6 The system shall ensure that any class or subject deactivation automatically propagates to linked schedule packages, updating their availability accordingly.

2.5.7 The system shall automatically update the status of scheduled class entries within the timetable when a linked class, subject, tutor assignment, or room allocation changes or is deactivated.

2.6 Centralized Messaging

2.6.1 The system shall allow administrators to cancel any scheduled class within the generated timetable, ensuring the cancellation is instantly reflected in all parent-facing, student-facing and staff-facing views.

2.6.2 The system shall support creating a replacement class to compensate for a canceled lesson, enabling administrators to select a new time slot, tutor and room where applicable.

2.6.3 The system shall prevent replacement class creation if the proposed replacement time slot conflicts with existing student schedules, tutor schedules, room allocations or subject-level constraints.

2.6.4 The system shall ensure replacement classes follow the same validation rules and constraints as regular scheduled classes, including tutor availability, room usage and cross-subject enrollment conflicts.

2.6.5 The system shall maintain a complete replacement history, enabling administrators to review all canceled-and-replaced lesson chains.

3.0 Chat Module

3.1 Centralized Messaging

3.1.1 The system shall allow staff and parents to send and receive one-to-one messages.

3.1.2 The system shall store and display message history capability.

3.1.3 The system shall send in-app push notifications to alert staff and parents of new messages.

4.0 Payment Module

4.1 Online and Offline Payment

4.1.1 The system shall support online payment through Stripe, including validation, checkout redirection and webhook confirmation.

4.1.2 The system shall allow staff to record offline payments, including cash, bank transfer or partial payments.

4.1.3 The system shall validate received cash amounts and automatically compute the change amount for offline transactions.

4.1.4 The system shall prevent duplicate payments for the same billing record regardless of online or offline method.

4.2 Fee Calculation

4.2.1 The system shall calculate tuition fees based on enrolled class type such as Normal or Focus, subject fee configuration and class count.

4.2.2 The system shall support automatic application of class-based tiered discounts when students enroll in multiple tuition classes.

4.2.3 The system shall calculate service fees such as transport, childcare using the appropriate billing model such as monthly fixed fee, attendance-based fee or deposit logic.

4.2.4 The system shall dynamically recompute total payable amounts when enrollment changes occur such as class enrollment or cancellation or service modifications.

4.2.5 The system shall allow parents to preview a full fee breakdown before confirming payment.

4.3 Payment Generation & Billing Lifecycle

4.3.1 The system shall automatically generate a pending payment record upon class enrollment or service registration, regardless of whether initiated by administrators or parents.

4.3.2 The system shall generate recurring monthly payments automatically based on active enrollments and active services at the beginning of each billing cycle.

4.3.3 The system shall prevent monthly charges from being created for deactivated classes or canceled services.

4.3.4 The system shall automatically cancel pending payments when the related enrollment, class or service is canceled before payment is made.

4.3.5 The system shall automatically link billing items to the correct enrollment, service or replacement attendance record for audit accuracy.

4.4 Payment Status & Enrollment Synchronization

4.4.1 The system shall automatically update payment status such as Paid or Unpaid when receiving Stripe webhook results or offline payment confirmation.

4.4.2 The system shall confirm class enrollments only when payment status is updated to “Paid”.

4.4.3 The system shall automatically mark enrollment as inactive (canceled) if the corresponding payment is removed or fails under specific business rules.

4.4.4 The system shall automatically bar student enrollment when payments remain overdue beyond a configurable threshold.

4.4.5 The system shall automatically unbar a student’s enrollment eligibility once outstanding payments are settled.

4.5 Payment Plans

4.5.1 The system shall enable direct-pay tuition fees that require full payment at enrollment.

4.5.2 The system shall support deposit-based payment plans for childcare and transport services.

4.5.3 The system shall validate whether deposits have been fully cleared before generating monthly service charges.

4.5.4 The system shall support hybrid billing models where a student may have both tuition charges and attendance-based service fees simultaneously.

4.6 Digital Receipts & Payment History

4.6.1 The system shall allow parents to view, download or print past receipts in the application.

4.6.2 The system shall clearly distinguish payment types such as enrollment fees, monthly tuition fees, service fees, deposit payments and adjustments.

5.0 Other Service Module

5.1 Service Management

5.1.1 The system shall allow administrators to view all created services with sorting, filtering, and pagination support.

5.1.2 The system shall allow staff to create new services, including transport and childcare, with customizable fee rules and availability constraints.

5.1.3 The system shall allow staff to edit service details, including naming, fee structure, billing model and deposit requirements.

5.1.4 The system shall allow administrators to deactivate services, preventing future enrollments while preserving historical records.

5.1.6 The system shall allow the configuration of multiple service billing types, including fixed monthly fees, deposit with attendance-based fees.

5.1.7 For transportation services, the system shall allow parents and staff to specify pickup and drop-off addresses during enrollment.

5.1.8 The system shall validate required fields for each service type.

5.2 Service Enrollment Management

5.2.1 The system shall allow parents to self-register their children for available services through the mobile application.

5.2.2 The system shall allow administrators to register students for services on behalf of parents.

5.2.3 The system shall validate duplicate registrations and prevent students from enrolling in the same service twice.

5.2.4 The system shall validate that the selected service is active and eligible before confirming enrollment.

5.2.5 The system shall enforce service-type–specific enrollment rules, including transport requiring valid pickup or drop-off addresses.

5.2.6 The system shall automatically deactivate service enrollments when students are barred or withdrawn.

5.3 Payment Integration

5.4.1 The system shall automatically generate pending payments when a new service enrollment is created.

5.4.2 The system shall generate monthly payments for services that operate under recurring billing models.

5.4.3 The system shall ensure accurate calculation of service-related fees using the correct billing model such as fixed monthly and deposit with attendance-based.

5.4.4 The system shall automatically cancel or adjust pending service-related payments when an enrollment is canceled or modified.

3.3.3.1.2 Non-Functional Requirements

1.0 Performance

1.1 The system shall support at least 100 concurrent users without significant degradation in response time.

1.2 The system shall respond to user actions within 2 seconds on average.

2.0 Reliability

2.1 The system shall be available 97.5% of the time during operational hours.

2.2 The system shall ensure that critical data (e.g., payments, enrollments) is never lost and always recoverable.

3.0 Security

3.1 The system shall require authenticated login for all users.

3.2 The system shall encrypt sensitive data in storage and transmission.

3.3 The system shall provide role-based access control to protect restricted features and data.

4.0 Usability

4.1 The system shall provide a user-friendly interface accessible to staff, parents, and tutors.

4.2 The system shall provide help documentation and tooltips for important actions.

5.0 Maintainability

5.1 The system shall be modular and follow coding best practices to support future updates.

Organization

- The system shall be developed using Python for backend APIs and React Native (JavaScript/TypeScript) for the mobile application.
- The system shall use XAMPP with phpMyAdmin for managing the MySQL database and backend services.
- Version control shall be managed via Git and hosted on GitHub.
- UI/UX designs shall be created and iterated using Figma.

External

- The app shall support online payments processed using Stripe in Malaysian Ringgit (MYR).
- All user data handling shall comply with local privacy regulations (e.g., Malaysia PDPA) and follow best practices for security and data protection using XAMPP and phpMyAdmin.

3.4 Development Environment

3.4.1 System Architecture (Client-Server Environment)

EazeTuition utilizes a **client-server architecture**, where the backend of the system is hosted on a local or remote server and accessed by different types of clients:

- **Web-based administration panel**
  Used by administrators and staff to manage subjects, schedules, users, expenses and other services. Built with HTML, CSS, jQuery and Jinja2 and AJAX for seamless interaction.

- **Mobile application (React Native)**
  Designed for parents and counselors to view schedules, make payments, communicate and manage course registration. Built with React Native and tested with Expo Go.

This architecture clearly separates the business logic (server) from the user interface (clients) and supports multi-user concurrent access. All client interactions with the server occur via RESTful API endpoints built using Flask (Hutri).

3.4.2 Software Environment

Software Component	Description
Backend Framework	Python (Flask) for API development and system logic.
Database	MySQL (phpMyAdmin via XAMPP) for storing users, subjects, classes, payment histories, etc.
Frontend (Admin)	HTML, CSS, jQuery, AJAX
Mobile App	React Native with Expo for development and testing
Version Control	Git & GitHub for collaborative code management and backup
Payment Integration	Stripe API for handling online fee payments in MYR
IDE/Editors	Visual Studio Code for both backend and mobile development
API Testing	Postman for testing backend API endpoints

Figure 3.4.2.1 Software Environment Specification

3.4.3 Hardware Environment

Developer Machine

Component	Specification
Processor	Intel Core i5 / Ryzen 5 or higher
RAM	8GB minimum (16 GB recommended)
Storage	256 GB SSD minimum
OS	Windows 10/11 or Ubuntu 20.04+
Internet	Stable connection for Git and API usage

Figure 3.4.3.1 Developer Machine Specification

Server Requirement (For Local Development)

Component	Specification
Server Type	Localhost during dev
Processor	Dual-core minimum
RAM	2 GB minimum
OS	XAMPP for local

Figure 3.4.3.1 Server Requirement of Hardware Environment

Mobile Testing Environment

Tool	Description
Expo Go App	Allows real-time mobile testing without building APK
Devices USed	Android Phone and Windows PC
OS Support	Android 9.0+

Figure 3.4.3.3 Mobile Testing Environment Specification

3.5 Chapter Summary and Evaluation

This chapter summarizes the core planning and analysis activities for the EazeTuition system. An incremental development model was first selected to support modularization and phased development. Requirements were gathered through interviews, observations, and document reviews.

A set of functional and non-functional requirements were identified, and use case diagrams and descriptions were created for all key modules including subjects, course schedules, payments, chat, and other services. These provided guidance on the scope and interaction design of the system.

Finally, the development environment is described in detail, including the system architecture, tools, frameworks, and hardware for web and mobile platforms.

In summary, this chapter ensures that the system was built based on a clear plan, defined requirements, appropriate methodology, and a structured technical setup.

Chapter 4

System Design

4 System Design

This chapter focuses on the system design of EazeTuition, outlining the structural and behavioral models guiding its implementation which include sequence diagrams, statecharts, class diagrams, entity-relationship diagrams, a data dictionary and flow charts. It also showcases the user interface design and report layout to visualize the interaction process, details the software architecture through deployment diagrams and introduces artificial intelligence algorithms to support intelligent timetable evaluation.

4.1 Sequence Diagram

4.1.1 Subject Module

View Subject List

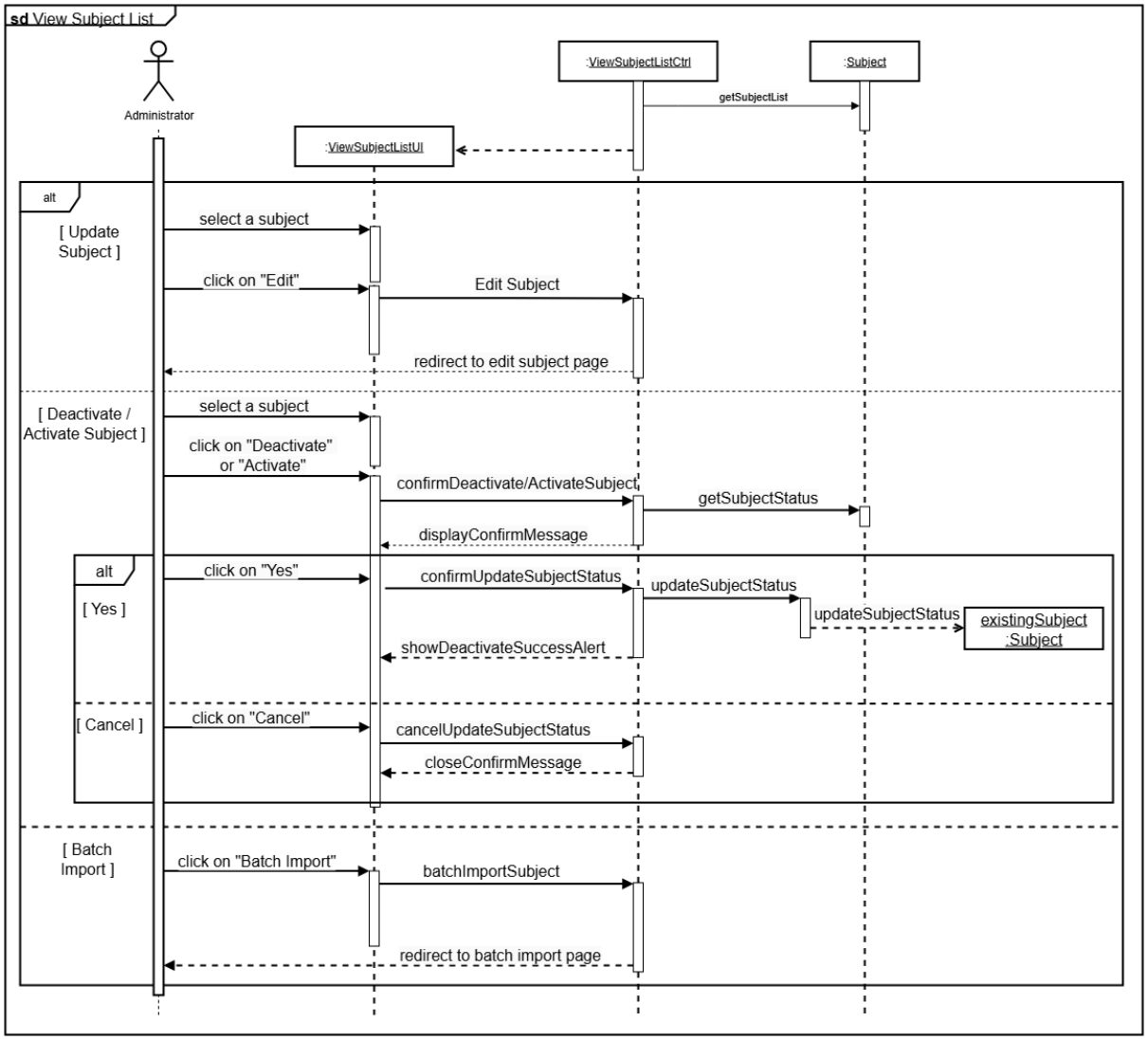


Figure 4.1.1.1 Sequence Diagram for View Subject List

Add New Subject

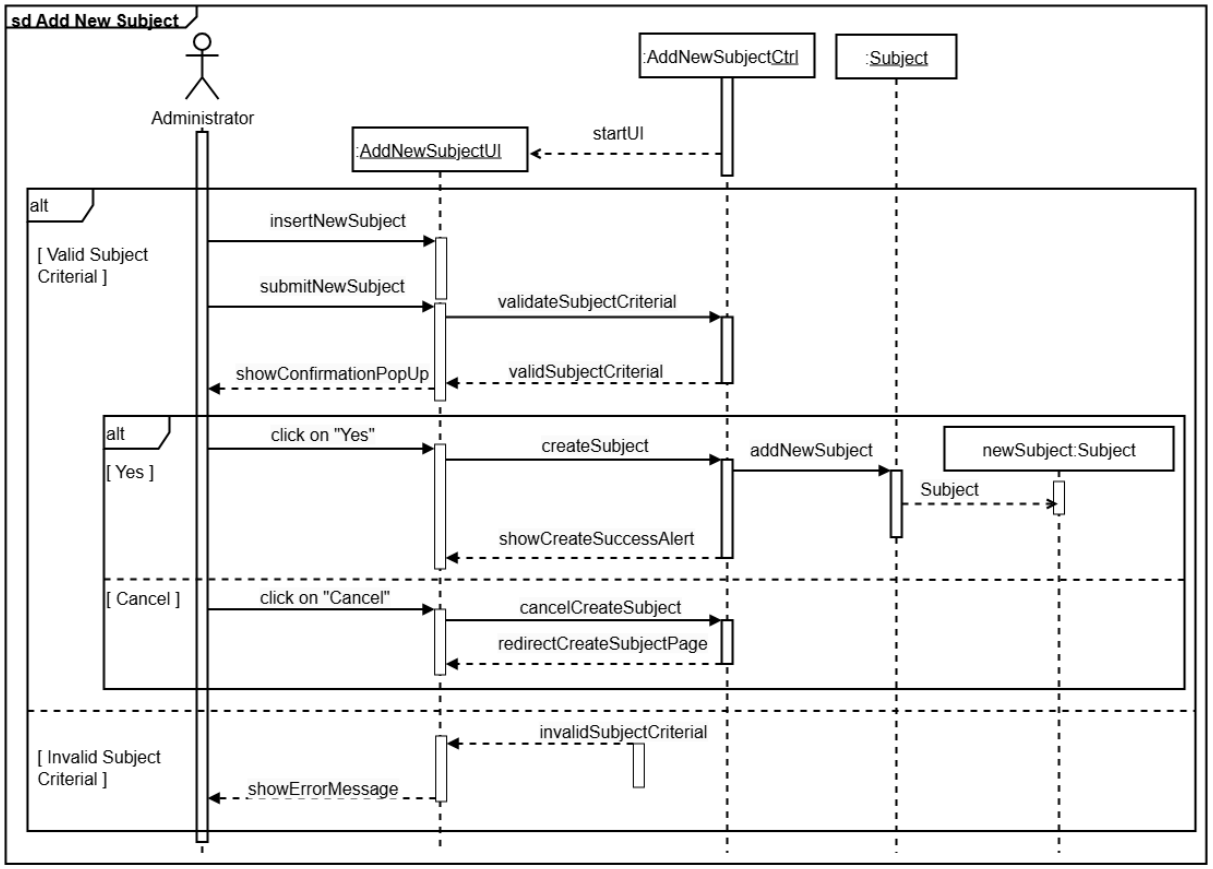


Figure 4.1.1.2 Sequence Diagram for Add New Subject

Update Subject

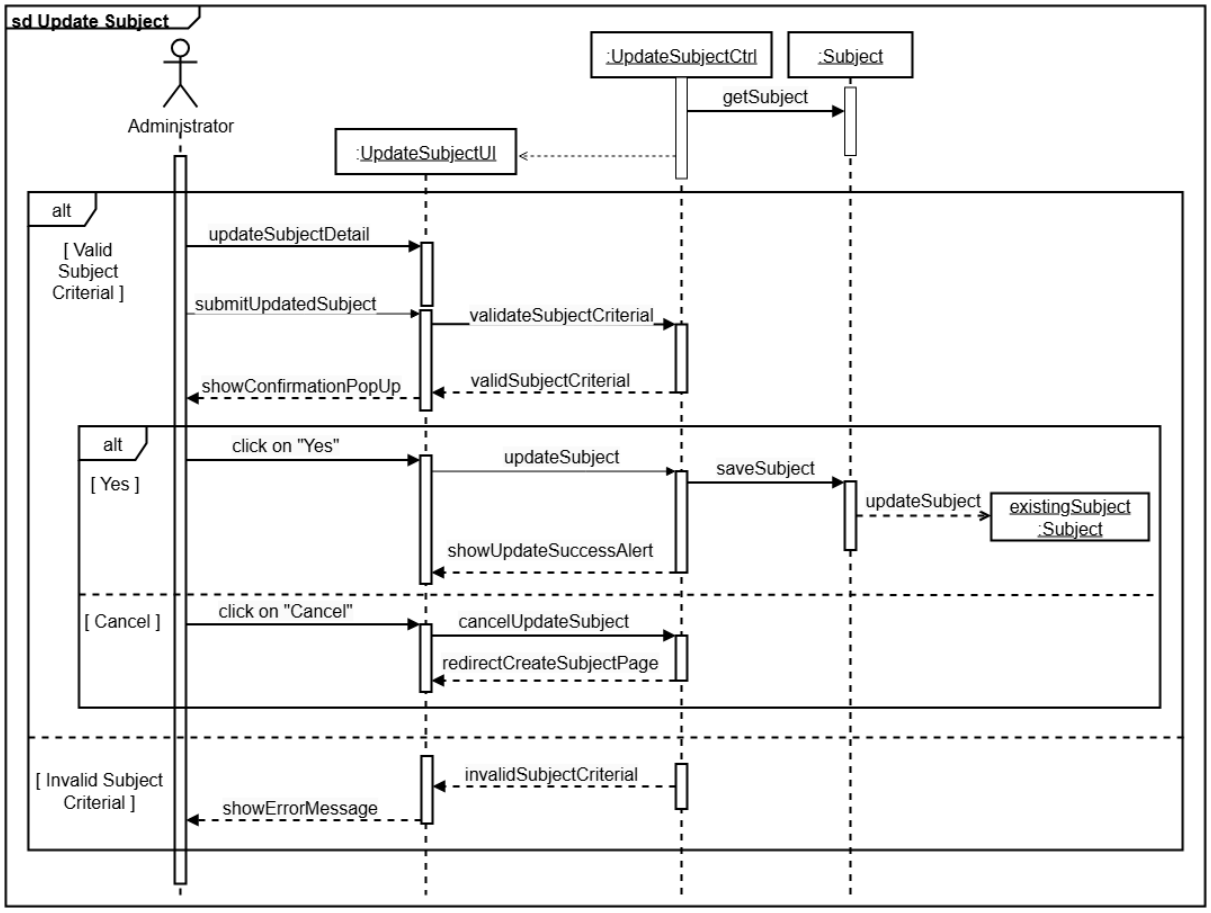


Figure 4.1.1.3 Sequence Diagram for Update Subject

Deactivate / Reactivate Subject

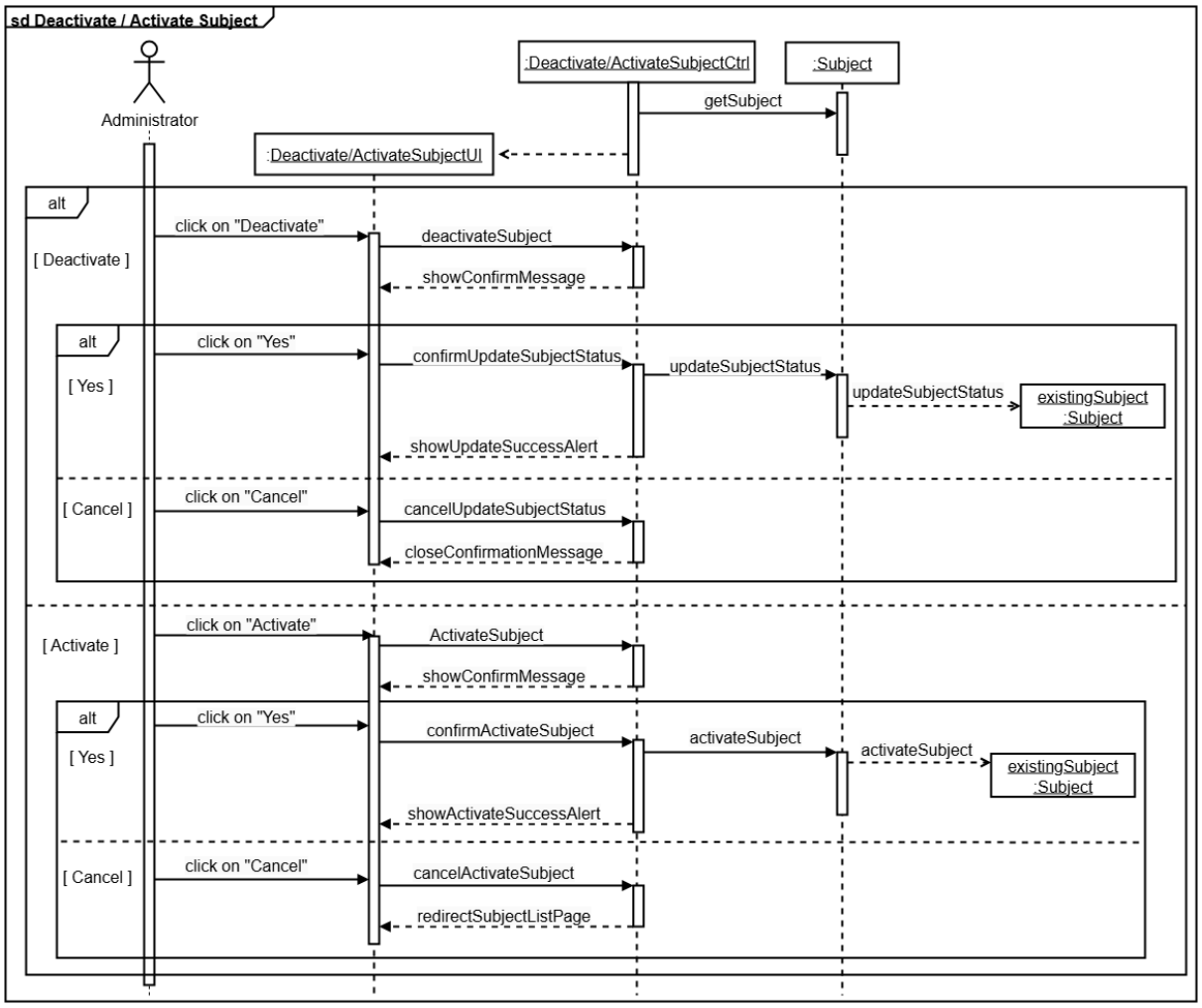


Figure 4.1.1.4 Sequence Diagram for Deactivate / Reactivate Subject

Batch Import Subject

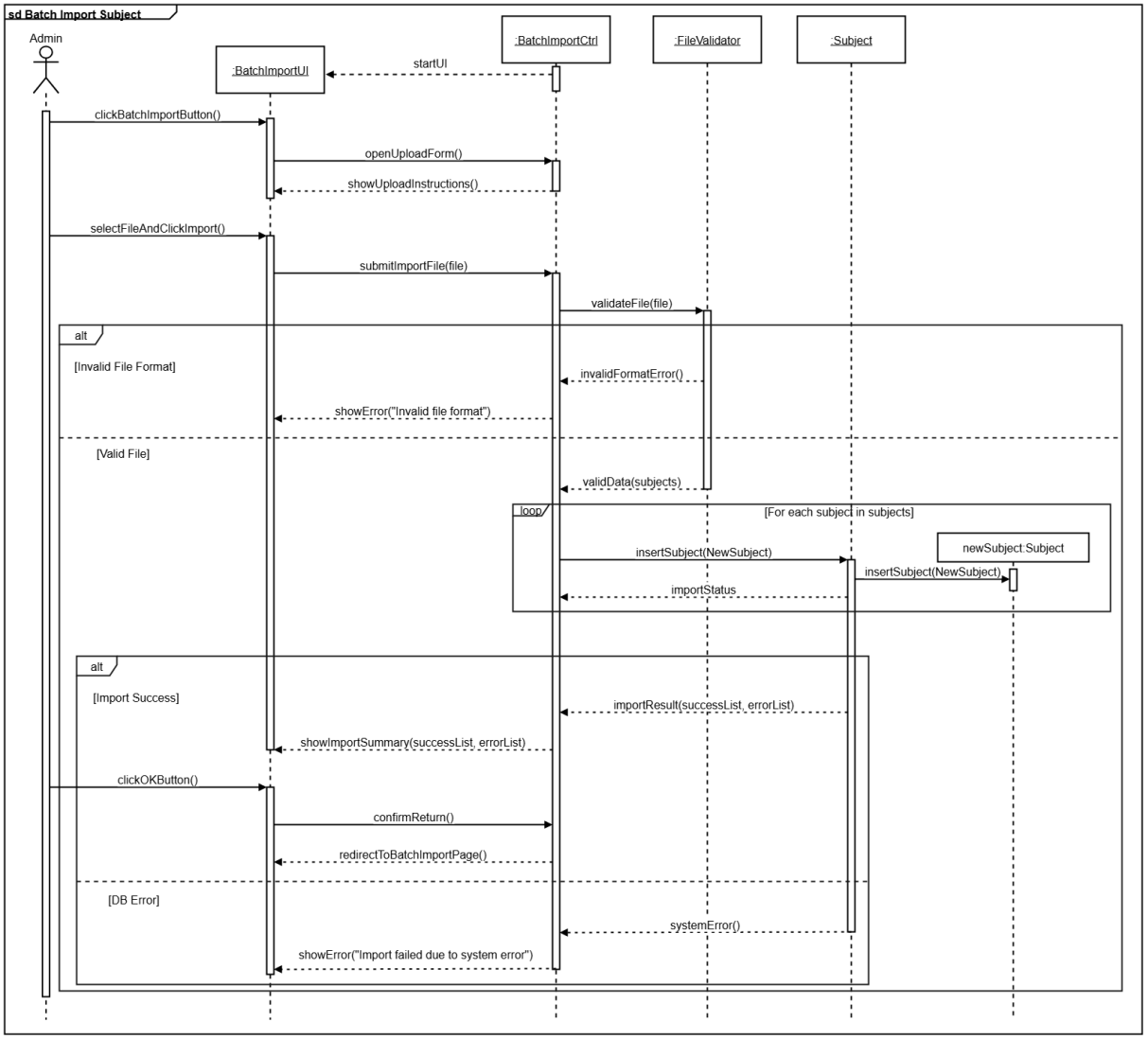


Figure 4.1.1.5 Sequence Diagram for Batch Import Subject

4.1.2 Class Module

View Class List

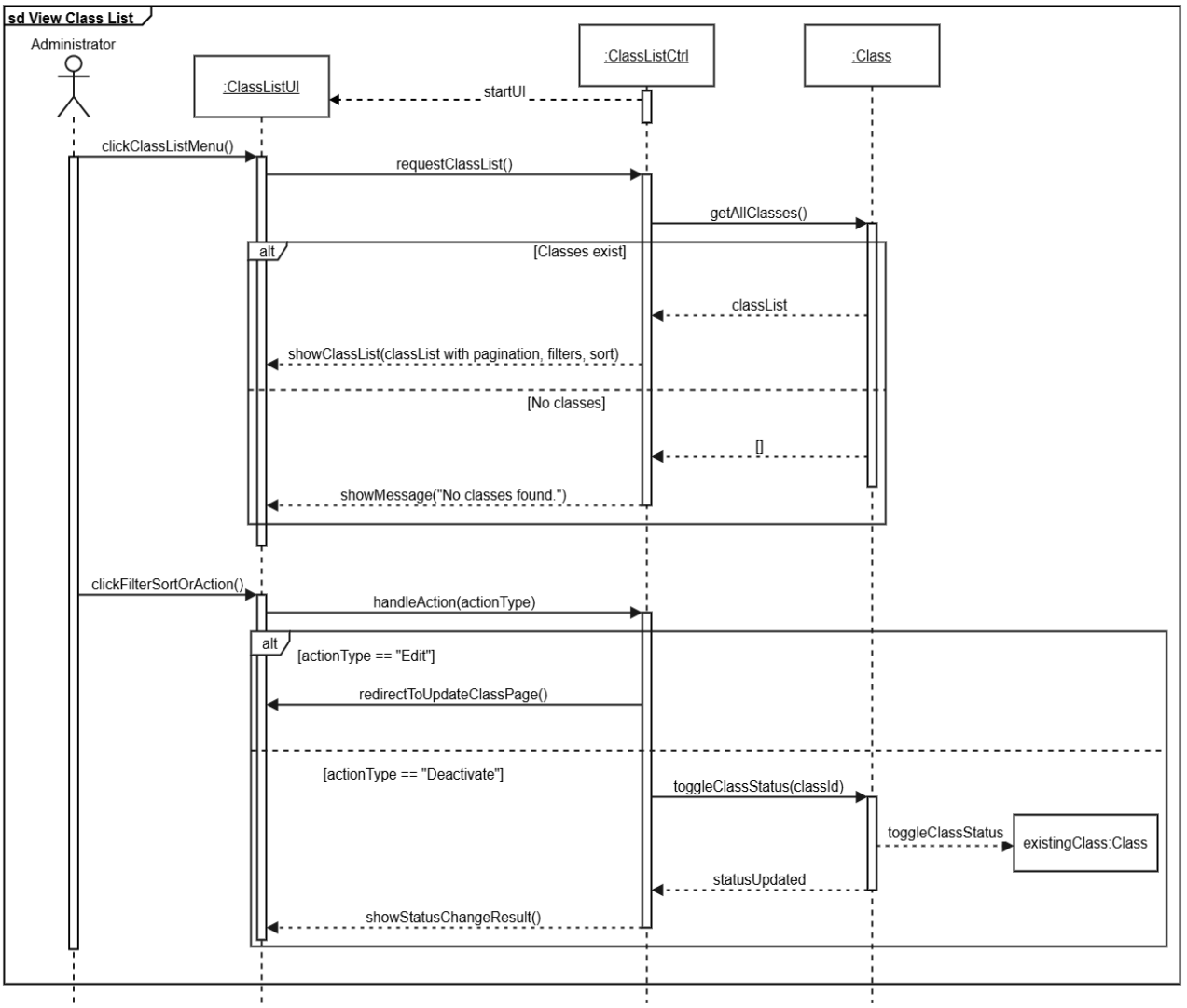


Figure 4.1.2.1 Sequence Diagram for View Class List

Create Class

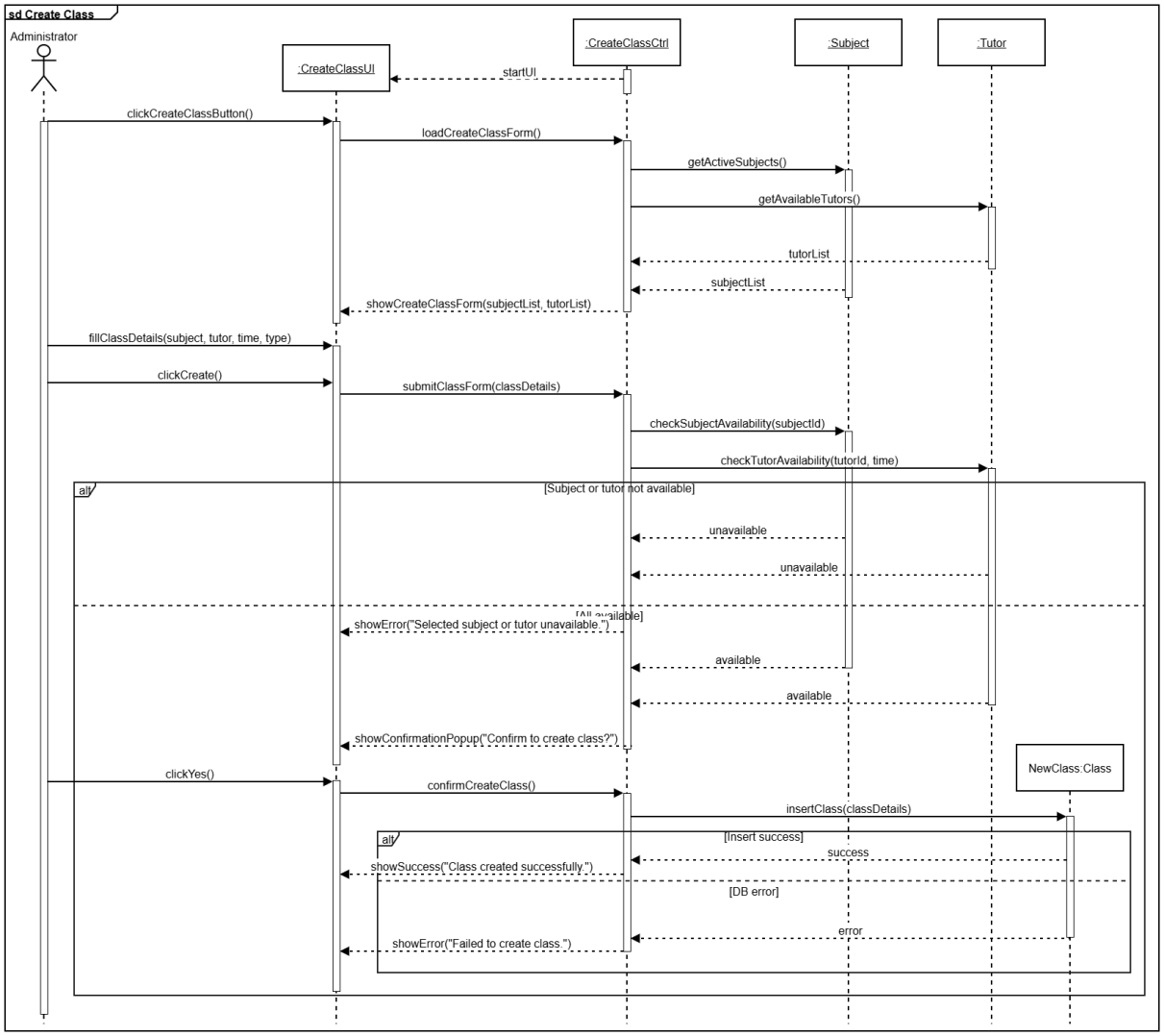


Figure 4.1.2.2 Sequence Diagram for Create Class

Update Class

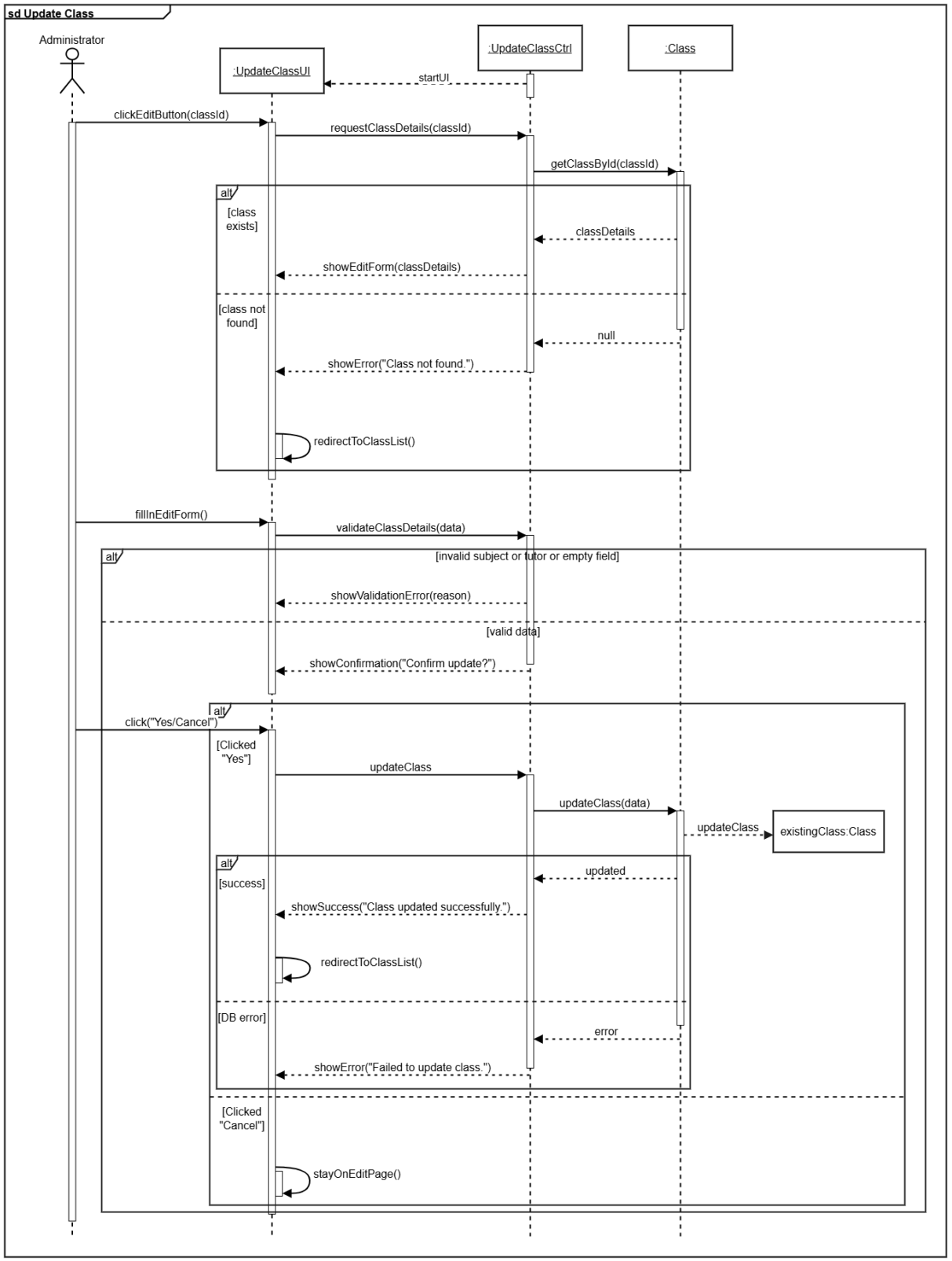


Figure 4.1.2.3 Sequence Diagram for Update Class

View Enrolled Class

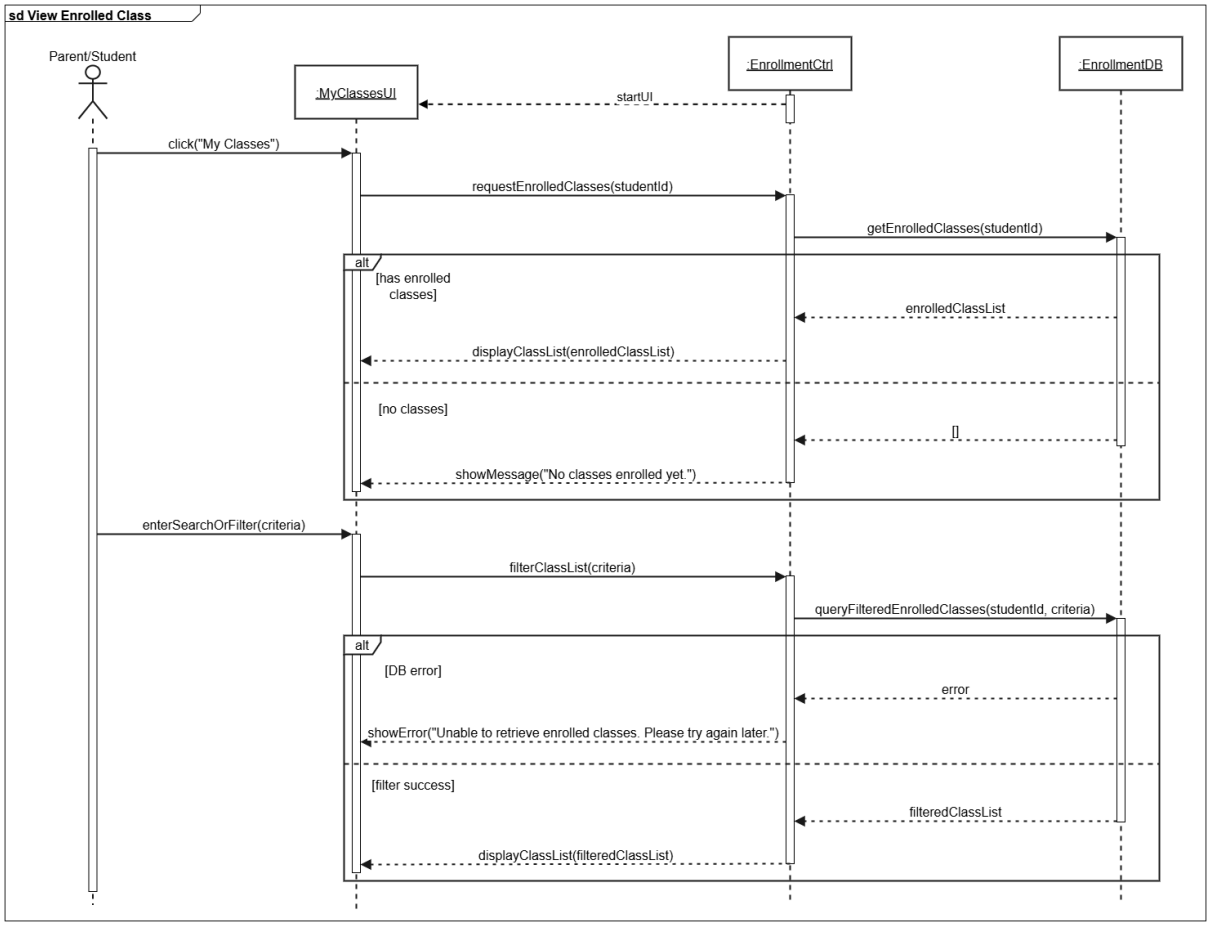


Figure 4.1.2.4 Sequence Diagram for View Enrolled Class

Enroll In Multiple Class

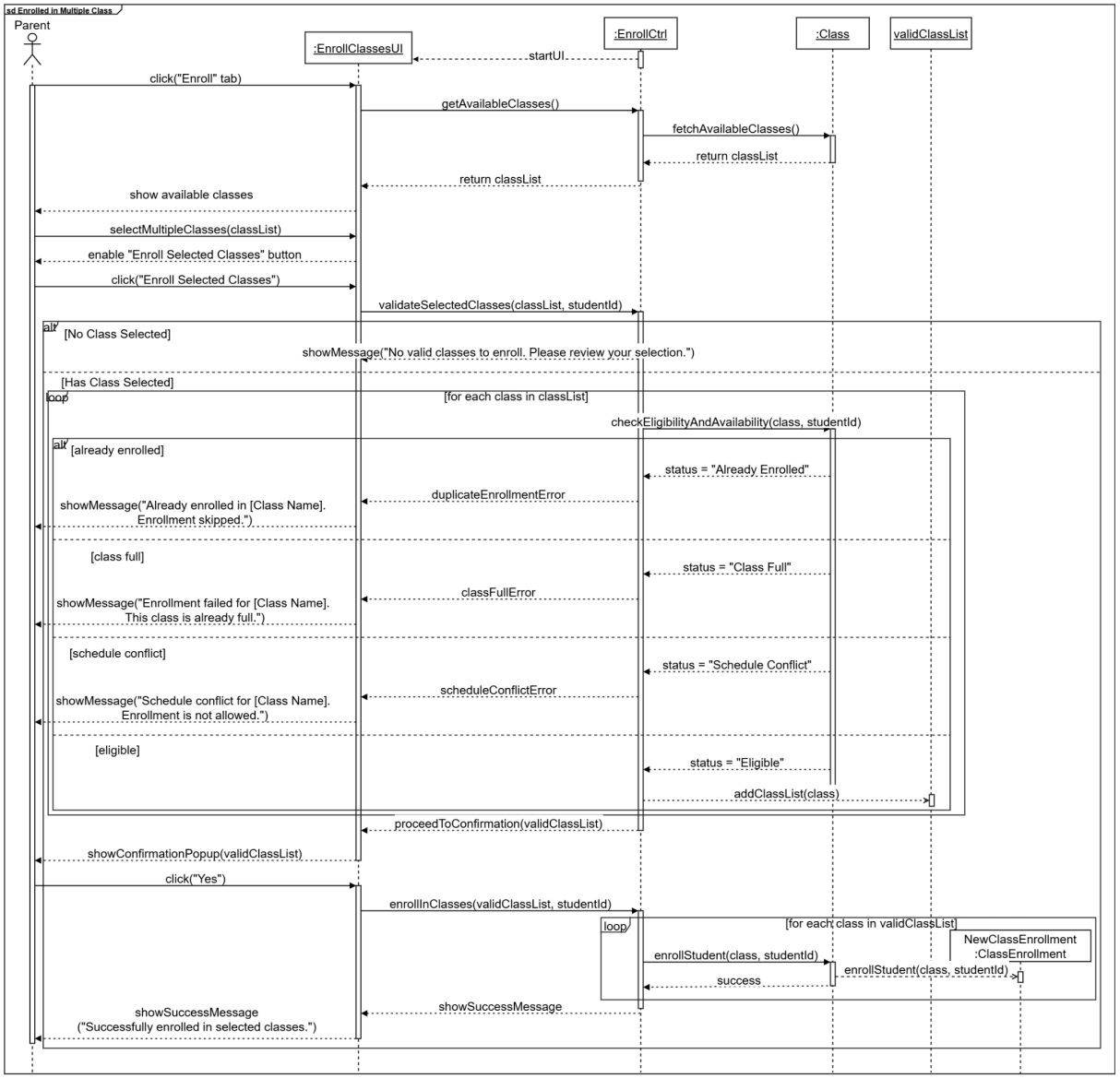


Figure 4.1.2.5 Sequence Diagram for Enrolled In Multiple Class

Generate Class Schedule

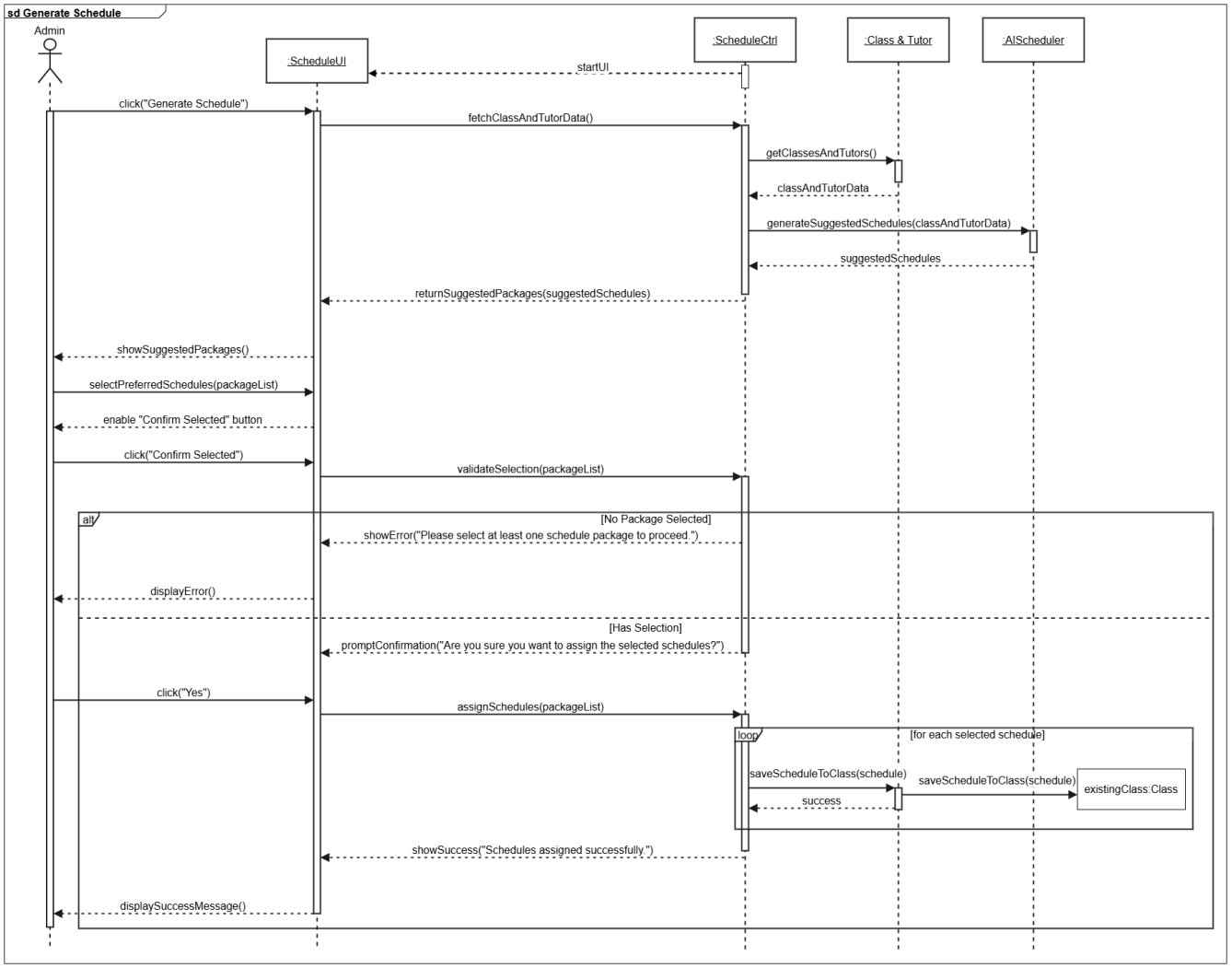


Figure 4.1.2.6 Sequence Diagram for Generate Class Schedule

Update Class Session Status

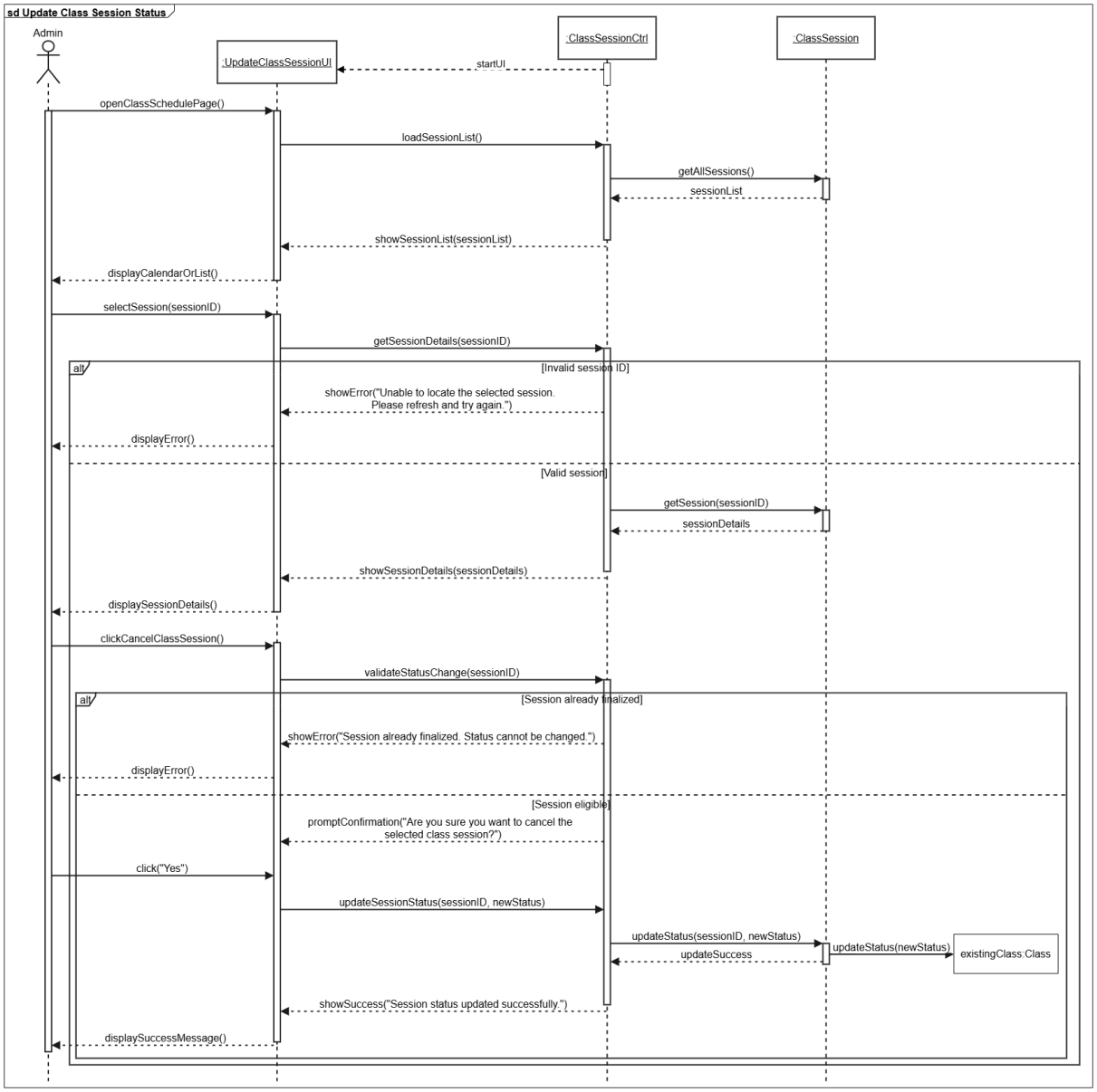


Figure 4.1.2.7 Sequence Diagram for Update Class Session Status

View Class Schedule

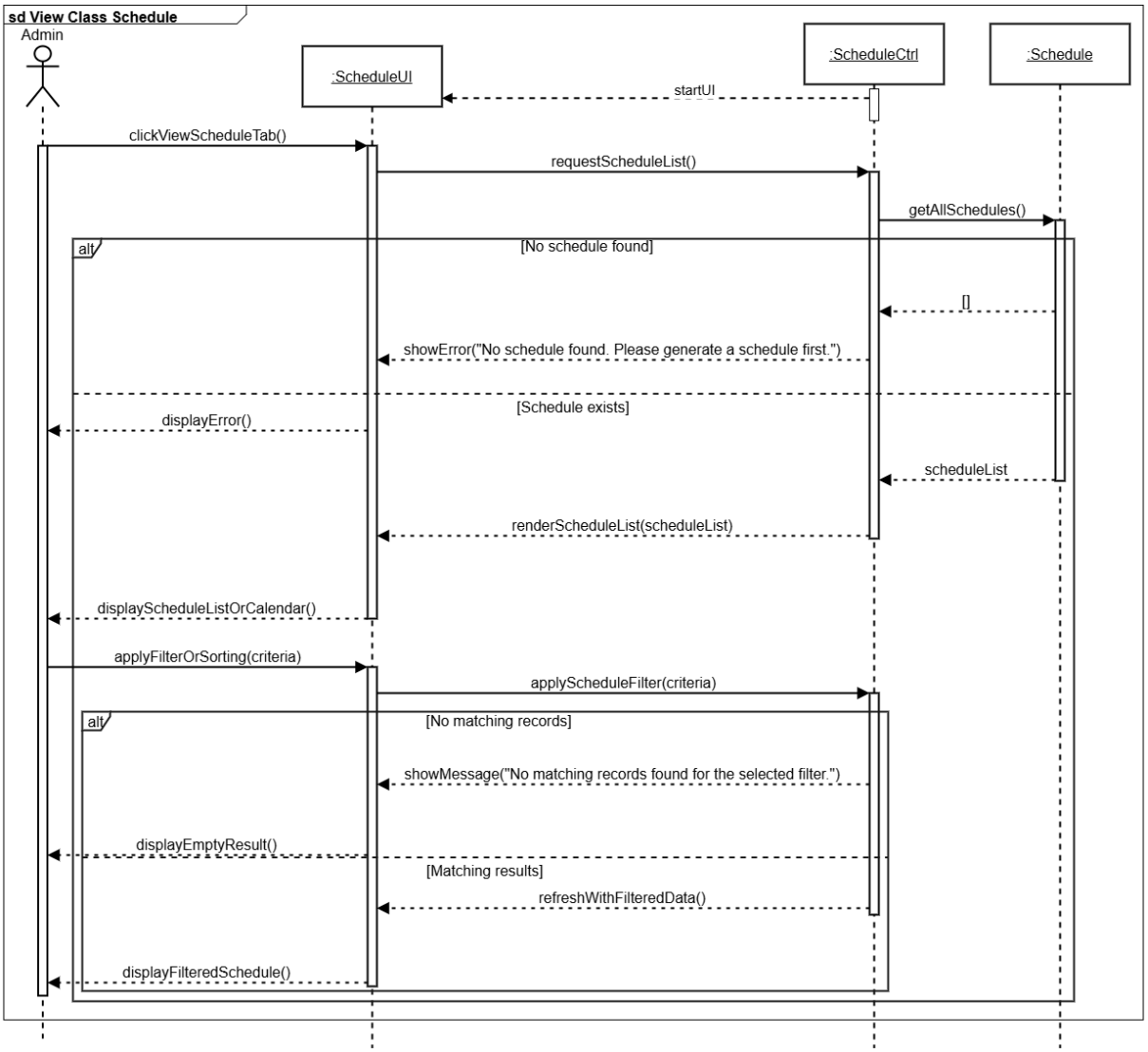


Figure 4.1.2.8 Sequence Diagram for View Class Schedule

View Personal Schedule

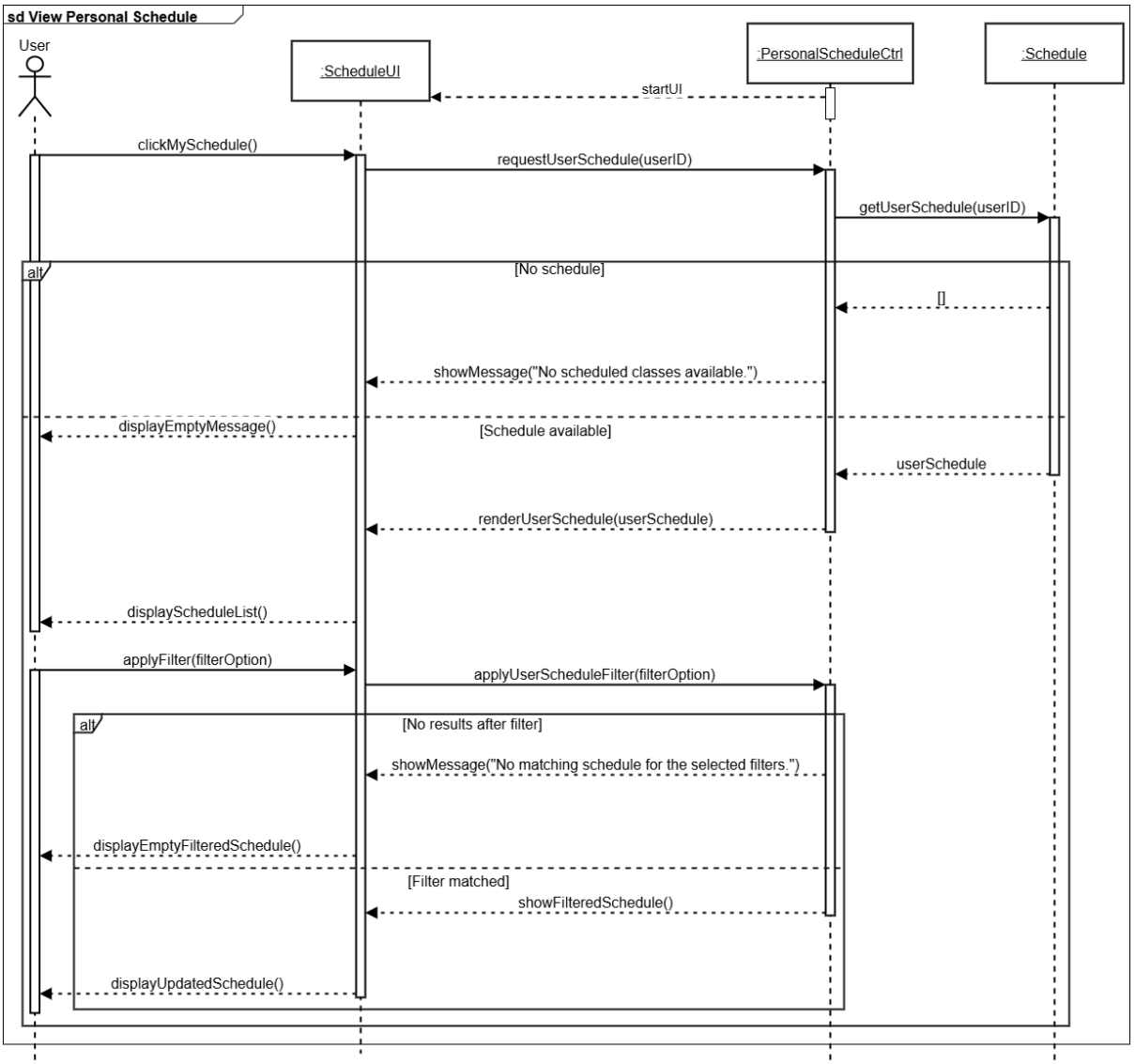


Figure 4.1.2.9 Sequence Diagram for View Personal Schedule

4.1.3 Payment Module

Generate Invoice

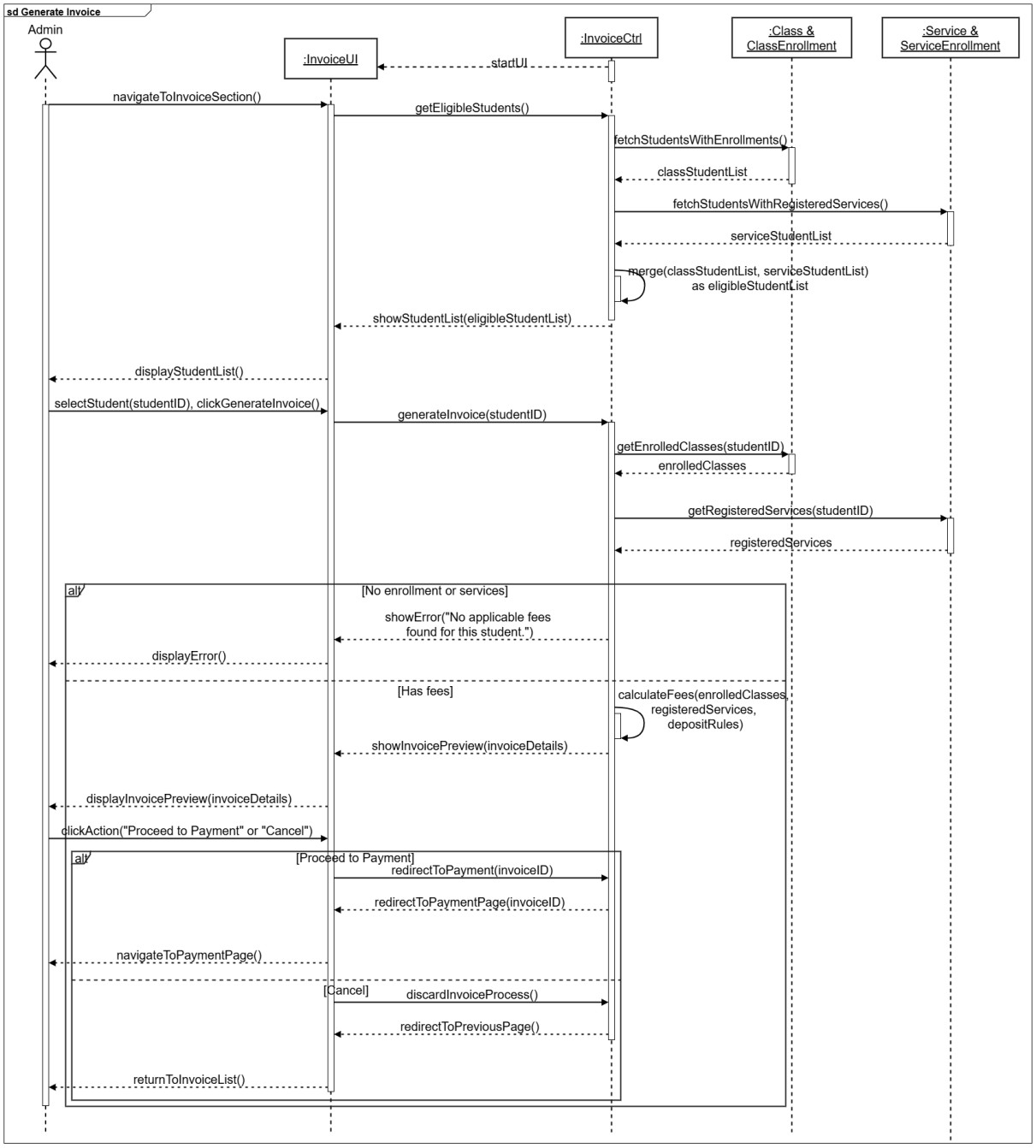


Figure 4.1.3.1 Sequence Diagram for Generate Invoice

Record Offline Payment

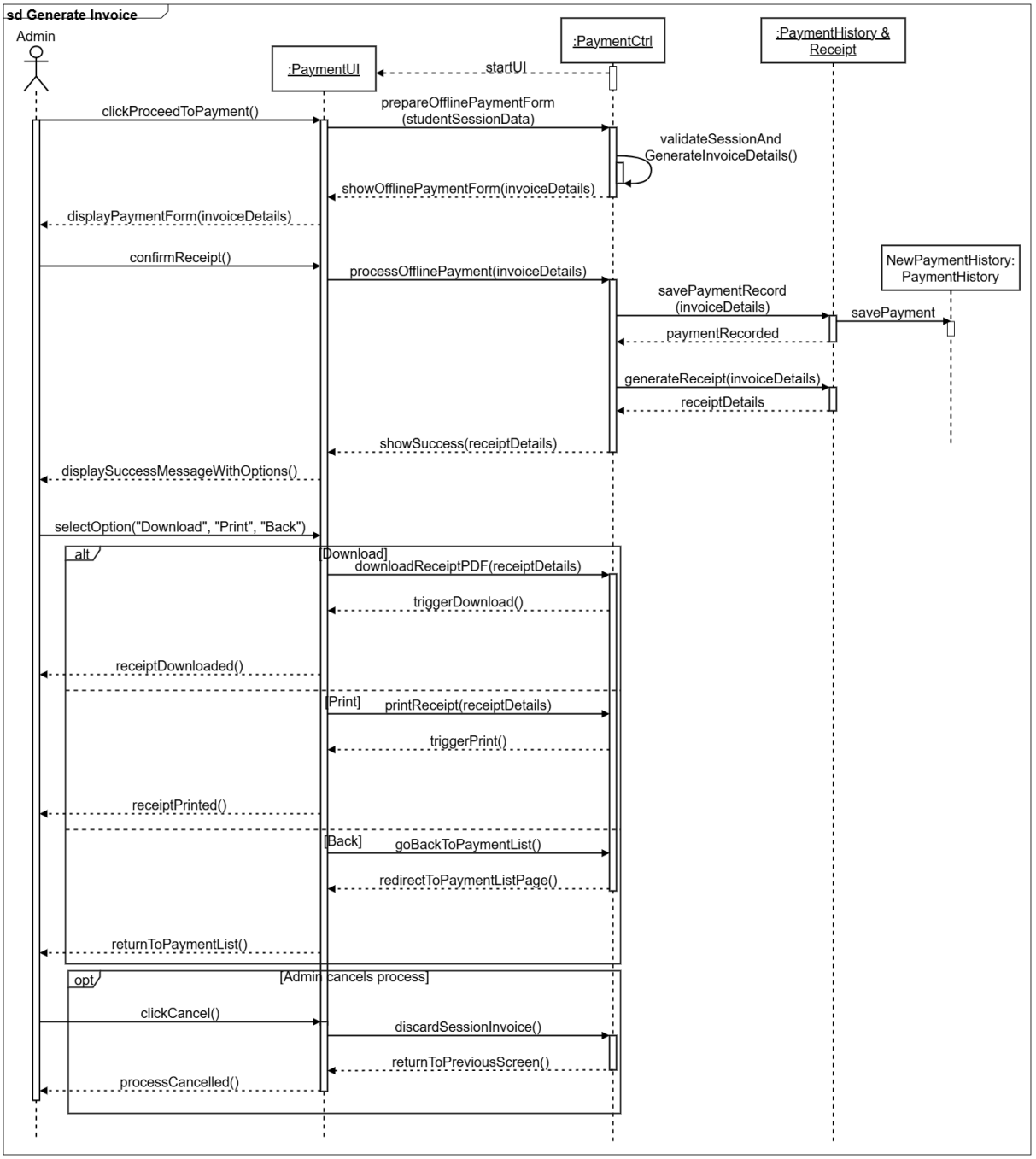


Figure 4.1.3.2 Sequence Diagram for Record Offline Payment

View Payment History

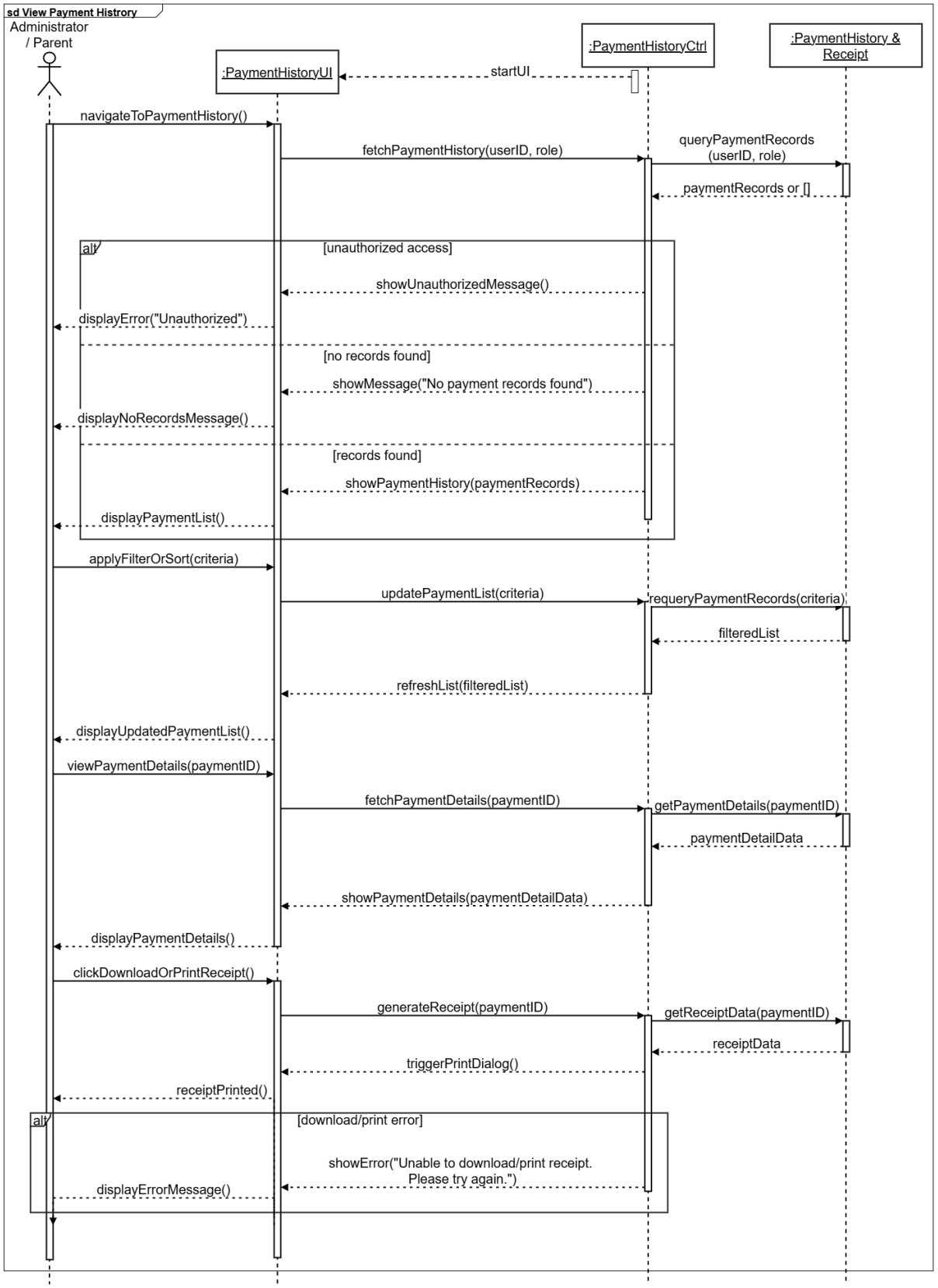


Figure 4.1.3.3 Sequence Diagram for View Payment History (Part 1)

Make Online Payment

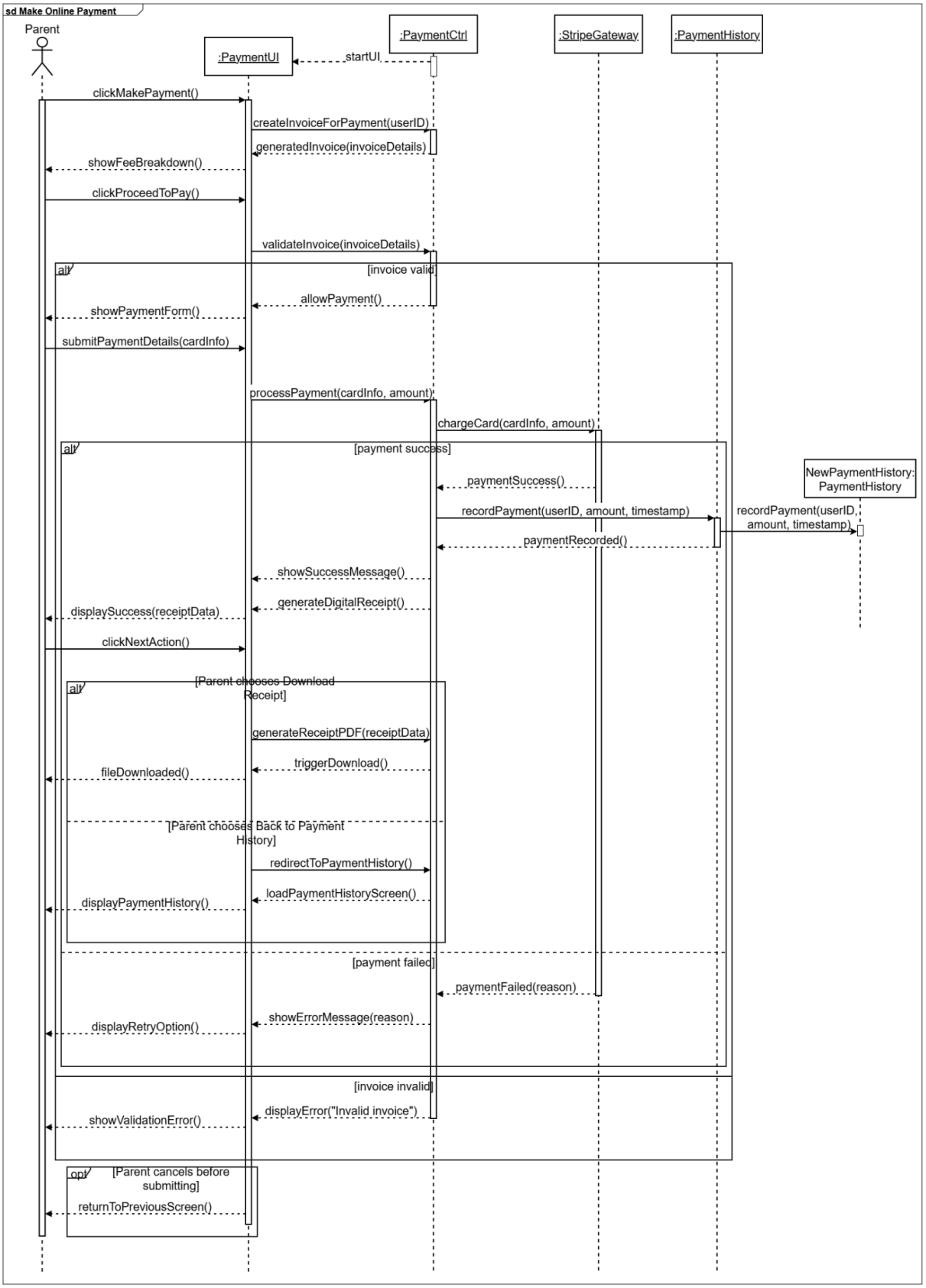


Figure 4.1.3.4 Sequence Diagram for Make Online Payment

4.1.4 Other Service Module

View Other Service List

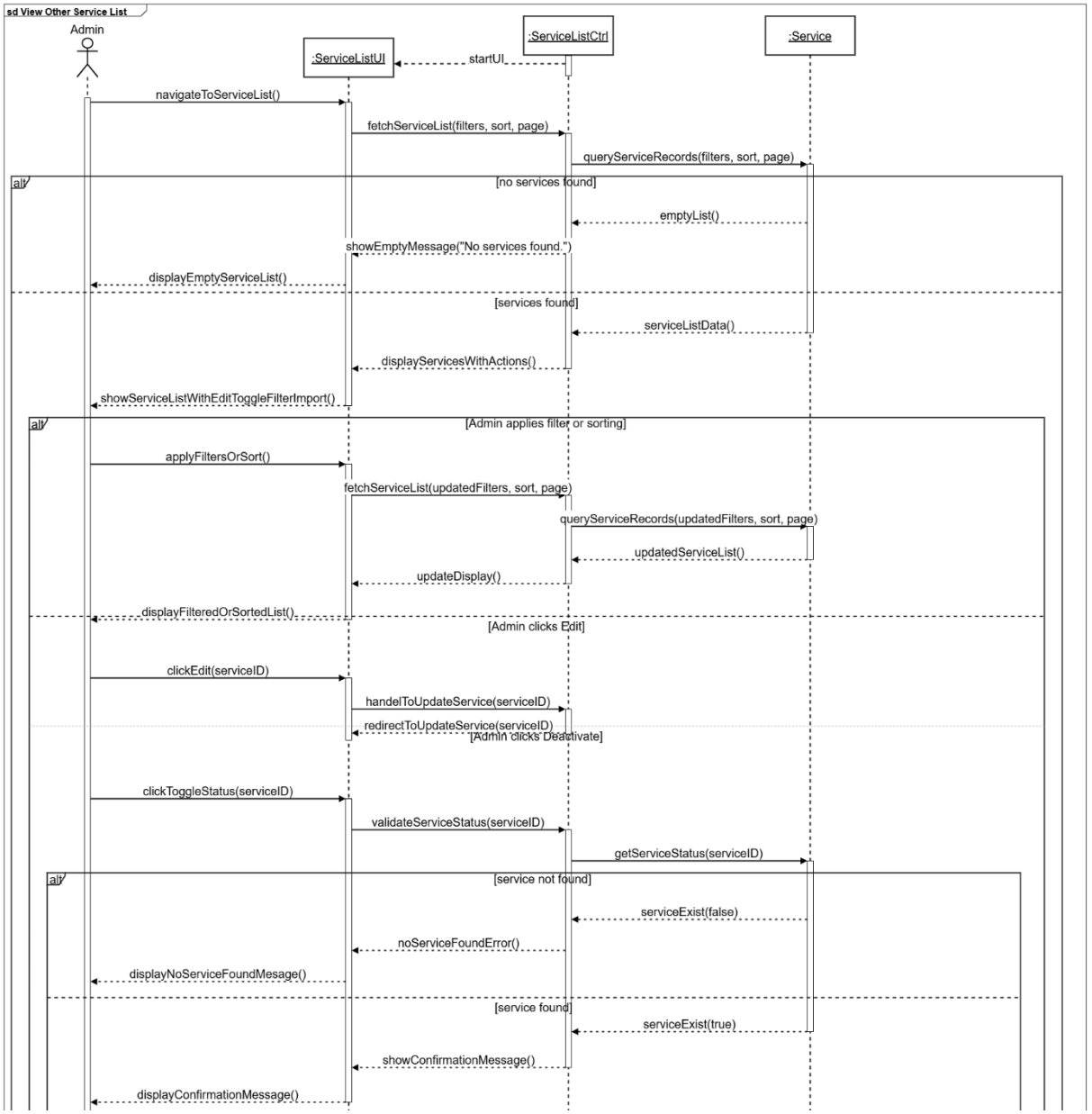


Figure 4.1.4.1.1 Sequence Diagram for View Other Service List (Part 1)

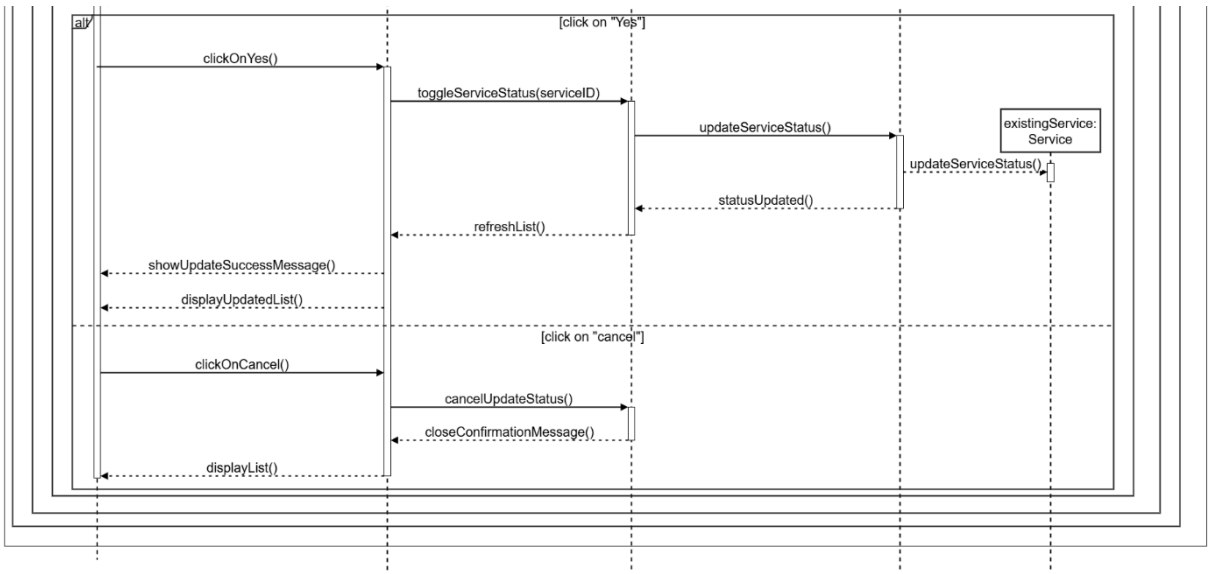


Figure 4.1.4.1.2 Sequence Diagram for View Other Service List (Part 2)

Create Other Service

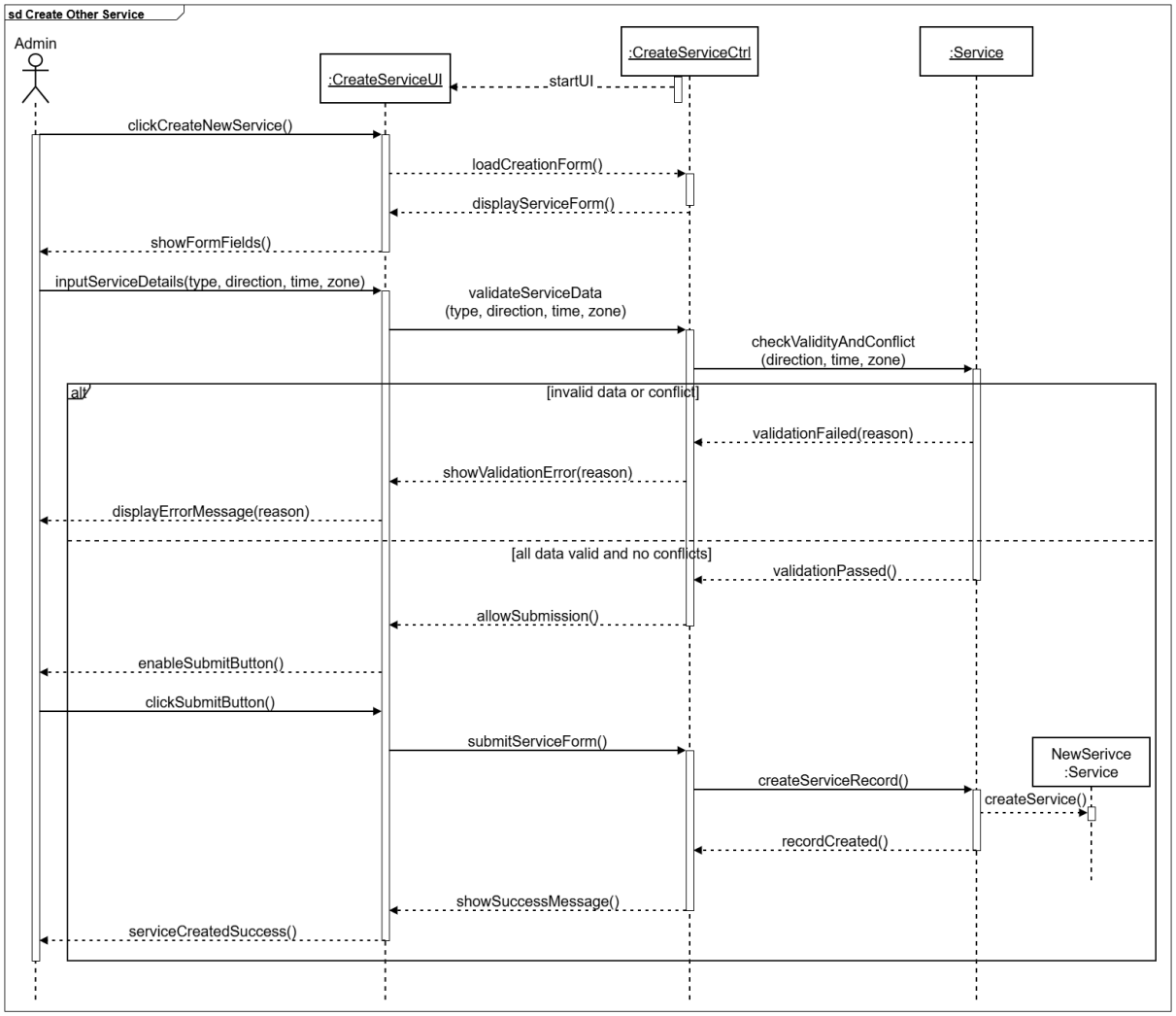


Figure 4.1.4.2 Sequence Diagram for Create Other Service

Update Other Service

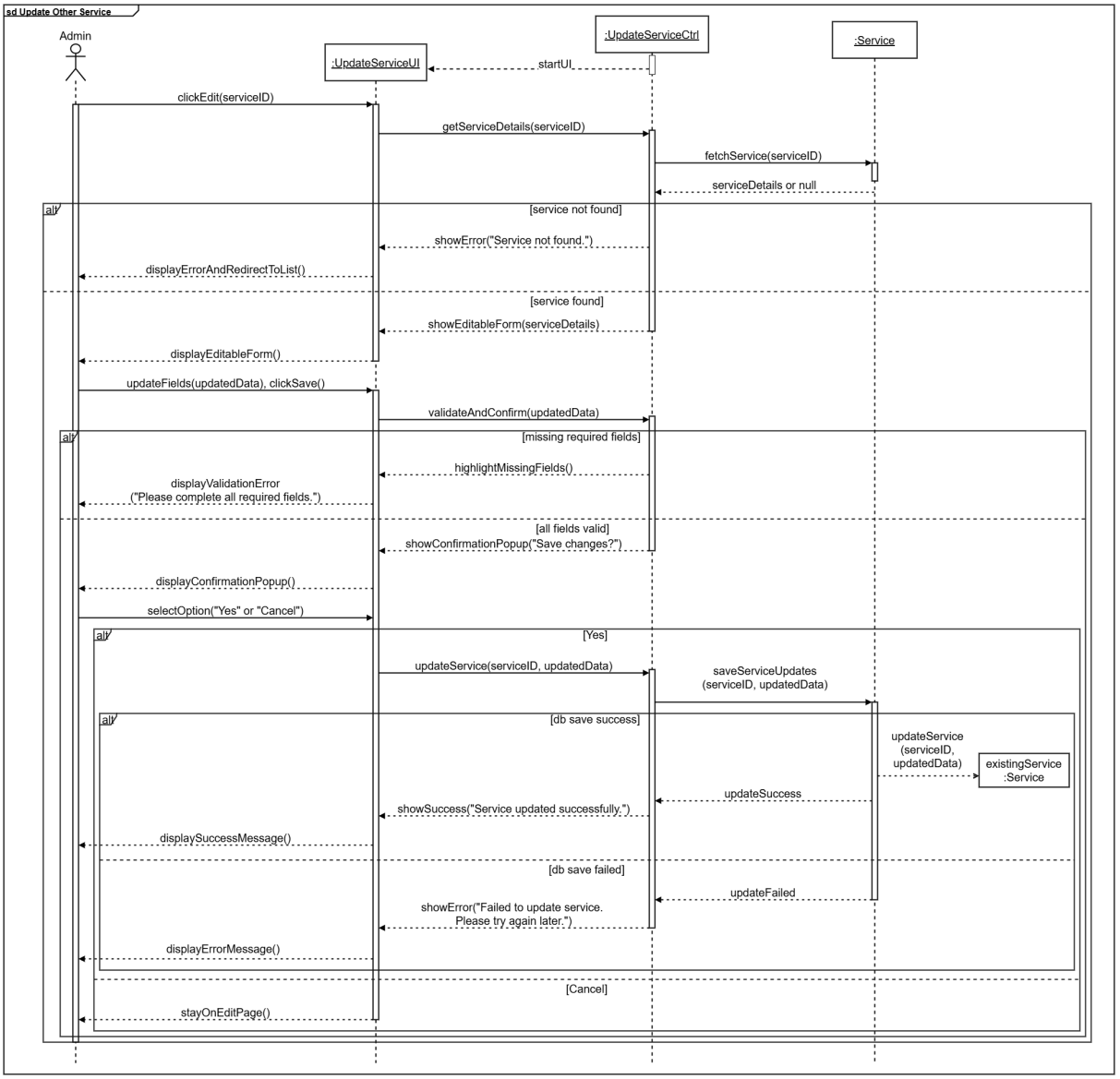


Figure 4.1.4.3 Sequence Diagram for Update Other Service

Cancel Service Enrollment

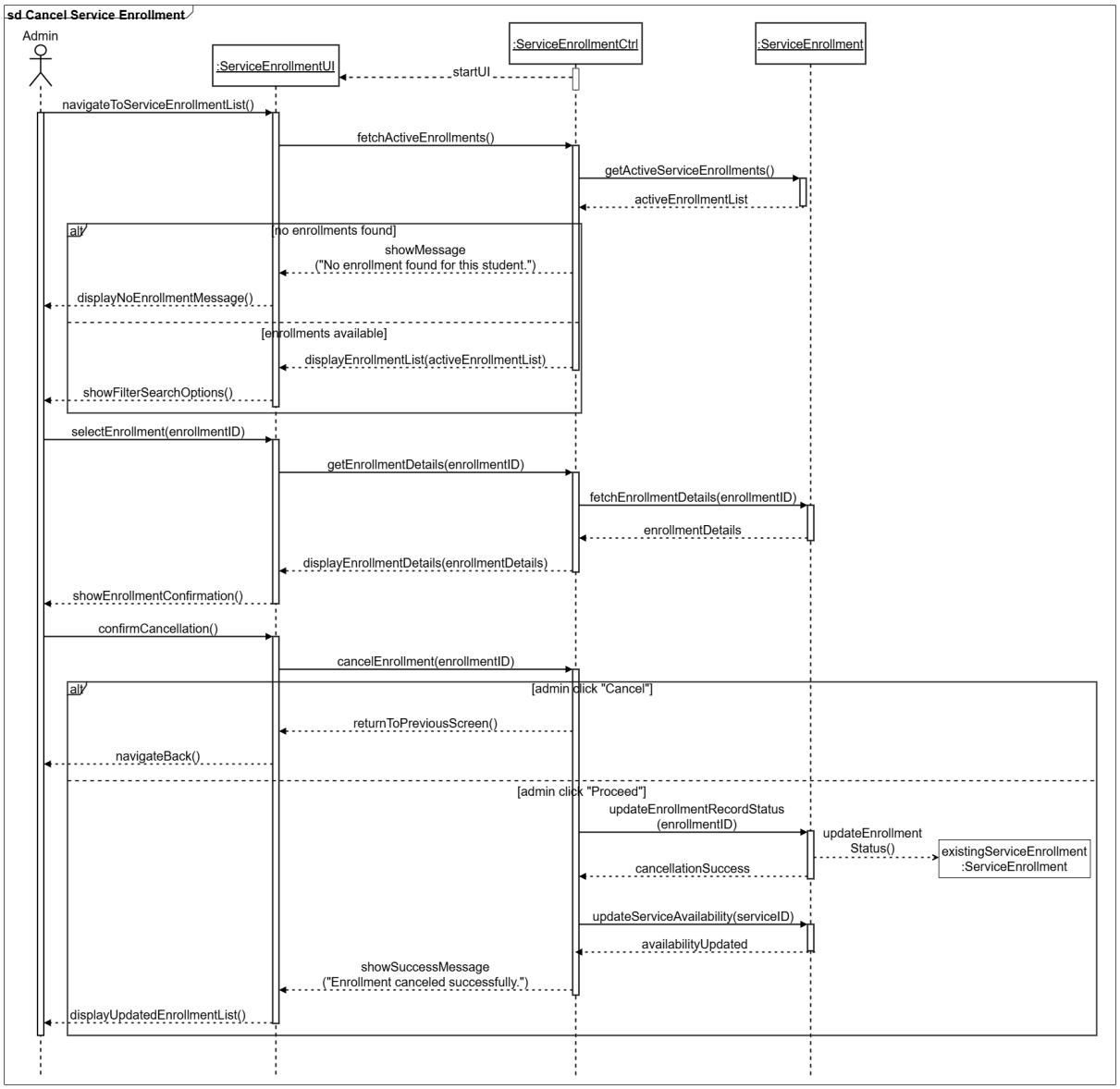


Figure 4.1.4.4 Sequence Diagram for Cancel Service Enrollment

Register Other Service

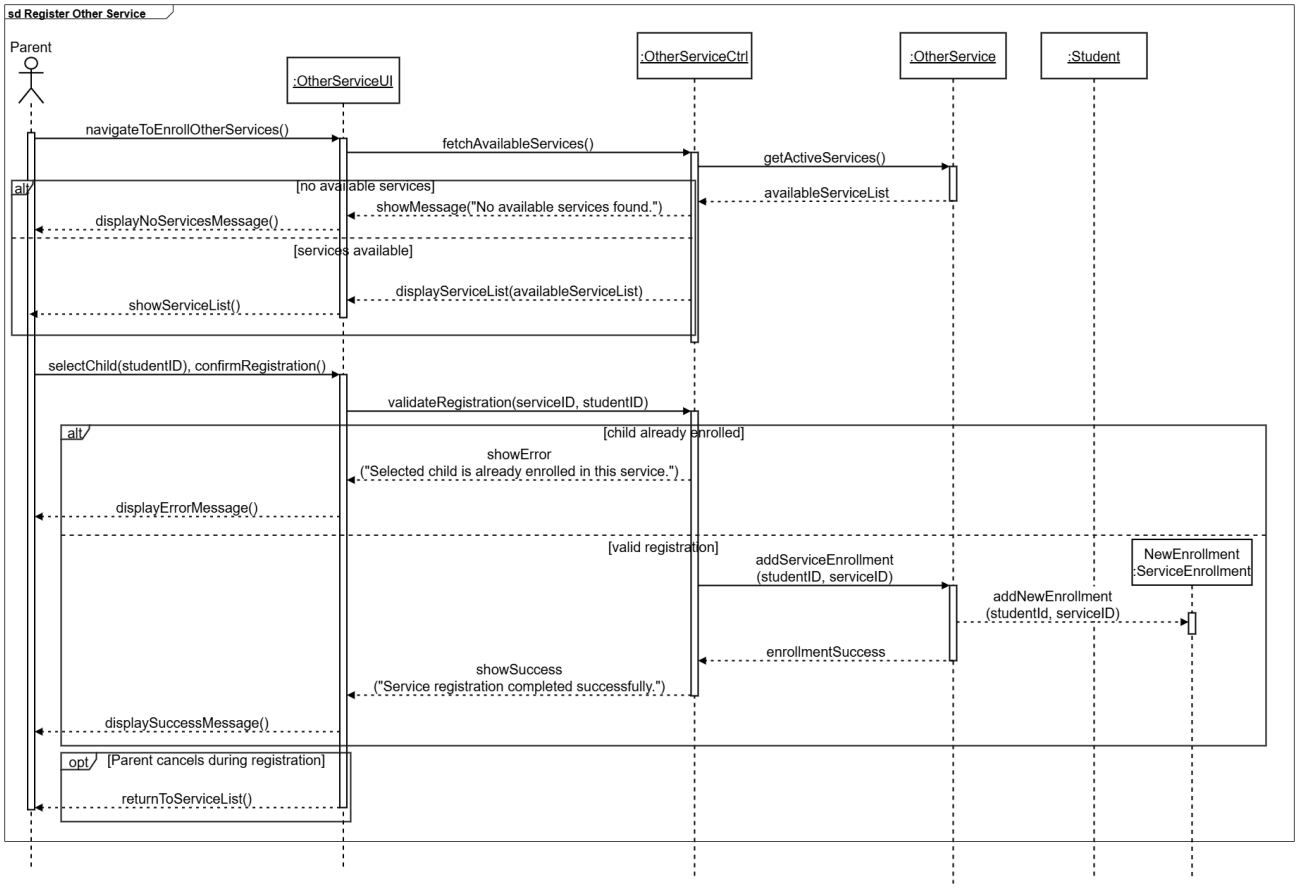


Figure 4.1.4.5 Sequence Diagram for Register Other Service

4.1.5 Chat Module

Create Chat Room

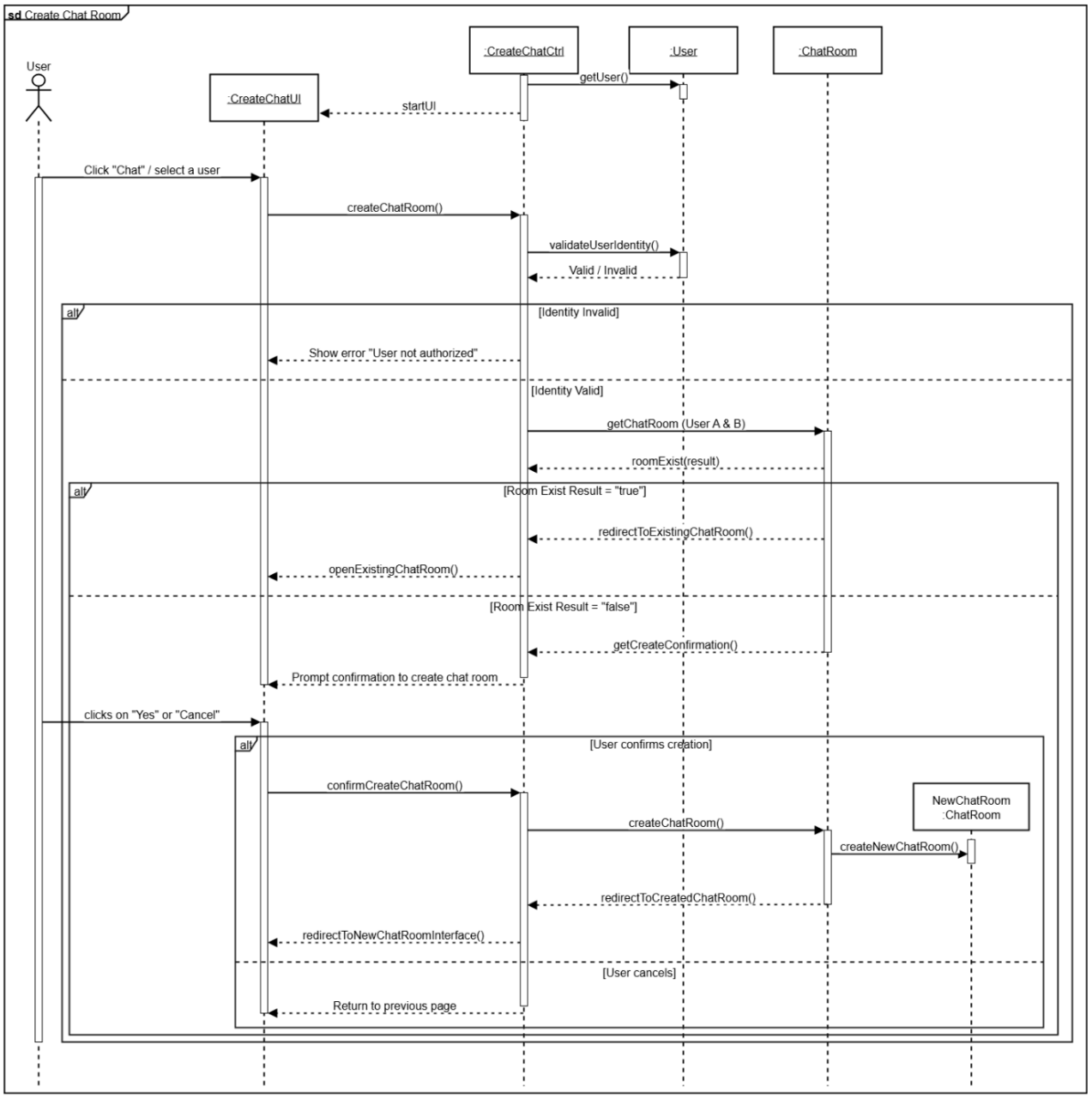


Figure 4.1.5.1 Sequence Diagram of Create Chat Room

Exchange Message

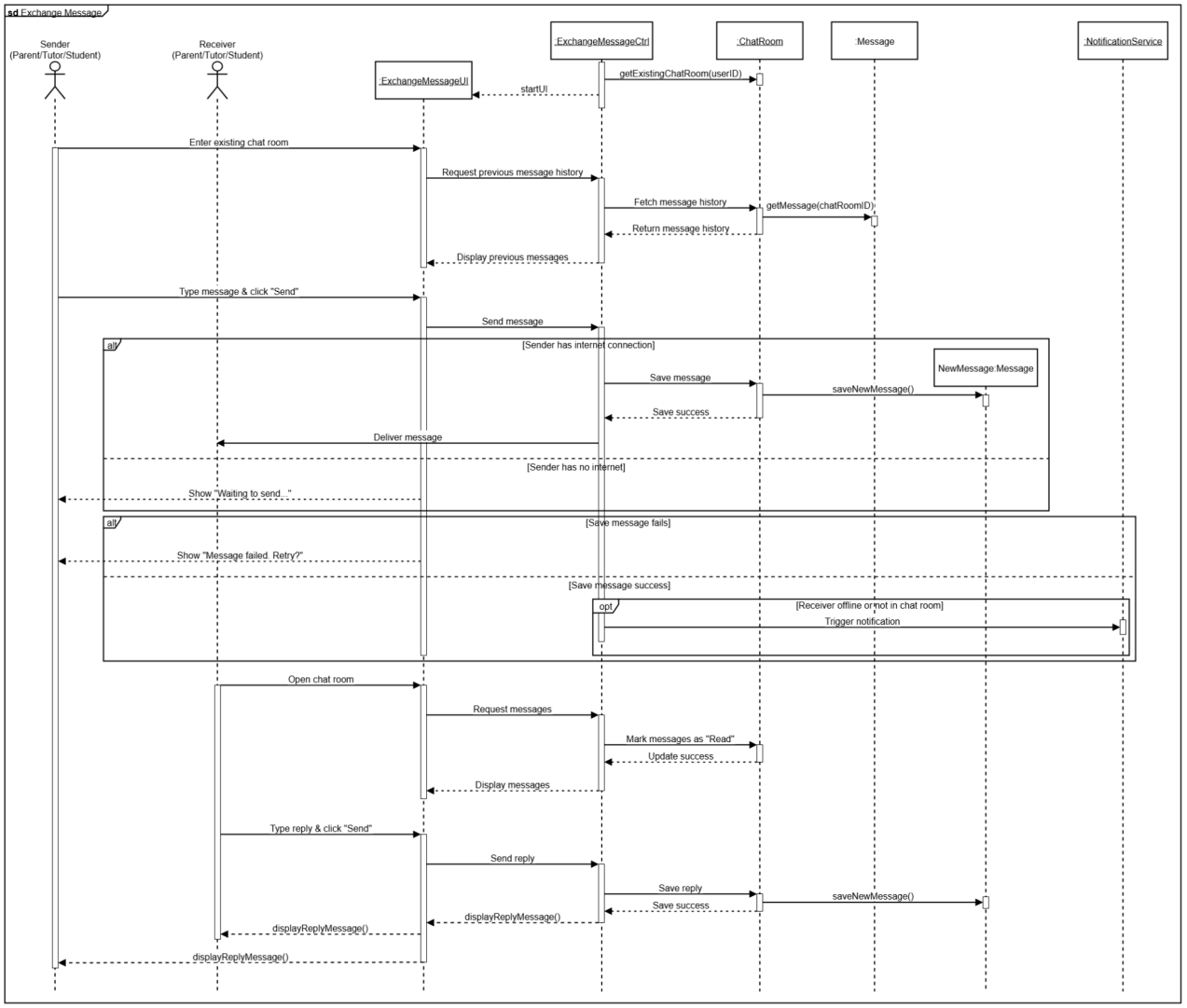


Figure 4.1.5.2 Sequence Diagram of Exchange Message

Close Chat Room

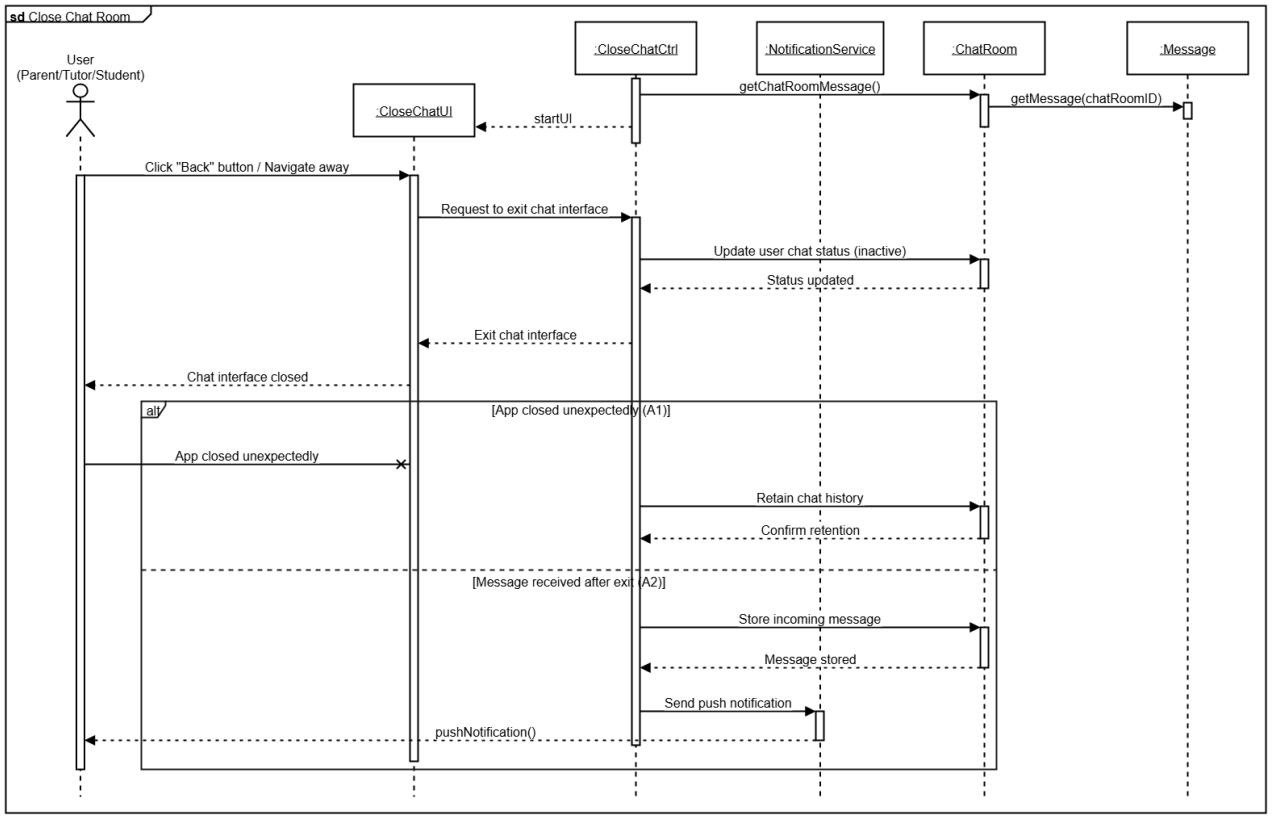


Figure 4.1.5.3 Sequence Diagram of Close Chat Room

4.2 State Chart Diagram

4.2.1 Subject Module

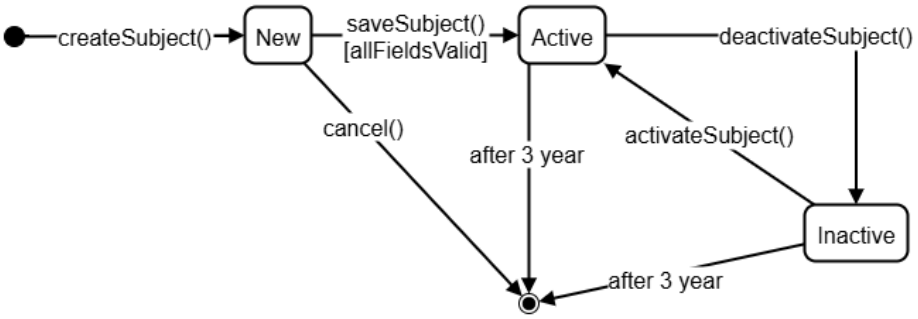


Figure 4.2.1 State Chart Diagram of Subject Module

4.2.2 Class Module

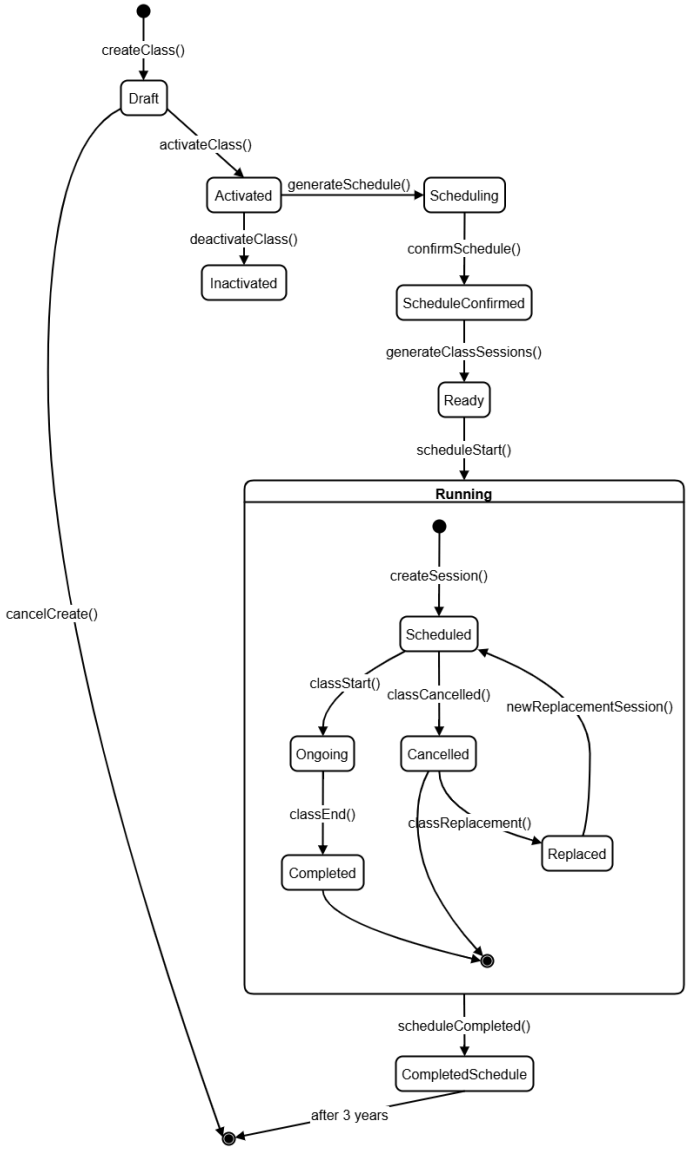


Figure 4.2.2 State Chart Diagram of Class Module

4.2.3 Payment Module

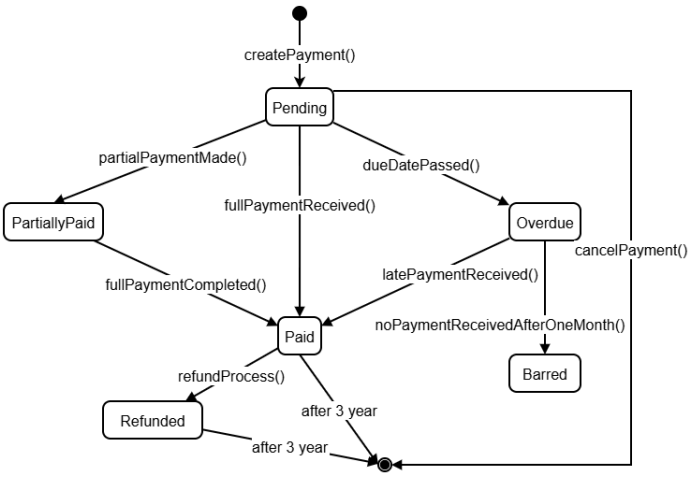


Figure 4.2.3 State Chart Diagram of Payment Module

4.2.4 Other Service Module

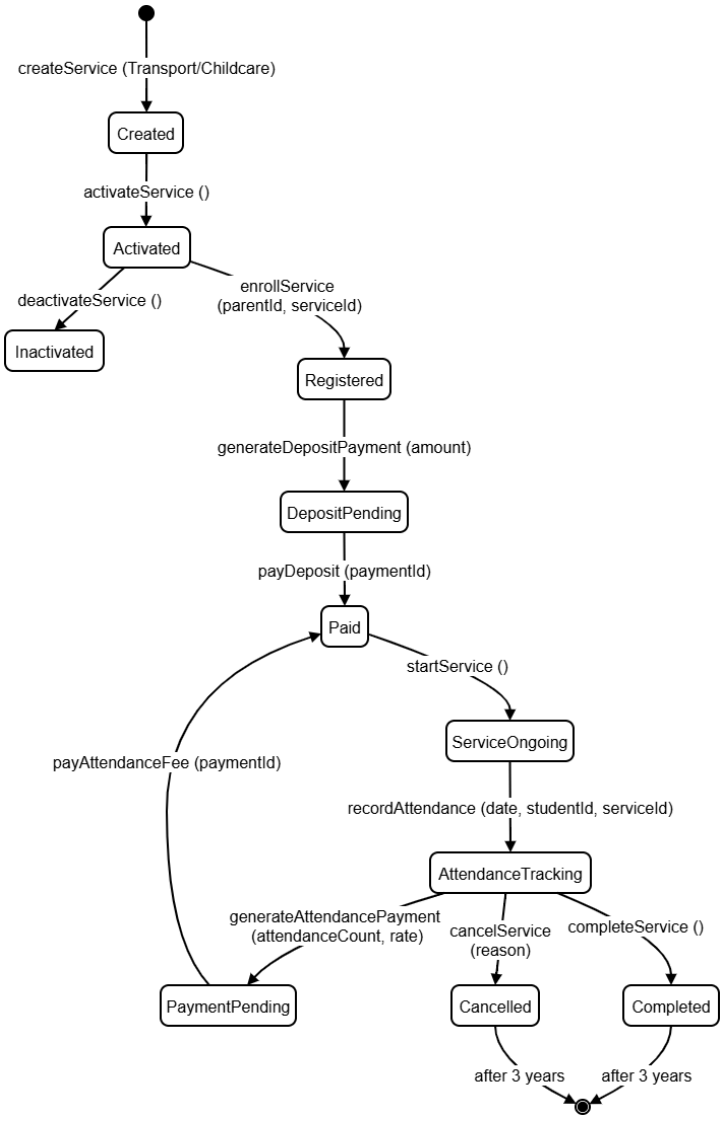


Figure 4.2.4 State Chart Diagram of Other Service Module

4.2.5 Chat Module

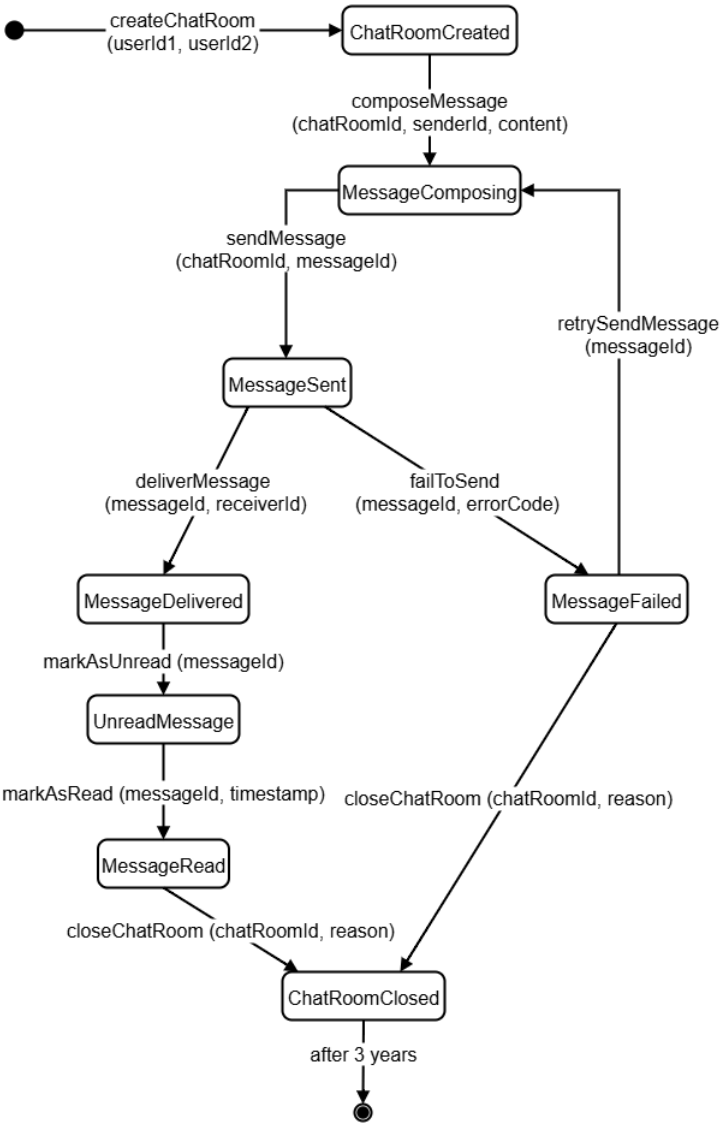
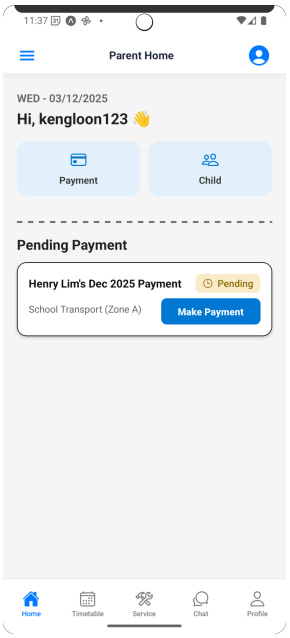
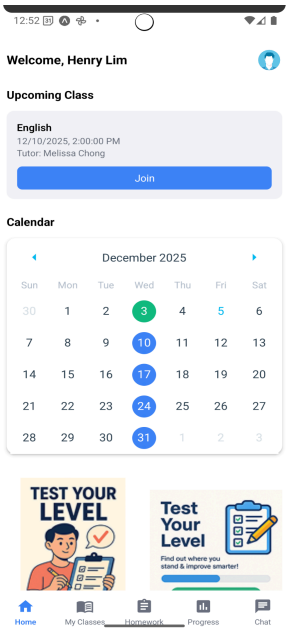


Figure 4.2.5 State Chart Diagram of Chat Module

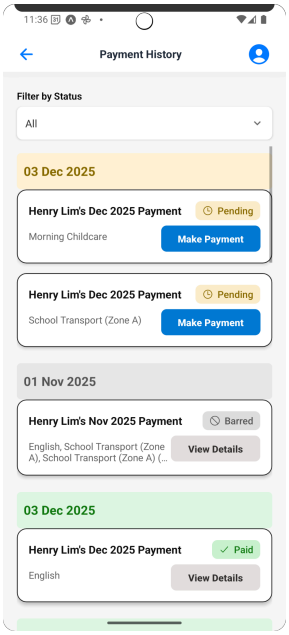
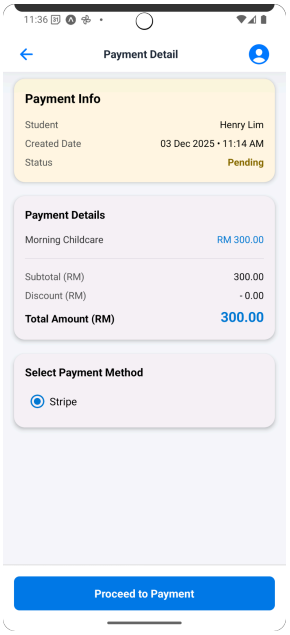
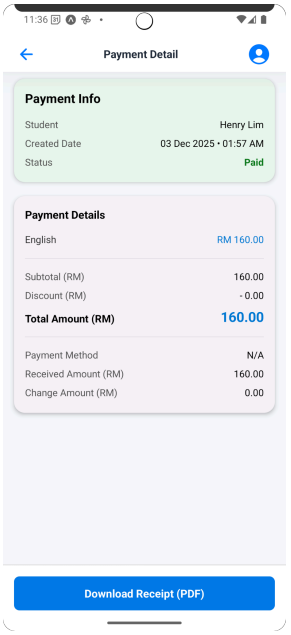
4.3 User Interface Design

4.3.1 Parent / Student / Tutor Mobile View

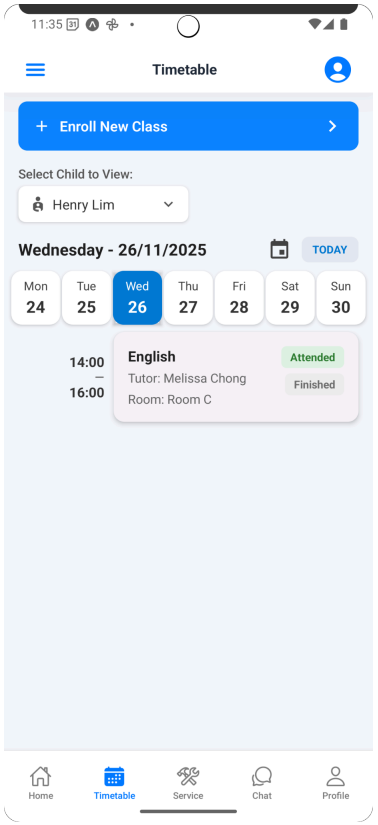
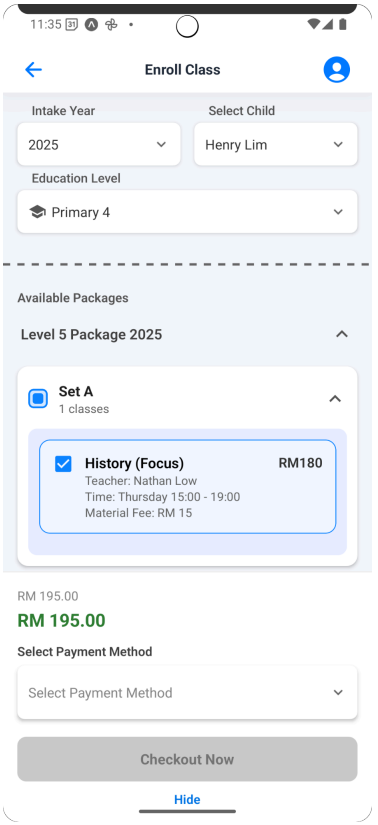
Home

Home view for parent	Home view for student
	

Payment UI

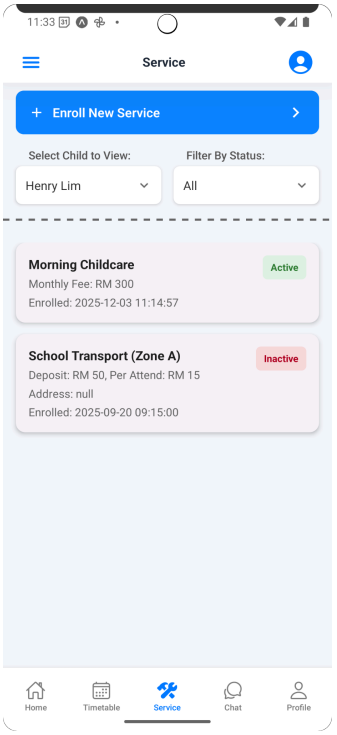
List to **view payment history** with status filtering	**Make payment** with selecting payment method	**Payment detail** with E-receipt (PDF) provided
		

Class Enrollment UI

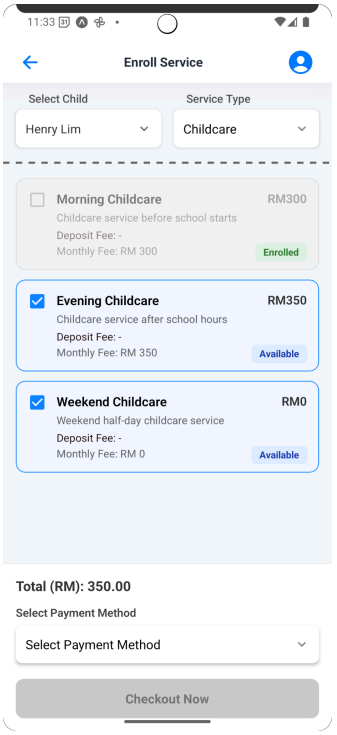
List to **view enrolled class** and **timetable** of child with selecting specific child.	Form for enroll new class with **self select multiple class** to enroll.
	

Service Enrollment UI

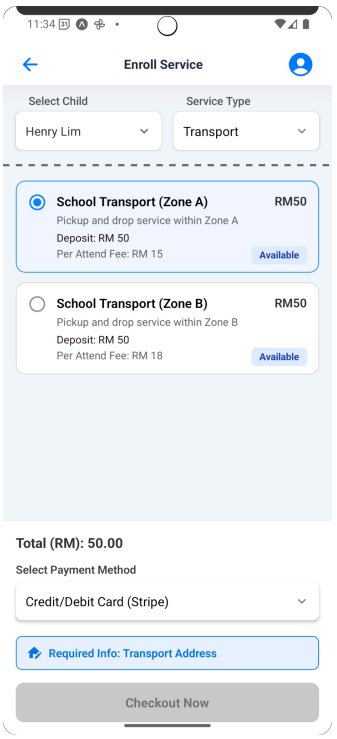
List to **view enrolled services** and **timetable** of child with selecting specific child.



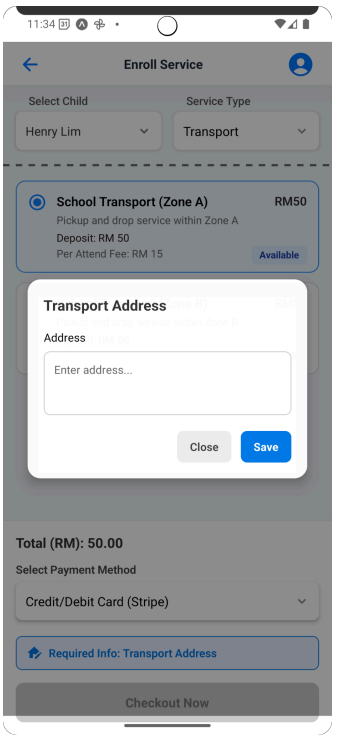
Form to enroll new childcare service with **self select multiple services** to enroll.



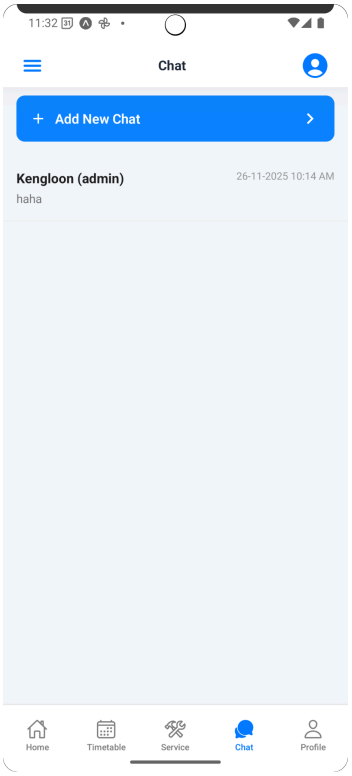
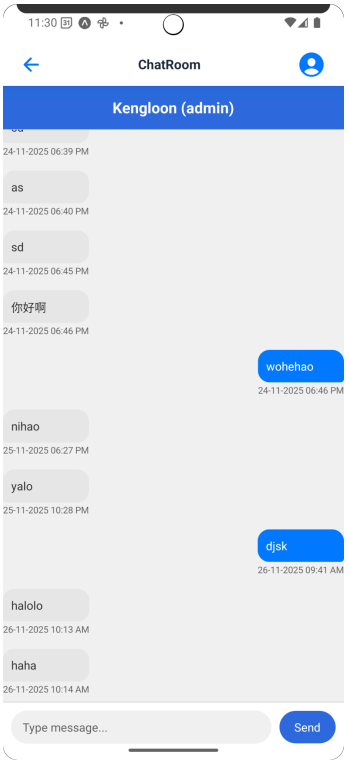
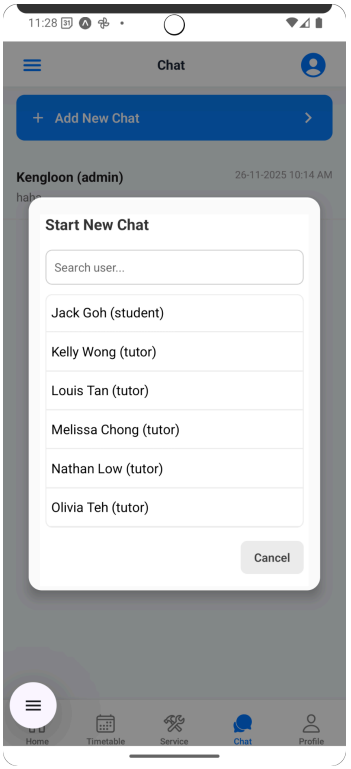
Form to enroll new transport service with **self select single services** to enroll and required to fill in required info.



Form to fill in required info of transport service.

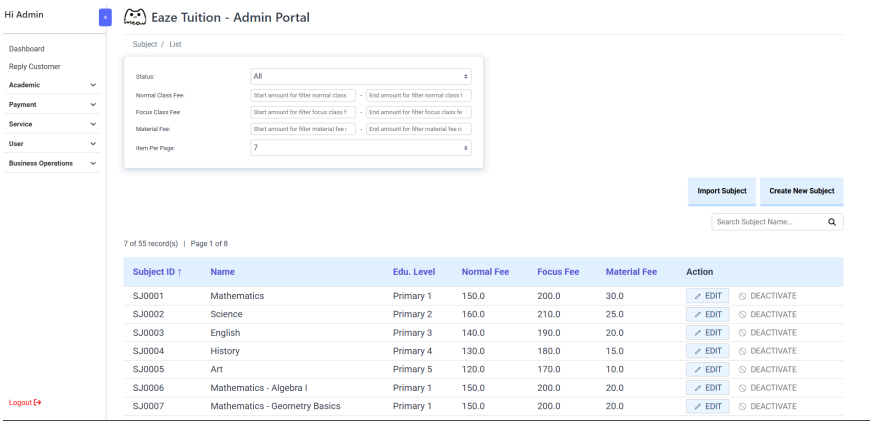
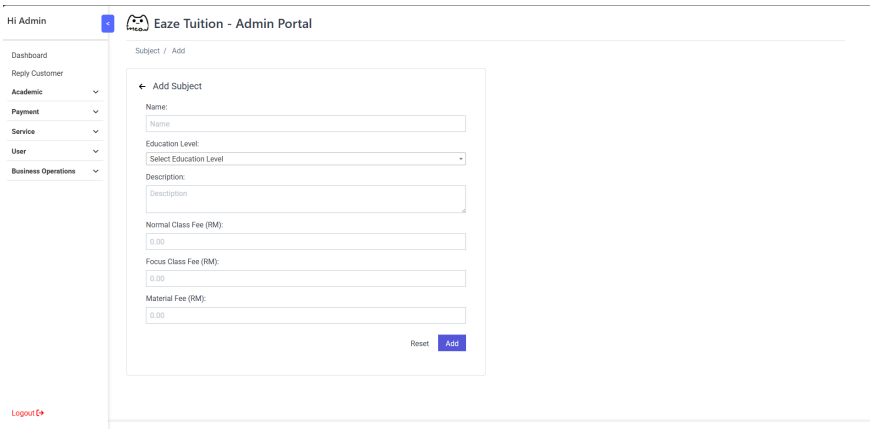
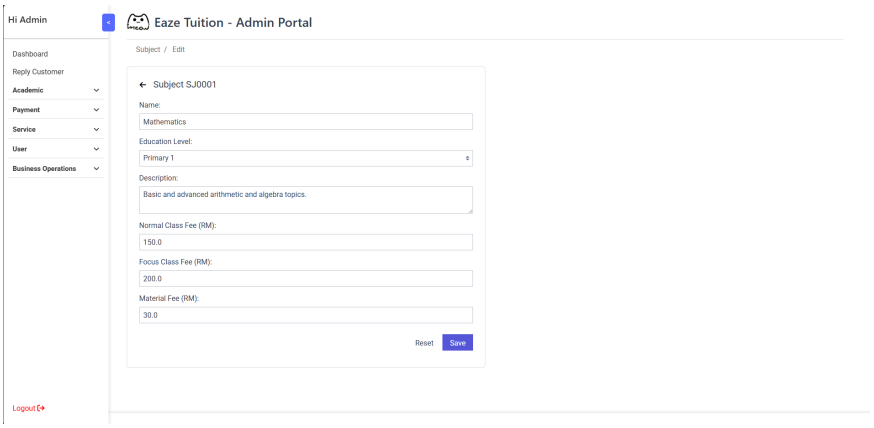


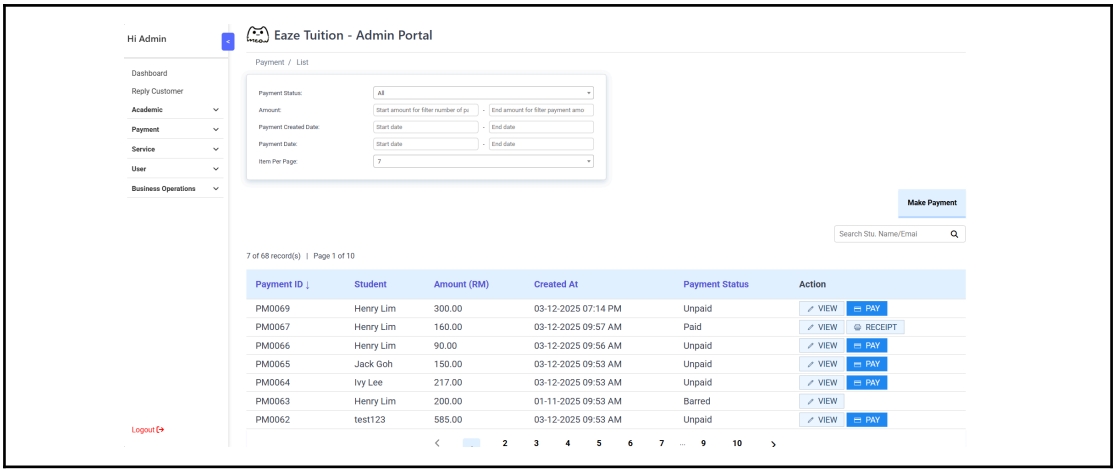
Chat UI

Chat list with previous contacted users.	Messaging view with another user.	Add new chat room by selecting a user
		

4.3.2 Admin / Staff Web Browser View

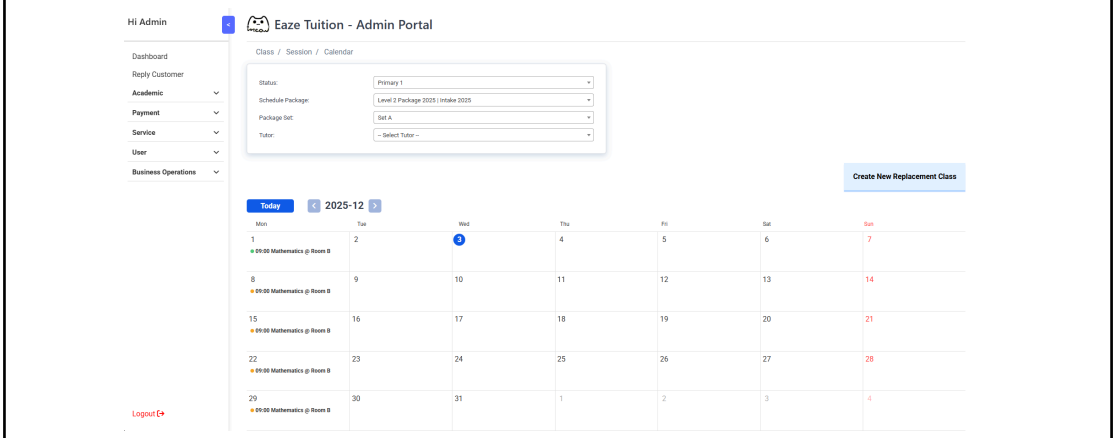
Admin Normal Management UI

Subject, class, class package schedule, service and payment management list layout.

Create subject, class and service form layout.

Update subject, class and service form layout.

Payment detail layout

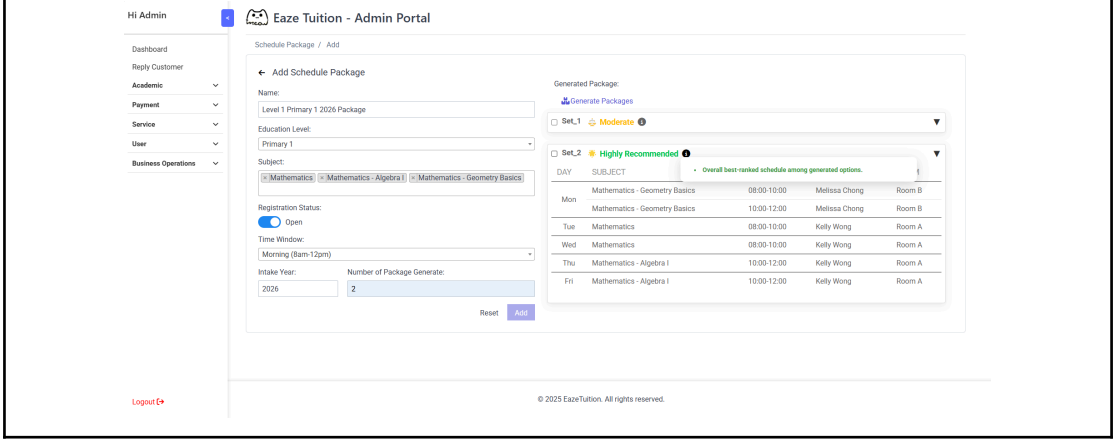


Class Schedule Management (Generate & Update) UI

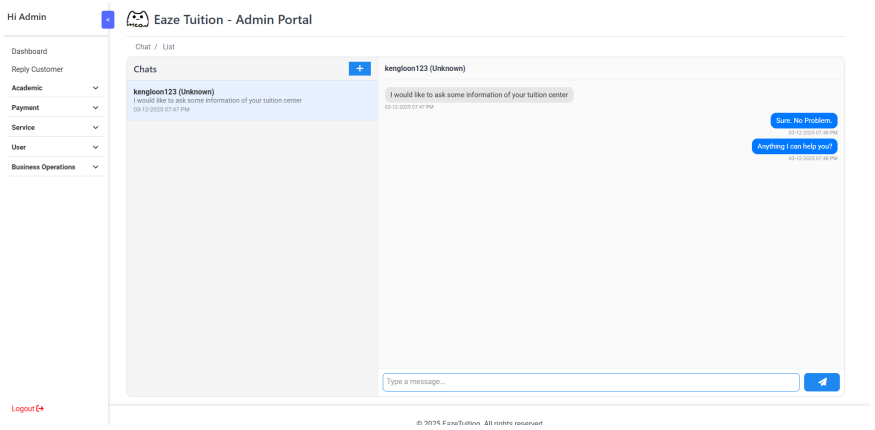
Class schedule package calendar view with filter.



Generate class schedule package layout with **AI recommendation and comment**.



Chat UI

Chat list with previous contacted user and messaging view with another user.


4.4 Data Design

4.4.1 Class Diagram

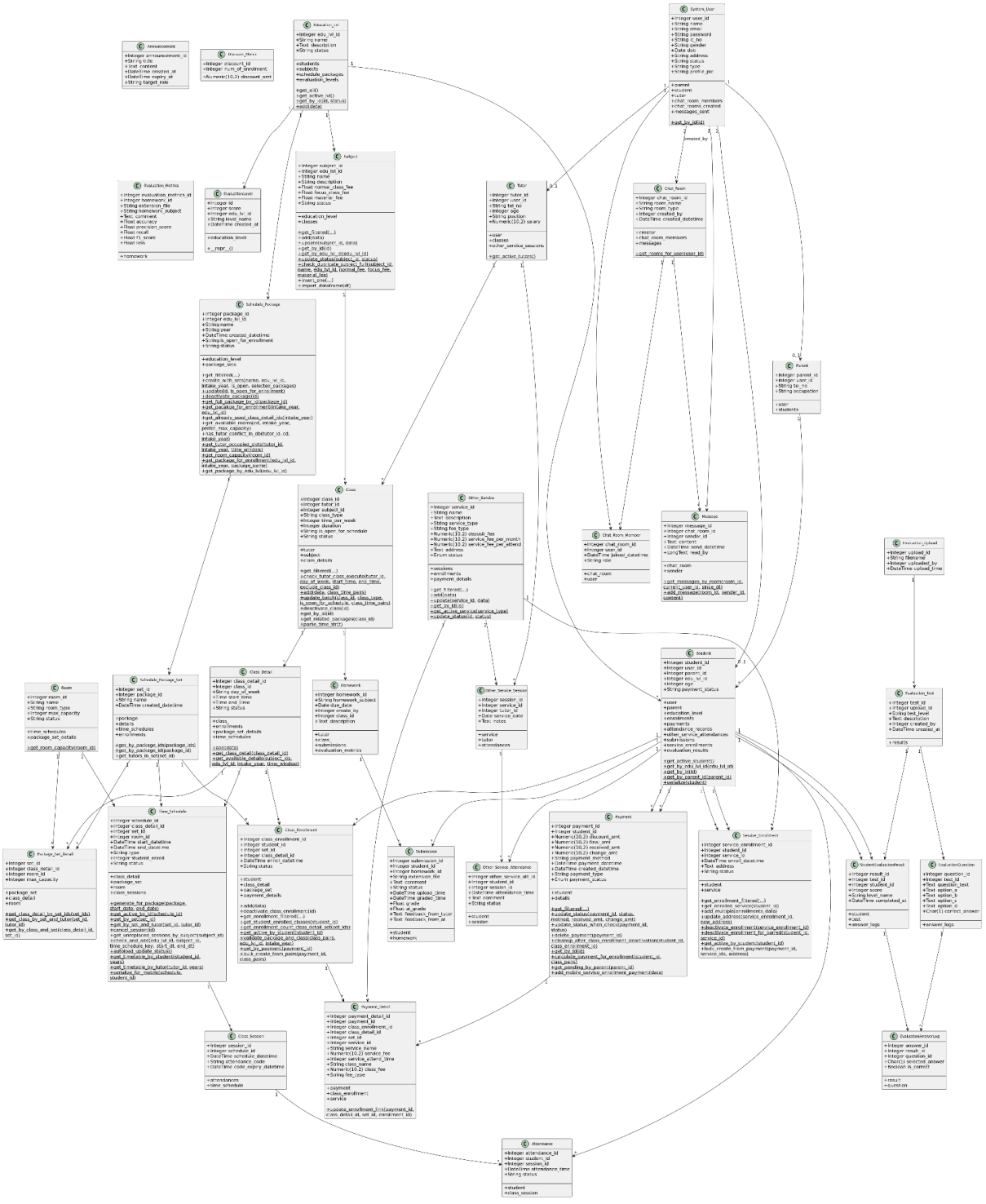


Figure 4.5.1 Class Diagram Page

4.4.2 Entity Relationship Diagram (ERD)

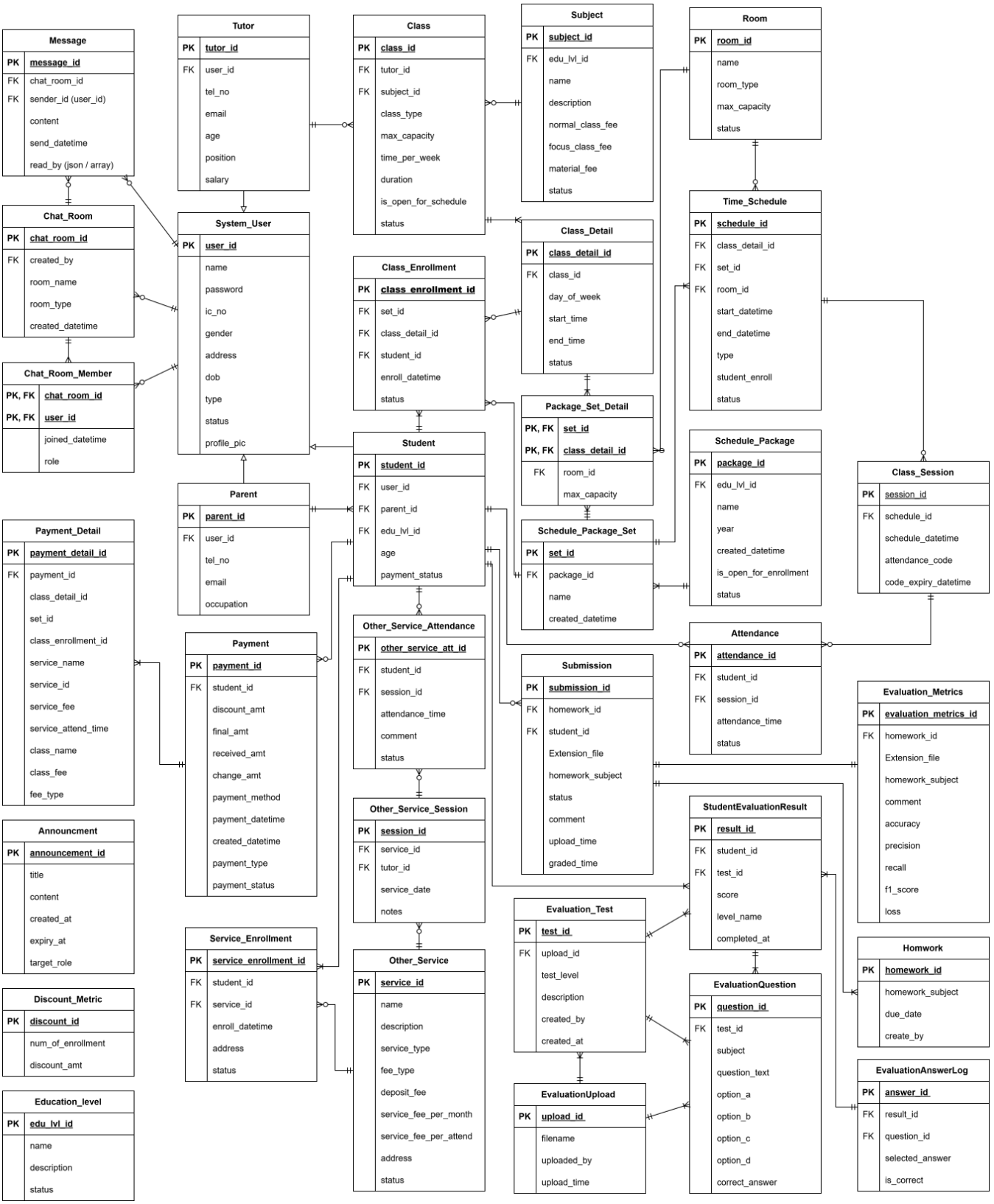


Figure 4.5.2 Entity Relationship Diagram Page 1

4.4.3 Data Dictionary

Subject Module

Subject Table

Field Name	Data Type	Size	Description	Constraints / Notes
subject_id	INT	11	Unique identifier for each subject	**Primary Key**, Auto Increment
edu_lvl_id	INT	11	Education level of the subject	**Foreign Key → education_level(edu_lvl_id)**, Not Null
name	VARCHAR	100	Name of the subject	Not Null
description	TEXT	—	Description of subject	Nullable
normal_class_fee	DECIMAL	10,2	Fee for normal class	Default 0.00
focus_class_fee	DECIMAL	10,2	Fee for focus class	Default 0.00
material_fee	DECIMAL	10,2	Material fee	Default 0.00
status	VARCHAR	1	Subject status ('0'=Inactive, '1'=Active)	Default '1'

Table 4.4.3.1.1 "Subject" Table Data Dictionary

Class Module

Class Table

Field Name	Data Type	Size	Description	Constraints / Notes
class_id	INT	11	Unique identifier for class	PK, Auto Increment
tutor_id	INT	11	Tutor assigned to the class	FK → Tutor(tutor_id), Not Null
subject_id	INT	11	Subject being taught	FK → Subject(subject_id), Not Null
class_type	VARCHAR	5	Class type (e.g., Normal, Focus)	Not Null
max_capacity	INT	3	Maximum number of students allowed	CHECK (0–999), Not Null
time_per_week	INT	2	Number of class sessions per week	CHECK (0–99), Not Null

duration	INT	3	Duration per session (minutes)	Not Null
is_open_for_schedule	VARCHAR	5	Whether the class is included in scheduling	Default '1'
status	VARCHAR	11	Class status ('0'=Inactive, '1'=Active)	Default '1'

Table 4.4.3.2.1 "Class" Table Data Dictionary

Class Detail Table

Field Name	Data Type	Size	Description	Constraints / Notes
class_detail_id	INT	11	Unique detail record of class time	Primary Key, Auto Increment
class_id	INT	11	Parent class	Foreign Key → class(class_id), Not Null
day_of_week	VARCHAR	20	Day of week	Nullable
start_time	TIME	—	Class start time	Nullable
end_time	TIME	—	Class end time	Nullable
status	VARCHAR	1	Detail status ('0'=Inactive, '1'=Active)	Default '1'

Table 4.4.3.2.2 "Class_Detail" Table Data Dictionary

Class Enrollment Table

Field Name	Data Type	Size	Description	Constraints / Notes
class_enrollment_id	INT	11	Unique enrollment record	PK, Auto Increment
student_id	INT	11	Student enrolled in the class	FK → Student(student_id)
class_detail_id	INT	11	Class detail enrolled	FK → class_detail(class_detail_id), Not Null
set_id	INT	11	Package set	FK → schedule_package_set(set_id), Not Null
enroll_datetime	DATETIME	-	Date and time of enrollment	Not Null

status	VARCHAR	11	Enrollment status ('0'=Inactive, '1'=Active)	Default '1'

Table 4.4.3.2.3 "Class_Enrollment" Table Data Dictionary

Time Schedule Table

Field Name	Data Type	Size	Description	Constraints / Notes
schedule_id	INT	11	Unique identifier for schedule	PK, Auto Increment
class_detail_id	INT	11	Linked class detail	FK → class_detail(class_detail_id), Not Null
set_id	INT	11	Linked package set	FK → schedule_package_set(set_id), Not Null
room_id	INT	11	Room assigned	FK → Room(room_id)
start_datetime	DATETIME	-	Schedule start datetime	Not Null
end_datetime	DATETIME	-	Schedule end datetime	Not Null
type	VARCHAR	11	Type of the class session	Not Null
student_enroll	INT	11	Number of enrolled students	Not Null
status	VARCHAR	11	Schedule status ('0'=Cancelled, '1'=Scheduled, '2'=Ongoing, '3'=Finished, '4'=Replacement)	Default '1'

Table 4.4.3.2.4 "Time_Schedule" Table Data Dictionary

Class Session Table

Field Name	Data Type	Size	Description	Constraints / Notes
session_id	INT	11	Unique identifier for class session	PK, Auto Increment
schedule_id	INT	11	Schedule associated with session	FK → Time_Schedule(schedule_id)
schedule_datetime	DATETIME	-	Date and time of the session	Not Null

attendance_code	VARCHAR	20	Unique attendance code for session	Nullable
code_expiry_datetime	DATETIME	-	Expiry time of attendance code	Nullable

Table 4.4.3.2.5 "Class_Session" Table Data Dictionary

Room Table

Field Name	Data Type	Size	Description	Constraints / Notes
room_id	INT	11	Unique identifier for the room	PK, Auto Increment
name	VARCHAR	100	Room name/label (e.g., Room A, Room B)	Not Null
room_type	INT	11	Type of the room (1=Small, 2=Medium, 3=Big)	CHECK (>0), Not Null
max_capacity	INT	11	Maximum number of students the room can hold	CHECK (>0), Not Null
status	VARCHAR	11	Room status ('0'=Inactive, '1'=Active)	Default '1'

Table 4.4.3.2.6 "Room" Table Data Dictionary

Schedule Package Table

Field Name	Data Type	Size	Description	Constraints / Notes
package_id	INT	11	Unique identifier	Primary Key, Auto Increment
edu_lvl_id	INT	11	Education level	FK → education_level (edu_lvl_id)
name	VACHAR	50	Name of the Package	Not Null
year	INT	11	Intake year	Not Null
created_datetime	DATETIME	—	Created timestamp	Not Null
is_open_for_enrollment	VARCHAR	11	Is it allow for student to register	Not Null
status	VARCHAR	10	'1'=Active, '0'=Inactive	Default '1'

Table 4.4.3.2.7 "Schedule Package" Table Data Dictionary

Schedule Package Set Table

Field Name	Data Type	Size	Description	Constraints / Notes
set_id	INT	11	Unique package set	Primary Key, Auto Increment
package_id	INT	11	Parent package	FK → schedule_package(package_id)
name	VARCHAR	50	Name of the package set	Not Null
created_datetime	DATETIME	—	Created timestamp	Not Null

Table 4.4.3.2.8 "Schedule Package Set" Table Data Dictionary

Package Set Detail Table

Field Name	Data Type	Size	Description	Constraints / Notes
set_id	INT	11	Parent set	PK, FK → schedule_package_set(set_id)
class_detail_id	INT	11	Linked class detail	PK, FK → class_detail(class_detail_id)
room_id	INT	11	Linked assigned room	FK → room(room_id)
max_capacity	INT	-	Maximum number of student to be enrolled	Not Null

Table 4.4.3.2.9 "Package Set Detail" Table Data Dictionary

Payment Module

Payment Table

Field Name	Data Type	Size	Description	Constraints / Notes
payment_id	INT	11	Unique identifier for each payment	PK, Auto Increment
student_id	INT	11	The student who the payment belongs to	FK → Student(student_id), Not Null

discount_amt	DECIMAL	10,2	Discount amount applied to the payment	Default 0.00
final_amt	DECIMAL	10,2	Amount payable after discount	Not Null, CHECK(>=0)
received_amt	DECIMAL	10,2	Actual amount received from student/parent	Not Null, CHECK(>=0)
change_amt	DECIMAL	10,2	Change returned to student/parent (received - final_amt)	Default 0.00
total_amount	DECIMAL	10,2	Original total amount before discount	Not Null, CHECK(>0)
payment_method	VARCHAR	11	Payment method (cash, stripe, tng)	Not Null
payment_datetime	DATETIME	—	Date and time when payment is made	Not Null
created_datetime	DATETIME	—	When payment record was created	Not Null
payment_type	VARCHAR	20	Payment category (e.g., class_enrollment, service_enrollment)	Nullable
payment_status	ENUM	('0','1','2','3')	Payment status ‘0’=Pending, ‘1’=Paid, ‘2’=Failed, ‘3’=Cancelled	Default '1'

Table 4.4.3.3.1 "Payment" Table Data Dictionary

Payment Detail Table

Field Name	Data Type	Size	Description	Constraints / Notes
payment_detail_id	INT	11	Unique identifier for each payment detail	PK, Auto Increment
payment_id	INT	11	Reference to the related payment	FK → Payment(payment_id), Not Null
class_enrollment_id	INT	11	Class enrollment paid (if applicable)	FK → class_enrollment(class_enrollment_id), Nullable
class_detail_id	INT	11	Class detail ID for reference	Nullable
set_id	INT	11	Package set reference	Nullable

service_id	INT	11	Service paid (if applicable)	FK → other_service(service_id), Nullable
service_name	VARCHAR	100	Name of the service paid (e.g., Transport, Childcare)	Nullable (used when service fee applies)
service_fee	DECIMAL	10,2	Fee charged for the service	Nullable, Default 0.00
service_attend_time	VARCHAR	50	Attendance period for service (e.g., "Morning", "Evening")	Nullable
class_name	VARCHAR	100	Name of the class/subject paid for	Nullable (used when class fee applies)
class_fee	DECIMAL	10,2	Fee charged for the class	Nullable, Default 0.00
fee_type	VARCHAR	50	Type of fee ('1'=monthly, '2'=Deposit, '3'=by_attend)	Not Null

Table 4.4.3.3.2 "Payment_Detail" Table Data Dictionary

Other Service Module

Other Service Table

Field Name	Data Type	Size	Description	Constraints / Notes
service_id	INT	11	Unique identifier for each service	PK, Auto Increment
name	VARCHAR	100	Name of the service (e.g., Transport, Childcare)	Not Null
description	TEXT	—	Additional details of the service	Nullable
service_type	VARCHAR	—	Type of service ('1'=Daycare, '2'='Transport, '3'=Other)	Not Null
fee_type	VARCHAR	50	Type of fee ('1'=Pay Direct, '2'=Deposit&Attendance)	Not Null
deposit_fee	DECIMAL	10,2	Deposit amount required for the service	Nullable, CHECK(>=0)

service_fee_per_month	DECIMAL	10,2	Monthly service fee	Not Null, CHECK(>=0)
service_fee_per_attend	DECIMAL	10,2	Fee charged per attendance (if applicable)	Nullable, CHECK(>=0)
address	VARCHAR	255	Address related to the service (e.g., pickup location for transport, childcare center address)	Nullable
status	VARCHAR	11	Service status ('0'=Inactive, '1'=Active)	Default '1'

Table 4.4.3.4.1 "Other_Service" Table Data Dictionary

Service Enrollment Table

Field Name	Data Type	Size	Description	Constraints / Notes
service_enrollment_id	INT	11	Unique enrollment record	PK, Auto Increment
student_id	INT	11	Student enrolled in the class	PK, FK → Student(student_id)
service_id	INT	11	Service enrolled	PK, FK → Class(class_id)
enroll_datetime	DATETIME	-	Date and time of enrollment	Not Null
address	TEXT	-	Service address (transport only)	Nullable
status	VARCHAR	11	Enrollment status ('0'=Inactive, '1'=Active)	Default '1'

Table 4.4.3.4.2 "Service_Enrollment" Table Data Dictionary

Other Service Session Table

Field Name	Data Type	Size	Description	Constraints / Notes
session_id	INT	11	Unique identifier for class session	PK, Auto Increment
service_id	INT	11	Service associated with session	FK → Other_Service(service_id)
tutor_id	INT	11	Tutor in charge for the service session	FK → Tutor(tutor_id)
service_datetime	DATETIME	-	Date and time of the session	Not Null

notes	TEXT	-	Note or reminder for the service session	Nullable

Table 4.4.3.4.3 "Other_Service_Session" Table Data Dictionary

Chat Module

Chat Room Table

Field Name	Data Type	Size	Description	Constraints / Notes
chat_room_id	INT	11	Unique identifier for each chat room	PK, Auto Increment
room_name	VARCHAR	100	Optional room name (e.g., Group Chat)	Nullable
room_type	VARCHAR	100	Type of the chat ('1'= Personal, '2'=Group)	Not Null
created_by	INT	11	User who created the room	FK → User(user_id), Not Null
created_datetime	DATETIME	—	Date and time when the chat room was created	Not Null, Default CURRENT_TIMESTAMP

Table 4.4.3.5.1 "Chat_Room" Table Data Dictionary

Message Table

Field Name	Data Type	Size	Description	Constraints / Notes
message_id	INT	11	Unique identifier for each message	PK, Auto Increment
chat_room_id	INT	11	Reference to the chat room	FK → Chat_Room(chat_room_id), Not Null
sender_id	INT	11	User who sent the message	FK → User(user_id), Not Null
content	TEXT	—	Message content	Nullable (in case of attachments)
send_datetime	DATETIME	—	Timestamp when the message was sent	Not Null, Default CURRENT_TIMESTAMP
read_by	JSON	100	Whether the message read by who	Default [] (Unread)

Table 4.4.3.5.2 "Message" Table Data Dictionary

Chat_Room_Member Table

Field Name	Data Type	Size	Description	Constraints / Notes
chat_room_id	INT	11	Chat room the user involved	PK, FK →Chat_Room(chat_room_id)
user_id	INT	11	User involved in the chat room	PK, FK → User(user_id)
joined_datetime	DATETIME	-	Date and time of joining the chat room	Not Null
role	VARCHAR	10	Role of the user in chat room ('1'=Admin, '2'=Member)	Default '1'

Table 4.4.3.5.3 "Chat_Room_Member" Table Data Dictionary

4.5 Reports Design

4.5.1 Revenue Summary Report

Heading:	Revenue Summary Report
Purpose:	Provides management with a comprehensive overview of tuition and service revenue performance. With this report, managers can: ● Monitor total revenue from all revenue sources (including disciplines, courses, and services) ● Track payment collection efficiency ● Compare revenue trends across different months and years ● Support financial planning and resource allocation decisions
Filter:	● Filter by Month ● Filter by Year ● Filter by Module Type (Tuition / Service) ● Filter by Payment Method (Stripe / Cash)
Report Content:	**Summary Section (Top)** ● Total revenue (collected) ● Revenue by module type ● Average revenue per student

	Graphs / Visualization Line Chart → Revenue trend over selected period

Table 4.5.1 Detail of Revenue Summary Report

4.5.2 Outstanding Payment Report

Heading:	Outstanding Payment Report
Purpose:	Provides management with a detailed overview of unpaid and partially paid fees. With this report, administrators can: ● Identify students and parents with outstanding fees ● Support follow-up measures to ensure timely collection ● Analyze trends in outstanding payments to assist with financial planning
Filter:	● Filter by Month ● Filter by Year ● Filter by Module Type (Tuition / Service)
Report Content:	**Summary Section (Top)** ● Total outstanding amount ● Number of unpaid invoices ● Number of overdue payments ● Average outstanding amount per student **Graphs / Visualization** Line Chart → Outstanding balance trend over selected months

Table 4.5.1 Detail of Revenue Summary Report

4.5.3 Top 5 Service Subscription Report

Heading:	Service Subscription Report
Purpose:	Provides management with insights into the most subscribed services. With this report, administrators can: ● Identify the top-performing services based on subscription count ● Evaluate service demand to support resource allocation and

	capacity planning ● Determine whether additional sessions, slots, or similar services should be introduced ● Monitor subscription-driven service revenue ● Understand seasonal or monthly trends in service enrollment
Filter:	● Filter by Month ● Filter by Year ● Filter by Service Type (Transport / Childcare / Other)
Report Content:	**Summary Section (Top)** ● Total number of subscriptions **Graphs / Visualization** Pie Chart → Subscription Count Distribution by Service

Table 4.5.3 Detail of Top 5 Service Subscription Report

4.6 Process Design

4.6.1 Subject Module

Create Subject

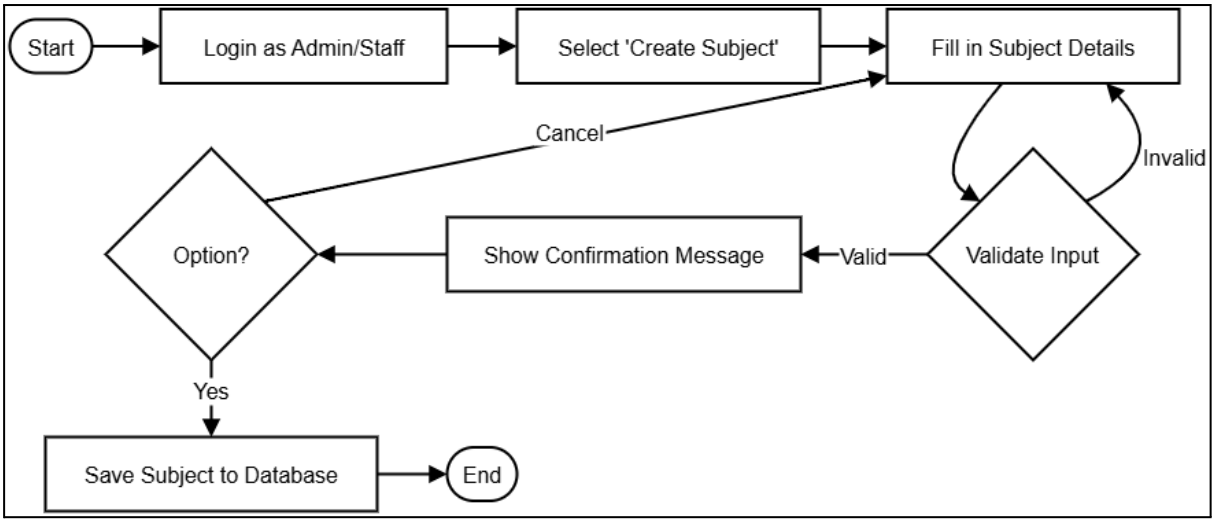


Figure 4.6.1.1 Flowchart of Create Subject

View Subject List

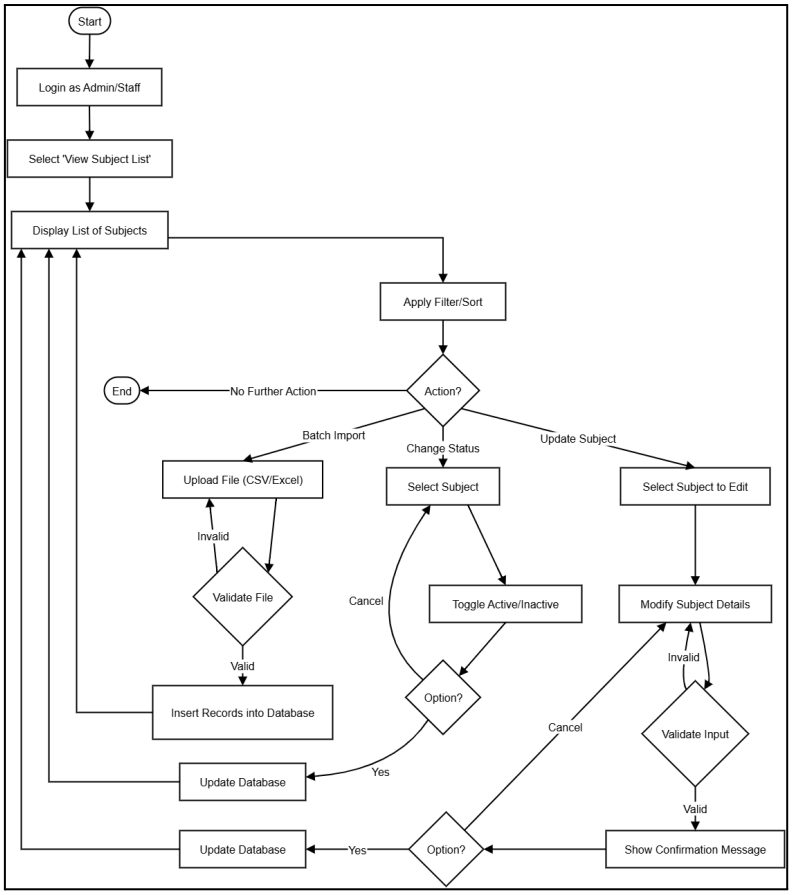


Figure 4.6.1.2 Flowchart of View Subject List

4.6.2 Class Module

Class Management

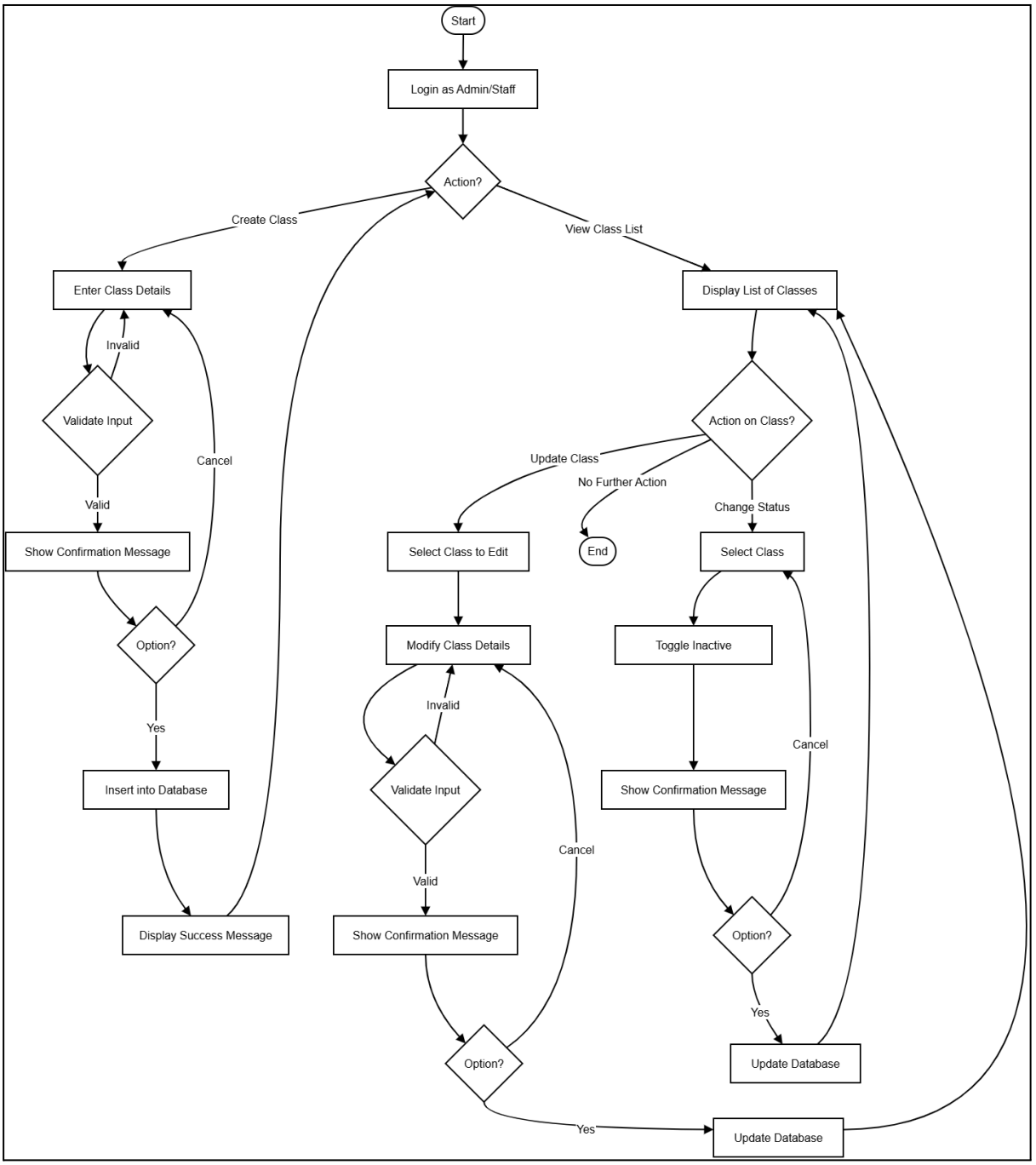


Figure 4.6.2.1 Flowchart of Class Management

Generate Class Schedule

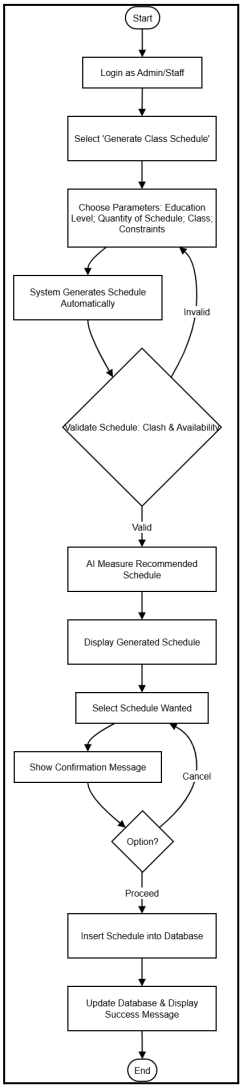


Figure 4.6.2.2 Flowchart of Generate Class Schedule

Cancel Class Enrollment

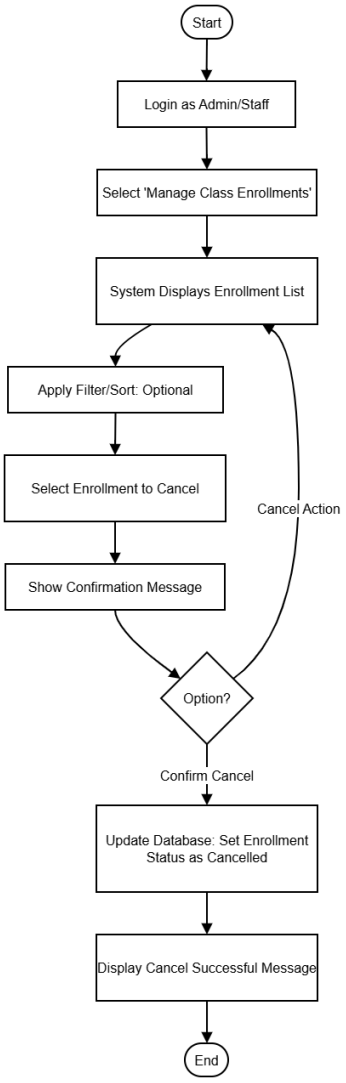


Figure 4.6.2.3 Flowchart of Cancel Class Enrollment

Class Enrollment

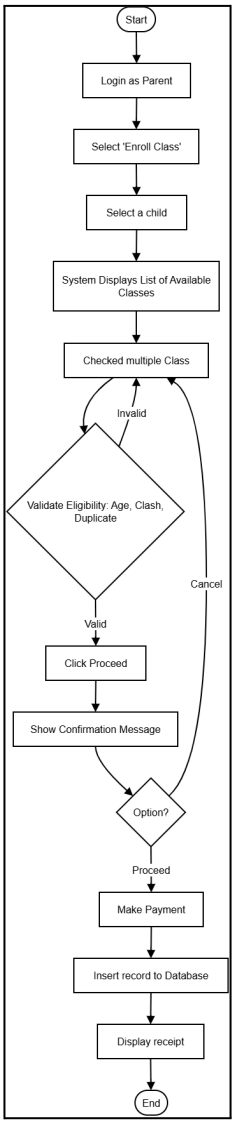


Figure 4.6.2.5 Flowchart of Class Enrollment

4.6.3 Payment Module

Make Online Payment

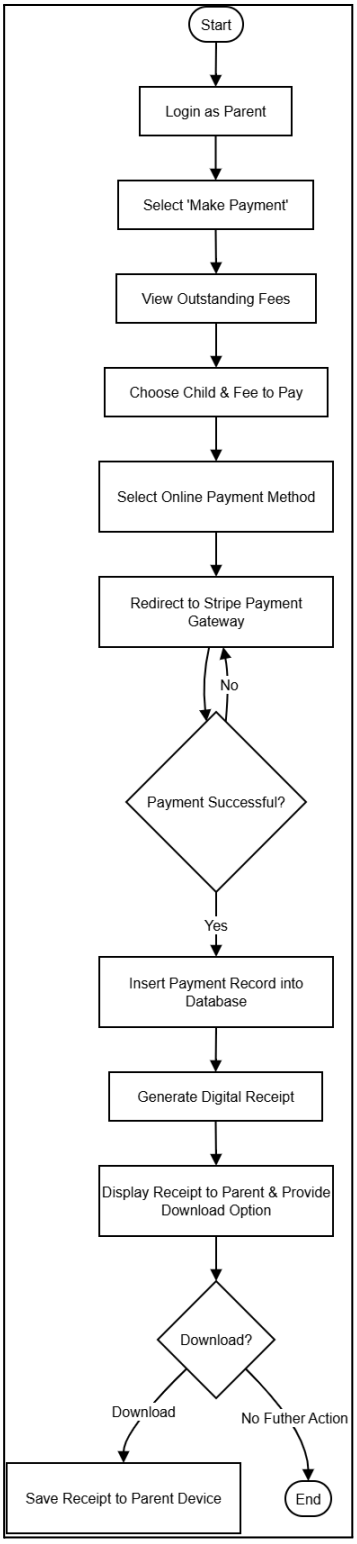


Figure 4.6.3.1 Flowchart of Make Online Payment

Make Offline Payment

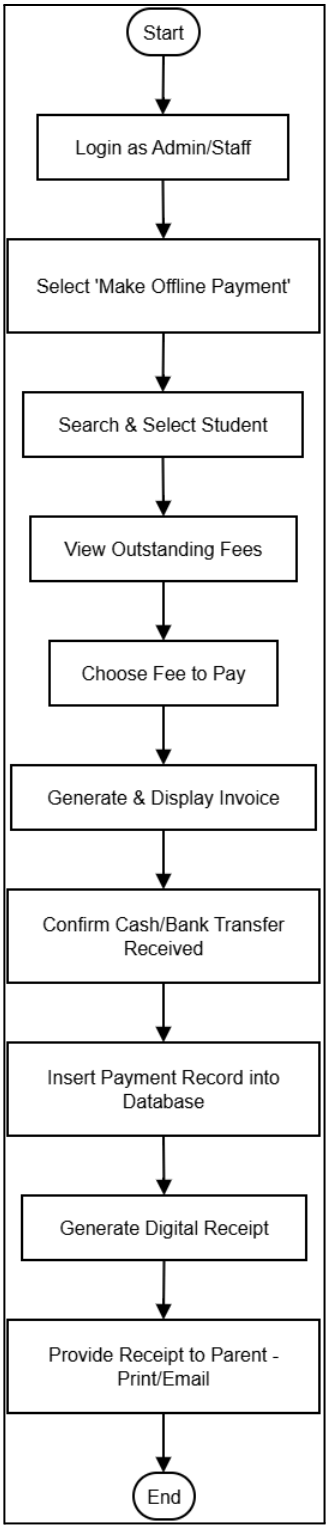


Figure 4.6.3.2 Flowchart of Make Offline Payment

View Pending / History Payments

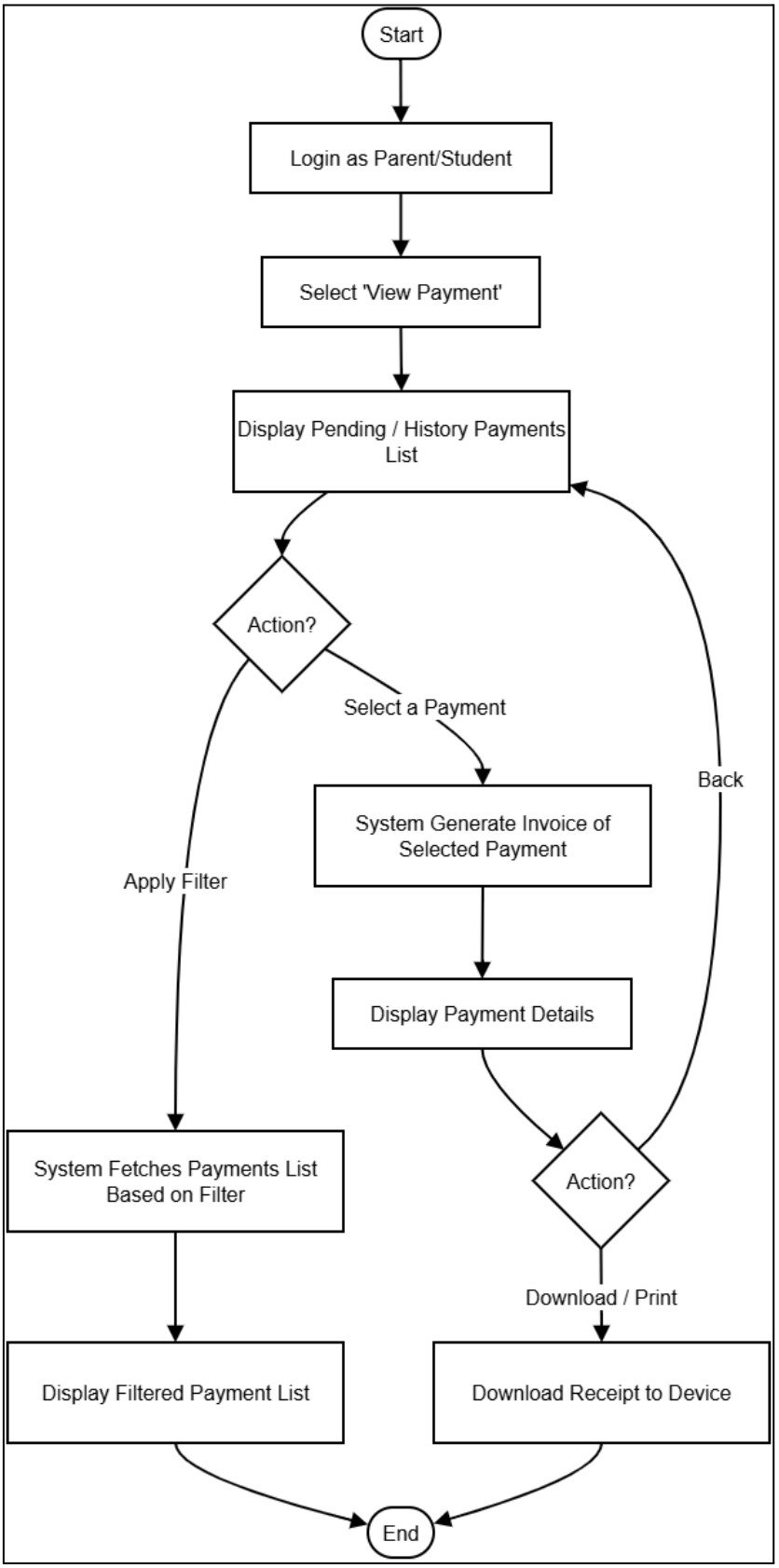


Figure 4.6.3.3 Flowchart of View Pending / History Payments

4.6.4 Other Service Module

Manage Other Service

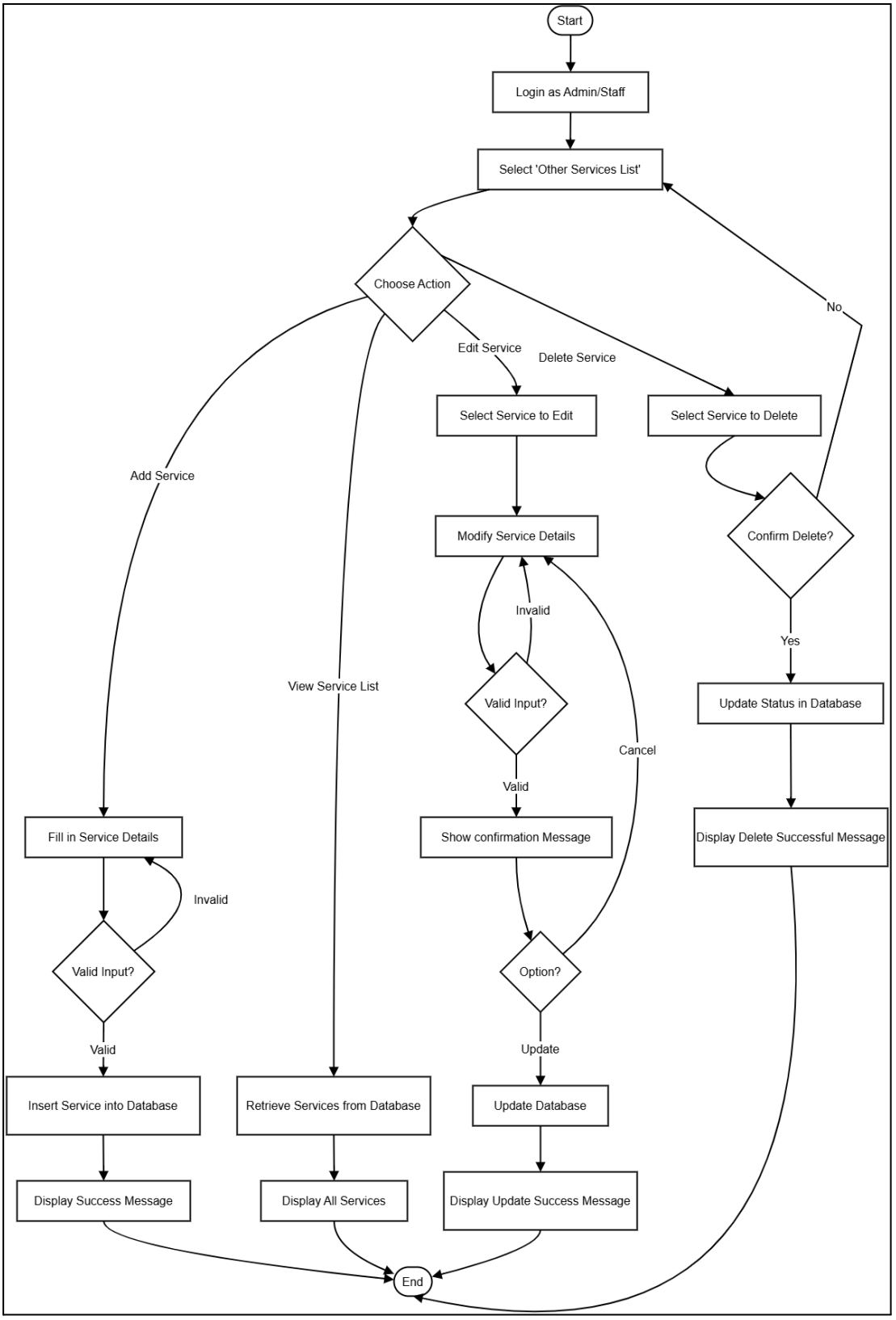


Figure 4.6.4.1 Flowchart of Manage Other Service

Cancel Service Enrollment

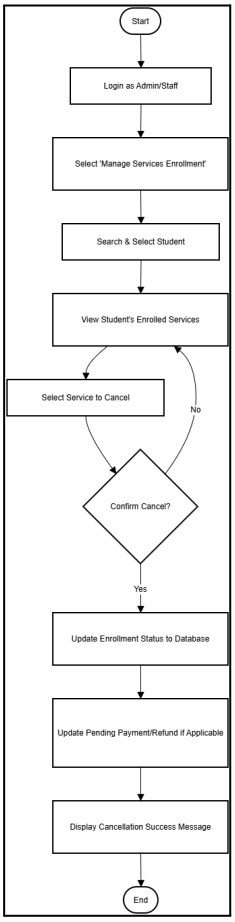


Figure 4.6.4.2 Flowchart of Cancel Service Enrollment

Service Enrollment

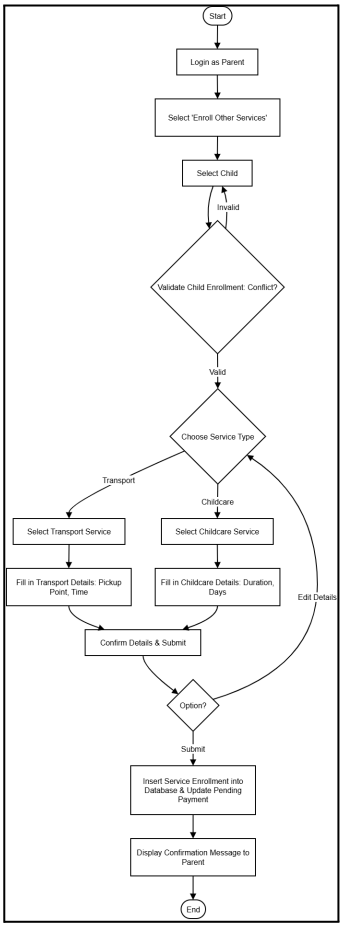


Figure 4.6.4.3 Flowchart of Service Enrollment

View Enrolled Service

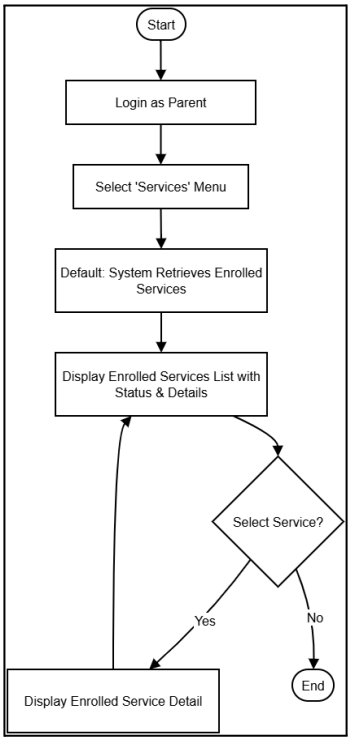


Figure 4.6.4.4 Flowchart of View Enrolled Service

4.6.5 Chat Module

Chat Management

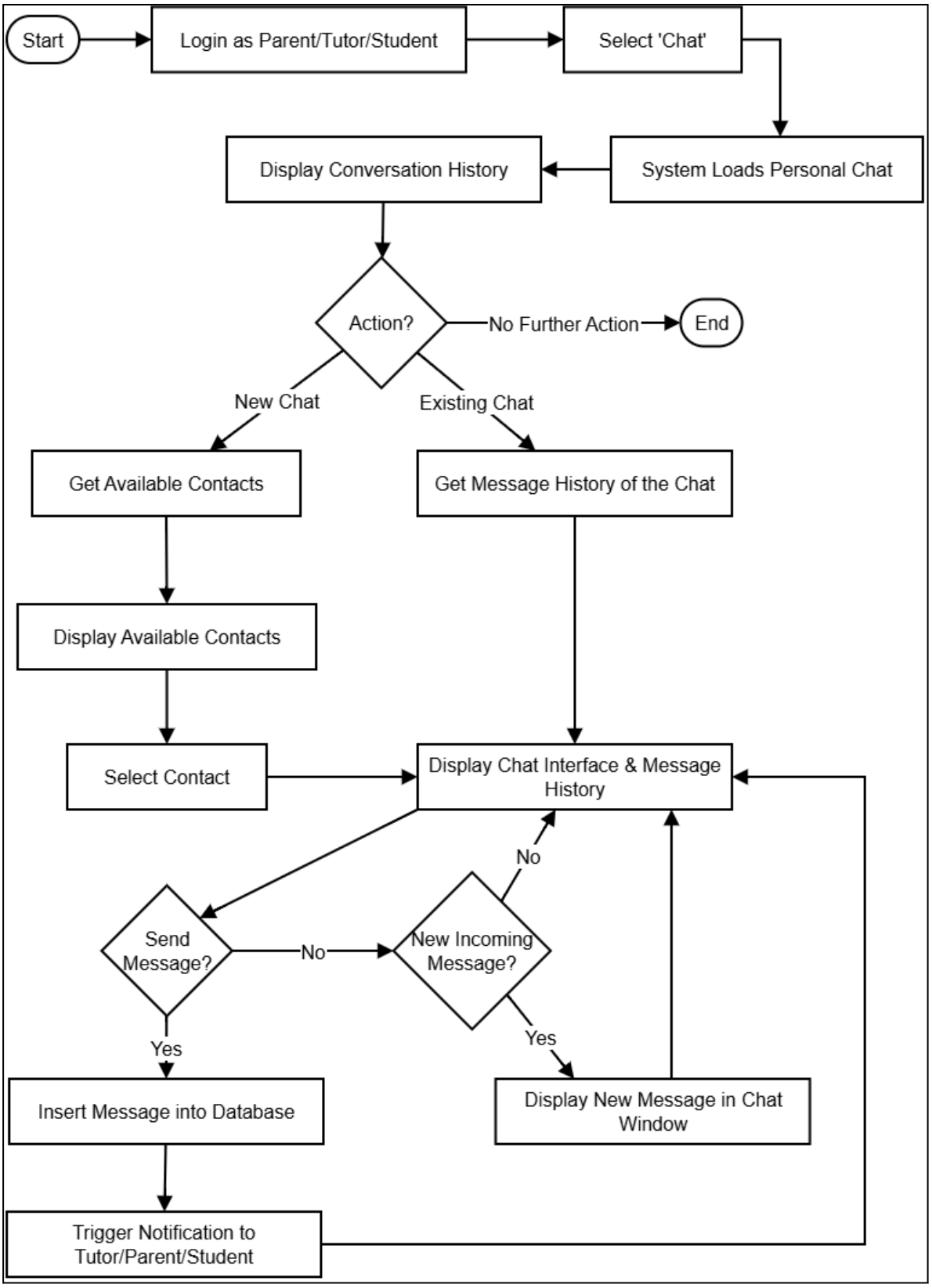


Table 4.6.5.1 Flowchart of Chat Management

4.7 Software Architecture Design

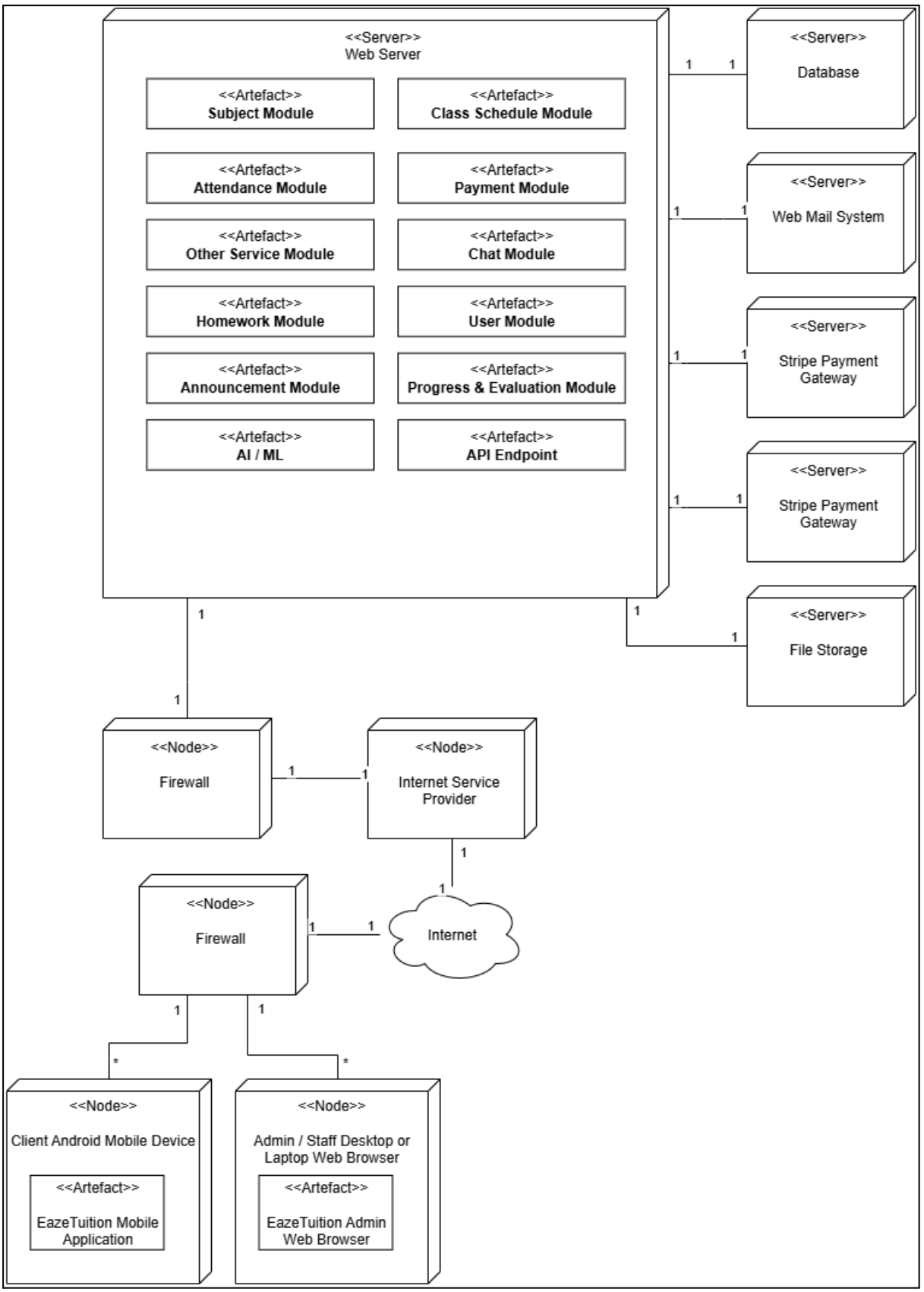


Table 4.7.1 Deployment Diagram of EazeTuition

4.8 4 x AI Algorithm

4.8.1 Dataset Source & Record

1. Student LifeStyle Dataset
    - Records: 2,000 Row
    - This dataset focuses on student's daily lifestyle habits and academic performance (GPA).
    - Source: Kaggle (Steve1215Rogg)
    - Column:

Attribute	Description
Student_ID	Unique identifier for each student
Study_Hours_Per_Day	Average number of study hours per day
Sleep_Hours	Average number of sleeping hours per night
Physical_Activity	Time spent on physical activities (e.g., exercise, sports)
Extracurricular_Hours	Hours spent on extracurricular activities
GPA (or Performance)	Academic performance measured by GPA or grade points
Stress_Level	Level of stress (measured by a scale or rating)
Social_Media_Hours	Average daily time spent on social media

Table 4.8.1 Column Detail of Student LifeStyle Dataset

2. Student Performance Factor Dataset
    - Records: Over 200 records
    - Includes learning habits, family background and exam performance
    - Source: Kaggle (LaiNguy123)
    - Column:

Attribute	Description
Hours_Studied	Number of study hours per week
Sleep_Hours	Average number of sleeping hours per night
Previous_Scores	Previous academic scores or grades

Family_Income	Household income level (Low / Medium / High)
Parental_Education_Level	Education level of parents (e.g., High School, Bachelor)
Distance_from_Home	Distance from home to school (Near / Moderate / Far)
Gender	Gender of student (Male / Female)
Exam_Score	Final examination score (numerical value)
Attendance	Attendance rate (percentage)
Tutoring_Sessions	Number of tutoring sessions per month

Table 4.8.2 Column Detail of Student Performance Dataset

4.8.2 Algorithm 1: Lifestyle Wellness Estimation (Random Forest Regression)

Purpose

Based on students' lifestyle and study-related attributes such as study hours, sleep hours and activity hours to predict whether a weekly study–rest pattern aligns with healthy lifestyle behaviour based on real student lifestyle data.

Why

The capability of students to handle academic tasks may be different. Some of them may be highly motivated under heavy coursework but the others may be depressed or lose motivation. By employing the K-means clustering algorithm, the system can automatically identify these naturally formed groups without the staff manual labelling. This helps to ensure the generated timetable can be evaluated accurately which considers each student's capacity and mental well-being.

How

1. Load and clean the dataset by lower-casing all column names, convert the categorical stress values to numeric levels and convert non-numeric fields to safety.
2. Select the required lifestyle features including study hours, sleep hours and stress level.
3. Train a RandomForestRegressor to predict stress probability (0–2).
4. During evaluation:

- Convert weekly study & rest hours → daily study / daily sleep.
- Feed into the model to predict stress likelihood.
- Convert stress likelihood to lifestyle_score equal 1 deduct normalized_stress.

Output: A lifestyle_score between 0 and 1 which the higher score represents the more balanced lifestyle and the lower score represents the potential stress or imbalance. The score will become the first component used in the timetable evaluation.

4.8.3 Algorithm 2: Academic Performance Estimation (Random Forest Hybrid Model)

Purpose

Estimate whether the weekly workload supports productive academic outcomes.

Why

The timetable which looks efficient on paper may not be equally effective for all students. For example, a "high-achieving" students are able to handle a six-hour daily study schedule, while "balanced" students may be struggling within the same schedule. This hybrid model combines ML prediction with a rule-based ideal study ratio.

How

- Load performance dataset (19 behavioural & environmental features).
- Encode categorical fields; normalize numbers.
- Train a RandomForestRegressor to predict Exam_Score
- For evaluation:
  - Construct a synthetic student profile based on weekly study & rest hours.
  - Predict expected exam score then normalize to 0–1.
- Blend with rule-based ideal ratio curve:
  - $performance\_score = 0.7 * ML\_Prediction + 0.3 * IdealStudyBalanceRule$

Output: A lifestyle_score between 0 and 1.

4.8.4 Algorithm 3: Timetable Structural Quality Analysis (Feature Extraction + Heuristic Model)

Purpose

Measure how "well-structured" the timetable is (rest gaps, stability, daily density).

Why

Even with the right study load, a badly structured timetable increases fatigue.

How

1. Extract one best class per subject in the generated package set.
2. Compute structural metrics:
    - session_count
    - subject_count
    - avg_gap / max_gap
    - density
    - room_switches
3. Use a lightweight heuristic RF model which trained in rule mode to score structure:
    - Penalize excessive session_count
    - Penalize room_switches

Output: A timetable_score between 0 and 1.

4.8.5 Algorithm 4: Combined Synergy Scoring + Rank Normalization

Purpose

Fuse multiple AI scores (performance, lifestyle, workload fitness, timetable structure) into a final evaluation with human-friendly ranking.

Why

A holistic score is needed to judge a complete timetable.

How

1. Compare weekly study hours against target ranges for Morning / Afternoon / Full time windows and compute a Gaussian fitness score.
2. Synergy score using weighted fusion:
    - $0.45 * performance$
    - $0.25 * lifestyle$
    - $0.20 * timetable$
    - $0.10 * workload$
3. Convert synergy (0–1) into a 45–95 range for clearer human interpretation:
    - $scaled\_score = 45 + normalized\_rank * 50$
4. Based on severity and AI score to do recommendation classification:
    - Highly Recommended
    - Recommended
    - Moderate
    - Not Recommended

Output: ai_score represents the score calculated, ai_recommended represents the classified class and also the ai_reason represent the classified explanation.

4.9 Chapter Summary and Evaluation

This chapter describes the **design of the EazeTuition system**, covering both structural and behavioral aspects, translating user needs into practical solutions. **Sequence diagrams** and **statecharts** illustrate processes and module interactions, while **class diagrams**, **ERDs** and **data dictionaries** define the logical and physical data design. **Flowcharts** outline the business logic, while **interface design** foregrounds the user experience.

The **deployment diagram** details the overall system architecture and outlines integration with external services such as payment gateways, file storage, webmail, and notification systems. Additionally, **four AI algorithms** which are lifestyle scoring, academic performance prediction, timetable structural analysis and a synergy-driven recommendation engine to assess and rank generated class schedules. These designs ensure that the system is fully functional, scalable and fully meets the project goals.

The system is designed to effectively translate requirements into clear models, including sequence diagrams, state diagrams, entity-relationship diagrams, and user interface prototypes. Its core strength lies in its integrated AI algorithms (K-means clustering and decision trees) to provide personalized course recommendations. Despite challenges in data preprocessing and synchronizing learning time across datasets, the overall comprehensive and user-centric design laid a solid foundation for subsequent implementation.

Chapter 5

Implementation and Testing

5 Implementation and Testing

This chapter outlines the core implementation of the system, including AI-driven schedule evaluation, controller logic, module integration, and comprehensive system testing through structured integration tests covering all functional components.

5.1 Implementation and Coding

5.1.1 AI Schedule Evaluation Invocation & Scoring Logic

To enhance the quality and realism of automatically generated timetable solutions, the controller integrates a multi-stage AI evaluation pipeline. This pipeline trains the required models, extracts structural features, generates performance and lifestyle predictions, and ultimately produces collaborative scores and recommendation tiers.

Purpose

This AI-driven scheduling mechanism ensures that every generated class schedule package undergoes automated quality assessment before submission to administrators. The controller employs trained models and customized scoring logic to identify unstable, unreasonable, or academically unsuitable schedule proposals. This approach provides administrators with reliable recommendations, effectively preventing the release of low-quality schedules.

How

1. Controller Loads and Trains the AI Models

```
lifestyle_model = ai.LifestyleModel("data/student_lifestyle.csv").train()
performance_model =
ai.PerformanceModel("data/student_performance.csv").train()
tt_extractor = ai.TimetableFeatureExtractor()
tt_model = ai.TimetableAIModel()
 tt_model.train()
```

2. Controller Extracts Timetable Features

```
tt = tt_extractor.extract_from_sessions(sessions)
timetable_score = tt_model.predict(tt)
```

3. Controller Computes Lifestyle & Performance Scores

```
        # ----- Lifestyle -----
        lifestyle_score = lifestyle_model.predict_score(
            stress_level=2,
            daily_study=total_study_hours / 6,
```

```
        sleep_time=total_rest_hours / 6
    )

    # ----- Performance (window-normalized) -----
    perf_raw = performance_model.estimate_performance(
        study_hours=total_study_hours,
        rest_hours=total_rest_hours
    )

    # Adjust performance according to selected time window
    if time_window == "M":
        performance_score = min(1.0, perf_raw + 0.10)
    elif time_window == "A":
        performance_score = min(1.0, perf_raw + 0.04)
    else:
        performance_score = min(1.0, perf_raw + 0.02)
```

4. Controller Generates the Final Synergy Score

```
     synergy = ai.calculate_synergy(
        lifestyle=lifestyle_score,
        performance=performance_score,
        timetable=timetable_score,
        workload=workload_fitness
    )
```

5. Score Stretching and Ranking

```
    scaled_scores = [45 + (s - lo) / diff * 50 for s in raw_scores]
    pkg["ai_score"] = round(float(sc), 2)
```

6. Controller Assigns Recommendation Levels & Reasons

```
    if severe_issue:
        level = "Not Recommended"
    else:
        # Separate to three level if no issue
        if score >= 88:
            level = "Highly Recommended"
        elif score >= 72:
            level = "Recommended"
        else:
            level = "Moderate"

    pkg["ai_recommended"] = level
    pkg["ai_reason"] = reasons
```

5.2 Testing

5.2.1 Testing Strategies

Integration Testing

Due to the **tightly coupled nature** of the system modules, **integration testing** was selected as the primary testing strategy for this project. This is because almost every functional module in the system **interacts with multiple datasets** including subjects, tutors, students, enrollment information, class schedules, payment records and chat messages that cannot be evaluated in isolation. Testing modules in isolation would fail to capture real-world behavior patterns and make it difficult to detect cross-module data inconsistencies. Instead of verifying functions independently, the testing may need to focus on observing **how modules behave** when **combined into realistic operational flows**.

In the **Subject** and **Class Schedule** module, many actions may **ripple outward** into other components. For example, a single subject deactivation may invalidate multiple classes, class details, time schedules and even affect enrollment availability. This is because these **behaviors depend on database links** and **cascading logic**, integration testing can help to comprehensively validate these relationships. Similarly, the **class creation, scheduling and timetable conflict detection** all rely on the tutors, timetables and also existing classes, thus requiring a combined testing to ensure the correctness.

For the **Other Service** and **Payment modules**, integration testing is used to verify the **lifecycle changes across interconnected functions**, including registration processes, attendance records, payment status, transitions, overdue verification and automated rule enforcement. These behaviors **only appear** when modules **operate at the same time**, thus using integration testing becomes the most authentic method for capturing overall system consistency.

Integration testing also benefits the **Chat Module** as the chat rooms, members or users and notification workflows **depend on the coordinated interactions**. Testing them **separately** may **miss important behavioral links** such as message visibility updates or moderation notifications.

In conclusion, integration testing aligns closely with the project architecture as the system heavily depends on chained logic, shared datasets and multi-step workflows. This approach ensures the expected outcomes hold true across the entire application runtime ecosystem by validating modules through combination rather than isolated testing,

5.2.2 Test Plan

Testing Objective

The objective of this testing phase is to validate whether all integrated modules within the EazeTuition Management System function properly when interacting with each other. The testing ensures the data flows seamlessly between modules such as Subjects, Classes and Scheduling, Other Service, Payments and Chat. Ensure that cross-module operations produce accurate and consistent results during actual system operation.

Testing Process

Integration Testing

For the EazeTuition application, integration testing is applied across all core modules. This is because each of the modules all rely on the **datasets**, **processes** and **results** that are **generated by other modules**. Integration testing not only can ensure each of the modules can complete its own functionality, but also ensure the correct operation when modules are interconnected.

Testing will follow a **dependency-based sequence**, ensuring prerequisites are validated before testing higher-level modules. The **testing sequence** will be **Subject** module, **Other Service** module, **Class** module, **Payment** module then **Chat** module:

1. Subject Module

   The testing will start from the subject module as it is the foundation of the classes and schedule packages. This module must be validated first because all subsequent modules depend on correct subject creation, activation, deactivation and fee configuration.

2. Other Service Module

   The other service module also depends only on the foundational dataset structure which is similar to thematic modules. Due to the payment functionality depends on data from other services, this module must be tested early to stabilize the workflow before testing downstream dependencies.

3. Class Module and Schedule Package

   The class module relies directly on the subject information. All the key flows such as class creation, tutor assignment, duration rules and manual or auto timetable creation will be tested after the subject module is confirmed to be stable.

The operation of the class schedule package feature highly relies on the class details, class periods and scheduling. Therefore, this feature can only be tested after class modules have been created and their details confirmed to be functioning properly.

4. Payment Module

   The payment amount depends on the subject, other services and class enrollment data. The integration testing aims to verify fee calculations, payment detail generation and consistency of cross-module enrollment data.

5. Chat Module

   The chat module functionality will be completed at last, as it relies on the system users and potentially involves registration data. Testing ensured that add new chat and message flow functions operate correctly within a fully operational environment.

Tested Items

1. Subject Module
   - Add new subject
   - Edit existing subject
   - Deactivate an active subject
2. Other Service Module
   - Add new service
   - Edit existing service
   - Deactivate an active service
   - Add new service enrollment
   - Cancel service enrollment
3. Class Module
   - Add new class with class detail
   - Edit class detail of existing class
   - Deactivate class
   - Add new schedule package
   - Deactivate a schedule package
   - Add new class enrollment
   - Cancel class enrollment
4. Payment Module
   - Make payment from admin website
   - Make payment from parent mobile site

5. Chat Module
    - Add new chatroom with active user
    - Send messages inside an existing chatroom.

Testing Environment

- **Hardware**

Desktop Device	Model: MSI Modern 15 Operating System: Windows 10 RAM: 16 GB Processor: Intel Core i7 (11th Gen) Storage: 512 GB SSD Network Connectivity: ● Wi-Fi: 2.4 GHz
Android Device	Model: Vivo Y21 Android OS Version: Android 11 RAM: 4GB Storage: 64GB Network Connectivity: ● Wi-Fi: 2.4 GHz ● Cellular: 4G LTE support

- **Web Browser**

Google Chrome	Version: Latest stable release Supported Features: JavaScript, CSS3, HTML5, ES6, WebSockets, REST API calls
Microsoft Edge	Version: Latest stable release Supported Features: JavaScript, CSS3, HTML5, ES6, WebSockets, REST API calls

5.2.3 Test Cases and Results

Subject Module

Tester's Name	Ong Yi Xin	Date Tested	27-Nov-2025	Test Case (Pass/Fail/Not Executed)	Pass

S #	Prerequisites:		S #	Test Data
1	Access to Chrome Browser		1	SubjectName="Mathematics Premium"
2	Access to Admin Panel		2	UpdatedName="Mathematics Elite"
3	Valid admin login credentials		3	NormalFee=100.00
4			4	FocusFee=150.00
5			5	MaterialFee=20.00
6			6	EducationLevelId=1
7			7	CSVFile="subjects_sample.csv"

Test Scenario	Add a new subject, edit the new added subject, deactivate the subject and import subjects via CSV and XLSX file.

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Click on "Add New Subject" button	Navigate to add subject page and add subject form is displayed	Form displayed successfully	Pass
2	Enter valid subject name and fees	System accepts input without validation error	Inputs accepted successfully	Pass
3	Click on "Add" button	Subject is created and listed in subject table	Subject created and listed	Pass
4	Click on "Edit" button of the newly added subject	Edit forms loads with existing data	Edit form displayed successfully	Pass
5	Update the subject name to "Mathematics Elite"	System accepts new input	New name accepted	Pass
6	Click on "Update" button	A confirmation message should be displayed	Confirmation message displayed successfully	Pass
7	Click on "Confirm" button of the confirmation message	System updates subject data and a success message displayed	Subject updated successfully and success message displayed	Pass
8	Click on "Deactivate" button of the updated subject in list	A confirmation message should be displayed	Confirmation message displayed successfully	Pass
9	Click on "Confirm" button of the confirmation message	Subject status will change to "0" and display a success message. All related classes and their schedules are also deactivated.	Subject successfully deactivated. All linked classes, class Details and time schedules were automatically set to “0” and success message displayed.	Pass
10	Click on "Import Subject" button	Import forms loads and displayed	Form displayed successfully	Pass
11	Upload valid CSV file	System accept file	Inputs accepted successfully	Pass

		upload without error		
12	Click on "Import" button	All subjects in file imported and success message displayed	subjects in CSV imported successfully and message displayed	Pass

Table 5.2.3.1 Test Case of Subject Module

Other Service Module

Tester's Name	Ong Yi Xin	**Date Tested**	30-Nov-2025	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	**Prerequisites:**	**S #**	**Test Data**
1	Access to Chrome Browser	1	ServiceName="Transport Service"
2	Access to Admin Panel	2	UpdatedServiceName = "Transport Service Plus"
3	Valid admin login credentials	3	ServiceType = "2"
4	Valid parent login credentials	4	FeeType="1"
5		5	MonthlyFee = 80.00
6		6	DepositFee = 50.00
7		7	StudentIdForEnrollByAdmin = 1
8		8	Address = "123 Jalan ABC, Kuala Lumpur"
9		9	StudentIdForEnrollByParent = 1
10		10	ServiceIdForEnrollByParent = 1

Test Scenario	Add a new service, edit new added service, deactivate the servcie, add new enrollment for an active student and cancel the new added enrollment. Parent add a new service enrollment for child.

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin login and click on "Add New Service" button	System navigates to Add Service page and form is displayed	Add Service form displayed successfully	Pass
2	Enter valid service name, type, fee type and values	System accepts all inputs with no validation error	Inputs accepted successfully	Pass
3	Click "Add" button	Service is created and listed in service table	Service created and listed successfully	Pass
4	Click "Edit" button on newly added service	Edit form loads with existing service data	Edit form loaded successfully	Pass
5	Update the service name to UpdatedServiceName	System accepts the new name	Updated name accepted	Pass
6	Click "Update" button	Confirmation popup displayed	Confirmation popup displayed successfully	Pass
7	Click "Confirm" in popup	Service updated and success message displayed	Service updated successfully and success message shown	Pass
8	Click "Deactivate" on updated service from service list	Confirmation popup is shown	Confirmation popup displayed successfully	Pass
9	Click "Confirm" in popup	Service status changes to “0”, success message displayed	Service deactivated successfully with success message	Pass

10	Click "Create New Enrollment" button	Enrollment form is displayed	Enrollment form displayed successfully	Pass
11	Select student, service, enter required address	System validates input and accepts values	Inputs accepted successfully	Pass
12	Click "Add" button	Enrollment is created and payment record generated accordingly	Enrollment and related payment created successfully	Pass
13	Click "Cancel" on new enrollment from enrollment list	Confirmation popup appears	Confirmation popup displayed successfully	Pass
14	Click "Confirm" in popup	Enrollment status changes to “0”. Payment is deleted if constraints allow. Success message displayed.	Enrollment cancelled, related payment deleted according to rules, success message shown	Pass
15	Parent logs in and navigates to “Enroll New Service”	Available services are displayed	Services displayed correctly	Pass
16	Parent selects a child and service to enroll	System loads service info & validates availability	Service info loaded successfully	Pass
17	Parent clicks “Enroll”	System shows fee summary and confirmation popup	Enrollment confirmation displayed	Pass
18	Parent confirms enrollment	Pre-payment pending enrollment record created	Enrollment record (pending payment) created	Pass

Table 5.2.3.2 Test Case of Other Service Module

Class Module

Tester's Name	Ong Yi Xin	**Date Tested**	28-Nov-2025	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	**Prerequisites:**		**S #**	**Test Data**
1	Access to Chrome Browser		1	SubjectId1=1
2	Access to Admin Panel		2	EducationLevelId=1
3	Valid admin login credentials		3	TutorId=1
4	Access to User Site (Parent Portal)		4	ClassType="N"
5	Valid parent login credentials		5	UpdatedClassType="F"
6			6	AutoSchedule=false
7			7	IncludeInSchedule=true

8			8	TimePerWeek=2
9			9	Duration=90
10			10	Day1 = "1", StartTime1="13:00"
11			11	Day2 = "2", StartTime2="12:00"
12			12	SubjectIdForSchedule1=1
13			13	SubjectIdForSchedule2=2
14			14	PackageName="Primary 2 2026 Package"
15			15	EducationLevelIdForSchedule=1
16			16	RegistrationStatus=true
17			17	NumberOfSetGenerate=2
18			18	IntakeYear=2026
19			19	TimeWindow="M"
20			20	StudentIdEnrollByAdmin=1
21			21	ClassDetailIdForEnrollByAdmin=1
22			22	PaymentMethod="2"
23			23	StudentIdEnrollByParent=2
24			24	ClassDetailIdForEnrollByParent=1

Test Scenario	Admin add a new class, edit the new added class, deactivate the new added class, add a new schedule package including AI evaluation, deactivate the new added schedule package, add a class enrollment for an active student and cancel the new added enrollment. Parent add a class enrollment for child.

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Admin login and click on "Add New Class" button	Add Class form appears	Add Class form displayed successfully	Pass

2	Select subject, tutor, class type, time per week and duration	System accepts input without validation errors	Inputs accepted successfully	Pass
3	Add day and time pairs	System validates time inputs and prevents conflicts	Time pairs input accepted successfully with no conflict	Pass
4	Click "Create Class" button	New class is created with class details generated and display sucess message	Class and class details created successfully. Success message displayed	Pass
5	Click "Edit" button on newly added class record from list	Edit form loads with existing class data	Edit form loaded successfully	Pass
6	Update class type to UpdatedClassType	System accepts the new class type	New class type accepted	Pass
7	Click "Update" button	Confirmation message displayed	Confimation message displayed successfully	Pass
8	Click "Confirm" on confirmation popup	Class data updated and success message displayed	Class updated successfully and success message shown	Pass
9	Click "Deactivate" button on the updated class record from list	Confirmation popup appears	Confirmation popup displayed successfully	Pass
10	Click on "Confirm" button on the confirmation popup	Class, all related class detail and all related time schedule change status to "0" and success message displayed	Class, class_details and time_schedules all deactivated successfully with success message	Pass
11	Click "Add Schedule Package" button	Schedule Package form displayed	Schedule Package form displayed successfully	Pass
12	Enter packageName, select education level, subject, time window, number of package genrated, intake year and checked the registration status checkbox	System accepts input without validation errors	Inputs accepted successfully	Pass

13	Click on "Generate Package" button	System generates timetable sets, evaluates each using AI, displays AI scores and recommendations	Timetable sets generated and display successfully together with AI evaluation result	Pass
14	Select desired timetable set	System accepts selection	Selection accepted successfully	Pass
15	Click on "Add" button	Package is created and shown in list	Schedule package saved and listed successfully	Pass
16	Click "Deactivate" button on the new added schedule pacakge record from list	Confirmation popup appears	Confirmation popup displayed successfully	Pass
17	Click on "Confirm" button on the confirmation popup	Package status changes to "0" and all time schedule record from package status changes to "0"	Package and linked schedules deactivated successfully	Pass
18	Click on "Create New Enrollment" button	Enrollment form is displayed	Enrollment form displayed successfully	Pass
19	Select student and class to enroll	System check with conflicts and accept input without error	Inputs accepted successfully	Pass
20	Click on "Add" button	Enrollment created and related payment generated	Enrollment and payment created sucessfully	Pass
21	Click on "Cancel" button of the new added enrollment from list	Confirmation popup appears	Confirmation popup displayed successfully	Pass
22	Click on "Confirm" button on the confirmation popup	Enrollment status set to “0”, payment removed if constraint conditions met, success message shown	Enrollment cancelled, payment removed correctly and success message shown	Pass
23	Parent logs in and navigates to “Enroll New Class”	Available classes are displayed	Classes displayed correctly	Pass

24	Parent selects a child and class to enroll	System loads class info & validates availability	Class info loaded successfully	Pass
25	Parent clicks “Enroll”	System shows fee summary and confirmation popup	Enrollment confirmation displayed	Pass
26	Parent confirms enrollment	Pre-payment pending enrollment record created	Enrollment record (pending payment) created	Pass

Table 5.2.3.3 Test Case of Class Module

Payment Module

Tester's Name	Ong Yi Xin	**Date Tested**	31-Nov-2025	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	**Prerequisites:**		**S #**	**Test Data**
1	Access to Chrome Browser		1	StudentId=1
2	Access to Admin Panel		2	PaymentId1=1 (PaymentMethod="1")
3	Valid admin login credentials		3	PaymentId2=2 (PaymentMethod="3")
4	Parent account with pending payments		4	ReceivedAmount=200.00
5	Stripe test environment enabled		5	PaymentId3=3 (PaymentMethod="2")
6	A pending payment generated from a class or service enrollment exists (Enrollment becomes permanent only after successful payment)		6	StripeCard="4242424242424242"
7			7	Expiry="12/30"
8			8	CVC="567"

Test Scenario	Admin creates two different types of payments (Payment Method 3 and Payment Method 1). Parents complete online payment using Stripe.

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin navigates to “Make Payment” page	Payment form is displayed	Payment form displayed successfully	Pass
2	Admin selects Student, PaymentId1 (PaymentMethod="3)	System aceepts selection without error	Inputs accepted successfully	Pass
3	Admin clicks "Add" button	Payment created immediately and success message displayed	Payment created successfully with success message	Pass
4	Admin selects Student, PaymentId2 (PaymentMethod="1")	Additional field “Received Amount” becomes visible	Received Amount field displayed successfully	Pass
5	Admin enters ReceivedAmount	System calculates and updates ChangeAmount	Change amount calculated correctly and updated	Pass

6	Admin clicks "Add" button	Payment recorded with correct change amount and success message shown	Payment saved correctly with accurate change value and success message displayed	Pass
7	Parent logs in and navigates to “Home”	List of pending payments displayed	Pending payments displayed correctly	Pass
8	Parent clicks “Make Payment”	System loads and displays selected payment details	Payment details loaded successfully	Pass
9	Parent selects PaymentMethod “2” (Stripe) and clicks “Proceed”	Stripe Checkout page is displayed	Stripe checkout page displayed successfully	Pass
10	Parent enter card number, expiry and CVC	System validates card info	Card info accepted	Pass
11	Parent clicks “Pay”	Stripe returns “Payment Success” and updates payment record	Payment succeeded and updated payment displayed	Pass

Table 5.2.3.4 Test Case of Payment Module

Chat Module

Tester's Name	Ong Yi Xin	**Date Tested**	1-Dec-2025	**Test Case (Pass/Fail/Not Executed)**	Pass

S #	**Prerequisites:**		**S #**	**Test Data**
1	Access to Chrome Browser		1	AdminUserId=1
2	Access to Admin Panel		2	NormalUserId = 5
3	Valid login credentials for Admin & Users		3	SearchKeyword = "John"
4	Active chat module enabled		4	MessageContent1 = "Hello, this is the admin test message."
5			5	MessageContent2 = "Hello, this is the user test message."
6			6	

Test Scenario	Create a new chat room, send a message and verify message retrieval with unread tracking.

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Admin logs in and navigates to “Chat” page	Chat room list is displayed	Chat room list displayed successfully	Pass
2	Click “Create New Chat Room”	User selection popup with search bar appears	User selection popup displayed successfully	Pass
3	Admin enters SearchKeyword in the search bar	System returns matching user list	Matching users returned successfully	Pass
4	Admin selects a user from search results	A new chat room is created or existing room is opened	Chat room created or opened successfully	Pass
5	Admin types MessageContent1 and clicks “Send”	Message is saved and shown in chat window	Message stored and displayed successfully	Pass

6	Admin selects a user from search results	Chat tab shows unread message indicator	Unread indicator displayed correctly	Pass
7	User opens the chat room	All messages load, message marked as “Read”	Messages displayed and status updated to Read	Pass
8	User replies with a MessageContent2	Message saved and shown in chat window	Reply message displayed successfully	Pass

Table 5.2.3.5 Test Case of Chat Module

5.3 Chapter Summary and Evaluation

This chapter comprehensively outlines the **full implementation process** of the EazeTuition system, detailing the development of core modules including subject management, class configuration, AI-powered timetable package generation, other services, payment processing, and personal chat functionality. It also explains the **integration approach** for the **AI timetable scoring mechanism**, outlining how the controller trains models, extracts timetable features, calculates lifestyle and performance scores and ultimately generates recommendation tiers for timetable packages.

This chapter further documents the **integrated testing process** adopted by the project. Given the tight coupling of system modules, the testing team conducted cross-module integration **tests across all modules** to validate real data flows, dependency chains and cross-module interactions. Detailed **test plans** and comprehensive **test cases** were executed on actual hardware devices and browsers, covering both web-based which is the admin site and mobile which is the parent site environments.

Overall, the evaluation **confirmed** that all modules **functioned correctly** in combination together with cascading rules performing as expected. AI scoring integrated seamlessly with the scheduling workflow and the process from registration to payment to confirmation operated consistently throughout. The successful execution of all test cases demonstrated the system's stability and reliability, confirming it is ready for deployment.

Chapter 6

Discussions and Conclusion

6 Discussions and Conclusion

Summary

This project aims to address operational challenges faced by tuition centers such as **inefficient manual scheduling**, **fragmented communication**, **rigid service processes** and **error-prone payment processes**. EazeTuition achieves optimized academic planning workflows, enhanced communication clarity and improved overall administrative efficiency by introducing **automated and AI-assisted solutions** across subject management, class scheduling, other services, payment processing and chat modules.

To resolve these problems, the system introduced **structured class creation**, **automated generation of class details**, **rule-based scheduling** and **AI-driven timetable evaluation**, ensuring balanced learning schedules while minimizing manual scheduling errors. **Class and service management** is enhanced through **flexible creation**, **bulk import**, **controlled activation** and **integrated fee processing**. The payment module delivers **transparent fee calculations**, **multi-channel payment suppor**t and **automated receipt generation** eliminating manual accounting entirely. The chat module significantly boosts center-parent communication efficiency by providing a secure, **centralized messaging channel** with tracking capabilities.

The choice of tools is based on the need for scalability, **fast development** and **compatibility across web and mobile platforms** which are selected using Python (Flask), MySQL, React Native, JavaScript and Stripe. **Python** provided **reliable backend logic** for complex scheduling and financial rules, **MySQL** supported **structured relational data** for academic and payment records and **React Native** enabled a **unified mobile application** for tutors, parents and students.

Development has been selected to use the **incremental model** which enables modules to be **built, tested and optimized in phases**. This is because of the **project team's limited size** and **tight timeline**. This approach was adopted to deliver core functionality early while progressively integrating complex features such as payment processing and AI evaluation. This iterative methodology also ensures continuous feedback, effectively mitigating the risk of system-level failures.

Overall, the system has successfully achieved its goal of providing an automated, intelligent and centralized management platform for tuition centers. The project has also delivered significant innovations through an AI-driven scheduling system, multi-level conflict detection, automated financial processing and secure structured communication. Both the technical decisions and methodological choices were based on sound rationale.

Achievements

Objective 1: Intelligent Scheduling and Academic Planning

Subject Module Achievements

- Flexible academic planning is achieved through the creation of subjects, multi-tiered tuition configurations (normal or focus), batch import capabilities and duplicate verification functions. This directly supports scalable academic settings while reducing administrative workload.

Class Module Achievements

- Automated scheduling supports creating classes with or without time slots, the system automatically assigning times based on availability.
- Conflict avoidance scheduling is fully implemented: covering tutor conflict detection, student conflict detection within timetable set, time slot overlap prevention and classroom conflict identification.
- AI timetable evaluation ranks scheduling proposals using lifestyle models, performance models and workload models to enhance scheduling quality.
- Cascading management ensures that when subjects or classes are deactivated, related courses, course details and timetables are simultaneously invalidated.

Strengths

- Highly automated, reducing manual scheduling time and minimizing operational inconsistencies.

Weakness

- AI assessment can be further expanded by incorporating real student datasets to enhance accuracy.

Objective 2: Centralized and Secure Communication

Chat Module Achievements

- Achieves fully centralized one-to-one messaging between parents and administrators.
- Messages are stored with read or unread status indicators to enhance accountability in communication.
- Replaces fragmented WhatsApp communication with structured, traceable in-system messaging.

Strengths

- Clear message history, secure communication, enhanced engagement.

Weaknesses

- Does not yet support real-time WebSocket messaging, currently relies on polling mechanisms.

Objective 3: Streamlined Payments and Service Integration

Other Service Module Achievements

- Implemented service registration for childcare and transport with validation such as required address for transport.
- Every confirmed service enrollment automatically generates a pending payment record.
- Supports enrollment cancellation with automated payment removal following business rules.

Payment Module Achievements

- The automatic payment generation feature triggered by class or service registration completely eliminates the need for manual invoicing.
- The dynamic fee calculation system handles class-type fees, service charges, deposit schemes and discounts based on the number of enrolled courses.
- Monthly fees automatically generate based on active enrollments.
- Supports online and offline payments, including Stripe settlement, change calculation and e-receipt generation in PDF format.
- An automated status update mechanism ensures enrollments are confirmed only after successful payment.

Strengths

- High accuracy, complete transparency, meets parents' demand for convenience.

Weaknesses

- Future enhancements could include advanced features like installment payments or automatic deductions.

Contributions

Creativity and Innovativeness of the System

1. **AI-Enhanced Timetable Evaluation and Recommendation**

   One of EazeTuition's most innovative components is its AI-driven timetable evaluation pipeline. While traditional tuition center systems focus only on rule-based scheduling, this project introduces a multidimensional scoring system which covering lifestyle patterns, academic performance predictions and learning load compatibility to intelligently rank timetable options.

   This innovation transforms scheduling from a purely administrative task into a data-driven decision support system, ultimately generating more balanced timetables that better align with student needs.

2. **Multi-Layered Conflict Detection and Auto Assignment**

   The scheduling module incorporates highly innovative design, when administrators do not define time slots, the system automatically allocates time. Simultaneously, it detects conflicts among instructors, students and classrooms and combines capacity checks to achieve automatic classroom assignment.

   This multi-constraint validation is uncommon in traditional education management systems, demonstrating exceptional system intelligence that transcends basic CRUD operations.

3. **Signature-Based Schedule Integrity Protection**

   A unique technological contribution lies in the implementation of an encrypted signature verification mechanism. When scheduling class packages, this ensures data packets remain fully intact throughout the entire transmission process from backend to frontend during the scheduling workflow.

   This not only prevents data tampering but also guarantees the reliability of decision-making. This advanced mechanism is exceptionally rare in traditional tuition applications.

4. **Automated Monthly Payment Generation and Cross-Module Fee Integration**

   The payment module incorporates automated workflows that generate recurring monthly invoices based on active enrollments.

   The dynamic billing engine automatically applies class-quantity discounts and recalculates amounts upon enrollment changes, demonstrating a level of automation and financial intelligence rarely seen in standard education platforms.

5. **Automated Lifecycle Handling Through Cascading Logic**

This system innovatively implements cascading deactivation rules. When a subject, class or schedule package is deactivated, all associated class details and scheduling arrangements are automatically updated and become deactivated.

This mechanism prevents isolated data, ensures data consistency and reduces administrative errors, significantly enhancing the robustness of operational workflows.

Why the Proposed System is Necessary

1. **Addresses Inefficiencies in Manual Scheduling and Communication**

   Traditional tutoring centers heavily rely on social media messages such as WhatsApp and manual scheduling. These methods lead to frequent class conflicts, miscommunication with parents, and inconsistent record management.

   EazeTuition replaces these fragmented workflows with structured modules, centralizing all scheduling, communication, and fee management to create a smoother, more reliable operational environment.

2. **Reduces Administrative Burden and Minimizes Human Errors**

   Manual fee calculations and class management are prone to errors, impacting both parents and institutions. This system automates these processes, including fee calculations, monthly invoice generation, multiple service billing models and payment creation linked to enrollment.

   These enhancements are crucial for modern tutoring institutions, enabling scalable growth without increasing staff workload.

3. **Provides Transparency and Trust for Parents**

   Parents often struggle to stay informed about their children's enrollment status, payment records, latest course schedules and administrative notices.

   EazeTuition addresses this need by providing a unified app where parents can instantly access all information, thereby reduce uncertainty and improve home-center communicate

Marketability of the System

1. **High Demand Among Tuition Centers in Malaysia and Southeast Asia**

   Most tuition centers still rely on manual management or limited tools like Excel and WhatsApp.

   Centralized systems with intelligent scheduling, service integration and digital payment capabilities hold significant commercial appeal, particularly for mid-sized tutoring centers unable to afford enterprise-level software.

2. **Competitive Advantage Through AI and Automation**

   Existing tuition management systems rarely integrate AI-powered timetable evaluation, multi-constraint automated scheduling, cascading deactivation logic, security signature-based verification mechanisms or automated monthly billing functionality.

   These unique features give EazeTuition a distinct competitive edge in the market, making it the ideal choice for educational institutions seeking modern upgrades that combine premium quality with practical value.

3. **Excellent Fit for Subscription-Based Business Models**

   This system architecture integrates a web-based administrative backend with a user (tutor, parent, student) mobile app, supporting multi-center deployment. It operates on a recurring subscription model with additional modules such as other services management and AI scheduling. This enables commercialization through multiple channels, including SaaS (Software as a Service) and tiered pricing plans (Basic Edition, Professional Edition, AI Edition).

4. **Scalability for Expansion Beyond Tuition Centers**

   Based on a modular design, EazeTuition system can be scaled to expand into learning centers, training academies and daycare and after-school programs.

   This significantly enhances long-term market potential, positioning EazeTuition as a versatile management platform for educational centers.

Limitations and Future Improvements

Identify the limitations of the research or project. Provide suggestions for improvement or further development of the system or research in the future.

Describe the incomplete parts of ur modules for further enhancements

1. Real-Time Communication Enhancement

In EazeTuition System, although the current chat module supports secure one-to-one messaging and read receipts, it still relies on periodic polling rather than fully real-time communication. This means that while functionality remains stable, messaging may experience slight delays compared to modern instant messaging platforms.

Future enhancements can be achieved by integrating real-time communication technologies based on WebSockets such as Socket.IO or Firebase Cloud Messaging. This will enable push notifications, instant message updates and a smoother conversational experience, creating a communication system comparable to commercial-grade mobile applications.

2. AI Scheduling Model Expansion and Accuracy Improvement

The AI-driven class schedule evaluation feature provides meaningful recommendations based on life balance, academic performance prediction and workload distribution. However, the model in this project currently relies on pre-set weights and manually adjusted scoring rules, limiting its personalized adaptation for diverse student groups.

Future enhancements could involve training adaptive machine learning models using real student behavior data, enabling the system to learn patterns over time. Incorporating additional parameters such as commute distance, extracurricular schedules, or instructor teaching styles would further optimize scheduling quality, generating more personalized data-driven recommendations.

3. Scalability of Payment Automation and Billing Models

The current payment module of EazeTuition automatically generates fees based on enrollment status, supporting monthly billing, tiered discounts and deposit-based services. However, the system does not yet support more complex billing models such as installment plans, early bird discounts, pro-rated monthly fees for mid-semester enrollment or automatic late payment penalties.

As training institutions scale operations or adopt more sophisticated financial management models, enhancing the billing engine to accommodate diverse fee structures will significantly boost the system's adaptability and market competitiveness. Integration with accounting platforms like Xero or QuickBooks can also streamline financial reconciliation processes.

4. Infrastructure Performance, Caching and Multi-Branch Support

Although the EazeTuition system operates reliably within a single tuition center, its architecture does not yet support multi-branch or multi-location management, meaning all operational data is managed within a single-tenant structure. As more tuition centers transition to multi-branch operations, future enhancement directions could include building a multi-tenant architecture, implementing caching mechanisms such as Redis to improve performance during peak loads and optimizing database indexes for large-scale data growth. These improvements will enhance the system's scalability and broaden its market appeal, enabling it to serve educational chains with thousands of students.

Issues and Solutions

1. Complexity of AI Algorithm Design and Machine Learning Implementation

One of the most challenging aspects during the development of the AI timetable evaluation component was transforming the existing dataset into meaningful machine learning outputs that could be seamlessly integrated into the scheduling module. The dataset contained numerous columns unrelated to evaluation, making the selection of attributes suitable for lifestyle and performance scoring the primary hurdle.

Beyond dataset filtering, integrating multiple AI outputs such as stress level predictions, workload balance assessments, density evaluations and classroom swap penalty values into a unified recommendation score required meticulous design of the model architecture and scoring logic. Seamlessly incorporating the trained model into the scheduling workflow without compromising performance presented another significant technical challenge.

Solution

To resolve this issue, the dataset was first analyzed column by column to identify attributes with substantive relevance. The problem was decomposed into the following subtasks:

- Validating the accuracy of the baseline scores derived from AI model training,
- Independently implementing additional scoring logic such as class density and classroom swapping,
- Finally integrating these components into a weighted composite score.

After individually validating each score, the complete evaluation logic was integrated into the scheduling package's generation function. This incremental approach ensured correctness, minimized errors and enabled the AI evaluation module to operate seamlessly within the full scheduling workflow.

2. Stripe Mobile Payment Integration and Testing Constraints

Implementing online payments with Stripe in mobile applications presented another significant challenge. Stripe's test mode functions smoothly in browser-based systems, but mobile applications rely on external callbacks and webhooks to complete the payment process. Since Stripe typically requires production-environment domains or live server endpoints for webhook communication, running all components locally on development

devices causes payment confirmations to fail. This limitation makes end-to-end testing extremely difficult during early development stages.

Solution

To resolve this issue, the Stripe test webhook listener was configured using the following command "stripe listen --forward-to localhost:5000/payment/stripe/webhook".

This command creates a temporary public tunnel, enabling Stripe's test mode to forward webhook events to the local Flask backend. The system can then simulate real payment flows using Stripe test cards. With this configuration, the mobile app successfully initiated payment sessions and received accurate payment completion callbacks, achieving the goal of conducting full local testing without deploying to production.

3. Local Data Storage Challenges with SQLite in Mobile App

Developing the EazeTuition mobile application requires using SQLite for local data storage. However, implementing SQLite on mobile devices introduces significant complexities, including data synchronization, table structure design, field filtering and ensuring that parents' or tutors' devices do not download any sensitive information restricted to administrator access. Unlike the administrator web system that displays the complete dataset, the mobile application only needs to store user-specific data. This necessitates restructuring the table design to eliminate redundant foreign keys or relationships. Designing synchronization logic that prevents data inconsistencies while safeguarding confidential information has become a major technical challenge.

Solution

The solution involves redesigning the local database structure specifically for mobile usage. It no longer replicates the full MySQL relational schema, retaining only essential fields. Primary and foreign keys are simplified or removed as mobile applications require only lightweight references to display functionality.

During synchronization, the mobile app calls a backend controller, which returns JSON results pre-filtered based on user identity and permissions. This ensures only locally relevant data is stored and sensitive or administrator-level data is never downloaded. All synchronized records remain consistently up-to-date with the server's latest state. Through this approach, the mobile application achieves secure and precise offline caching while eliminating risks of data leakage or structural conflicts.

4. Highly Complex Interdependencies Between Subjects, Classes, Time Slots and Schedule Package Generation

One of the most challenging aspects of the system is the cascading logic involved in creating class and schedule packages. Unlike simple CRUD operations, this process involves complex interactions across multiple layers, which are subject availability, course types, instructor availability, classroom constraints, time window rules and scheduling rules at the package level. The difficulty extends beyond deactivation logic, the true challenge lies in ensuring every generated class detail, auto-assigned time slot, AI-scored recommendation and class package proposal maintains internal consistency. This prevents hidden time conflicts, duplicate classroom assignments or invalid course package structures. The sheer scale of interactions renders this feature logically intricate and prone to edge cases.

Solution

To resolve this challenge, the entire scheduling and timetable generation logic was decomposed into detailed business rules prior to implementation. All necessary constraints such as grade alignment, cross-class conflicts, advisor availability windows, minimum class requirements per subject, and time window rules were clearly documented and enumerated. The workflow was then divided into a few phases such Step 1, Step 2 and so on, ensuring the system processes data in a deterministic sequence.

Each step was independently implemented and validated before being integrated into the scheduling engine. A final validation process was designed to compare generated results against the rule set, ensuring no constraints were overlooked. This structured approach enabled developers to effectively manage complexity and maintain logical accuracy throughout the scheduling process.

Unsolved Technical Problems for Future Enhancement

- Real-Time Communication Using WebSockets
- Integration of Touch ‘n Go eWallet Payment Gateway
- Fully Automated Class Replacement & Intelligent Rescheduling

Conclusion

The development of the EazeTuition system has **successfully achieved** the **core objectives**. Through functional modules such as intelligent scheduling, structured academic management, integrated service enrollment, centralized communication and a modern payment ecosystem, it effectively **resolves longstanding operational challenges** faced by tuition institutions which include particularly manual scheduling conflicts, fragmented communication and inconsistent payment processes. The system **encompasses modules** for **subject management**, **course scheduling**, **supplementary services**, payment settlement and online consultation. It comprehensively optimizes the efficiency of teaching resource allocation and significantly enhances the quality of educational services.

This project has demonstrated outstanding results in automated precision, **AI-assisted decision-making**, **conflict-free timetable generation**, **secure data processing** and **multi-module integration**. These innovations have significantly **reduced administrative workload** while enhancing transparency and user experience for tutors, parents and students.

Despite these achievements, several areas remain ripe for **further optimization** like incorporating **WebSocket real-time communication capabilities**, integrating **local Malaysian payment gateways** such as the Touch 'n Go e-wallet and **performing deep optimizations** to the scheduling engine. These enhancements represent natural extensions of the system's existing functionalities.

In conclusion, this project has established a robust and scalable foundation for future academic management platforms, demonstrating both the feasibility and impact of integrating automation, AI-powered assessment and centralized communication into a unified teaching center management system.

References

DOSM. "Department of Statistics Malaysia." Dosm.gov.my, MINISTRY OF ECONOMY DEPARTMENT OF STATISTICS MALAYSIA OFFICIAL PORTAL, 18 Dec. 2024, www.dosm.gov.my/portal-main/release-content/statistics-on-women-empowerment-in-selected-domains-malaysia-2024. Accessed 7 July 2025.

"The Impact of Digital Technology on Education in 25 Stats." Impactmybiz.com, Impact Networking, LLC, 2 Sept. 2020, www.impactmybiz.com/blog/impact-of-digital-education-on-technology-25-stats/?utm_source=chatgpt.com. Accessed 7 July 2025.

Velasquez, Jose. "5 Challenges Schools Face Managing Tuition Payments Manually - FACTS Management." FACTS Management, FACTS, 16 Dec. 2024, factsmgt.com/blog/challenges-schools-face-managing-tuition-payments-manually/?utm_source=chatgpt.com. Accessed 7 July 2025.

Sanner, M. F. (1999). Python: a programming language for software integration and development. J Mol Graph Model, 17(1), 57-61.

Walia, Anish Singh. "Best Python Libraries for Machine Learning in 2025." Digitalocean.com, DigitalOcean, 20 Mar. 2025, www.digitalocean.com/community/conceptual-articles/python-libraries-for-machine-learning. Accessed 9 July 2025.

"What Are the Pros & Cons of Using Python for Backend Development." Hire Python Expert, 2025, www.hirepythonexpert.com/blog/what-are-the-pros-cons-of-using-python-for-backend-development?utm_source=chatgpt.com. Accessed 8 July 2025.

Hadley, Linda. "Top 10 vs Code Extensions Every Developer Should Use in 2025." Tech Research Online, 27 June 2025, techresearchonline.com/blog/best-vscode-extensions-developers/. Accessed 9 July 2025.

Eisenman, Bonnie. "Learning React Native." Google Books, O'Reilly Media, Inc., 1 Dec. 2015, books.google.com.my/books?hl=en&lr=&id=274fCwAAQBAJ&oi=fnd&pg=PR9&dq=React+Native&ots=tHui8Ih4lZ&sig=Samuwj-eUU_WrkiNZ21_Eanywdo&redir_esc=y#v=onepage&q=React%20Native&f=false. Accessed 8 July 2025.

eSparkBiz. "ESparkBiz." ESparkBiz Technologies Pvt Ltd, 17 Jan. 2025, www.esparkinfo.com/software-development/technologies/reactjs/react-with-flask. Accessed 8 July 2025.

Skuza, Bartosz, et al. "Flutter vs React Native – Which Is Better for Your Project?" Droids on Roids, Droids On Roids, 14 Sept. 2023, www.thedroidsonroids.com/blog/flutter-vs-react-native-comparison. Accessed 8 July 2025.
"Performance Overview · React Native." Reactnative.dev, Meta Platforms, 17 June 2025, reactnative.dev/docs/performance. Accessed 8 July 2025.

Samonte, M. J. C., & Samonte, S. A. Evaluation of Database Management Systems and Techniques Applied in Distributed Systems.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.

scikit-learn. "1.10. Decision Trees — Scikit-Learn 0.22 Documentation." Scikit-Learn.org, scikit-learn developers, 2025, scikit-learn.org/stable/modules/tree.html. Accessed 9 July 2025.

Farid, Ashraf. "Building Apps with Expo & React Native: Pros & Cons." Upstackstudio.com, 20 Feb. 2024, upstackstudio.com/blog/expo-react-native/. Accessed 10 July 2025.

Hutri, Hugo. "COMPARISON of REACT NATIVE and EXPO." LUTPub, 2023.

Sarker, Iqbal H., et al. "A Survey of Software Development Process Models in Software Engineering." International Journal of Software Engineering and Its Applications, vol. 9, no. 11, 30 Nov. 2015, pp. 55–70, pdfs.semanticscholar.org/6274/c4f2ab1aaa00be03a59356565dfff77b6a5b.pdf, https://doi.org/10.14257/ijseia.2015.9.11.05. Accessed 23 July 2025.

Appendices

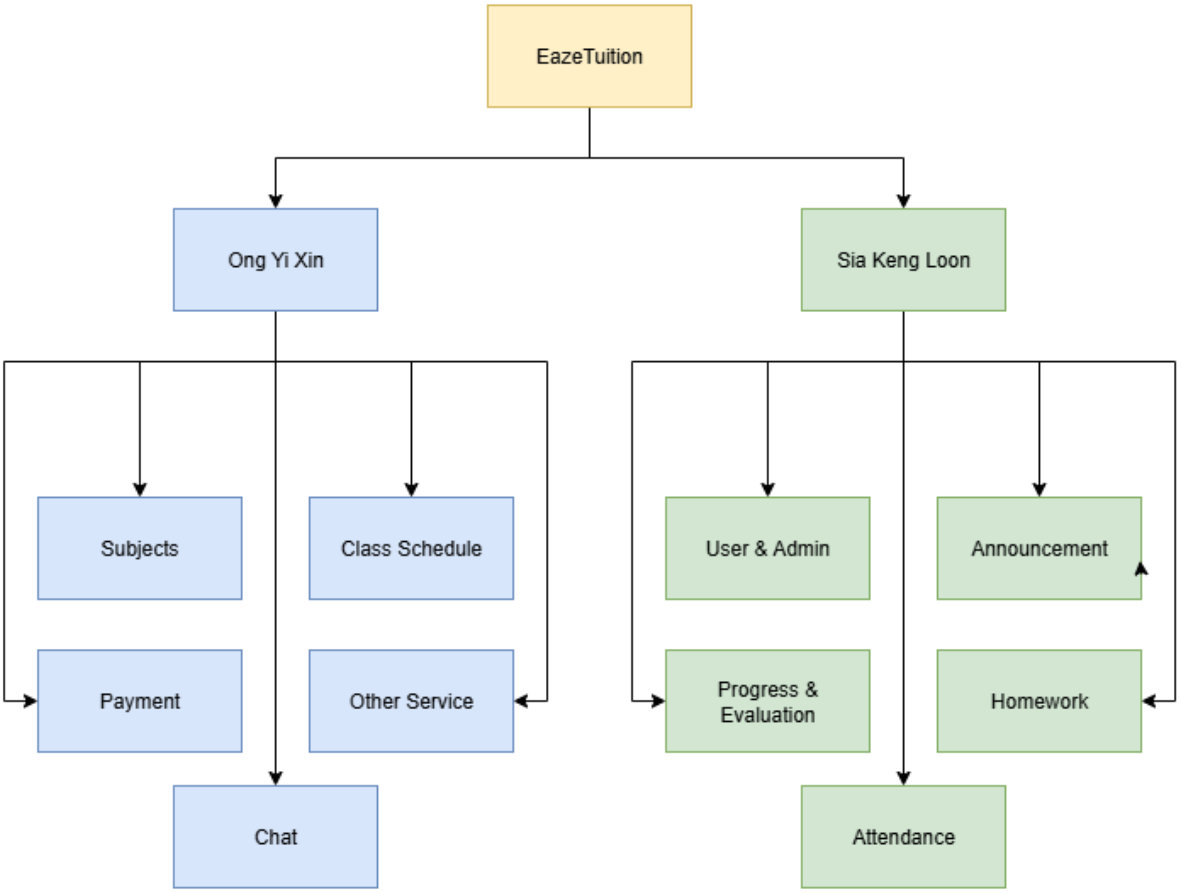


Figure 1.4.1 Milestone of EazeTuition Project

User Guide

Account For Login To Different Role After System Start

Role	Username / Email	Password
Admin (Web)	Kengloon	12345
Parent (Mobile App)	kid2dingzed@gmail.com	12345
Student (Mobile App)	henry.lim@example.com	12345
Tutor (Mobile App)	kelly.wong@example.com	12345

Table 1.2.1 Table of Account Information

Developer Guide

Full Step for First Time Start System

Step	Description
1. Install Python & Pip Dependencies	Ensure Python 3.10+ is installed. Then navigate to the backend folder and install required packages:
2. Install Node.js & Expo CLI	Install Node.js (v18+) and NPM. Then install Expo CLI globally:
	Commands: `npm install -g expo-cli`
3. Install Mobile App Dependencies (First Time Setup)	Before running `npx expo start -c`, all required mobile dependencies must be downloaded.

	Commands:
	`cd EazeTuition`
	`npm install`

	This installs React Native, Expo packages, UI libraries, and all mobile-side dependencies.
4. Install Additional Tools (Windows PowerShell Compatibility)	To bypass Windows execution restrictions (only required if you see permission errors)
	Command: `Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass`
5. Start XAMPP	Start Apache and MySQL from the XAMPP Control Panel. Ensure MySQL runs on port 3306 and the database is imported correctly.
6. Configure Backend Environment	Create your `.env` file based on the provided template. Ensure DB credentials, Stripe keys, and server URLs are correct.
7. Start Backend Server	Start the Flask server with a new terminal.
	Commands: `python app.py`
8. Start Expo Mobile App	After all dependencies are installed, start the mobile app in development mode:
	Commands: `npx expo start -c`
	Scan the QR code using Expo Go on your mobile device (Android) to launch the app.

Table 1.3.1 Table of Start System Information

Command for Start System After Finish First Time Setup

Terminal	Command
1 (Run python backend)	python app.py
2 (Run Expo Go)	cd EazeTuition Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass npx expo start -c
3 (Stripe Webhook)	stripe login --api-key sk_test_51SRpYQBBVMH0sIbu6jURiYubRrdvCoZUHoAaprqnenU0boKeApfwiVaB20RCS1yo4Urz8OKHPcTmpssyyDYmlLhR00AFLroMrc stripe listen --forward-to localhost:5000/payment/stripe/webhook

Table 1.3.1 Table of Start System Information

This page is intentionally left blank to indicate the back cover. Ensure that the back cover is black in color.