2024

# SOFTWARE ENGINEERING PROJECT - TIMETABLE MANAGEMENT SYSTEM



## PREPARED BY

SUDEEP KUMAR SINGH

SHUBHAM KUSHWAHA

ANKIT SARAWAG

ALOK SHARMA

RAJVEER SINGH


## FACULTY: MS. UMA OJHA

# SoftWare Engineering Project - Timetable Management System

Sudeep Kumar Singh (28021)
Shubham Kushwaha (28090)
Ankit Sarawag (28006)
Rajveer Singh (28072)
Alok Sharma (28086)

November 24, 2024

# Contents

**Contents** i

**List of Figures** iv

**List of Figures** iv

**List of Tables** v

**List of Tables** v

**1 Problem Statement** 1
- 1.1 Overview . . . 1
- 1.2 Challenges in the Current System . . . 1
- 1.3 Proposed Solution: Timetable Management System with Enhanced Functionality 2
- 1.4 Conclusion . . . 2

**2 Process Model: Agile Development Model** 3
- 2.1 Phases of the Agile Process . . . 3
  - 2.1.1 Requirement Gathering & Analysis . . . 3
  - 2.1.2 Planning . . . 3
  - 2.1.3 Design . . . 3
  - 2.1.4 Development . . . 3
  - 2.1.5 Testing . . . 4
  - 2.1.6 Deployment . . . 4
  - 2.1.7 Maintenance & Updates . . . 4
- 2.2 Why Agile? . . . 4

**3 Data Flow Diagram (DFD) - Level 0: Timetable Management System (TTMS)** 5
- 3.1 DFD Level 0 Diagram . . . 5
- 3.2 DFD Level 1 Diagram . . . 5
- 3.3 Structured Chart and Sequence Diagram . . . 6

**4 Data Dictionary** 9
- 4.1 Data Dictionary . . . 9

**5 Use Case Diagram: Timetable Management System (TTMS)** 11
- 5.1 Use Case Diagram: Timetable Management System . . . 11
  - 5.1.1 Description . . . 11
  - 5.1.2 Main Features and Functionalities . . . 11

5.1.3 Actors and Their Roles . . . 12
5.2 Use Case 1: fetchTimeTable . . . 13
Use Case 1: fetchTimeTable . . . 13
5.3 Use Case 2: pdfpage . . . 15
Use Case 2: pdfpage . . . 15

**6 SRS** **17**
6.1 Introduction . . . 17
6.1.1 Purpose . . . 17
6.1.2 Scope . . . 17
6.1.3 Definitions, Acronyms, and Abbreviations . . . 18
6.1.4 References . . . 18
6.1.5 Overview . . . 18
6.2 Overall Description . . . 18
6.2.1 Product Perspective . . . 19
6.2.2 Product Functions . . . 21
6.2.3 User Characteristics . . . 21
6.2.4 Constraints . . . 21
6.2.5 Assumptions and Dependencies . . . 22
6.2.6 Apportioning of Requirements . . . 22
6.3 Specific Requirements . . . 22
6.3.1 Functional Requirements . . . 22
6.3.2 External Interface Requirements . . . 23
6.3.3 Performance Requirements . . . 24
6.3.4 Design Constraints . . . 25
6.3.5 Software System Attributes . . . 25
6.4 Appendices . . . 25
6.4.1 Glossary . . . 26

**7 Function Point Analysis** **27**
7.1 Function Point Analysis . . . 27
7.1.1 Components Identification . . . 27
7.1.2 Complexity Weighting . . . 29
7.1.3 Function Point Calculation . . . 29

**8 Gantt Chart** **30**
8.1 Gantt Chart . . . 30
8.1.1 Project Timeline . . . 30
8.1.2 Task Dependencies . . . 31

**9 Testing** **33**
9.1 Unit Testing Overview . . . 33
9.1.1 Black Box Testing in Unit Testing . . . 33
9.1.2 White Box Testing in Unit Testing . . . 33
9.2 System Testing Overview . . . 39
9.2.1 Black Box Testing in System Testing . . . 39
9.2.2 White Box Testing in System Testing . . . 39
9.3 User Acceptance Testing (UAT) Overview . . . 40
9.3.1 Black Box Testing in UAT . . . 40

9.3.2 Alpha Testing . . . . . 41
9.3.3 Beta Testing . . . . . 42
9.3.4 Summary . . . . . 42

**10 Risk** **43**
10.1 Risk Table . . . . . 43
10.1.1 Identified Risks . . . . . 43
10.1.2 Mitigation Strategies . . . . . 43
10.1.3 Explanation of Probability and Impact . . . . . 45
10.1.4 Conclusion . . . . . 46

**11 Feasibility Study** **47**
11.1 Feasibility Study . . . . . 47
11.1.1 Technical Feasibility . . . . . 47
11.1.2 Operational Feasibility . . . . . 48
11.1.3 Financial Feasibility . . . . . 48
11.1.4 Conclusion . . . . . 49

**12 User Interface Design** **50**
12.1 UI Design Images . . . . . 50

# List of Figures


3.1 DFD Level 0: Overview of Timetable Management System . . . . . 5
3.2 DFD Level 1: Detailed Breakdown of Timetable Management System . . . . . 6
3.3 Structured Chart: Timetable Management System . . . . . 7
3.4 Sequence Diagram: Timetable Management System . . . . . 8
5.5 Use Case Diagram: Timetable Management System . . . . . 12
5.6 Use Case Diagram: fetchTimeTable . . . . . 14
5.7 Use Case Diagram: PDF Generation . . . . . 16
8.8 Gantt Chart For Timeline . . . . . 31
9.9 Flow Graph for Saving Timetable Locally . . . . . 34
12.10UI Design - Course Wise Timetable Screen . . . . . 50
12.11UI Design - Faculty Wise Timetable Screen . . . . . 50
12.12UI Design - Room Wise Timetable Screen . . . . . 50
12.13UI Design - Vacant Room Screen . . . . . 50
12.14UI Design - Admin Panel Screen . . . . . 51
12.15UI Design - Bookmarked Timetable Screen . . . . . 51
12.16UI Design - Timetable Interface . . . . . 51
12.17UI Design - Feedback Interface . . . . . 51
12.18UI Design - View Course(Admin only) Screen . . . . . 52
12.19UI Design - Update/Add Course Interface . . . . . 52
12.20UI Design - All Feedback Screen . . . . . 52
12.21UI Design - All Added Timetables Screen . . . . . 52

# List of Tables


4.1 Data Dictionary . . . . . 10
6.2 Definitions, Acronyms, and Abbreviations . . . . . 18
7.3 Complexity Weighting . . . . . 29
8.4 Project Timeline . . . . . 31
8.5 Task Dependencies . . . . . 32
10.6 Identified Risks for the Project . . . . . 43

# Problem Statement

## 1.1 Overview

The current timetable system at Atma Ram Sanatan Dharma College (ARSD) is inefficient and cluttered, presenting numerous challenges for students and faculty.

## 1.2 Challenges in the Current System

The issues in the current timetable system include:

1. **Cluttered Design:** The timetable is overcrowded with information, leading to confusion when trying to locate specific details about classes, rooms, and faculty. This results in a lack of clarity, making it difficult for users to navigate.
2. **Missing Timetables for GE, SEC, and VAC:** General Elective (GE), Skill Enhancement Course (SEC), and Value Added Course (VAC) timetables are missing, leaving students without proper access to complete schedule information.
3. **Difficulty in Access:** The timetable is hard to navigate, especially when filtering for course-wise, room-wise, or faculty-wise timetables. This leads to time-consuming searches for specific schedules.
4. **Lack of Real-Time Updates:** Any changes or updates to the timetable, such as room changes or canceled classes, are not communicated efficiently. This results in students attending the wrong classes or missing schedule adjustments altogether.
5. **Inability to Find Vacant Rooms:** The current system does not provide a way for students or faculty to identify vacant rooms for impromptu meetings or study sessions, causing underutilization of available spaces.
6. **Limited Accessibility:** The timetable is not available offline, and there is no easy way to generate or download the schedule in formats like PDF, making it inaccessible in scenarios where students may need offline access.

These challenges hinder the overall user experience, creating unnecessary confusion and making it harder to maintain an organized academic schedule. A new system is required to generate well-structured, easily accessible timetables, with real-time updates, proper filtering, and offline accessibility options like PDFs.

## 1.3 Proposed Solution: Timetable Management System with Enhanced Functionality

To address these challenges, a new Timetable Management System is proposed, leveraging modern web technologies to provide a clean, user-friendly interface that improves accessibility and usability for students and faculty. This system will streamline the process of timetable creation, viewing, and updating by focusing on the following key features:

1. **Clarity and Structure:** The new system will present the timetable in a well-structured format, ensuring that each cell contains clear, non-overlapping information about class times, subjects, room numbers, and faculty names. GE, SEC, and VAC timetables will be integrated seamlessly alongside regular course schedules, providing a comprehensive view for students enrolled in these courses.

2. **User-Friendly Navigation:** The system will offer filtering options to view timetables based on specific criteria, such as course-wise, room-wise, and faculty-wise filters. This will enable users to quickly access the relevant information they need without sifting through an overwhelming amount of data.

3. **Real-Time Updates:** The system will be capable of real-time updates, allowing for any changes to be reflected immediately and communicated to students via notifications or alerts. This will prevent confusion caused by last-minute changes to class schedules.

4. **Vacant Room Identification:** A room vacancy tracking feature will be implemented, allowing users to easily find available rooms for impromptu meetings, study sessions, or extra-curricular activities. This feature will also provide insights into room bookings and availability.

5. **Offline Accessibility:** The system will support generating timetables in various formats such as PDF, enabling users to download the timetable and access it offline for future reference.

6. **Mobile-Friendly Design:** The system will be designed to be accessible via mobile devices, ensuring that students and faculty can easily view and manage their schedules from their phones or tablets.

## 1.4 Conclusion

By implementing this Timetable Management System, Atma Ram Sanatan Dharma College will provide its students and faculty with an efficient, easy-to-use platform for viewing and managing schedules, reducing confusion and improving overall productivity. The new system will not only enhance the user experience but also support better academic planning and time management for everyone involved.

# Process Model: Agile Development Model

Given the dynamic nature of timetable adjustments and the need for constant improvements, the Agile Development Model is the most suitable approach for developing the new timetable system at ARSD College. Agile allows for iterative development, frequent feedback, and real-time changes based on user needs and feedback.

## 2.1 Phases of the Agile Process

### 2.1.1 Requirement Gathering & Analysis

- Gather detailed requirements from students, faculty, and administration to understand their challenges with the existing system.
- Conduct user interviews and surveys to define what features (e.g., GE, SEC, VAC timetables, room vacancy tracking, etc.) are needed in the new system.
- Identify the need for real-time updates and offline availability (e.g., PDF download).

### 2.1.2 Planning

- Divide the project into smaller tasks or ”sprints,” with each sprint focusing on delivering a specific feature (e.g., user-friendly layout, filtering system, GE timetable integration).
- Create a product backlog to manage priorities and ensure features like vacant room tracking or PDF generation are included.

### 2.1.3 Design

- Create wireframes and mockups for the new timetable interface, ensuring user-friendly navigation and clear separation of subjects, room numbers, and faculty names.
- Design a system architecture that supports filtering by room, course, and faculty, and integrates real-time updates.
- Plan for both web and mobile accessibility.

### 2.1.4 Development

- Development is carried out incrementally, with each sprint focusing on completing a functional part of the system.

- The first sprint may involve setting up the basic timetable structure, followed by adding real-time updates, GE/SEC/VAC timetables, room vacancy features, and finally, PDF generation.
- Regular stand-up meetings are held to track progress.

### 2.1.5 Testing

- Each feature is tested individually as it's developed, followed by comprehensive integration testing.
- Testing includes ensuring that the timetable can filter correctly by faculty, course, and room, and that real-time updates are correctly reflected.
- Perform user testing to ensure the interface is clear and easy to use.

### 2.1.6 Deployment

- Once testing is complete, the system is deployed incrementally, with new features being rolled out after each sprint.
- A feedback loop is maintained to address any bugs or usability issues that arise.

### 2.1.7 Maintenance & Updates

- After deployment, regular maintenance will be conducted to ensure that real-time updates are working smoothly.
- User feedback will be incorporated to improve the system over time, and any necessary updates (e.g., to accommodate new courses or changes in faculty) will be implemented.

## 2.2 Why Agile?

- **Flexibility:** Agile allows the development team to respond to real-time feedback from students and faculty and make necessary adjustments quickly.
- **Continuous Delivery:** With Agile, the timetable system can be developed and improved in iterations, allowing for a more adaptable and user-focused system.
- **User-Centric:** The approach ensures that user experience issues like clutter, missing timetables, or real-time updates are addressed early on based on feedback, improving overall satisfaction.

# Data Flow Diagram (DFD) - Level 0 and 1: Timetable Management System (TTMS)

The DFD Level 0 for the Timetable Management System outlines the system's functionalities and interactions with external entities, focusing on how students and faculty can retrieve timetable information and check for vacant rooms based on specific criteria.

## 3.1 DFD Level 0 Diagram

The diagram below shows the high-level overview of the Timetable Management System, depicting the major processes, data stores, and interactions with external entities.

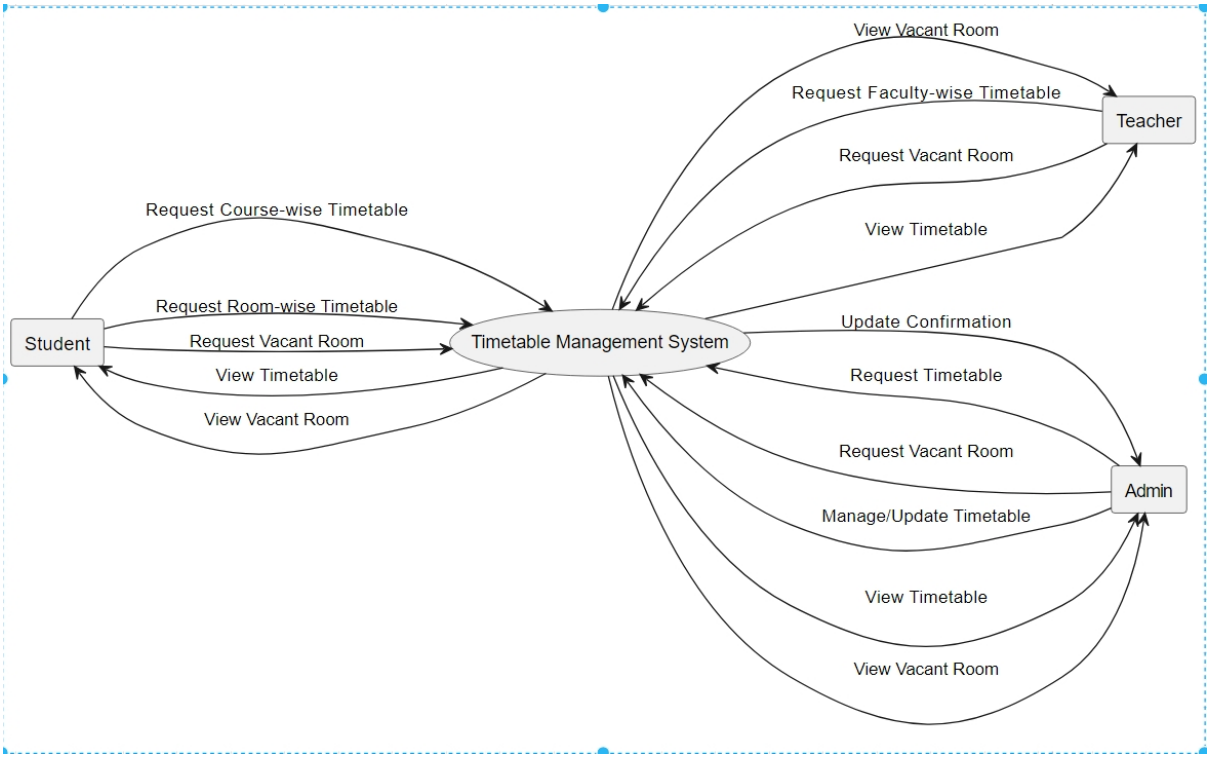


Figure 3.1: DFD Level 0: Overview of Timetable Management System

## 3.2 DFD Level 1 Diagram

In DFD Level 1, we break down the main processes further to provide a more detailed view of the system. This level illustrates the interactions between the different components of the

Timetable Management System.

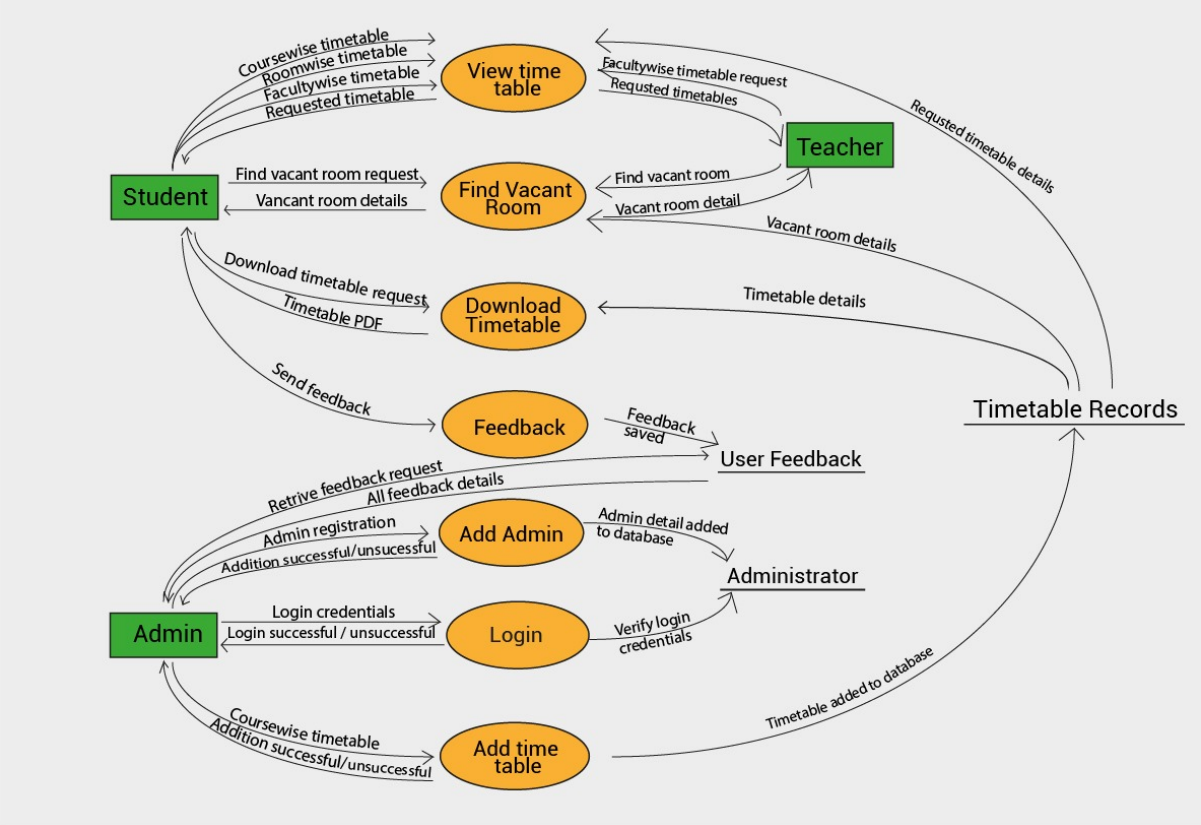


Figure 3.2: DFD Level 1: Detailed Breakdown of Timetable Management System

## 3.3 Structured Chart and Sequence Diagram

Below are the **Structured Chart** and **Sequence Diagram** for the Timetable Management System. These diagrams provide a more detailed view of the system's architecture and interaction flow.

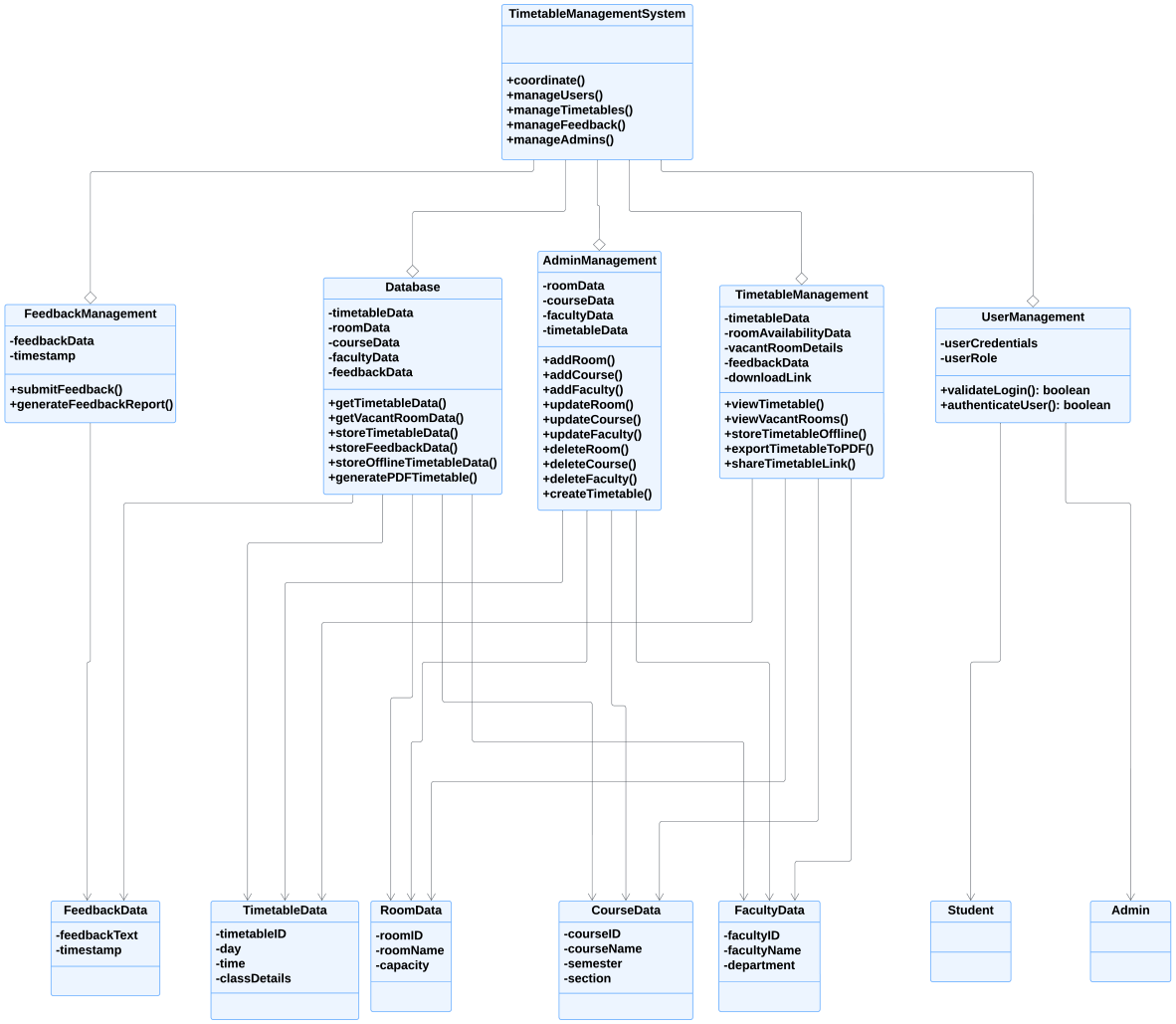


Figure 3.3: Structured Chart: Timetable Management System

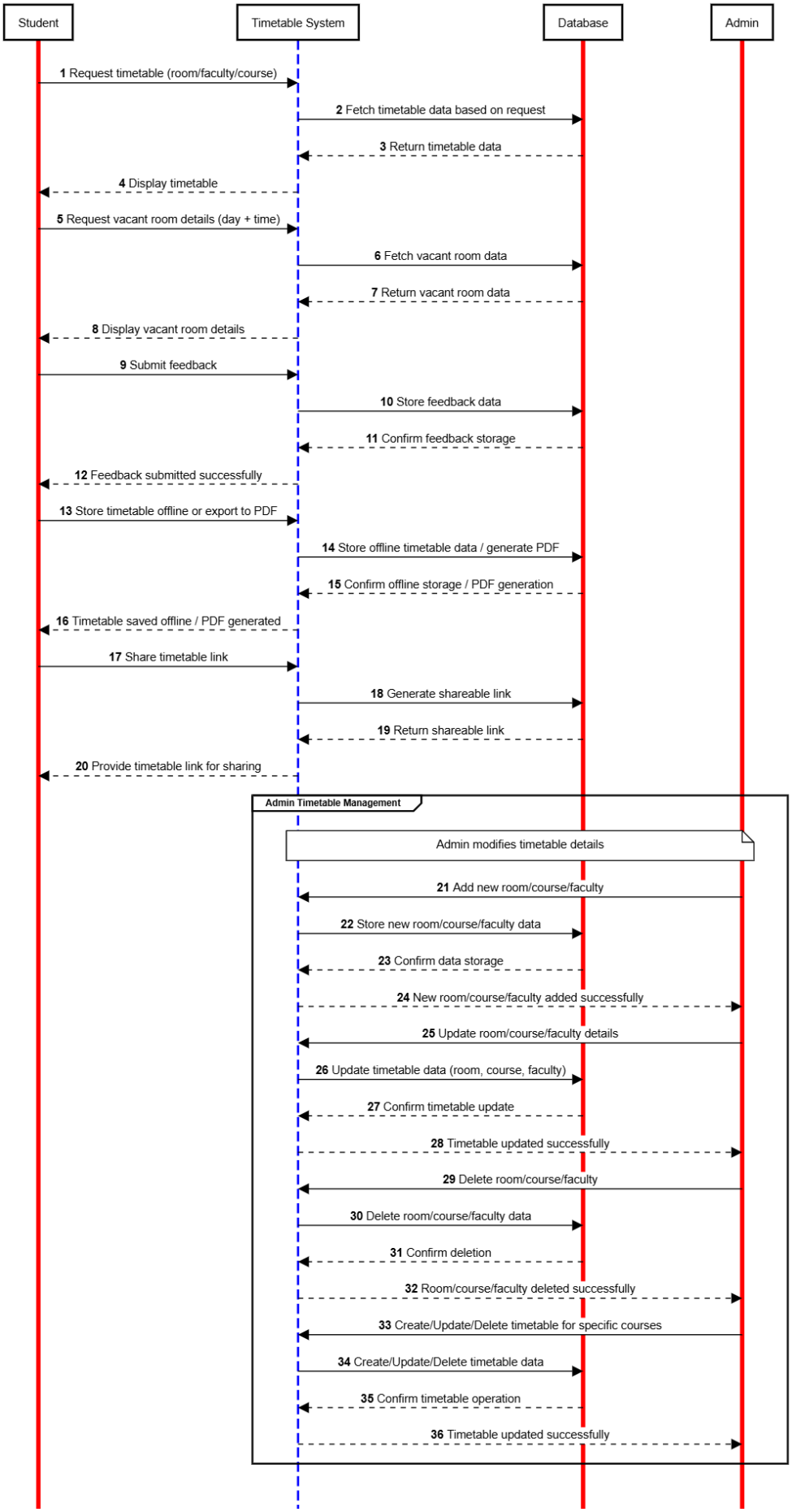


Figure 3.4: Sequence Diagram: Timetable Management System

# Data Dictionary

## 4.1 Data Dictionary

| Entity | Description | Attributes | Type |
|---|---|---|---|
| Login Details | User credentials for authentication. | {userName + password} | {String} |
| **Timetable Request**<br>Data required to retrieve timetables based on the user's criteria. | | | |
| Faculty Wise | Inputs for retrieving a specific timetable based on faculty. | {facultyName} | {String} |
| Room Wise | Inputs for retrieving a timetable based on room availability. | {roomName} | {String} |
| Course Wise | Inputs for retrieving a course-wise timetable. | {course + semester + section} (for regular courses) / {course + semester + paperName} (for GE/VAC/SEC) | {String} |
| Timetable Data | Information returned to the user about their timetable, organized by day and time. | {timetableID + {day + {time + {classDetails}}}} | {Object} |
| Day | Represents each day in the timetable. | {"Monday", "Tuesday", etc.} | {String} |
| Time Slot | Represents the specific time slots for each class on a given day. | {"09:30", "11:30", etc.} | {String} |

| Entity | Description | Attributes | Type |
|---|---|---|---|
| Class Details | Contains the specific details of the class, such as paper name, lecture type, room number, and faculty. | {timetableID + {day + {time + {paperName + paperName2 + paperName3 + roomName + roomName2 + lectureType + lectureType2 + lectureType3 + faculty1 + faculty2 + faculty3 + faculty4 + faculty2_1 + faculty3_1 + grpName + grpA + grpB}}}} | {Object} |
| Vacant Room Request | Inputs required to check room availability. | {day + time} | {String} |
| Vacant Room Data | List of available rooms for a particular day and time. | {availableRoom} | {Array} |
| Feedback Details | User feedback provided about the system. | {feedbackText + timestamp} | {String, Date} |
| Download Options | Data related to the download of timetable files. | {timetableID + downloadLink + fileType (PDF) + timestamp} | {String, Date} |
| Cache Data | Offline storage of timetable data in the user's browser for offline access. | {generalID + {timetableID + {day + {time + {classDetails}}}}} | {Object} |

Table 4.1: Data Dictionary

# Use Case Diagram: Timetable Management System (TTMS)

## 5.1 Use Case Diagram: Timetable Management System

### 5.1.1 Description

The Use Case Diagram for the Timetable Management System (TTMS) represents the interaction between different types of users and the functionalities they can access. The primary actors identified are:

- **Student:** Accesses features like viewing timetables, downloading rooms, and sharing timetable information.
- **Teacher:** Utilizes the system for viewing timetables and fetching specific information such as vacant rooms or timetables based on faculty.
- **Admin:** Manages the timetable system, including modifying, adding, or deleting rooms, faculty, and courses.

### 5.1.2 Main Features and Functionalities

1. **View Timetable:**
    - Students, teachers, and admins can view timetables based on different filters such as room, course, and faculty.
    - Includes fetching vacant rooms and timetables.
2. **Manage Timetable:**
    - The admin can change timetables based on rooms, faculty, and courses.
    - Includes options to add, delete, or modify details of rooms, faculty, and courses.
3. **Vacant Room Feature:**
    - Allows teachers and students to find vacant rooms and share this information if needed.
4. **Additional Features:**
    - Users can save timetables offline and share them.
    - Feedback can be submitted to improve the system.

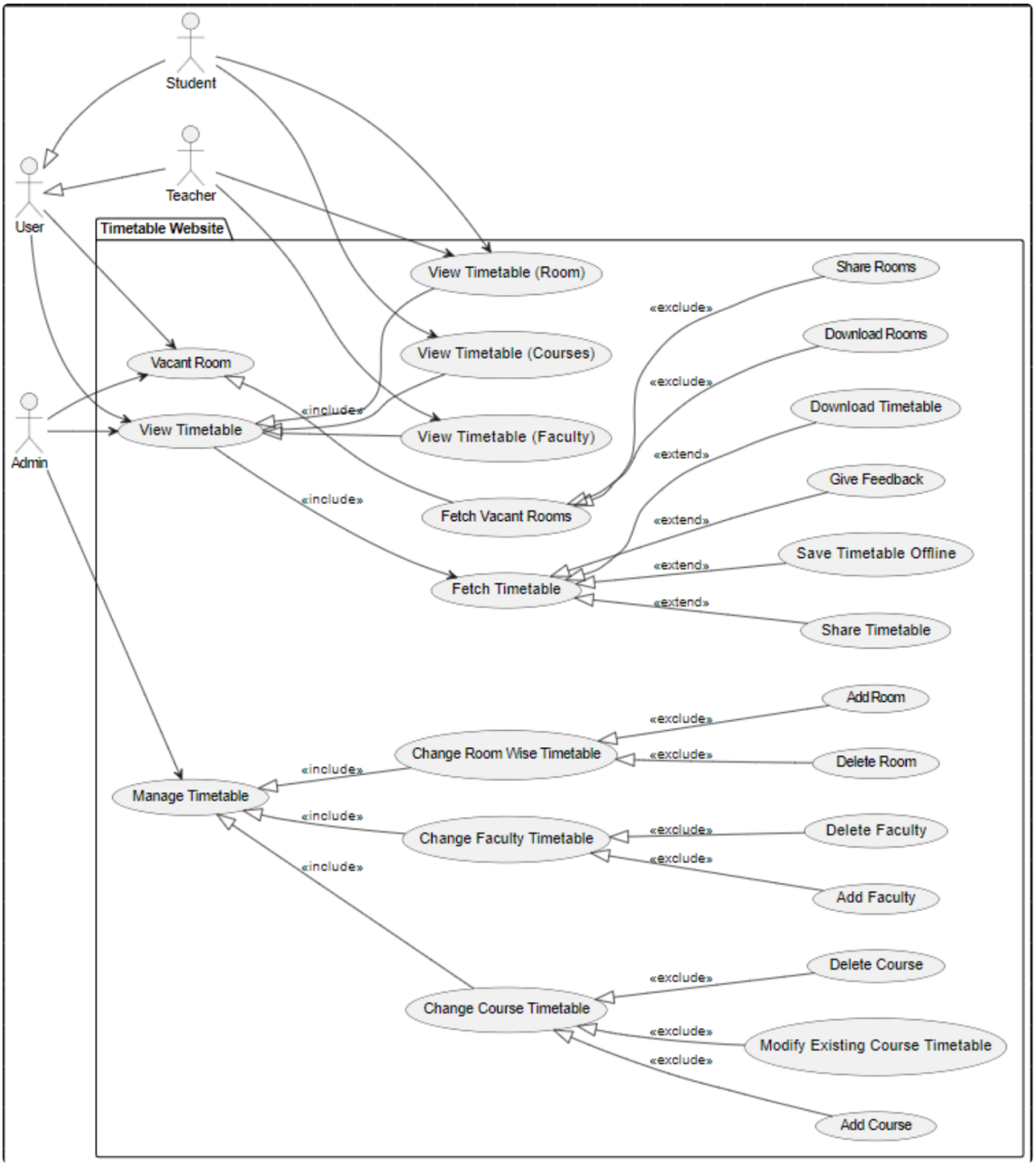


Figure 5.5: Use Case Diagram: Timetable Management System

### 5.1.3 Actors and Their Roles

- **User:** A general role encompassing students and teachers, who interact with the system primarily for timetable viewing and related functionalities.
- **Student:** Focused on viewing and downloading timetable details.
- **Teacher:** Uses the system to fetch details about faculty schedules, vacant rooms, and related features.
- **Admin:** Responsible for managing and maintaining the system, including CRUD operations for timetable details.

## 5.2 Use Case 1: fetchTimeTable

**Use Case Name:** fetchTimeTable

**Actors:** - Student - Faculty

**Description:** This use case defines the process of fetching the timetable based on the selected filter criteria such as room, course, faculty, etc. The function either fetches the timetable from local storage or from the Firebase database based on the conditions.

**Preconditions:** - User has access to the Timetable Management System (TTMS). - User is logged in with appropriate access rights.

**Postconditions:** - The timetable data is fetched and returned as a map containing the details for each day of the week.

**Flow of Events:**

1. The user initiates the timetable fetching process.
2. The system checks if the timetable is available in the local storage.
3. If the timetable is not in the local storage, the system queries the Firebase database for the timetable based on the user's criteria (room, course, faculty).
4. The data is processed and returned as a map of days and periods.
5. The fetched timetable is displayed to the user.

**Alternative Flows:**

- If the query does not return any data, the system will show an appropriate error message.
- If there is a failure in fetching data from Firebase, a retry mechanism may be initiated.

**Exceptions:**

- Timetable data not found in the local storage.
- Firebase database query failure.

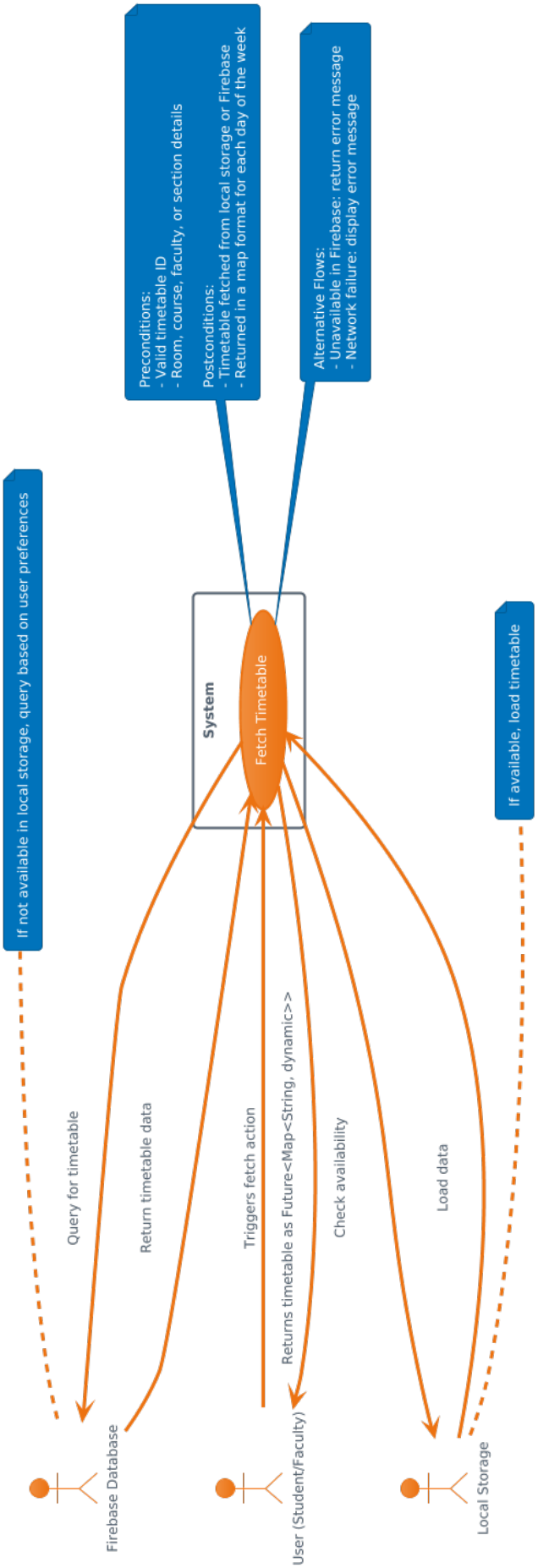


Figure 5.6: Use Case Diagram: fetchTimeTable

## 5.3 Use Case 2: pdfpage

**Use Case Name:** pdfpage

**Actors:** - Student - Faculty

**Description:** This use case generates a PDF page with the timetable, including the details of periods, rooms, and faculty for each time slot, based on the user's course and faculty details.

**Preconditions:** - User has a valid timetable for the selected course, semester, and faculty. - User has the necessary permission to generate a PDF.

**Postconditions:** - A PDF document is generated and available for download or sharing.

**Flow of Events:**

1. The user triggers the PDF generation process.
2. The system gathers data related to the selected course, faculty, semester, and room.
3. The system generates a formatted timetable in a PDF document.
4. The generated PDF is displayed or made available for download.

**Alternative Flows:**

- If the user does not have the required data (e.g., course, semester, faculty), an error message is shown.

**Exceptions:**

- Failure to generate the PDF due to missing data.
- Error while rendering the timetable in the PDF.

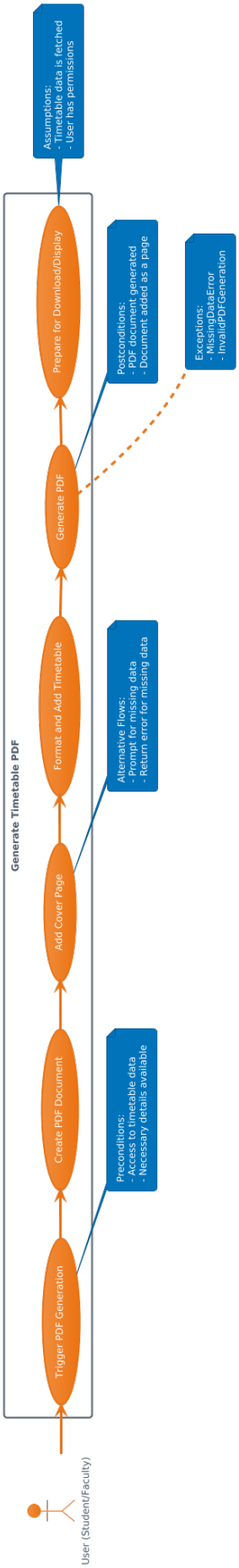


Figure 5.7: Use Case Diagram: PDF Generation

# SRS

## 6.1 Introduction

The Introduction section provides an overview of the project, defining its goals, scope, and terminology. This section serves as a foundation to understand the document's purpose and its relevance to the stakeholders.

### 6.1.1 Purpose

The purpose of this document is to define the Software Requirements Specification (SRS) for the Timetable Management System (TTMS).

This document outlines:

- The functionalities and capabilities of the system.
- The non-functional requirements such as performance, reliability, and scalability.
- The interface requirements for users, software, and hardware components.

The TTMS aims to automate and optimize timetable creation for educational institutions, minimizing conflicts and maximizing resource utilization. It also ensures seamless access to schedules, updates, and notifications for students and faculty members.

### 6.1.2 Scope

The Timetable Management System (TTMS) is designed to address challenges in scheduling classes, faculty allocation, and room management. The system's primary objectives are:

- **Automated Scheduling:** Generate conflict-free timetables.
- **Resource Management:** Allocate rooms and faculty efficiently.
- **Accessibility:** Provide real-time access to timetables for students and staff.
- **Notifications:** Send alerts for changes in schedules.
- **Analytics:** Provide insights into resource usage and timetable patterns.

**Key Users:**

- **Students:** Access class schedules and updates.
- **Faculty:** View teaching schedules and make adjustments.
- **Administrators:** Manage overall scheduling and resolve conflicts.

### 6.1.3 Definitions, Acronyms, and Abbreviations

| Term | Definition |
|---|---|
| TTMS | Timetable Management System |
| SRS | Software Requirements Specification |
| Admin | ”A user with the authority to manage timetables, faculty, and room assignments” |
| API | ”Application Programming Interface, facilitating communication between components” |
| UI | ”User Interface, the design and layout through which users interact with the system” |
| Resource Allocation | ”Assigning available resources (rooms, faculty) for specific times and purposes” |

Table 6.2: Definitions, Acronyms, and Abbreviations

### 6.1.4 References

- IEEE 830-1998 Standard for Software Requirements Specifications
- Institution’s policies on academic scheduling
- Documentation on Firebase (for backend integration)
- System documentation for compatible platforms (Linux, Windows, etc.)

### 6.1.5 Overview

The document is organized as follows:

- **Section 1 (Introduction):** Provides the purpose, scope, and background of the project.
- **Section 2 (Overall Description):** Describes the system’s high-level functionality, user interaction, and constraints.
- **Section 3 (Specific Requirements):** Details the functional, non-functional, and interface requirements.
- **Section 4 (Appendices):** Contains supplementary information like a glossary and references.

This SRS document aims to serve as a guideline for developers, testers, and stakeholders, ensuring clarity and consistency throughout the development lifecycle.

## 6.2 Overall Description

This section provides an overarching view of the Timetable Management System (TTMS), describing its context, interaction with external components, and the environment it operates in. It also outlines the system’s functions, user characteristics, constraints, assumptions, and requirements distribution.

### 6.2.1 Product Perspective

The Timetable Management System is a standalone application with integration capabilities, aimed at automating academic scheduling. It leverages algorithms to manage resource allocation and ensure conflict-free schedules.

**System Architecture:** TTMS operates in a client-server model, where the backend server handles scheduling logic and resource management, while the frontend provides user access.

**Position in the Environment:** TTMS interacts with external systems, such as institutional databases, for fetching faculty and student data. It may also interface with communication tools like email and notification systems.

#### System Interfaces

TTMS interacts with the following external systems:

- **Student Information System (SIS):** To retrieve student enrollment data.
- **Faculty Management System:** For faculty availability and workload.
- **Notification Services:** For sending email and SMS alerts.
- **Database Management System:** For persistent storage of schedules and logs.

#### User Interfaces

The system offers a modern and intuitive interface designed for three user types:

- **Students:**
  - View class schedules, room details, and exam timetables.
  - Receive notifications for changes or cancellations.
- **Faculty:**
  - Access assigned teaching schedules.
  - Request adjustments or report unavailability.
- **Administrators:**
  - Create and manage timetables.
  - Resolve scheduling conflicts and allocate resources.

**UI Design includes:**

- **Dashboard:** Displays schedules, pending tasks, and notifications.
- **Responsive Design:** Ensures compatibility across devices.

### Hardware Interfaces

TTMS requires interaction with the following hardware components:

- **Client Devices:** Computers, tablets, and smartphones for end-users.
- **Servers:** For backend processing and database management.
- **Network Devices:** Routers and switches to enable communication.

### Software Interfaces

The system integrates with various software components:

- **Operating Systems:** Compatible with Windows, Linux, Android, and iOS.
- **Web Browsers:** Supports Chrome, Firefox, and Edge for web-based access.
- **Backend APIs:** RESTful APIs for data exchange and integration.
- **Database Software:** Firebase or MySQL for data storage and retrieval.

### Communications Interfaces

TTMS employs secure communication protocols to ensure data integrity and confidentiality:

- **HTTPS:** For encrypted web communication.
- **SMTP:** To send email notifications.
- **WebSocket:** For real-time updates on schedule changes.

### Memory Constraints

The system is optimized for efficient memory usage:

- **Client Devices:** Requires at least 2GB RAM and 1GB storage for local data caching.
- **Server Requirements:** Needs 8GB RAM and scalable storage for large datasets, especially for institutions with over 10,000 students.

### Operations

The system supports the following operational features:

- **Scheduling Algorithms:** Automatically resolve conflicts and optimize resource allocation.
- **Backup Operations:** Daily backups ensure data recovery in case of failures.
- **Real-Time Updates:** Changes to schedules are immediately reflected across all interfaces.

#### Site Adaptation Requirements

TTMS can be adapted for different institutions with minimal changes:

- **Custom Branding:** Allows integration of institutional logos and color schemes.
- **Localization:** Supports multiple languages for diverse user bases.
- **Scalability:** Configurable for institutions ranging from small schools to large universities.

### 6.2.2 Product Functions

The primary functions of TTMS include:

- Automated Timetable Generation: Create conflict-free timetables for students and faculty.
- Resource Management: Allocate classrooms, labs, and other facilities efficiently.
- Real-Time Notifications: Alert users of schedule changes or resource reallocations.
- User Access Control: Restrict permissions based on roles (e.g., student, faculty, admin).
- Reports and Analytics: Generate detailed reports on resource usage and schedule adherence.

### 6.2.3 User Characteristics

The user base is diverse, requiring different levels of technical proficiency:

- **Students:** Moderate technical knowledge; accustomed to mobile apps and web platforms.
- **Faculty:** Moderate to advanced proficiency in digital tools.
- **Administrators:** Advanced technical expertise for managing backend operations.

### 6.2.4 Constraints

#### Regulatory Policies

TTMS must adhere to institutional regulations, including:

- Maximum hours per week for faculty workloads.
- Legal compliance with data privacy laws such as GDPR or India's Data Protection Bill.

#### Platform Limitations

- Must run on devices with limited processing power (e.g., older smartphones).
- Compatibility with existing infrastructure, such as legacy databases.

### 6.2.5 Assumptions and Dependencies

The following assumptions and dependencies are considered:

- **Stable Internet Connectivity:** Required for real-time updates and cloud synchronization.
- **Data Accuracy:** TTMS assumes that input data (e.g., faculty availability) is accurate and up-to-date.
- **Third-Party Services:** Dependence on Firebase for backend support and email APIs for notifications.

### 6.2.6 Apportioning of Requirements

- **Critical Requirements:** Automated timetable generation, real-time conflict resolution, and notifications.
- **Secondary Requirements:** Report generation, advanced analytics, and localization.
- **Future Enhancements:** Integration with attendance systems and advanced AI for predictive scheduling.

This detailed description ensures that all stakeholders have a clear understanding of the system's context, functionality, and limitations.

## 6.3 Specific Requirements

This section provides a comprehensive breakdown of the specific functional, external, and performance requirements, design constraints, and attributes for the Timetable Management System (TTMS). The following subsections detail these specifications.

### 6.3.1 Functional Requirements

The functional requirements describe the primary operations the system must support to meet user needs.

#### Timetable Management

**Description:** TTMS should be able to automatically generate and manage academic timetables for students, faculty, and classrooms, ensuring there are no conflicts. **Functions:**

- Allocate classes to faculty and rooms based on their availability.
- Consider room capacity and resources (e.g., projectors, labs).
- Provide administrators with manual overrides and the ability to edit generated timetables.
- Allow students and faculty to view their schedules by day, week, or semester.

#### Vacant Room Identification

**Description:** The system must identify vacant classrooms and resources during the scheduling process. **Functions:**

- Display available classrooms based on time slots.
- Provide real-time availability status for rooms and resources.
- Allow administrators to reserve rooms for special events or lectures.

#### Real-Time Updates

**Description:** The system must provide real-time updates for schedule changes, cancellations, or room reallocations. **Functions:**

- Automatically update timetables across all user interfaces when a change occurs.
- Send notifications to users about changes or cancellations.

#### Offline Accessibility

**Description:** TTMS should allow offline access to timetables and related information. **Functions:**

- Cache the last updated timetable on the client device for offline viewing.
- Sync changes when the device reconnects to the internet.
- Ensure that offline access does not affect the overall integrity and synchronization of the system.

#### Notifications and Alerts

**Description:** The system should alert users about important schedule changes, announcements, or upcoming events. **Functions:**

- Send automatic notifications via email, SMS, or in-app alerts.
- Allow users to configure notification preferences (e.g., time before an event).
- Provide real-time alerts on mobile and web interfaces for schedule updates or conflicts.

### 6.3.2 External Interface Requirements

External interfaces define how TTMS interacts with other software or systems.

#### User Interfaces

**Description:** The system must provide accessible and intuitive user interfaces for students, faculty, and administrators. **Requirements:**

- A web-based interface accessible on multiple devices.
- Mobile-friendly for smartphones and tablets.
- A dashboard for users to view their schedules, upcoming events, and notifications.
- Admin interfaces for manual timetable creation, conflict resolution, and report generation.

#### Software Interfaces

**Description:** TTMS must integrate with external systems for data synchronization and communication. **Requirements:**

- Integration with Student Information Systems (SIS) for student data.
- Integration with Faculty Management Systems to fetch availability data.
- RESTful APIs for data exchange with external services like notifications, email, or analytics systems.
- Compatibility with external Calendar Apps for schedule exports and imports.

#### Communications Interfaces

**Description:** The system should support secure communication channels to facilitate data exchange. **Requirements:**

- Use HTTPS for secure data transfer.
- Integration with SMTP for email notifications.
- WebSockets or Polling for real-time updates.
- SMS Gateway integration for text message alerts.

### 6.3.3 Performance Requirements

**Response Time:**

- The system must provide a response time of less than 2 seconds for generating timetables and searching for available rooms.
- Timetable changes must be reflected within 5 seconds across all user interfaces.

**Concurrency:**

- The system should support at least 500 concurrent users without performance degradation.

**Availability:**

- The system should have an uptime of 99.9% to ensure uninterrupted access.

### 6.3.4 Design Constraints

Design constraints include limitations in terms of hardware, software, and system integration.

**Hardware Limitations**

**Client Devices:**

- The system should function on devices with a minimum of 2GB RAM and 1GB of available storage.
- Mobile devices should run on iOS 11+ or Android 6.0+.

**Server Requirements:**

- Servers hosting the backend should have at least 8GB RAM and 500GB storage for scalability.

**Software Framework Dependencies**

- The system will be built on Flutter for mobile and web app development.
- Firebase or MySQL will be used for the database.
- Node.js or Django will be used for backend services.
- The system will require OAuth for secure user authentication.

### 6.3.5 Software System Attributes

These attributes define the overall performance, security, and usability characteristics of the system.

**Security**

**Authentication:**

- Implement role-based access control (RBAC) for users (students, faculty, admin).
- Integrate multi-factor authentication (MFA) for administrators and sensitive access.

**Data Encryption:**

- Encrypt sensitive data at rest and in transit using AES and TLS.

**Access Control:**

- Ensure that only authorized users can modify schedules or view confidential data.

## 6.4 Appendices

The appendices provide supplementary information that supports the document's content, helping users better understand technical terms, figures, and tables referenced throughout the Software Requirements Specification (SRS).

### 6.4.1 Glossary

This section includes definitions of technical terms, acronyms, and concepts used in the document.

- **Timetable Management System (TTMS):** A system designed to automate the creation, management, and display of academic timetables for students, faculty, and staff.
- **API (Application Programming Interface):** A set of tools and protocols for building and interacting with software applications.
- **RBAC (Role-Based Access Control):** A security mechanism that restricts system access based on the roles of individual users.
- **RESTful API:** A lightweight, stateless architecture used for designing networked applications, where resources are represented by URLs and can be accessed via HTTP methods (GET, POST, PUT, DELETE).
- **SIS (Student Information System):** A software application for managing student data, including enrollment, grades, and course schedules.
- **OAuth:** An open standard for access delegation, commonly used for secure user authentication and authorization in web services.
- **MFA (Multi-Factor Authentication):** A security process that requires users to provide two or more verification factors to gain access to a resource.

# Function Point Analysis

## 7.1 Function Point Analysis

Function Point Analysis (FPA) is a method used to estimate the size and complexity of a software system based on its functionality from the user's perspective. This method helps in predicting the time and resources required for the development process, as well as estimating the system's overall performance and maintainability. FPA is based on the function provided to the user and focuses on the system's functional requirements.

### 7.1.1 Components Identification

**External Inputs (EI)**

Inputs refer to user-driven data entering the system.

- Admin Panel Login Screen
- Dropdown Selection for Timetable Display
- Add/Edit/Delete Courses
- Add/Edit/Delete Faculties
- Add/Edit/Delete Rooms
- Add/Edit/Delete Sections
- Add/Edit/Delete Papers
- Add/Edit/Delete Semesters
- Add/Edit/Delete Timings
- Add/Edit/Delete Timetable
- Feedback Screen

**Total Inputs:** 11

**External Outputs (EO)**

Outputs are processed data or reports displayed by the system.

- Timetable Display (Selected via Dropdowns)
- Vacant Room Search Results
- Offline Saved Timetable View
- Admin Dashboard (Aggregated Data View)

**Total Outputs:** 4

**External Inquiries (EQ)**

Interactive queries that retrieve data without modifying it.

- Vacant Room Inquiry
- Dropdown-Based Timetable Search
- Offline Saved Timetable Search

**Total Inquiries:** 3

**Internal Logical Files (ILF)**

Logical storage managed by the system.

- Courses Database
- Faculties Database
- Rooms Database
- Sections Database
- Semesters Database
- Timetable Database
- Feedback Database

**Total ILFs:** 7

**External Interface Files (EIF)**

External files or databases accessed by the system but not maintained by it.

- None explicitly mentioned

**Total EIFs:** 0

### 7.1.2 Complexity Weighting

Assign weights to each component based on its complexity.

| Component | Simple | Average | Complex | Count | Weight | Total Weight |
|---|---|---|---|---|---|---|
| External Inputs (EI) | 3 | 4 | 6 | 11 | 4 | 44 |
| External Outputs (EO) | 4 | 5 | 7 | 4 | 5 | 20 |
| External Inquiries (EQ) | 3 | 4 | 6 | 3 | 4 | 12 |
| Internal Logical Files (ILF) | 7 | 10 | 15 | 7 | 10 | 70 |
| External Interface Files (EIF) | 5 | 7 | 10 | 0 | 0 | 0 |

Table 7.3: Complexity Weighting

### 7.1.3 Function Point Calculation

**Unadjusted Function Points (UFP)**

$$\text{UFP} = \text{Total Weight of all components} = 44 + 20 + 12 + 70 + 0 = 146$$

**Adjustment Factor (AF)**

Use the same adjustment factor formula:

$$\text{AF} = (\text{Sum of 14 complexity factors} \times 0.01) + 0.65$$

Assuming average complexity:

$$\text{AF} = (3 \times 14 \times 0.01) + 0.65 = 1.07$$

**Adjusted Function Points (AFP)**

$$\text{AFP} = \text{UFP} \times \text{AF} = 146 \times 1.07 = 156.22 \approx 156$$

**Final Function Point Estimate:** 156

# Gantt Chart

## 8.1 Gantt Chart

### 8.1.1 Project Timeline

The timeline is divided into phases with major milestones and approximate task durations.

| ID | Phase/Task | Start Date | End Date | Duration | Completion (%) |
|---|---|---|---|---|---|
| 1 | Phase 1: Initiation | 2024-02-19 | 2024-03-07 | 14 days | 100 |
| 2 | Requirement Gathering | 2024-02-19 | 2024-02-27 | 7 days | 100 |
| 3 | Feasibility Analysis | 2024-02-28 | 2024-03-07 | 7 days | 100 |
| 4 | Phase 2: Planning | 2024-02-28 | 2024-03-07 | 7 days | 100 |
| 5 | Task Allocation | 2024-02-28 | 2024-02-28 | 1 day | 100 |
| 6 | Timeline Planning | 2024-02-28 | 2024-02-28 | 1 day | 100 |
| 7 | Risk Analysis | 2024-02-28 | 2024-03-07 | 7 days | 100 |
| 8 | Phase 3: Design | 2024-03-08 | 2024-03-27 | 14 days | 100 |
| 9 | System Architecture Design | 2024-03-08 | 2024-03-18 | 7 days | 100 |
| 10 | UI/UX Design | 2024-03-19 | 2024-03-27 | 7 days | 100 |
| 11 | Phase 4: Development | 2024-03-28 | 2024-05-20 | 38 days | 100 |
| 12 | Backend Development (APIs, Database Setup) | 2024-03-28 | 2024-04-16 | 14 days | 100 |
| 13 | Frontend Development (UI Implementation) | 2024-04-17 | 2024-04-25 | 7 days | 100 |
| 14 | Integration of Backend and Frontend | 2024-04-26 | 2024-05-01 | 4 days | 100 |
| 15 | Data Population | 2024-05-01 | 2024-05-20 | 14 days | 100 |
| 16 | Phase 5: Testing & Deployment | 2024-05-20 | 2024-06-14 | 20 days | 100 |
| 17 | Unit Testing | 2024-05-20 | 2024-05-27 | 6 days | 100 |

| | | | | | |
|---|---|---|---|---|---|
| 18 | System Testing | 2024-05-28 | 2024-06-04 | 6 days | 100 |
| 19 | User Acceptance Testing (UAT) | 2024-06-04 | 2024-06-11 | 6 days | 100 |
| 20 | Deployment on Production | 2024-06-12 | 2024-06-14 | 3 days | 100 |
| 21 | Maintenance | 2024-03-28 | Till Now | – | – |
| 22 | Bug Fixes | 2024-03-28 | Till Now | – | – |
| 23 | Monitoring | 2024-03-28 | Till Now | – | – |
| 24 | Updates | 2024-03-28 | Till Now | – | – |

Table 8.4: Project Timeline

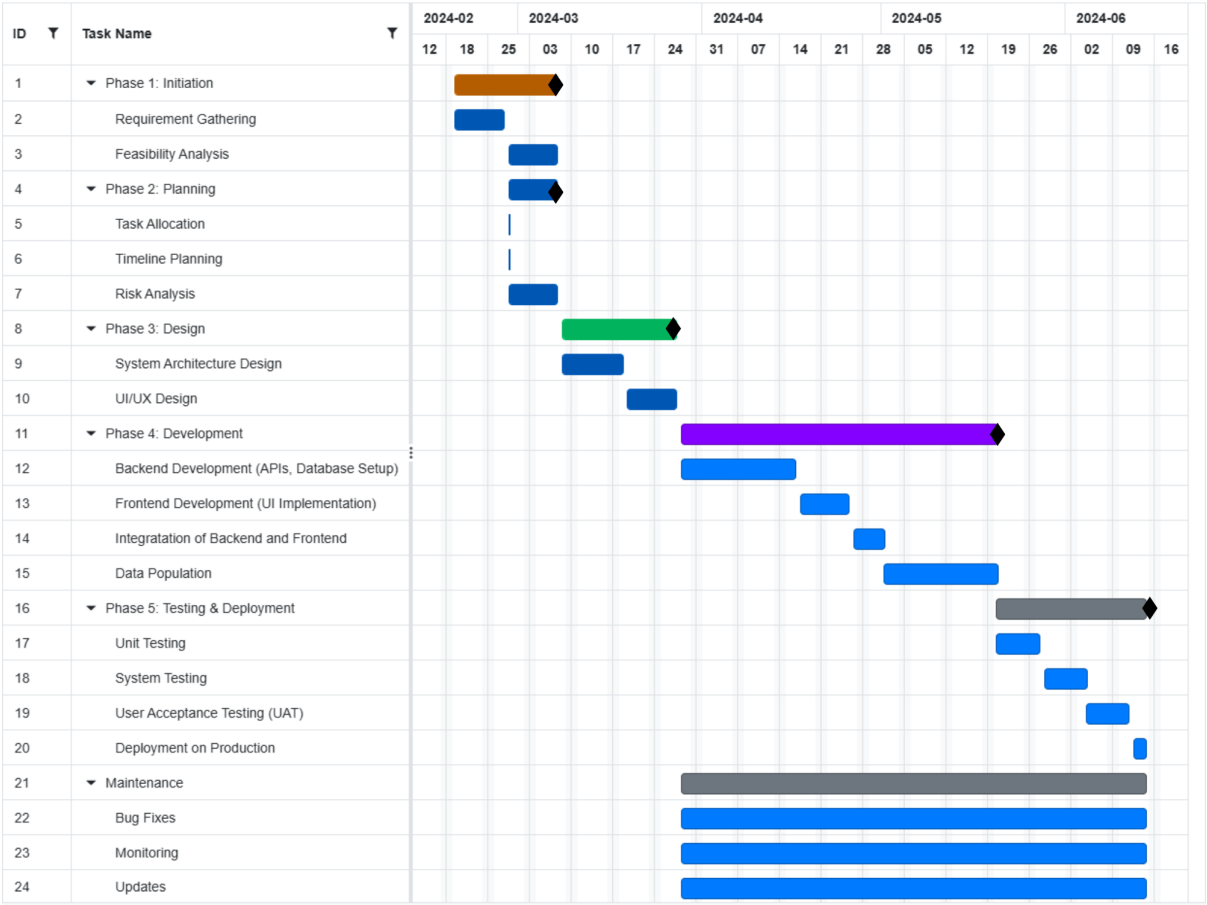


Figure 8.8: Gantt Chart For Timeline

### 8.1.2 Task Dependencies

The following dependencies indicate which tasks must be completed before others can begin:

| **Task** | **Dependent On** |
|---|---|
| Requirement Gathering | None |
| Feasibility Analysis | Requirement Gathering |

| | |
|---|---|
| Task Allocation and Risk Analysis | Feasibility Analysis |
| System Architecture Design | Planning Completion (Task Allocation) |
| UI/UX Design | System Architecture Design |
| Backend Development | System Architecture Design |
| Frontend Development | UI/UX Design |
| Integration of Backend and Frontend | Backend Development, Frontend Development |
| Unit Testing | Integration Completion |
| System Testing | Unit Testing |
| User Acceptance Testing | System Testing |
| Deployment | User Acceptance Testing |
| Maintenance | Deployment |

Table 8.5: Task Dependencies

# Testing

## 9.1 Unit Testing Overview

Unit testing is the process of testing individual units or components of the system in isolation to verify that they function as expected. A unit is the smallest testable part of the software, and unit tests aim to ensure that each unit is free of errors. Unit testing is typically performed by developers and helps catch bugs early in the development process.

### 9.1.1 Black Box Testing in Unit Testing

Black box testing focuses solely on the functionality of the unit by evaluating its outputs for given inputs, without considering the internal implementation. This approach is essential for validating the behavior of the unit based on its specified requirements.

- **Test Case 1: Timetable Saving Functionality**
  - **Input:** Timetable details (e.g., class schedules, timings, room numbers)
  - **Expected Output:** A confirmation message indicating successful saving of the timetable
- **Test Case 2: Admin Login Validation**
  - **Input:** Admin username and password
  - **Expected Output:** A boolean value indicating whether the login was successful or not
- **Test Case 4: Room Allocation Validation**
  - **Input:** Room number, date, and time range (e.g., Room 202, 11:00 AM to 1:00 PM)
  - **Expected Output:** A boolean value indicating whether the room is available or not during the given time range

### 9.1.2 White Box Testing in Unit Testing

In contrast to black box testing, white box testing examines the internal structure, logic, and implementation of the unit. It focuses on testing the logic of the code itself, ensuring that all conditions, branches, and loops function correctly.

**Test Case 1: Saving Timetable Locally**

- **Description:** Verifying the function that saves timetable data and associated details in local storage.
- **Expected Output:** The function should successfully save the timetable and details in local storage and update the count of saved timetables.

**Flow Graph 1: The Flow Graph for Saving Timetable Locally**

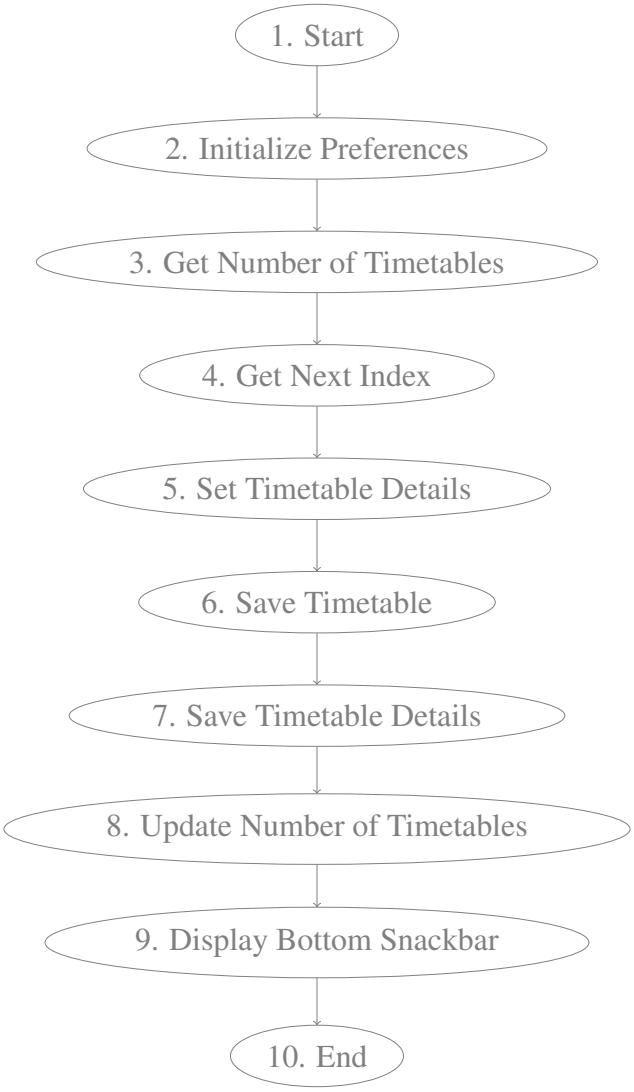


Figure 9.9: Flow Graph for Saving Timetable Locally

**Code Snippet for Saving Timetable Locally:**

```
void saveTimetableLocally(Map<String, dynamic> timetable) async {
  final SharedPreferences prefs = await SharedPreferences.getInstance();
  int numberOfTimetables = prefs.getInt('numberOfTimetables') ?? 0;
  int nextIndex = numberOfTimetables;
  Map<String, dynamic> timetableDetails = {    'course': widget.course,
     'section': widget.section,    'semester': widget.semester,
     'faculty': widget.faculty,    'room': widget.room  };
  prefs.setString('timetable_$nextIndex', jsonEncode(timetable));
```

```
prefs.setString('timetable_${nextIndex}_details',
    jsonEncode(timetableDetails));
  prefs.setInt('numberOfTimetables', nextIndex + 1);
  bottomSnackbar();
}
```

Listing 9.1: Code for Saving Timetable Locally

**Cyclomatic Complexity of `saveTimetableLocally()`** Cyclomatic complexity measures the structural complexity of a function, providing insight into its maintainability and testability. It can be calculated using three methods: *Code Analysis (Decision Points)*, *Flow Graph Analysis*, and *Binary Decision Criterion*.

### 1. Cyclomatic Complexity Through Code Analysis

Cyclomatic complexity can be calculated by identifying the decision points (e.g., `if`, `for`, `while`, `switch-case`) in the code.

$$\text{CC} = D + 1$$

where:

- $D$: Number of decision points.

**Decision Points:** The code does not include any explicit decision points (`if`, `for`, etc.), as the `??` operator is not a branching construct.

$$\text{CC} = 0 + 1 = 1$$

**Conclusion:** The cyclomatic complexity of the function is 1, indicating simple, linear logic.

### 2. Cyclomatic Complexity Through Flow Graph Analysis

In this method, cyclomatic complexity is derived from the flow graph representation of the function.

$$\text{CC} = E - N + 2P$$

where:

- $E$: Number of edges in the flow graph.
- $N$: Number of nodes in the flow graph.
- $P$: Number of connected components (typically $P = 1$).

From the graph:

- $N = 10$ (nodes)
- $E = 9$ (edges)
- $P = 1$ (connected components)

$$\text{CC} = 9 - 10 + 2(1) = 1$$

**Conclusion:** The cyclomatic complexity is 1, consistent with the simplicity of the function.

### 3. Cyclomatic Complexity Through no. of regions

This method calculates complexity by counting numbers of regions.

$$\text{CC} = \text{no. of regions}$$

**Analysis:** The function contains no explicit binary decisions (`if`, `for`, etc.).

$$\text{CC} = 1$$

**Final Analysis:**

- Across all methods, the cyclomatic complexity of `saveTimetableLocally()` is 1.
- This indicates a simple function with no branching or decision-making logic.
- The function is easy to test, maintain, and extend as it exhibits linear control flow.

**Test Case 2: Fetching Timetable**

- **Description:** Verifying the function that fetches timetable data from local storage or Firebase Firestore based on the user's input criteria (e.g., room, course, semester, or faculty).
- **Expected Output:** The function should return a timetable object containing the relevant data for the specified criteria, such as room availability or course schedule.

```
Future<Map<String, dynamic>> fetchTimeTable() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  String? timetableString =
      prefs.getString('timetable_${widget.timetableId}');
  FirebaseFirestore db = FirebaseFirestore.instance;
  CollectionReference timetableRef = db.collection('Timetables');
  if (widget.bookmarkedtimetable) {
    timetable.addAll(jsonDecode(timetableString!));
  } else {
    if (widget.room != null) {
      QuerySnapshot querySnapshot = await timetableRef.get();
      for (var day in ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
          'Friday', 'Saturday']) {
        Map<String, dynamic> dayData = {};
        for (var doc in querySnapshot.docs) {
          QuerySnapshot daySnapshot = await
              timetableRef.doc(doc.id).collection(day).where(Filter.or(
            Filter('Room Number', isEqualTo: widget.room),
                Filter('Room Number2', isEqualTo: widget.room),
                Filter('Room Number3', isEqualTo: widget.room),
                Filter('Room Number4', isEqualTo: widget.room),
                Filter('Room Number_2', isEqualTo: widget.room),
                Filter('Room Number_3', isEqualTo: widget.room),
                )).get();
          for (var dayDoc in daySnapshot.docs) {
            dayData[dayDoc.id] = dayDoc.data();
            dayData[dayDoc.id]['Course'] = doc.id;
          }
```

```
        }
        timetable[day] = dayData;
      }
    } else if (widget.course != null && widget.semester != null &&
        widget.section != null) {
      for (var day in ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
          'Friday', 'Saturday']) {
        String? courseCode = widget.course == 'GE' ? 'GE' : widget.course
            == 'SEC' ? 'SEC' : widget.course == 'VAC' ? 'VAC' :
            widget.course;
        QuerySnapshot querySnapshot = await timetableRef.doc('$courseCode
            - ${widget.section} -
            ${widget.semester}').collection(day).get();
        Map<String, dynamic> dayData = {};
        for (var doc in querySnapshot.docs) {
          dayData[doc.id] = doc.data();
        }
        timetable[day] = dayData;
      }
    } else if (widget.faculty != null) {
      QuerySnapshot querySnapshot = await timetableRef.get();
      for (var day in ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
          'Friday', 'Saturday']) {
        Map<String, dynamic> dayData = {};
        for (var doc in querySnapshot.docs) {
          QuerySnapshot daySnapshot = await
              timetableRef.doc(doc.id).collection(day).where(Filter.or(
            Filter('Faculty 1', isEqualTo:
                getFacultyNameByCode(widget.faculty)),
                Filter('Faculty 2', isEqualTo:
                getFacultyNameByCode(widget.faculty)),
                Filter('Faculty 3', isEqualTo:
                getFacultyNameByCode(widget.faculty)),
                Filter('Faculty 4', isEqualTo:
                getFacultyNameByCode(widget.faculty)),
                Filter('Faculty 1_2', isEqualTo:
                getFacultyNameByCode(widget.faculty)),
                Filter('Faculty 1_3', isEqualTo:
                getFacultyNameByCode(widget.faculty)),          )).get();
          for (var dayDoc in daySnapshot.docs) {
            dayData[dayDoc.id] = dayDoc.data();
            dayData[dayDoc.id]['Course'] = doc.id;
          }
        }
        timetable[day] = dayData;
      }
    }
  }
  return timetable;
}
```

Listing 9.2: Code for Fetching Timetable

```
Function fetchTimeTable():
  Initialize SharedPreferences instance
  Retrieve timetable data from SharedPreferences using timetableId
  Initialize FirebaseFirestore instance
  Get the 'Timetables' collection reference from Firestore
```

```
If the timetable is bookmarked:
  Parse timetable data from the SharedPreferences
  Add parsed timetable data to the timetable variable
Else:
  If room information is provided:
    Get all documents from the 'Timetables' collection
    For each day (Monday to Saturday):
      Initialize an empty dictionary for that day's data
      For each document in the collection:
        Query the day collection for matching room numbers
        For each day document:
          Add the day data to the day's dictionary
          Add the course identifier to the day's data
      Assign the day's data to the timetable for that day
  Else if course, semester, and section are provided:
    For each day (Monday to Saturday):
      Determine the course code based on the course type
      Get the documents from the specific course-section-semester
          collection for that day
      For each document:
        Add the day's data to the day's dictionary
      Assign the day's data to the timetable for that day
  Else if faculty is provided:
    Get all documents from the 'Timetables' collection
    For each day (Monday to Saturday):
      Initialize an empty dictionary for that day's data
      For each document in the collection:
        Query the day collection for matching faculty members
        For each day document:
          Add the day data to the day's dictionary
          Add the course identifier to the day's data
      Assign the day's data to the timetable for that day
Return the final timetable
```

# Cyclomatic Complexity Calculation

The cyclomatic complexity ($CC$) of a function can be calculated using the following formula:

$$CC = D + 1$$

where:

- $D$ is the number of decision nodes.

For the `fetchTimeTable` function, we identified:

- 4 (Decision nodes, corresponding to the `if` conditions)
- 8 (Loops, corresponding to the `for` loops)

Thus, the cyclomatic complexity is:

$$CC = 4 + 8 + 1 = 13$$

Therefore, the cyclomatic complexity of the `fetchTimeTable` function is $\boxed{13}$.

## 9.2 System Testing Overview

System testing is the process of verifying the complete system's functionality, performance, and behavior against the specified requirements. Unlike unit testing, which focuses on individual components, system testing evaluates the overall system's integration and interactions. It ensures that the entire system works together as intended and meets the specified requirements.

### 9.2.1 Black Box Testing in System Testing

Black box testing in system testing involves testing the system as a whole, focusing on the output generated by the system for a given input, without considering the internal logic of the system. The goal is to ensure that the system meets the functional requirements and produces the expected results when interacting with the user and other systems.

- **Test Case 1: Room Availability Query**
  - **Input:** day, and time range
  - **Expected Output:** A list of available rooms during the given time slot.
- **Test Case 2: Admin Login Validation**
  - **Input:** Admin credentials (username and password)
  - **Expected Output:** If valid, the admin should gain access to the dashboard; if invalid, an error message should be shown.
- **Test Case 3: Faculty Timetable View**
  - **Input:** Faculty ID
  - **Expected Output:** A list of the timetable entries for the given faculty, showing courses, rooms, and timings.
- **Test Case 4: Student Course Timetable View**
  - **Input:** course, section and semester
  - **Expected Output:** The system should display the student's timetable, including courses and room numbers for the specified semester.

Black box testing in system testing focuses on ensuring the system meets user expectations and requirements by validating its external functionality and behavior.

### 9.2.2 White Box Testing in System Testing

White box testing in system testing involves testing the internal logic and components of the integrated system. It aims to ensure that all the system's internal processes work as expected and that data flows correctly between modules. It also verifies that the system can handle various edge cases and exceptional conditions.

- **Test Case 1: Backend Logic for Room Availability**
  - **Description:** Testing the system's backend logic for getting list of available rooms.

  - **Expected Output:** The system should process the request correctly and return accurate availability for the room at the requested time.

- **Test Case 2: Data Integrity Between Modules**
  - **Description:** Ensuring that data entered into one module (e.g., rooms, courses) is correctly passed to and used by other modules (e.g., timetables).
  - **Expected Output:** All modules should correctly receive, process, and display relevant data without any inconsistency.

- **Test Case 3: Authentication and Authorization Logic**
  - **Description:** Verifying that the system properly handles user authentication and authorization for accessing different parts of the system (e.g., admin, faculty, student).
  - **Expected Output:** The system should grant access based on user roles and prevent unauthorized access.

- **Test Case 4: Data Flow Consistency**
  - **Description:** Ensuring that data flows consistently between modules, for instance, between faculty, rooms, and course schedules.
  - **Expected Output:** All modules should correctly pass and receive data for each transaction without errors or miscommunication.

White box testing in system testing focuses on validating the correctness of internal logic, ensuring all components function as expected, and verifying that the system processes data correctly at every step.

## 9.3 User Acceptance Testing (UAT) Overview

User Acceptance Testing (UAT) is the final phase of testing, where the system is tested from the user's perspective. The goal of UAT is to ensure that the software meets the business needs, requirements, and expectations of the users. UAT is typically performed by the end-users or clients, who validate that the system works as expected in real-world scenarios before it is deployed into production.

### 9.3.1 Black Box Testing in UAT

Black box testing in UAT is primarily focused on validating the system's overall functionality from the user's perspective. This includes testing all features and ensuring they work as expected, without any concern for how the system is implemented.

- **Test Case 1: Room Availability Query**
  - **Input:** Day and time range
  - **Expected Output:** A list of available rooms during the given time slot.

- **Test Case 2: Admin Login Validation**

  - **Input:** Admin credentials (username and password)
  - **Expected Output:** If valid, the admin should gain access to the dashboard; if invalid, an error message should be shown.

- **Test Case 3: Faculty Timetable View**
  - **Input:** Faculty ID
  - **Expected Output:** A list of the timetable entries for the given faculty, showing courses, rooms, and timings.

- **Test Case 4: Student Course Timetable View**
  - **Input:** Course, section, and semester
  - **Expected Output:** The system should display the student's timetable, including courses and room numbers for the specified semester.

In black box testing for UAT, the focus is on verifying that all features and functionality meet the users' needs and that the system provides expected outcomes based on user inputs.

### 9.3.2 Alpha Testing

Alpha Testing is the initial phase of testing where the development team tests the system for bugs, issues, and functionality. It focuses on identifying defects early before the software is released to external users.

- **Test Case 1: System Performance under Load**
  - **Description:** Test the system's performance under heavy load conditions.
  - **Expected Outcome:** The system should continue to operate without significant performance degradation or crashes.

- **Test Case 2: Integration of Components**
  - **Description:** Ensure that different modules (e.g., room availability, timetable view) are properly integrated.
  - **Expected Outcome:** The system should process data correctly and pass it between modules without any errors.

- **Test Case 3: Usability Testing**
  - **Description:** Ensure the user interface is intuitive and user-friendly.
  - **Expected Outcome:** The system should be easy to navigate, and all necessary functions should be accessible.

Alpha Testing is typically a combination of both black box and white box testing. While most of the testing is focused on user-facing features (black box), developers may also conduct white box testing to ensure that the internal code works as intended.

### 9.3.3 Beta Testing

Beta Testing is the phase where a larger group of external users tests the system in real-world scenarios. The goal is to identify any bugs or usability issues that were missed during Alpha Testing and to gather feedback from users.

- **Test Case 1: Real-world Usability Test**
  - **Description:** Allow real users to test the system in their actual work environment.
  - **Expected Outcome:** Collect feedback on usability, bugs, and any issues that affect user experience.
- **Test Case 2: System Compatibility**
  - **Description:** Test the system on different devices, operating systems, and browsers.
  - **Expected Outcome:** The system should function consistently across all platforms.
- **Test Case 3: Error and Bug Reporting**
  - **Description:** External users report any bugs or issues encountered during usage.
  - **Expected Outcome:** The system should handle errors gracefully, and any issues reported should be addressed by the development team.

Beta Testing is primarily black box testing since it focuses on the system's external functionality and user experience, with external users testing the system as they would in the real world.

### 9.3.4 Summary

Both Alpha and Beta Testing are critical for ensuring that the system works as expected and meets user needs. Alpha Testing focuses on internal testing by the development team, often involving both black box and white box techniques, while Beta Testing involves real users and is focused on black box testing to identify usability issues and bugs in real-world scenarios.

# Risk

## 10.1 Risk Table

In this section, we outline the potential risks associated with the project and the corresponding mitigation strategies. These risks are identified during different phases of the project and could affect various aspects such as schedule, resources, functionality, and quality.

### 10.1.1 Identified Risks

| Risk ID | Risk Description | Probability | Impact | Risk Category |
|---|---|---|---|---|
| 1 | Delay in Requirements Gathering | 3 | 2 | Critical |
| 2 | Lack of Adequate Resources (e.g., manpower, tools) | 3 | 3 | Catastrophic |
| 3 | Integration Issues Between Frontend and Backend | 3 | 3 | Catastrophic |
| 4 | Delays in User Acceptance Testing (UAT) | 2 | 2 | Critical |
| 5 | Bugs or Errors in Critical Features (e.g., Timetable Generation) | 2 | 3 | Catastrophic |
| 6 | User Training and Familiarization with the System | 1 | 2 | High Priority |
| 7 | Unexpected Changes in Project Scope or Requirements | 3 | 2 | Critical |
| 8 | System Performance Issues (e.g., load handling) | 2 | 2 | Critical |
| 9 | Data Loss During Migration or Backup | 2 | 3 | Catastrophic |
| 10 | Security Vulnerabilities (e.g., Data Breaches) | 3 | 3 | Catastrophic |

Table 10.6: Identified Risks for the Project

### 10.1.2 Mitigation Strategies

For each identified risk, appropriate mitigation strategies are outlined to minimize the impact on the project and ensure the smooth progress of the development process. The strategies are as follows:

- **Risk 1: Delay in Requirements Gathering**

  - **Probability:** 3 (High likelihood of occurring due to dependencies on external stakeholders for timely input.)
  - **Impact:** 2 (Moderate impact, as delayed requirements will push other phases forward but won't stop the entire project.)
  - **Mitigation Strategy:** Set clear deadlines for requirements gathering, prioritize critical features, and involve key stakeholders to ensure timely feedback.

- **Risk 2: Lack of Adequate Resources (e.g., manpower, tools)**
  - **Probability:** 3 (Likely to occur due to the finite number of developers and potential unforeseen demands.)
  - **Impact:** 3 (High impact, as it could significantly delay development and cause quality issues.)
  - **Mitigation Strategy:** Allocate sufficient resources in advance, ensure proper training for team members, and assess resource needs periodically throughout the project lifecycle.

- **Risk 3: Integration Issues Between Frontend and Backend**
  - **Probability:** 3 (High likelihood of problems arising when combining separate development efforts for frontend and backend.)
  - **Impact:** 3 (High impact, as integration issues can lead to significant delays and bugs.)
  - **Mitigation Strategy:** Use continuous integration practices, maintain clear communication between frontend and backend teams, and perform early integration testing to identify issues early.

- **Risk 4: Delays in User Acceptance Testing (UAT)**
  - **Probability:** 2 (Moderate likelihood, depending on user availability and involvement.)
  - **Impact:** 2 (Moderate impact, as any delays in UAT will push back the final release.)
  - **Mitigation Strategy:** Start UAT early in the development phase, allocate sufficient time for testing, and ensure active participation from end-users during UAT.

- **Risk 5: Bugs or Errors in Critical Features (e.g., Timetable Generation)**
  - **Probability:** 2 (Moderate likelihood, considering the complexity of features like timetable generation.)
  - **Impact:** 3 (High impact, as bugs in such core functionality can severely disrupt the system's operation.)
  - **Mitigation Strategy:** Perform thorough unit testing and integration testing for critical features, ensure robust error handling, and include backup processes to manage unexpected issues.

- **Risk 6: User Training and Familiarization with the System**
  - **Probability:** 1 (Low likelihood, as training is typically managed well before deployment.)

  - **Impact:** 2 (Moderate impact, as insufficient user training may cause confusion but not halt the project.)
  - **Mitigation Strategy:** Prepare clear documentation and provide training sessions for users prior to system deployment. Offer ongoing support for user queries.

- **Risk 7: Unexpected Changes in Project Scope or Requirements**
  - **Probability:** 3 (High likelihood, as scope changes can happen due to evolving business needs.)
  - **Impact:** 2 (Moderate impact, depending on the scale of the change and how well it is managed.)
  - **Mitigation Strategy:** Define a clear scope early in the project, establish change control procedures, and ensure all stakeholders approve any scope changes.

- **Risk 8: System Performance Issues (e.g., load handling)**
  - **Probability:** 2 (Moderate likelihood, as performance issues are usually revealed under load testing.)
  - **Impact:** 2 (Moderate impact, as this can affect user experience but can be fixed in later stages.)
  - **Mitigation Strategy:** Conduct performance testing under various load conditions, optimize code for performance, and scale resources based on user needs.

- **Risk 9: Data Loss During Migration or Backup**
  - **Probability:** 2 (Moderate likelihood, though data loss is rare with proper planning.)
  - **Impact:** 3 (High impact, as data loss can have catastrophic consequences for the system.)
  - **Mitigation Strategy:** Implement regular backups, use reliable data migration tools, and test backup and restore processes regularly.

- **Risk 10: Security Vulnerabilities (e.g., Data Breaches)**
  - **Probability:** 3 (High likelihood, especially with increasing cybersecurity threats.)
  - **Impact:** 3 (High impact, as security breaches can result in data loss, reputation damage, or legal implications.)
  - **Mitigation Strategy:** Perform regular security audits, implement encryption and secure authentication mechanisms, and ensure data is securely stored and transmitted.

### 10.1.3 Explanation of Probability and Impact

The **Probability** and **Impact** values are assigned based on the following scales:

- **Probability:**
  - 1: Low probability (unlikely to happen)
  - 2: Medium probability (may happen, but not common)

  - 3: High probability (likely to happen)

- **Impact:**
  - 1: Low impact (minimal disruption, easily resolved)
  - 2: Medium impact (moderate disruption, can be managed)
  - 3: High impact (significant disruption, requires urgent resolution)

### 10.1.4 Conclusion

By identifying the potential risks early in the project and assigning probabilities and impacts to each risk, we can effectively prioritize mitigation strategies to ensure that the project stays on track. Through proper planning and execution of these mitigation strategies, the risks can be reduced or eliminated, contributing to the overall success of the project.

# Feasibility Study

## 11.1 Feasibility Study

A feasibility study is a critical analysis that helps determine if a project is technically, operationally, and financially viable. It ensures that the project aligns with the company's resources, capabilities, and objectives before proceeding. The feasibility study for this project is divided into three main sections: Technical Feasibility, Operational Feasibility, and Financial Feasibility.

### 11.1.1 Technical Feasibility

Technical feasibility assesses whether the project can be developed and implemented using current technology and resources. It evaluates if the project's technical requirements can be met with the available tools, technologies, and expertise.

**Key Considerations for Technical Feasibility:**

- **Technology Stack:** The proposed project uses widely adopted technologies such as Flutter for frontend development, Firebase for backend services, and MySQL for data storage. These technologies are well-suited to the project's needs and are supported by a strong developer community.
- **Integration with Existing Systems:** The system must integrate seamlessly with existing institutional systems for managing student timetables. The APIs provided by the existing systems must be compatible with the new software, and any issues related to compatibility must be addressed during the design phase.
- **Scalability:** The system is designed to handle increasing loads, such as growing user numbers or more timetable entries. Cloud services like Firebase will provide the scalability needed for handling high traffic and large datasets.
- **Security:** As the system involves sensitive data (e.g., student schedules and academic records), technical feasibility includes implementing robust security measures such as data encryption, secure authentication, and regular security audits.
- **Development Resources:** The project team is proficient in the technologies required for this project (Flutter, Firebase, SQL), and there are adequate resources available to manage the development process.

**Conclusion for Technical Feasibility:** The technical feasibility of the project is high. The proposed technologies are reliable, and the team has the expertise to successfully implement the system. The necessary resources and tools are available to meet the project's technical requirements.

### 11.1.2 Operational Feasibility

Operational feasibility evaluates whether the project can be integrated smoothly into the organization's daily operations and if the stakeholders can operate and maintain the system effectively once it is developed.

**Key Considerations for Operational Feasibility:**

- **User Acceptance:** The system is designed to meet the needs of its primary users, including students, faculty, and administrators. User interfaces (UI) are intuitive, and feedback from stakeholders has been incorporated into the system design.
- **Ease of Use:** The system provides clear navigation and easy access to features. The admin panel allows for easy management of courses, timetables, and other data. Students can easily access their timetables through a straightforward interface.
- **Training and Support:** Adequate training will be provided to both end-users (students and faculty) and administrators. A user manual will be available, and technical support will be provided for troubleshooting. The team will ensure proper knowledge transfer during deployment.
- **System Maintenance:** The system is designed for ease of maintenance, with modular components and easy-to-understand code. Regular software updates and patches will be implemented as needed, and the system is built to handle bug fixes without significant disruption.
- **Operational Workflow:** The system is designed to integrate smoothly into the daily operations of the institution. It will help improve the management of course schedules, room assignments, and faculty availability without creating additional operational complexity.

**Conclusion for Operational Feasibility:** The system is operationally feasible. It aligns well with the organizational needs and integrates easily into existing processes. The system is user-friendly and will require minimal operational changes for implementation.

### 11.1.3 Financial Feasibility

Financial feasibility assesses whether the project is financially viable and if the required budget is available to complete the project successfully. It considers the cost of development, deployment, and maintenance, as well as any potential return on investment (ROI).

**Key Considerations for Financial Feasibility:**

- **Development Costs:** The development costs are estimated based on the team's hourly rates, the project timeline, and the required resources (e.g., cloud services, software tools, etc.). The estimated total cost of development is within the allocated budget.
- **Operational Costs:** The operational costs, including cloud hosting (e.g., Firebase), maintenance, and support, are expected to be manageable. The system will run on a cloud platform, minimizing upfront infrastructure costs while allowing for easy scaling.
- **Cost-Benefit Analysis:** The system will save the institution time and resources by automating timetable management, reducing administrative workload, and improving accuracy. This will provide long-term financial savings.

- **Return on Investment (ROI):** While the immediate ROI may not be large, the system's efficiency improvements will result in indirect benefits such as time savings, reduced errors, and increased student satisfaction, which can lead to higher retention and operational efficiency.
- **Contingency Funds:** A contingency fund of 10-15% of the total project cost is allocated to handle unforeseen expenses during development, testing, and deployment phases.

**Conclusion for Financial Feasibility:** The project is financially feasible. The estimated development and operational costs fit within the project budget. Additionally, the long-term benefits of automating and improving schedule management provide a strong financial justification for the system.

### 11.1.4 Conclusion

The feasibility study concludes that the project is technically, operationally, and financially viable. The project team has the expertise to develop the system using appropriate technologies, and the system will integrate smoothly into the institution's operations. Moreover, the project's financial costs are manageable and offer a favorable return on investment.

# User Interface Design

The following images demonstrate the design of the user interface for the Timetable Management System (TTMS). The UI has been designed to offer intuitive access to various features, including timetable viewing, room management, feedback submission, and more.

## 12.1 UI Design Images

Below are the images of the user interface design for TTMS, organized in pairs of 4 per page.

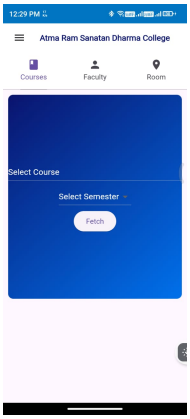


Figure 12.10: UI Design - Course Wise Timetable Screen

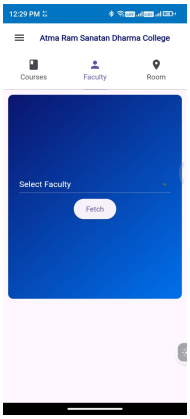


Figure 12.11: UI Design - Faculty Wise Timetable Screen

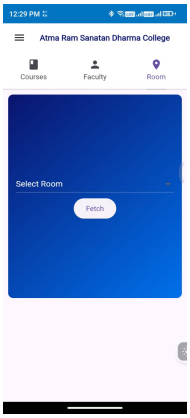


Figure 12.12: UI Design - Room Wise Timetable Screen

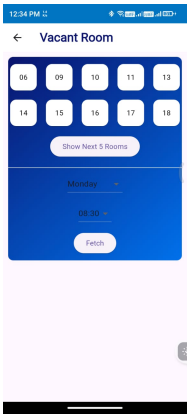


Figure 12.13: UI Design - Vacant Room Screen

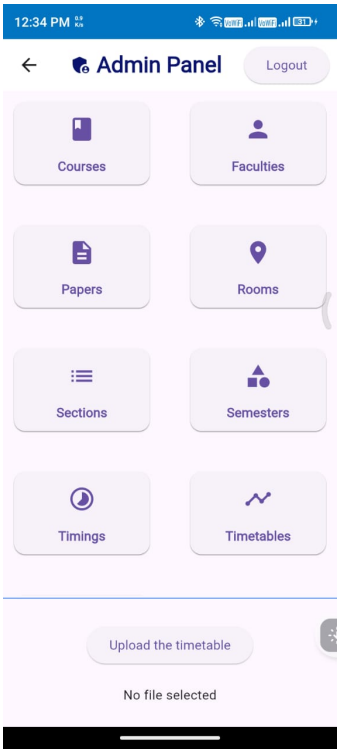


Figure 12.14: UI Design - Admin Panel Screen

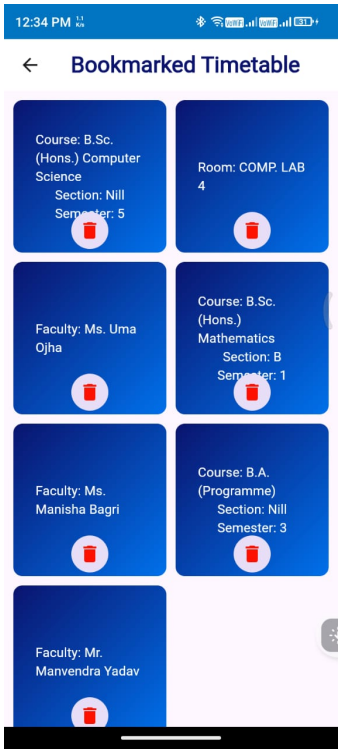


Figure 12.15: UI Design - Bookmarked Timetable Screen

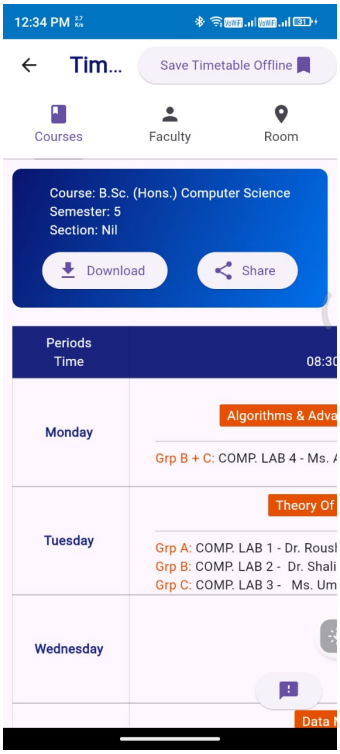


Figure 12.16: UI Design - Timetable Interface

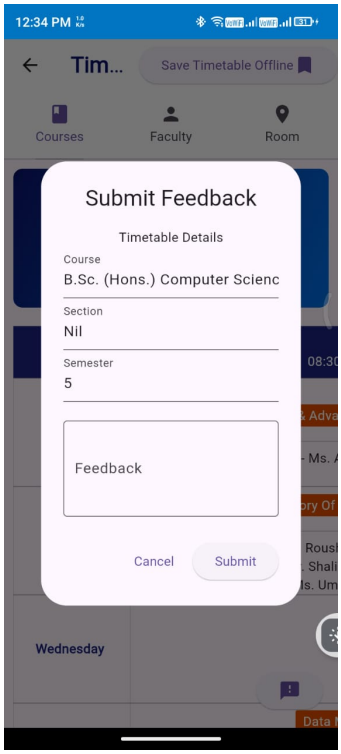


Figure 12.17: UI Design - Feedback Interface

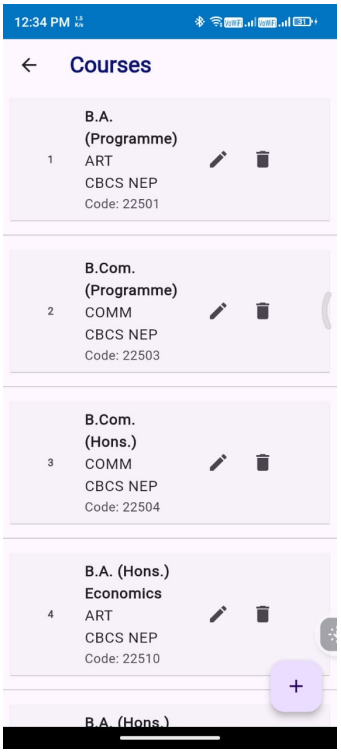


Figure 12.18: UI Design - View Course(Admin only) Screen

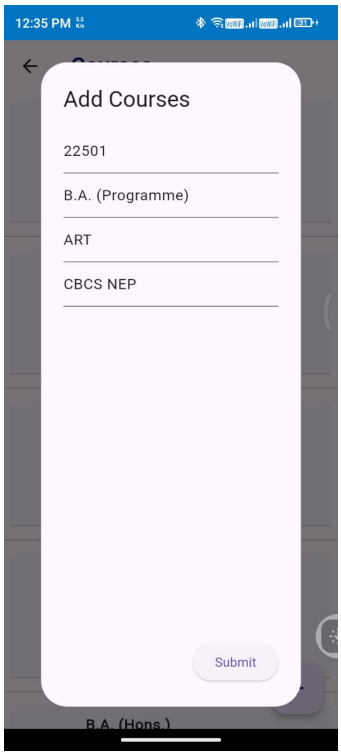


Figure 12.19: UI Design - Update/Add Course Interface

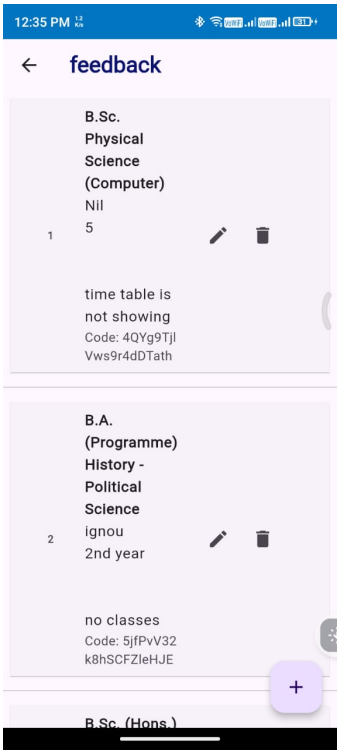


Figure 12.20: UI Design - All Feedback Screen

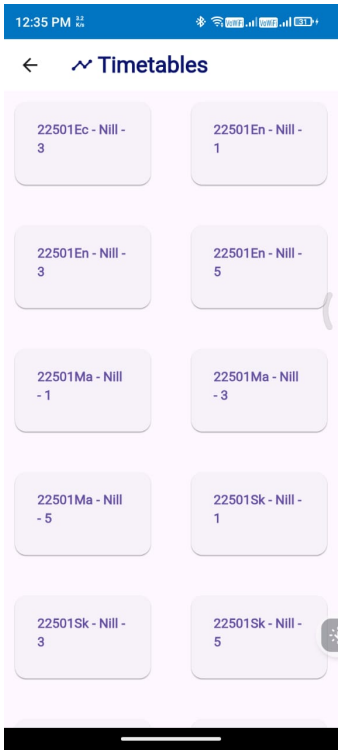


Figure 12.21: UI Design - All Added Timetables Screen

# Link to the Deployed Prototype

https://timetable.arsdcollege.ac.in/