EazeTuition: 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

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 Manual scheduling can lead to scheduling confusion and miscommunication, while outdated payment systems often result in delays and errors.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.

Key features include **AI-powered automatic scheduling**, **centralized chat**, **flexible subject and class management**, and automatic fee calculation. **Other service modules** support other services such as transportation and childcare, and provide flexible billing methods so that parents and tuition centers can have more services. 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 structured design and verification during the build process. Testing included unit tests for backend logic, integration tests to ensure proper communication between components, and system tests to verify functionality and performance using sample data.

In conclusion, EaseTuition delivers with the main goal of simplifying operations, reducing human errors, and improving communication. The expected outcome is a more efficient, transparent, and user-friendly experience for staff and parents.

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

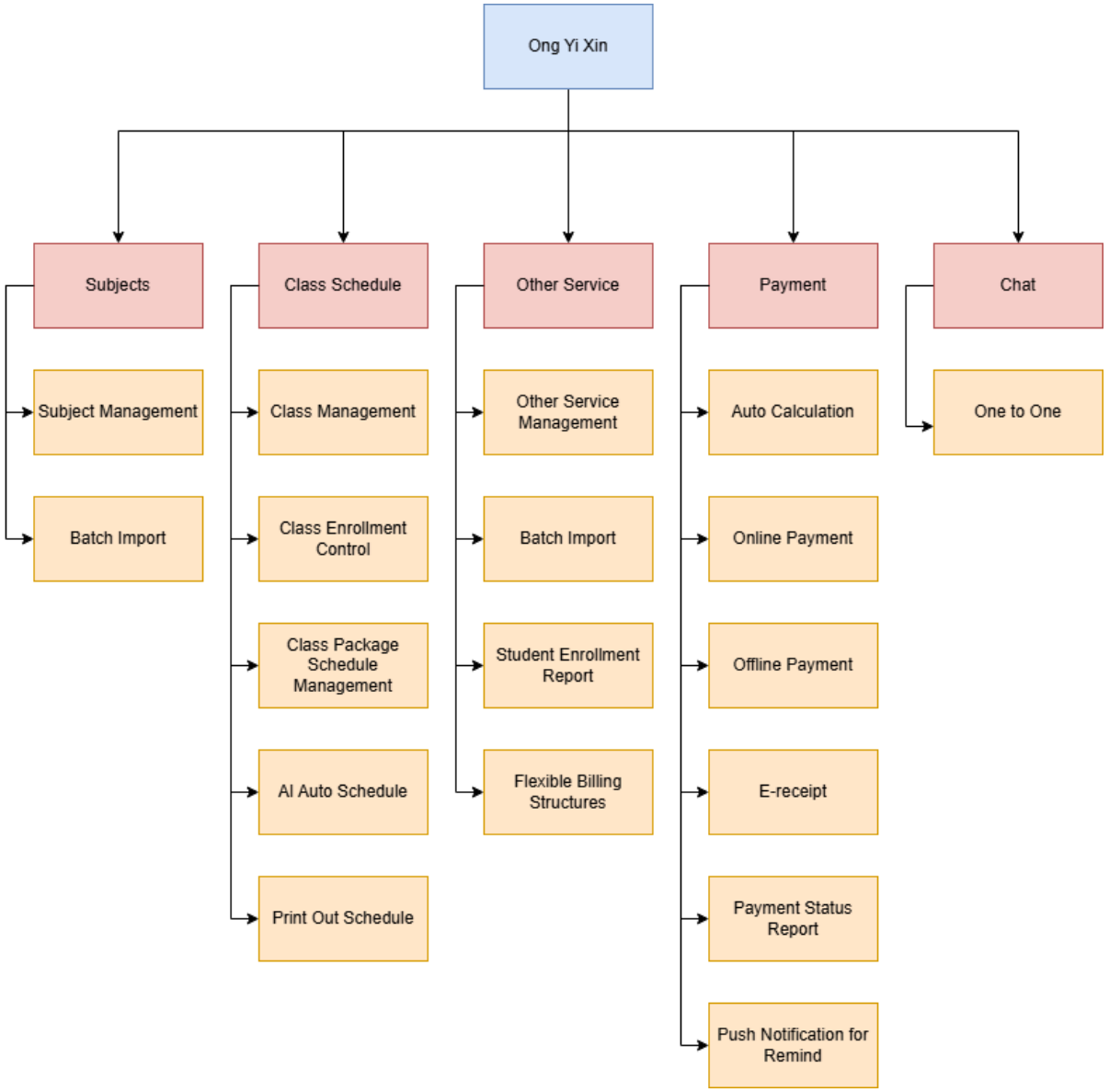


Figure 1.1 Hierarchical Chart of Project Scope

1.4.2 Functional Feature

1.0 Subject Module

1.1 Subject Management

1.1.1 The system shall allow administrators to view created subjects with sorting, filtering, and pagination.

1.1.2 The system shall allow administrators to create new subjects with customizable names and subject fees for different class types.

1.1.3 The system shall allow administrators to update subject details such as name and subject fees for different class types.

1.1.4 The system shall allow administrators to deactivate unused or outdated subjects.

1.1.5 The system shall allow administrators to reactivate previously deactivated subjects when needed.

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 the imported data to check for duplicate errors.

1.2.3 The system shall store all successfully imported subjects into the database in bulk.

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 staff 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 the configuration of class types (e.g., regular, focus).

2.1.5 The system shall allow tutors to view their assigned classes through a dedicated dashboard.

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

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

2.2 Class Enrollment Control

2.2.1 The system shall allow parents to select available classes (including specific class time slots) and enroll their children directly without requiring prior subject enrollment.

2.2.2 The system shall allow staff to enroll students in available classes (including specific class time slots) without requiring prior subject enrollment.

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 ensure that the selected class subject matches the student's education level.

2.2.6 The system shall prevent students from enrolling in multiple classes of the same subject at the same education level.

2.2.7 The system shall validate registration rules (capacity, duplication, conflicts) in real time and notify users when issues occur.

2.2.8 The system shall confirm class enrollment only after successful payment.

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

2.3 AI Auto Scheduling

2.3.1 The system shall automatically generate optimized schedules based on student stress level.

2.3.2 The system shall minimize wait times and avoid overlapping or inefficient class arrangements.

2.3.3 The system shall re-adjust schedules dynamically when a conflict or change is detected.

2.4 Schedule Printing

2.4.1 The system shall allow tutors to export class schedules into PDF or printable formats.

2.4.2 The system shall allow parents and students to view and print their personalized schedules.

2.4.3 The system shall ensure printed schedules include class times, subject names, tutor names, and room numbers.

2.5 Class Package Schedule Management

2.5.1 The system shall allow staff to view suggested class packages generated from the AI Auto Scheduling module.

2.5.2 The system shall ensure each package includes all core subjects required for the education level.

2.5.3 The system shall allow staff to deactivate packages but not edit them.

2.5.4 The system shall allow staff to assign packages to students after packages are reviewed.

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 payments via Stripe.

4.1.2 The system shall allow staff to record offline payments.

4.2 Fee Calculation

4.2.1 The system shall calculate total tuition fees based on registered subjects and the class type.

4.2.2 The system shall calculate the registered other service fees based on days per month.

4.2.3 The system shall calculate childcare fees based on attendance days and included services.

4.2.4 The system shall allow parents to preview fee breakdown before confirming payment.

4.3 Payment Plans

4.3.1 The system shall allow direct payment upon subject.

4.3.2 The system shall support deposit payment for other services like childcare.

4.4 Digital Receipts

4.4.1 The system shall automatically generate a receipt after each successful payment.

4.4.2 The system shall allow parents to download or print past receipts.

5.0 Other Service Module

5.1 Service Management

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

5.1.2 The system shall allow staff to create, edit, and deactivate services, including transport services with attributes such as direction and default pickup/drop-off times.

5.1.3 The system shall allow parents to register their children for services, and for transport services, specify a pickup/drop-off address.

5.1.4 The system shall allow staff to register students for services, including transport.

5.1.5 The system shall allow editing or cancellation of service enrollments as needed.

5.3 Payment Integration

5.3.1 The system shall pass service enrollment data directly to the payment module.

5.3.2 The system shall ensure accurate calculations of service-related fees based on attendance logs.

5.4 Batch Other Service Import

5.4.1 The system shall allow administrators to upload a CSV or Excel file containing other service data.

5.4.2 The system shall validate the imported data to check for duplicates and formatting errors.

5.4.3 The system shall store all successfully imported other services into the database in bulk.

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.

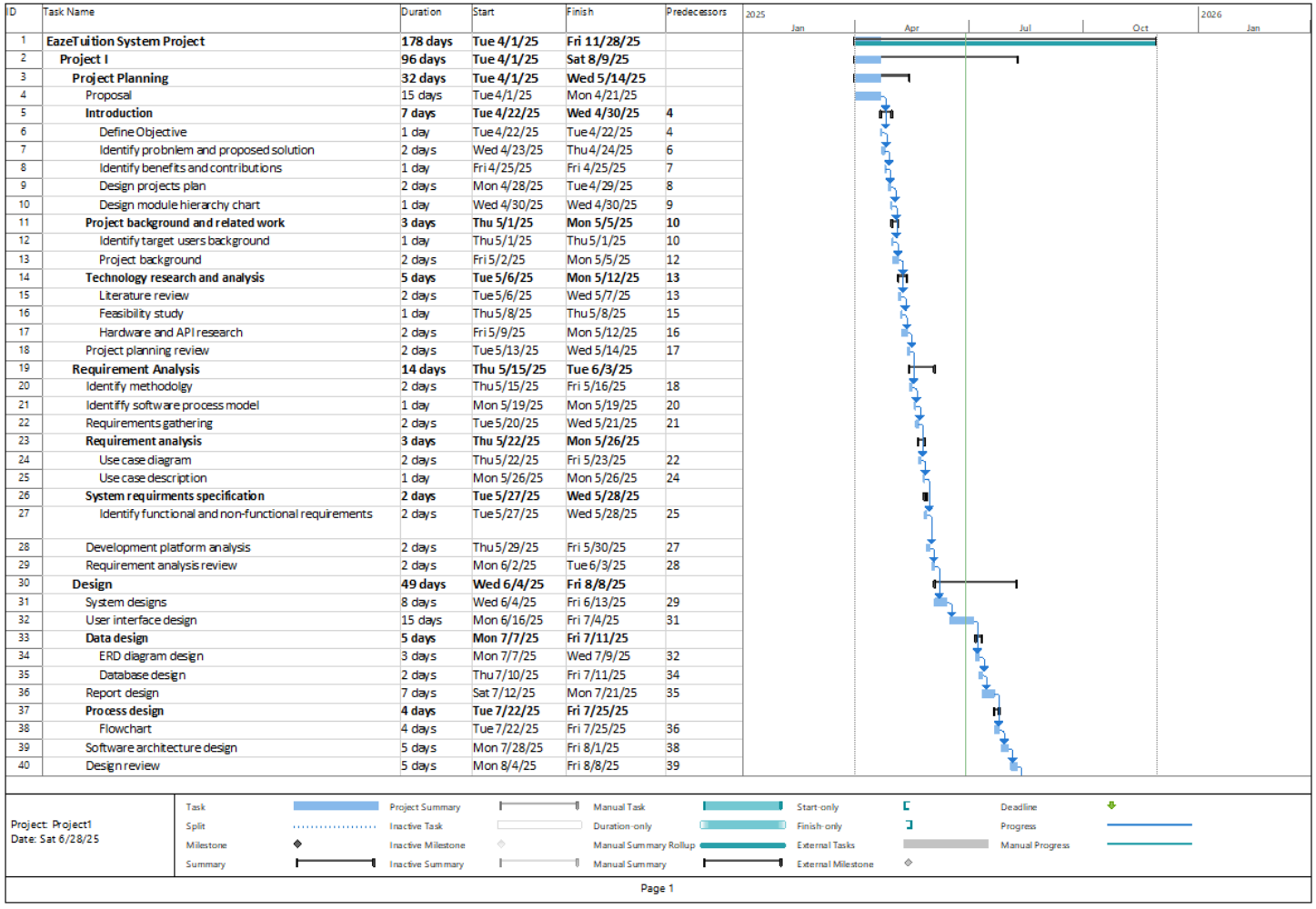
Organization

- 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

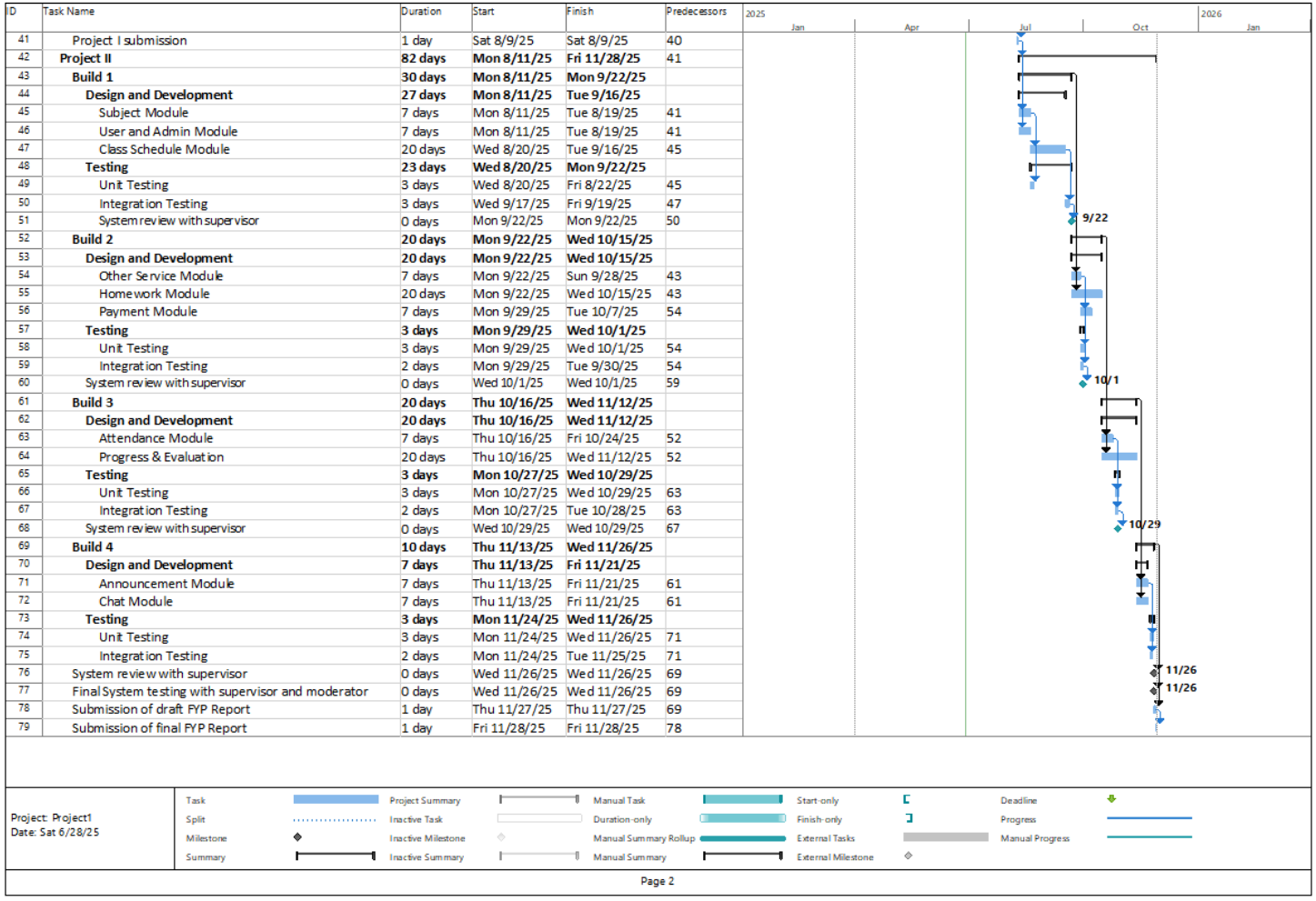
- The app shall support online payments processed using Stripe in Malaysian Ringgit (MYR).
- All user data handling shall comply with local privacy regulations (e.g., Malaysia PDPA) and follow best practices for security and data protection using XAMPP and phpMyAdmin.
- Future expansion may include compliance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

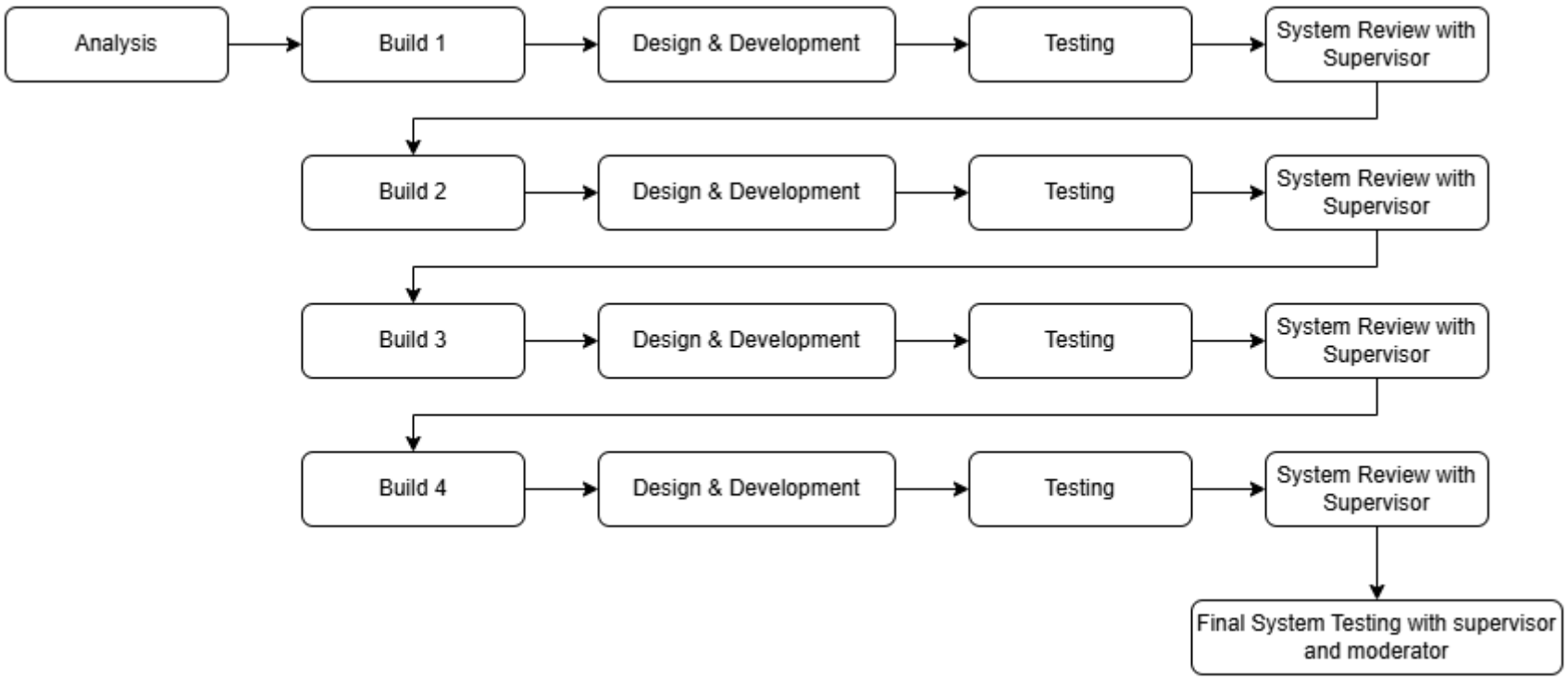


Figure 2.3 The Incremental Model

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 1.1 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

Scikit-learn

Scikit-learn is an open source Python library widely used for machine learning and data mining. It builds on NumPy, SciPy, and Matplotlib to provide efficient tools for classification, regression, clustering, dimensionality reduction, model selection, and data preprocessing (Pedregosa et al., 2011). It is designed to be user-friendly and easy to integrate into data science pipelines.

Unsupervised Learning with Structured Educational Data
Scikit-learn is ideal for unsupervised learning tasks such as K-Means clustering, which is commonly used to identify hidden patterns in students' behavior, preferences, or engagement levels (Pedregosa et al., 2011). It is best suited for structured datasets such as study time, sleep patterns, GPA, attendance frequency, and payment patterns

The EazeTuition application uses it to group students into achievement-based clusters

such as "high performance + high stress", "balanced", and "low focus" to provide scheduling and tutoring recommendations.

Rule-Based Decision Support
Scikit-learn can also be used for basic classification models or decision logic to produce actionable insights (scikit-learn). This works well when the goal is to suggest improvements based on patterns of cluster behavior.

In EazeTuition, it uses Scikit-learn to enable automatic adjustment of lesson suggestions or to notify administrators for better resource allocation. For example, "Students in cluster 2 benefit more from the morning session."

Integration with Python backends and data stacks
Scikit-learn integrates seamlessly with other Python libraries such as Pandas, NumPy, and Matplotlib, which are often used for preprocessing and visualization, and is compatible with Python-based backend systems such as Flask.

This allows EazeTuition to extract data directly from MySQL, process it via Pandas, analyze it via Scikit-learn, and present insights to the React Native front end via the Flask API.

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 K-Means clustering technology** to **group students by behavioral patterns** such as study time and availability. These clusters inform AI-generated scheduling packages that staff can review and adjust to reduce conflicts and personalize 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

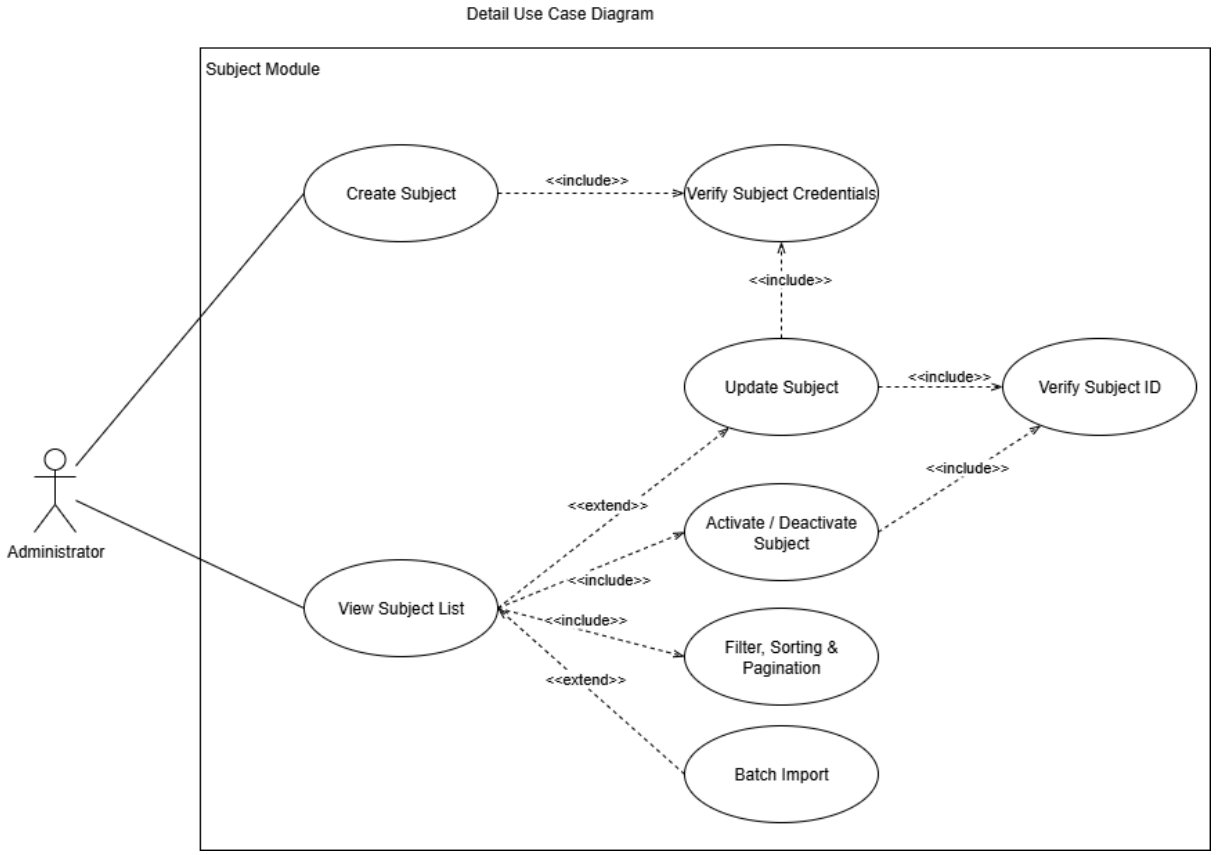


Figure 3.3.1.1 Use Case Diagram of Subject Module

3.3.1.2 Class Module

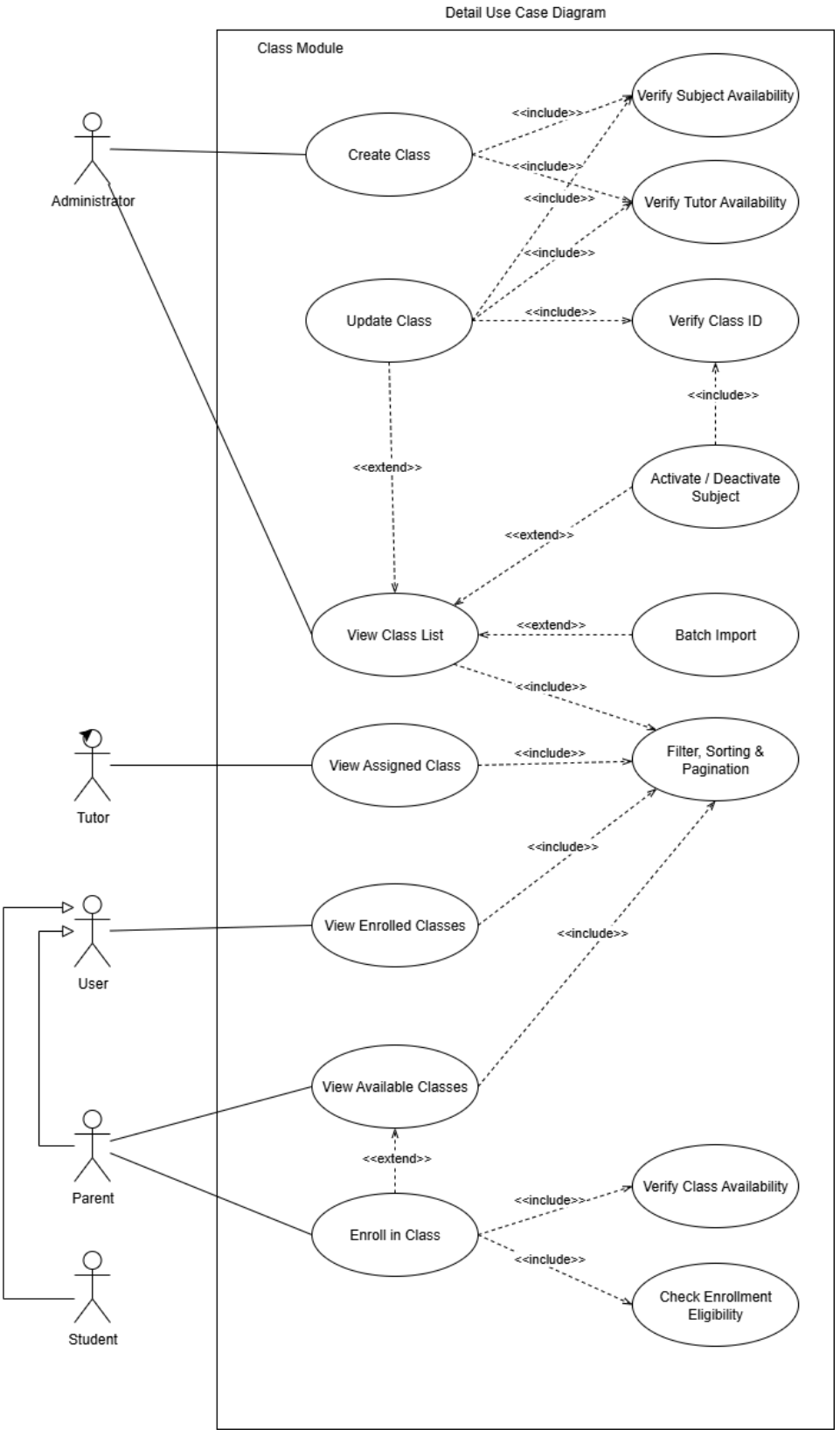


Figure 3.3.1.2 Use Case Diagram of Class Module

3.3.1.3 Class Schedule Module

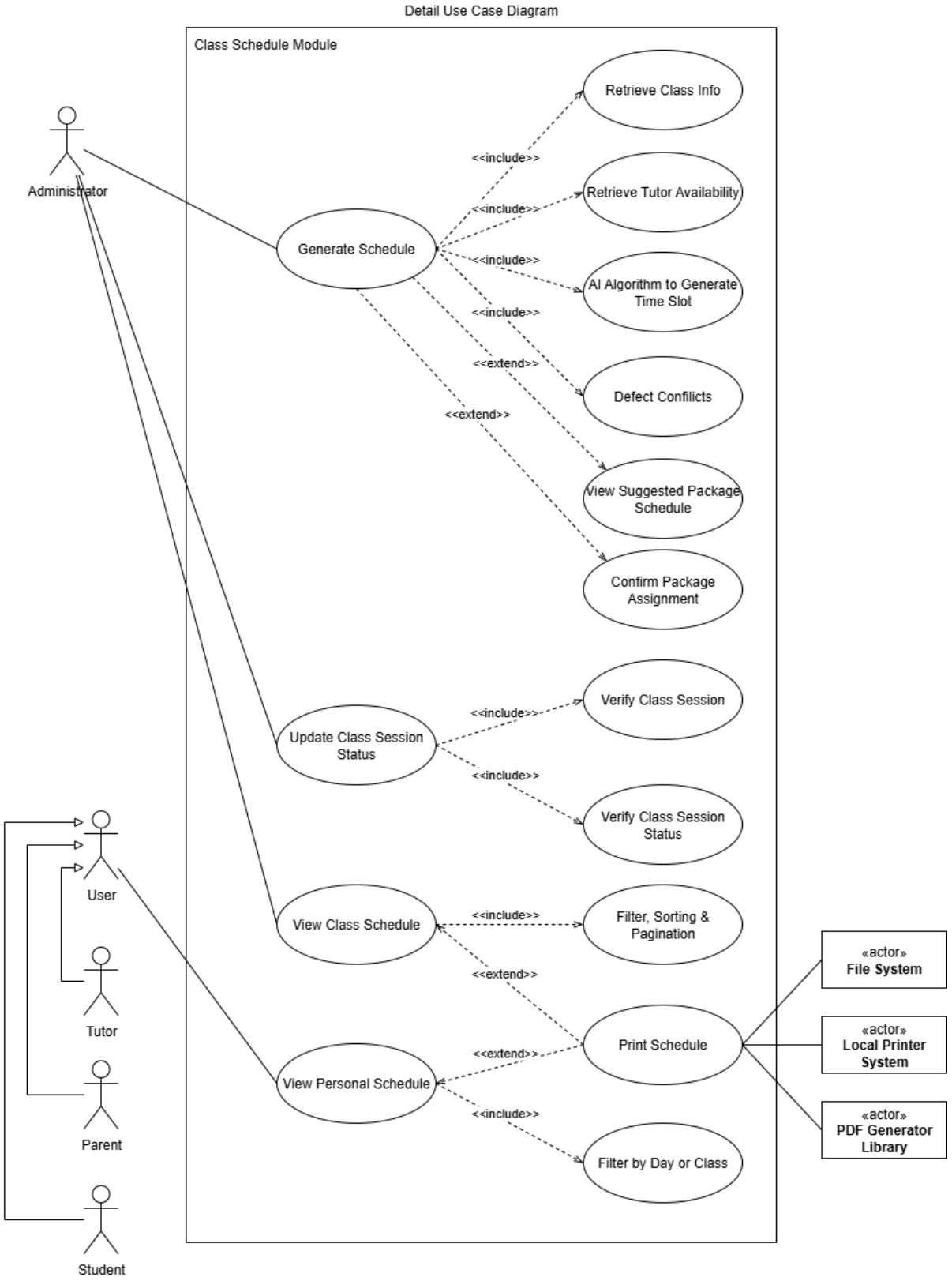


Figure 3.3.1.3 Use Case Diagram of Class Schedule Module

3.3.1.4 Payment Module

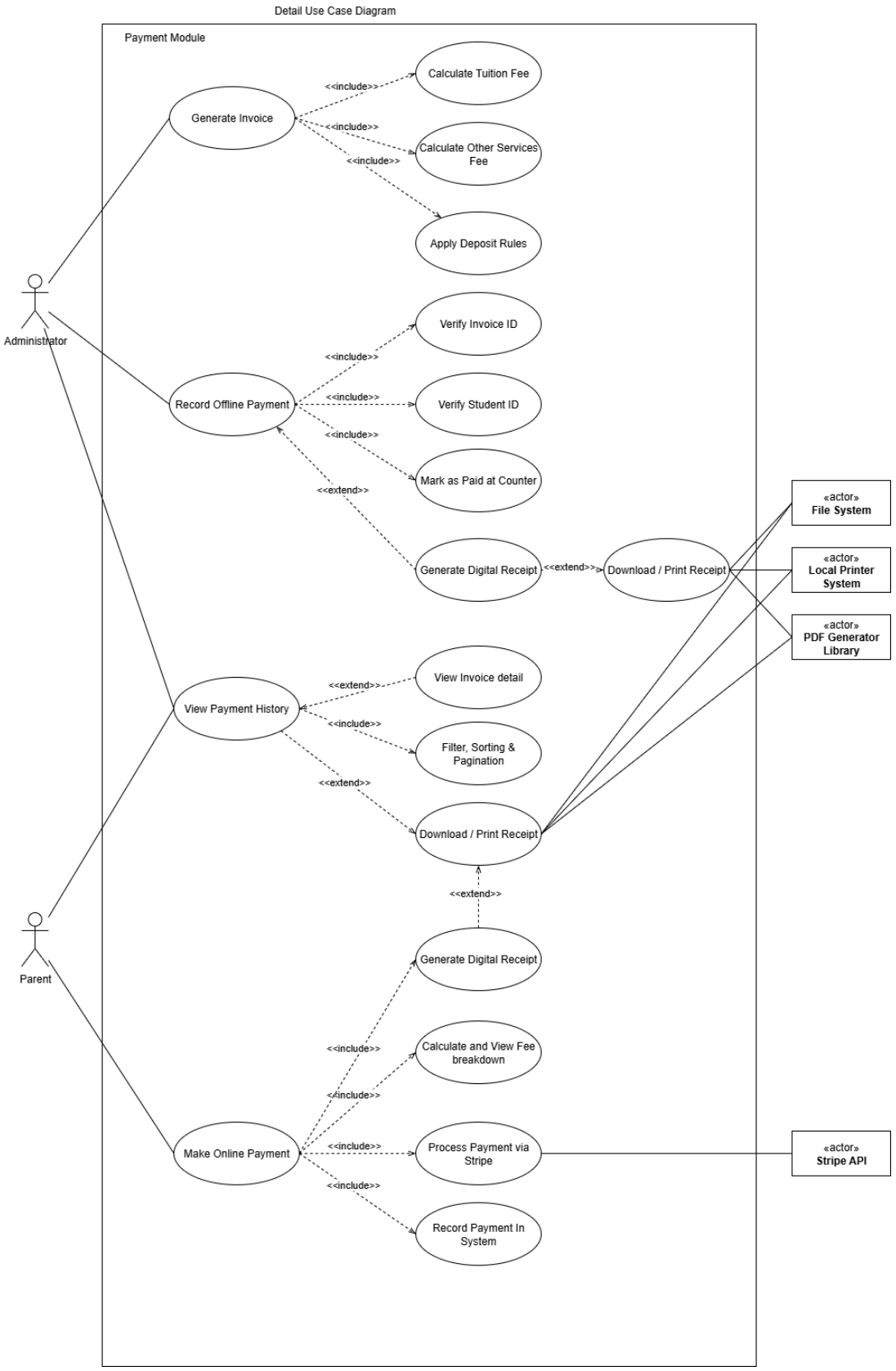


Figure 3.3.1.4 Use Case Diagram of Payment Module

3.3.1.5 Other Service Module

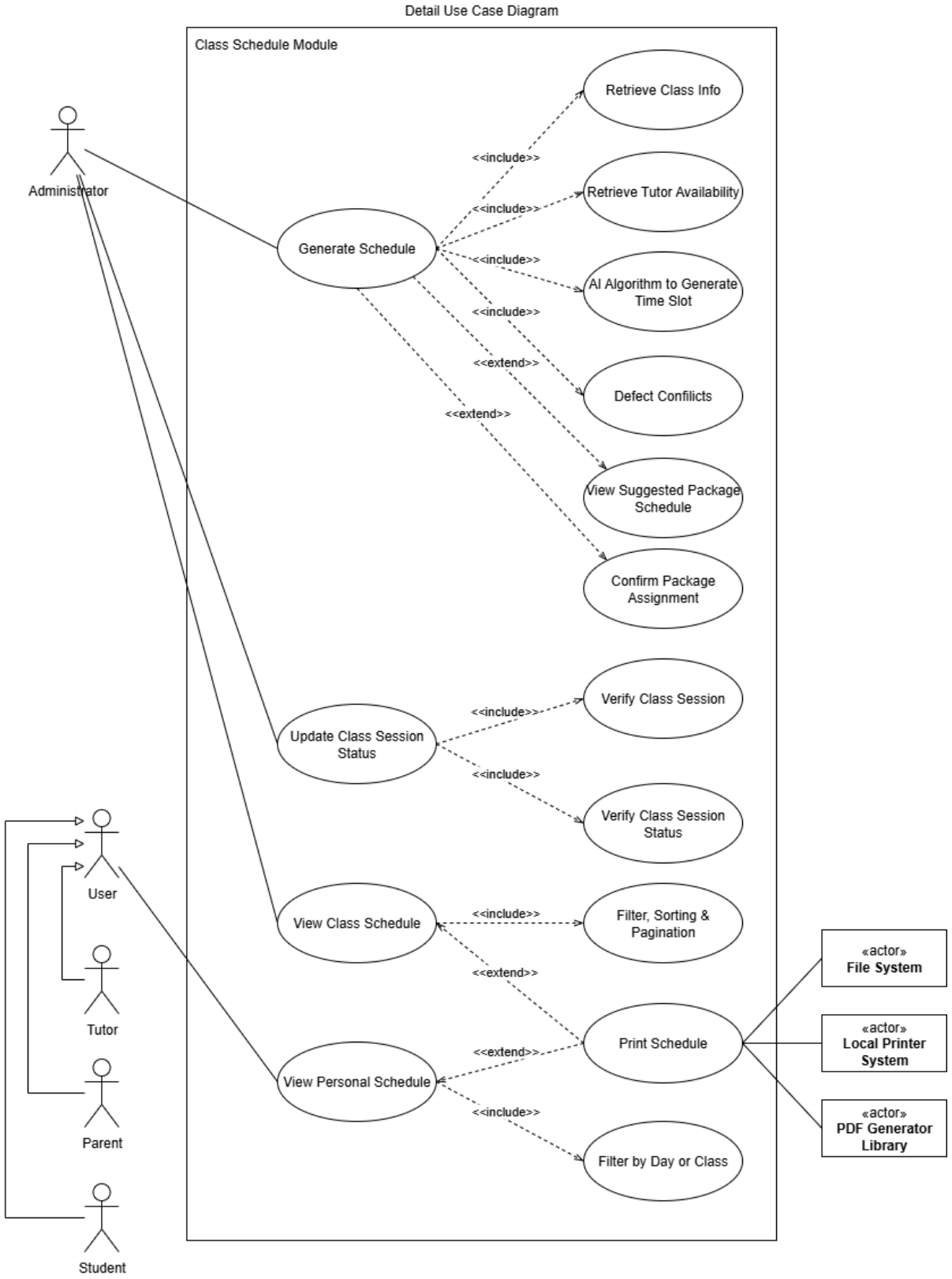
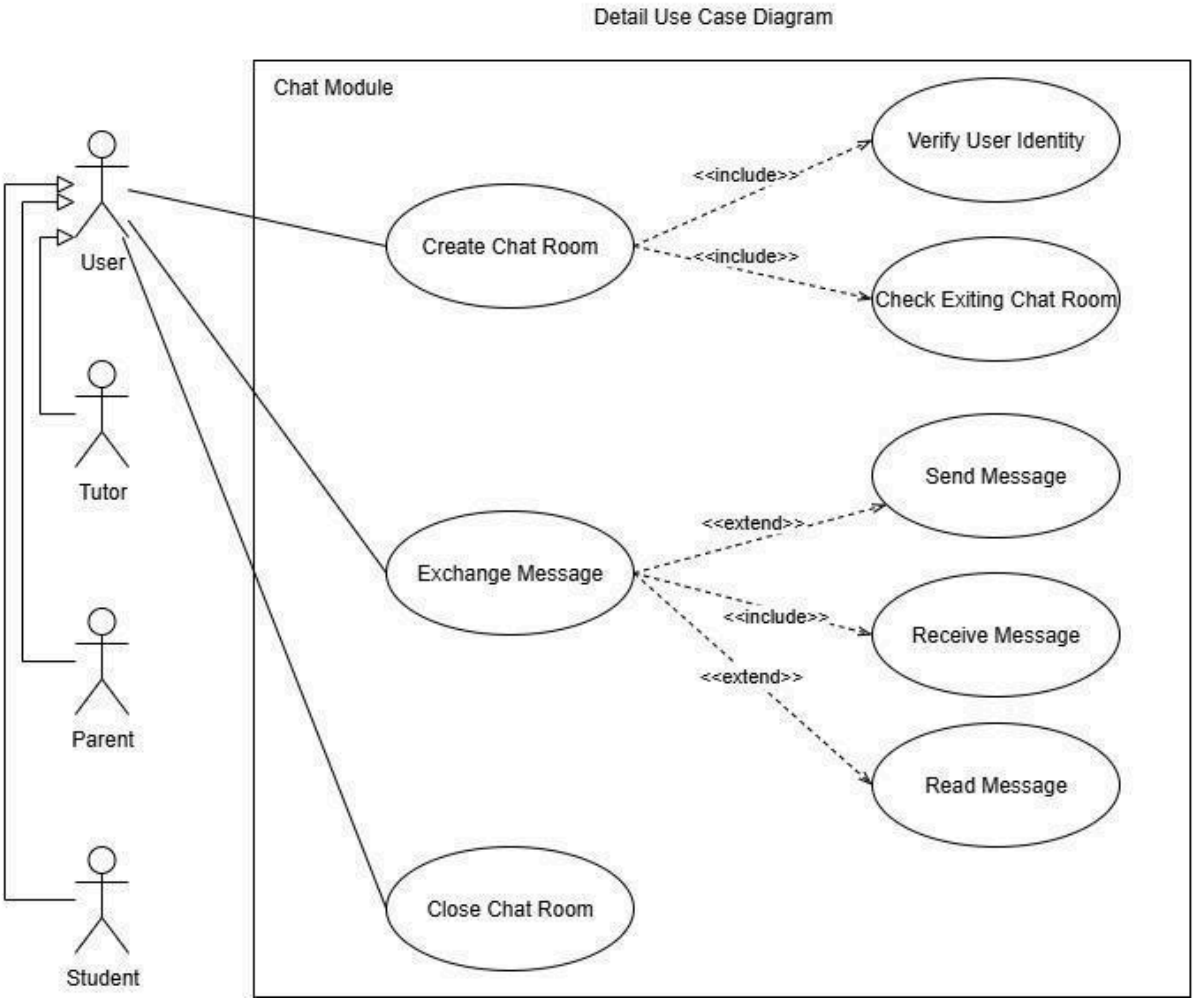


Figure 3.3.1.5 Use Case Diagram of Other Service Module

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 is active status, 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 empty, 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	4. The system verifies subject credentials and displays a

the "Save" button.	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 is active, 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" button on a subject from the subject list.	2. Verify if the subject exists.
	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, mode, and max capacity.
3. Administrator fills in the required fields and clicks	4. The system verifies the subject and the tutor is active

"Create".	and available and hows a confirmation popup: "Confirm to create class?" with "Yes" and "Cancel" buttons..
5. The administrator selects an option.	6. The system redirects the user to 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 "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 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 "Activate/Deactivate", the system extends to the Activate/Deactivate Class use case. If the administrator clicks on the "Batch Import" button, the system includes the Batch Import 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, time, subject, or assigned tutor. 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	4. The system verifies class

one or more fields and clicks the "Save" button.	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

Activate / Deactivate Class

Name of Use Case: Activate / 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	2. Verify if the class exists.

"Activate" or "Deactivate" button on a class from the class list.	
	3. The system displays a confirmation popup: "Are you sure you want to [Activate/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

Batch Import Classes

Name of Use Case: Batch Import Classes	
Brief Description: The administrator can upload a structured file (e.g., CSV or Excel) to import multiple class 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 class 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 class 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 class records are added to the database and displayed in the class list. Invalid entries are reported and not saved.	

Table 3.3.2.2.5 Use Case Description of Batch Import 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 table format.
3. Tutor uses search or filtering	4. The system filters and

options.	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 3.3.2.2.6 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 or table 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 3.3.2.2.7 Use Case Description of View Enrolled Class (Student / Parent)

View Available Classes

Name of Use Case: View Available Classes	
Brief Description: The parent views the list of classes that are open for enrollment. This allows the parent to explore available subjects, timings, and tutors before enrolling their child.	
Actor: Parent	
Precondition: The parent must be logged into the system. The system must contain at least one active class open for enrollment.	
Actor Action	**System Response**
1. Tutor clicks on the "Available Classes" tab.	2. The system retrieves and displays a list of all classes currently open for enrollment.
3. The parent uses search, filter, or sort options.	4. The system updates the list based on the selected filters or sorting order.
5. The parent selects a class to view more details.	6. The system displays detailed class information in a popup or separate page.
Alternative Flows: A1. Step 2. If there are no available classes, the system displays the message "No classes available for enrollment at this time." If a system error occurs while fetching data, the system shows an error message "Unable to load available classes. Please try again later."	
Postcondition: The parent is able to view all available classes and their details to support informed enrollment decisions.	

Table 3.3.2.2.8 Use Case Description of View Available Class

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. Tutor clicks on the "Enroll" tab.	2. The system highlights selected classes and enables the "Enroll Selected Classes" button.
3. Parent clicks the "Enroll Selected Classes" button.	4. The system displays a confirmation popup showing the list of selected classes, with "Yes" and "Cancel" options.
5. The parent selects one of the options.	6. 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 3.3.2.2.9 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 generates a class schedule automatically using an AI algorithm that takes into account class information and tutor availability. The system detects any conflicts and suggests available 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 retrieves and displays suggested package schedules grouped by class.
3. Administrator checks the box(es) for one or more preferred schedule packages.	4. The system enables the "Confirm Selected" button once at least one schedule is selected.
5. Administrator clicks the "Confirm Selected" button.	6. The system prompts for confirmation with a popup: "Are you sure you want to assign the selected schedules?"
7. The administrator selects one of the options.	8. 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 or list 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 "Update Status" 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 update the status 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 completed or canceled "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.
5. Administrator clicks "Print Schedule".	6. The system formats the schedule for printing and opens the print dialog.
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." 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 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 selected filter.
5. The user clicks "Print Schedule".	6. The system generates a print-friendly version of the schedule and opens the print dialog.
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	2. The system displays the

"Proceed to Payment" after generating an invoice.	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", "Download 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 "Download Receipt", the system downloads the receipt locally in PDF format. 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

<table>
<tr><td colspan="2">receipt.</td></tr>
<tr><td colspan="2">Actor: Parent</td></tr>
<tr><td colspan="2">Precondition: The parent is logged into the system. The parent has at least one unpaid invoice.</td></tr>
<tr><th>Actor Action</th><th>System Response</th></tr>
<tr><td>1. The parent selects "Make Payment" from the dashboard.</td><td>2. The system retrieves unpaid invoice(s) and displays the associated fee breakdown for review.</td></tr>
<tr><td>3. The parent reviews the fee breakdown and clicks "Proceed to Pay".</td><td>4. The system verifies invoice validity and payment eligibility.</td></tr>
<tr><td>5. The parent provides payment details and submits the payment.</td><td>6. The system processes the payment via Stripe and successfully completes.</td></tr>
<tr><td></td><td>7. The system records the payment, generates a digital receipt, and displays a success message.</td></tr>
<tr><td>8. The parent selects either "Download Receipt" or "Back to Payment History".</td><td>9. The system carries out the selected action.</td></tr>
<tr><td colspan="2">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.</td></tr>
<tr><td colspan="2">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.</td></tr>
</table>

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 pickup/drop-off times, direction, and valid zones.	
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 type, direction time, and service zone.	4. The system verifies the completeness of the data.
5. The administrator submits the form.	6. The system validates transport credentials, checks for conflicting zones or time overlaps, and confirms zone validity.
	7. The system creates the service record and displays a success message.
Alternative Flows: A1. Step 4. If the direction or time entered is invalid, the system displays an error message and prompts for correction. If the selected zone is not supported, the system notifies the administrator and halts the process. 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, activate/deactivate, and batch import service records. 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 "Activate/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 "Activate/Deactivate", the system extends to the Activate/Deactivate Service use case. If the administrator clicks on the "Batch Import" button, the system includes the Batch Import 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

Activate / Deactivate Other Service

Name of Use Case: Activate / Deactivate Other Service	
Brief Description: The administrator can activate or 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 "Activate" or "Deactivate" button on a specific service.	4. The system prompts for confirmation: "Are you sure you want to activate/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

Batch Import Service

Name of Use Case: Batch Import Service
Brief Description: The administrator uploads a batch file containing

multiple other service records to be imported into the system. The system validates the data and imports valid records.	
Actor: Administrator	
Precondition: The administrator is logged into the system. The batch import template is correctly formatted.	
Actor Action	**System Response**
1. The administrator clicks the "Batch Import" button on the Service List page.	2. The system displays the batch import interface with upload instructions and a sample template.
3. The administrator uploads a CSV/XLSX file containing multiple service records.	4. The system parses the uploaded file and validates each record.
5. The administrator clicks "Confirm Import."	6. The system imports all valid records into the database and skips any invalid rows, displaying relevant error messages.
	7. The system displays a summary: number of records imported, skipped rows , and provides an option to "Back to Service List".
Alternative Flows: A1. Step 4. If the file format is not supported, the system shows an error message "Unsupported file format. Please upload a CSV or XLSX file." If all rows are invalid, the system aborts the import and displays: "No valid records found. Please check your file." A2. Step 6. If the administrator cancels the operation, the system discards the upload and returns to the Service List without changes.	
Postcondition: All valid service records from the uploaded file are stored in the system. Invalid rows are skipped with feedback provided.	

Table 3.3.2.5.4 Use Case Description of Batch Import 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 3.3.2.5.5 Use Case Description of Cancel Other Service Enrollment

View Available Other Service

Name of Use Case: View Available Other Service
Brief Description: The parent views a list of available transport and childcare services offered by the tuition center. The system supports

filtering, sorting, and pagination for easier navigation.	
Actor: Parent	
Precondition: The parent is logged into the system. At least one active other service is available in the system.	
Actor Action	**System Response**
1. The parent navigates to the "Other Services" section.	2. The system retrieves and displays a paginated list of available services.
3. The parent applies filters or sorting.	4. The system updates the list based on the selected filters or sorting options.
5. The parent views details of a specific service.	6. The system displays the full information for the selected service.
Alternative Flows: A1. Step 2. If no services are currently available, the system displays a message "No available services found." A2. Step 4. If the parent applies filters and no services match, the system shows an empty filtered result with a corresponding message.	
Postcondition: The parent has successfully browsed the available other services and can proceed to register if desired.	

Table 3.3.2.5.6 Use Case Description of View Available Service (Parent)

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 "Register" on the chosen service.	2. The system displays the registration form, including

	child selection.
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, add 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 3.3.2.5.7 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 fees, pickup/drop-off times, direction, and valid zones. 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 3.3.2.5.8 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 created subjects with sorting, filtering, and pagination.

1.1.2 The system shall allow administrators to create new subjects with customizable names and subject fees for different class types.

1.1.3 The system shall allow administrators to update subject details such as name and subject fees for different class types.

1.1.4 The system shall allow administrators to deactivate unused or outdated subjects.

1.1.5 The system shall allow administrators to reactivate previously deactivated subjects when needed.

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 the imported data to check for duplicate errors.

1.2.3 The system shall store all successfully imported subjects into the database in bulk.

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 staff 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 the configuration of class types (e.g., regular, focus).

2.1.5 The system shall allow tutors to view their assigned classes through a dedicated dashboard.

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

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

2.2 Class Enrollment Control

2.2.1 The system shall allow parents to select available classes (including specific class time slots) and enroll their children directly without requiring prior subject enrollment.

2.2.2 The system shall allow staff to enroll students in available classes (including specific class time slots) without requiring prior subject enrollment.

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 ensure that the selected class subject matches the student's education level.

2.2.6 The system shall prevent students from enrolling in multiple classes of the same subject at the same education level.

2.2.7 The system shall validate registration rules (capacity, duplication, conflicts) in real time and notify users when issues occur.

2.2.8 The system shall confirm class enrollment only after successful payment.

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

2.3 AI Auto Scheduling

2.3.1 The system shall automatically generate optimized schedules based on student stress level.

2.3.2 The system shall minimize wait times and avoid overlapping or inefficient class arrangements.

2.3.3 The system shall re-adjust schedules dynamically when a conflict or change is detected.

2.4 Schedule Printing

2.4.1 The system shall allow tutors to export class schedules into PDF or printable formats.

2.4.2 The system shall allow parents and students to view and print their personalized schedules.

2.4.3 The system shall ensure printed schedules include class times, subject names, tutor names, and room numbers.

2.5 Class Package Schedule Management

2.5.1 The system shall allow staff to view suggested class packages generated from the AI Auto Scheduling module.

2.5.2 The system shall ensure each package includes all core subjects required for the education level.

2.5.3 The system shall allow staff to deactivate packages but not edit them.

2.5.4 The system shall allow staff to assign packages to students after packages are reviewed.

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 payments via Stripe.

4.1.2 The system shall allow staff to record offline payments.

4.2 Fee Calculation

4.2.1 The system shall calculate total tuition fees based on registered subjects and the class type.

4.2.2 The system shall calculate the registered other service fees based on days per month.

4.2.3 The system shall calculate childcare fees based on attendance days and included services.

4.2.4 The system shall allow parents to preview fee breakdown before confirming payment.

4.3 Payment Plans

4.3.1 The system shall allow direct payment upon subject.

4.3.2 The system shall support deposit payment for other services like childcare.

4.4 Digital Receipts

4.4.1 The system shall automatically generate a receipt after each successful payment.

4.4.2 The system shall allow parents to download or print past receipts.

5.0 Other Service Module

5.1 Service Management

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

5.1.2 The system shall allow staff to create, edit, and deactivate services, including transport services with attributes such as direction and default pickup/drop-off times.

5.1.3 The system shall allow parents to register their children for services, and for transport services, specify a pickup/drop-off address.

5.1.4 The system shall allow staff to register students for services, including transport.

5.1.5 The system shall allow editing or cancellation of service enrollments as needed.

5.3 Payment Integration

5.3.1 The system shall pass service enrollment data directly to the payment module.

5.3.2 The system shall ensure accurate calculations of service-related fees based on attendance logs.

5.4 Batch Other Service Import

5.4.1 The system shall allow administrators to upload a CSV or Excel file containing other service data.

5.4.2 The system shall validate the imported data to check for duplicates and formatting errors.

5.4.3 The system shall store all successfully imported other services into the database in bulk.

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.
- **Future expansion may include compliance with relevant e-commerce and consumer protection laws.**

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

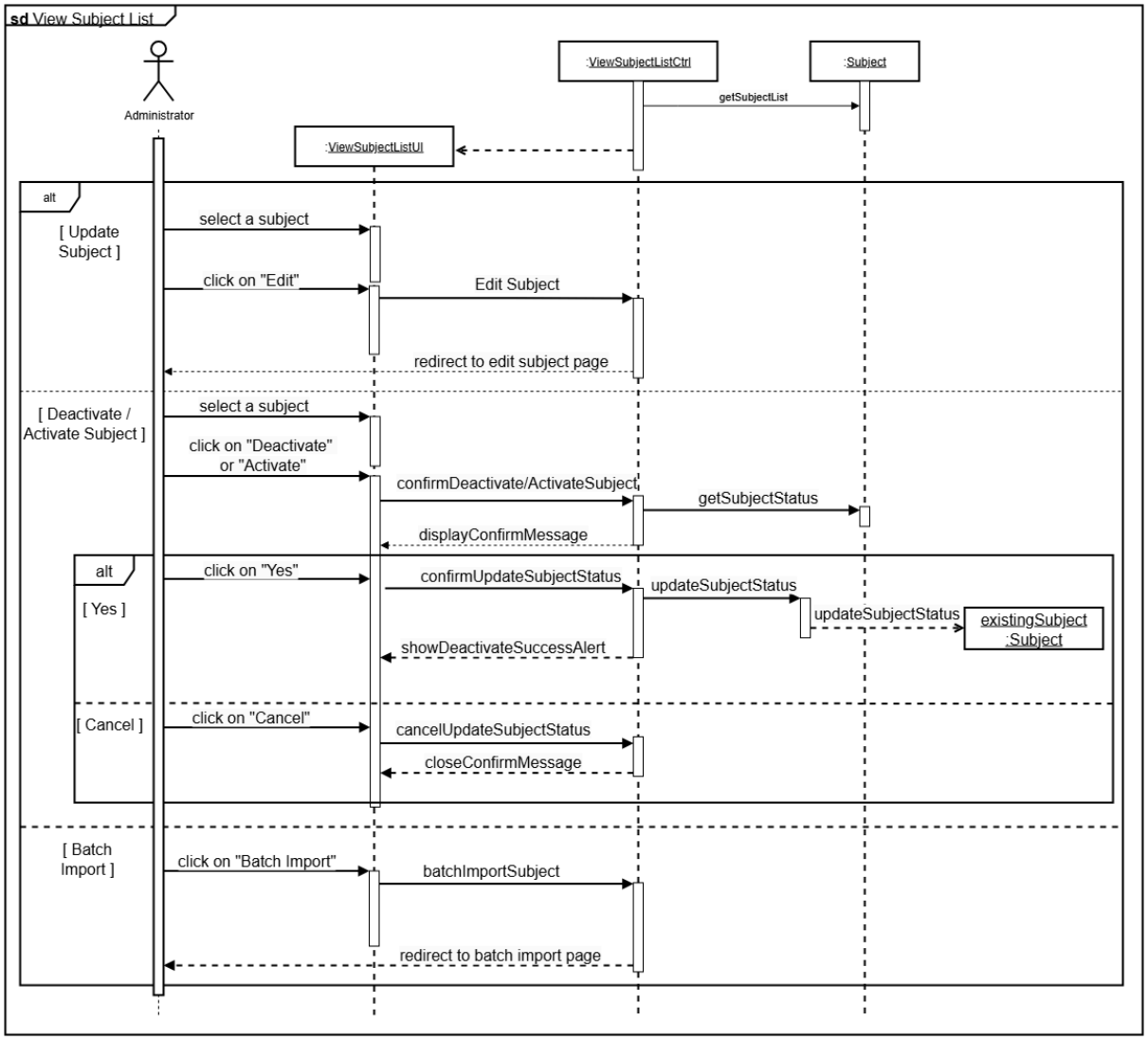


Figure 4.1.1.1 Sequence Diagram for View Subject List

Add New Subject

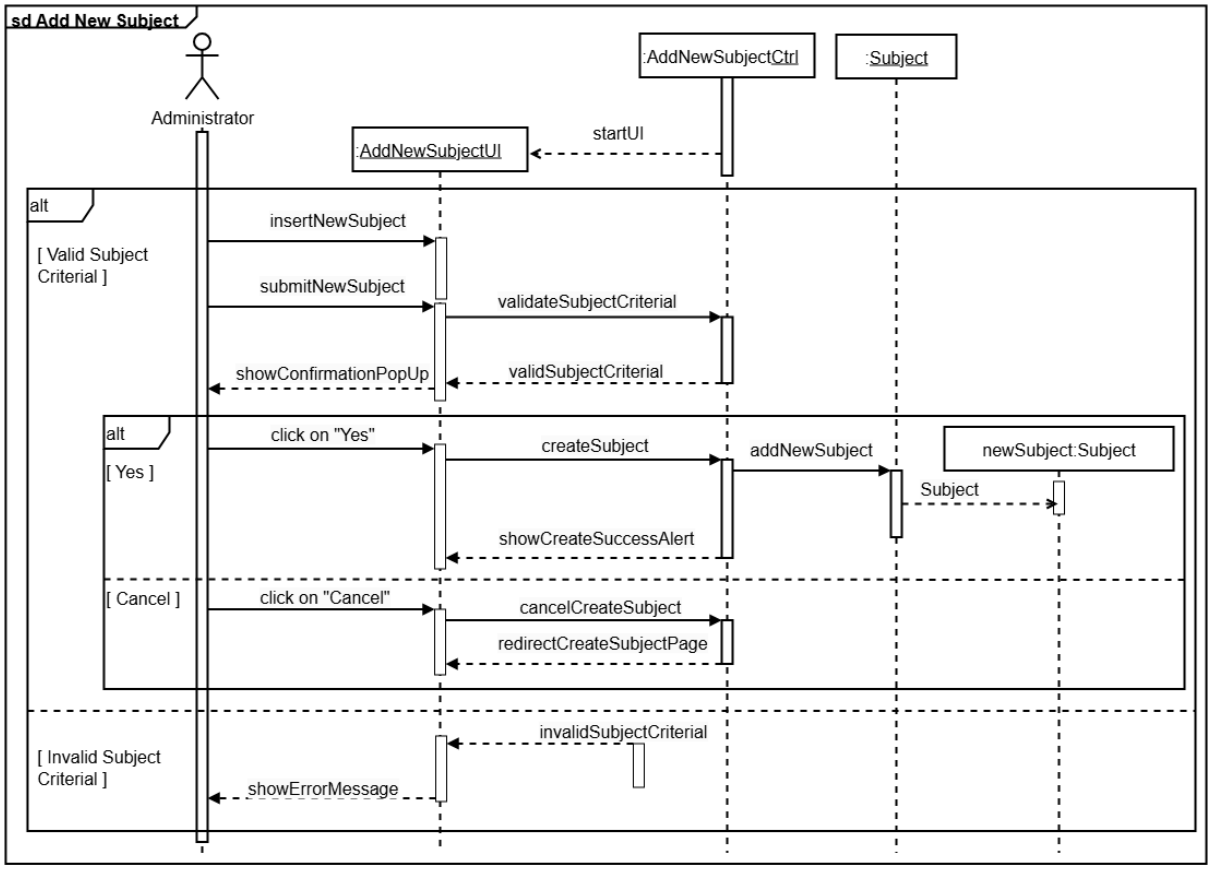


Figure 4.1.1.2 Sequence Diagram for Add New Subject

Update Subject

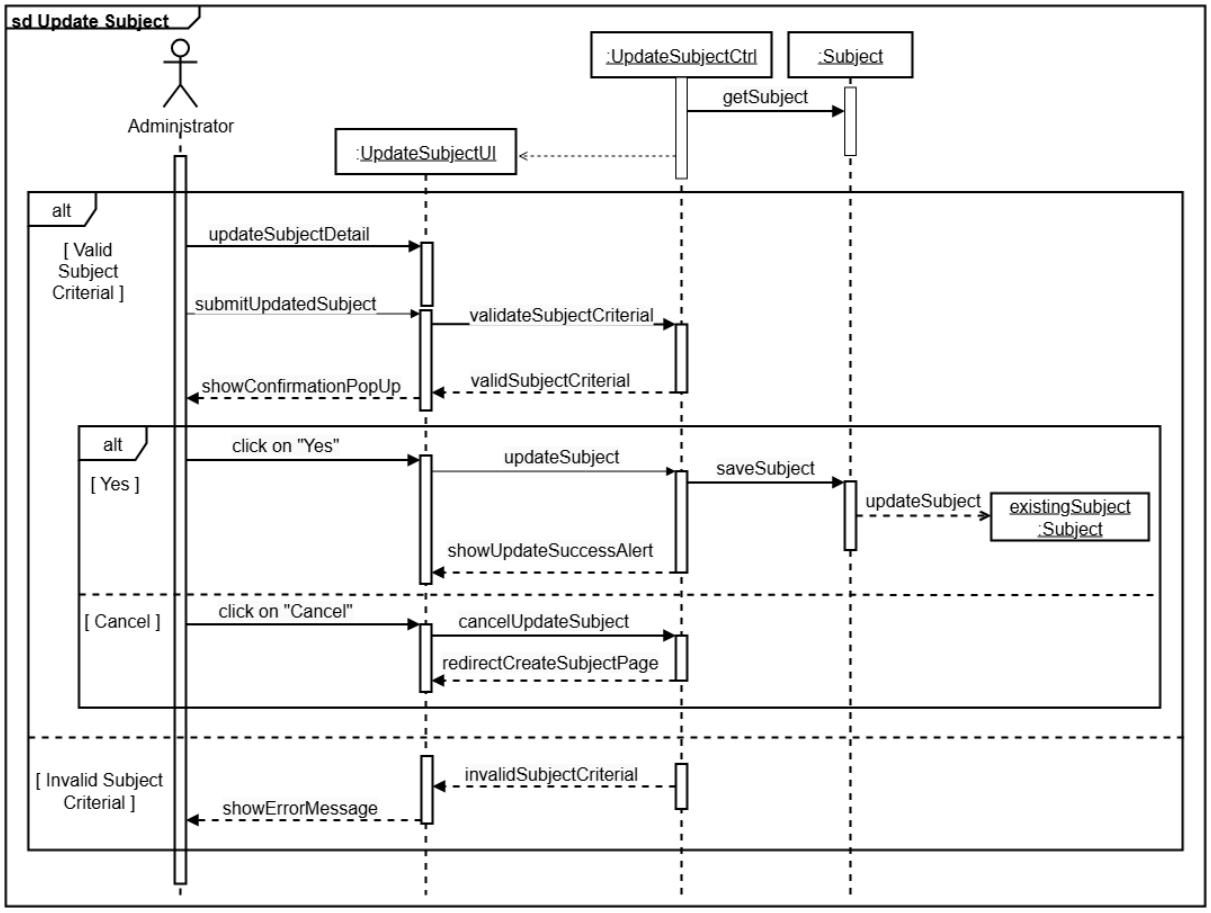


Figure 4.1.1.3 Sequence Diagram for Update Subject

Deactivate / Reactivate Subject

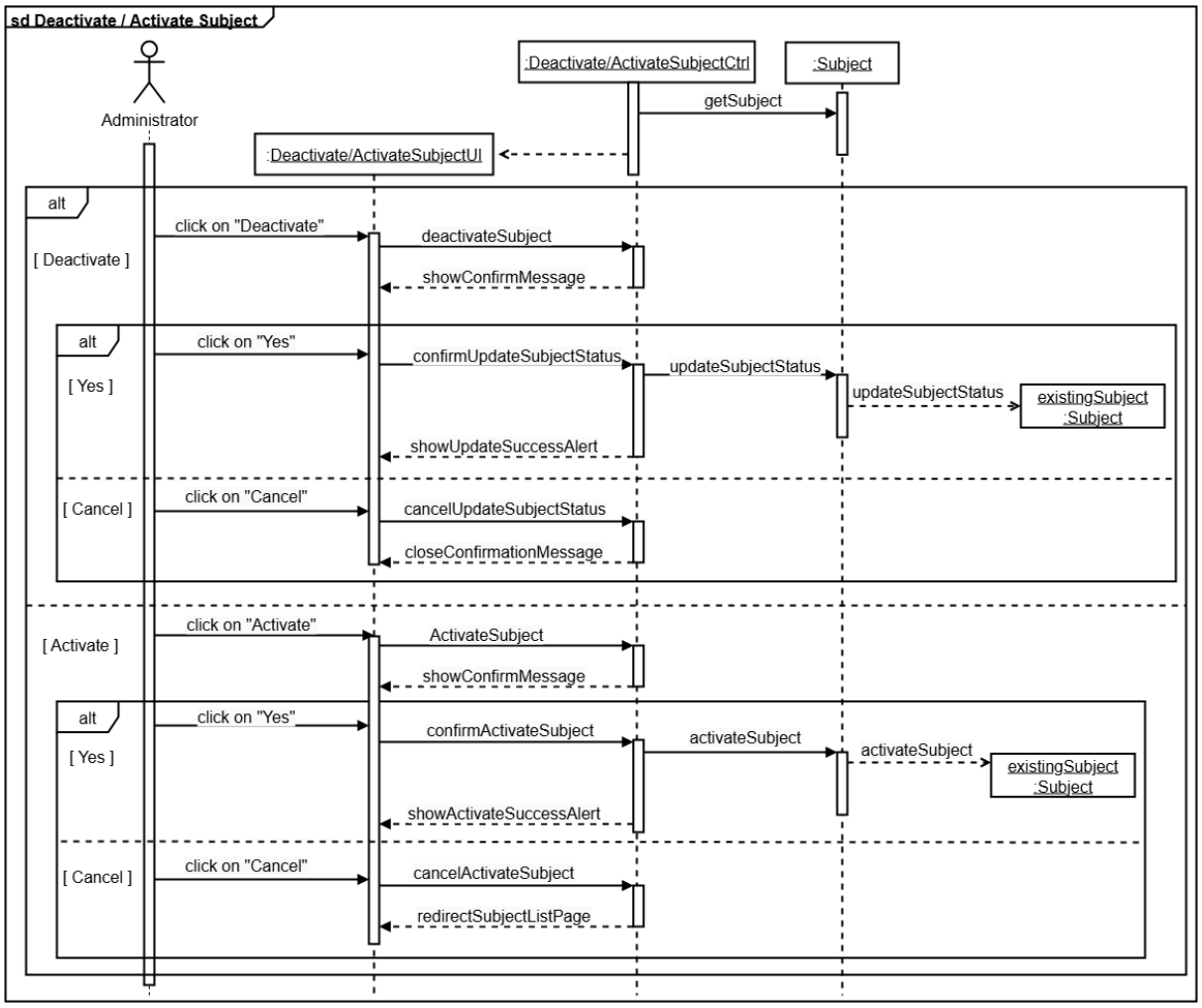


Figure 4.1.1.4 Sequence Diagram for Deactivate / Reactivate Subject

Batch Import Subject

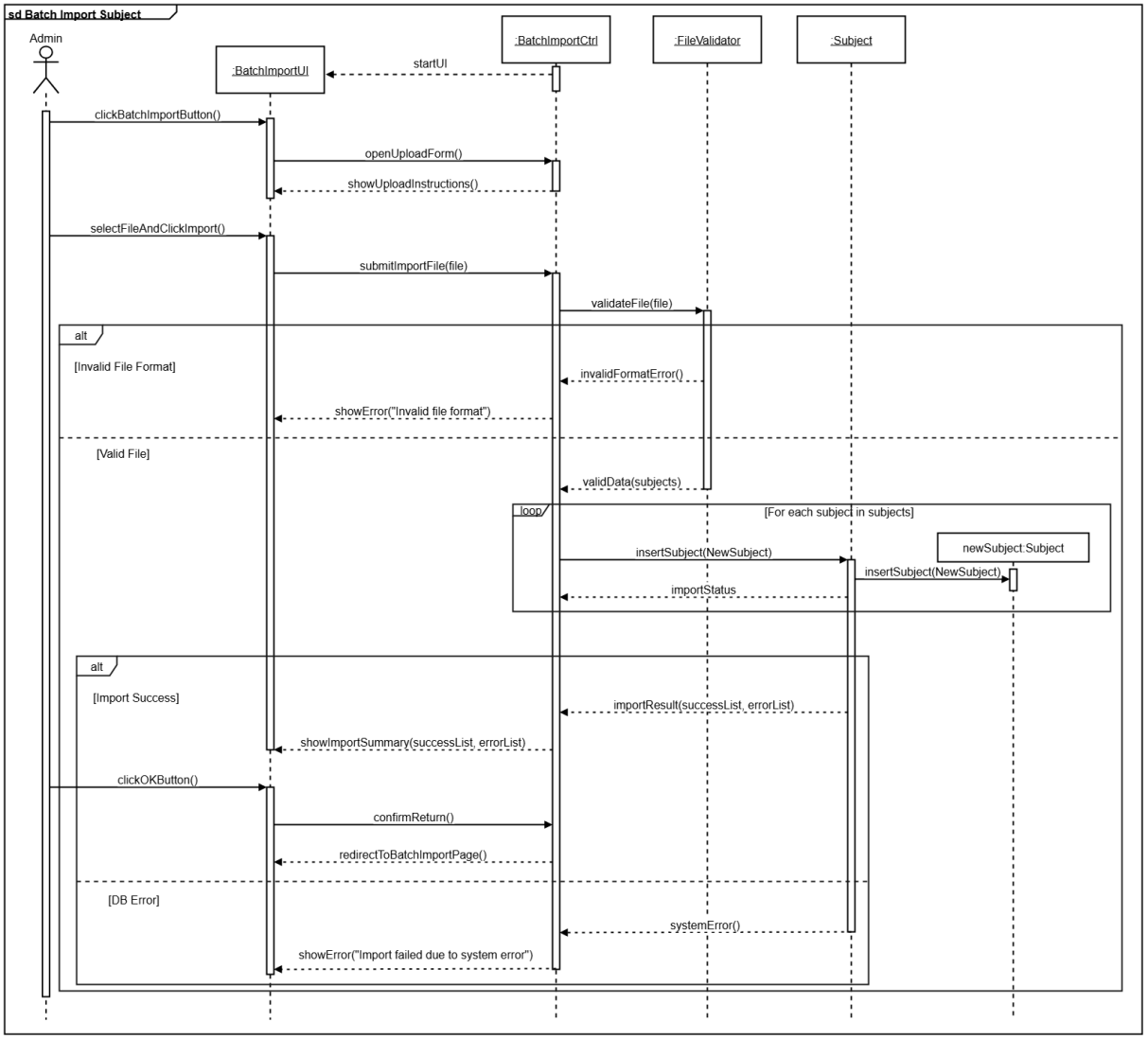


Figure 4.1.1.5 Sequence Diagram for Batch Import Subject

4.1.2 Class Module

View Class List

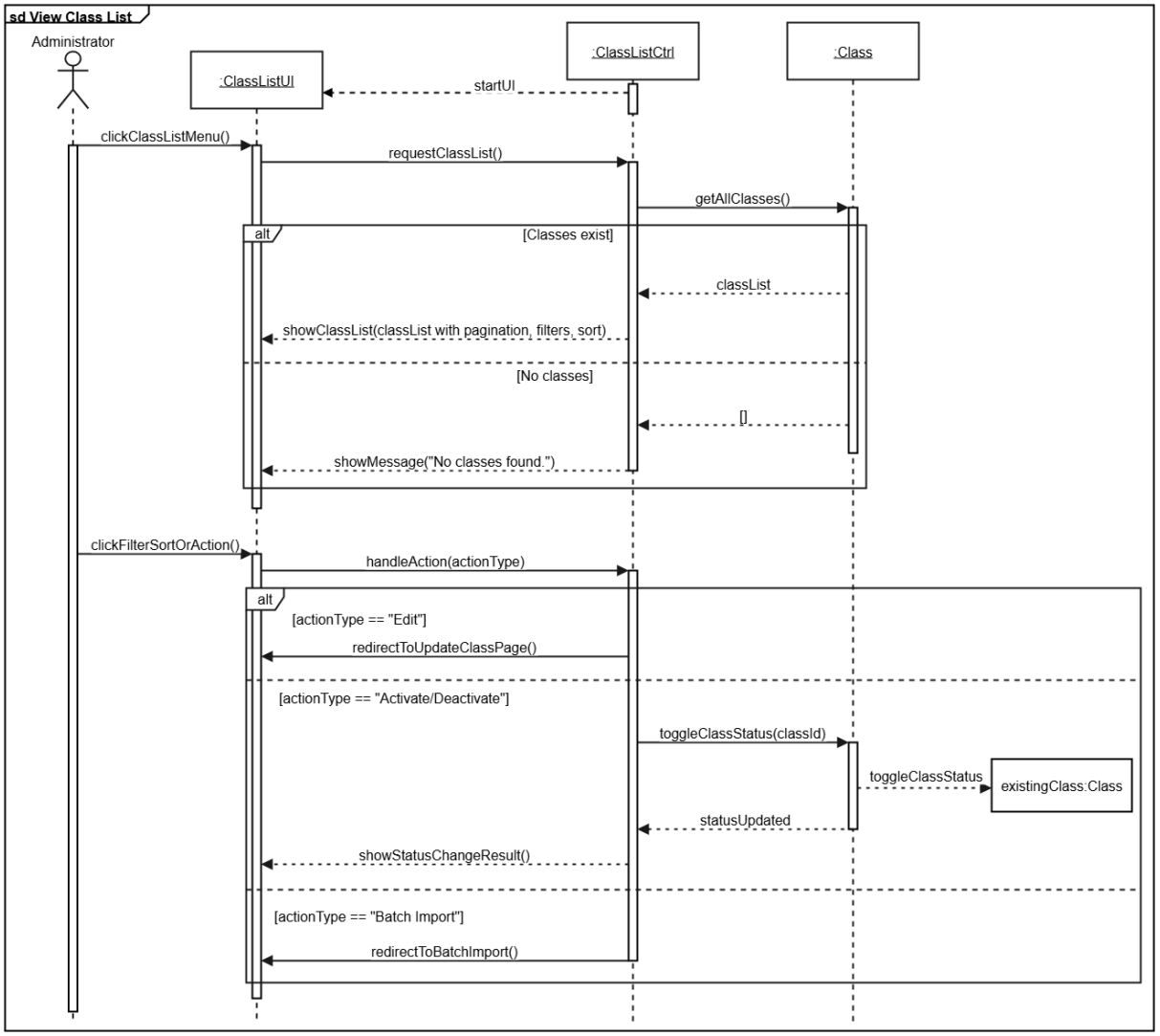


Figure 4.1.2.1 Sequence Diagram for View Class List

Create Class

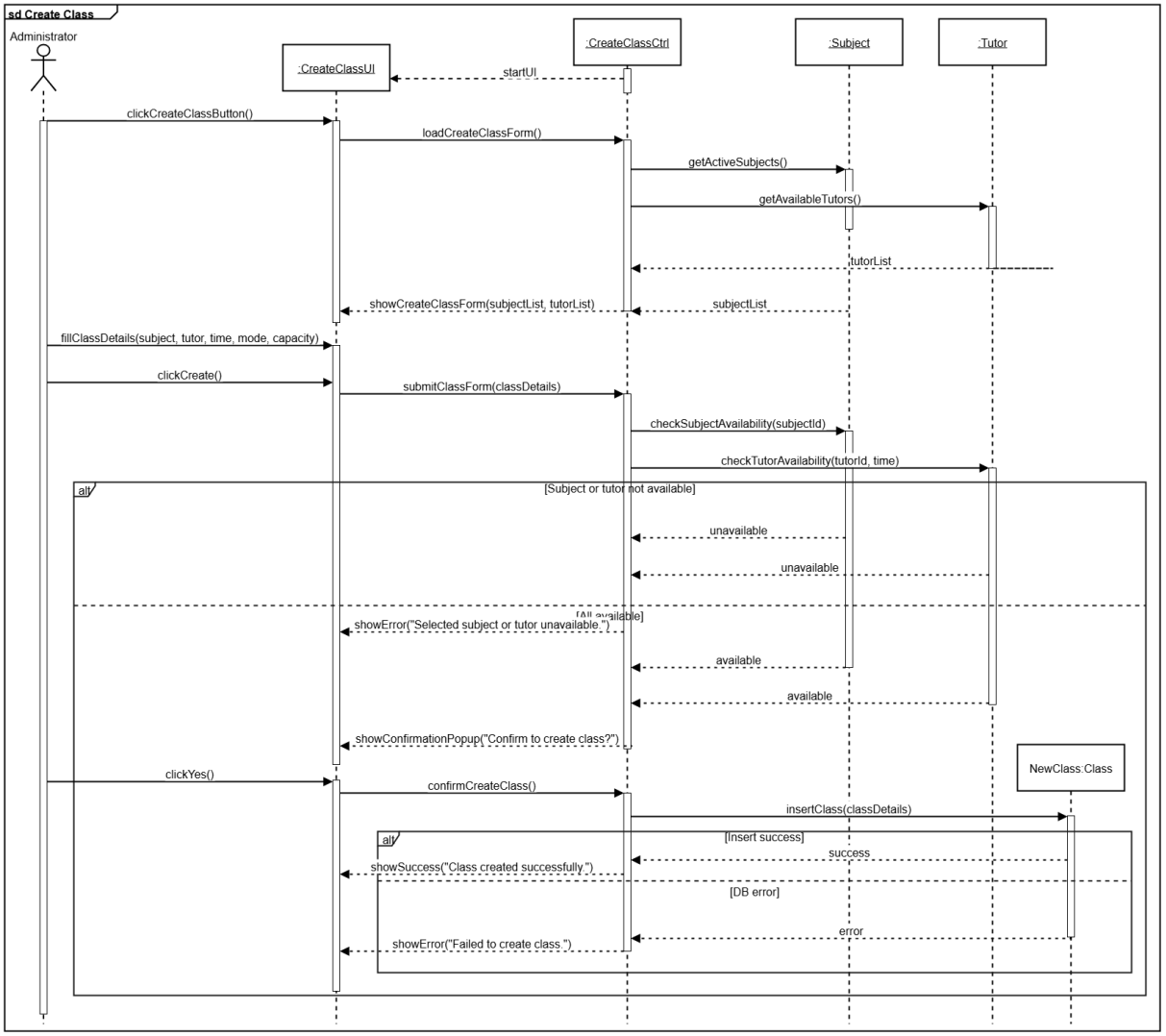


Figure 4.1.2.2 Sequence Diagram for Create Class

Update Class

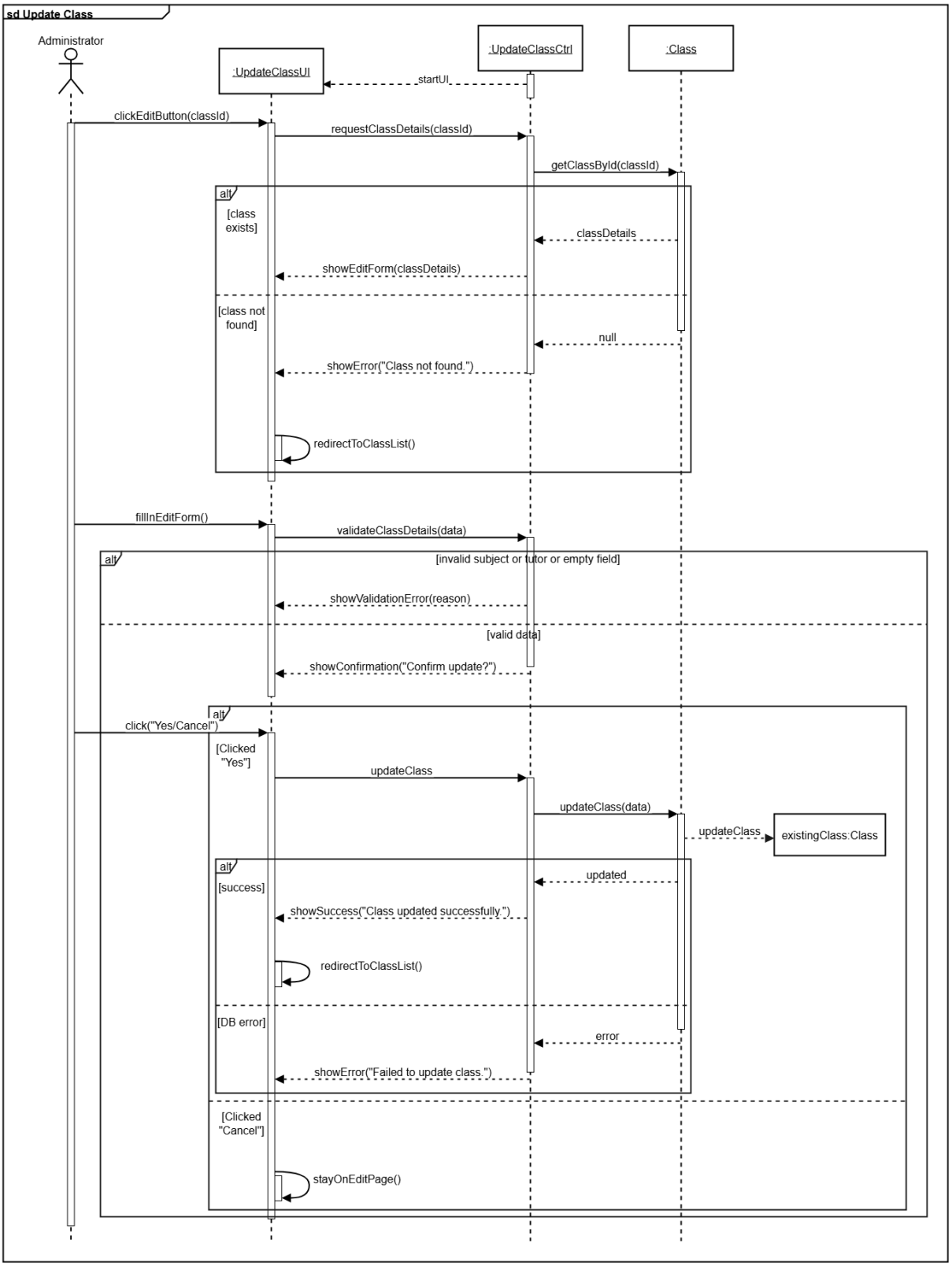


Figure 4.1.2.3 Sequence Diagram for Update Class

Batch Import Class

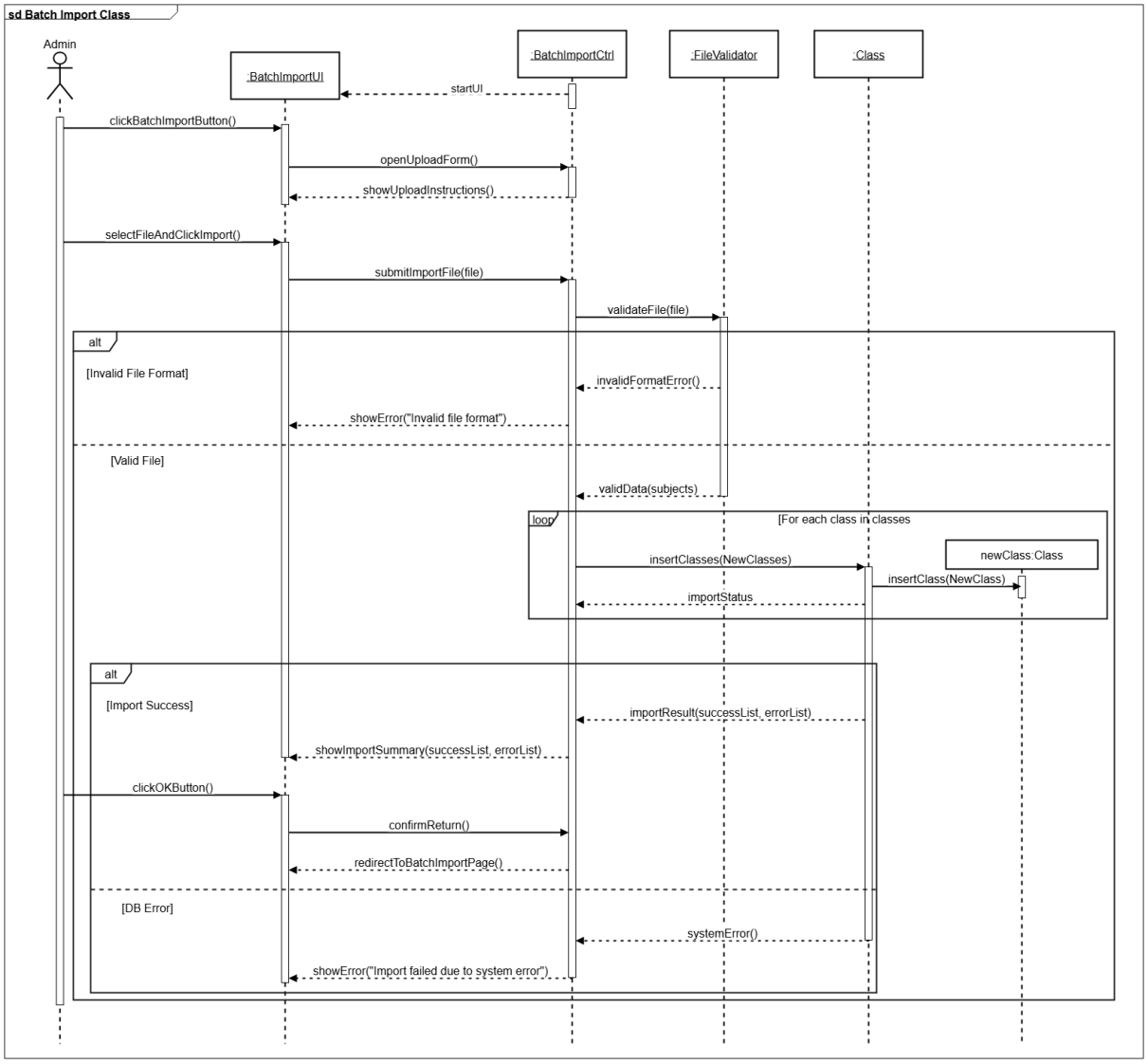


Figure 4.1.2.4 Sequence Diagram for Batch Import Class

View Enrolled Class

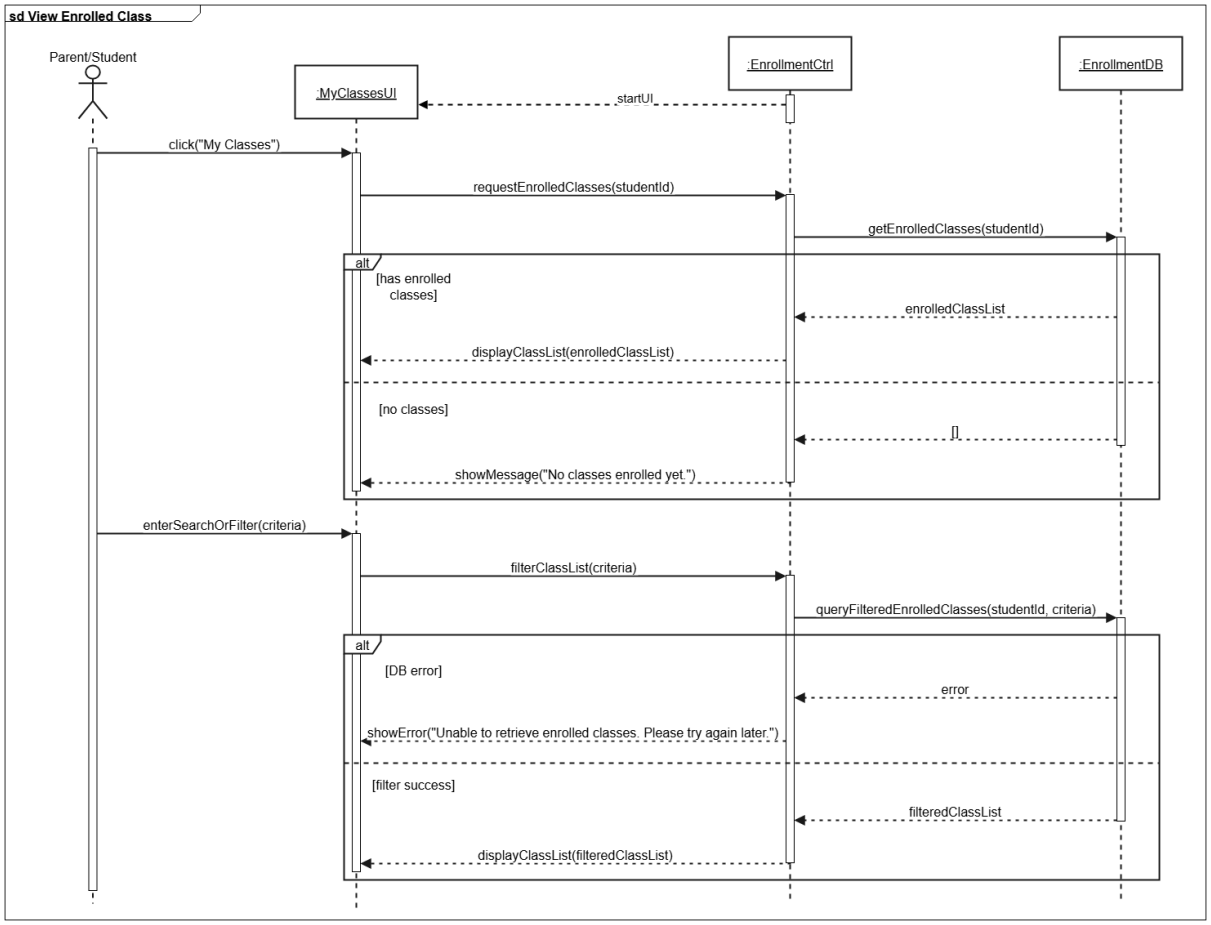


Figure 4.1.2.5 Sequence Diagram for View Enrolled Class

View Available Class

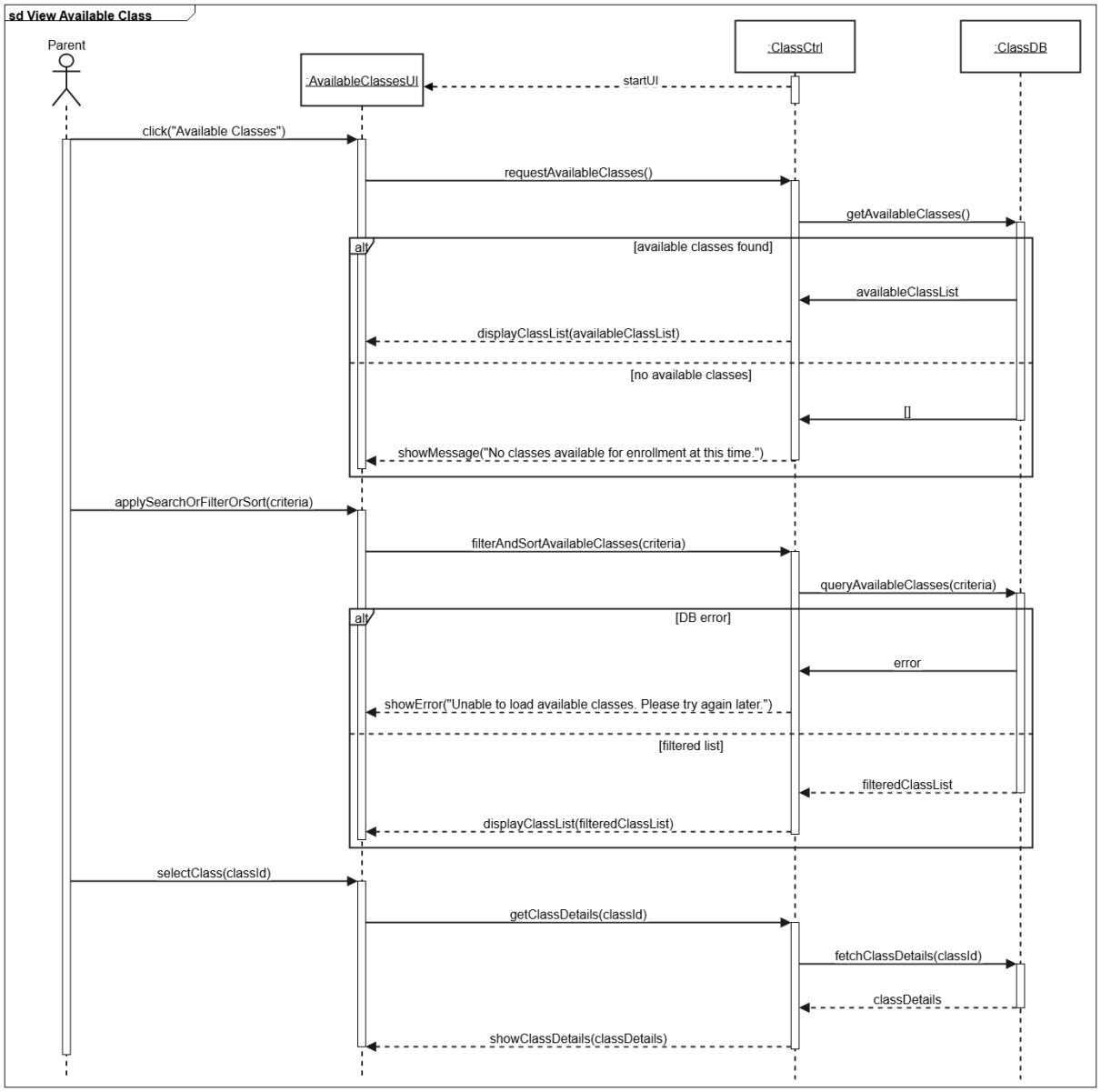


Figure 4.1.2.6 Sequence Diagram for View Available Class

Enroll In Multiple Class

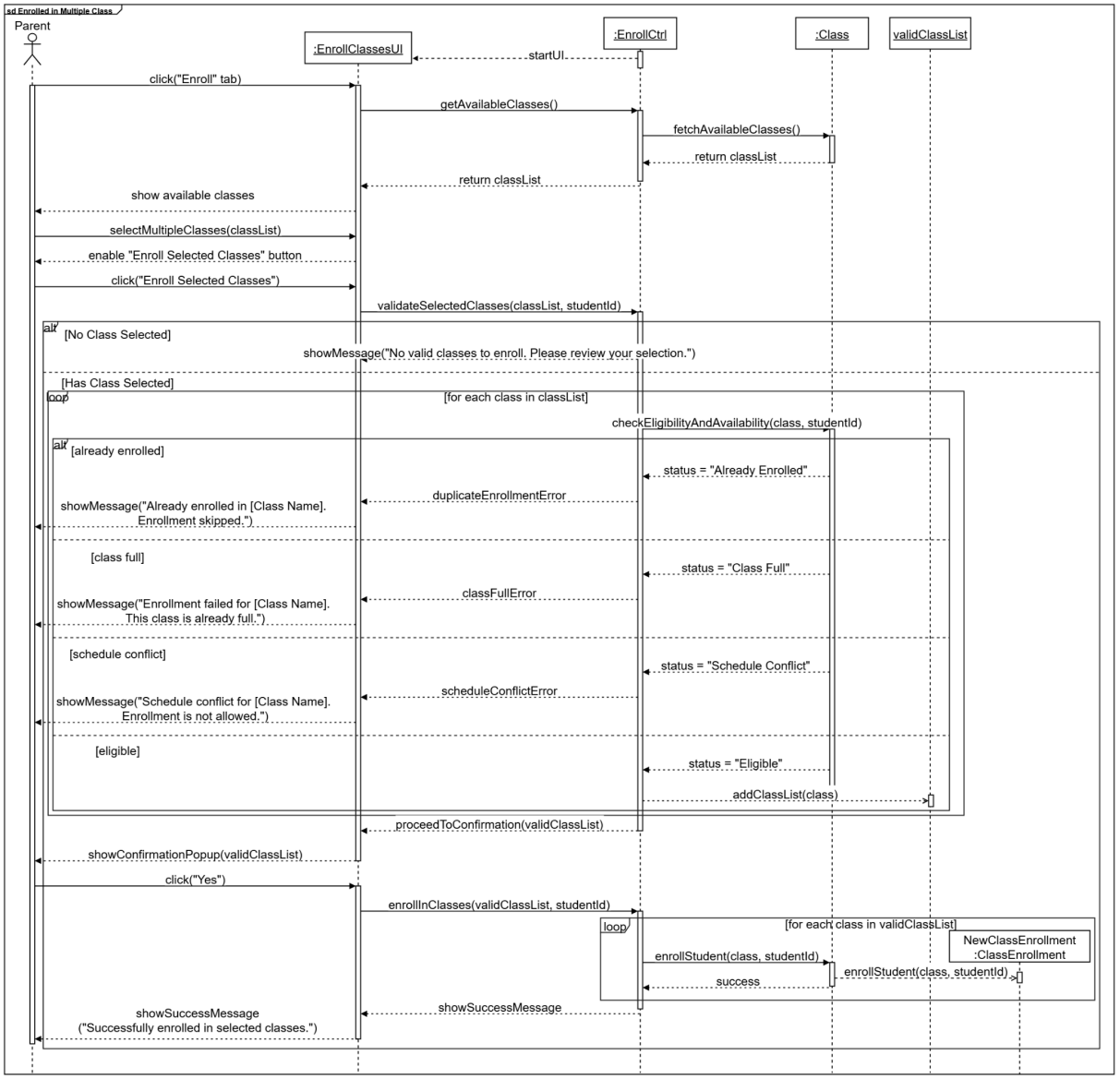


Figure 4.1.2.7 Sequence Diagram for Enrolled In Multiple Class

4.1.3 Payment Module

Generate Invoice

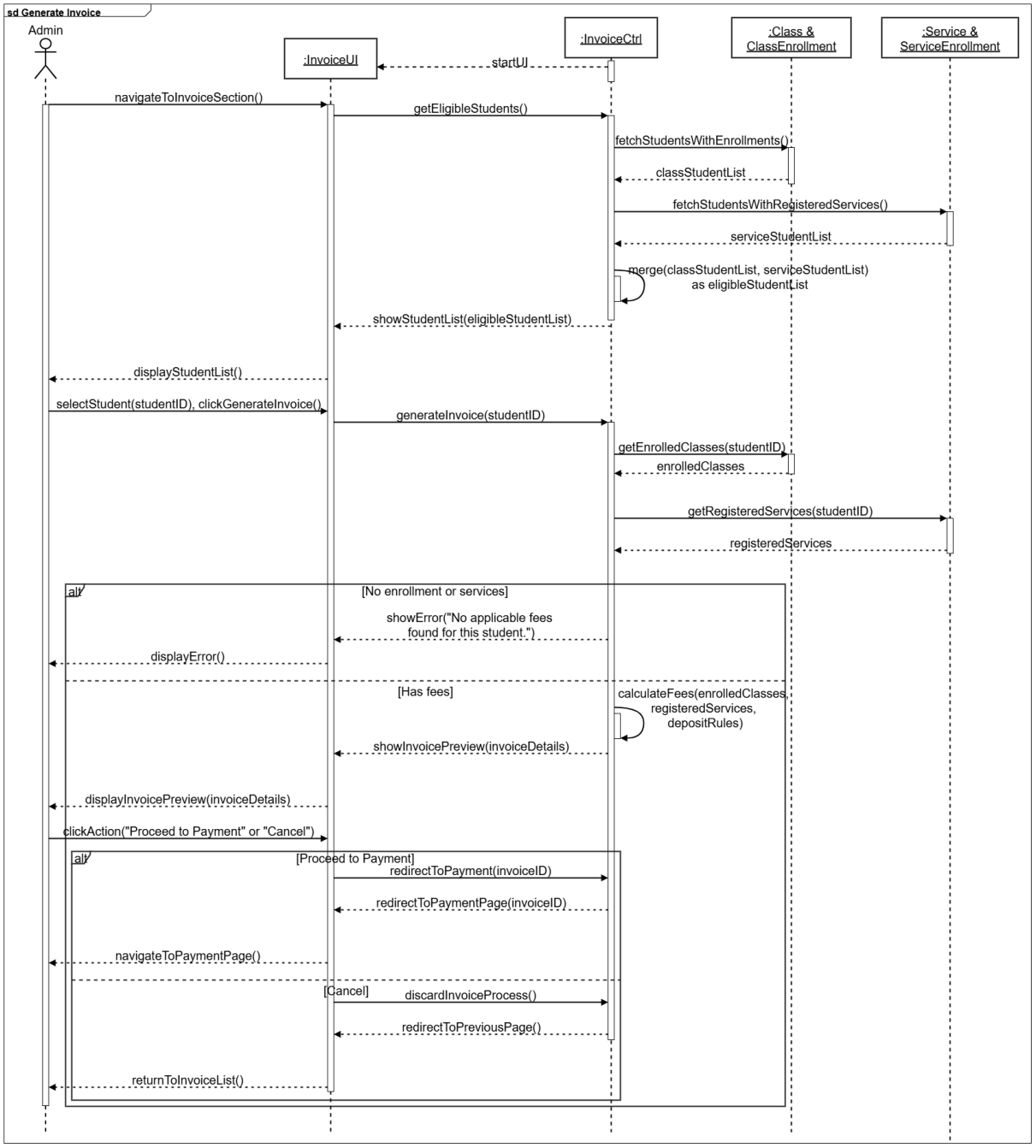


Figure 4.1.3.1 Sequence Diagram for Generate Invoice

Record Offline Payment

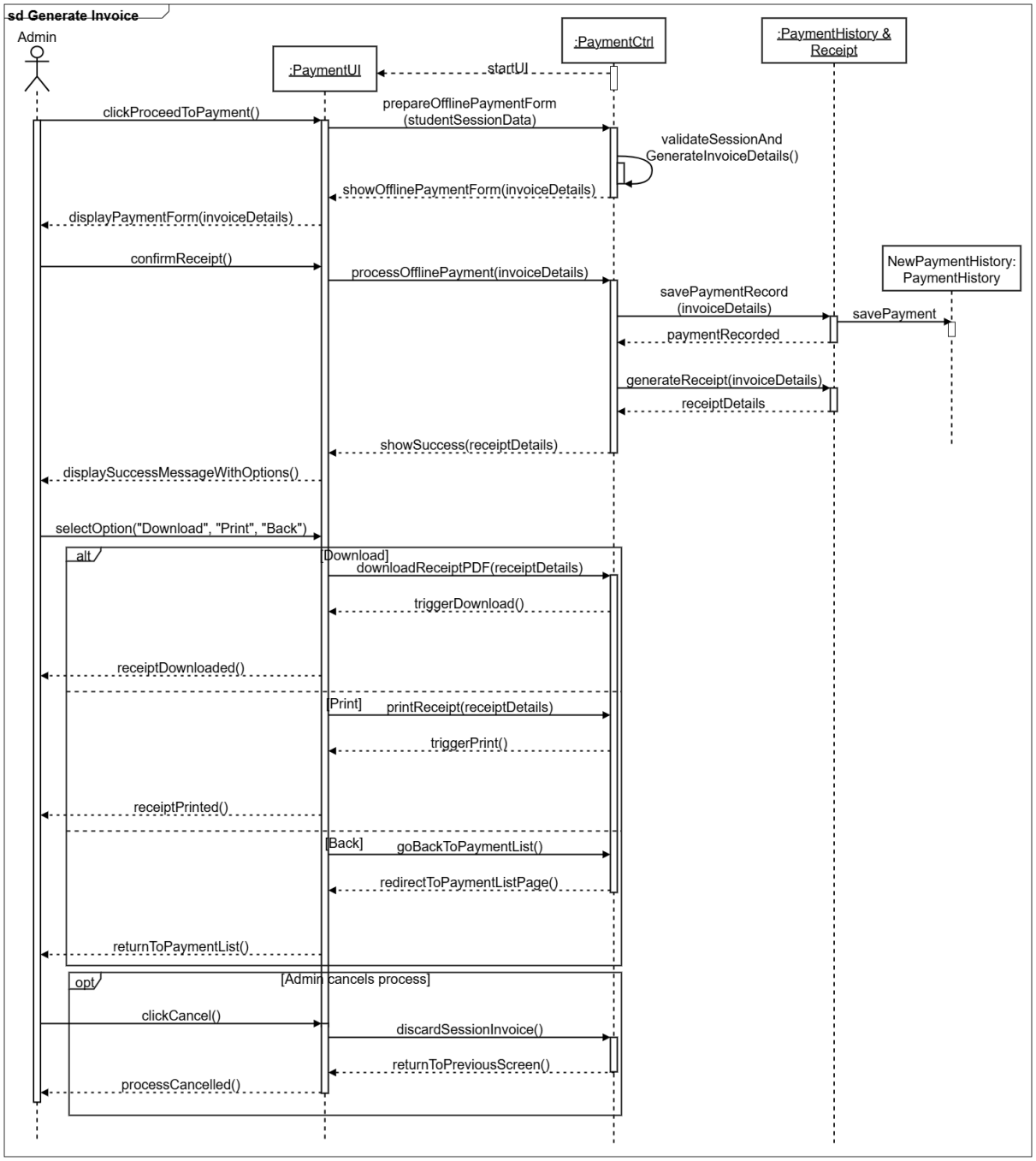


Figure 4.1.3.2 Sequence Diagram for Record Offline Payment

View Payment History

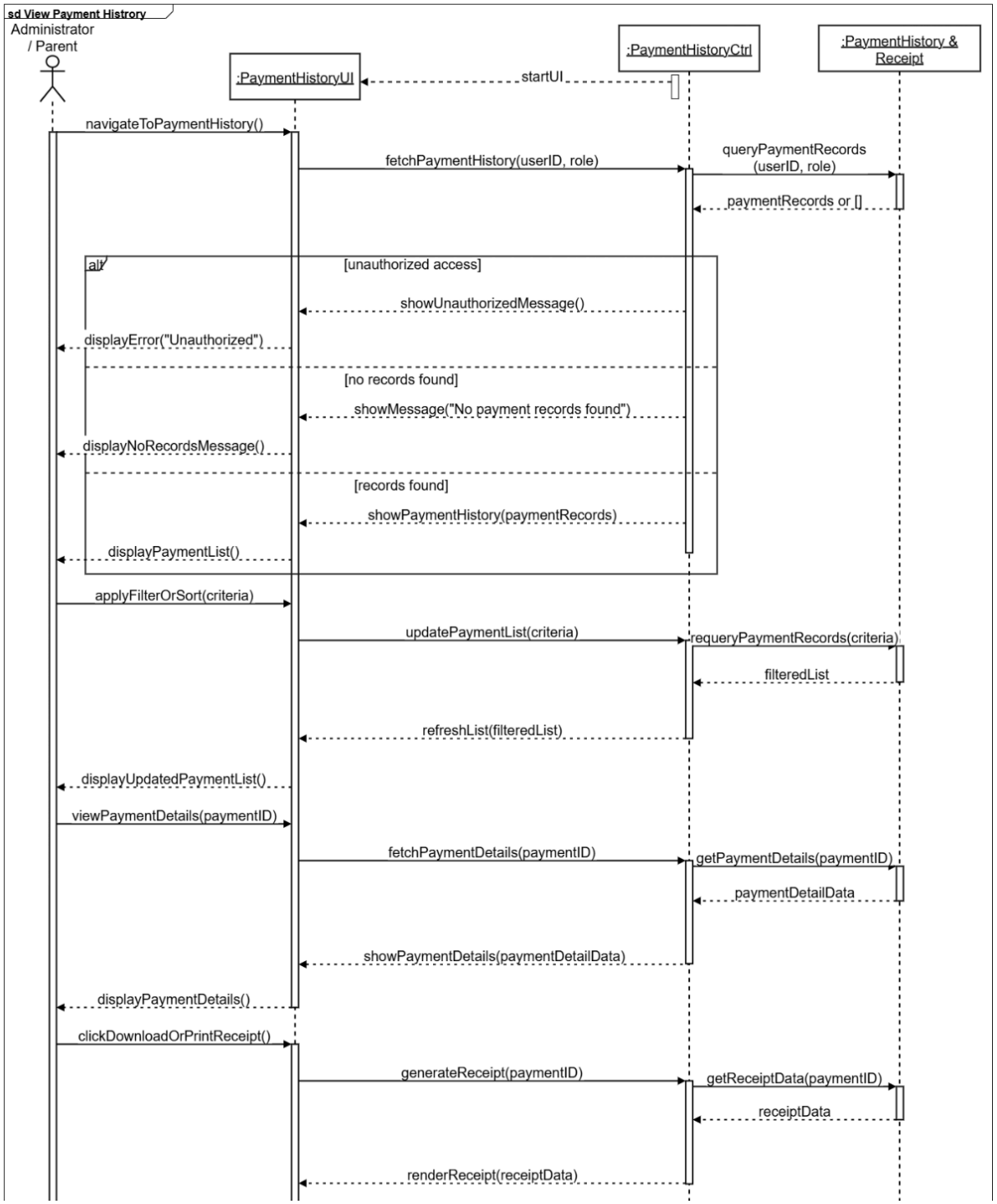


Figure 4.1.3.3 Sequence Diagram for View Payment History (Part 1)

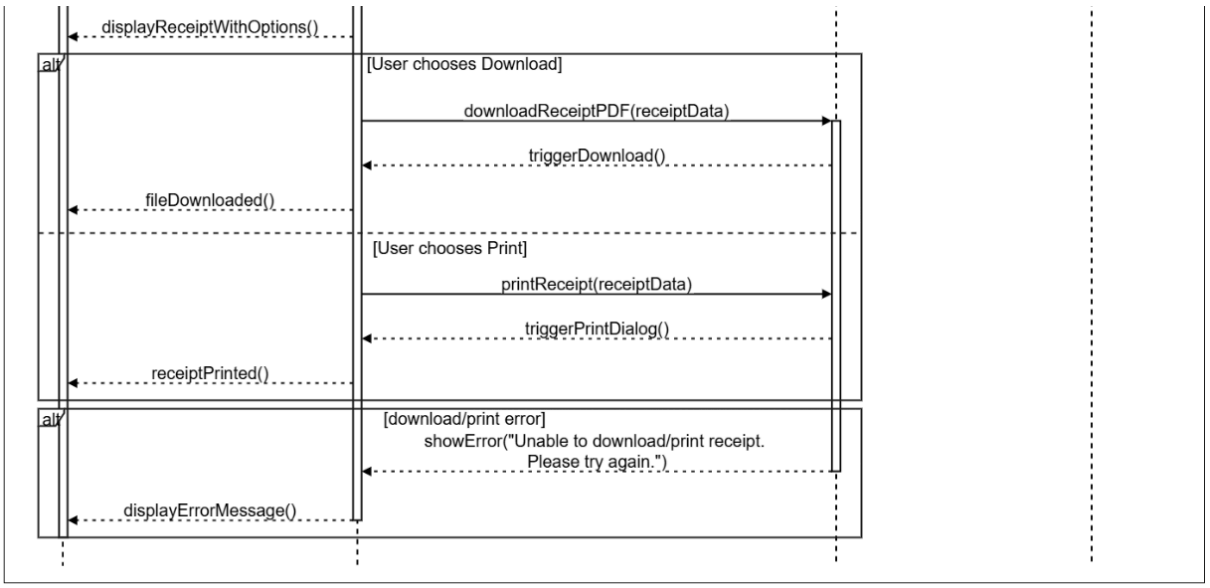


Figure 4.1.3.3 Sequence Diagram for View Payment History (Part 2)

Make Online Payment

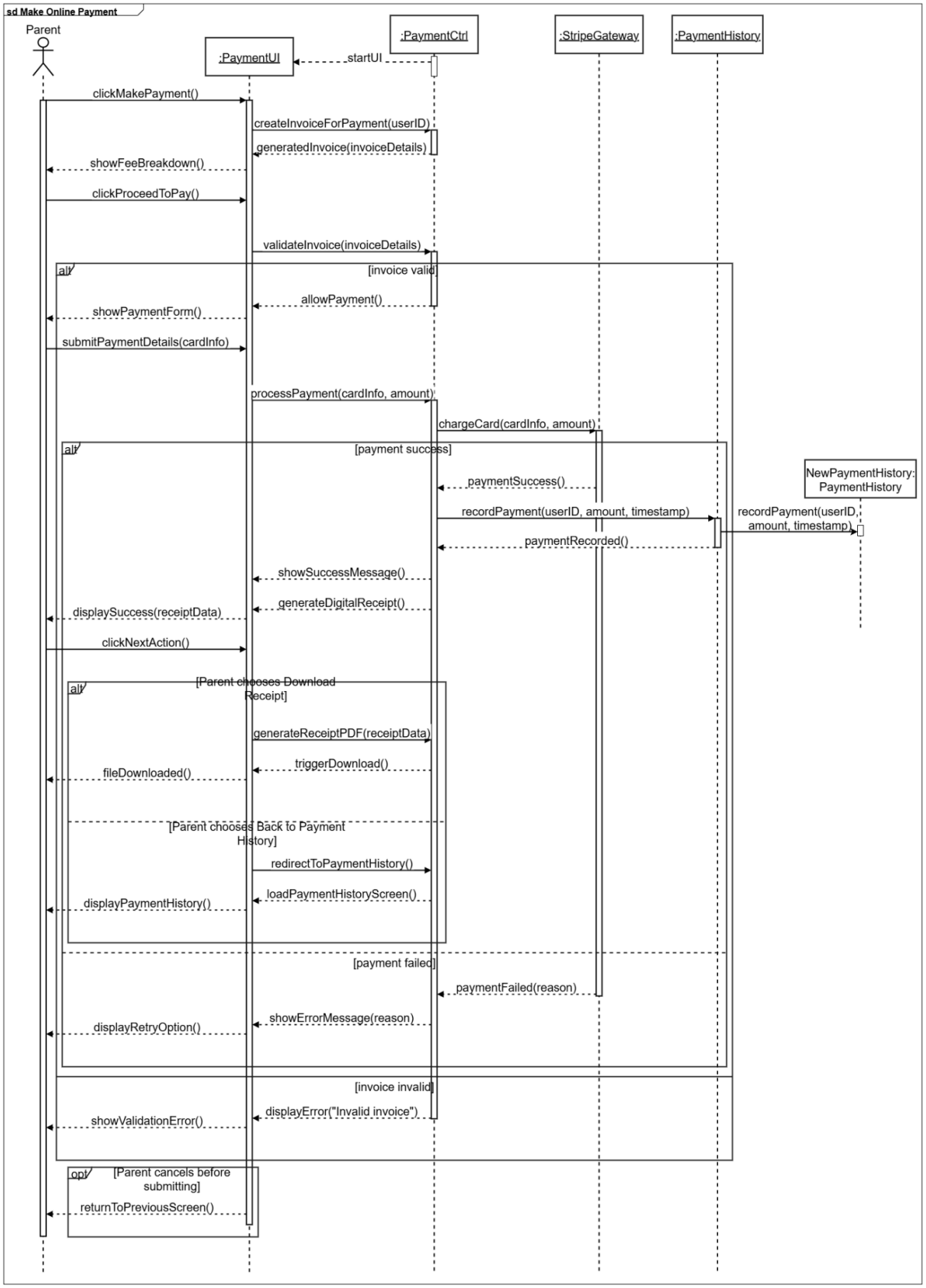


Figure 4.1.3.4 Sequence Diagram for Make Online Payment

4.1.4 Other Service Module

View Other Service List

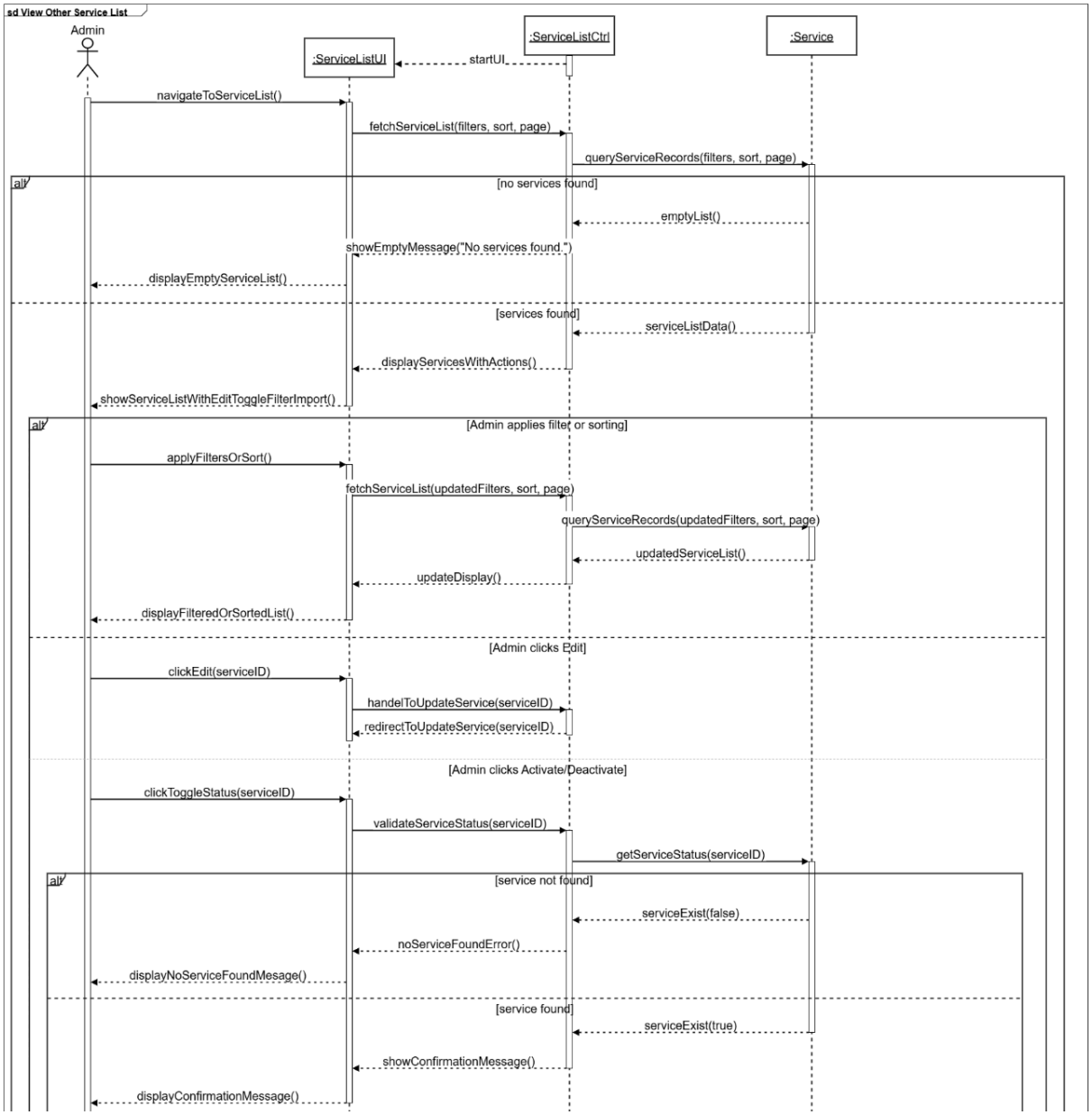


Figure 4.1.4.1.1 Sequence Diagram for View Other Service List (Part 1)

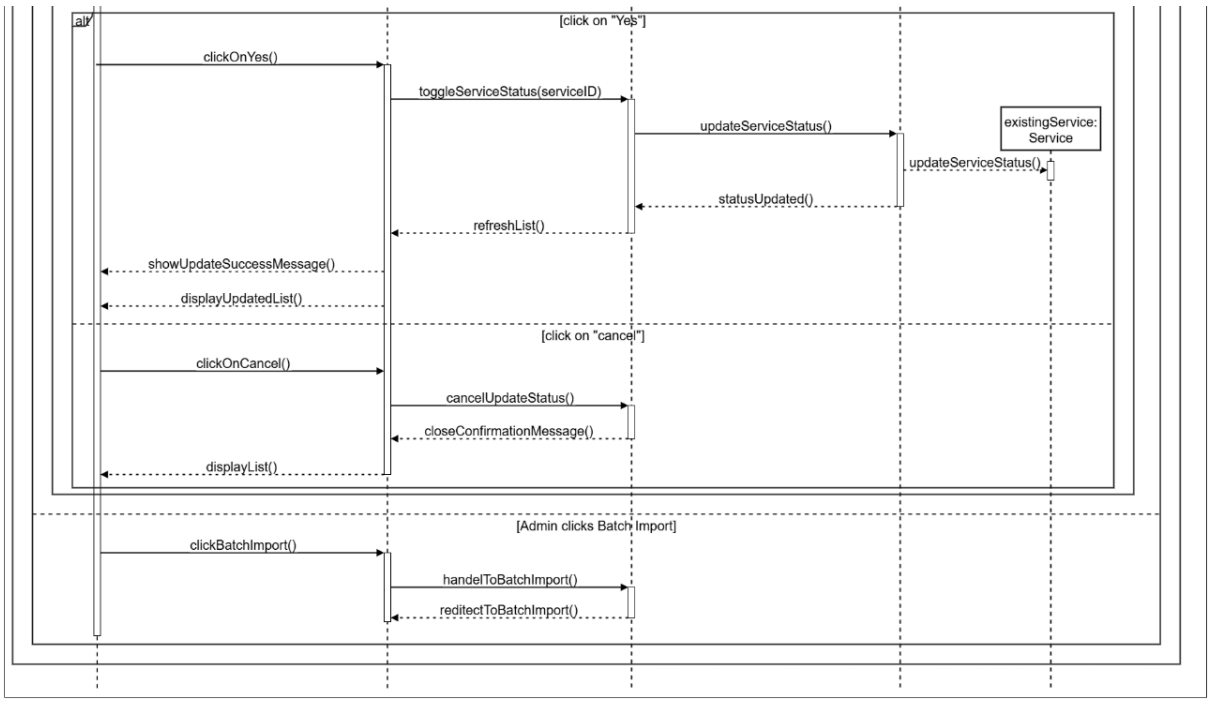


Figure 4.1.4.1.2 Sequence Diagram for View Other Service List (Part 2)

Create Other Service

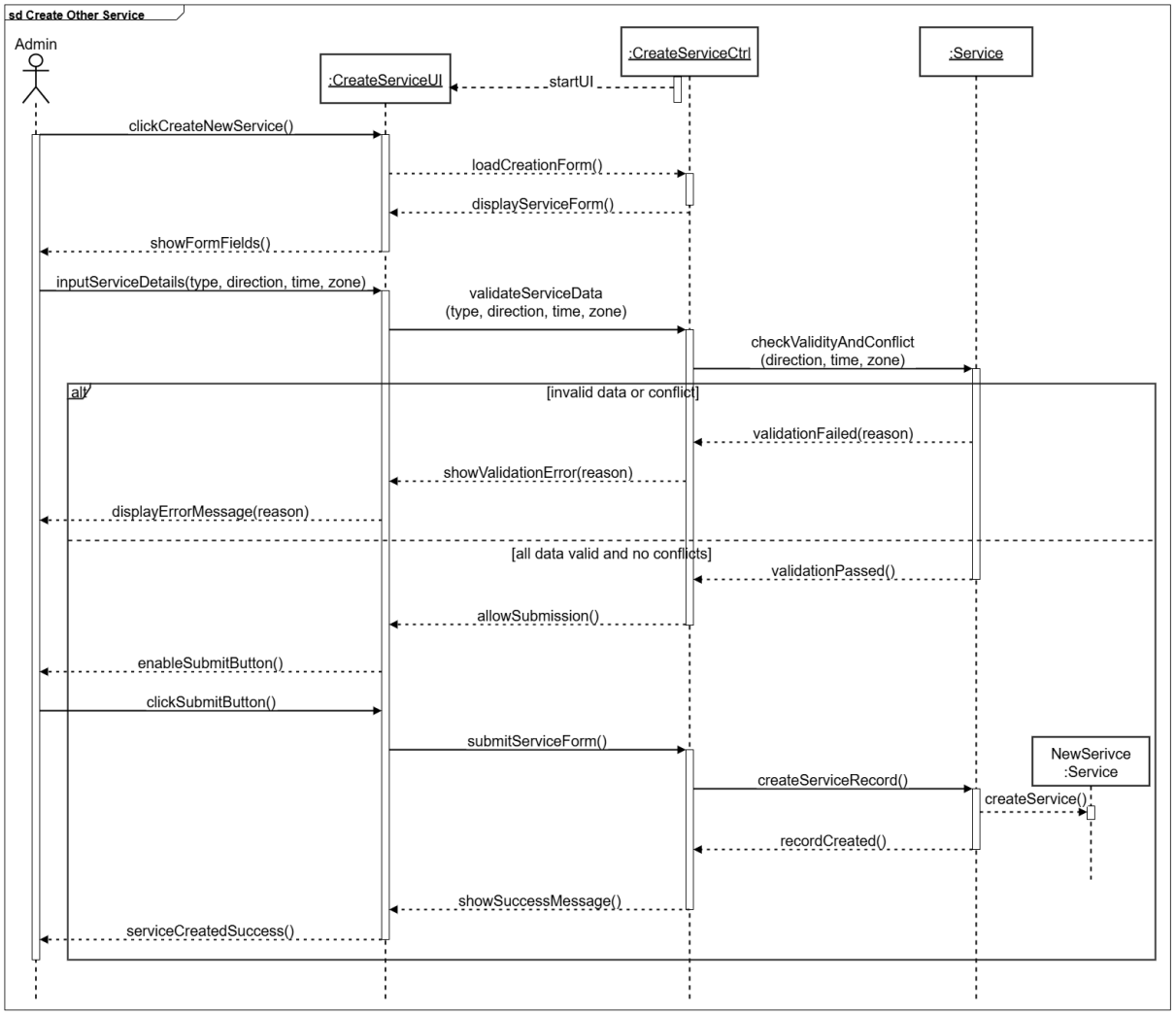


Figure 4.1.4.2 Sequence Diagram for Create Other Service

Update Other Service

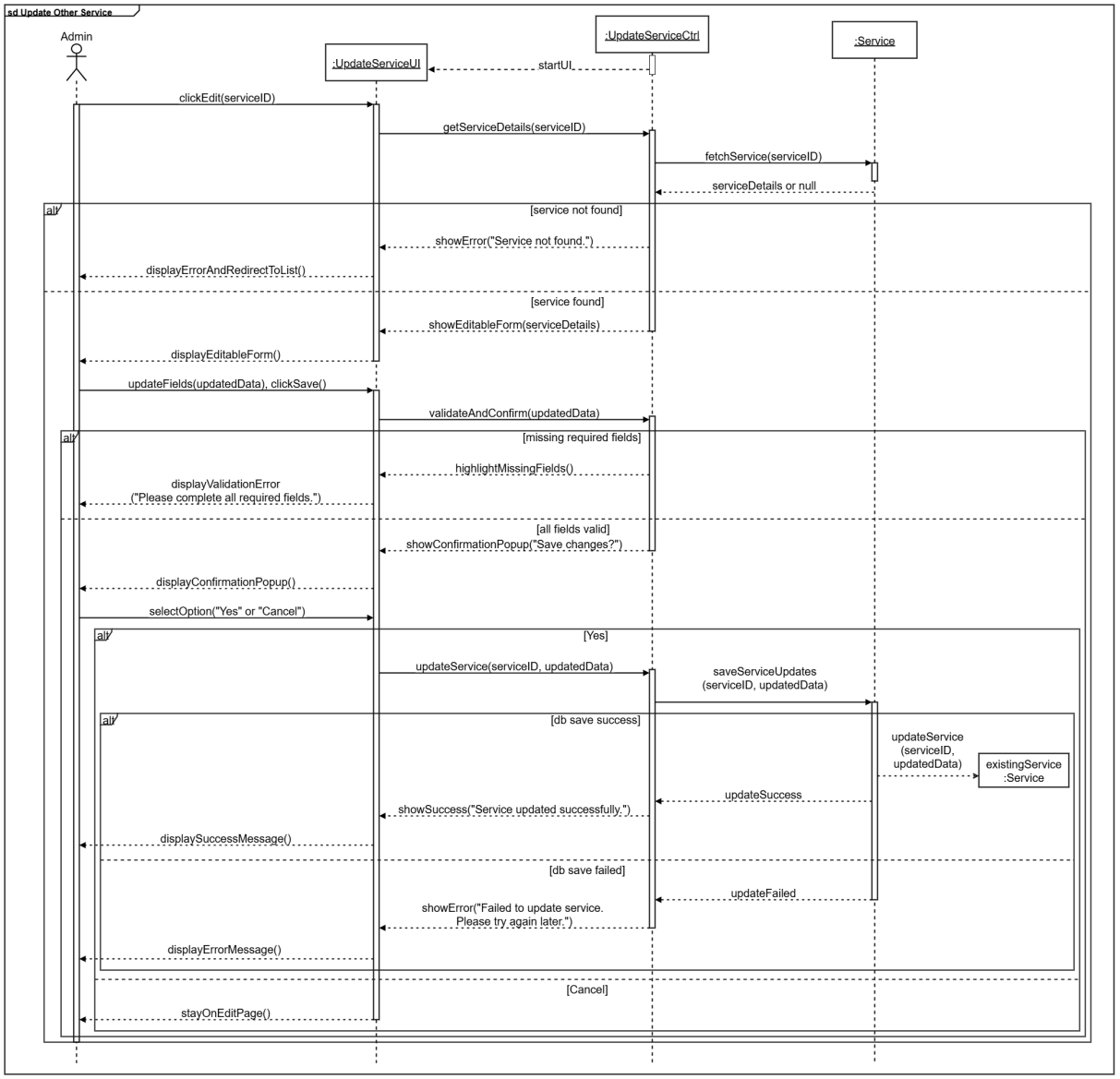


Figure 4.1.4.3 Sequence Diagram for Update Other Service

Batch Import Other Service

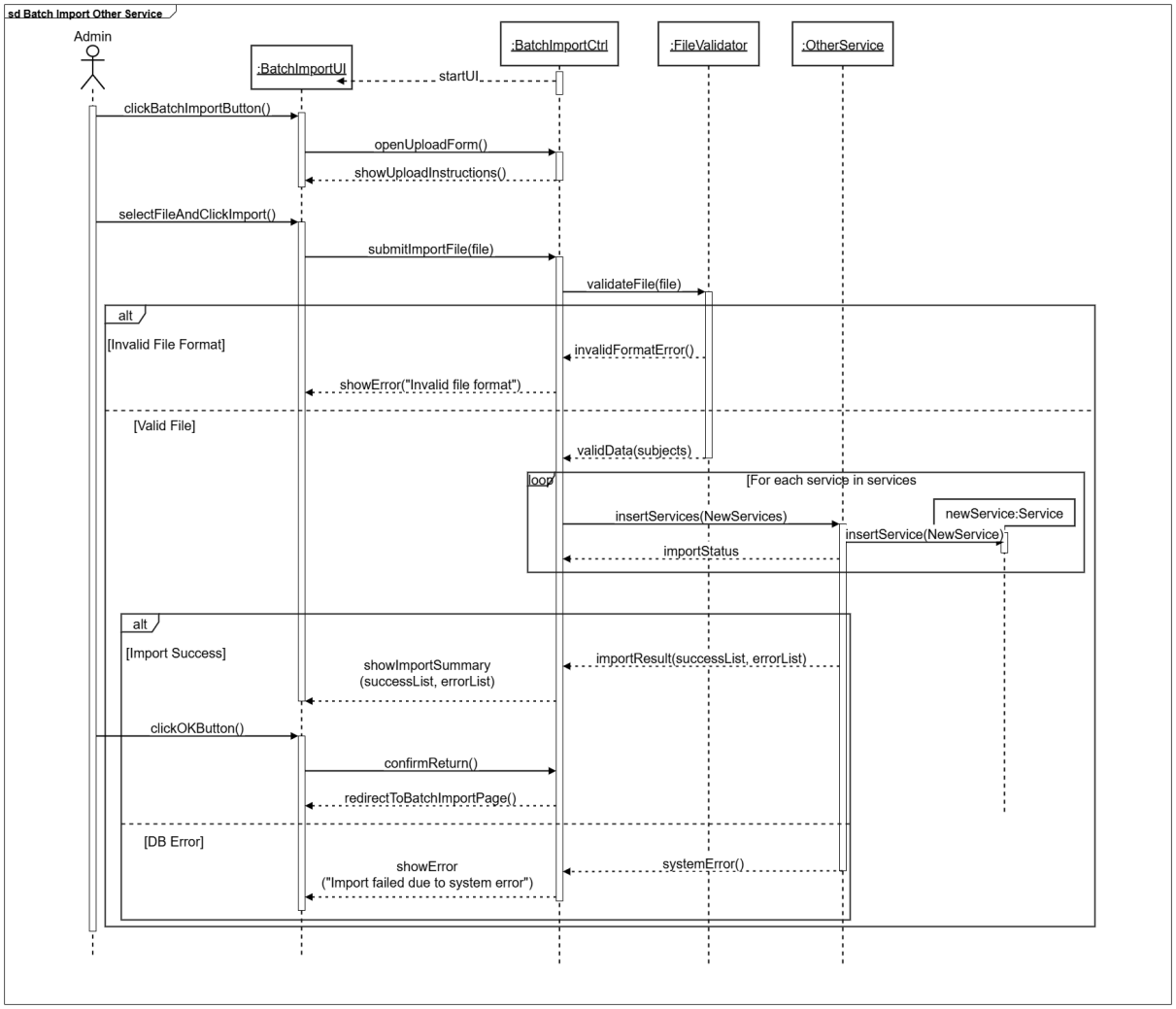


Figure 4.1.4.4 Sequence Diagram for Batch Import Other Service

Cancel Service Enrollment

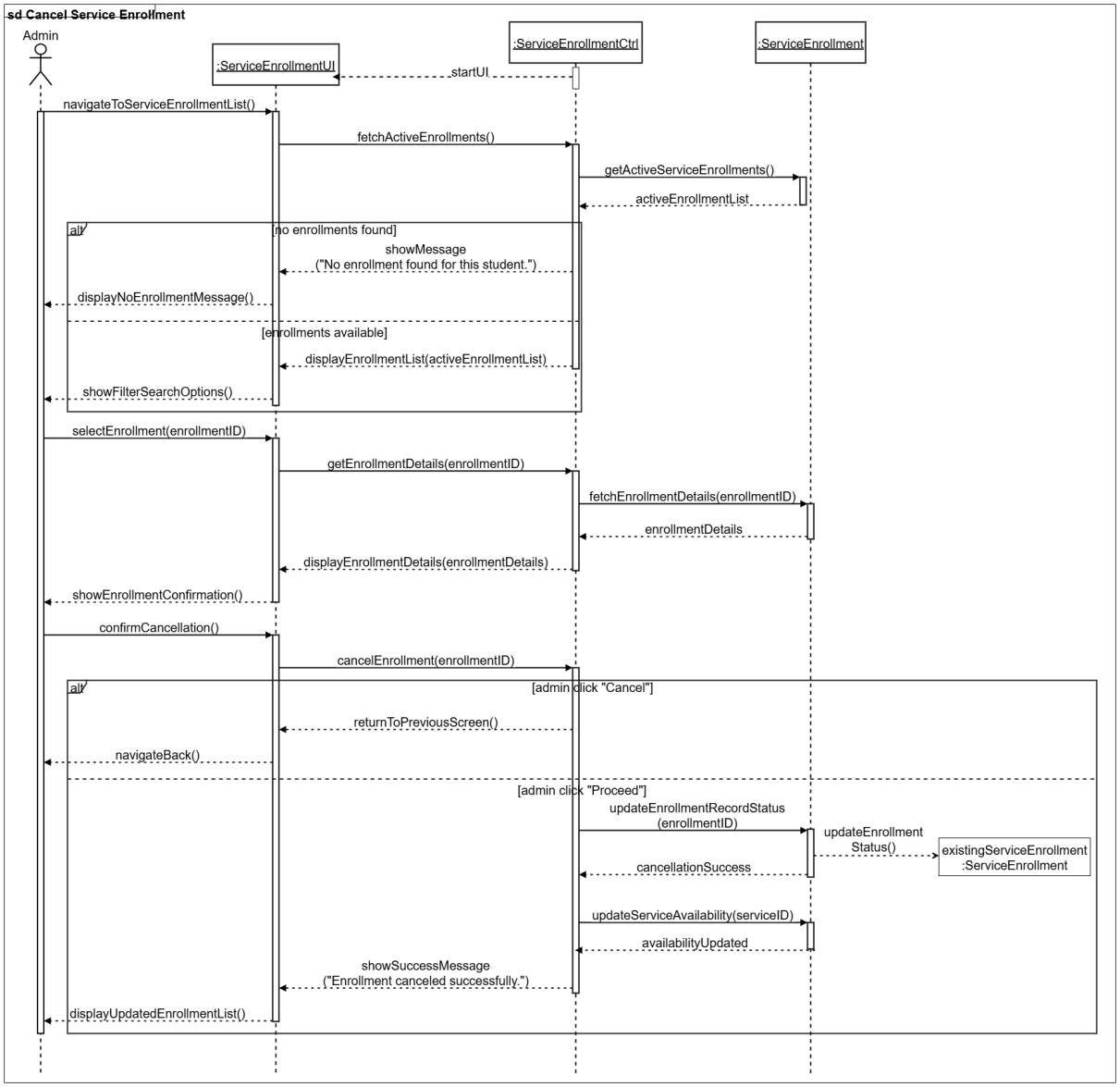


Figure 4.1.4.5 Sequence Diagram for Cancel Service Enrollment

View Available Service

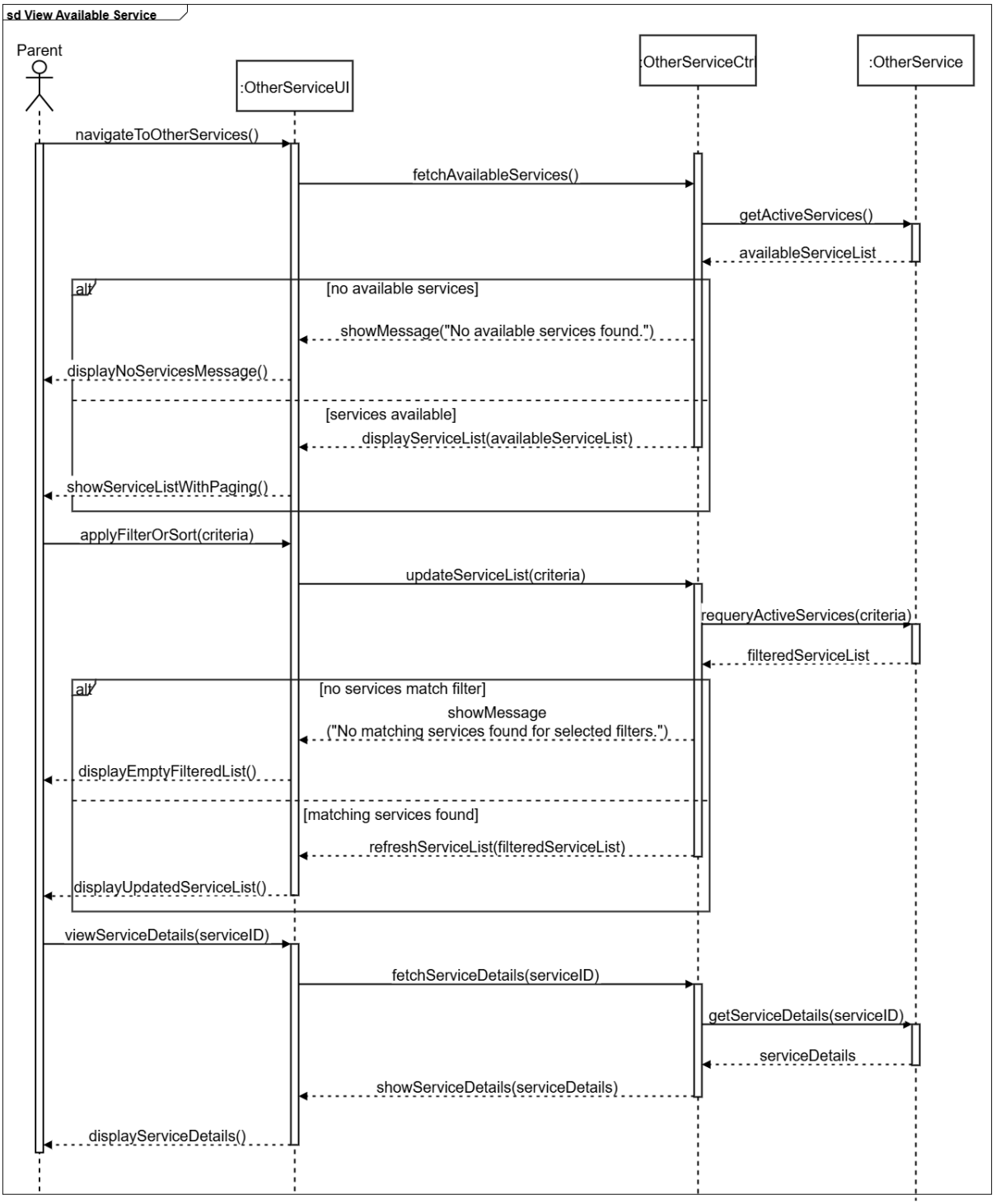


Figure 4.1.4.6 Sequence Diagram for View Available Service

Register Other Service

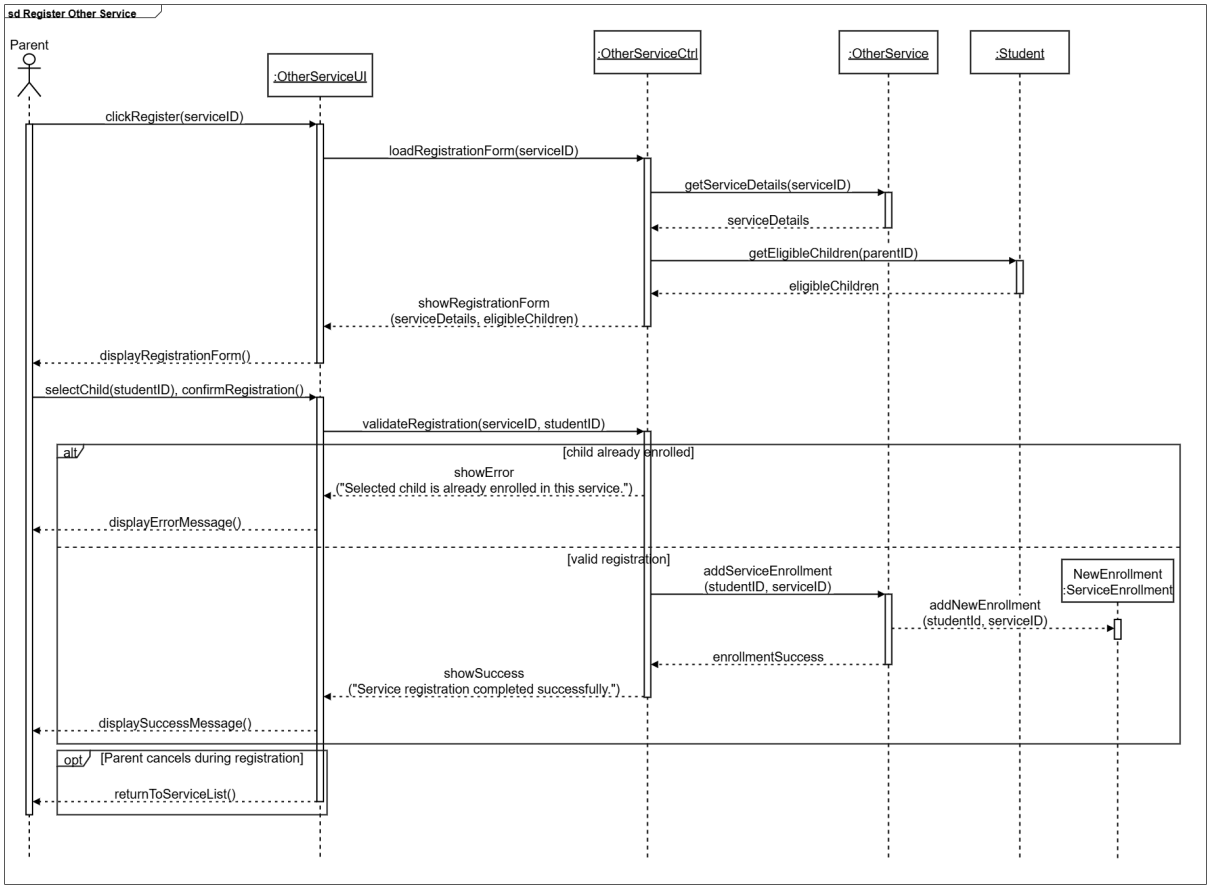


Figure 4.1.4.7 Sequence Diagram for Register Other Service

4.1.5 Chat Module

Create Chat Room

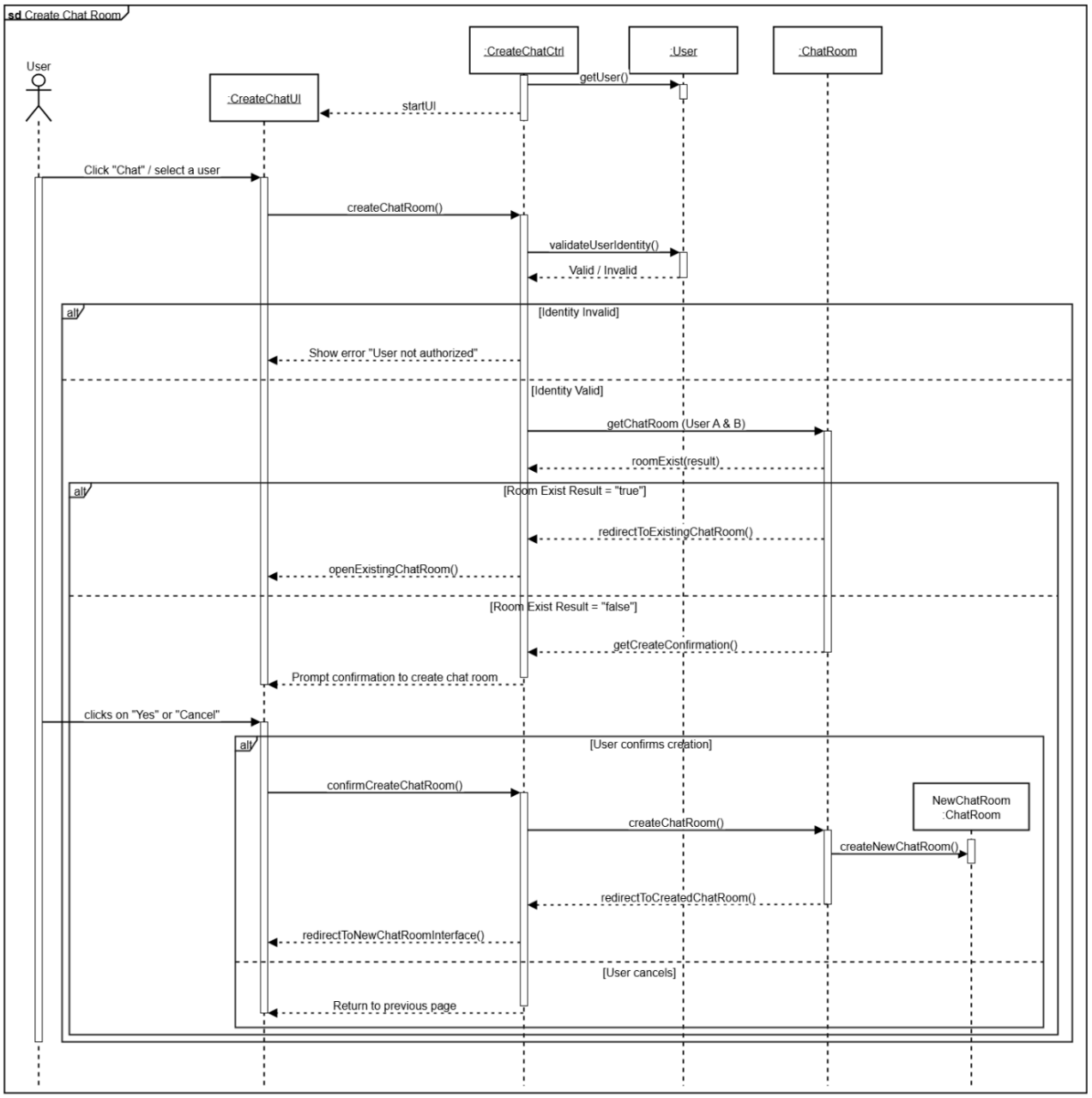


Figure 4.1.5.1 Sequence Diagram of Create Chat Room

Exchange Message

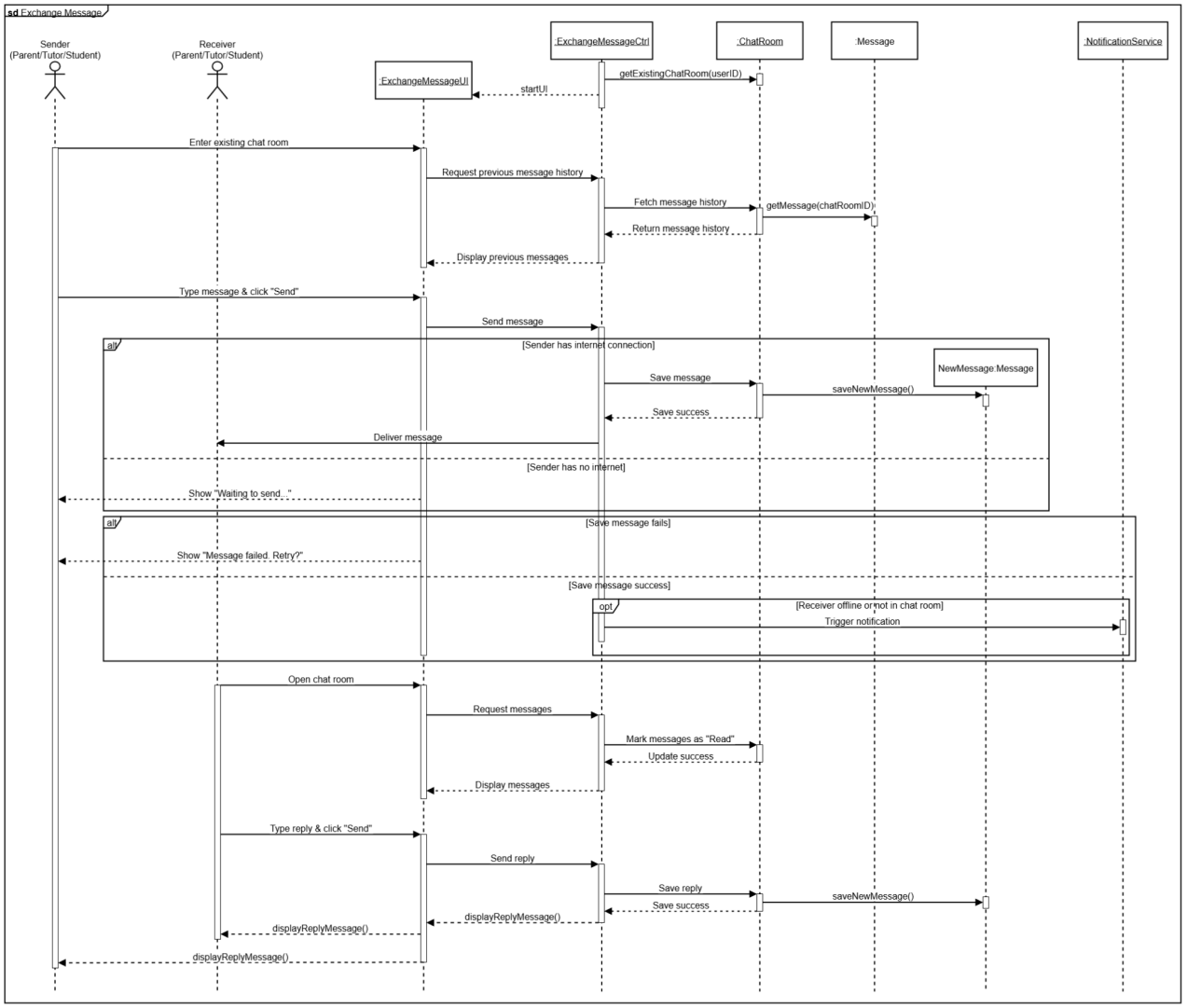


Figure 4.1.5.2 Sequence Diagram of Exchange Message

Close Chat Room

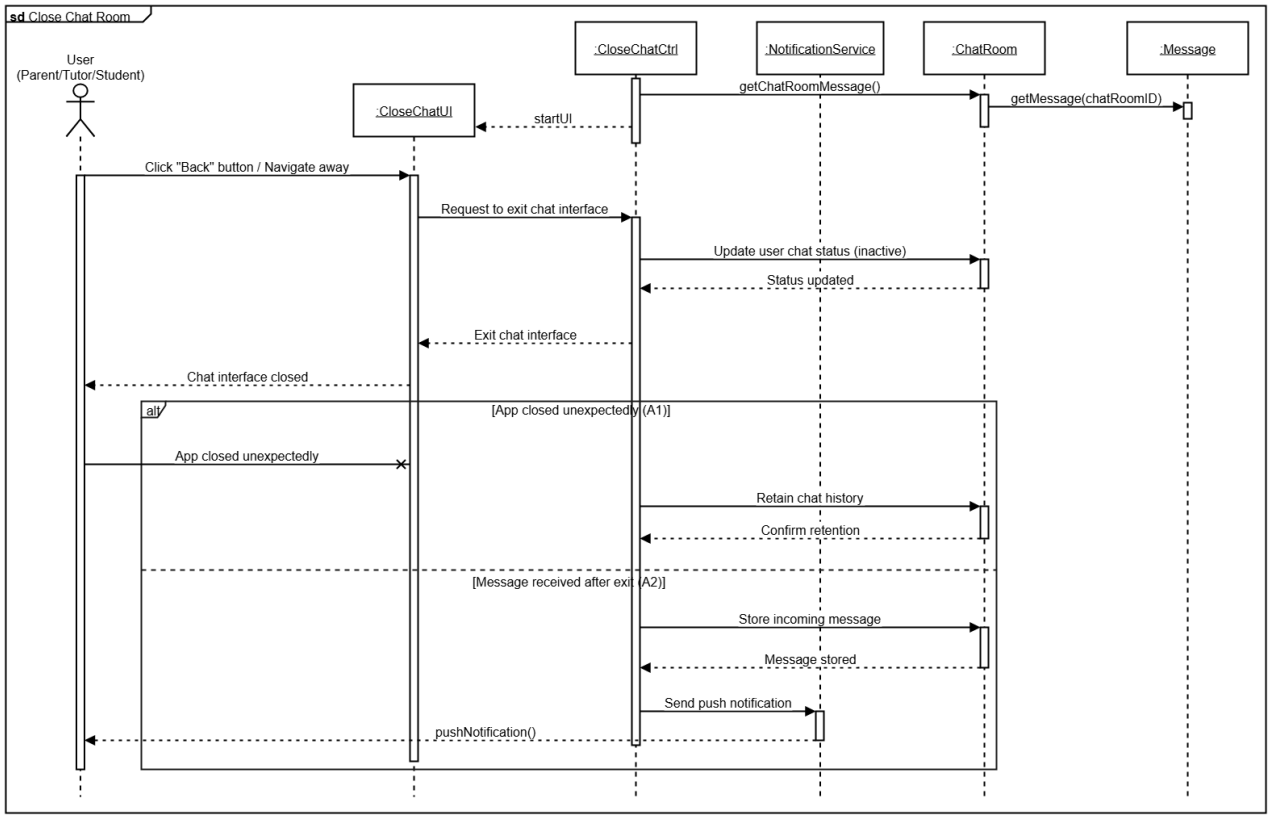


Figure 4.1.5.3 Sequence Diagram of Close Chat Room

4.2 State Chart Diagram

4.2.1 Subject Module

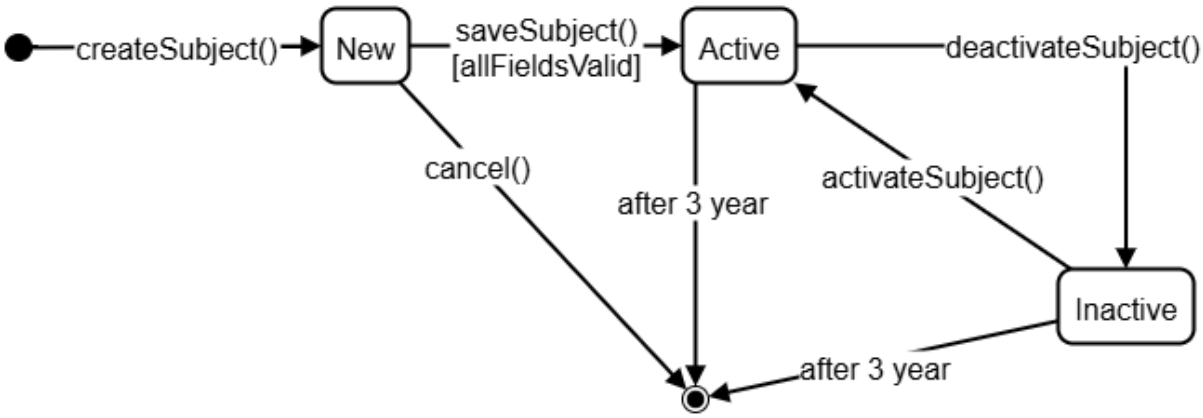


Figure 4.2.1 State Chart Diagram of Subject Module

4.2.2 Class Module

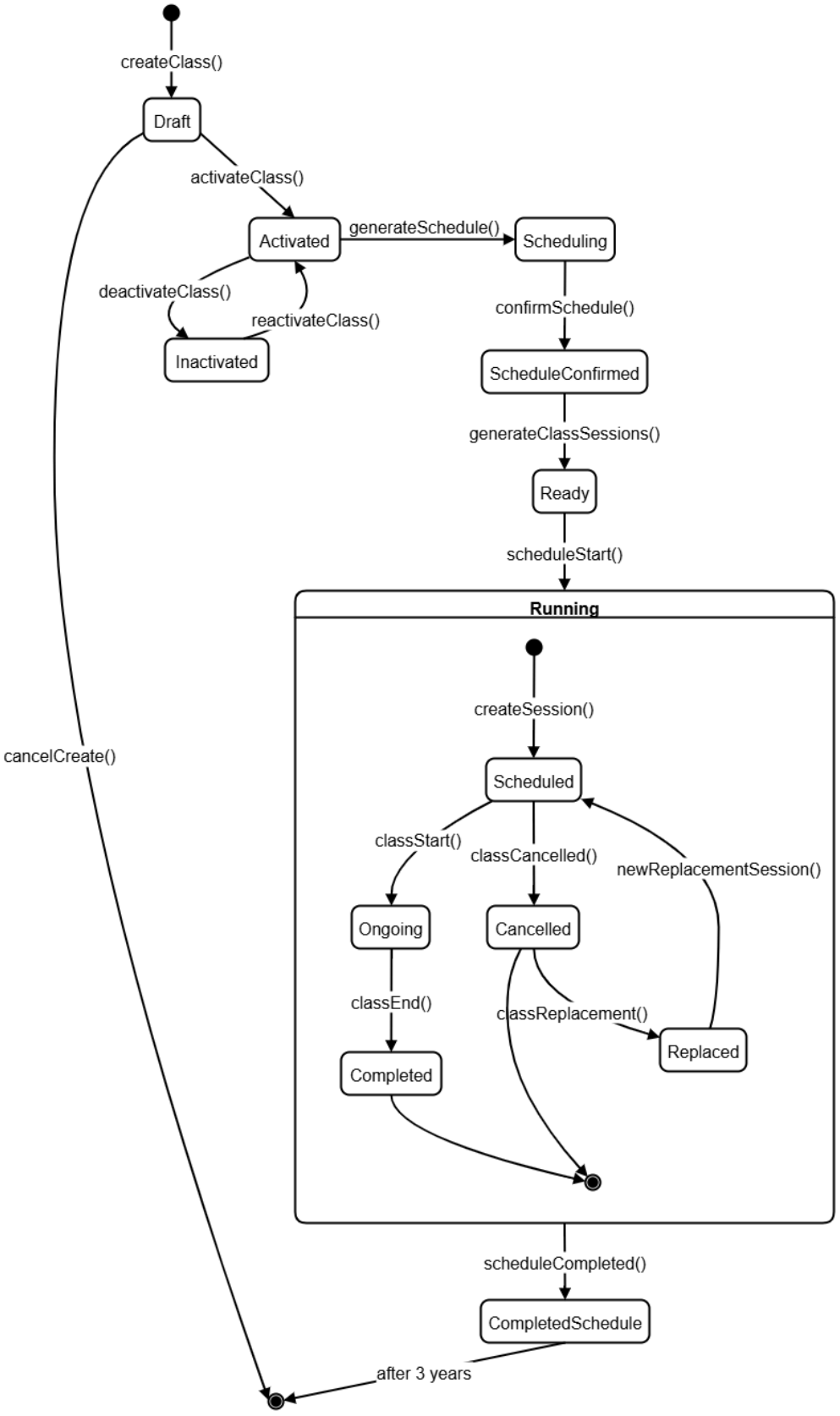


Figure 4.2.2 State Chart Diagram of Class Module

4.2.3 Payment Module

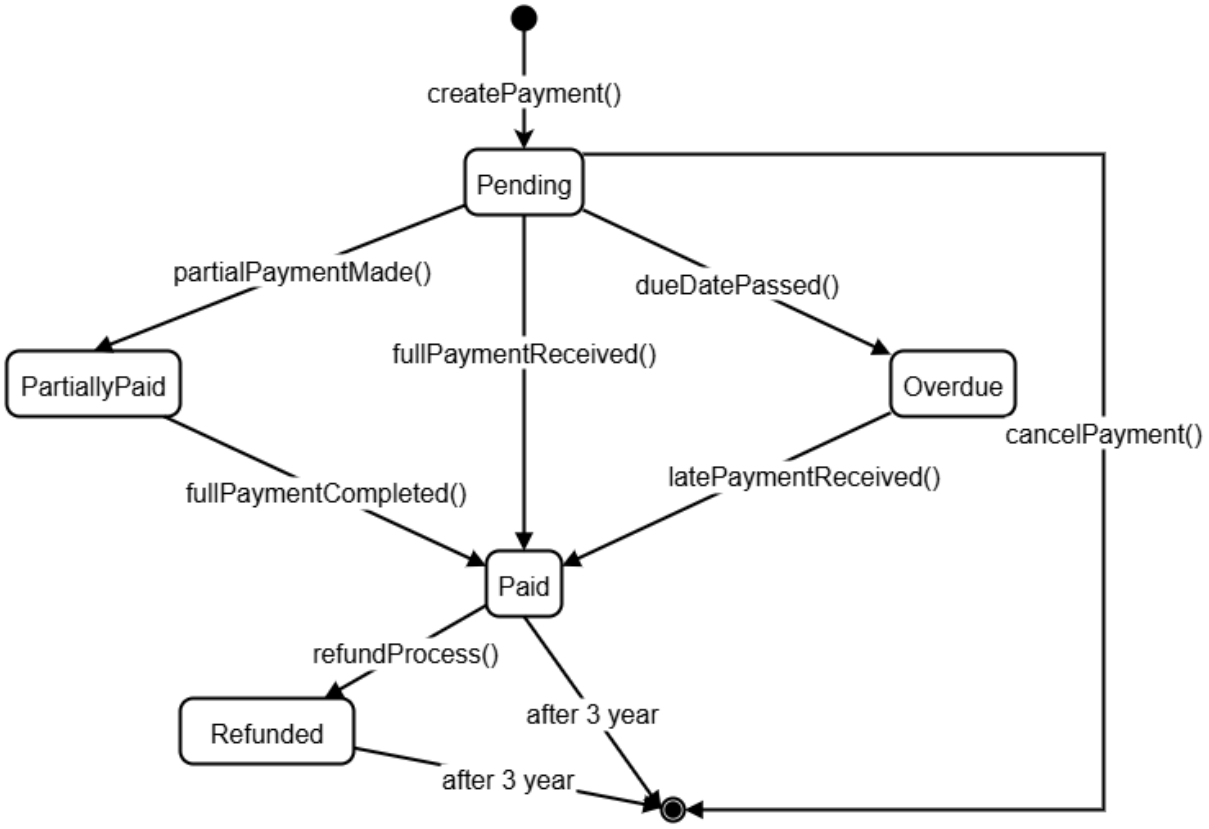


Figure 4.2.3 State Chart Diagram of Payment Module

4.2.4 Other Service Module

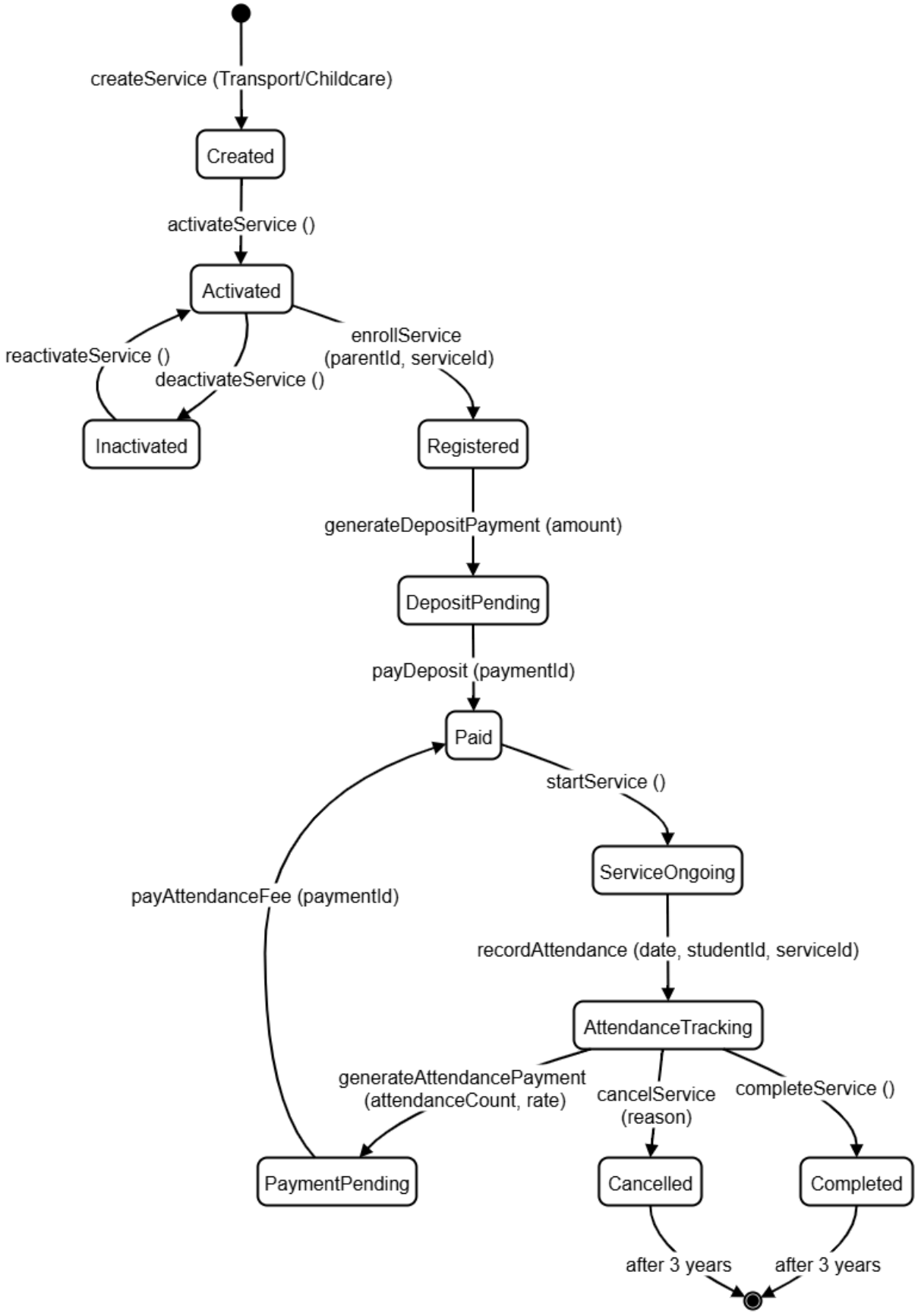


Figure 4.2.4 State Chart Diagram of Other Service Module

4.2.5 Chat Module

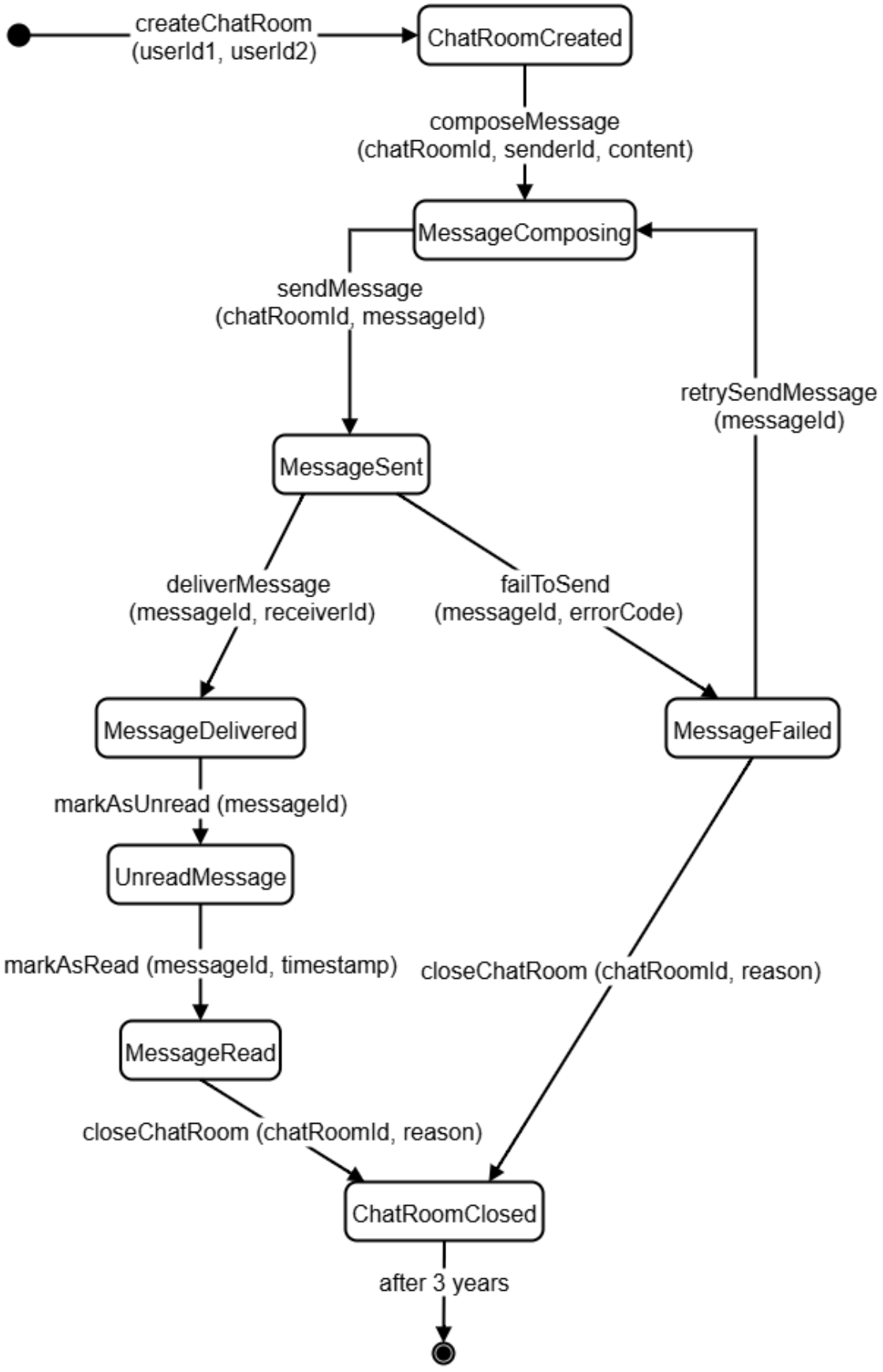
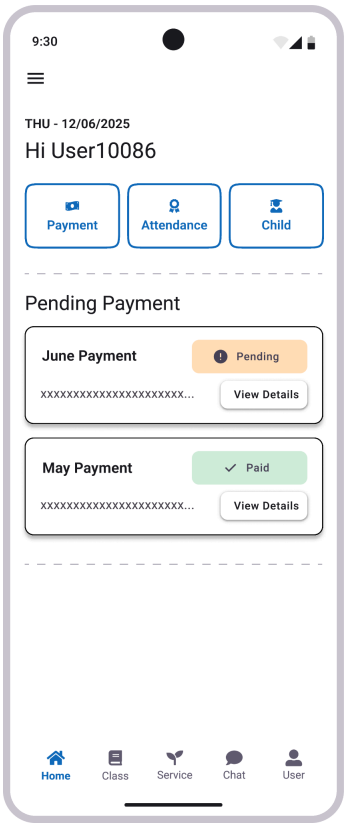
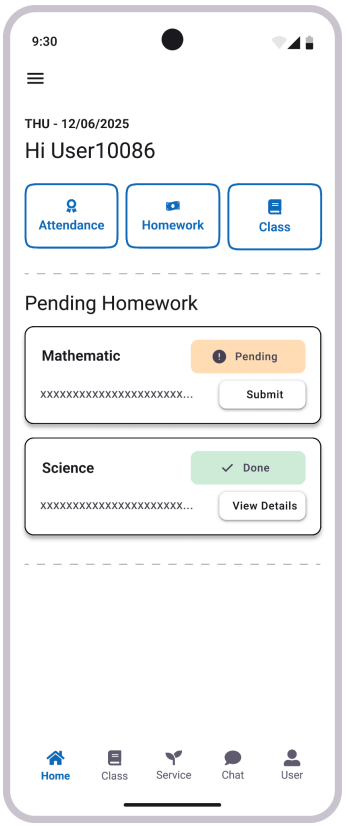


Figure 4.2.5 State Chart Diagram of Chat Module

4.3 User Interface Design

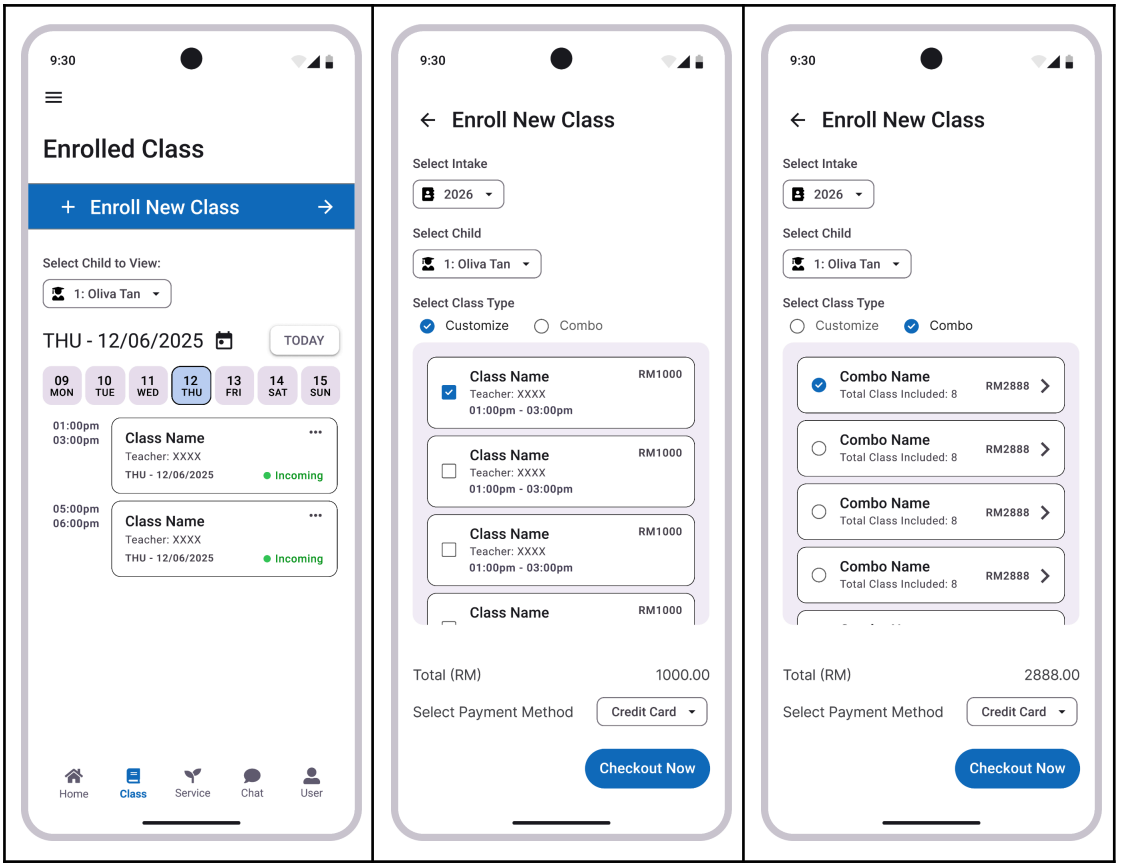
4.3.1 Parent / Student / Tutor Mobile View

Home

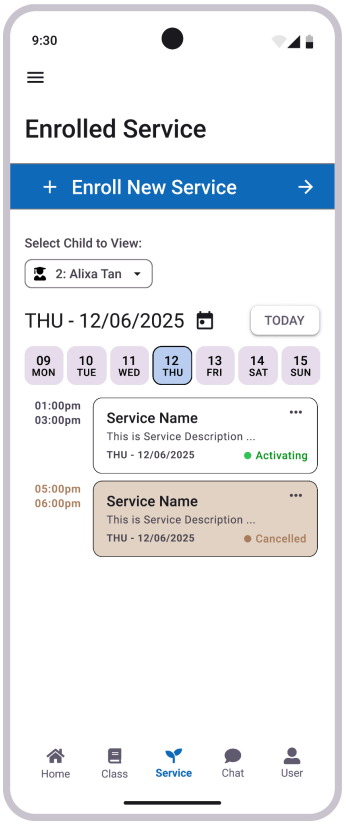
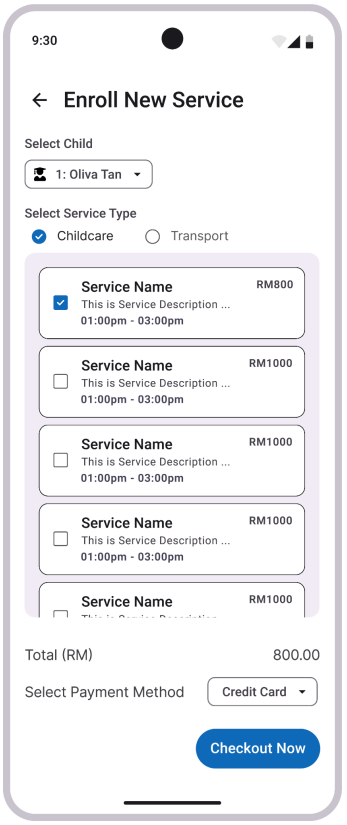
Home view for parent	Home view for student
	

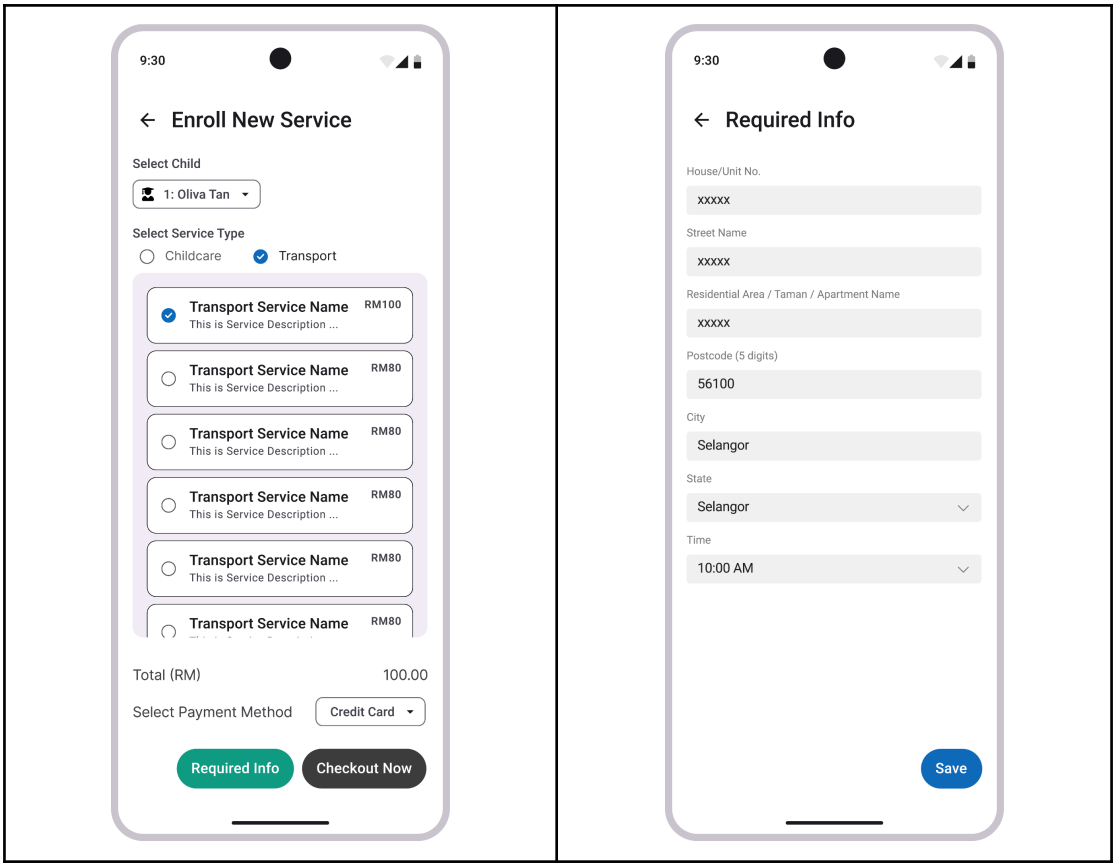
Class Enrollment UI

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.	Form for enroll new class with direct **select a class combo (package)** to enroll.

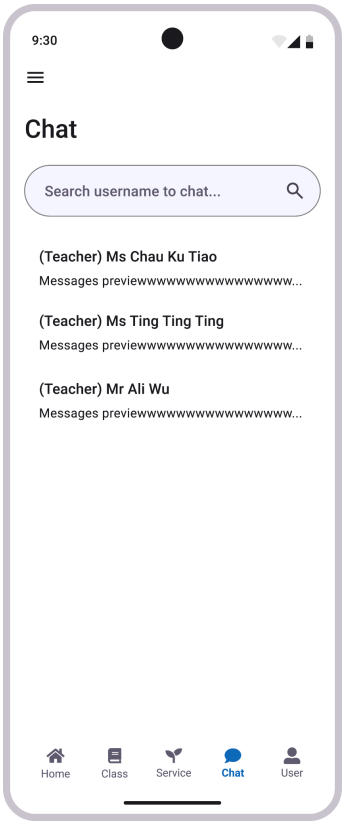
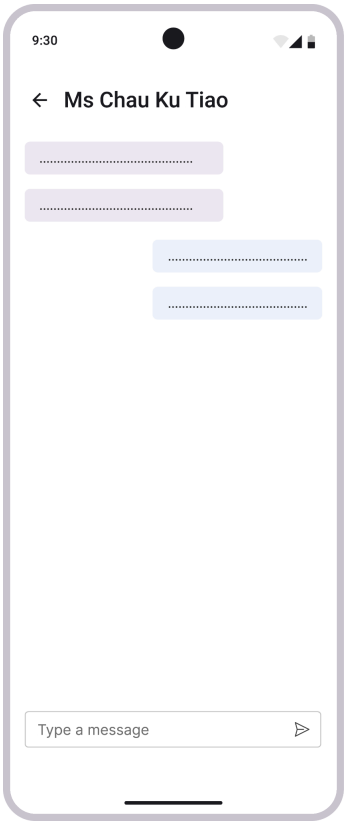

9:30
Enrolled Class
Enroll New Class
Select Child to View:
1: Oliva Tan
THU - 12/06/2025
TODAY
09 MON
10 TUE
11 WED
12 THU
13 FRI
14 SAT
15 SUN
01:00pm
03:00pm
Class Name
Teacher: XXXX
THU - 12/06/2025
Incoming
05:00pm
06:00pm
Class Name
Teacher: XXXX
THU - 12/06/2025
Incoming
Home
Class
Service
Chat
User
9:30
Enroll New Class
Select Intake
2026
Select Child
1: Oliva Tan
Select Class Type
Customize
Combo
Class Name
Teacher: XXXX
01:00pm - 03:00pm
RM1000
Class Name
Teacher: XXXX
01:00pm - 03:00pm
RM1000
Class Name
Teacher: XXXX
01:00pm - 03:00pm
RM1000
Class Name
RM1000
Total (RM)
1000.00
Select Payment Method
Credit Card
Checkout Now
9:30
Enroll New Class
Select Intake
2026
Select Child
1: Oliva Tan
Select Class Type
Customize
Combo
Combo Name
Total Class Included: 8
RM2888
Combo Name
Total Class Included: 8
RM2888
Combo Name
Total Class Included: 8
RM2888
Combo Name
Total Class Included: 8
RM2888
Total (RM)
2888.00
Select Payment Method
Credit Card
Checkout Now

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..


9:30
Enroll New Service
Select Child
1: Oliva Tan
Select Service Type
Childcare
Transport
Transport Service Name
RM100
This is Service Description ...
Transport Service Name
RM80
This is Service Description ...
Transport Service Name
RM80
This is Service Description ...
Transport Service Name
RM80
This is Service Description ...
Transport Service Name
RM80
This is Service Description ...
Transport Service Name
RM80
Total (RM)
100.00
Select Payment Method
Credit Card
Required Info
Checkout Now
9:30
Required Info
House/Unit No.
xxxxx
Street Name
xxxxx
Residential Area / Taman / Apartment Name
xxxxx
Postcode (5 digits)
56100
City
Selangor
State
Selangor
Time
10:00 AM
Save

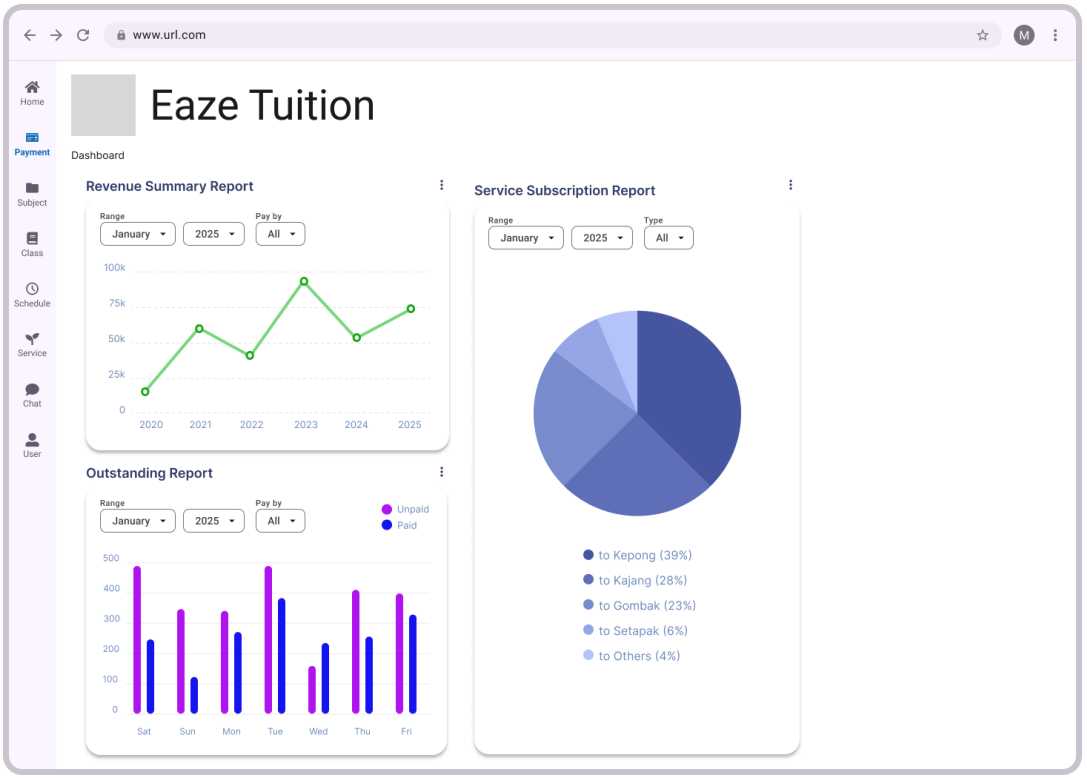
Chat UI

Chat list with previous contacted users.	Messaging view with another user.
	

4.3.2 Admin / Staff Web Browser View

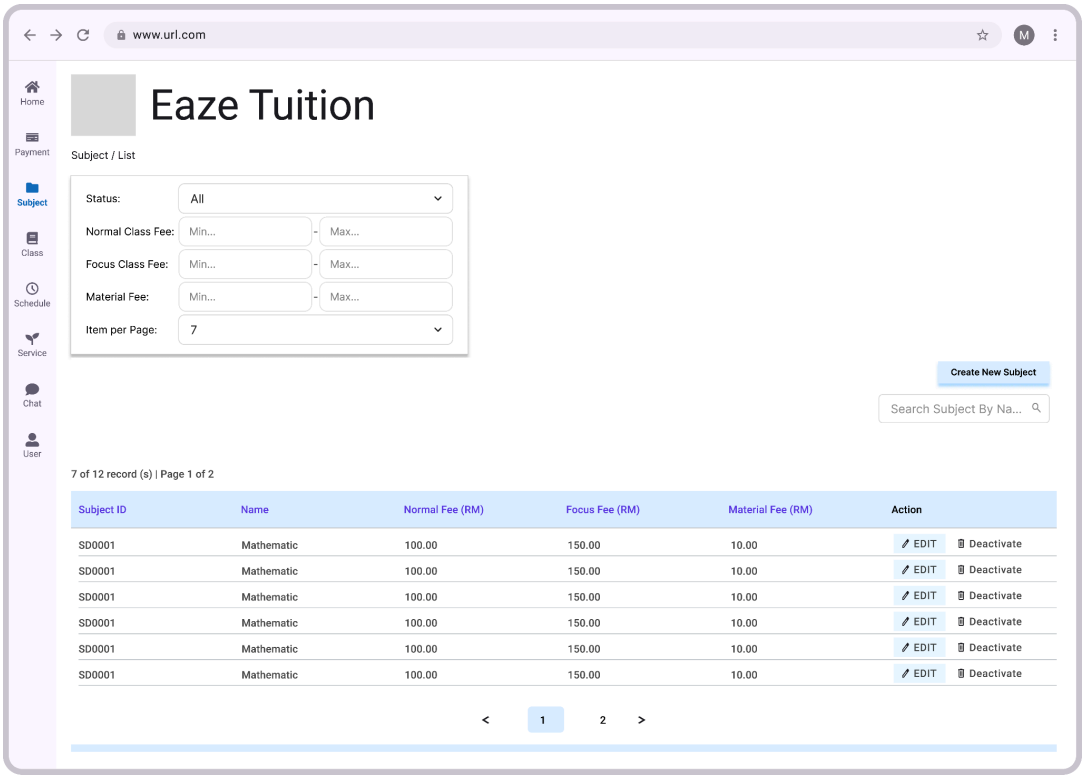
Admin Dashboard UI

Dashboard of admin or staff

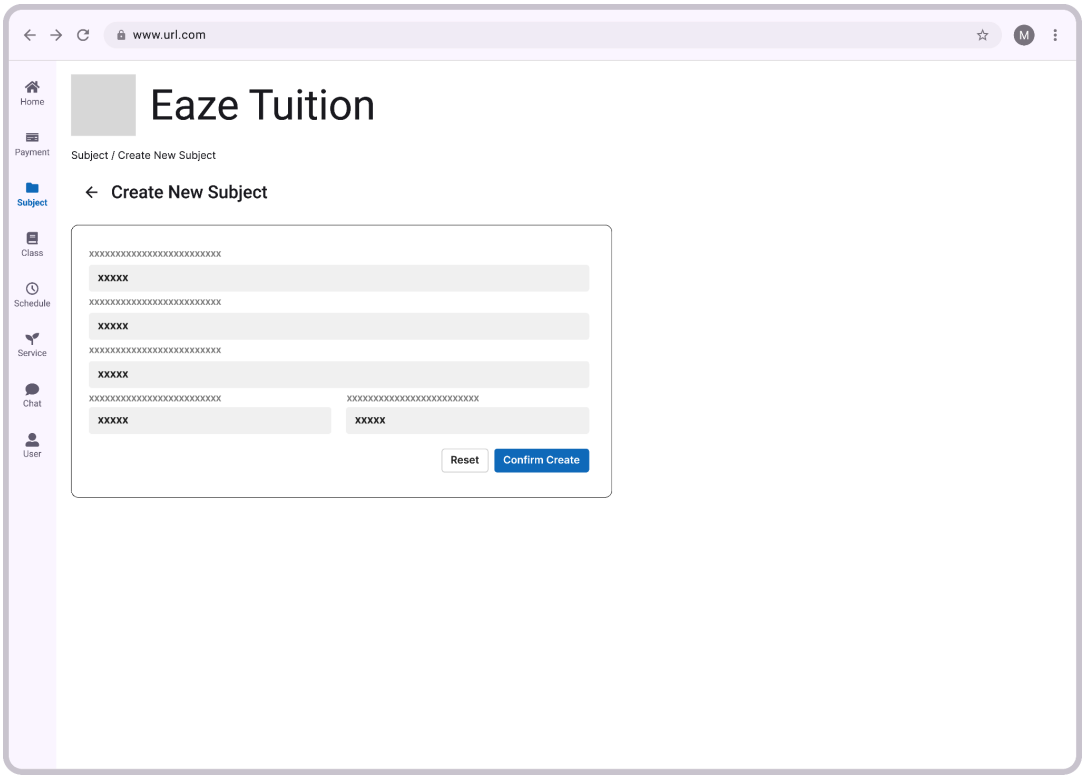


Admin Normal Management UI

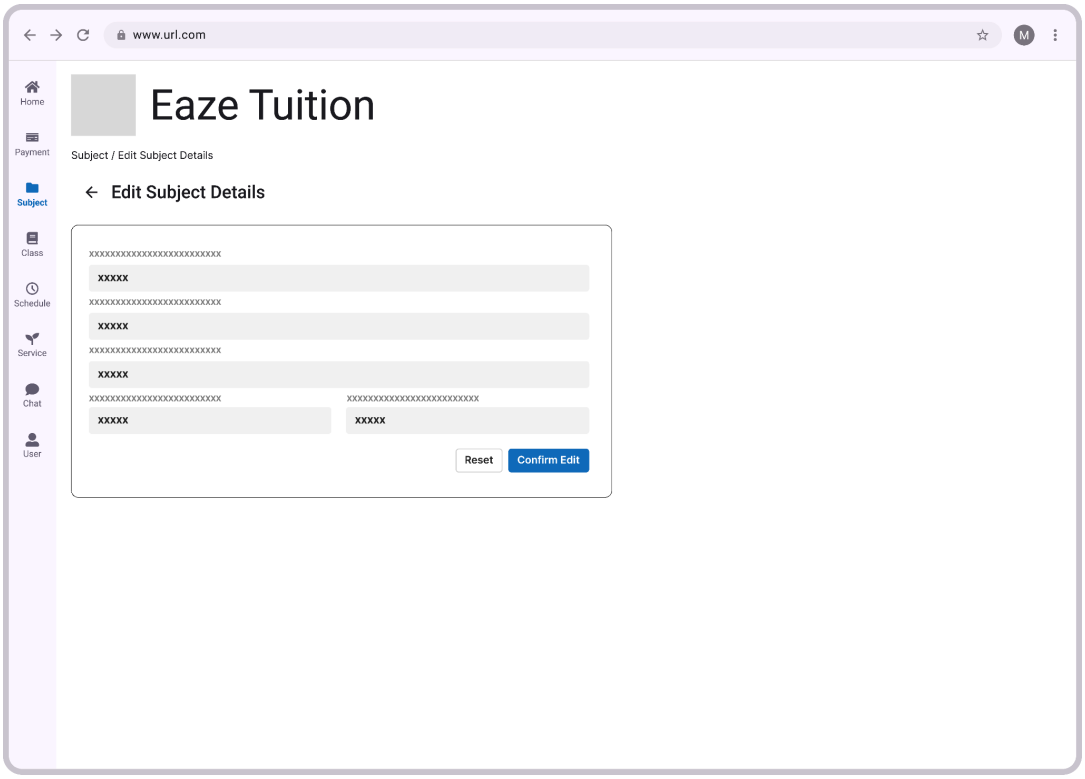
Subject, class, class package schedule, service and payment management list layout.



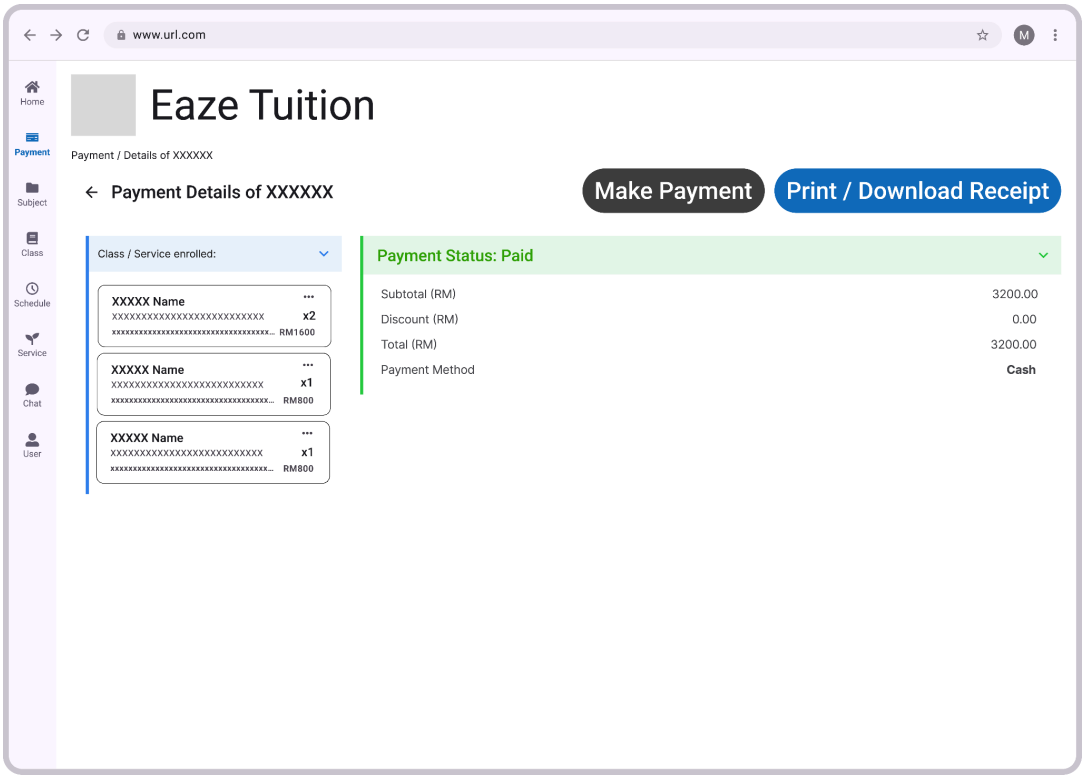
Create subject, class and service form layout.



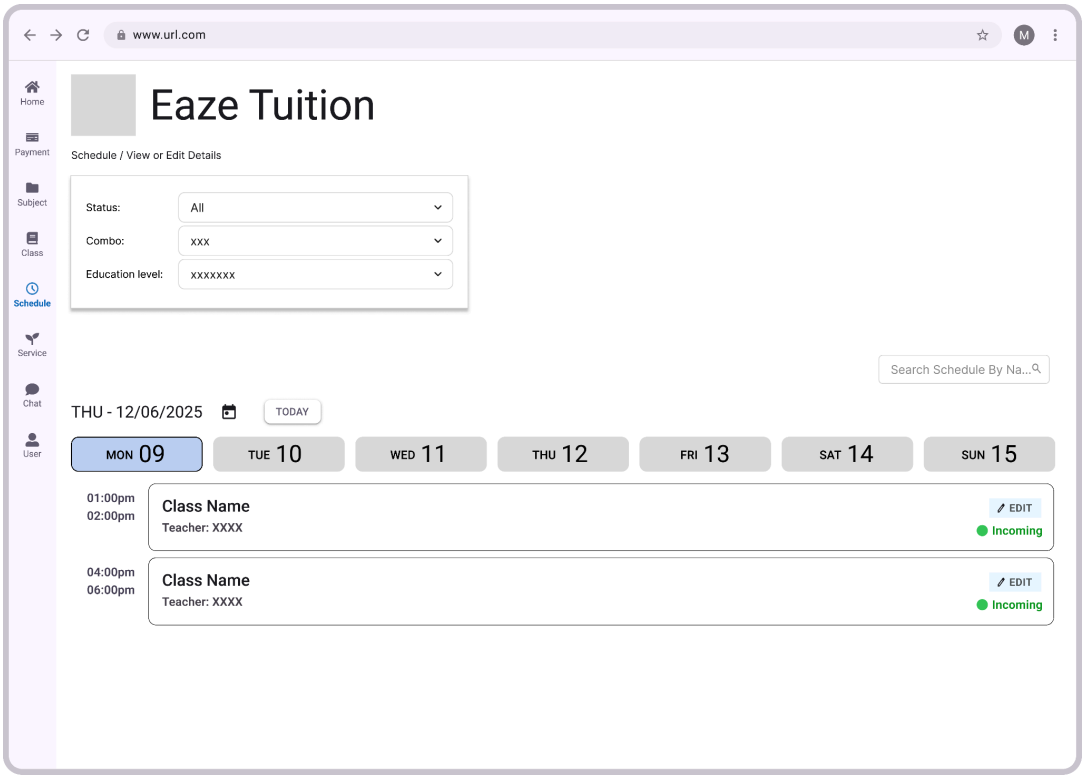
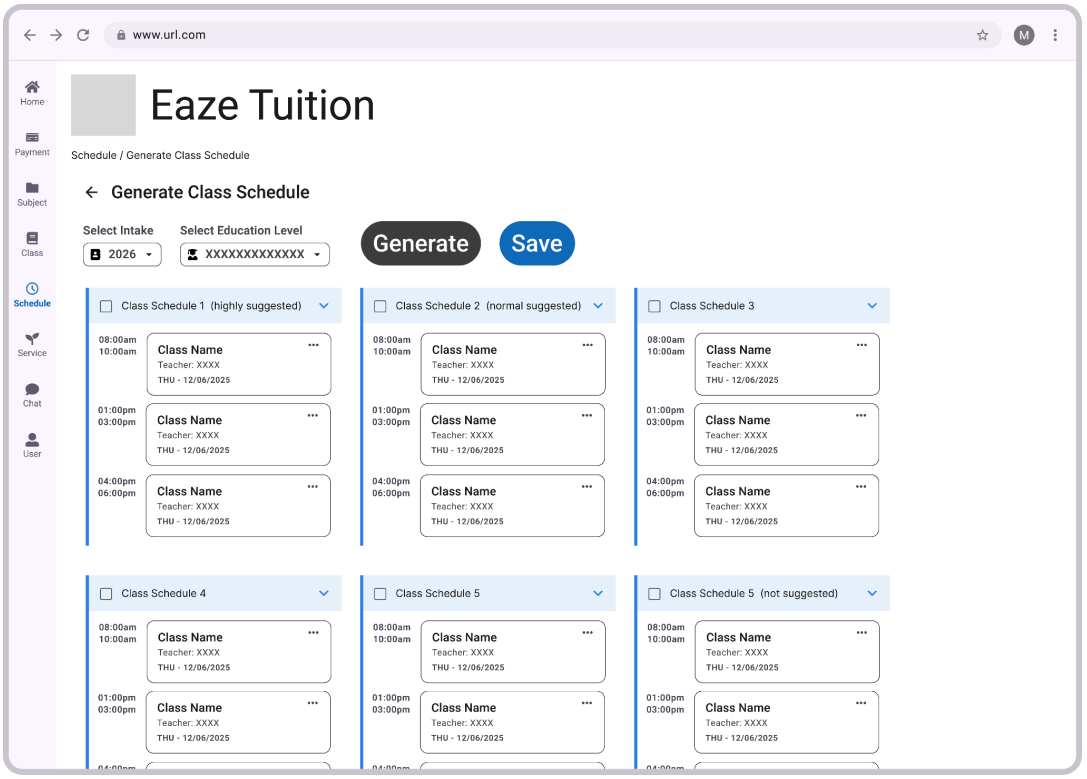
Update subject, class and service form layout.



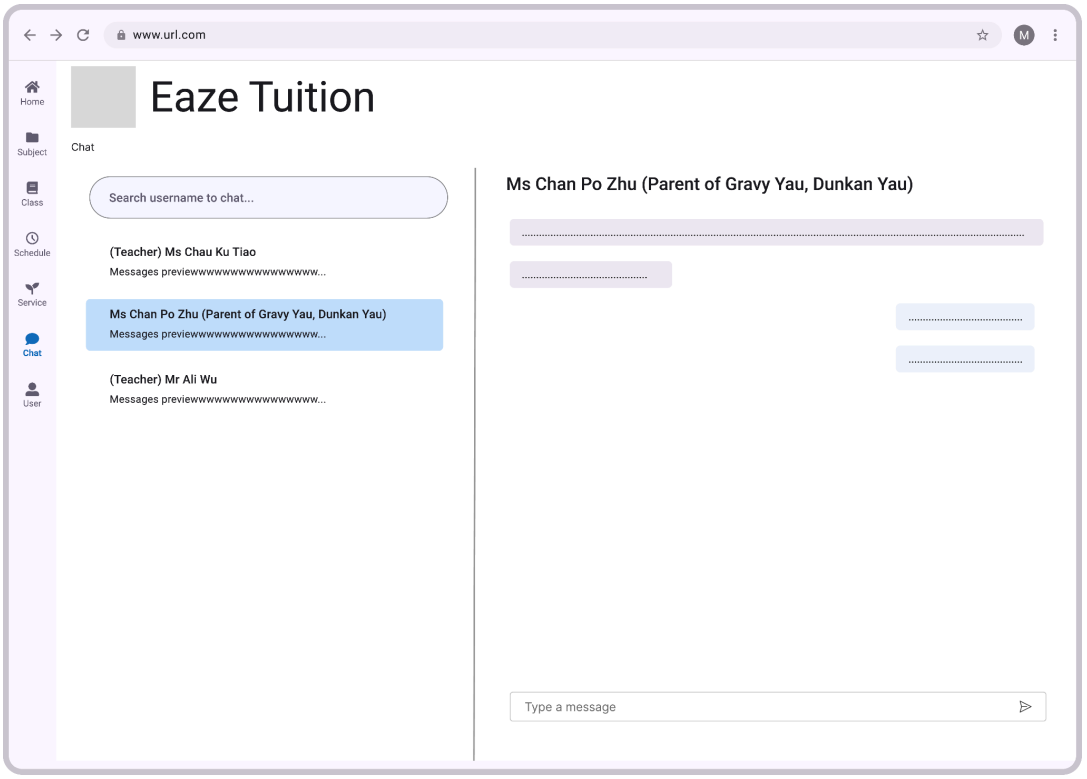
Payment detail layout



Class Schedule Management (Generate & Update) UI

Class Combo (Package) Schedule Detail List

Generate class schedule layout.


Chat UI

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


4.4 Data Design

4.4.1 Class Diagram

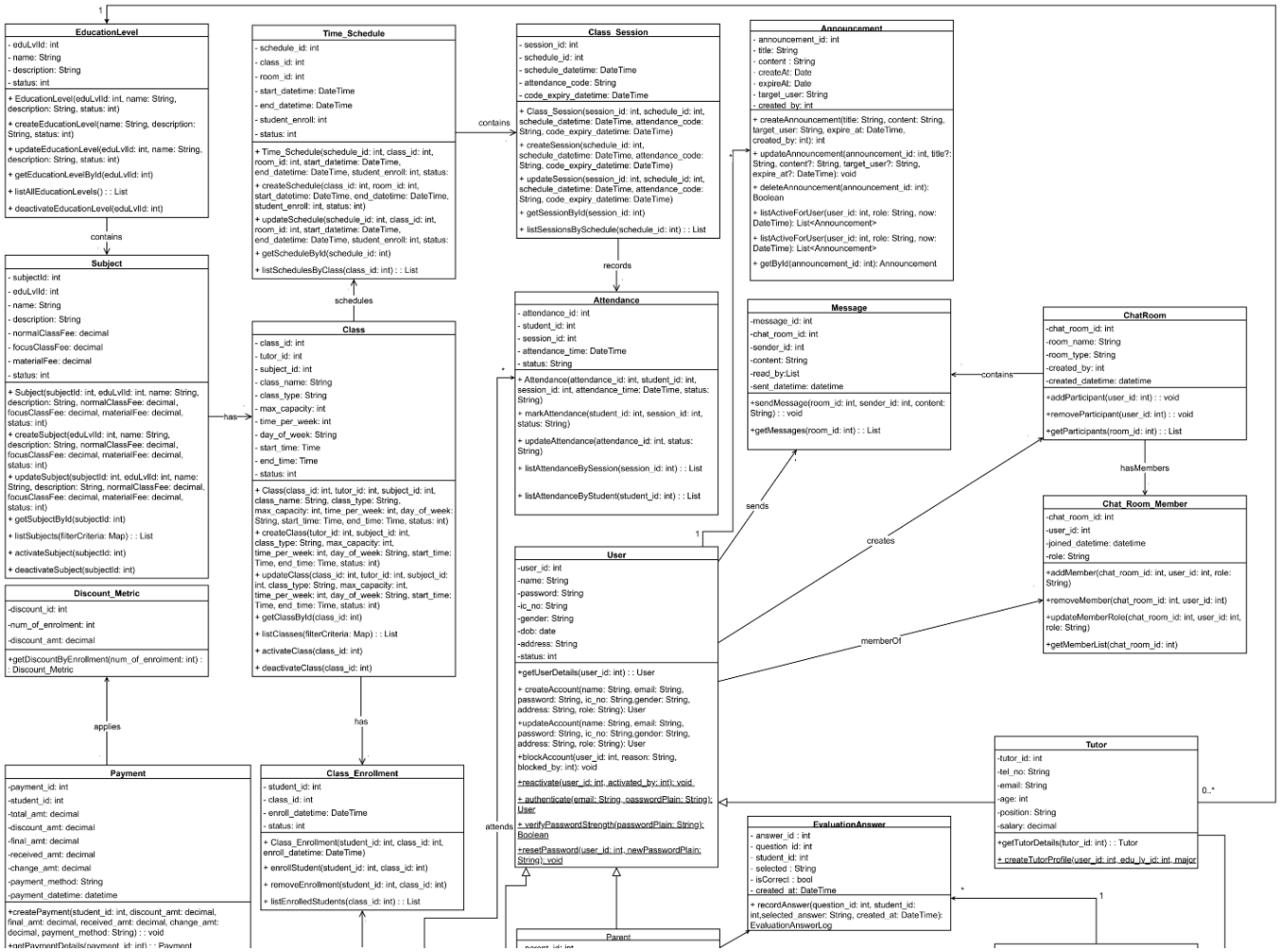


Figure 4.5.1.1 Class Diagram of EazeTuition Page 1

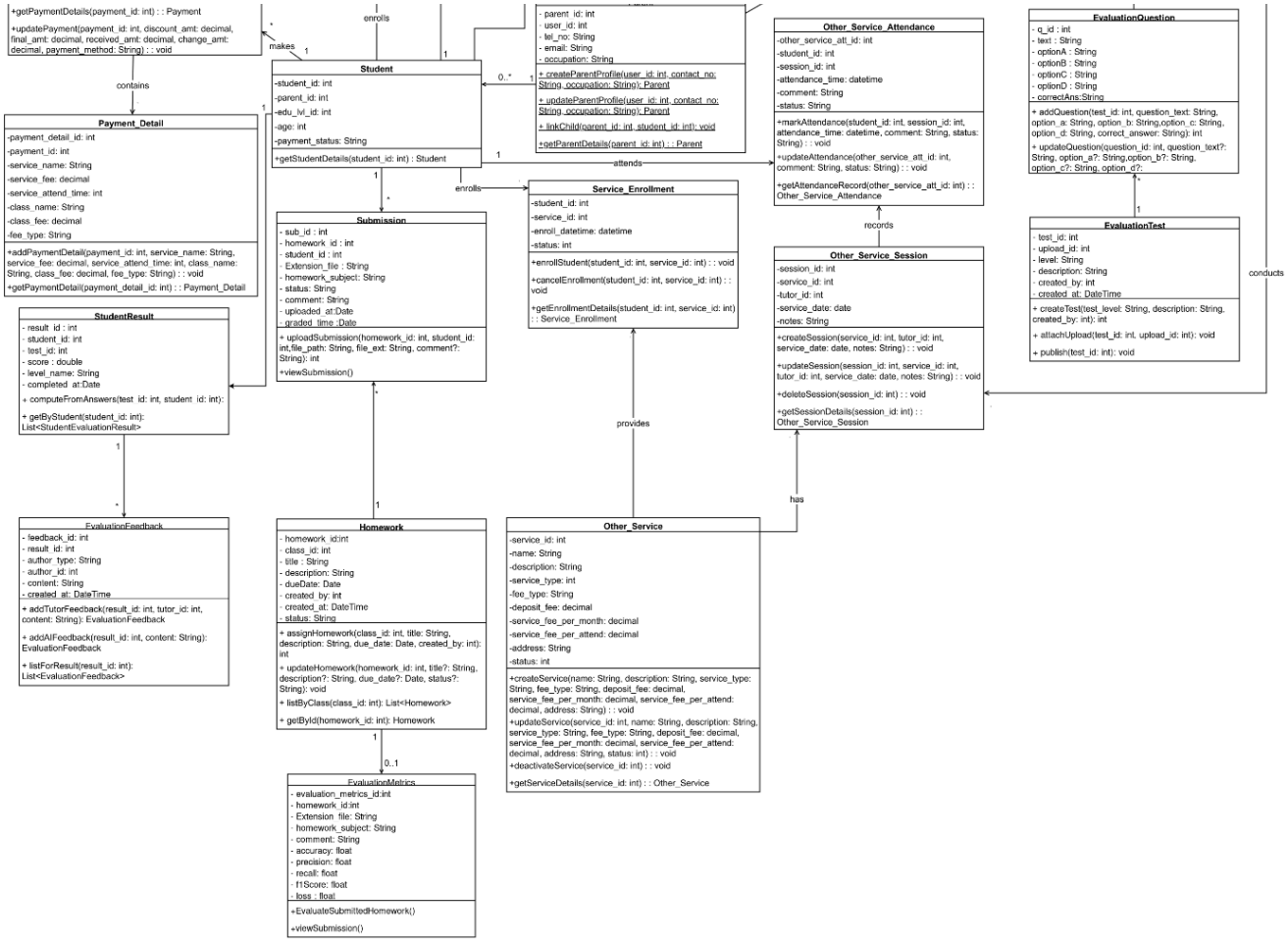


Figure 4.5.1.2 Class Diagram of EazeTuition Page 2

4.4.2 Entity Relationship Diagram (ERD)

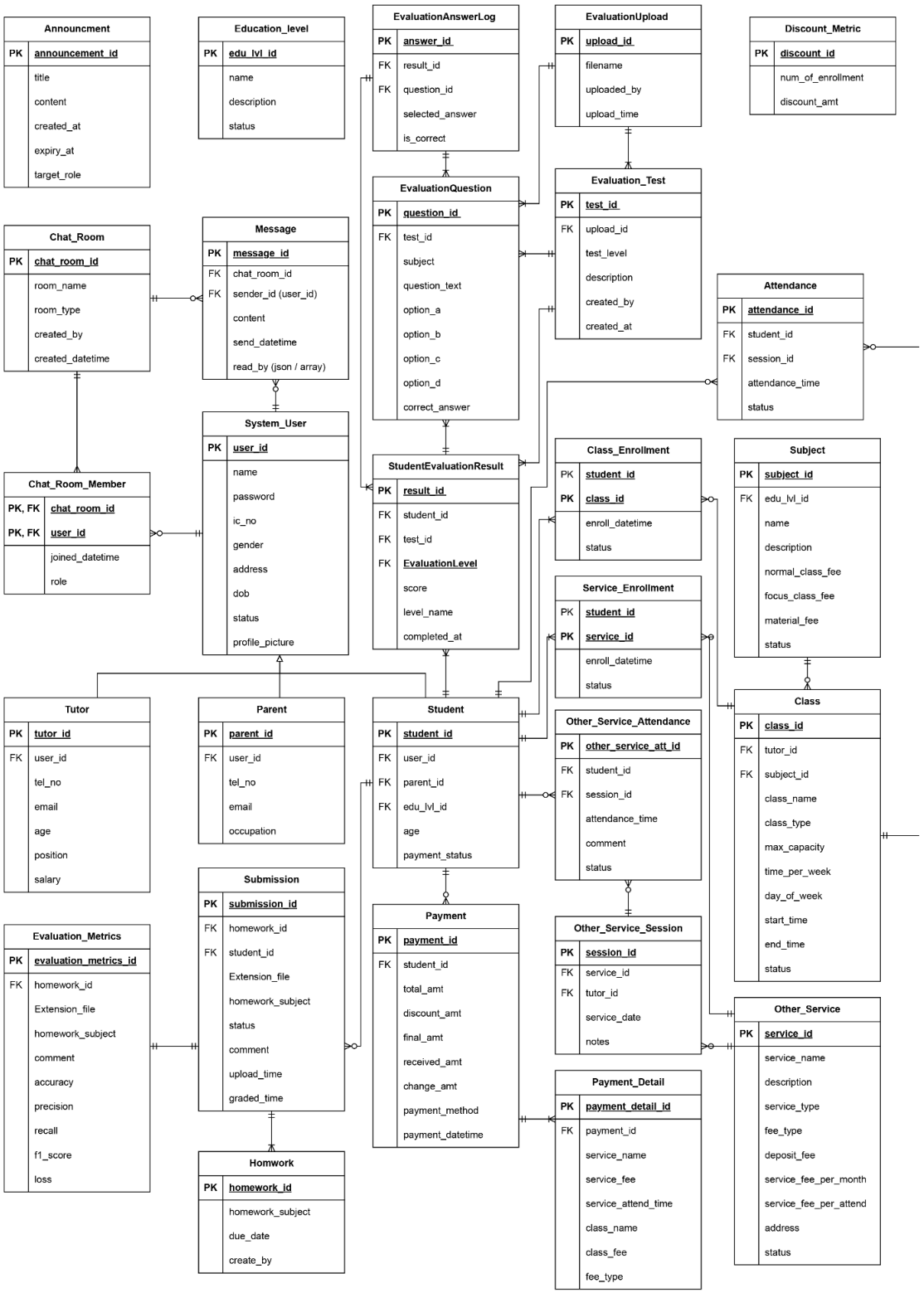


Figure 4.5.2.1 Entity Relationship Diagram Page 1

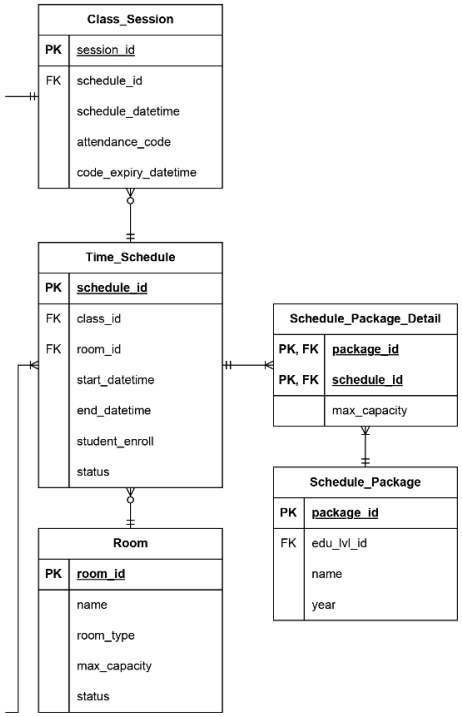


Figure 4.5.2.2 Entity Relationship Diagram Page 2

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 → EducationLevel(edu_lvl_id), Not Null
name	VARCHAR	100	Name of the subject (e.g., Mathematics)	Not Null
description	TEXT	-	Detailed description of the 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	Fee for materials (if applicable)	Default 0.00
status	VARCHAR	11	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_name	VARCHAR	100	Name of the class (e.g.,Primary Year 3 - Mathematic)	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
day_of_week	VARCHAR	20	Day of the week when the class is held	Nullable
start_time	TIME	-	Class start time	Nullable
end_time	TIME	-	Class end time	Nullable
status	VARCHAR	11	Class status ('0'=Inactive, '1'=Active)	Default '1'

Table 4.4.3.2.1 "Class" Table Data Dictionary

Class Enrollment Table

Field Name	Data Type	Size	Description	Constraints / Notes
student_id	INT	11	Student enrolled in the class	PK, FK → Student(student_id)
class_id	INT	11	Class enrolled	PK, FK → Class(class_id)
enroll_datetime	DATETIME	-	Date and time of enrollment	Not Null
status	VARCHAR	11	Enrollment status ('0'=Inactive, '1'=Active)	Default '1'

Table 4.4.3.2.2 "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_id	INT	11	Class linked to the schedule	FK → Class(class_id)
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
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 4.4.3.2.3 "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 4.4.3.2.4 "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 4.4.3.2.5 "Room" 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

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
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
service_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
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
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

Title:	Revenue Summary Report
Filter:	• Filter by Month • Filter by Year • Filter by Payment Method (Stripe / Cash / Touch'n Go)
Report Content:	**Graphs / Visualization** Line Chart → Revenue trend over selected period **Layout** 

Table 4.5.1 Detail of Revenue Summary Report

4.5.2 Outstanding Payment Report

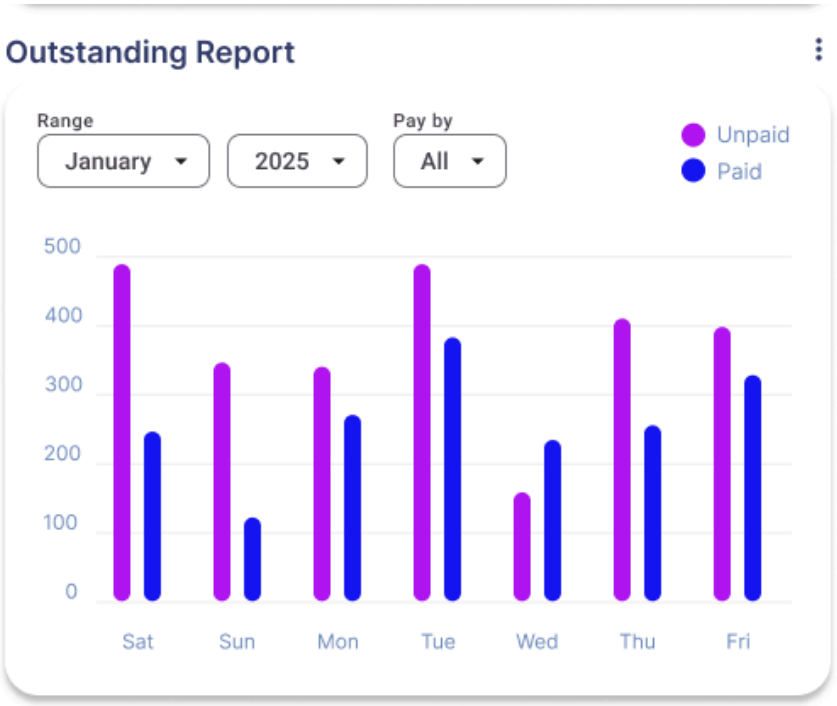
Heading:	Outstanding Payment Report
Filter:	• Filter by Month • Filter by Year • Filter by Payment Status (Unpaid / Overdue)
Report Content:	**Graphs / Visualization** Bar Chart → Outstanding student quantity of both payment status trend over selected months or year **Layout** 

Table 4.5.1 Detail of Revenue Summary Report

4.5.3 Service Subscription Report

Heading:	Service Subscription Report
Filter:	• Filter by Month • Filter by Year • Filter by Service Type (Transport / Childcare)
Report Content:	**Graphs / Visualization** Pie Chart → Service subscription percentage over a period **Layout**

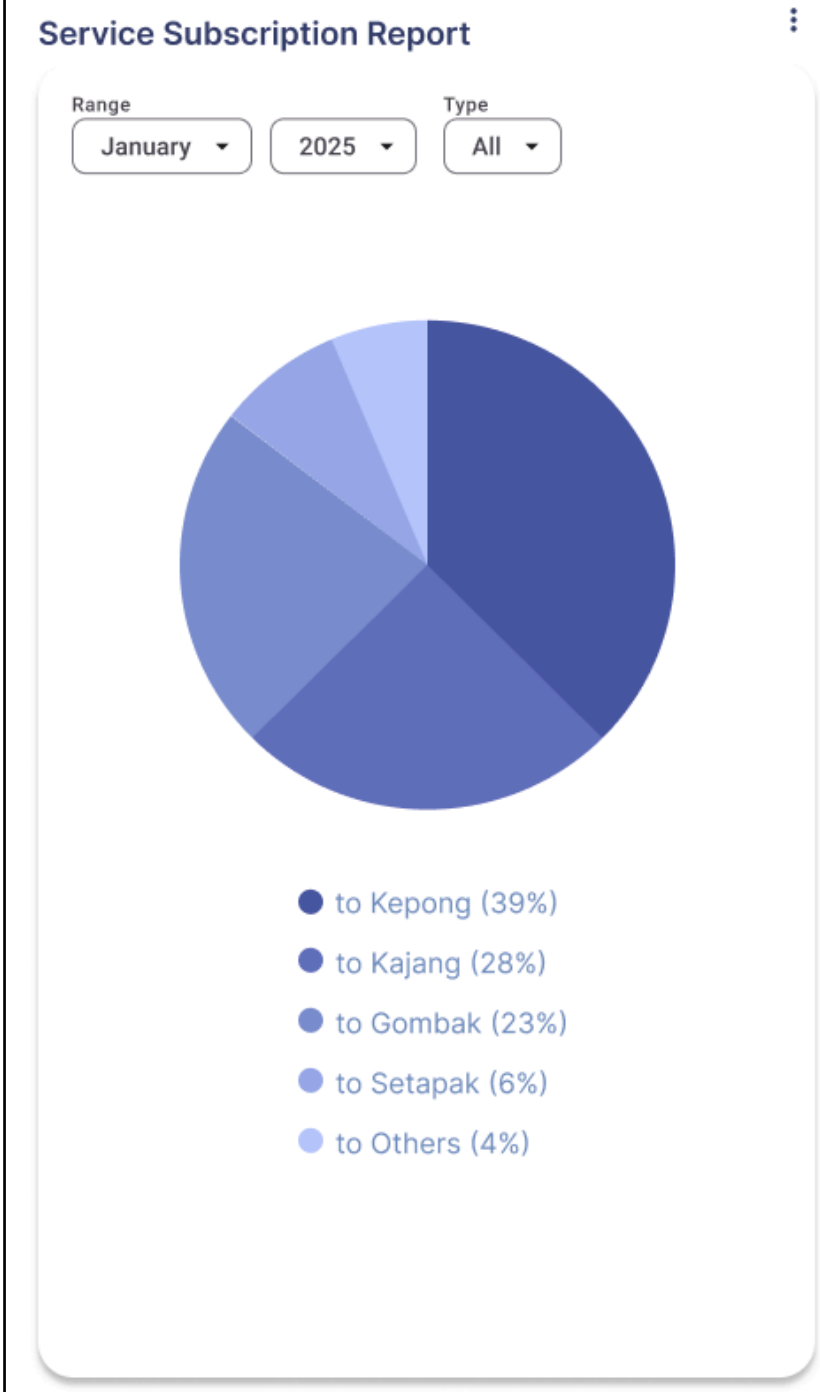


Table 4.5.3 Detail of Service Subscription Report

4.6 Process Design

4.6.1 Subject Module

Create Subject

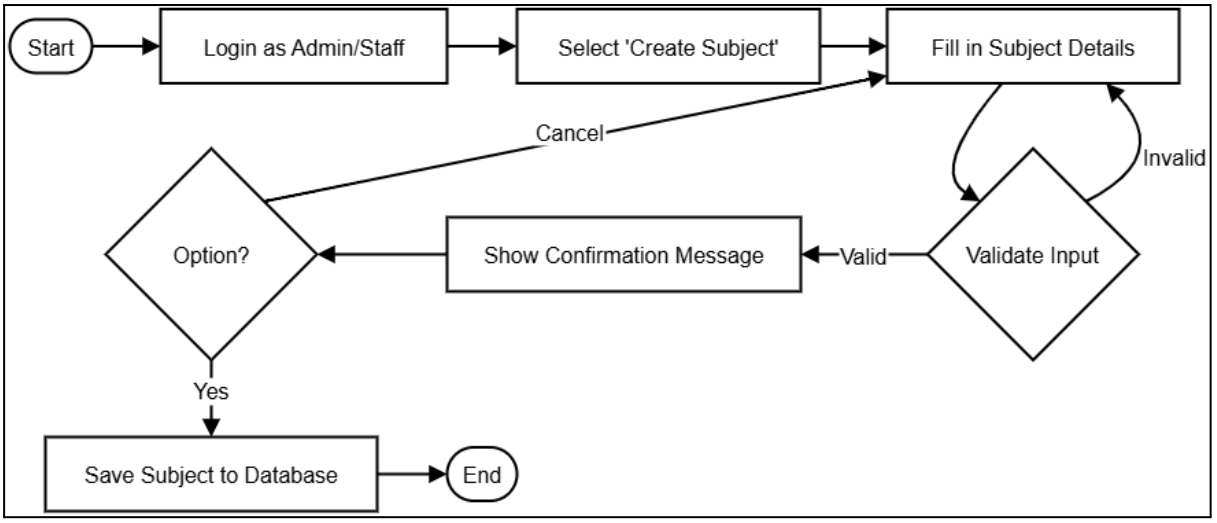


Table 4.6.1.1 Flowchart of Create Subject

View Subject List

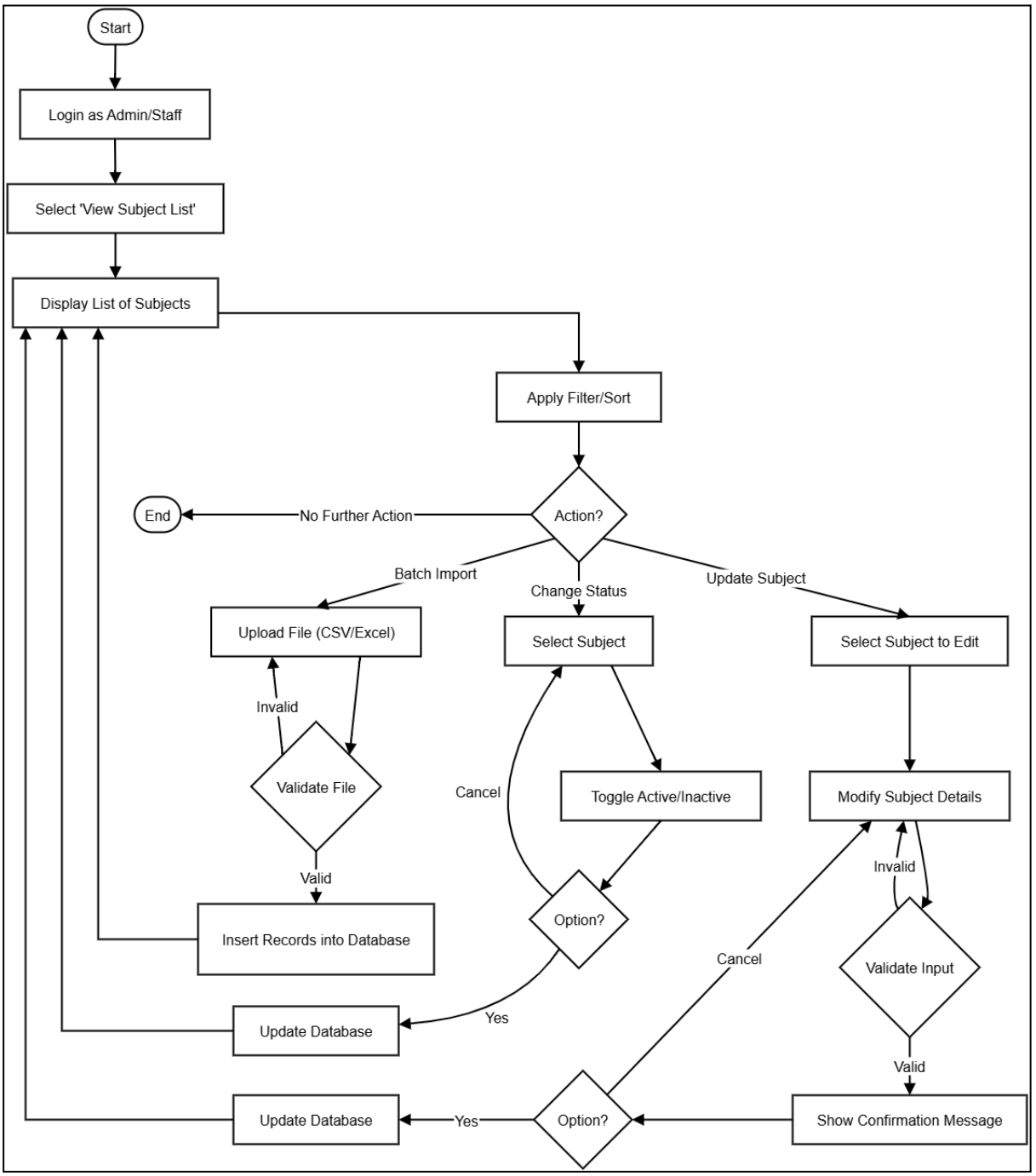


Table 4.6.1.2 Flowchart of View Subject List

4.6.2 Class Module

Class Management

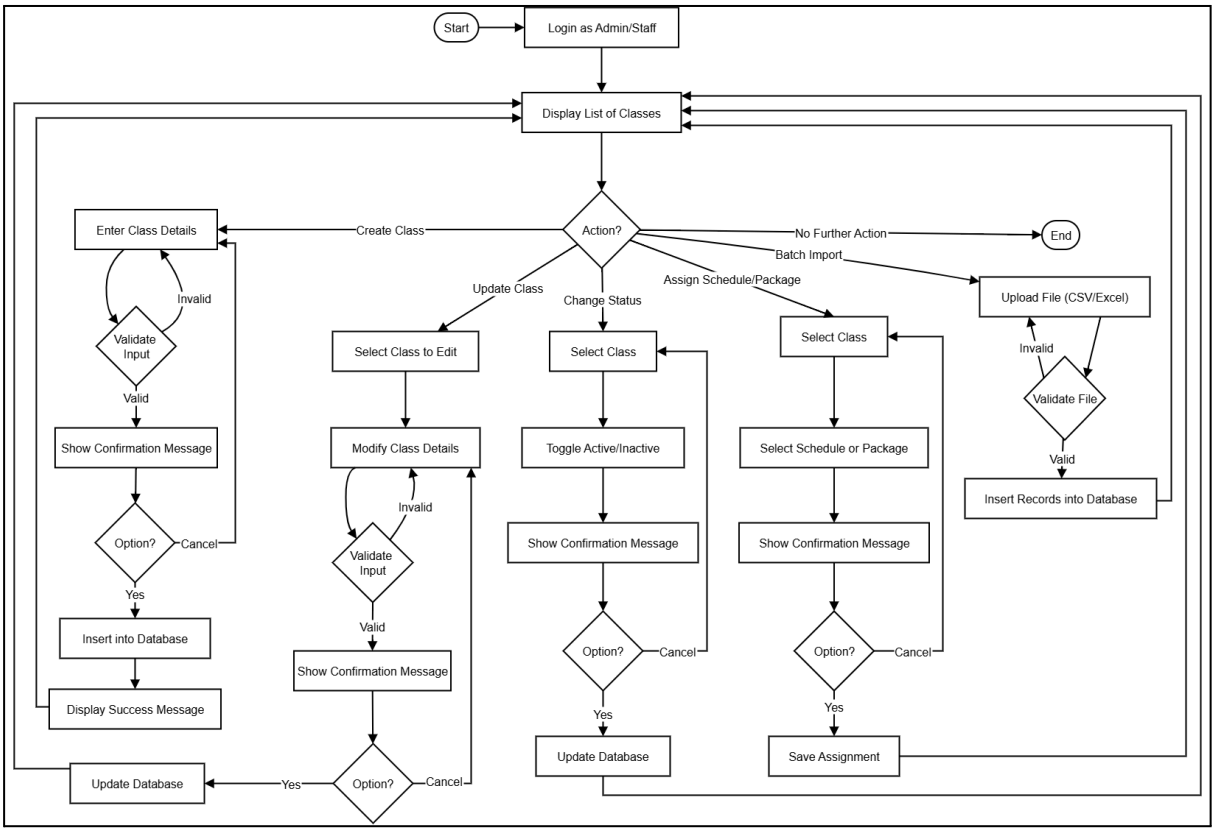


Table 4.6.2.1 Flowchart of Class Management

Generate Class Schedule

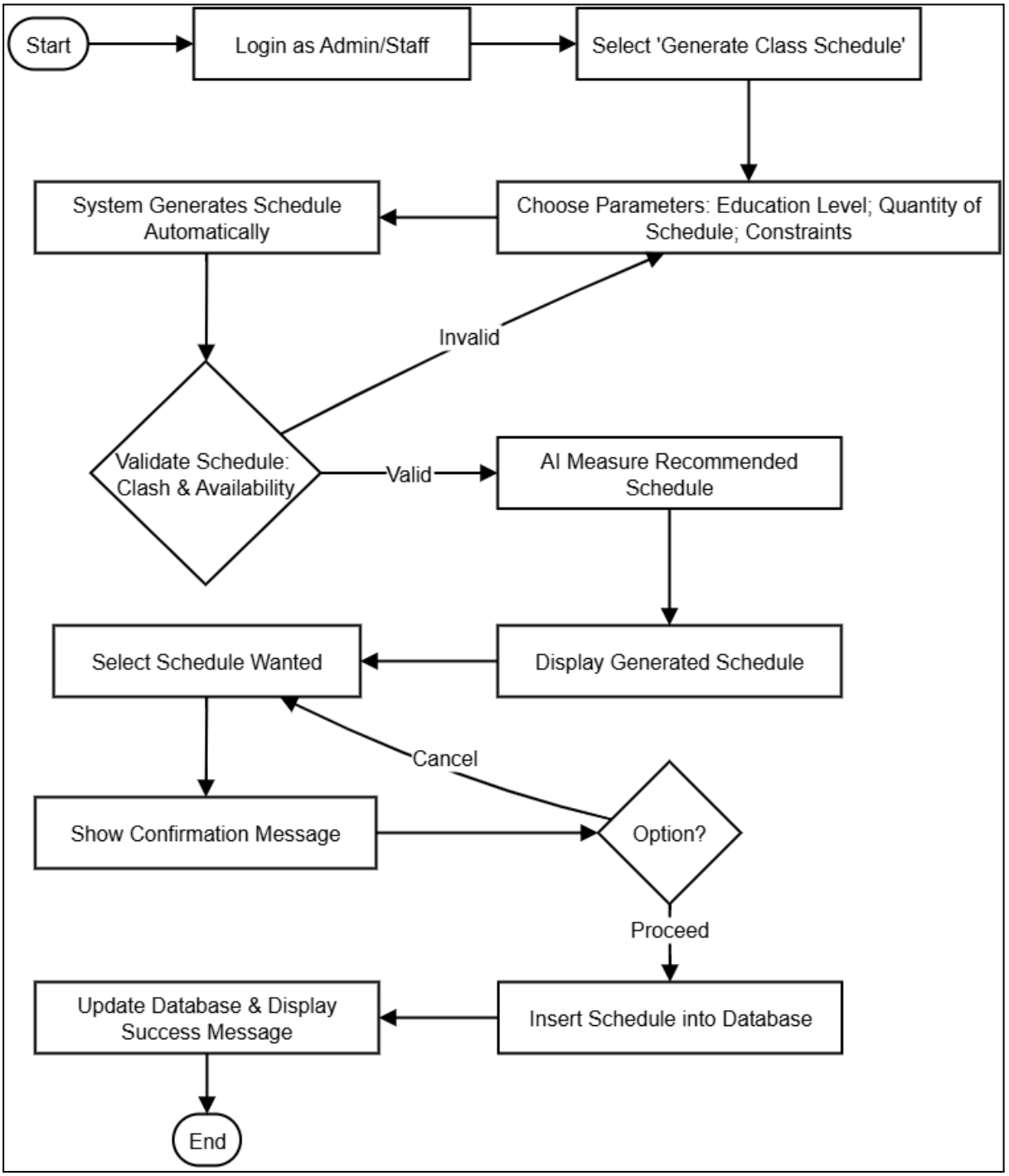


Table 4.6.2.2 Flowchart of Generate Class Schedule

Cancel Class Enrollment

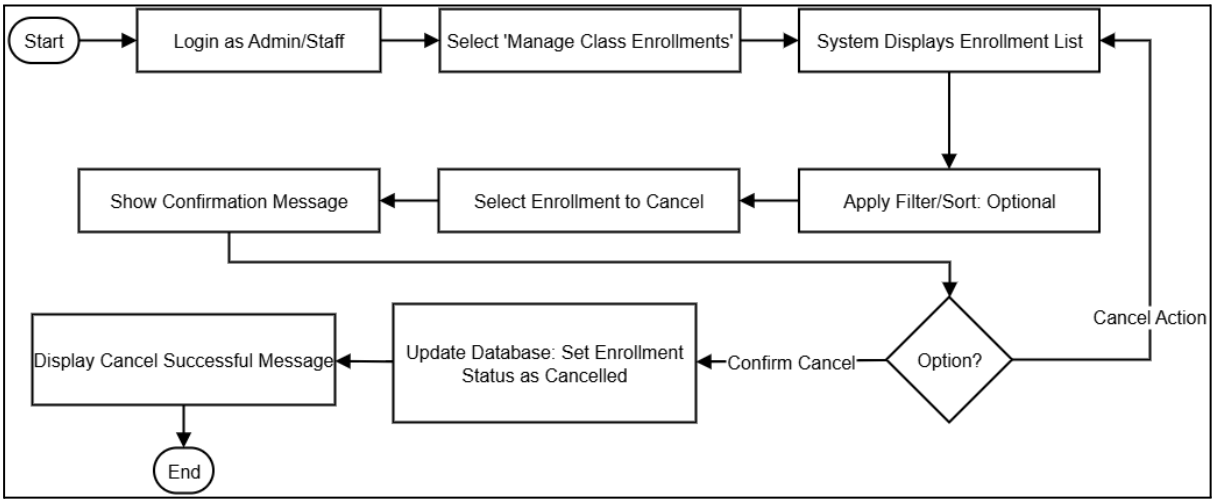


Table 4.6.2.3 Flowchart of Cancel Class Enrollment

View Available Class List

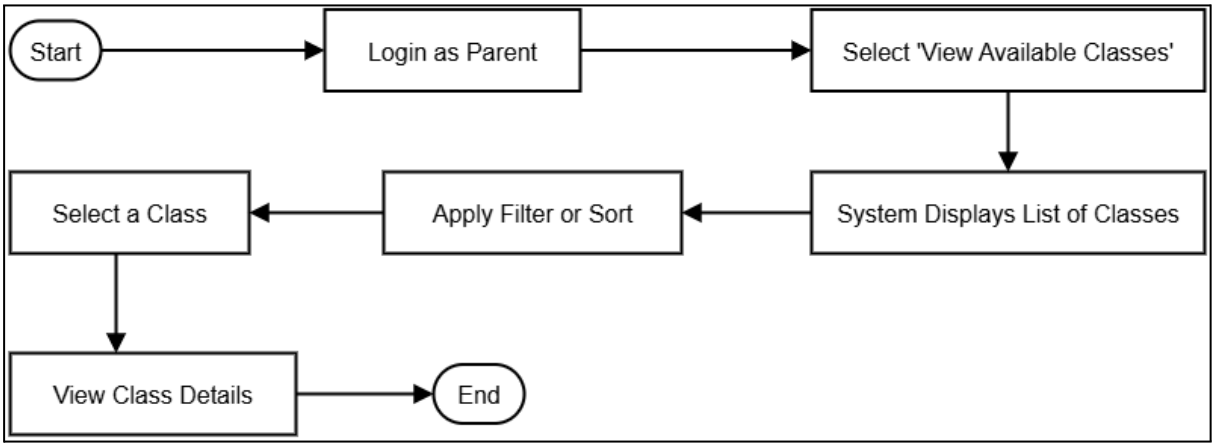


Table 4.6.2.4 Flowchart of View Available Class List

Class Enrollment

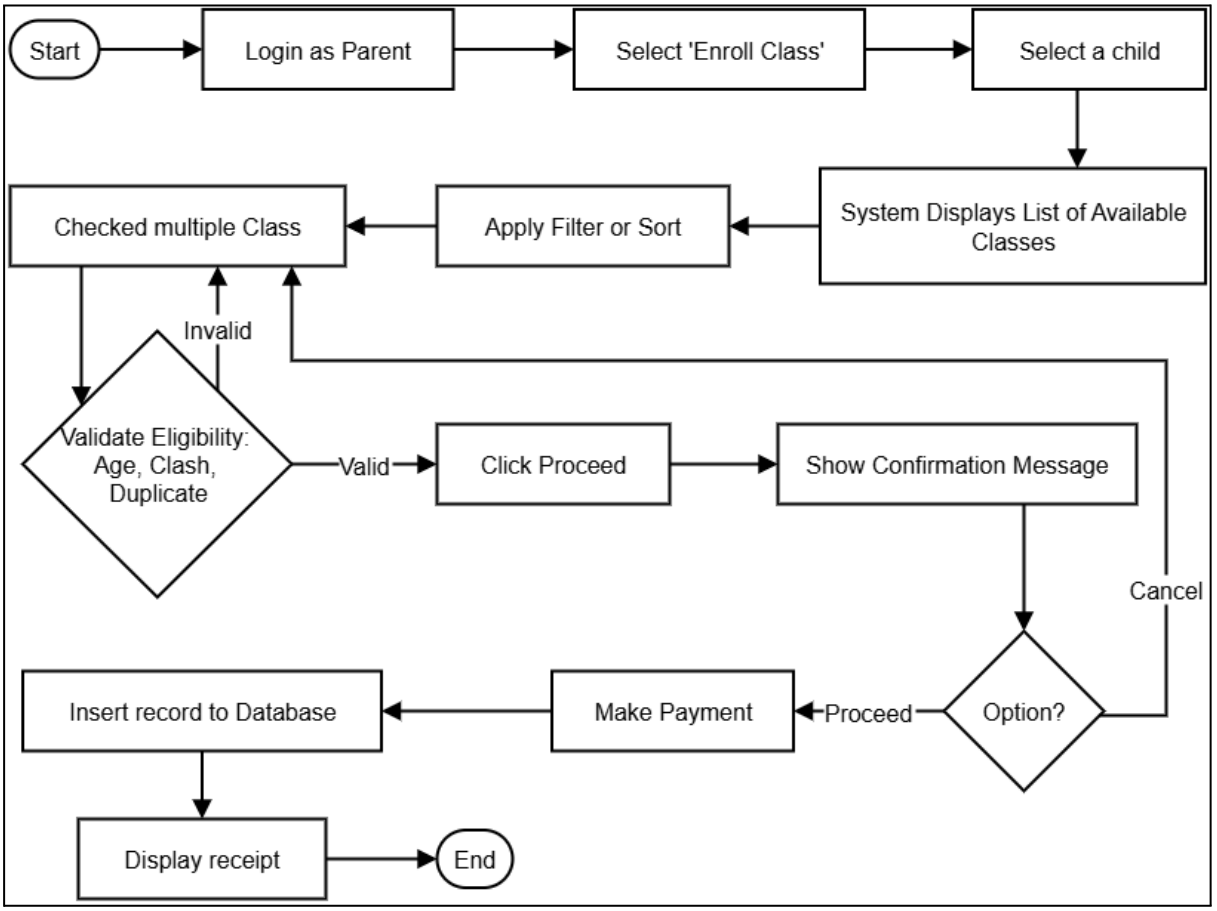


Table 4.6.2.5 Flowchart of Class Enrollment

4.6.3 Payment Module

Make Online Payment

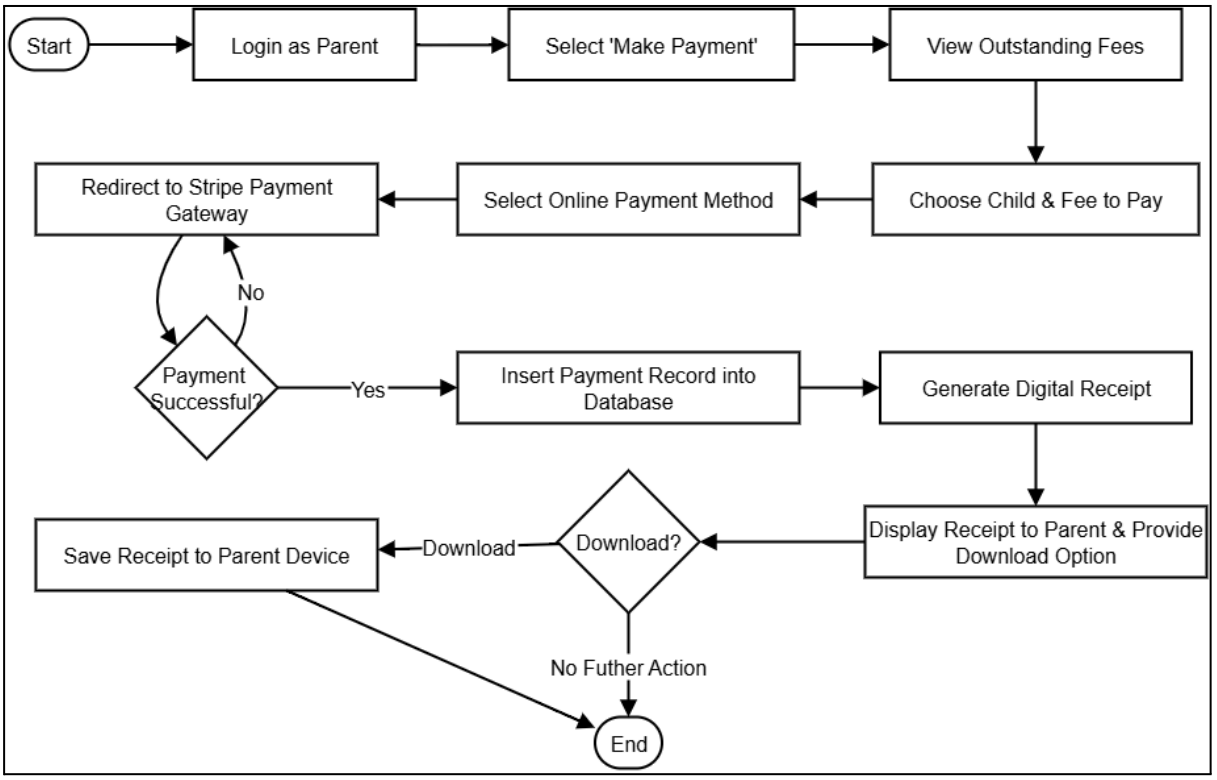


Table 4.6.3.1 Flowchart of Make Online Payment

Make Offline Payment

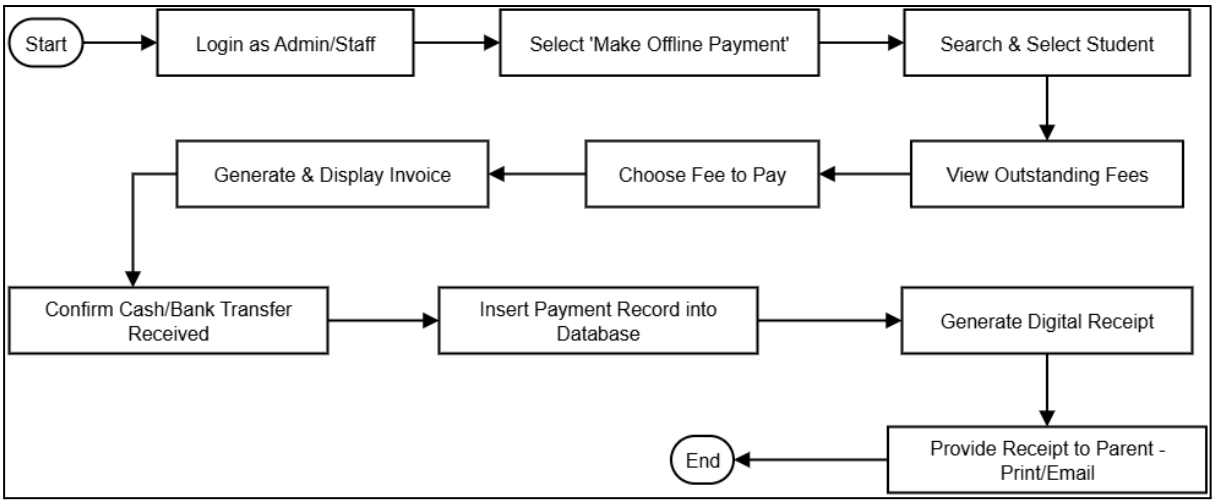


Table 4.6.3.2 Flowchart of Make Offline Payment

View Pending / History Payments

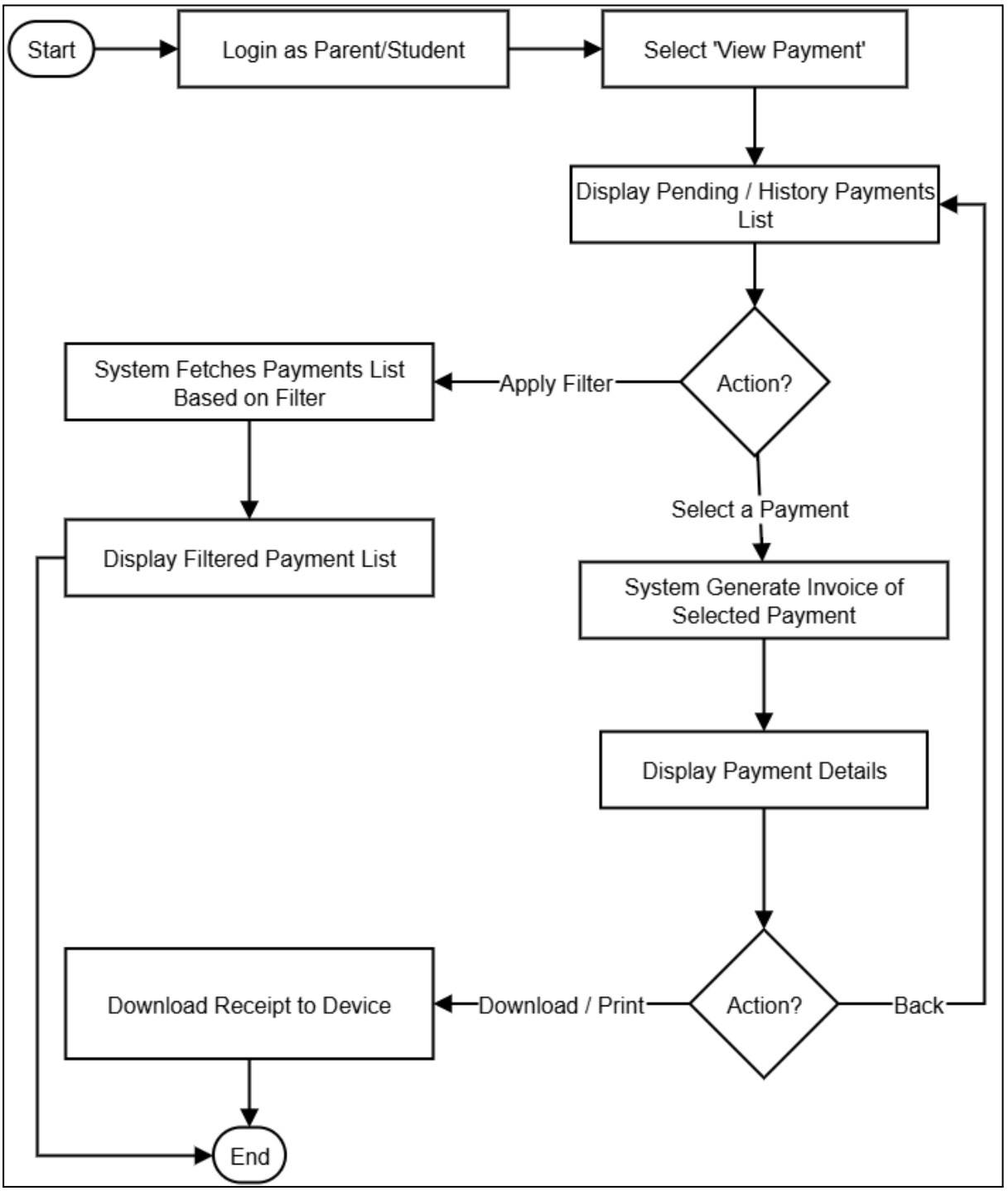


Table 4.6.3.3 Flowchart of View Pending / History Payments

4.6.4 Other Service Module

Manage Other Service

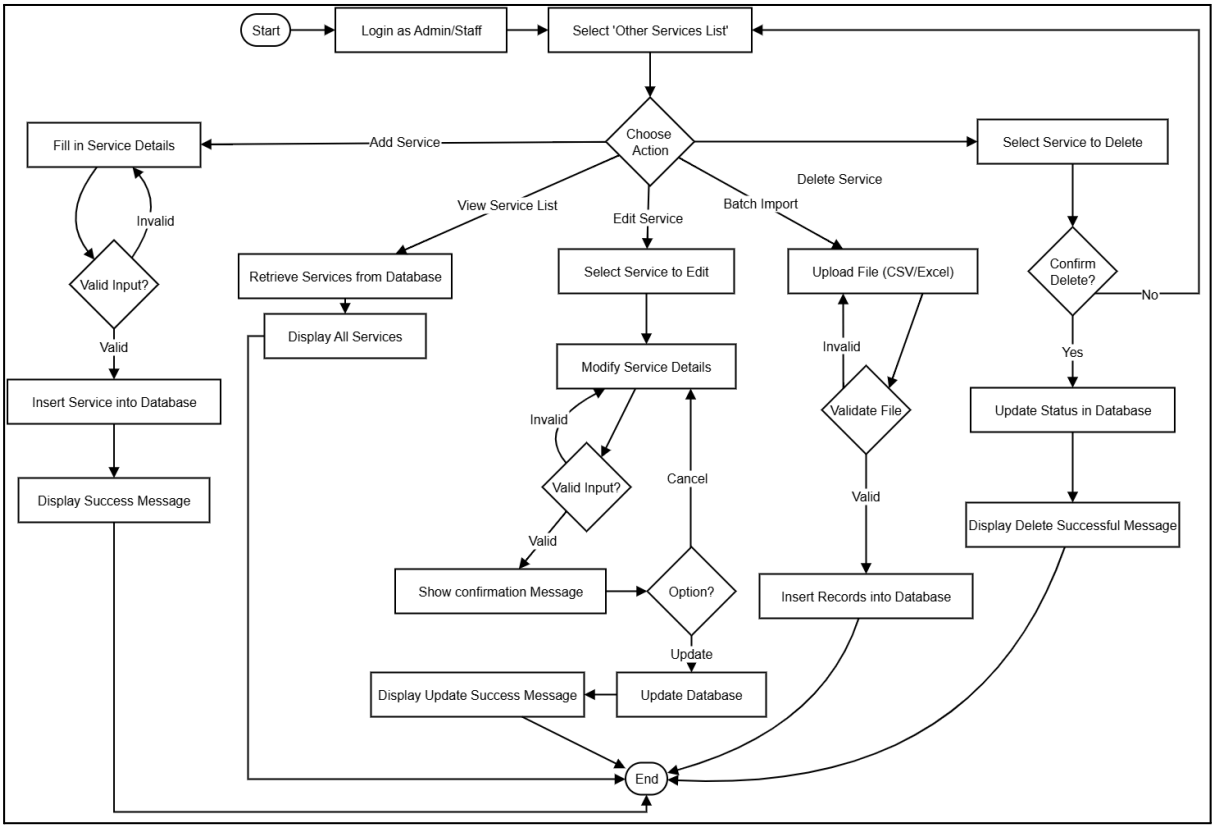


Table 4.6.4.1 Flowchart of Manage Other Service

Cancel Service Enrollment

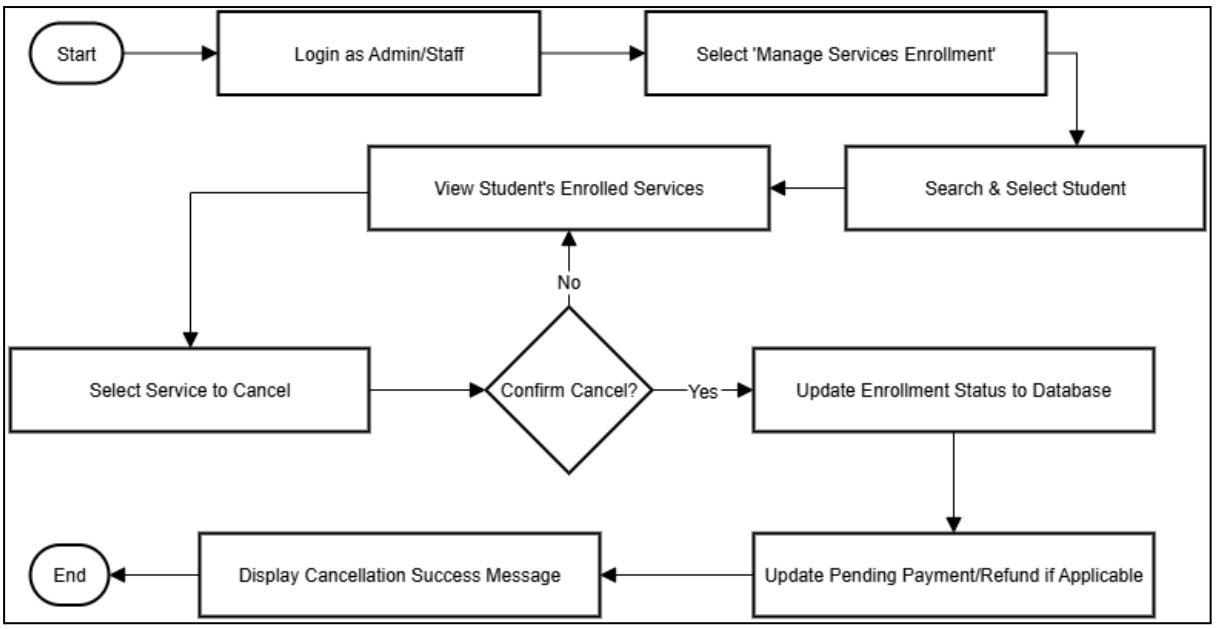


Table 4.6.4.2 Flowchart of Cancel Service Enrollment

Service Enrollment

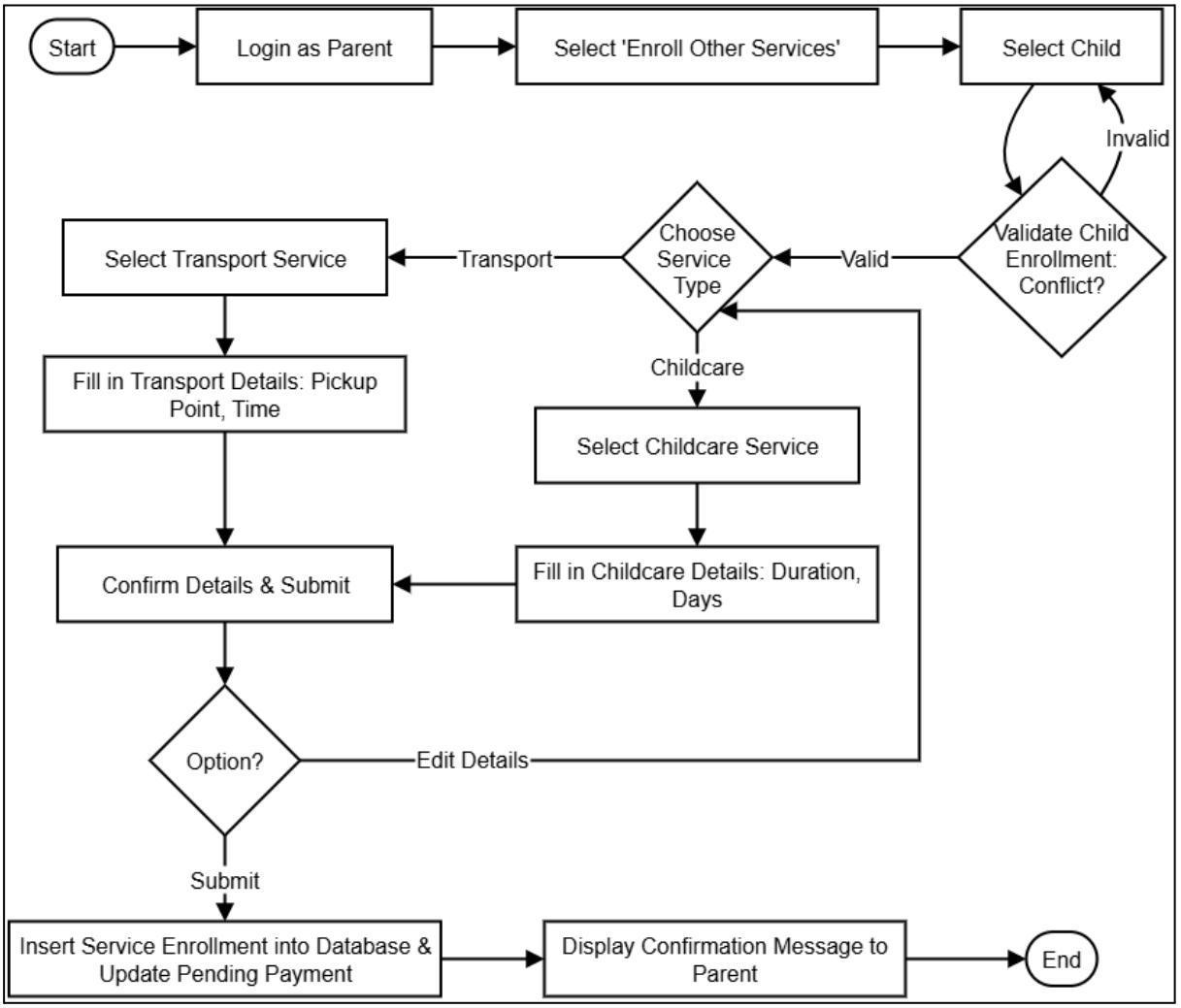


Table 4.6.4.3 Flowchart of Service Enrollment

View Enrolled Service

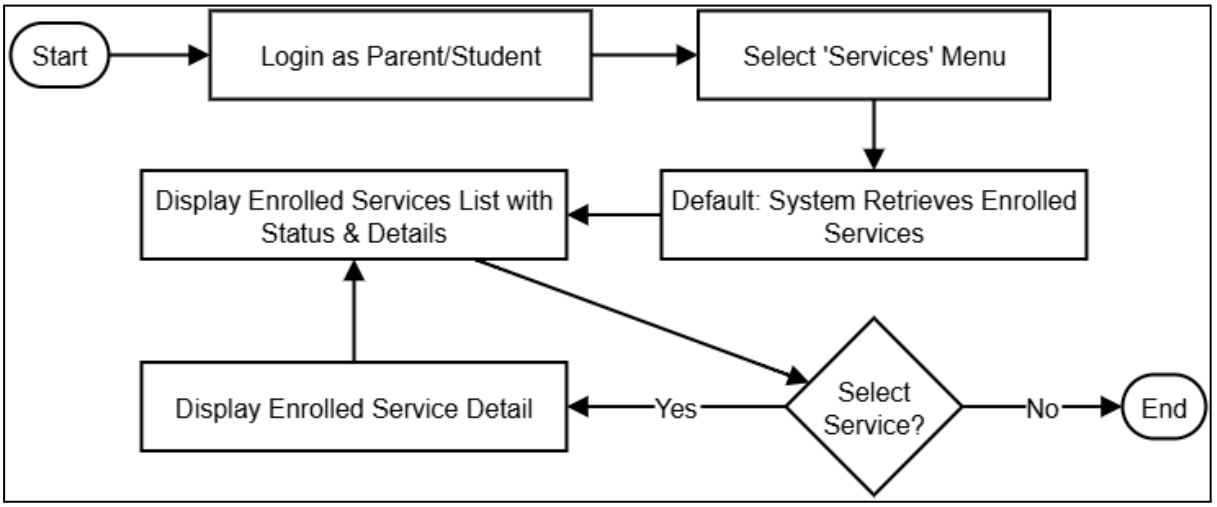


Table 4.6.4.4 Flowchart of View Enrolled Service

4.6.5 Chat Module

Chat Management

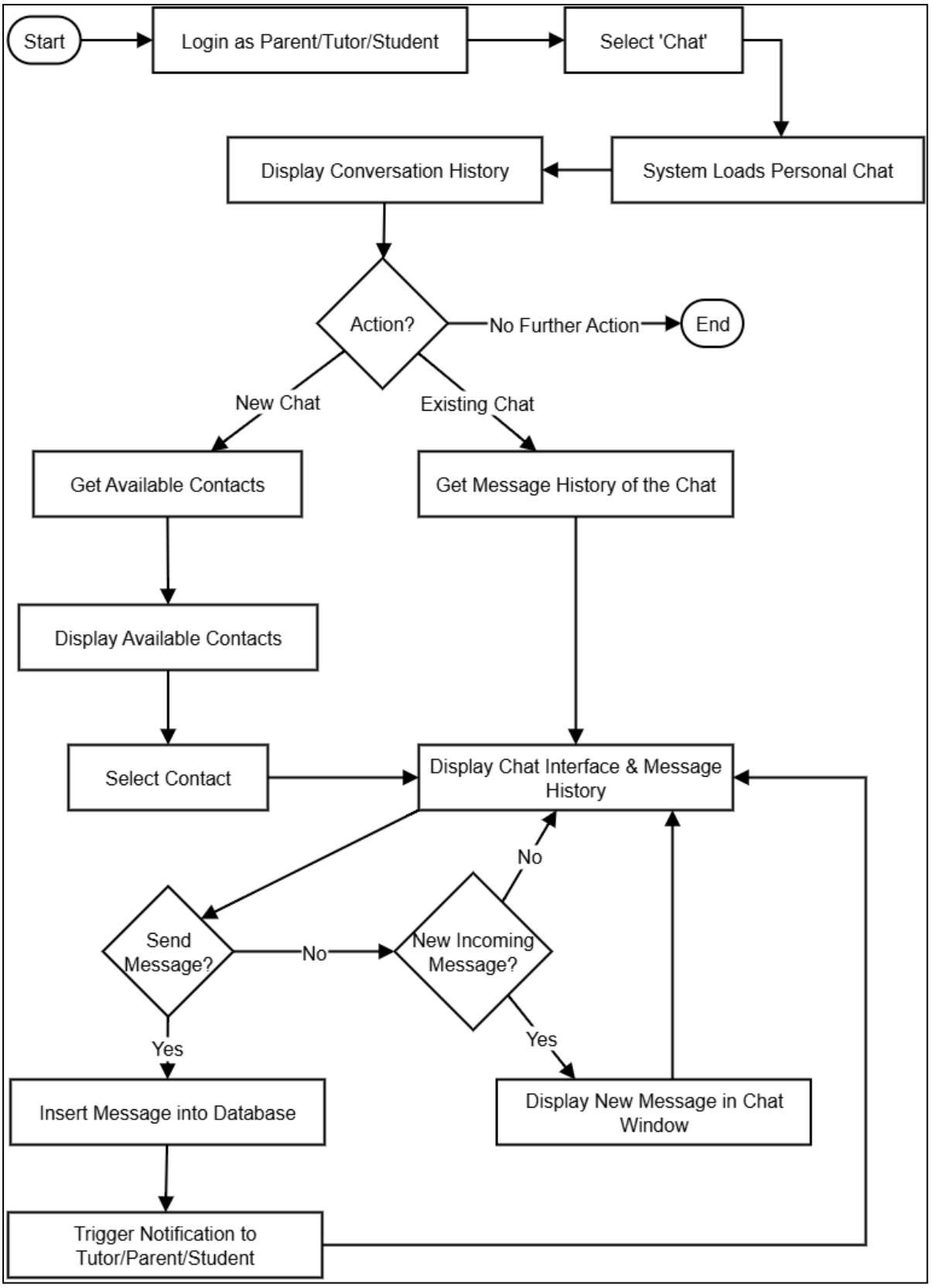


Table 4.6.5.1 Flowchart of Chat Management

4.7 Software Architecture Design

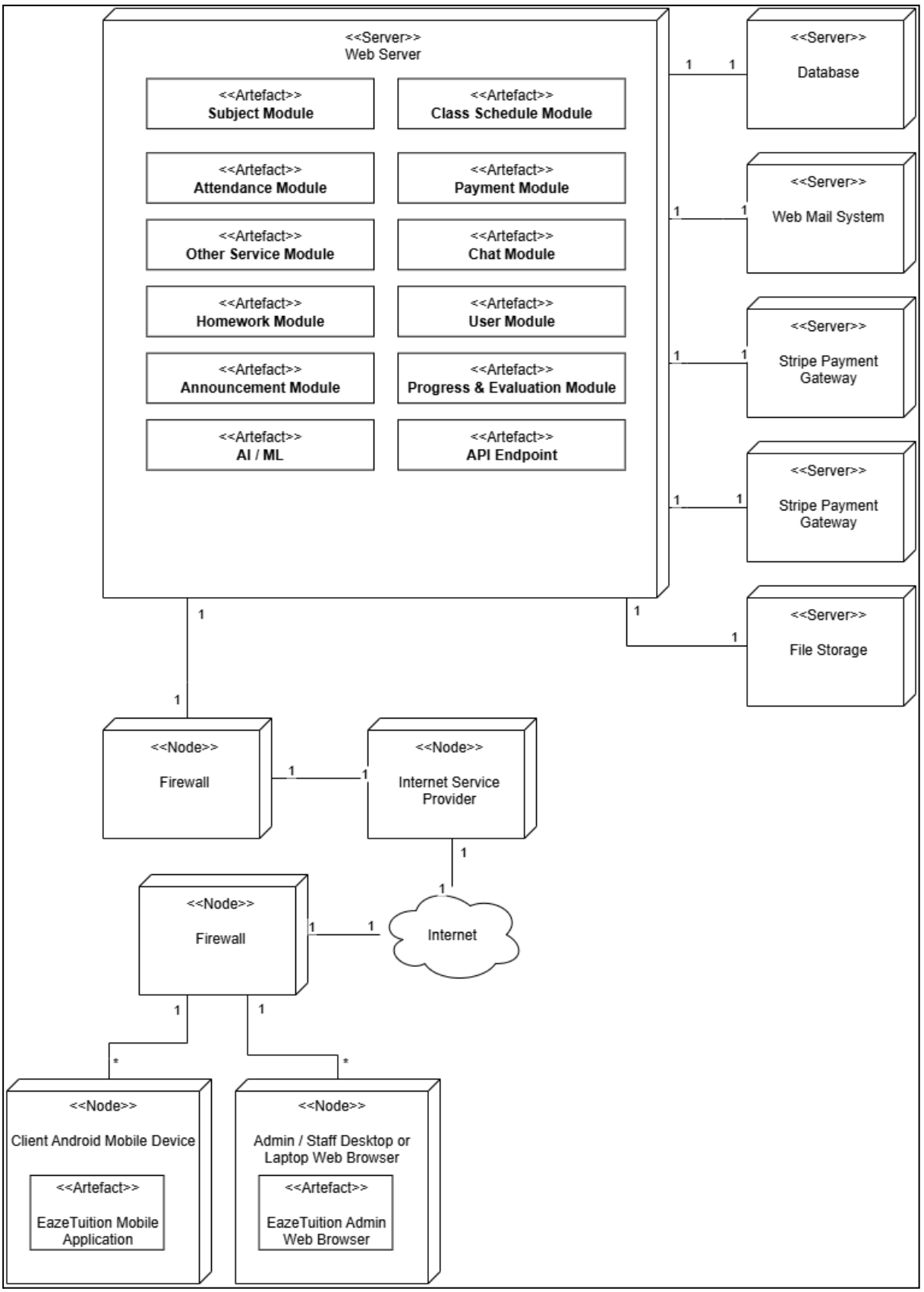


Table 4.7.1 Deployment Diagram of EazeTuition

4.8 2 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

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

4.8.2 Algorithm 1: K-Means Clustering

Purpose

Based on students lifestyle and study-related attributes such as study hours, sleep hours and activity hours to group students into categories. These groups will serve as profiles which represent different levels of stress, motivation and performance.

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

The algorithm will take selected columns from the merged datasets to group students into clusters representing different learning and lifestyle patterns.

Column Used:

- From **Student Lifestyle Dataset**
    - Study_Hours_Per_Day: average daily study time.
    - Sleep_Hours_Per_Day: average nightly sleep.
    - Physical_Activity_Hours_Per_Day OR Extracurricular_Hours_Per_Day: time spent on physical or non-academic activities.
- From **Student Performance Factors Dataset**
    - Exam Performance (used as an indirect validation feature)
    - Motivation Level
    - Stress Level

Process:

1. Merge tutoring sessions into Study Hours to get total academic load.

2. Apply K-means clustering on (Study Hours, Sleep Hours, Leisure Activity Hours, Motivation Level, Stress Level).
3. Students are grouped into clusters, e.g.:
    - Cluster 1: Balanced (moderate study, sufficient rest, low stress)
    - Cluster 2: Overloaded (high study hours, low sleep, high stress)
    - Cluster 3: Under-engaged (low study hours, high leisure, low motivation)
4. These clusters serve as the baseline profiles for interpreting whether a generated timetable is suitable.

Output: A cluster label for each student, which represents their typical lifestyle and performance tendency.

4.8.3 Algorithm 2: Decision Tree (Timetable Evaluation)

Purpose

Evaluate the generated time table is suitable for a student or not based on their assigned profiles from the clustering state. The Decision Tree determines whether the timetable supports balanced learning or poses risks that could induce stress and diminish learning effectiveness.

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. Decision trees provide a clear, rule-based mechanism to evaluate a timetable based on student characteristics and generate the recommendation level which is easy to understand.

How

1. **Timetable Analysis**.
    - From the generated timetable, calculate:
        - TotalStudyHours (including tutoring sessions)
        - TotalSleepHours
        - Leisure/ActivityHours
2. **Cluster Reference (from K-Means)**.
    - The student has already been grouped into a lifestyle cluster (Balanced, Overloaded, or Under-engaged).
3. **Classification Rules**
    - The Decision Tree compares the timetable's study, sleep, and leisure distribution against the expected ranges of the student's cluster.
        - If the timetable aligns with a healthy balance for that cluster → Suggested.

- If it slightly deviates (e.g., less sleep but still acceptable) → Lightly Suggested.
- If it strongly deviates (e.g., much more study, very little sleep) → Not Suggested.

Output: A recommendation label (Suggested, Lightly Suggested, Not Suggested) for a generated timetable

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, **two AI algorithms** which are the K-means clustering and decision tree classification, were designed to evaluate and recommend class schedules based on students' living habits and academic performance. 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.

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

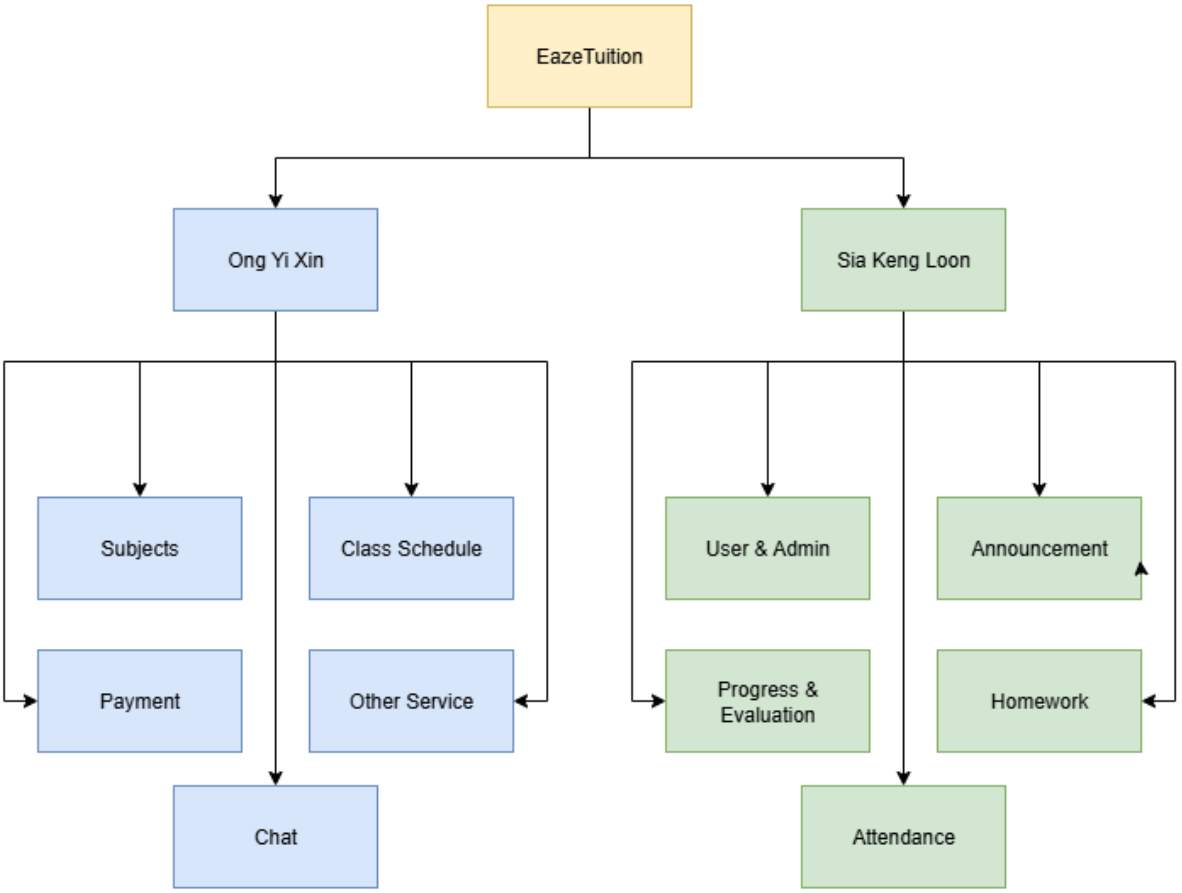


Figure 1.4.1 Milestone of EazeTuition Project

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