TIMETABLE MANAGEMENT SYSTEM

EELUTTMS

Supervised By: Dr. Kamal Hamza

Teaching Assistant: Eng. Radwa Sayed

Team Members:

Student Name	ID
Mohamed Gaber Mohamed Khalil	2001857
Reham Bakri Ahmed Saad	2001624
Mostafa Rabie Mostafa Fawzy	2002053
Noran Mohamed Mostafa Ahmed	2001658
Momen Magdy Rabie Youssef	2000411
Dalia Magdy Mohamed	2001659
Ahmed Aid Mahamoud	2001731
Eman Ahmady	2001754

ABSTRACT

The Egyptian University for E-Learning was established by Presidential Decree No. 233 of 2008 as the first Egyptian university to adopt the principle of e-learning. In 2018, Republican Decree No. 71 was issued to transform the university from a private university into a national university. Creating timetables manually is a complex and time-consuming process. By automating this process with computer assisted timetable generator can save a lot of precious time of administrators who are involved in creating and managing course timetables. Since every college has its own timetabling problem, the commercially available software packages may not suit the needs of every college. Hence, we have developed a practical approach for building lecture course timetabling system, which can be customized to fit to any colleges timetabling problem. This project introduces a practical timetabling algorithm capable of taking care of both strong and weak constraints effectively, used in an automated timetabling system. So that each DR and Teaching Assistant can view their timetable once they are finalized for a given semester and send it to their students, but they cannot edit them. The system will be developed in windows environment using Spring boot, java, and My SQL three open-source technologies will combine to develop web application.

ACKNOWLEDGEMENT

We take this opportunity to express our profound gratitude and deepest regards to our devoted Supervisor Dr. Kamal Hamza for his outstanding supervision and constant encouragement throughout the work of this Project. The blessing, help, and guidance given by him from time to time shall carry us a long way in the journey of life on which we are about to take the ship. We also take this opportunity to express our gratitude to the Board of the Faculty of Information Technology for their support and efforts in providing us with all useful resources. Finally, we express our gratitude to the Almighty, our team (including ourselves), our parents, and all those who helped, supported, and encouraged us to complete the first portion of our graduation project successfully, regardless of their background in the university or the industry.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

Word	Acronyms
Egyptian E-Learning University Timetable Management System	EELUTTMS
Structured Query Language	SQL
Hypertext Markup Language	HTML
Cascading Style Sheet	CSS
JavaScript	JS
Sassy Cascading Style Sheets	SCSS
Central Processing Unit	CPU
Random Access Memory	RAM
Tera Byte	TB
Mega Byte Per Second	MBPS
Giga Byte	GB
Software as a service	SaaS
Atomicity, Consistency, Isolation, and Durability	ACID
User Interface	UI
University of Vocational Technology	UNIVOTEC
University of Vocational Technology Timetable Management System	UVTTTMS
Entity Relationship Diagram	ERD
Data Flow Diagram	DFD
Nondeterministic Polynomial-time complete	NP
Safety Instrumented System ‘it is belong to the university’	SIS

TIME PLAN

Time Plan				
	Duration	**Start**	**Finish**	**Resource Names**
• Graduation project	**271 Days**	**15/10/2023**	**12/6/2024**	
• Search	**30 Days**	15/10/2023	14/11/2023	All Team Members
• Busniness Requirements	**61 Days**			
tools installation	1 Days	15/11/2023	16/11/2023	Mostafa
fields studing	60 Days	17/11/2023	17/1/2024	All Team Members
• system analysis	**7 Days**			
DFD	3 Day	18/1/2024	20/1/2023	Dalia, Moemn
USE CASE	2 Day	21/1/2024	22/1/2024	Ahmed
USE CASE senario	2 Day	23/1/2024	24/1/2024	Moemn
• Design	**12 Days**			
ERD	3 Day	25/1/2024	27/1/2024	Mohamed
Schema	2 Day	28/1/2024	29/1/2024	Mohamed, Eman, Ahmed
Sequence Digram	2 Day	30/1/2024	31/1/2024	Ahmed
Class Digram	3 Day	1/2/2024	3/2/2024	Mostafa
System Architecture	2 Day	4/2/2024	5/2/2024	Noran, Mostafa
• programming	**130 Days**			
DataBase creation & insertion	30 Days	1/2/2024	29/2/2024	Mohamed
BackEnd Implementation	60 Days	1/3/2024	30/4/2024	Mostafa, Dalia
frontend implementation	30 Days	15/4/2024	15/5/2024	Noran, Reham
system Integration	10 Days	16/5/2024	25/5/2024	Mostafa, Noran, Ahmed
• Testing	**10 Days**			
Back End Testing	5 Days	26/5/2024	30/5/2024	Mohamed, Mostafa, Dalia
Front Testing	5 Days	1/6/2024	5/6/2024	Dalia, Noran, Moemn
• Documentation	**200 Days**	27/12/2024	10/6/2024	Reham, Mohamed
• Meetings (review)	**2 Days**	11/6/2024	12/6/2024	All team members

CHAPTER 1
INTRODUCTION

1.1 Introduction

Universities are confronted with the challenge of enhancing the quality and efficiency of instruction. The imperative for greater accountability in learning outcomes, with increased technological integration, has compelled universities to seek innovative solutions. Recognizing the need for optimizing resource utilization and advancing technology use in courses, many educational entities have adopted Timetable Management Systems to automate the process of building that. Timetabling concerns all activities about producing a schedule that must be subjective to different constraints. Timetable can be defined as the optimization of given activities, actions, or events to a set of objects in space-time matrix to satisfy a set of desirable constraints. A key factor in running an educational Centre or an academic environment is the need for a well-planned, well-throughout and clash-free timetable. Back in the days when technology was not in wide use, (lecture/sec) timetables were manually created by the academic Team.

Every university is faced with the tedious task of drawing up academic timetables that satisfies the various courses and the respective sec being offered in separate ways. The timetable development process starts when each Head of Department provides the following information to be used for timetable scheduling.

The information provides the modules with dates, time, and venues suitable in a particular semester:

- Dates for lectures & sections to be held (Lectures & sections can be scheduled between Saturday and Thursday).
- Specified time for lectures & sections (i.e., Between 8am and 5pm)
- The venue of the scheduled lectures & sections.

A timetabling problem consists of four parameters, and they are:

- T (set of time)
- R (set of available resources)
- M (set of scheduled contacts)
- C (set of constraints)

This problem assigns time and resources to the contacts in such a way that the constraints will be satisfied. In various timetabling problems, educational timetabling has been examined from a practical standpoint. So, in this chapter we will briefly discuss this process.

1.2 Overview

The process of scheduling the timetables for various academic levels and colleges proves is done manually and be a challenging and time-consuming task, exacerbated by numerous constraints Primarily, timetable generation is laborious, and secondly, the management of updates and reporting is overwhelming. Additionally, the process is susceptible to unexpected clashes and inconsistencies. The solution is to automate this process using a software system. This software aims to simplify the scheduling process, making it more user-friendly, flexible for edits and updates, and enhancing the quality of the education process. This system has been developed as a web-based system which will full fill university requirements on given timeline.

The proposed web-based system will be accessed by all levels of staff in the Technical Services Unit (TSU) and course coordinators, lecturers' demonstrators and staff directly involved in the resource allocation process of the university. EELU Timetable Management System allows creating timetable considering following constraints.

1. Lecture Timetable only Created by the Admin in Doki branch.
2. Sec Timetable Can be Created by The Admin in Doki branch, and the SubAdmin for each branch.
3. The professor can teach a maximum of [X credits] as input.
4. The Teaching Assistant has only 4 Days of work.

1.3 Motivation

Planning Timetables is one of the most complex and error-prone applications. So, the motivation behind our project at the Egyptian E-Learning University (EELU) stems from the visionary commitment to automation and technological leadership in education. Seeking a challenging endeavor, we were inspired to address a novel aspect of timetable management, applying principles of programming, software engineering, and project management learned during our academic journey. Our project serves as a practical application of knowledge, transitioning EELU from a manual, ad hoc system to an automated web-based solution. Aligned with the university's expansion plans, we aim to enhance efficiency and productivity in timetable and resource management, providing a comprehensive, web-based system that not only resolves current challenges but is also designed to seamlessly adapt to future changes. It also bridges existing gaps, supporting EELU's vision, and contributing to the evolution of a more advanced and dynamic educational management system.

1.4 Objective of the system

Timetable Management System can contribute to the smooth functioning of educational universities by optimizing the allocation of resources and enhancing the overall scheduling process, and here are some common objectives that the system aims to achieve:

1.4.1 Time Optimization:

- By using algorithms that control the time and divide it into slots.
- That will provide a well-organized timetable and Minimize gaps between classes and utilize available time.

1.4.2 Automation:

- Automate the process by just taking the inputs from the administrator and automatically the system will process all the data entered by using algorithms and create the timetable.
- As a result of that manual effort and human errors will be reduced.

1.4.3 Conflict Resolution:

- Identify and resolve scheduling conflicts, such as double-booked classrooms or faculty members.

1.4.4 Flexibility:

- Allow the administrator to add, edit, delete, or change any input and sub-admin.
- That allows easy modifications and updates to the timetable in response to changes in schedules, faculty availability, branch accessibility or other unforeseen circumstances.

1.5 Project Scope

The Timetable Management System is a comprehensive solution for universities, featuring user modules for administrators, and sub-admins with role-based access control.

The following are the functionalities of the system:

- Provides an effective and efficient constraint management (add, update, delete) Lecturer, resource, module, time slot, class (batch) management (add, update, delete)
- Provides Management Information System reporting on demand. (by date, time slot, lecturer/ resource person/ resource, module, etc.)
- Produces a fully functional timetable generator engine/ algorithm that can automatically cater to each constraint.

Technology integration is a priority, with a user-friendly web interface and potential integration with other systems. The project focuses on cost efficiency through resource optimization and scalability for future growth.

1.6 Feasibility Study of Timetable Management System

Before embarking on the commencement of any project, it is advisable to conduct a thorough feasibility study regarding the proposed system. This procedural step is instrumental in substantiating that the expected benefits align appropriately with the aggregate costs and technical capacities involved. Considering this, a dedicated feasibility study was initiated to meticulously evaluate the viability of the intended system, specifically within the context of the Timetable Management System.

1.6.1 Technical Feasibility:

- Hardware Requirements: It requires a database server to save the system database and a web server to display the website.

- Software Requirements: The system will be completely web-based using HTML, CSS, SCSS, JS, Angular, JAVA SPRING as the primary development language, with MySQL serving as the system's database management system.

- Compatibility: the system is capable of the existing technology used by EELU System as it will be added to the general system as a new feature.

1.6.2 Operational Feasibility:

- User Training: provide a training workshop to learn how to use the system as it is a multi-user system so every user should know what his tasks are.
  Main stockholders of the system will be academic staff, executive staff, and other officer-level staff Therefore, there is no need to provide deep training and every one of the above categories knows how a web-based system operates.

1.7 Software process model

The Waterfall Model stands as the inaugural process model in software development. It follows a sequential structure where each phase must reach completion before the subsequent phase commences, avoiding any overlap. This model divides the entire software development process into distinct phases, with the output of one phase serving as the input for the next one in a sequential manner. The progression through phases—Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation, and Maintenance—resembles a continuous flow akin to a waterfall. This linear and sequential progression characterizes the Waterfall Model, earning it the designation of a Linear-Sequential Life Cycle Model.

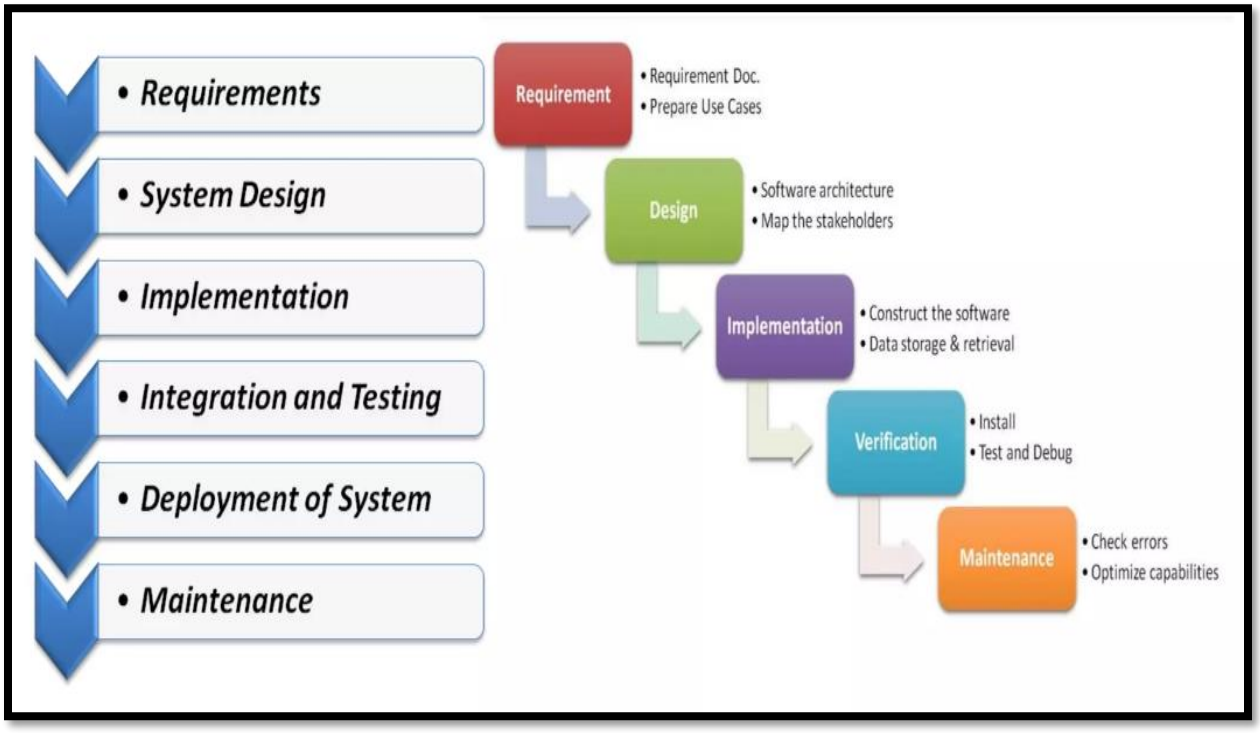

Figure 1.1 (Linear-Sequential Life Cycle Model)

CHAPTER 2

BACKGROUND

2.1 Introduction

The background chapter will provide detailed knowledge to understand the rest of the document. By giving a brief overview of the university's current situation, the type of system needed, and the concerned areas and technologies used.

2.2 Requirement Analysis

Requirement analysis sets the foundation for the entire software development process, influencing design, development, testing, and implementation phases. A well-executed requirement analysis helps mitigate risks, reduces project scope creep, and contributes to the overall success of the project by aligning the development efforts with the actual needs and expectations of stakeholders.

2.2.1 Gathering Requirements

The success of any system heavily depends on the proper identification of its major requirements and detailing them as appropriate to avoid the ambiguity of the functionality of the expected system. This could be achieved by using an appropriate set of requirements gathering techniques such as observation of the existing system, interviewing different stakeholders, carefully designed questionnaires, and implementing stage-wise small prototype versions. Therefore, almost all the mentioned techniques were used to gather the requirements of the system.

The stakeholders of the proposed system would comprise of:

a) Admin – who is the head of a university who is responsible for the management and the academic integrity of the branches and faculties.

b) Teaching Assistant - who is responsible for all resources allocation

and timetable with coordinating with staff.

Flow of current system:

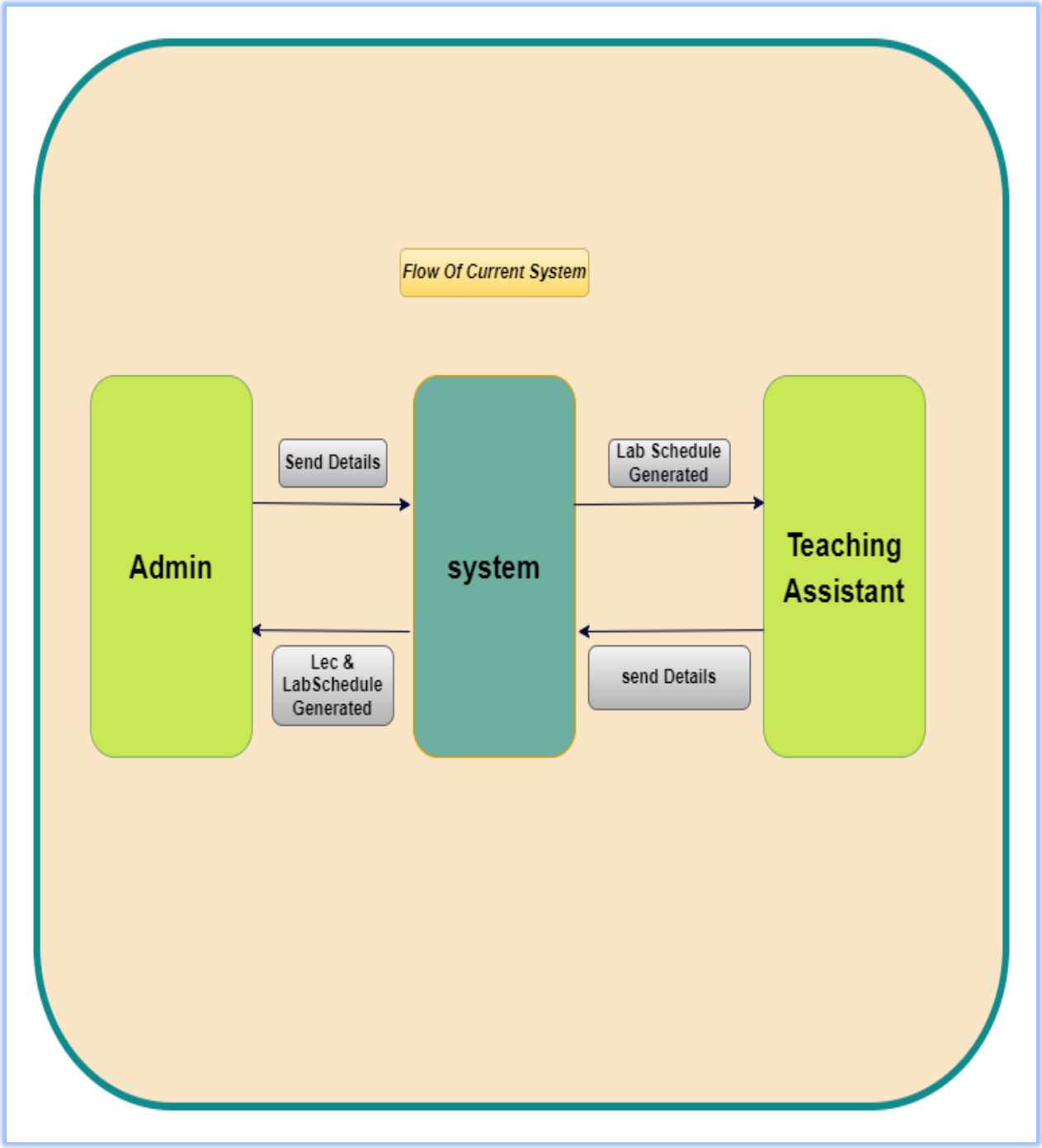

Figure 2.1 (flow of system)

2.2.2 Requirement Analysis - Project Goals

Modules of the System

The module is a different segment of the system that processes a specific task. This facilitates system development and makes it more user-friendly.

As per requirements, the system mainly contains six modules. Login module, home module, Branches Module, Faculty Module, timetable generating Module all in Nav Bar and the other module is User Profile on the other side of page contain Logout button.

Login Module:

This will assist the user in logging into the system with their username and password. Only a user with a valid username and password can access his account.

It will facilitate the authentication of the user who logs into the system. This prevents any anonymous user from entering the system and mishandling the records. It is superior to the manual method because it has no security measures governing who can and cannot access the system.

Home Module:

It contains the Nav Bar that Contain Other Modules and the workspace that facilitate to view all sections of work on this space and it the page that control all the system through it.

Branch Module:

That module is responsible for all branches as each branch contains room type and capacity for each room and No of Groups in that branch, also contain Lecturer and TA in each Branch.

Faculty Module:

It Contain the Faculties in the University and Study plan for each faculty and courses in each semester with lec / Sec hour for each course.

Timetable generating Module:

This is the module generating timetable according to the resources available and with fulfill of university constraints. In this module, the user can generate a timetable with few options. University uses maximum resources in semester, and it is essential to generate timetable with full optimization in Semester courses.

2.2.3 Functional Requirements

"Functional requirements in the context of software development refer to detailed specifications that define the specific functionalities, features, and behaviors that a system or software application must exhibit. These requirements serve as a blueprint for the development team, guiding the design, implementation, and testing processes. They articulate the desired outcomes, inputs, processes, and outputs of the system, ensuring that it meets the functional expectations of its users and stakeholders."

The following major components and their functions were recognized as the core business processes of the intended system.

• User Login

- System has two user login categories such as Admin, TA.
- Only Admin and TA for each branch have a user login.
- If Incorrect Username or Password‟ message appears, then TA need contact Admin to reset password.
- There is an accurate validate for username and password.
- Successful user login will show a welcome message and direct to home page based on the user category.

• System Home page

▪ System home page display only User Login form and help option.

▪ System identifies correct user group type and direct to appropriate pages with given user privileges.

• Admin home page

▪ This menu provides all administration task.

▪ Register all users to the system.

▪ Adding, editing, deleting entries in the master timetable.

▪ Admin can grant privileges to the other users as (Admin & SubAdmin).

• Teaching Assistant home page

▪ This menu provides only administration for each TA Branch.

▪ Responsible to free time and room for each lec that accessed from Lec Timetable.

▪ Adding, editing, deleting entries in the sections timetable.

2.2.4 Non-Functional Requirements

"Non-functional requirements may be more critical than the functional requirements. If these are not met, the system is useless" (Sommerville, 2004).

- **Performance:**

  Description: The system should respond quickly to user interactions.

  Criteria:

  - Timely generation of timetables.
  - Efficient data retrieval and processing.

- **Scalability:**

  Description: The system should manage an increasing number of users and data.

  Criteria:

  - Ability to scale with the addition of branches or workers.

- **Reliability:**

  Description: The system should be available and reliable.

  Criteria:

  - High availability.
  - Minimal downtime for maintenance.

- **Usability:**

  Description: The system should be easy to use and navigate.

  Criteria:

  - Intuitive user interface.
  - User training and support.

- **Security:**

  Description: Ensure the confidentiality and integrity of data**.**

Criteria:

- Secure user authentication.
- Role-based access control.

- **Maintainability:**

Description: The system should be easy to maintain and update**.**

Criteria:

- Modular and well-documented code.
- Easy system configuration and updates.

- **Compatibility:**

Description: The system should work across different browsers and devices**.**

Criteria:

- Compatibility with major web browsers.
- Responsive design for various screen sizes.

- **Data Backup and Recovery:**

Description: Regularly backup system data and provide a mechanism for data recovery.

Criteria:

- Scheduled data backups.
- Easy data restoration process.

- **Auditability:**

Description: The system should log user activities for auditing purposes**.**

Criteria:

- Log and monitor user actions.
- Provide audit reports when necessary.

2.2.5 User Requirements

The only end-user requirement is to have basic literacy of computer use, especially the techniques used in browsing the web.

2.2.6 System Requirements

The system architecture is such that it conforms to 3-Tier Architecture having.

a) a Backend in the bottom layer to store all data used in the system.

b) an Application Server (Web Server) in the middle layer to process requests from and result back to the user.

c) a front end (any standard web browser) provides an interface to the end-user to deal with the system.

2.2.7 The Backend (Database Server)

The database server is preferably a 2 CPU cores (recommended), 32GB or higher RAM, 1TB or higher storage space depending on the institution's data storage requirements. The backend is implemented with MySQL database management system.

2.2.8 The Application Server (Web Server)

It is observed that the performance of the webserver is one of the most vital factors pertaining to the speed of the system. Therefore, 4 CPU cores (recommended), 32GB or higher RAM, 1TB or higher storage or more free Hard disk capacity, and access to Information Superhighway at a rate of 1 Mbps is recommended.

2.2.9 Client

All that is needed by the client side is a computer that can run any standard web browser and access the Information Superhighway at a minimum rate of 1Mbps. Nevertheless, the more the performances of the computer and the network bandwidth, the faster the system processes as it would be.

2.3 Related Technologies

EELUTTMS should be accessing all branches, and therefore a web application is much more suitable for that. Users access the system via a uniform environment. It is critical to select the appropriate hardware, software, and technology while developing a better web-based system. The technology chosen as the development tools for the EELUTTMS is explained in more detail below.

2.3.1 Web Application

This is a web application. To create an application, a combination of server-side script and client-side.

Tools Used:

1. Design: Figma.
2. Coding: Visual Studio Code & NetBeans.
3. Database: MySQL.
4. Diagrams: Draw IO.

Techniques Used:

1. Front-End: HTML, CSS, SCSS, Java script, Angular
2. Back-End: Java + Spring Boot
3. Database: MySQL

2.3.2 Database Access

This is a real database that can store, update, delete and retrieve data.

EELUTTMS store all branches, faculties, study plans, courses, lectures, timeslots details. EELUTTMS will store data for many years back to analyze data.

MySQL is used as my database. MySQL is the database of choice for the most demanding Web, E-commerce, SaaS, and Online Transaction Processing (OLTP) applications. It is a transaction-safe, ACID-compliant database that supports full commit, rollback, crash recovery, and row-level locking. MySQL provides the usability, scalability, and performance required.

2.4 System Architecture

The architecture of a system reflects how the system is used and how it interacts with other systems and the outside world. It describes the interconnection of all the system's components and the data link between them. The architecture of a system reflects the way it is thought about in terms of its structure, functions, and relationships.

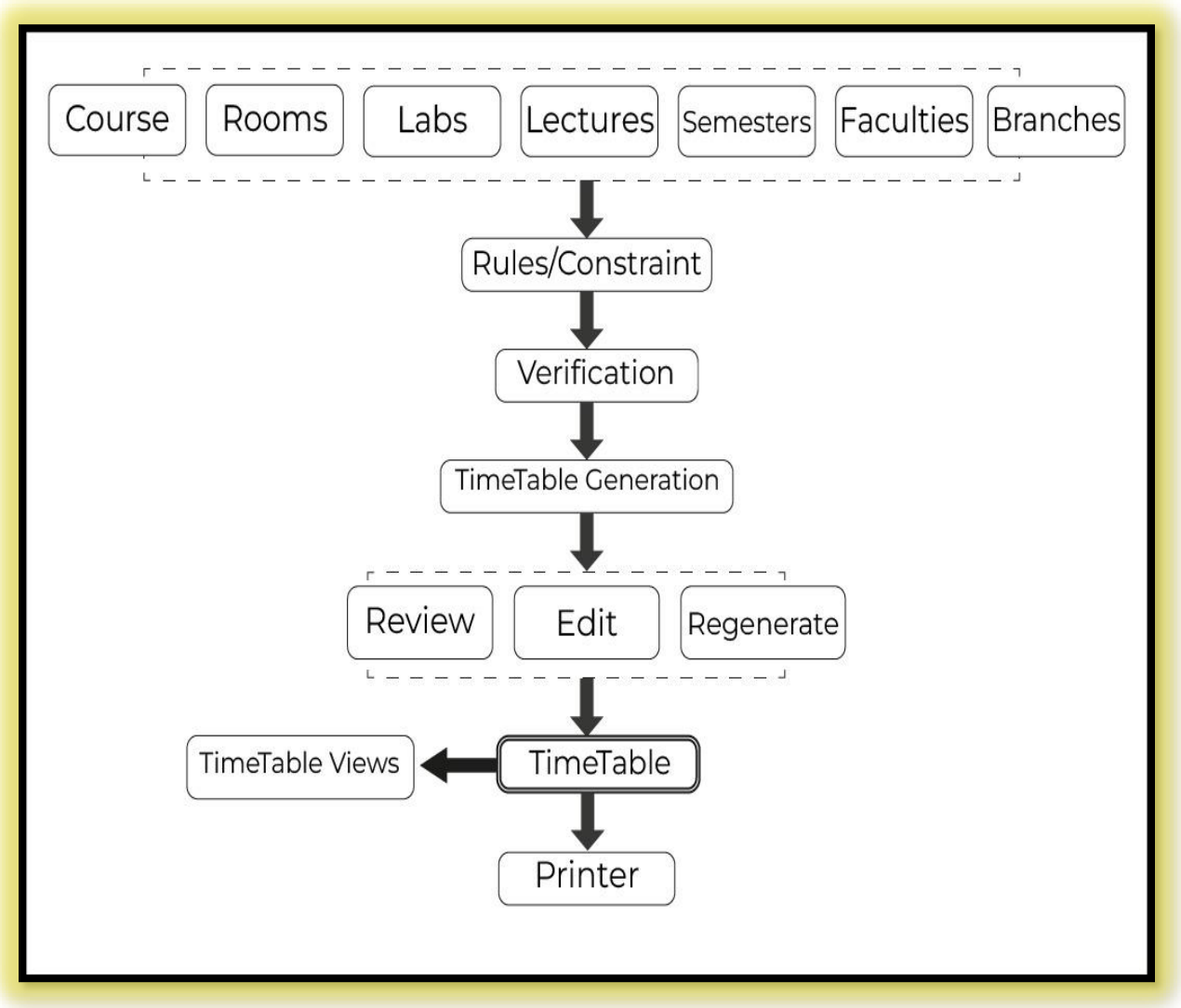

Figure 2.2 (System Architecture)

2.4.1 Client-Server Architecture

Users access the web application installed on the Application server through a web browser. Initially, the web application connects to the University Internet. Subsequently, the connection proceeds to a security layer responsible for user validation. Upon successful validation, users gain access to the Application server and MySQL database server.

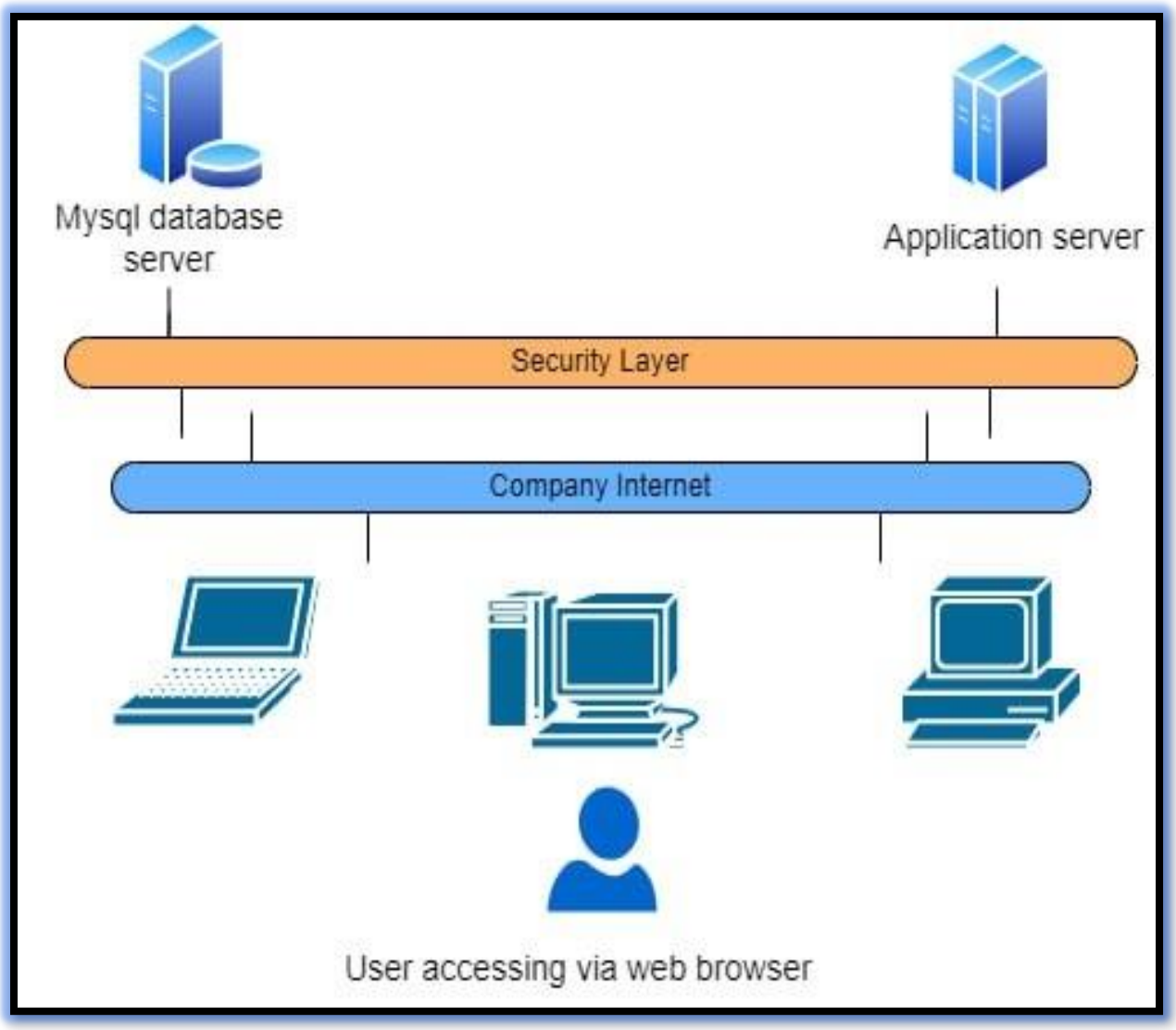

Figure 2.3(User accessing via web browser)

2.4.2 Tier Architecture

Tier architecture contains UI/ Presentation Layer (Branch), Middle tier layer and Data Access Layer.

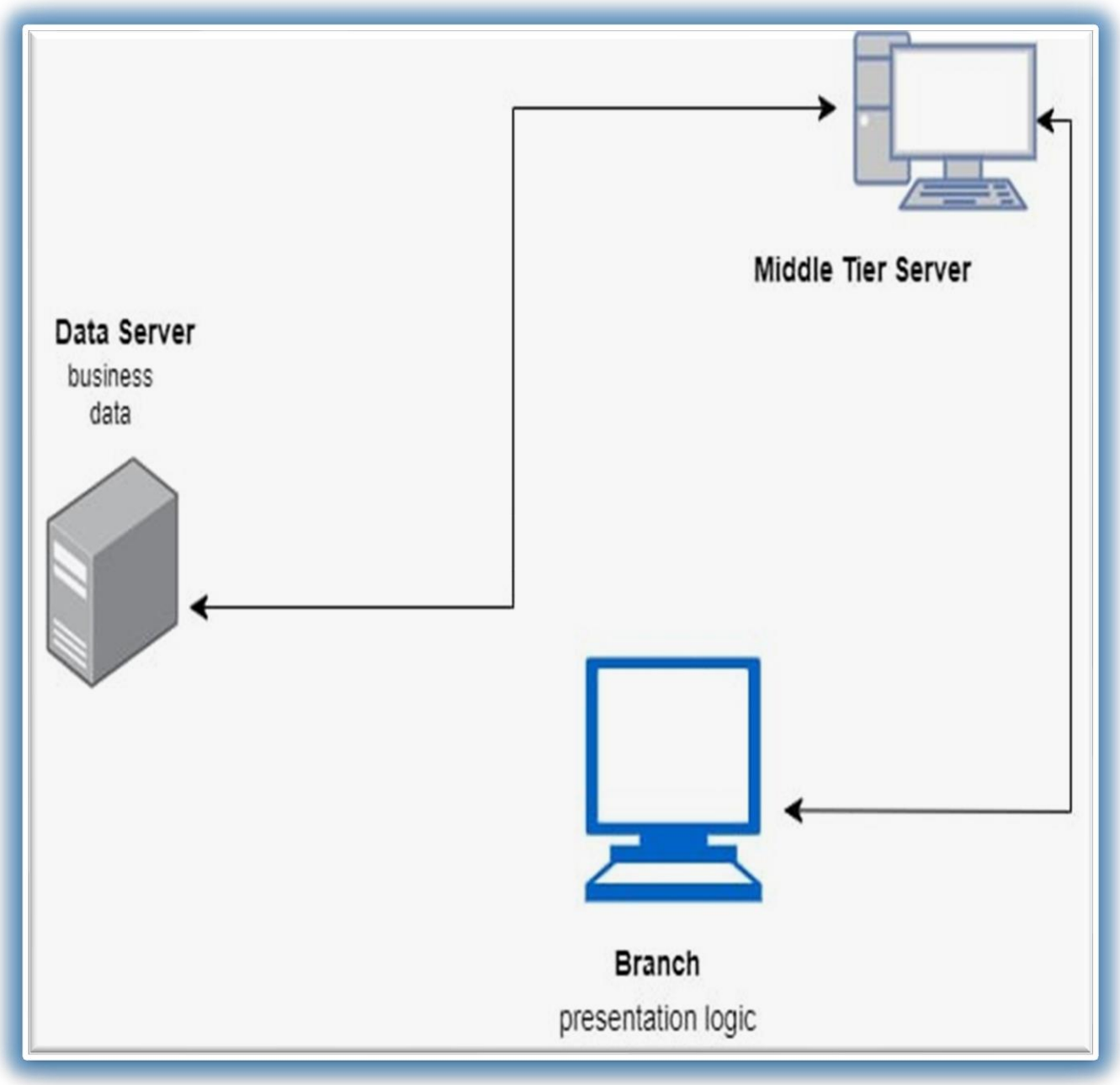

Figure 2.4 (Tier Architecture)

2.5 Existing Similar Systems

A significant amount of effort and time was expended in finding other similar systems and approaches to study their benefits and drawbacks to implement the system in an effective and efficient manner.

2.5.1 BULLET Education Suit-Automatic Timetabling Software

Bullet Education Suite timetable software offers higher education institutions a suite of software solutions designed to simplify the timetabling process. It comes with seven modules, but we will focus on the academic part in it.

- **BULLET Timetable Education**

  An automatic timetabling software capable of generating highly optimized timetables based on the interests of educational institutions. It enables you to make the most of your resources while also ensuring the satisfaction of your students and academic staff.

- **BULLET Exam Planer**

  With an automatic exam scheduling tool, you can easily accommodate all academic staff and students to their related activities, ensuring a perfect balance between students, supervisors, and classroom distribution.

Timetable created by BULLET Education Suit-Automatic Timetabling Software.

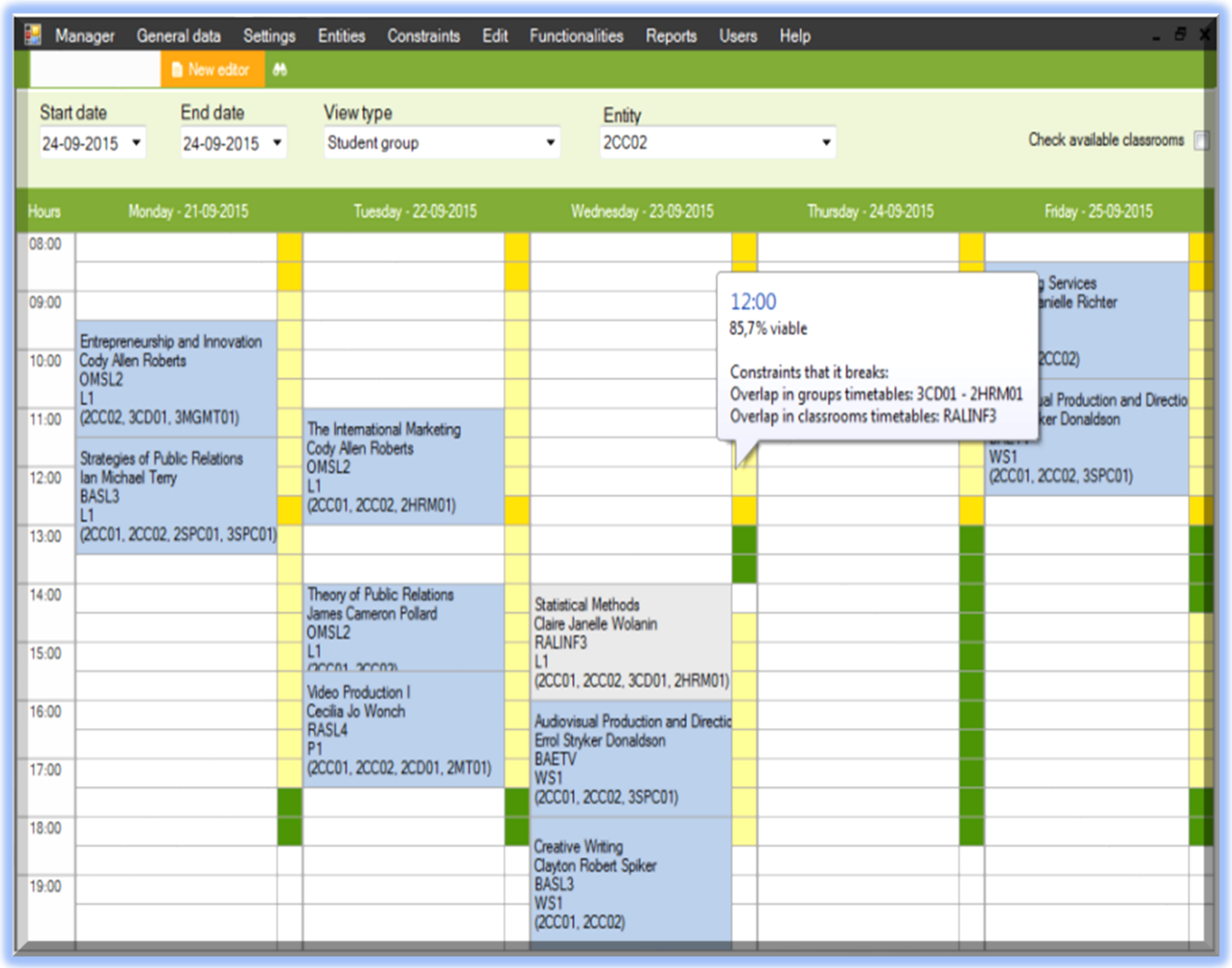

Figure 2.5 Timetable Created with BULLET Education Suit

Furthermore, the Bullet timetable solution includes numerous useful features such as fully automatic timetable generation, assisted manual adjustments, error-free timetabling, comprehensive incoherency reports, multiple optimization methods, simple online timetable publication, and full data import and export.

2.5.2 UNITIME- Timetable System

UNITIME is a comprehensive educational scheduling system that allows for the creation of course and exam timetables, the management of changes to these timetables, the sharing of rooms with other events, and the task of students to individual classes. It is a distributed system that enables multiple universities and departmental schedule managers to collaborate on developing and modifying a schedule that meets their diverse organizational needs while minimizing student course conflicts. It can be used on its own to create and maintain a school's class and/or exam schedule, or it can be integrated with an existing student information system.

The UniTime timetabling system comes with four main modules:

• Course Scheduling and Management

• Scheduling of Examinations

• Event Management

• Student Scheduling

Course Scheduling

The primary goal of course scheduling is to place each course at a time (or set of times) that does not conflict with the time(s) assigned to any other required course by the students enrolled in it. This is simple if only a few courses are taken in tandem with the course of interest. It becomes significantly more difficult as the number of courses requiring different time placements grows. The availability of faculty, rooms, and a variety of other constraints complicates matters even further.

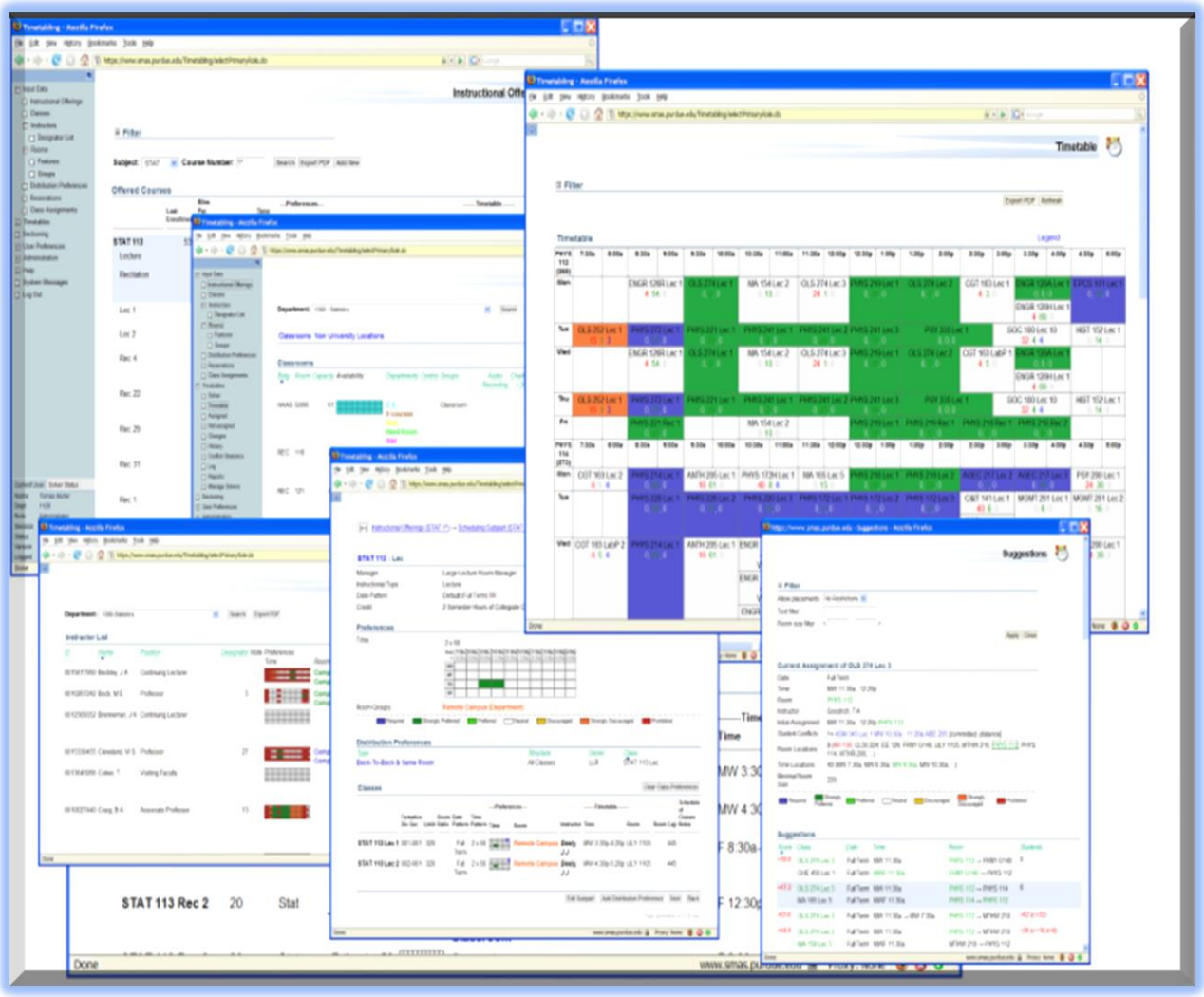

Figure 2.6 (Timetable created with UniTime Timetable Management System)

Scheduling of Examinations

Each term, UniTime creates a comprehensive exam schedule that minimizes the amount of conflicting exam placements for all students. It can also reduce the number of back-to-back tests or students who have more than a certain number of exams in a day. This is especially beneficial for schools and institutions with often changing class offerings. Or many multi-section courses that do not fit well into mapped exam timetables. UniTime can be used to generate schedules for both midterm and final exams.

2.5.3 UNIVOTEC Timetable Management System

the System able to.

1. generate the timetable based on the constraints entered by the Technical Coordinator.
2. generate summary reports for decision making (for example, workload of a specific lecturer, facility utilization, conformity checking against meeting stipulated quality requirements/ standards).

Figure 2.7 UNIVOTEC Timetable Management System

CHAPTER 3
SYSTEM ANALYSIS

3.1 Introduction

The most essential and time-consuming aspect of a project is system design. In this phase, the information gathered before is used to complete the logical design of the information system. This includes designing user interfaces, databases, and outputs in collaboration with users to satisfy their information needs. The technical or implementation component of the system development project was the emphasis of information system design. The system analysis phase of a system development project is carried out independently. The requirements from the requirement analysis phase are transformed into technological solutions by the system designer. Software architecture, database design, and interface design are all factors in system design.

3.2 Use-Case

3.2.1 Use Case Diagram

A Use Case diagram helps identify the main components and actions that make up a system. In this diagram, the main components are known as "actors," and the actions are referred to as "use cases." It illustrates how actors interact with each use case. The Use Case diagram focuses on the functional aspects of a system, specifically capturing the business processes it performs. Additionally, these diagrams define the requirements of the modeled system, serving as a basis for creating test scripts for the system being modeled.

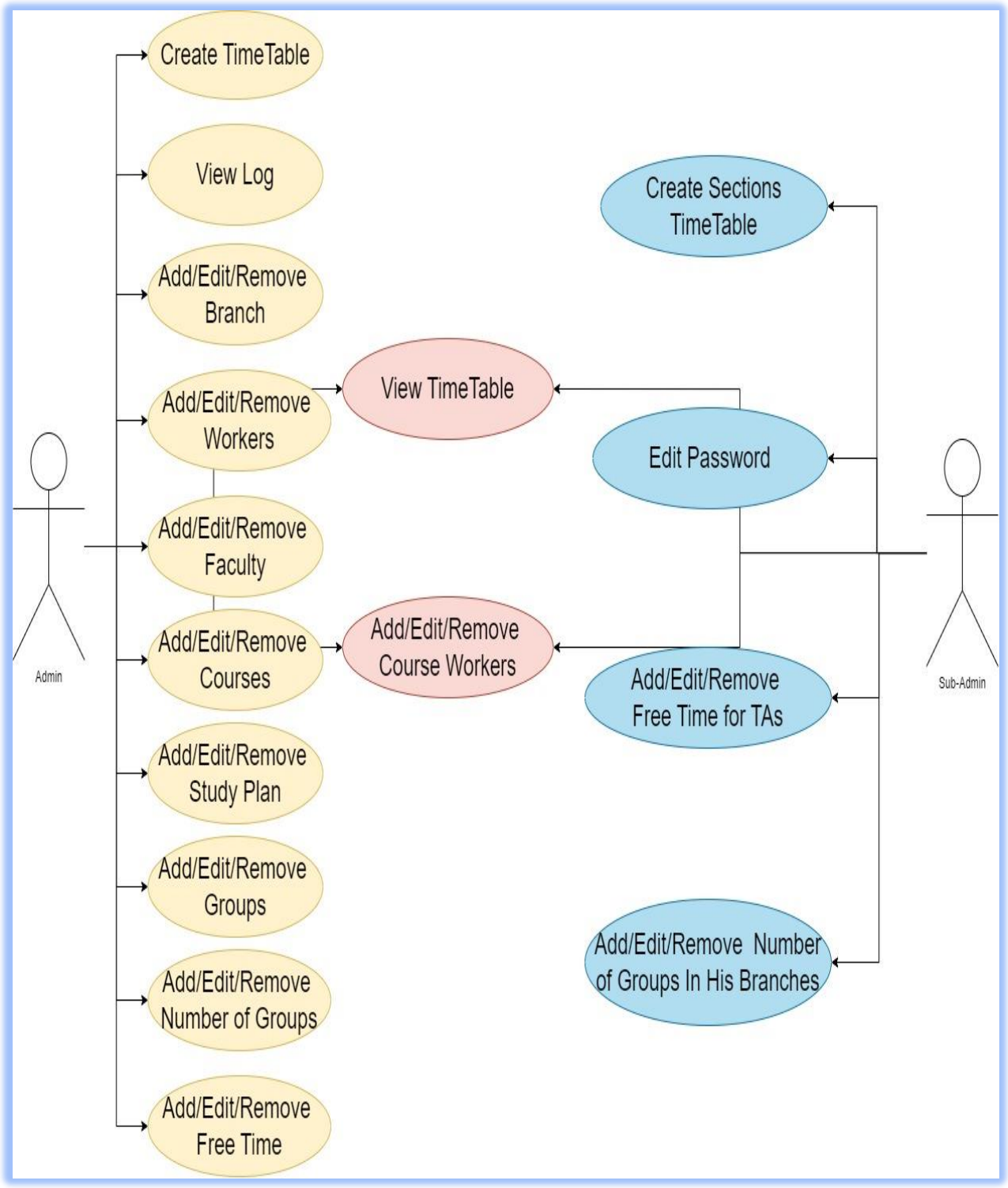

Figure 3.1(use-case diagram)

3.2.2 Use-Case Scenario

- **Actors:**
    - **Admin :**The person with overall responsibility for managing the system.
    - **Sub Admin (Teachers/Staff) :**Users with limited access to specific functions, such as viewing and managing their own schedules.

3.2.2.1 Use-Case Scenario: Staff Management

Name: Staff Management	
Actors:	Admin & SubAdmin
Description	This feature is used to add users to the system and scope the privileges for Admins and SubAdmin.
Successful Completion:	The Data of staff will be updated in Database.
Alternative:	SubAdmin will add data of staff in his branch.
Precondition:	The admin & SubAdmin will login first to access the dashboard.
Post Condition:	The admin & SubAdmin can search, add, update, and remove staff details in the system using this feature

Table 3.1 Staff Management

3.2.2.2 Use Case Scenario: Room Management

Name: Room Management	
Actors:	Admin & SubAdmin
Description	This feature is used to manage the rooms for the class schedules.
Successful Completion:	The Rooms data Will Be Updated in Database.
Alternative:	The SubAdmin can access and manage all room information For Only his Branch.
Precondition:	-The admin & SubAdmin will login first to access the Room Management module.
Post Condition:	The admin & SubAdmin can search, add, update, and remove room details in the system using this feature

Table 3.2 Room Management

3.2.2.3 Use Case Scenario: Course Management

Name: Course Management	
Actors:	Admin
Description	This feature is used to manage the courses of the students in the system.
Successful Completion:	Data Will be Updated in Database.
Alternative:	The admin can access and manage all courses Inside Study Plan.
Precondition:	The admin will login first to access the Course Management module.
Post Condition:	The admin can search, add, update, and remove course details in the system using this feature

Table 3.3 Course Management

3.2.2.4 Use Case Scenario: study plan Table.

Name: study plan	
Actors:	Admin
Description:	This feature is used to manage study plan into the system for Their timetable.
Successful Completion:	Study Plan Data Will be Updated in Database
Alternative:	The admin only Has Access to edit it.
Precondition:	Admin and will first login to access the study plan management module.
Post Condition:	The admin can search, add, update, and remove course details in the system using this feature.

Table 3.4 Study plan Management

3.2.2.5 Use Case Scenario: Lecture & Sec Groups

Name: Lec & Sec Group Management	
Actors:	Admin & SubAdmin
Description	This feature is used to manage the num of Groups for each level.
Successful Completion:	Data Will be Updated in Database.
Alternative:	• The admin can access and manage all Lecture groups. • The SubAdmin can access and manage all sec groups.
Precondition:	The admin & SubAdmin will login first to access the groups Management module.
Post Condition:	The admin can search, add, update, and remove course details in the system using this feature

Table 3.5 Lec & Ses Management

3.2.2.6 Use Case Scenario: Create Lecture Schedule Table

Name: Create Lecture Schedule	
Actors	Admin
Description:	The admin creates a schedule for all levels, ensuring no Conflicts in time or room assignments.
Steps:	1- Admin logs in to the system. 2- Admin selects "Create Lecture Schedule." 3- Admin enters lecture details (course name, instructor, duration, etc.). 4- The system displays available rooms and time slots. 5- Admin assigns rooms and times to lectures, considering room availability, instructor schedules, and avoiding course conflicts. 6- System verifies no conflicts exist. 7- Admin confirms schedule. 8- System generates and publishes lecture schedule
Constraints:	• Room availability • Instructor schedules • Course conflicts • Time Conflicts
Outcomes:	• Conflict-free lecture Timetable • Improved time management and resource allocation

Table 3.6 Create Lecture Schedule

3.2.2.7 Use Case Scenario: Manage Session Times Table

Name: Create Session Schedule	
Actors	SubAdmin
Description:	Sub Admins (branch officials) set session times for his branch only, based on the published lecture schedule.
Steps:	1. Sub Admin logs in to the system. 2. Sub Admin selects "Manage Session Times." 3. System displays lecture schedule all levels. 4. Sub Admin sets session times, ensuring compliance with lecture schedule and room availability. 5. System verifies no conflicts exist. 6. Sub Admin confirms session times. 7. System generates and publishes session schedule.
Constraints:	• Room availability • Instructor schedules • Course conflicts • Time Conflicts
Outcomes:	• Conflict-free Session Timetable
Alternative	• Admin can create sec Timetables for all branches.

Table 3.7 Create Session Schedule

3.3 Class Diagram

Class diagrams play a crucial role in object-oriented analysis and design. They visually represent the classes in a system, showcasing their relationships, such as inheritance, aggregation, and association, along with the activities and attributes of these classes. These diagrams serve multiple purposes, ranging from conceptual/domain modeling to detailed design modeling, offering a comprehensive view of the system's structure and functionality. To see the full diagram, click here.

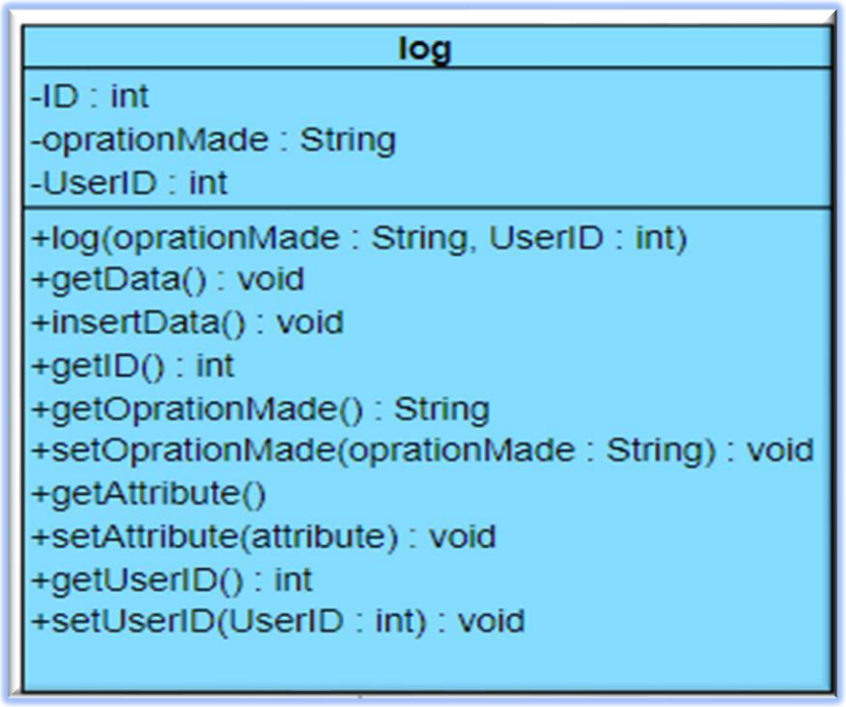

Figure 3.2(login class diagram)

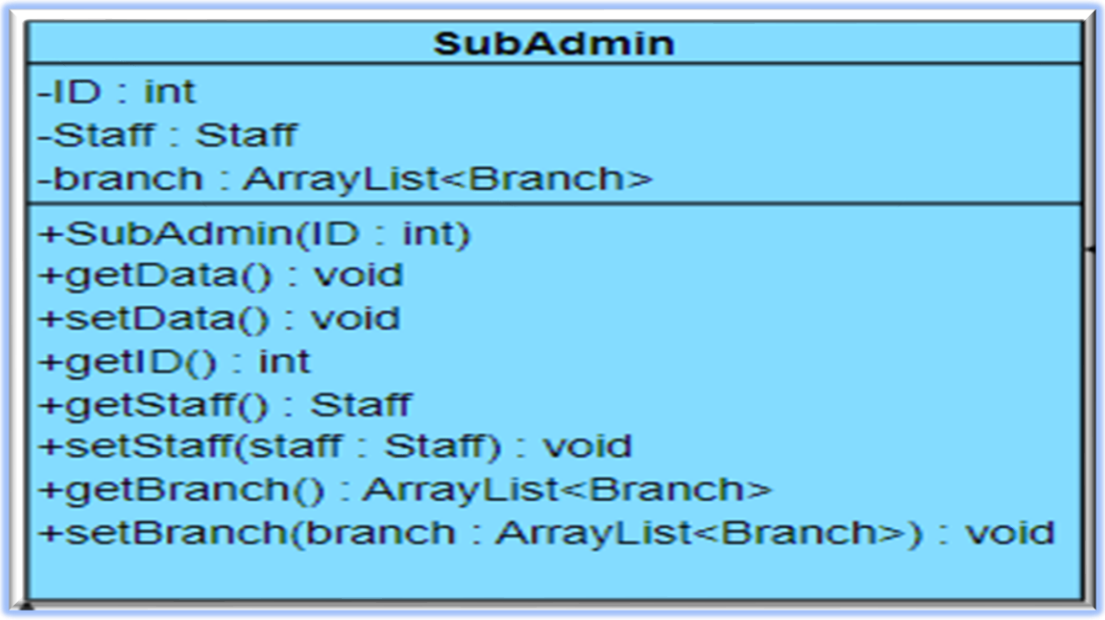

Figure 3.3(sub-admin class diagram)

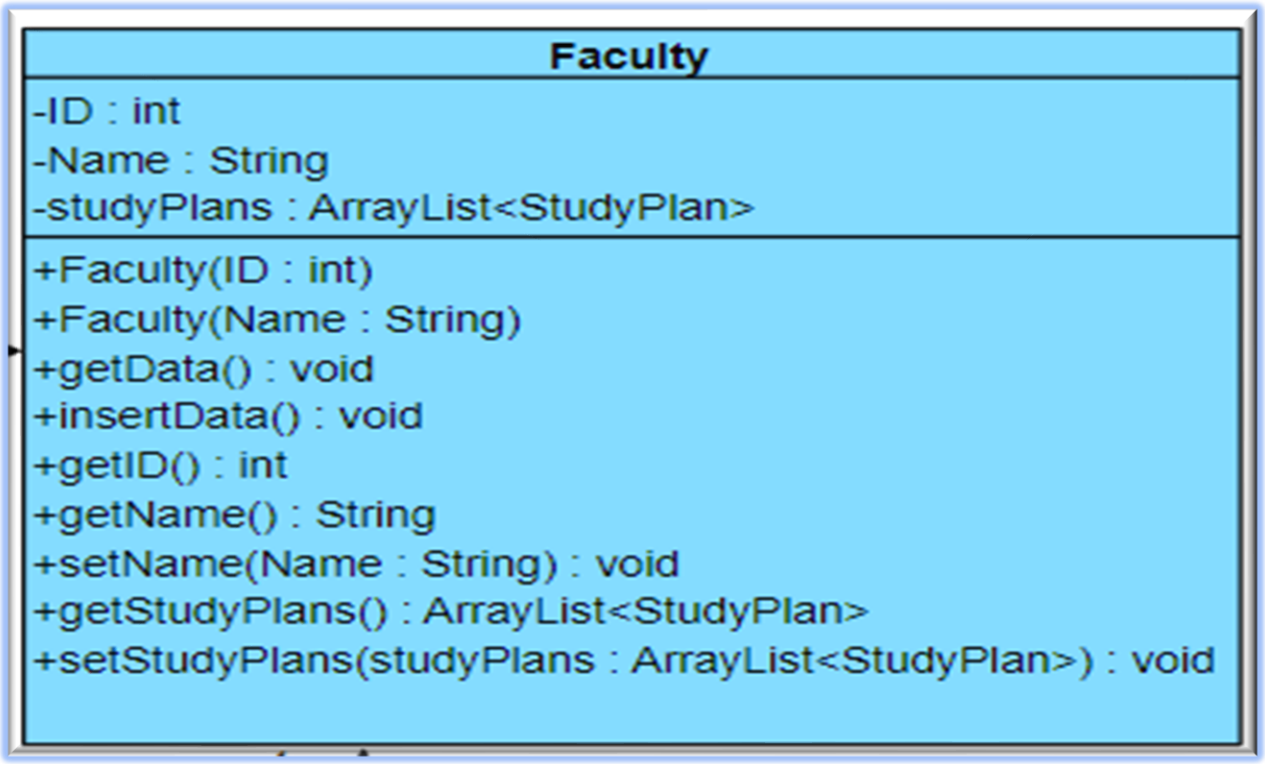

Figure 3.4(faculty class diagram)

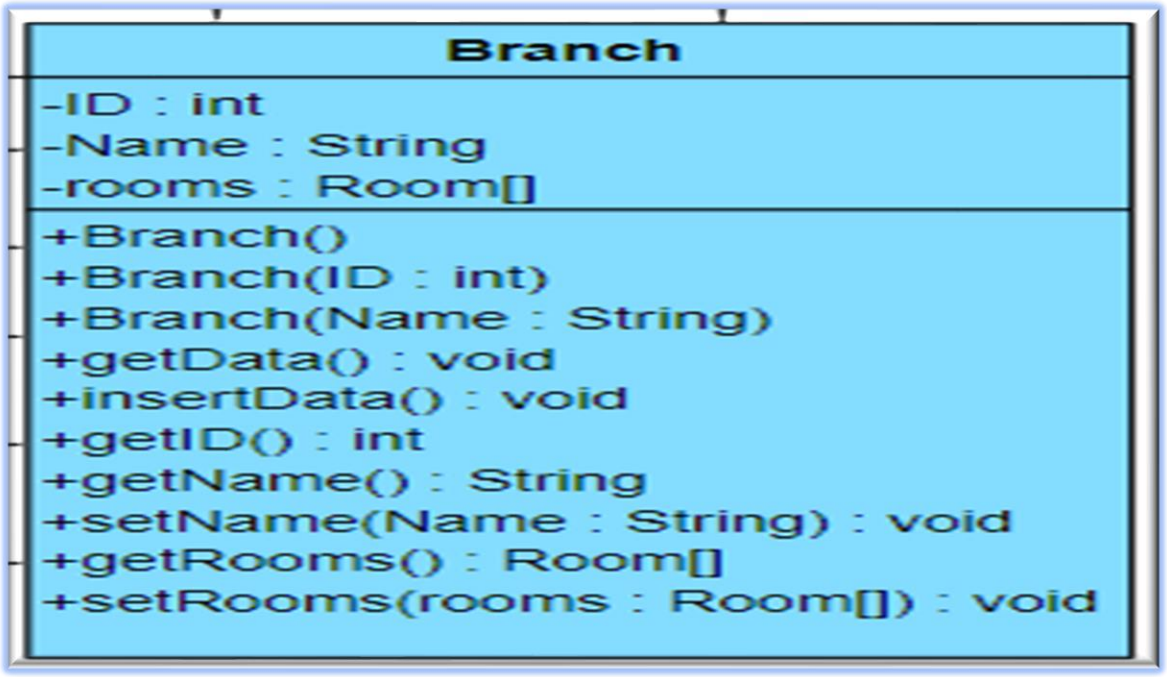

Figure 3.5(branch class diagram)

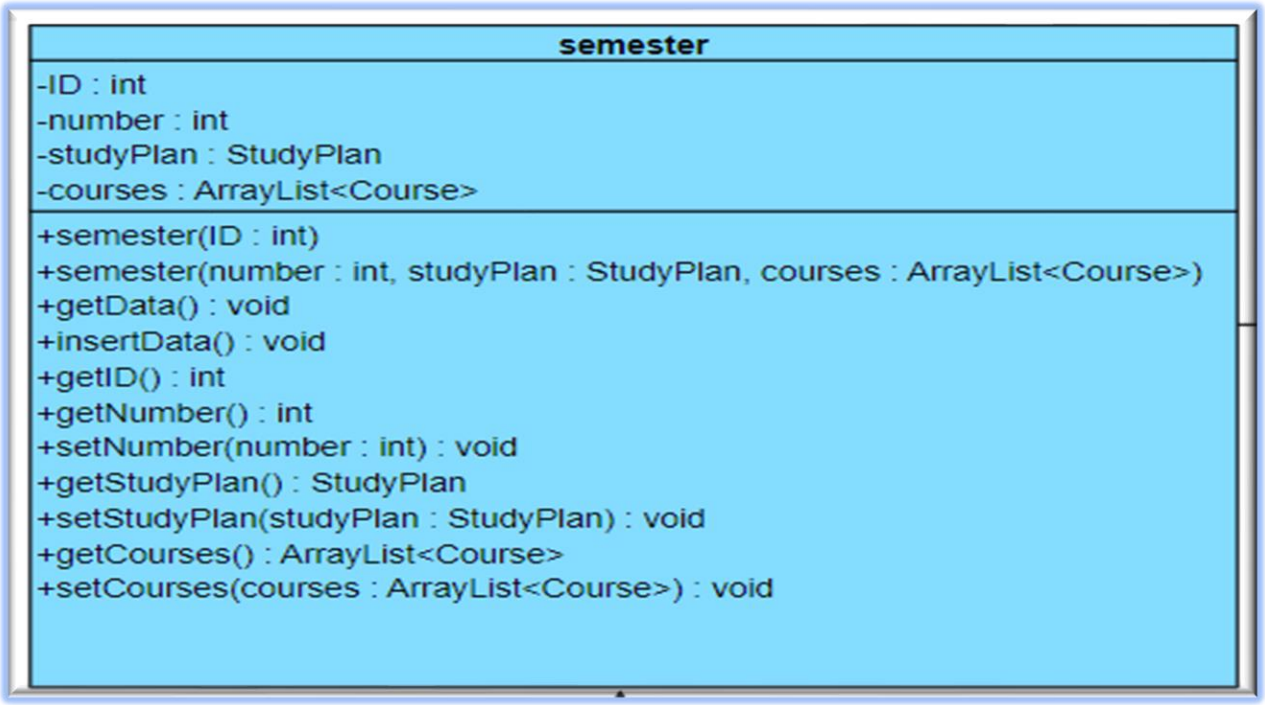

Figure 3.6(semester class diagram)

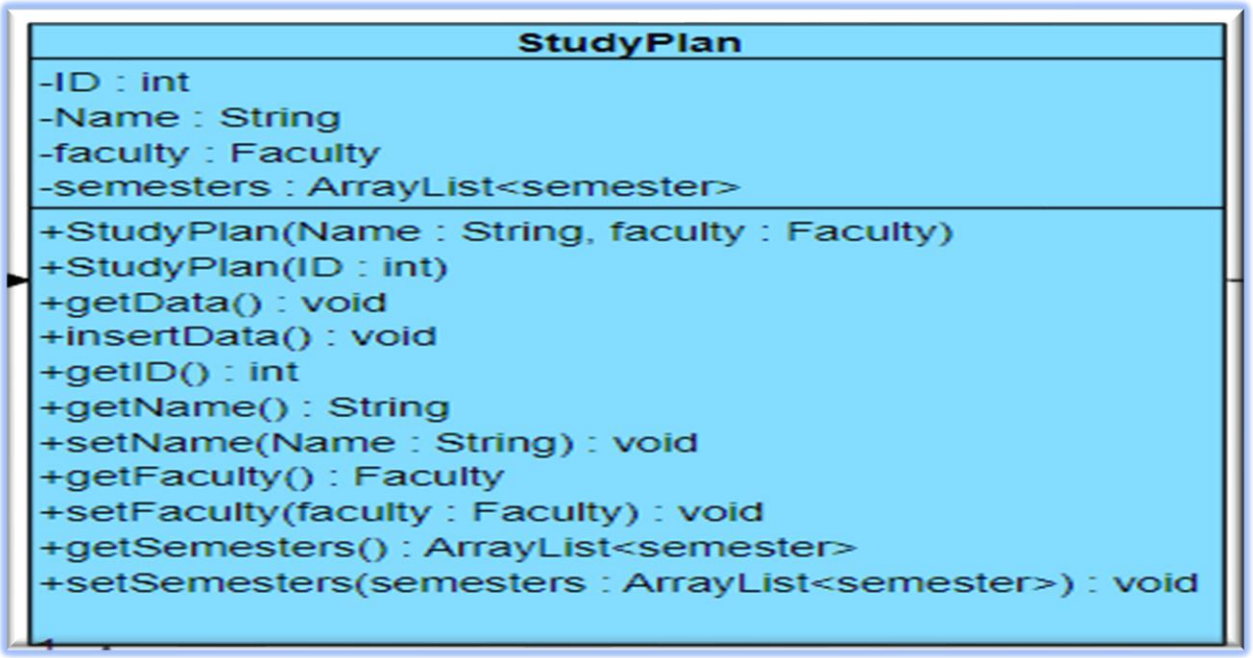

Figure 3.7(study plan class diagram)

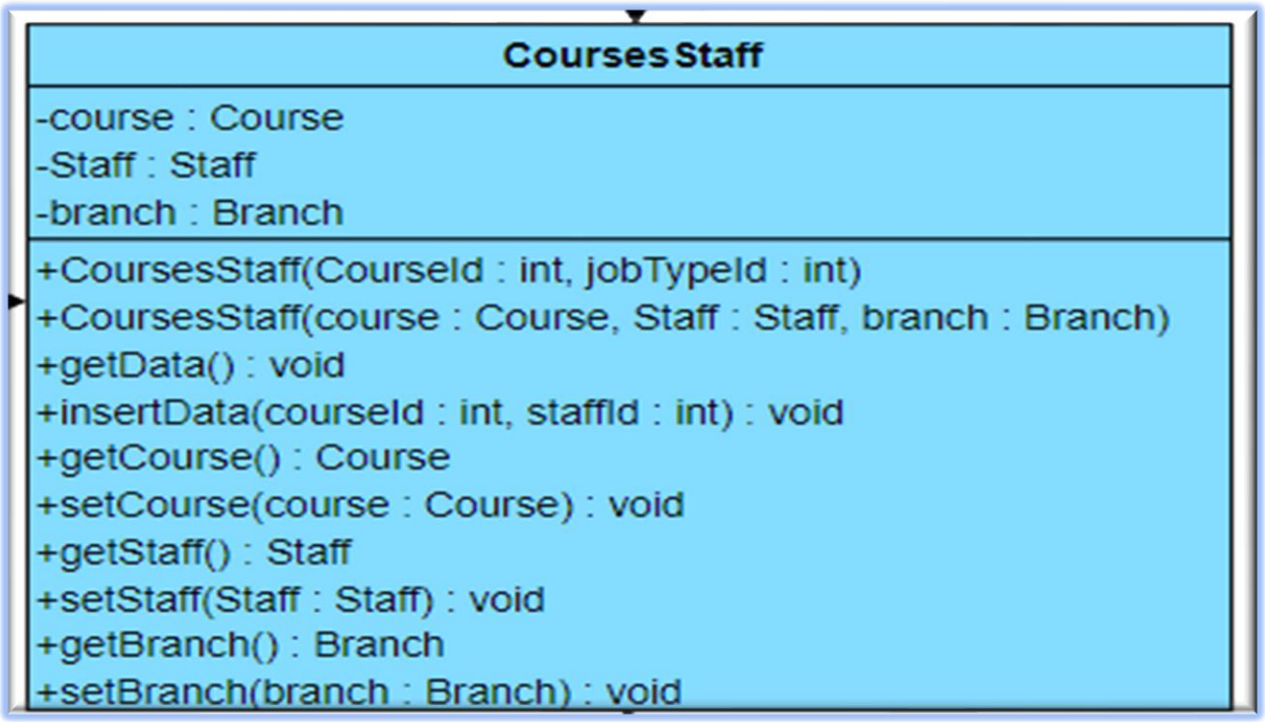

Figure 3.8 (courses staff class diagram)

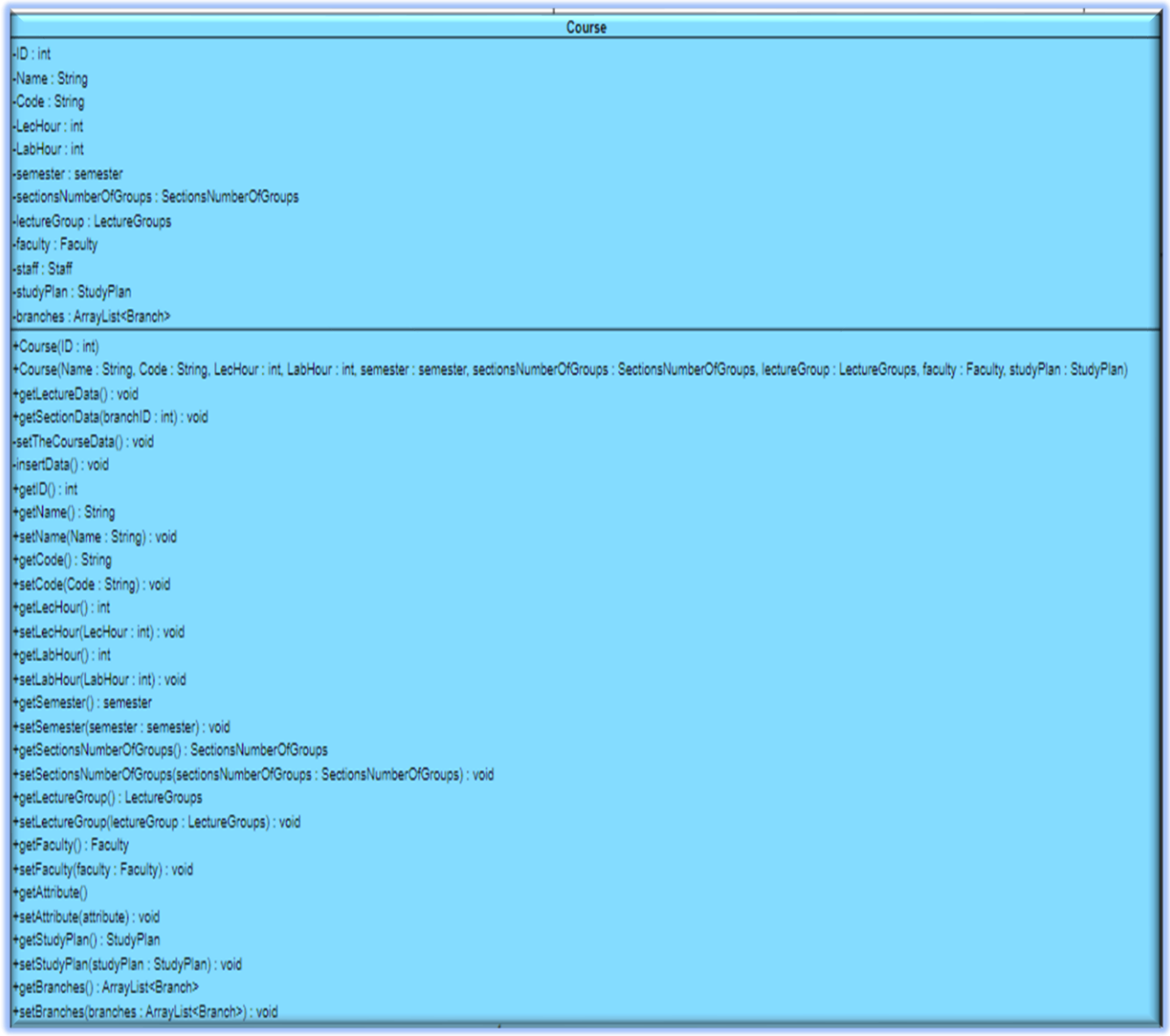

Figure 3.9(course class diagram)

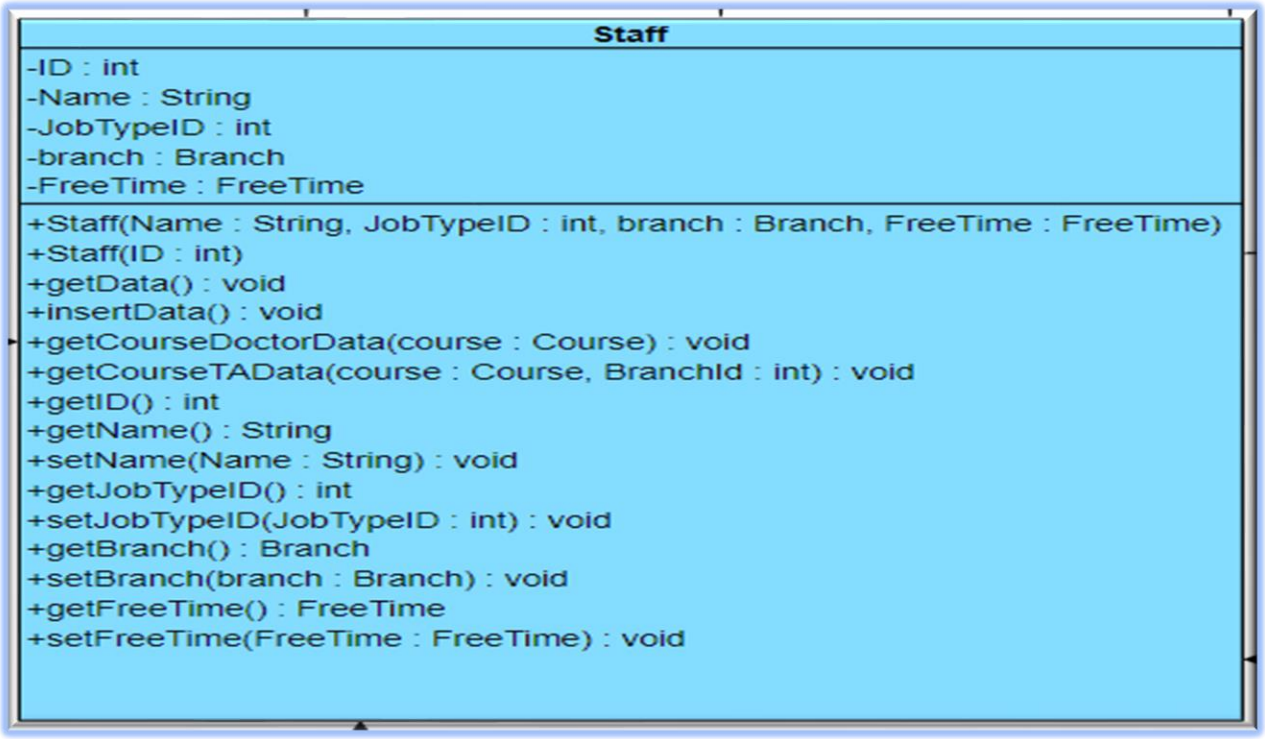

Figure 3.10 (staff class diagram)

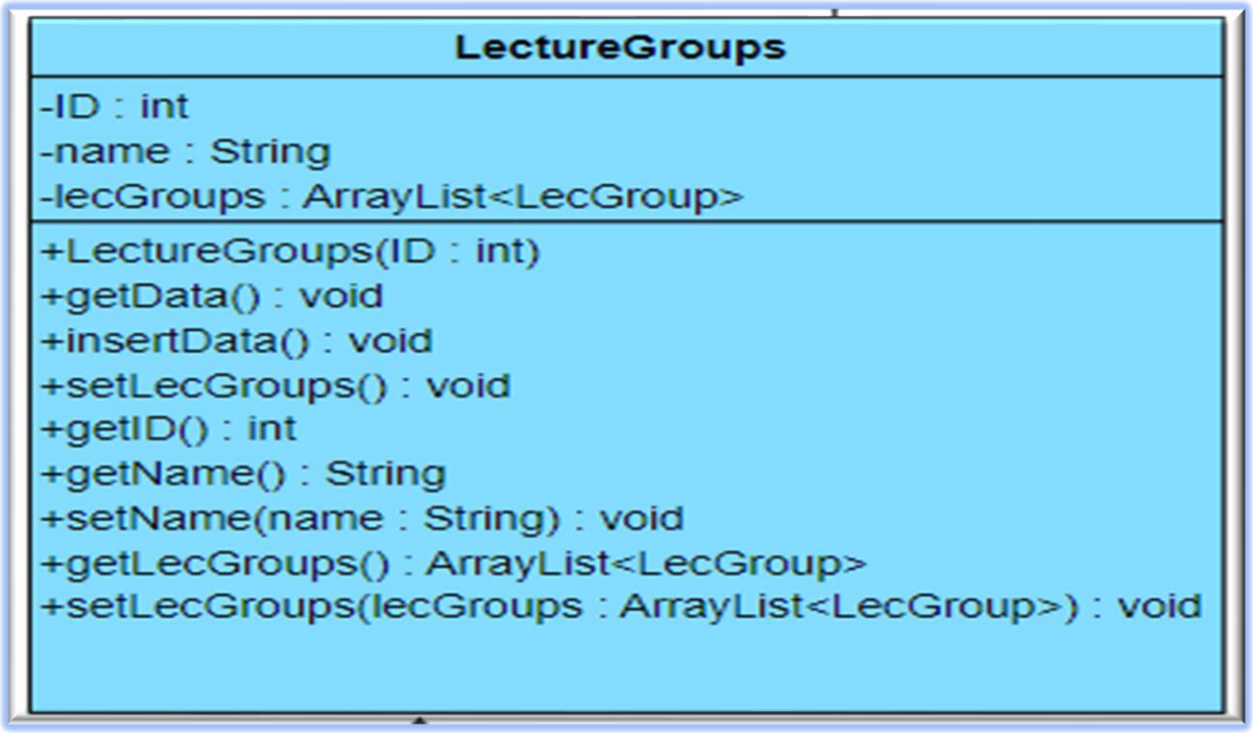

Figure 3.11 (lecture groups class diagram)

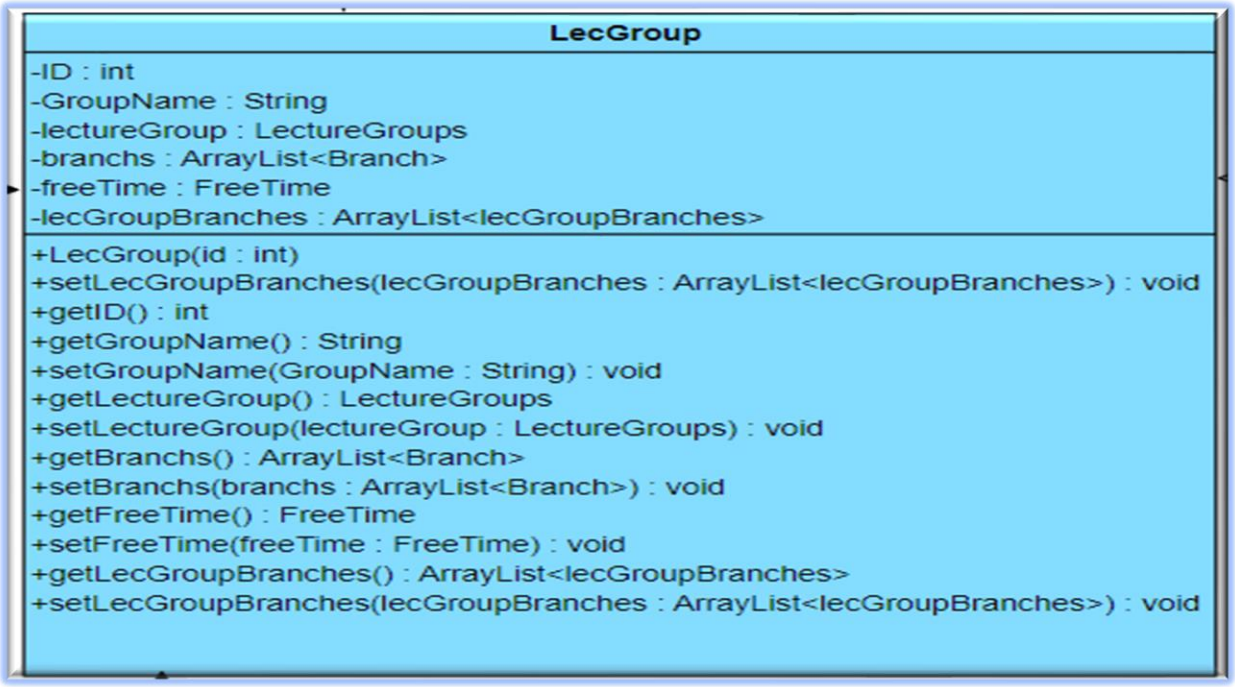

Figure 3.12 (lec groups class diagram)

Figure 3.13 (lec groups branches class diagram)

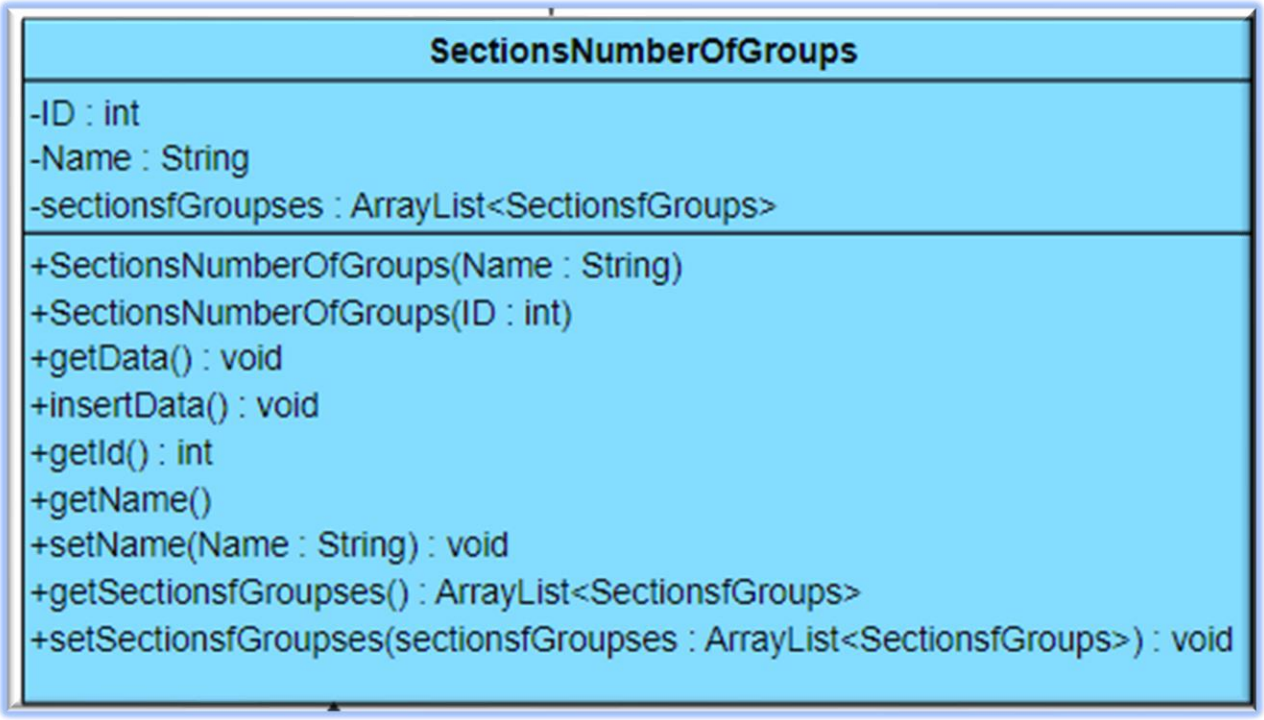

Figure 3.14 (section number of groups class diagram)

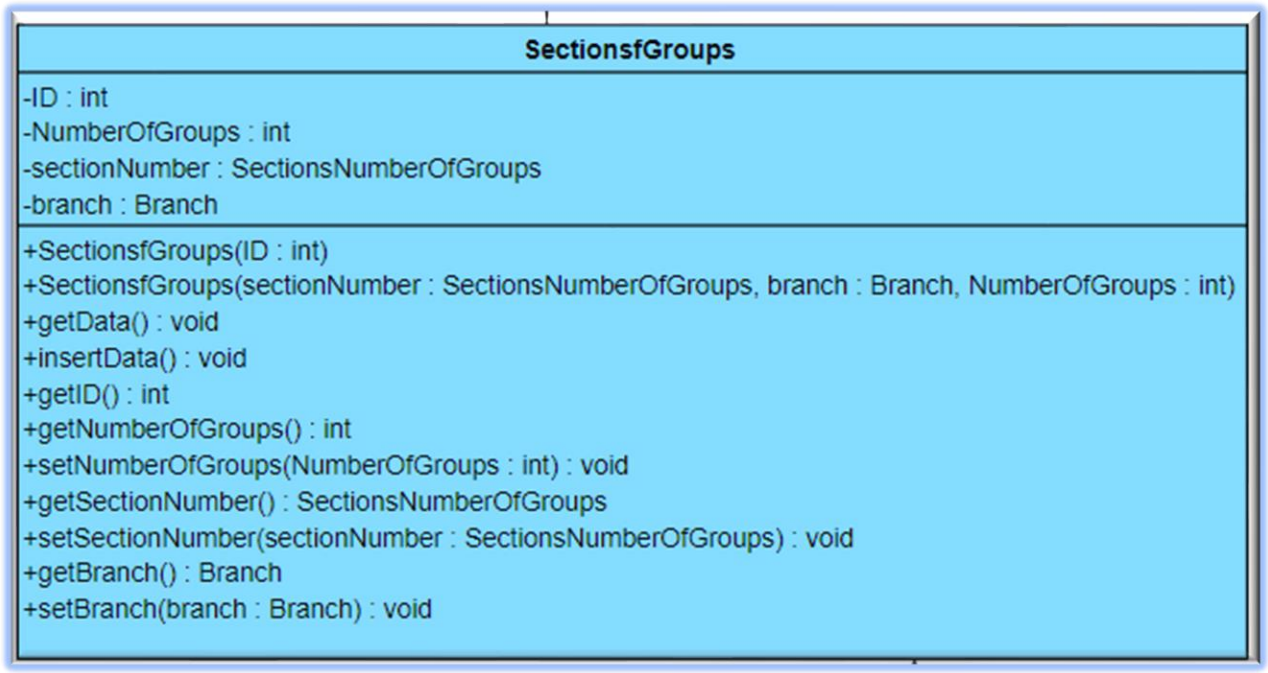

Figure 3.15 (section groups class diagram)

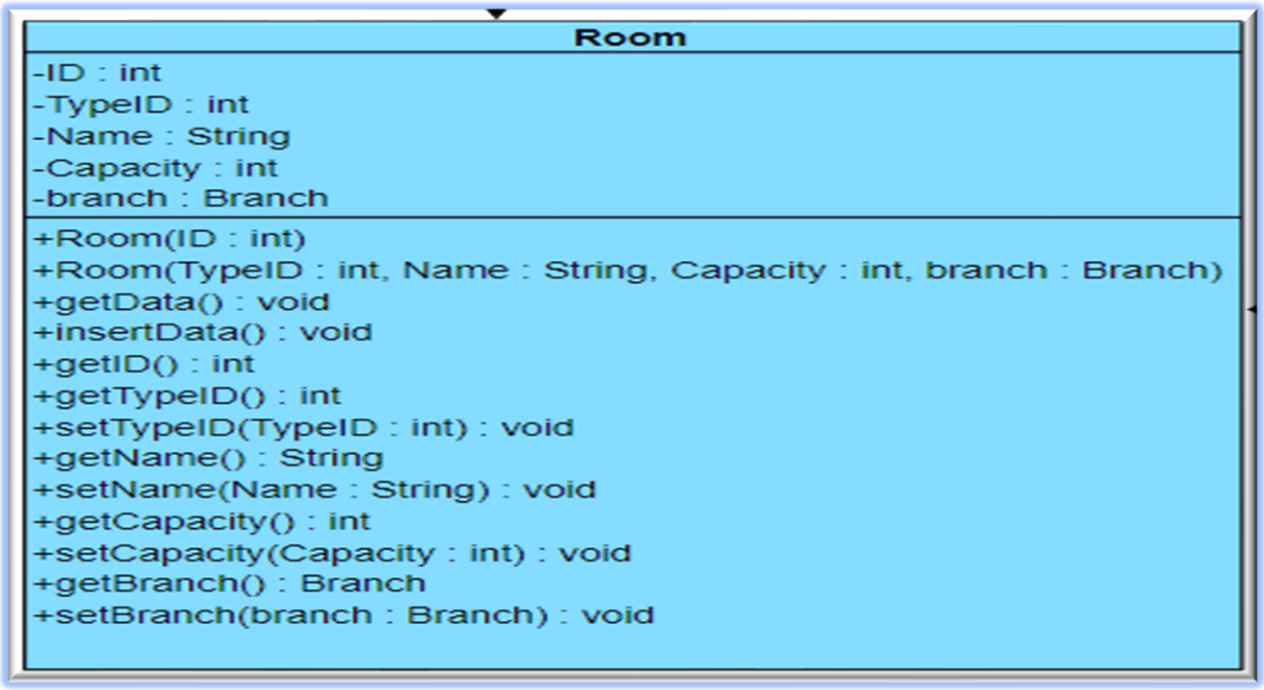

Figure 3.16 (room class diagram)

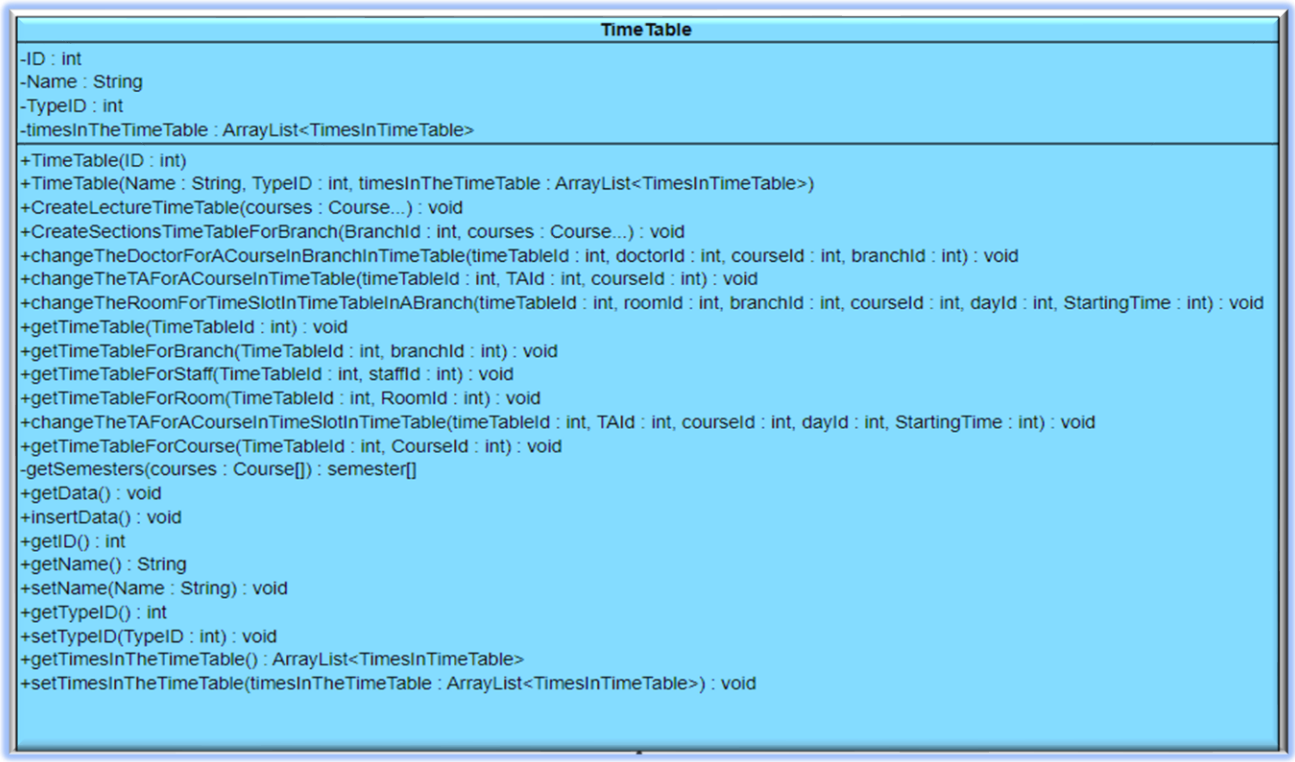

Figure 3.17 (time-table class diagram)

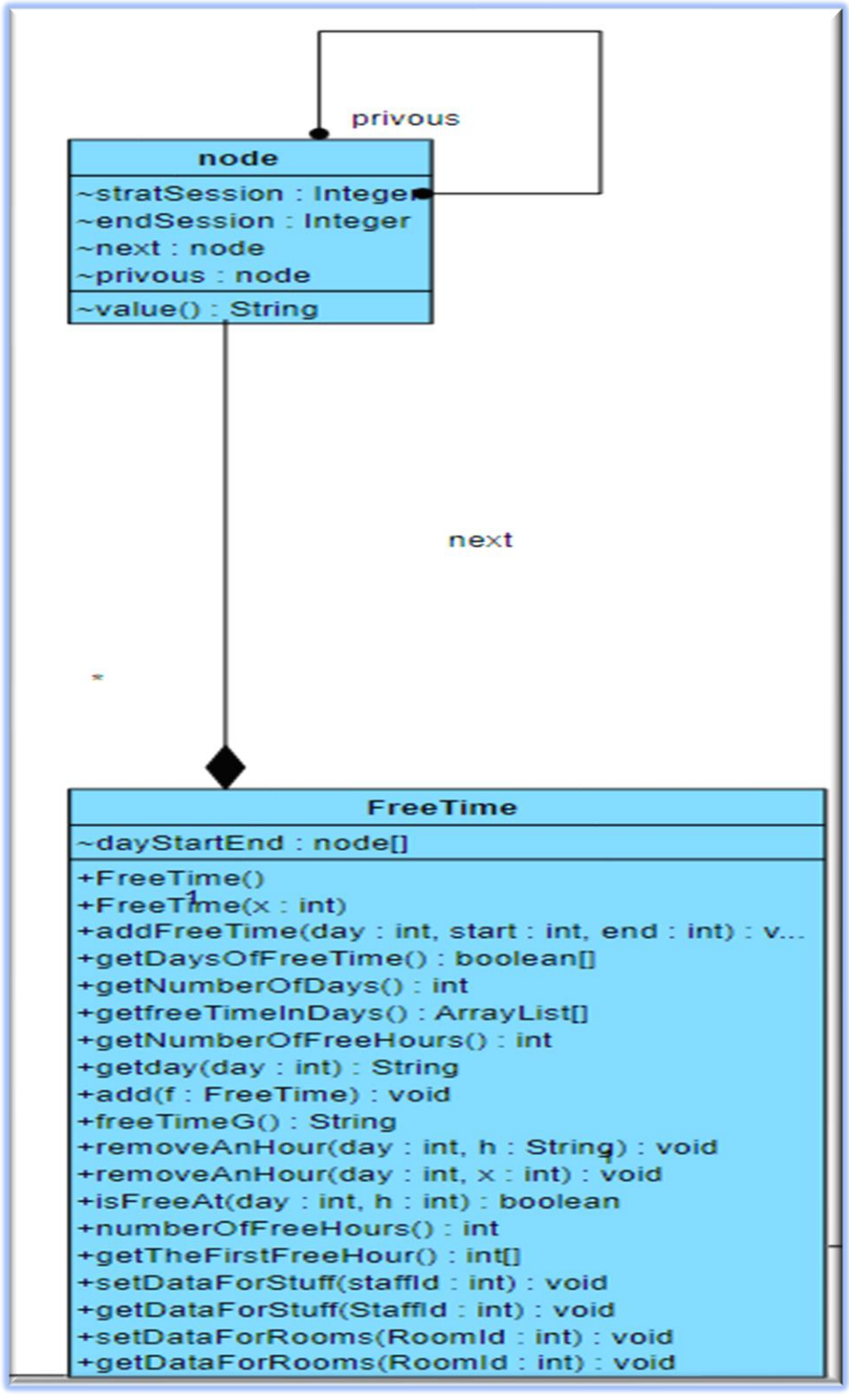

Figure 3.18 (free time class diagram)

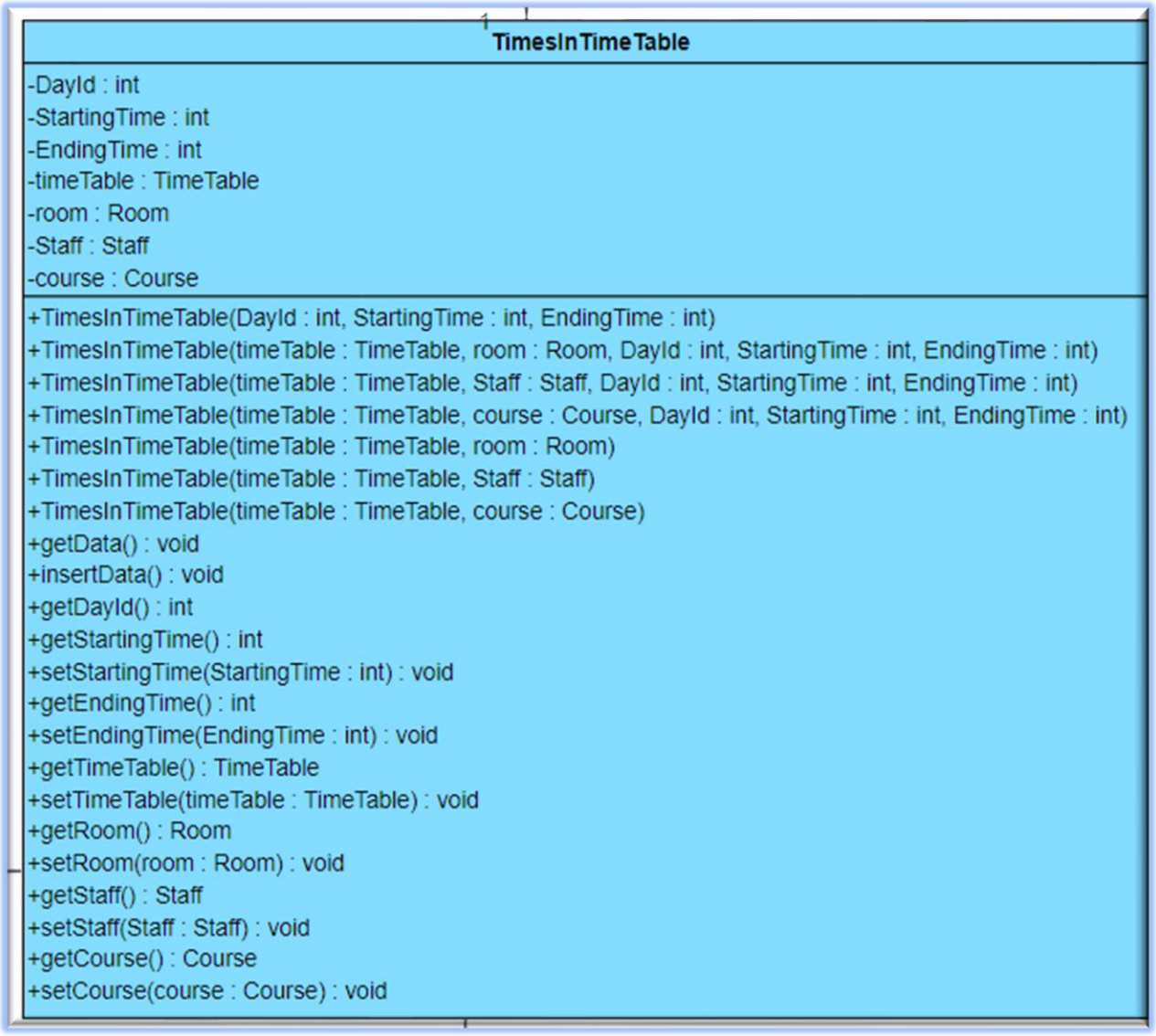

Figure 3.19 (times in time-table class diagram)

3.4 ERD Diagram

It is a visual representation used in system analysis to model and describe the relationships between different entities in a system. In the context of database design and software engineering, ERDs are a crucial tool for understanding the structure of a system and its underlying data. Click here!

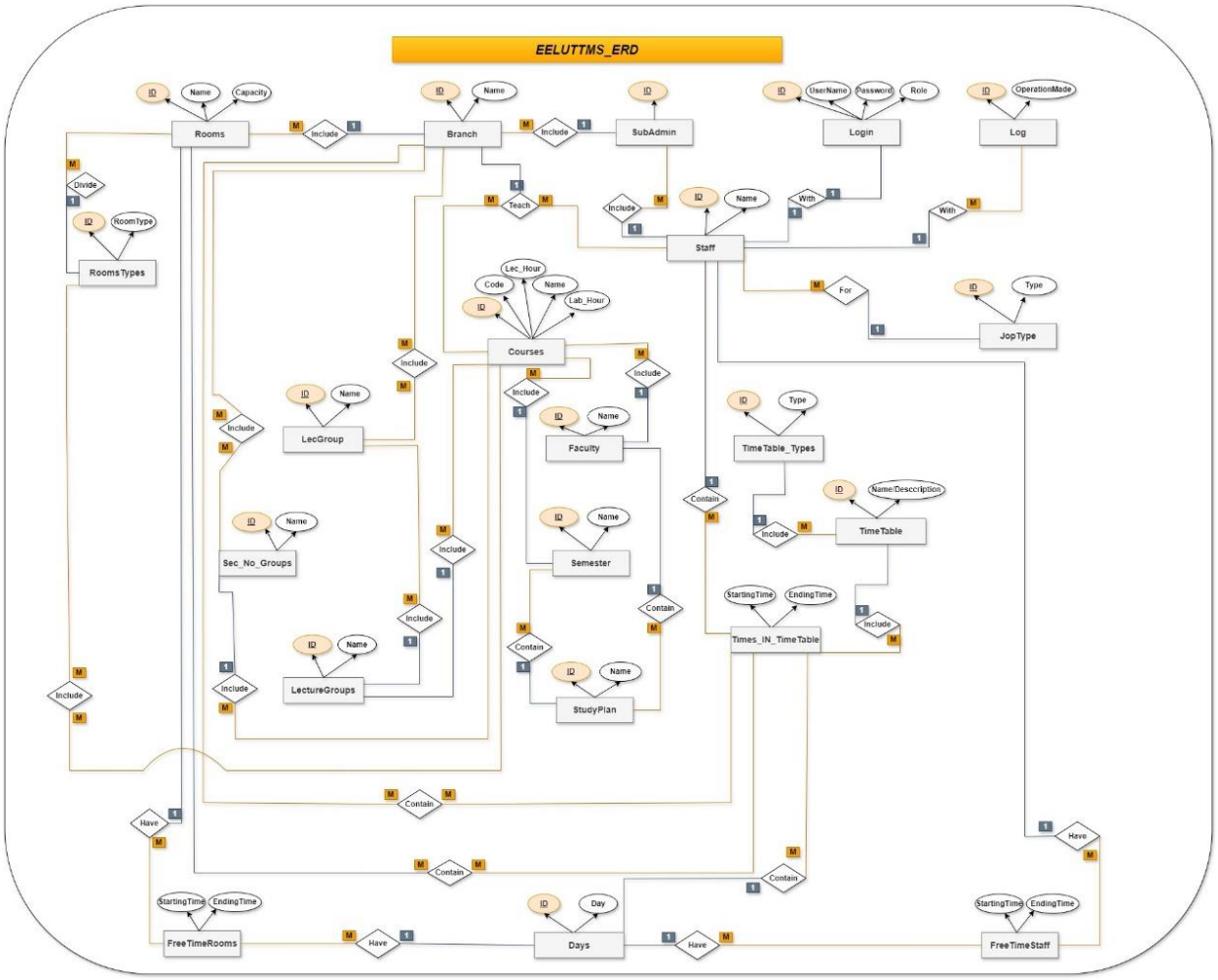

Figure 3.20 (Entity-Relationship Diagram)

3.5 Schema Diagram

A database schema represents the logical configuration of all or part of a relational database. It can exist both as a visual representation and as a set of formulas known as integrity constraints that govern a database. These formulas are expressed in a data definition language, such as SQL. As part of a data dictionary, a database schema indicates how the entities that make up the database relate to one another, including tables, views, stored procedures, and more.

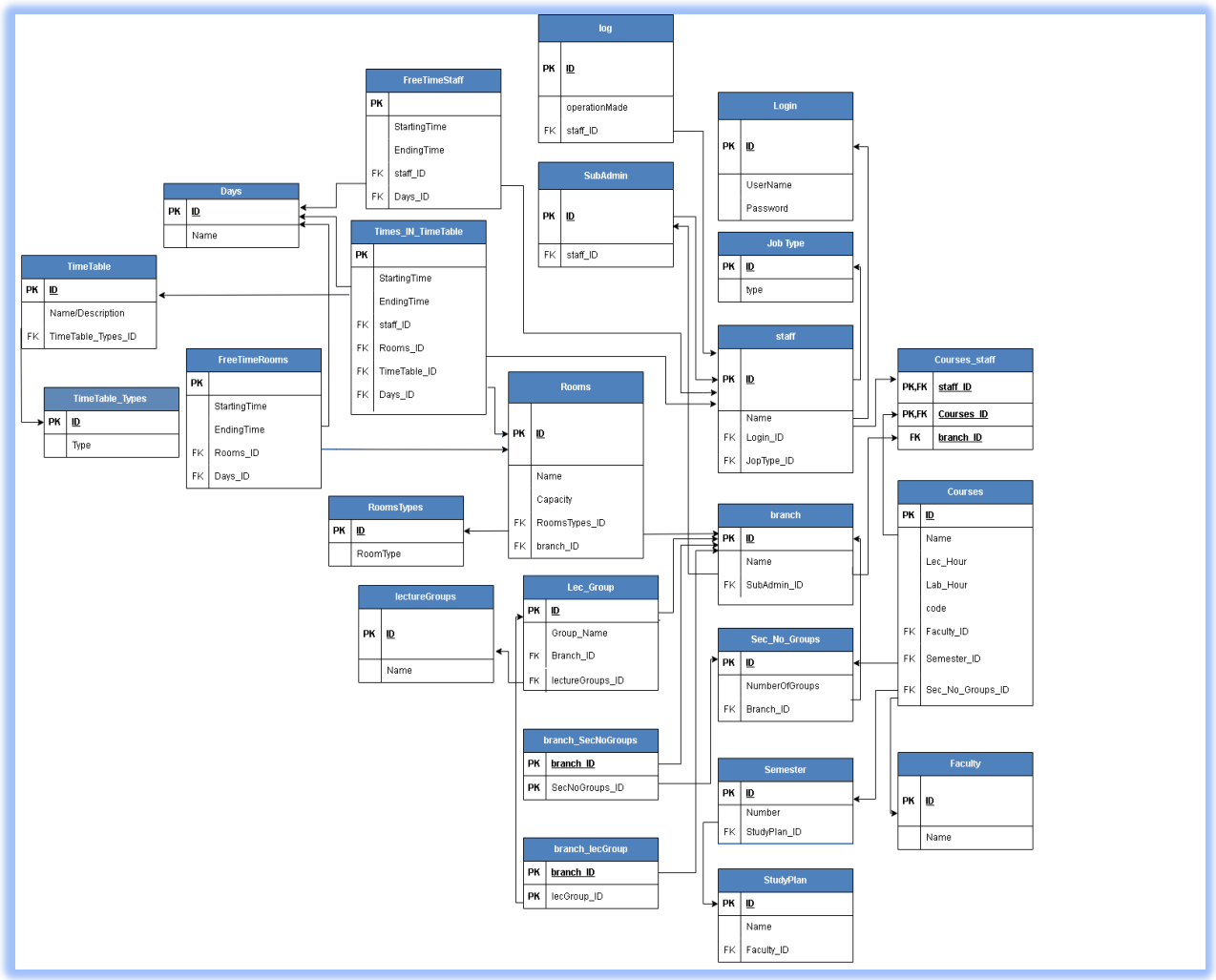

Figure 3.21 (Schema Diagram)

3.6 Sequence Diagram

During the execution of a use case or any action, this defines how objects communicate with one another via messages. They show how messages are delivered and received between objects, as well as the order in which messages are sent. It also describes how operations are carried out according to the time of operation.

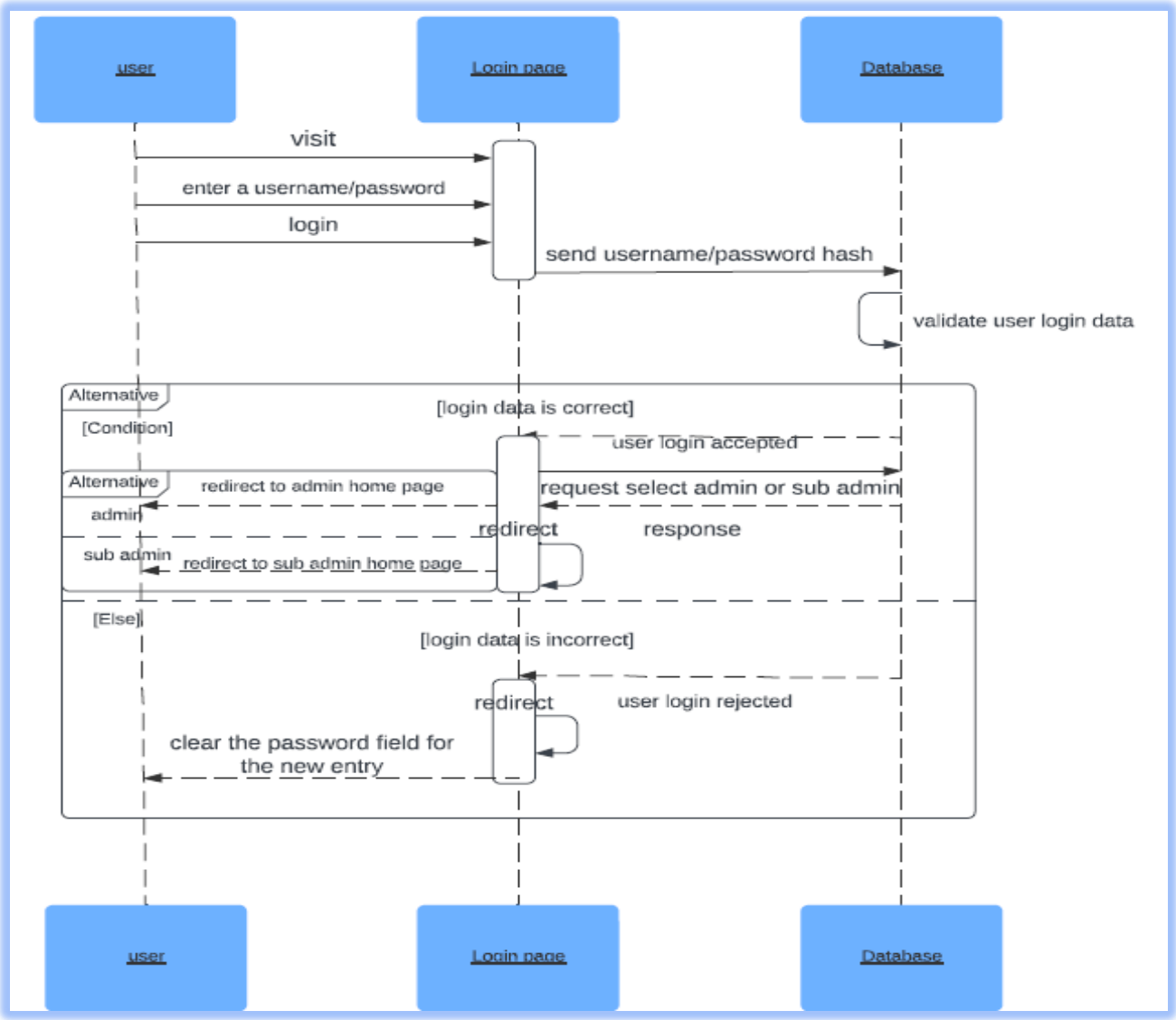

Figure 3.22 (login Sequence diagram)

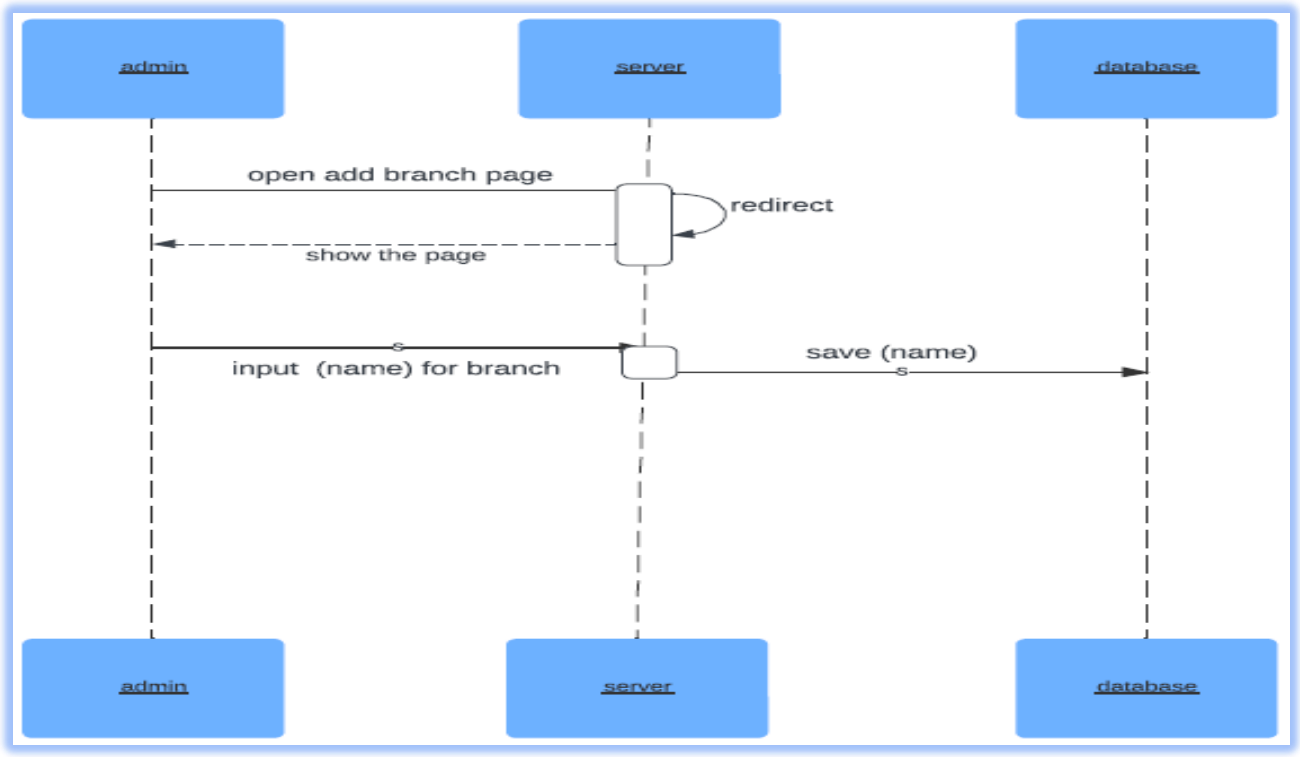

Figure 3.23 (open add branch Sequence diagram)

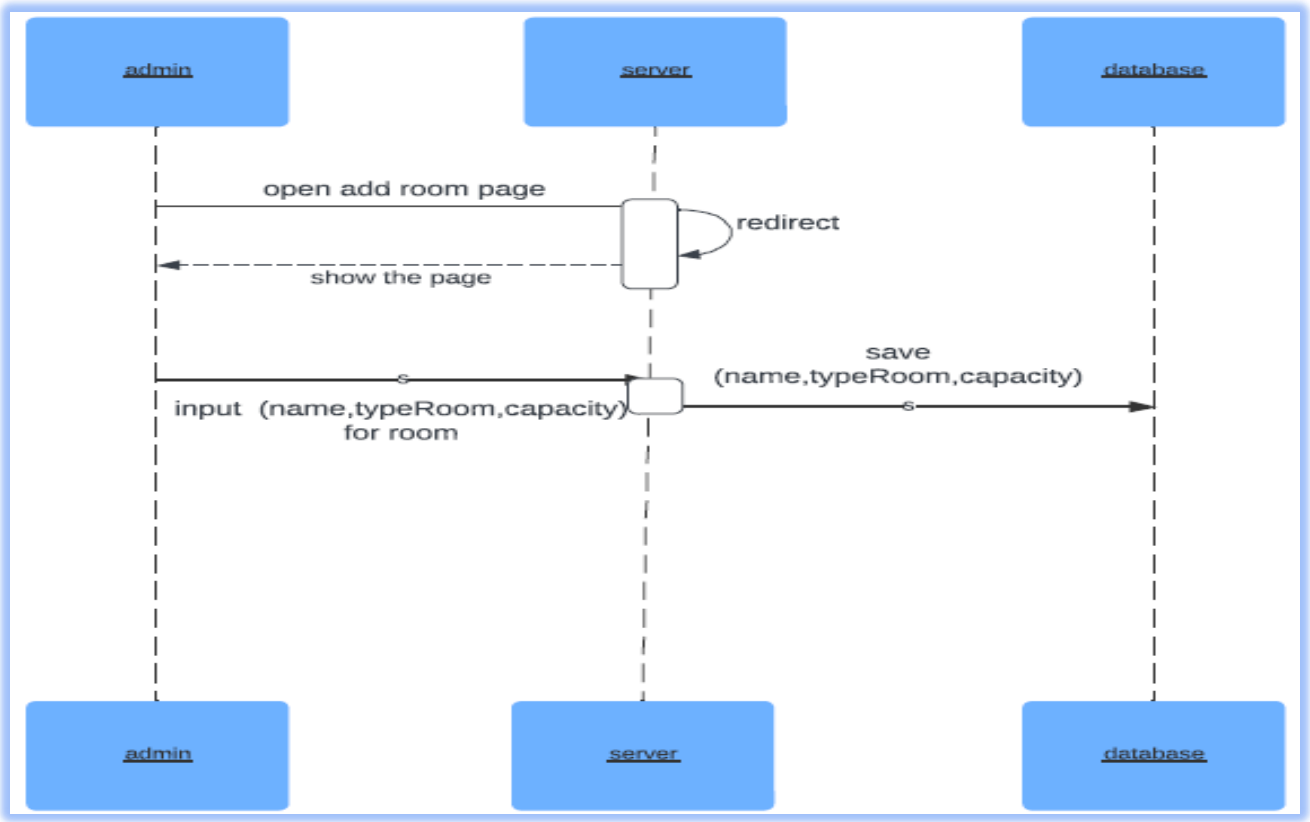

Figure 3.24 (Add-room Sequence diagram)

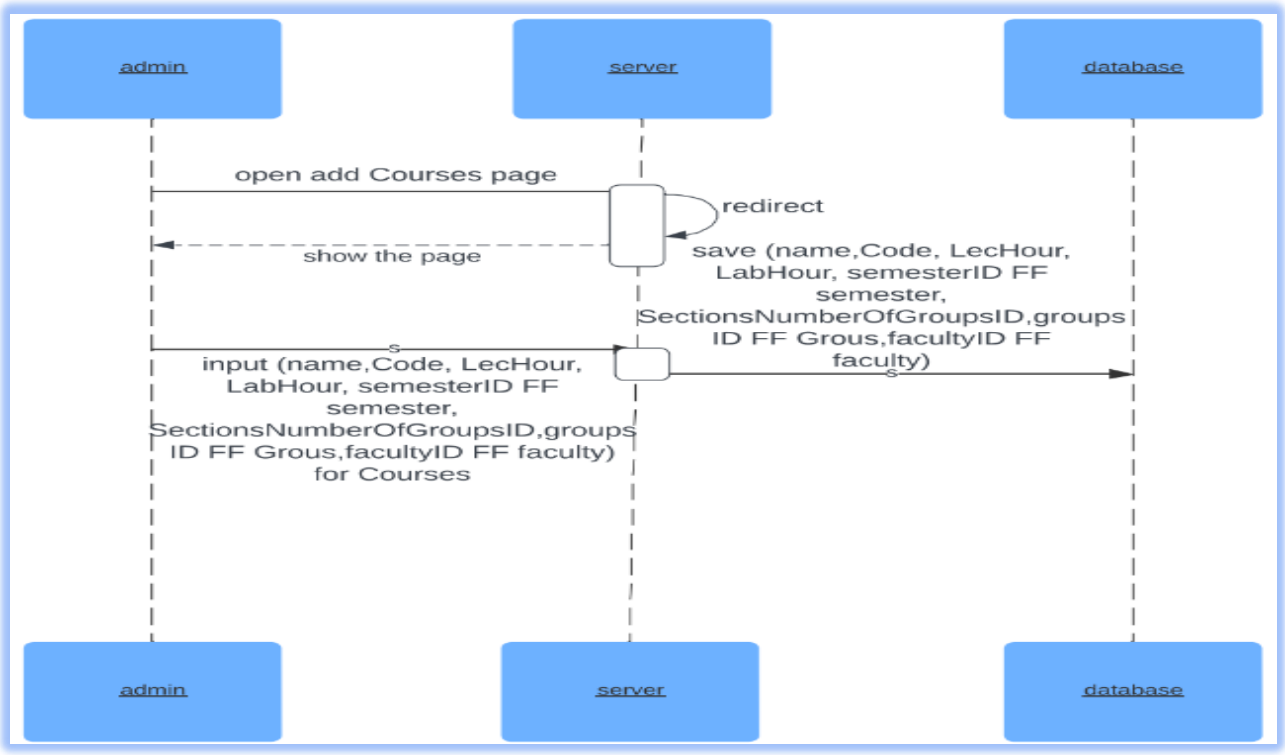

Figure 3.25 (Add courses Sequence diagram)

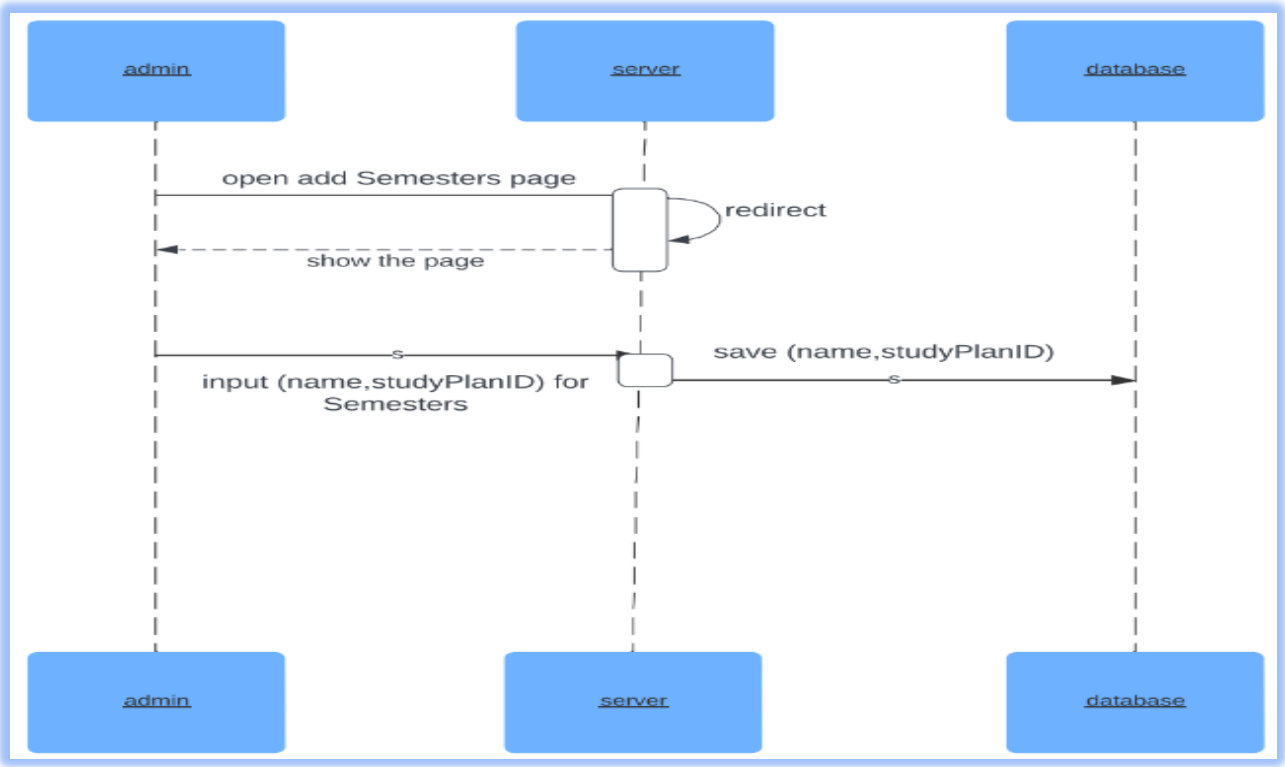

Figure 3.26 (Add Semester Sequence diagram)

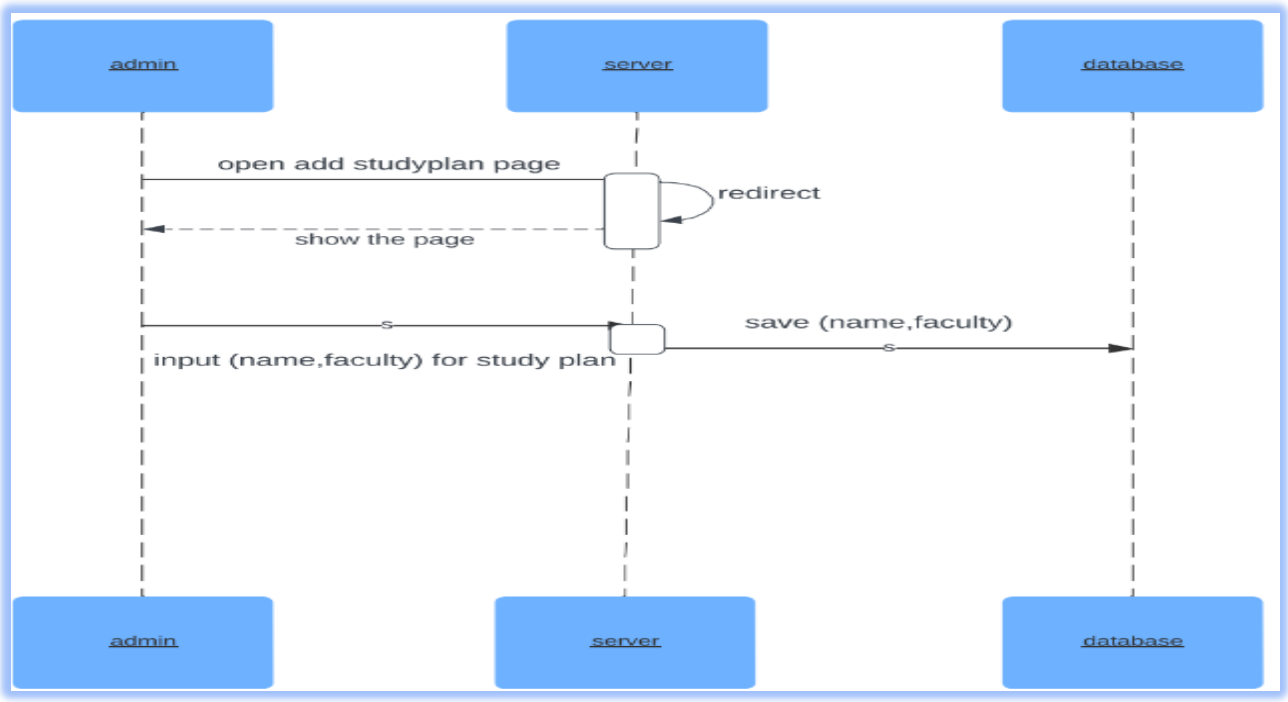

Figure 3.27 (Add Study Plane Sequence diagram)

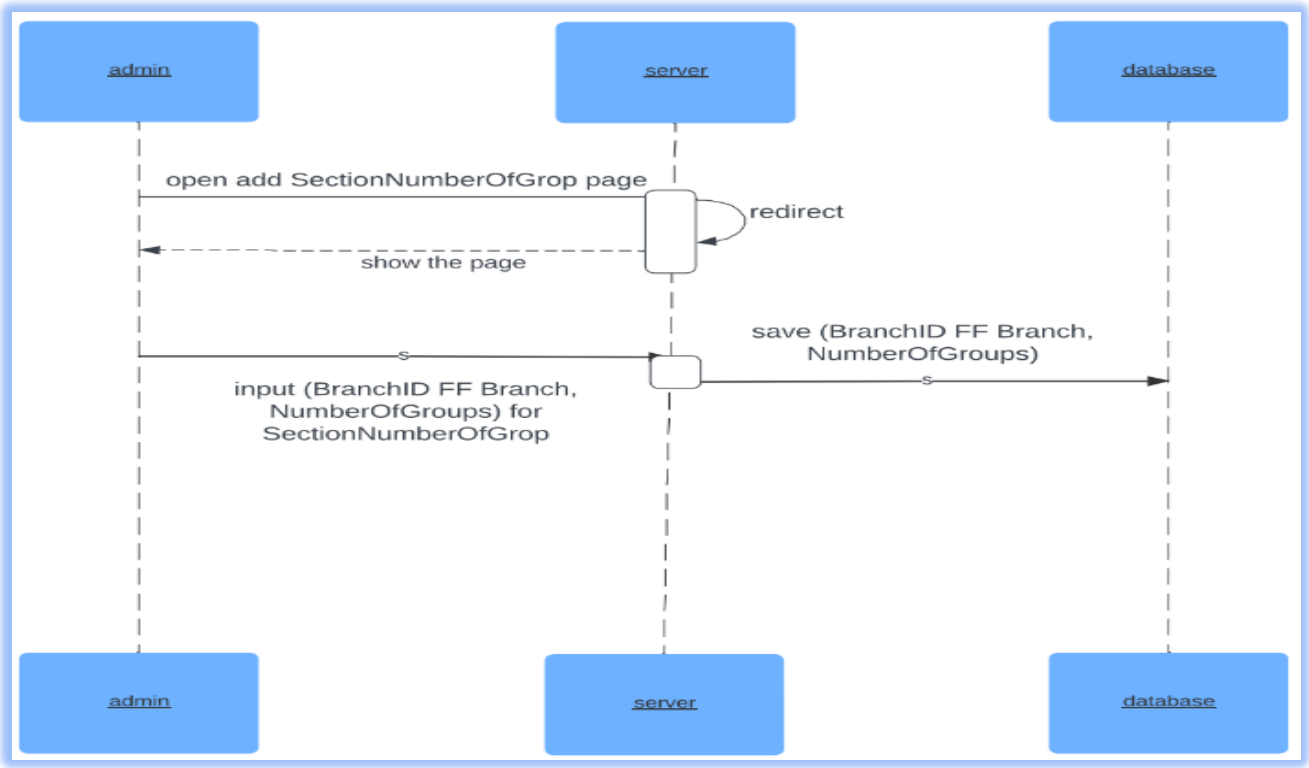

Figure 3.28 (Add Section Number of Group Sequence diagram)

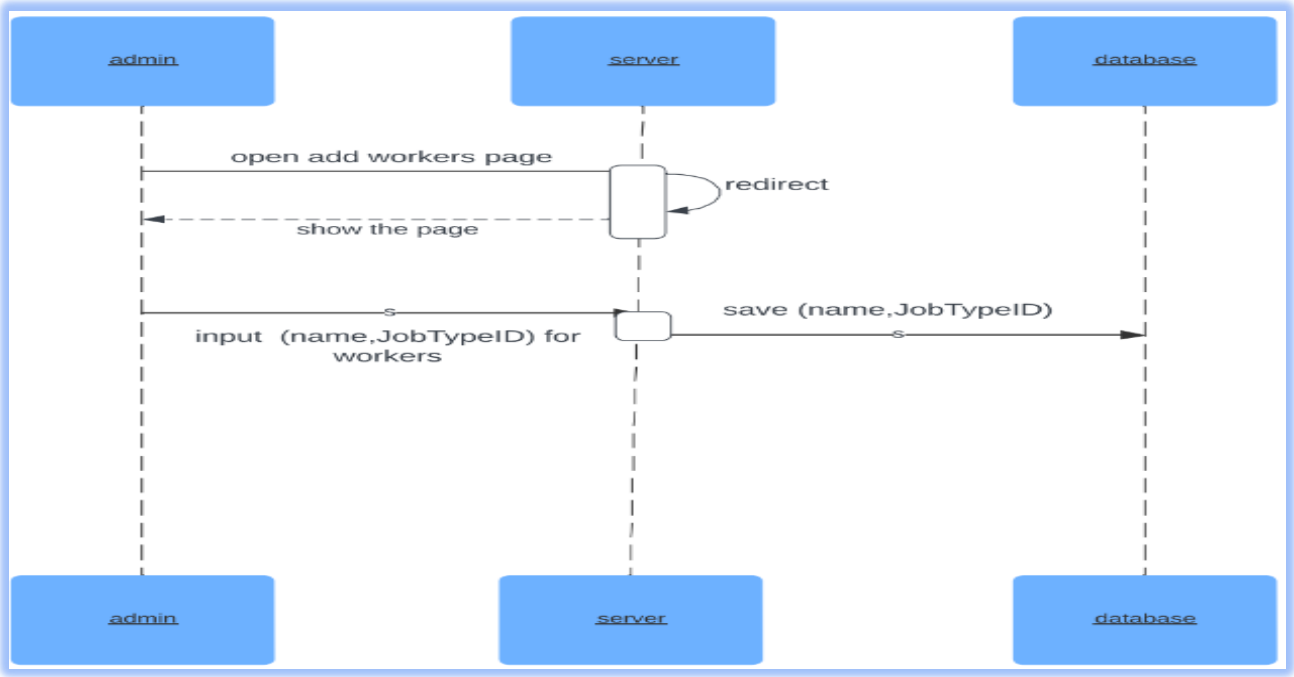

Figure 3.29 (Add Staff Sequence diagram)

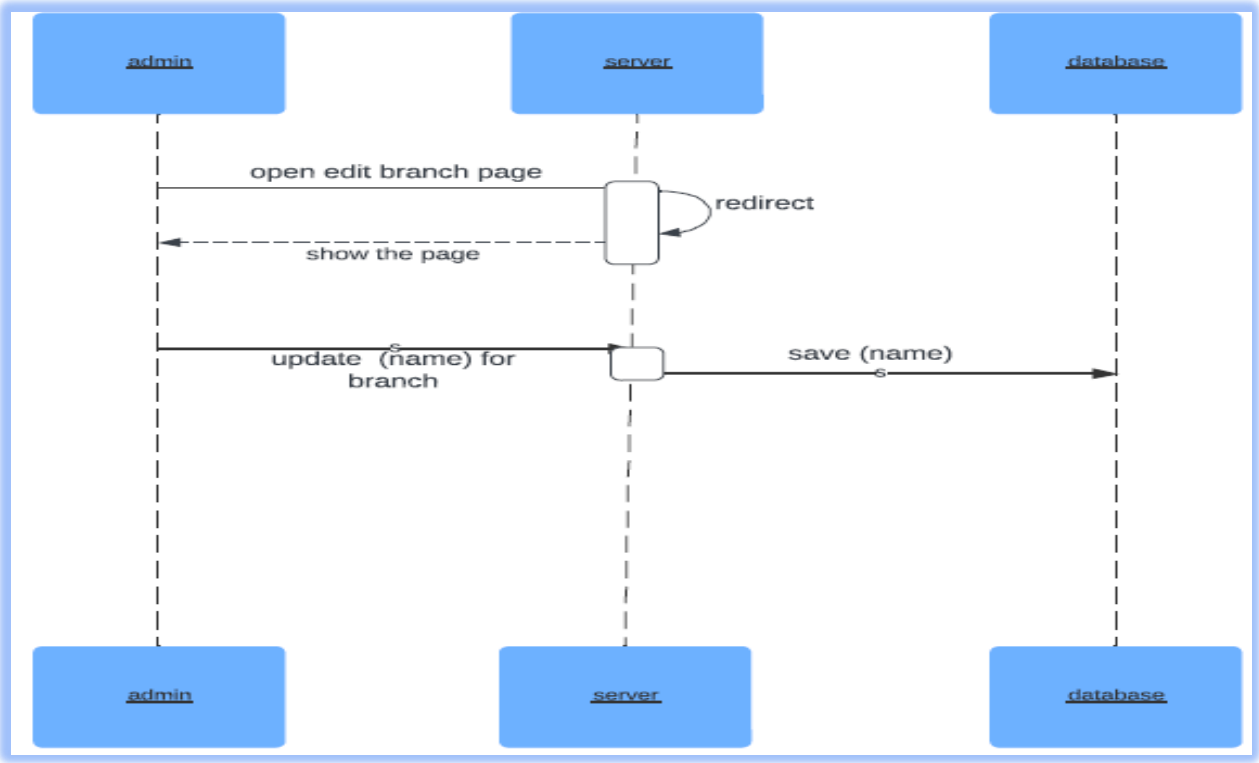

Figure 3.30 (Edit-branch Sequence diagram)

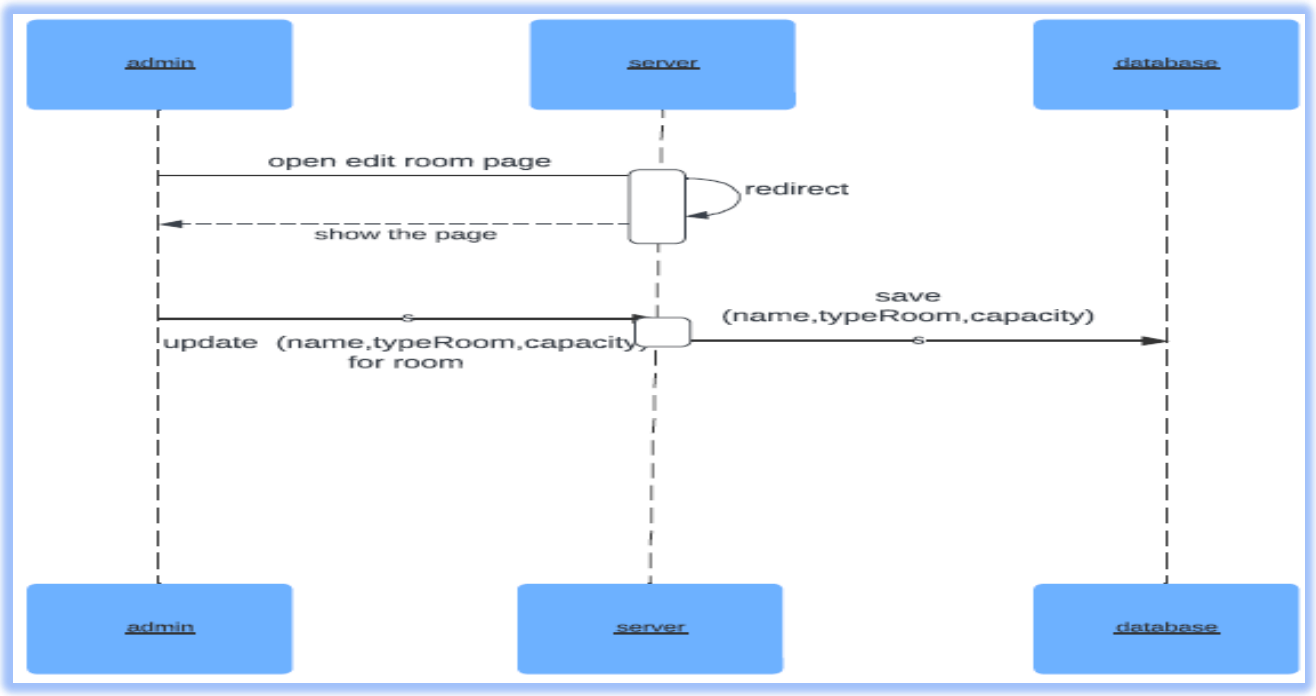

Figure 3.31 (Edit-room Sequence diagram)

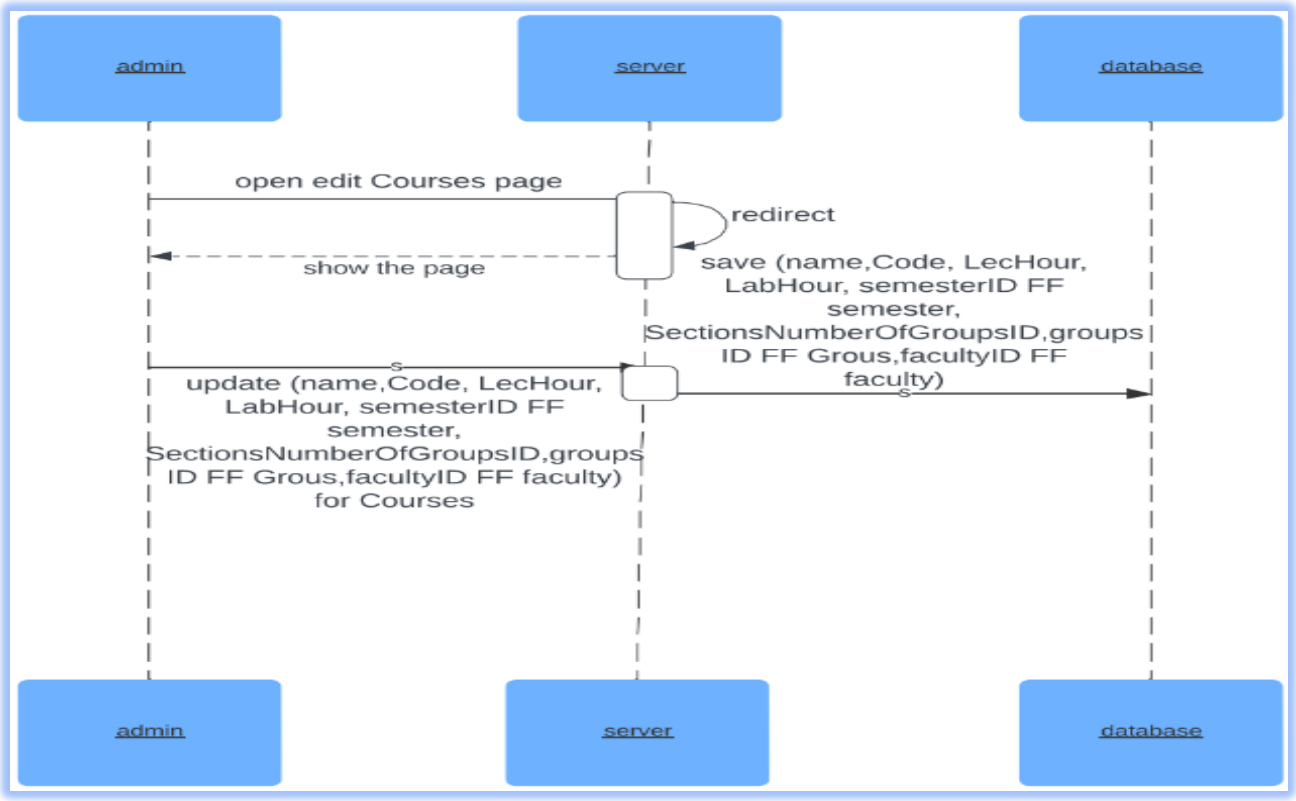

Figure 3.32 (Edit courses Sequence diagram)

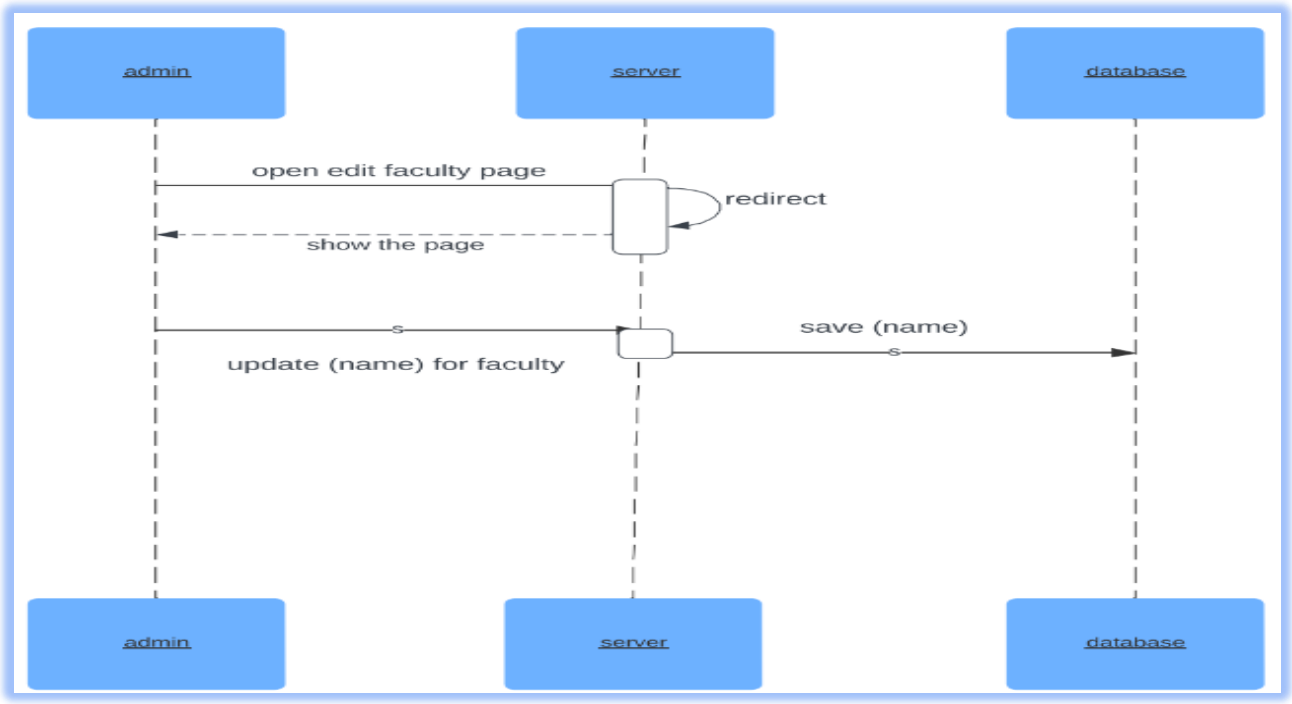

Figure 3.33(Edit Faculty Sequence diagram)

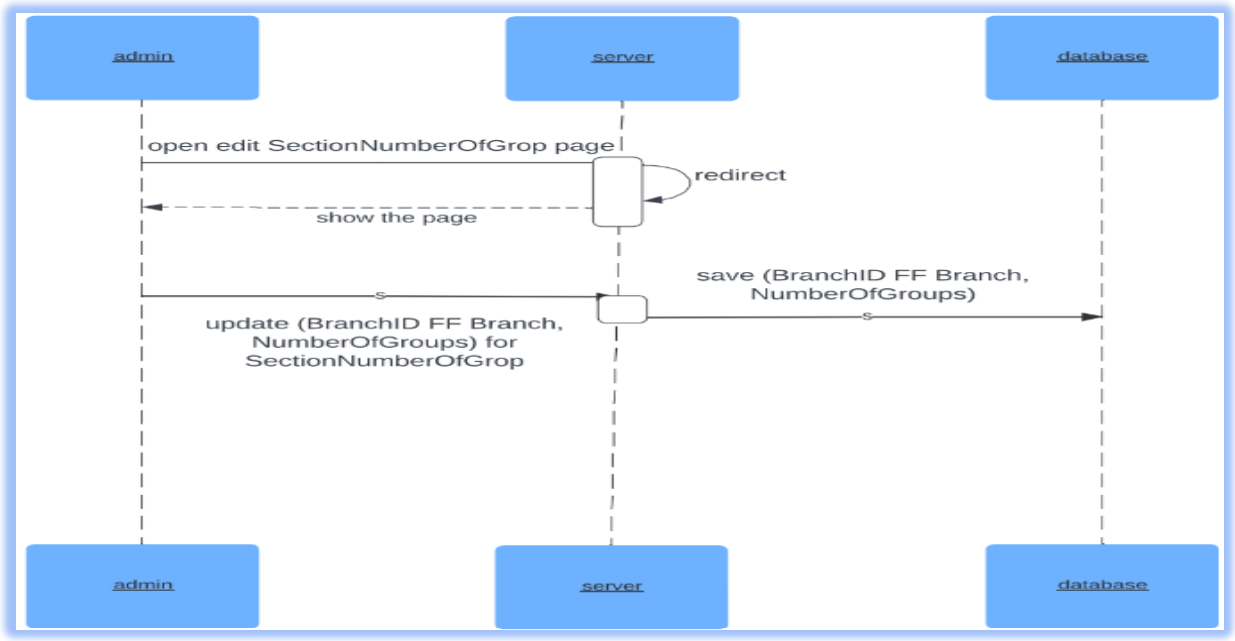

Figure 3.34 (Edit Section Number of Groups Sequence diagram)

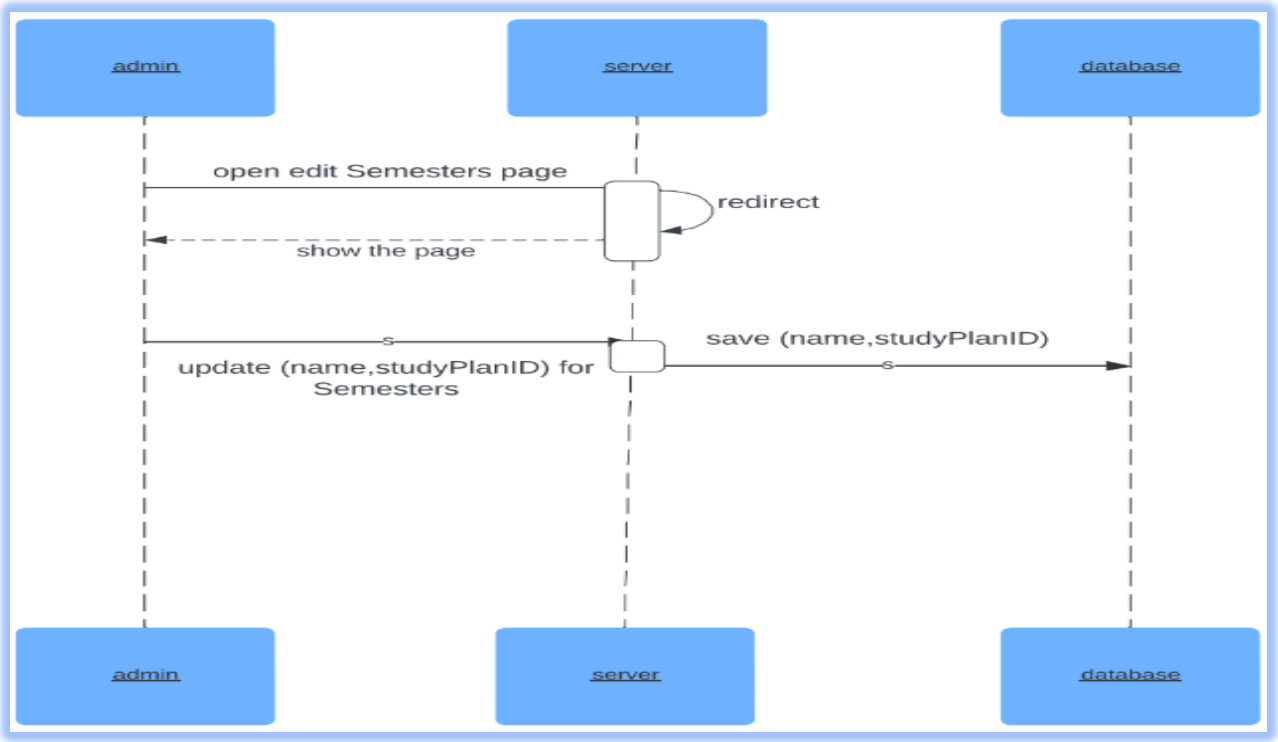

Figure 3.35 (Edit Semesters Sequence diagram)

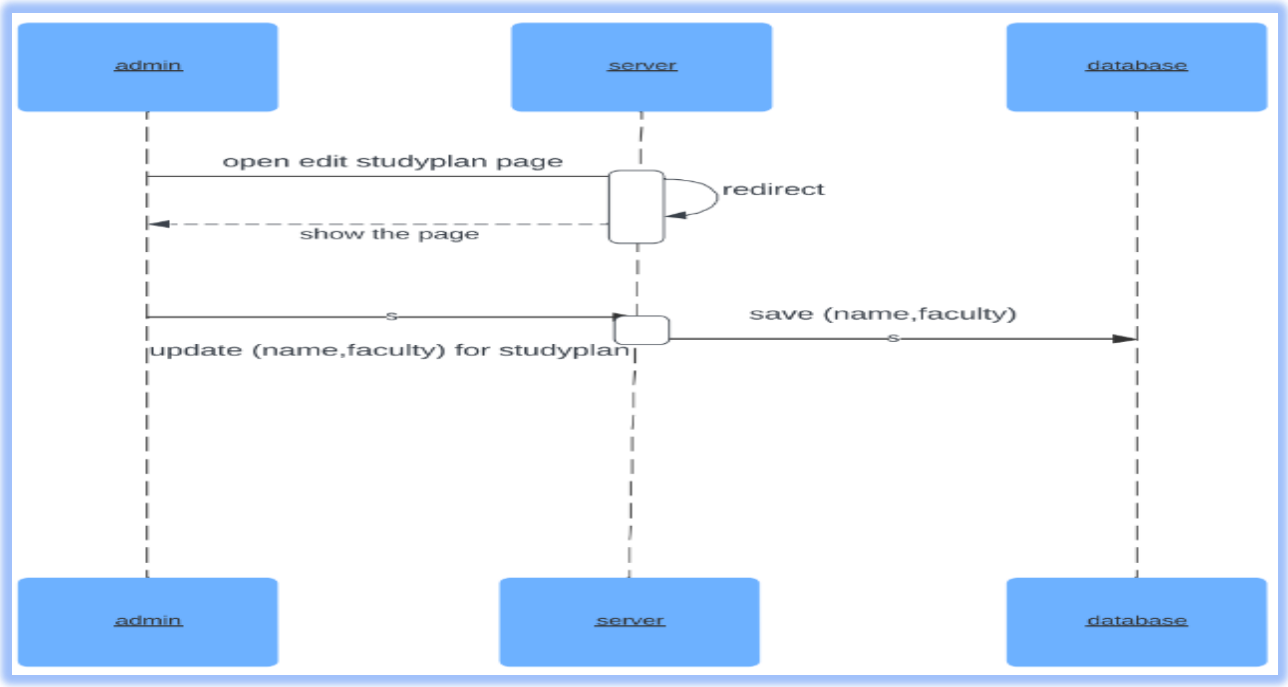

Figure 3.36 (Edit Study Plan Sequence diagram)

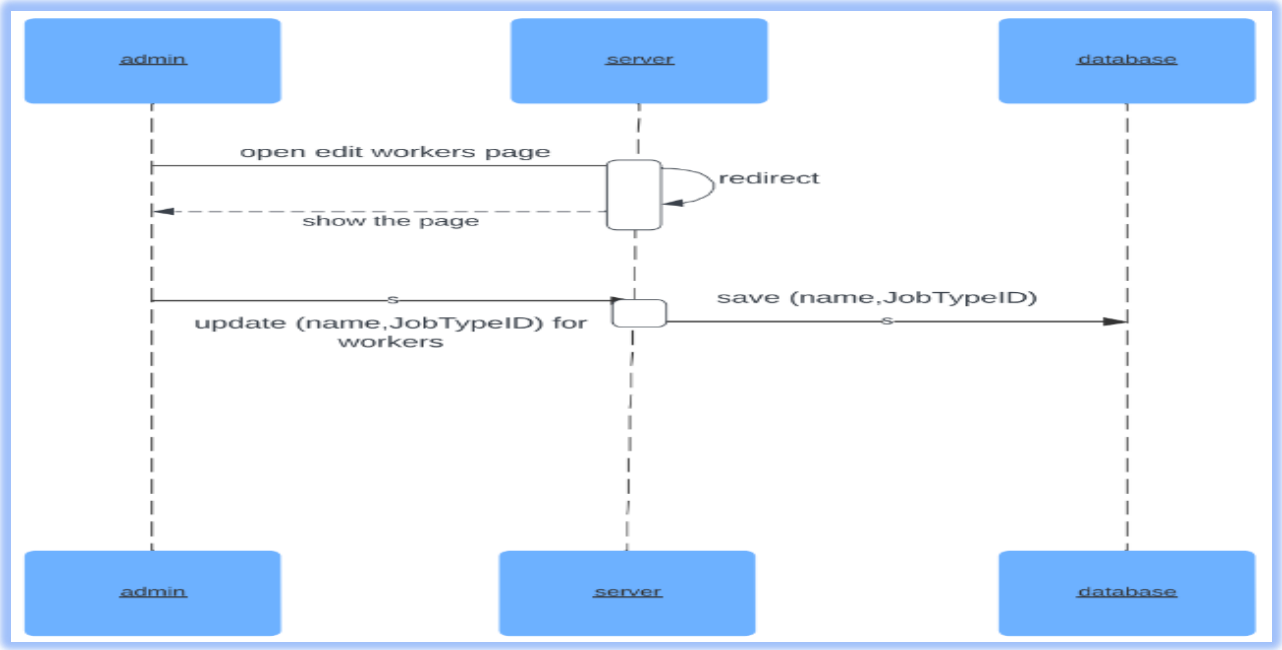

Figure 3.37 (Edit Workers Sequence diagram)

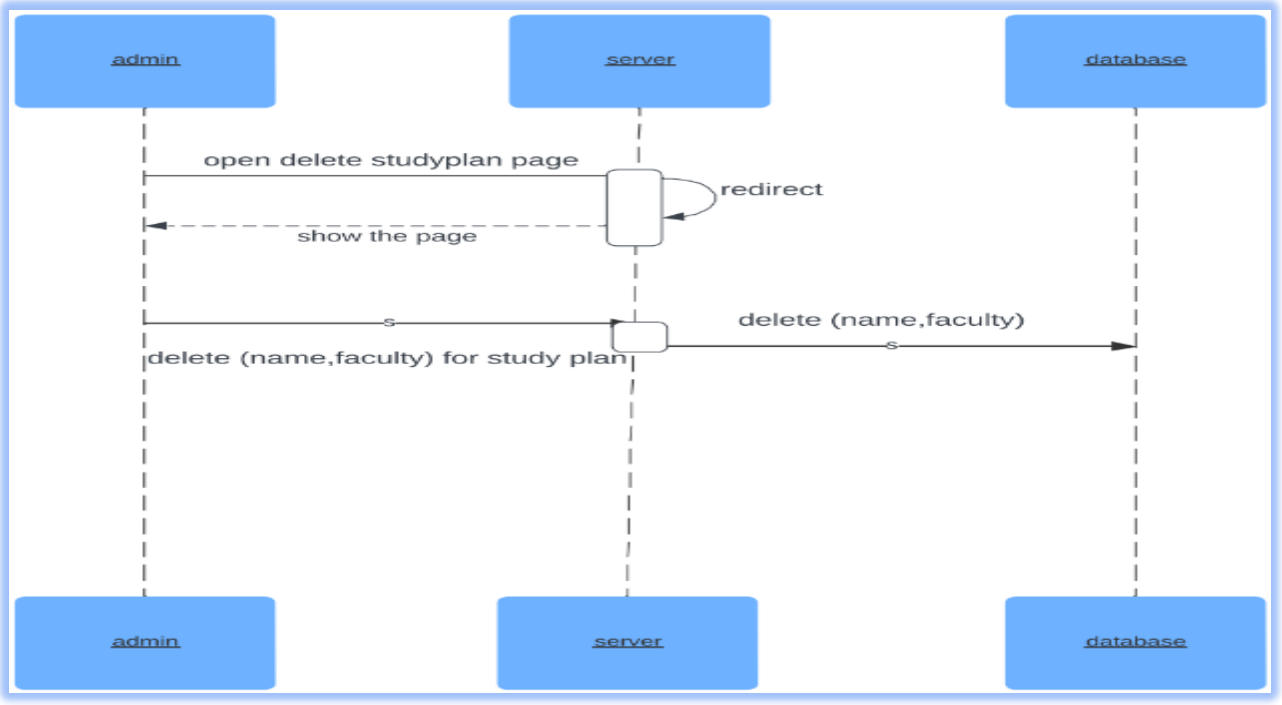

Figure 3.38 (Delete Study Plan Sequence diagram)

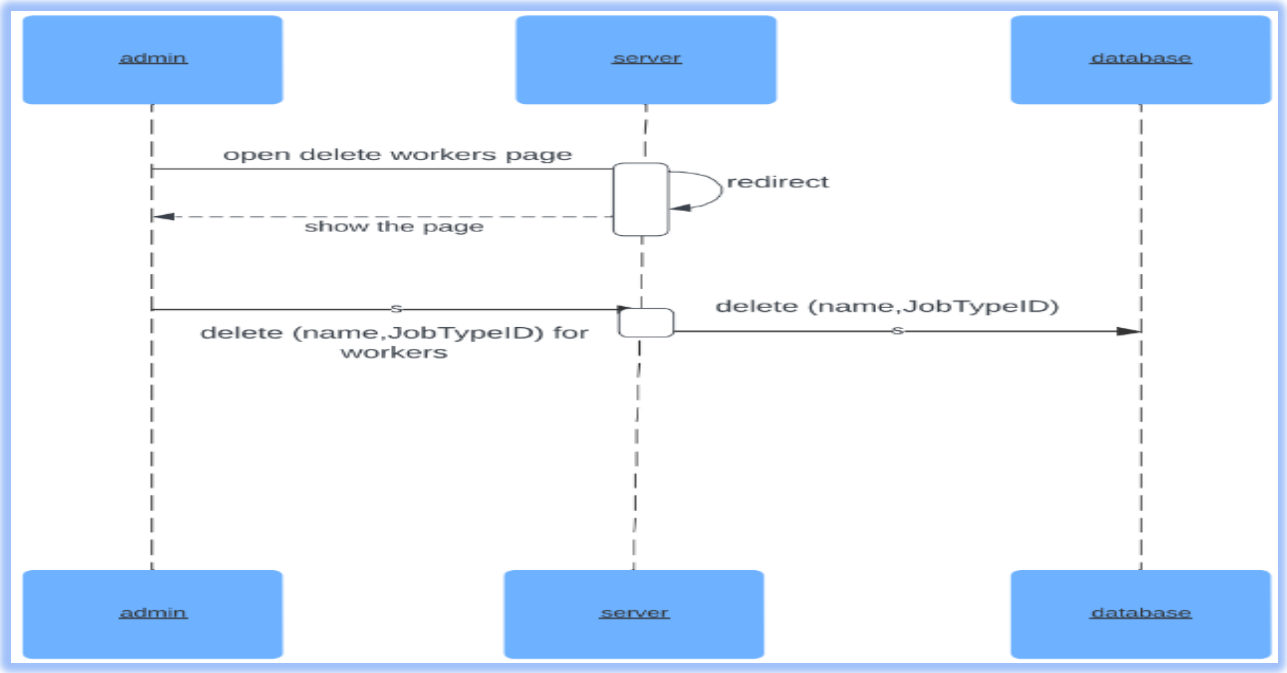

Figure 3.39 (Delete Staff Sequence diagram)

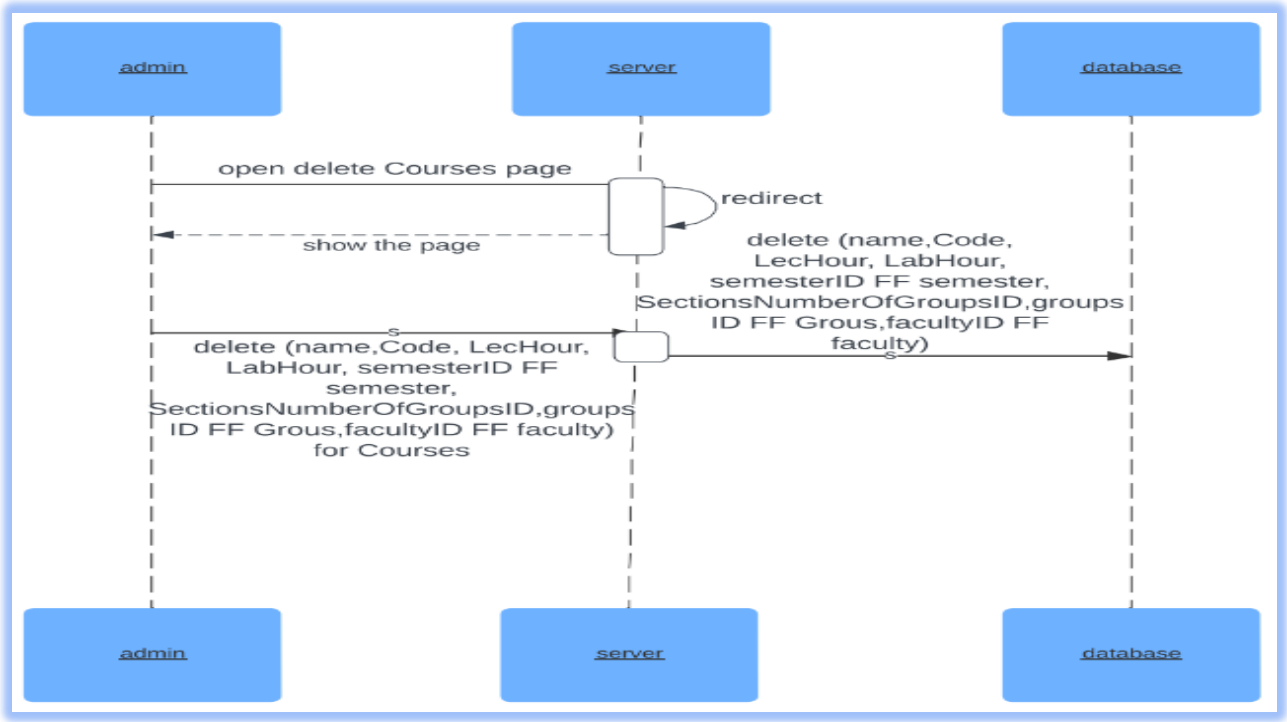

Figure 3.40 (Delete Courses Sequence diagram)

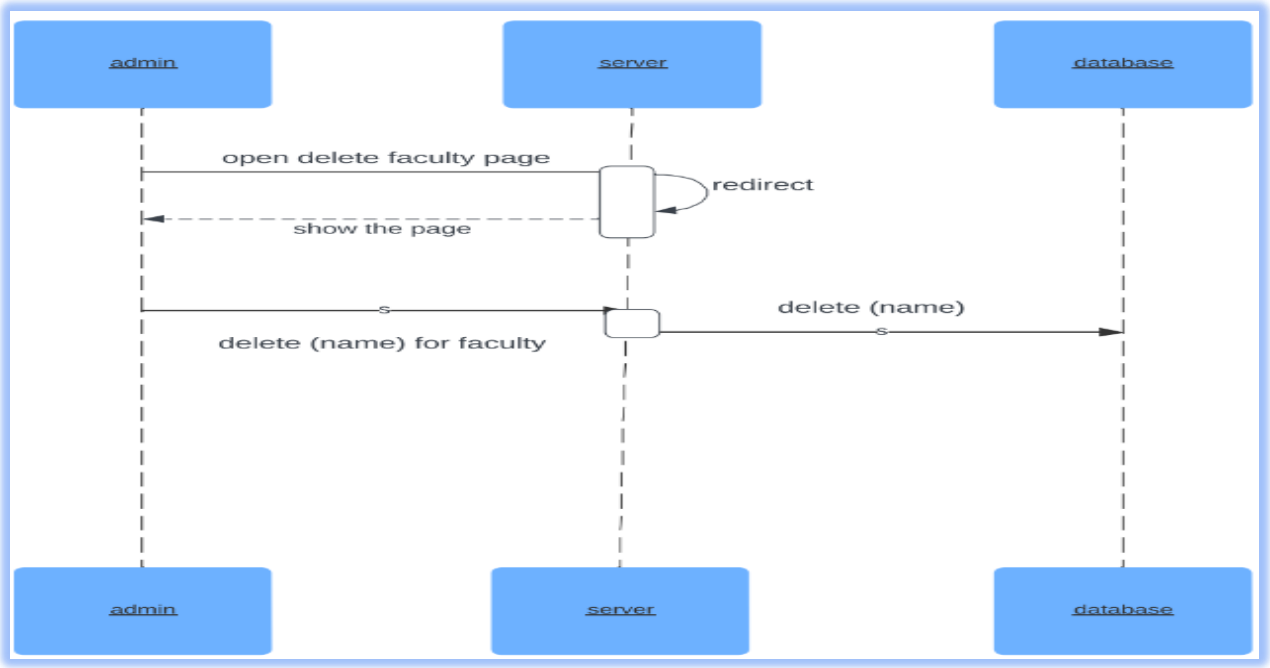

Figure 3.41 (Delete Faculty Sequence diagram)

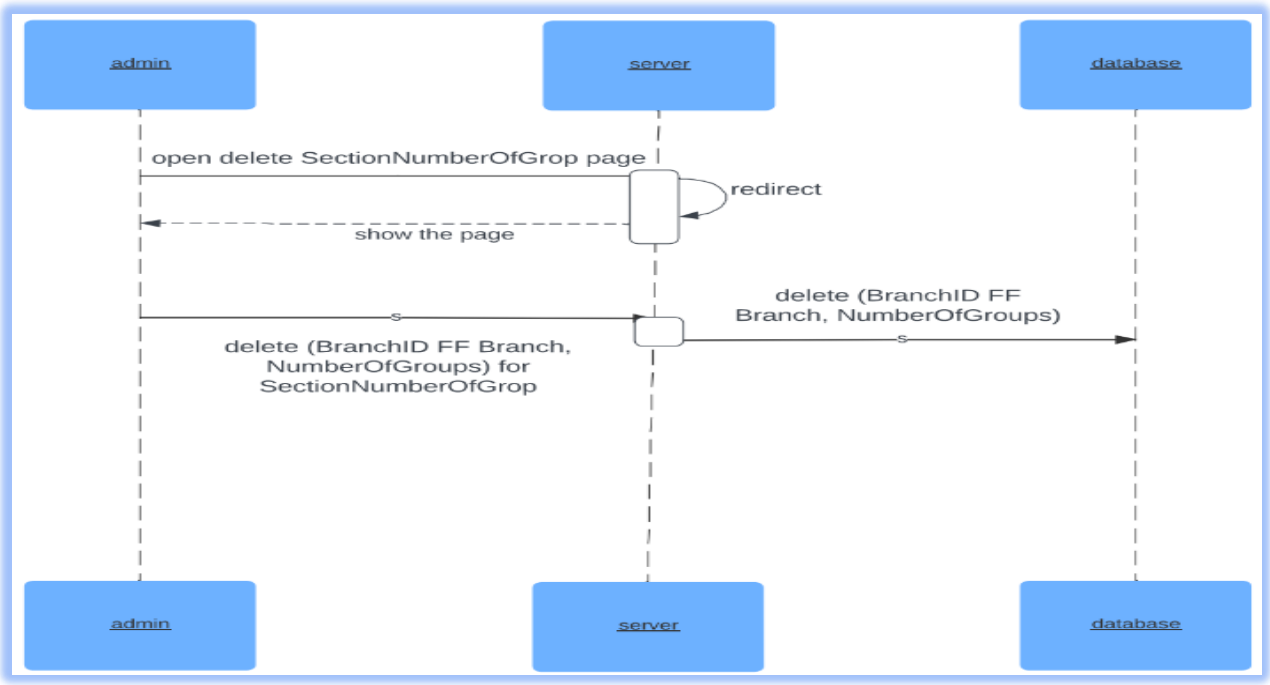

Figure 3.42 (Delete Section Number of Groups Sequence diagram)

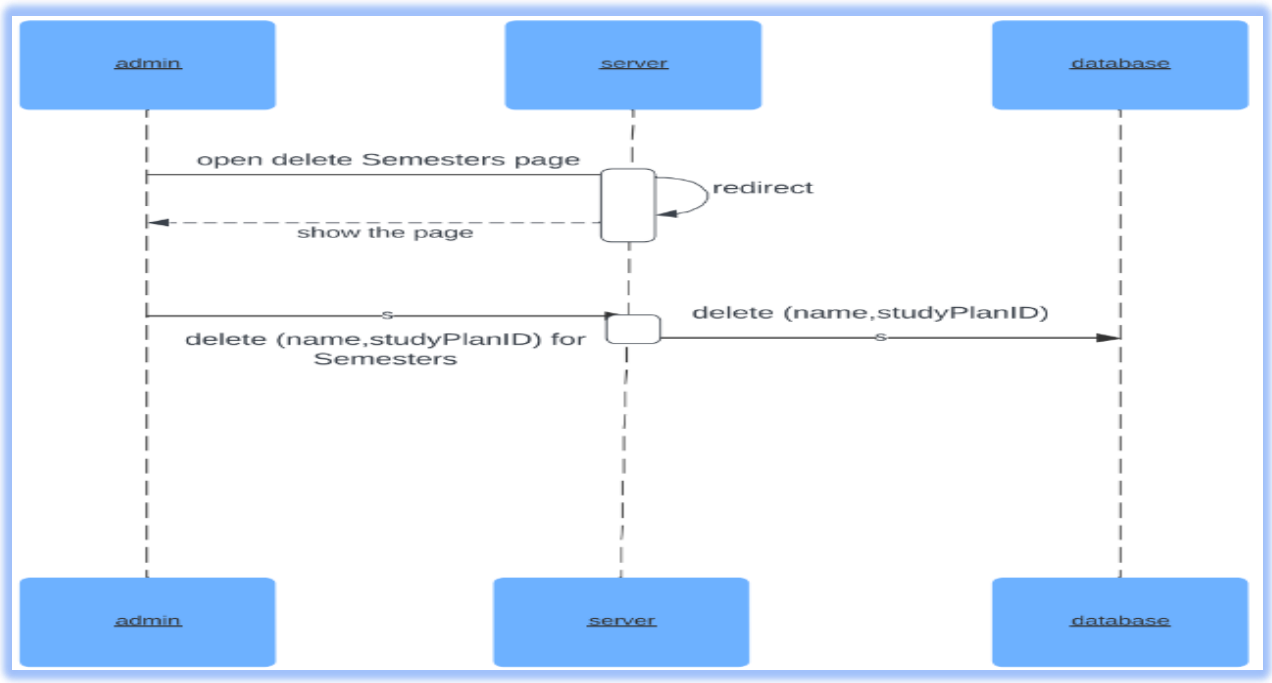

Figure 3.43 (Delete Semesters Sequence diagram)

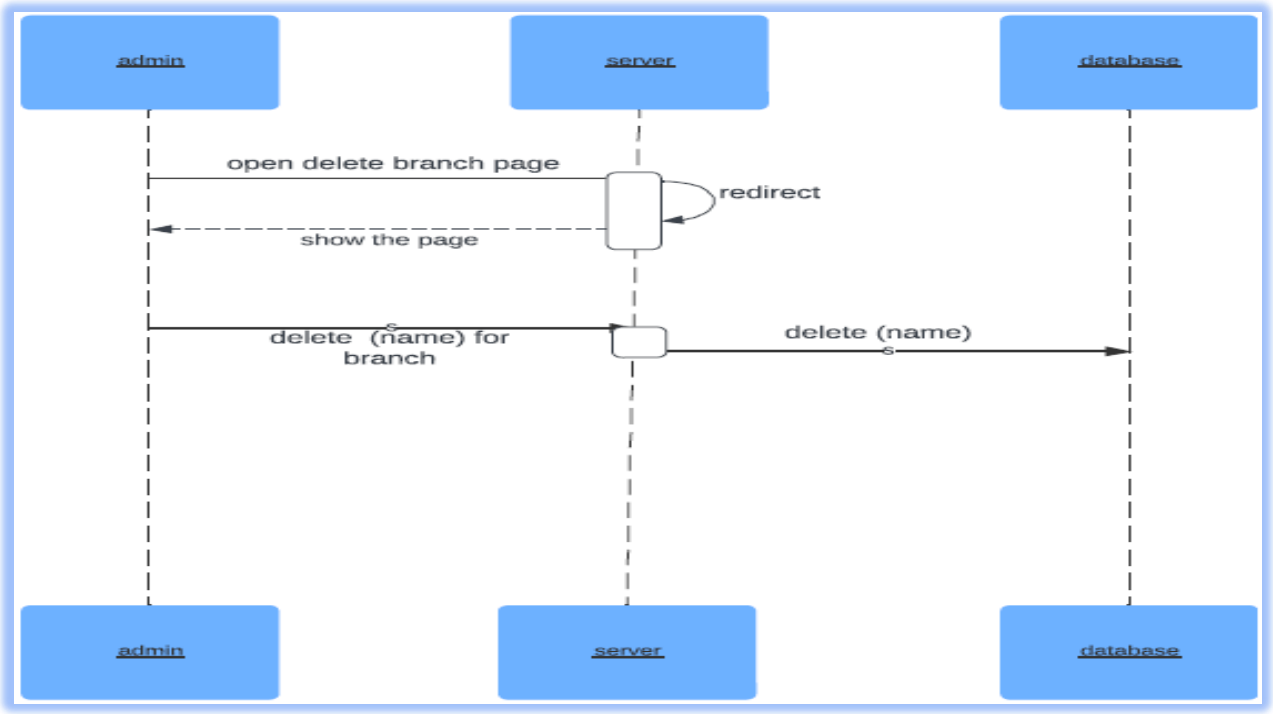

Figure 3.44 (Remove-branch Sequence diagram)

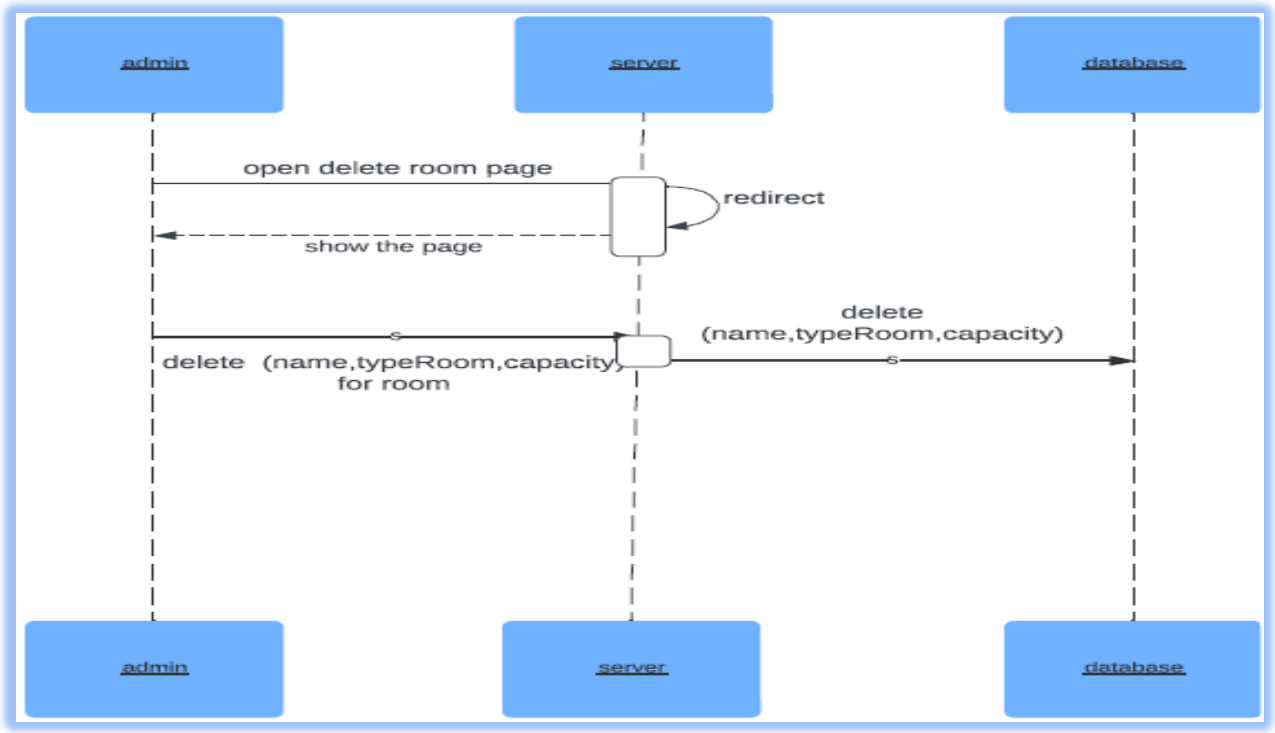

Figure 3.45 (Remove-room Sequence diagram)

3.7 Data Flow Diagram

A Data Flow Diagram (DFD) is a visualization tool that illustrates the information flow within a system, mapping out processes, data stores, data flows, and external entities, and depicting their interactions. Widely employed in system analysis and design, DFDs facilitate a comprehensive understanding and documentation of information flow dynamics within a given system.

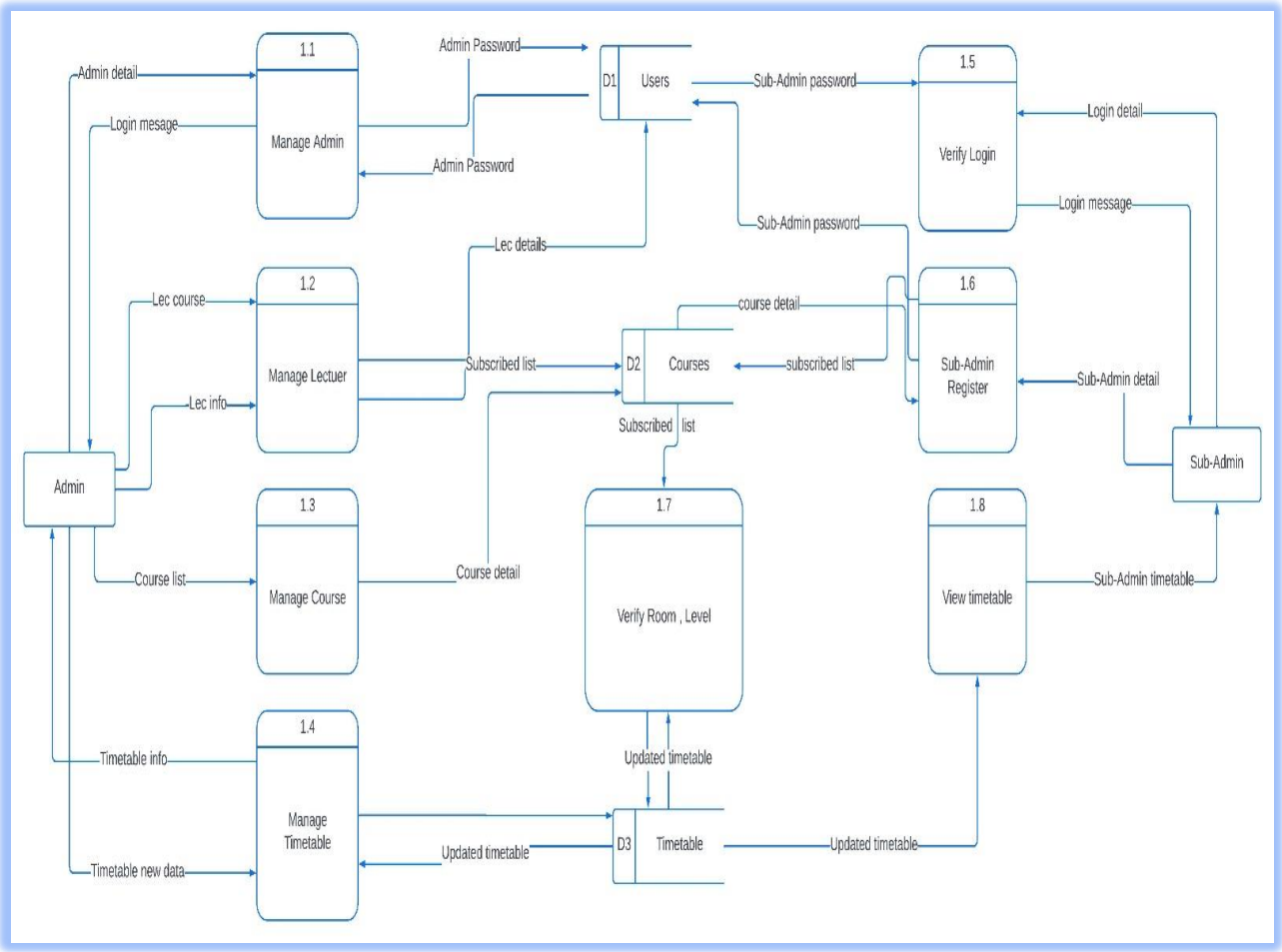

Figure 3.46 (System Data Flow Diagram)

CHAPTER 4

IMPLEMENTATION

4.1 Introduction

This chapter details the implementation of the Timetable Management System (TTMS). It covers the development environment key components, as well as the methodologies used for coding, the key components of project parts (database, frontend and backend) and how all these parts are linked together. The objective is to provide a comprehensive overview of how the system was built, the sufficient system.

4.2 Key Components Implementation

Implementation was guided by best practices in software development, ensuring a robust, scalable, and user-friendly application. By leveraging modern technologies and methodologies, the system is well-equipped to meet the needs of educational institutions in managing their timetables efficiently, the following key components will be showed in details on their implementation of database, frontend and backend:

4.2.1 Login component:

This component will cover the login process to manage user registration, login, and role-based access control.

- **Database:** here we start to create the “log” and “login” tables for storing username and password for the users.

```sql
create table if Not exists log(
    id int primary key auto_increment,
    oprationMade varchar(255) not null,
    userId int not null references staff(id)
);

create table if not exists login(
    userId int not null primary key references staff(id),
    username varchar(255) not null,
    password varchar(255) not null
);
```

Figure 4.1 (create “log” and “login” table)

- **Backend:** here the session will start if there is a valid user login the website by checking if it`s username and password was inserted to database or not.

```
public ResponseEntity<?> loginUser(@RequestBody user user, HttpSession session) {
        System.out.println("\t" + user.getUsername());

        int userRole;
        userRole = userService.userRole(user.getUsername(), user.getPassword());

        if (userRole == -1) {
            ArrayList<String> s = new ArrayList<>();
            s.add("Invalid Credentials");

            return new ResponseEntity<>(s, HttpStatus.UNAUTHORIZED);
        }

        session.setAttribute("username", user.getUsername());
        session.setAttribute("role", userRole);
        if (userRole == 2) {

        }
        ArrayList<String> s = new ArrayList<>();
        s.add("Login Successful");
        return new ResponseEntity<>(s, HttpStatus.OK);
    }
```

Figure 4.2 (login-user function)

- **Frontend:** We created a form and interfaces for login and managed using Angular.

```
export class LoginComponent implements OnInit {

  form: FormGroup = this.fb.group({
    username: ['', Validators.required],
    password: ['', Validators.required],
  });
  constructor(
    private _AuthService: AuthService,
    private fb: FormBuilder,
    private _Router: Router,
    @Inject(PLATFORM_ID) private platformId: Object,
     private http: HttpClient
  ) {
    _AuthService.loggedIn.next(false);
  }
  ngOnInit(): void {
    if (isPlatformBrowser(this.platformId)) {
      const apiUrl = 'http://localhost:7081/api/home';
      const headers = new HttpHeaders({
        'Content-Type': 'application/json',
      });
      this.http.get<any>(apiUrl, { headers, withCredentials: true }).subscribe(
        (response) => {
          if (response[0] === 'home') {
            this._Router.navigate(['/home']);
          } else {
          }
        },
        (error) => {
        }
      );
    }
  }

  login() {
    this._AuthService.login1(
      this.form.value.username,
      this.form.value.password
    );

  }
}

```

Figure 4.3 (login form implementation)

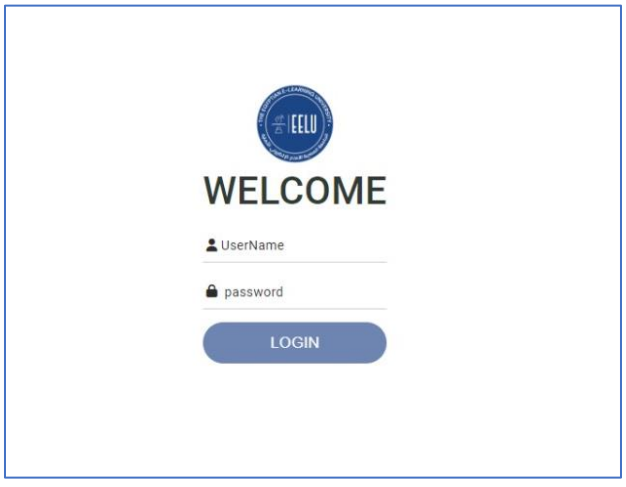

Figure 4.4 (login form interface)

4.2.2 Staff members component:

Staff members components will cover the information about members their job type, branch they belong and so on, and functions will be applied.

- **Database:** here we store information about staff members in "staffmembers" table then insert their job type; branch they belong to and so on.

```
create table if not exists staff(
    id int primary key auto_increment,
    name varchar(255) not null,
    JobTypeId int not null references JobType(id),
    branchId int not null references branch(id)
);
```

Figure 4.5 (create staff table)

```sql
insert into jobtype(name)
values('Doctor'),('TA');

insert into staff(name, JobTypeId, branchId)
values('Bassem Mohamed', 1, 5),('Walaa ElHady', 1, 5),
('Wafaa Samy', 1, 5),('Amany Magdy', 1, 5),
('Yasser Abd Elhameed', 1, 5),('Safi Shiha', 1, 5),
('Mayar Ali', 1, 5),('ElSayed ElDahshan', 1, 5),
('Shimaa Mosaad', 1, 5),('Mohamed ElMasry', 1, 5),
('mohamed Basiouny', 1, 5),('Mohamed Reda', 1, 5),
('Khaled Wasif', 1, 5),
('Marihan', 2, 5),('Norhan', 2, 5),
('Amr', 2, 5),('ElBanna', 2, 5),
('Omnia', 2, 5),('Amany', 2, 5);
```

Figure 4.6 (A sample of insertion into Job type and staff tables)

- **Backend:** we applied functions to staff data stored in database as mentioned below.

```java
void makeTheStaffRefrance(Course[] courses) {
    for (int i = 0; i < courses.length; i++) {
        for (int j = i + 1; j < courses.length; j++) {
            if (courses[i].getStaff().getId() == courses[j].getStaff().getId()) {
                courses[j].setStaff(courses[i].getStaff());
                break;
            }
        }
    }
}
```

Figure 4.7 (function staff reference in all the courses)

```
ArrayList<Staff> getStaffFreeTime(Course[] courses) {
        ArrayList<Staff> staff = new ArrayList<>();
        for (Course course : courses) {
            if (!staff.contains(course.getStaff())) {
                Staff s = course.getStaff();
                staff.add(s);
            }
        }
        staff.sort((t, t1) -> {
            return t.getId() - t1.getId();
        });
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < staff.size(); i++) {
            sb.append(staff.get(i).getId()).append(", ");
        }
        connection conn = new connection();
        try {
            int j = 0;
            ResultSet rs = conn.select("select * from freetimeforstaff where staffid in (" + sb.substring(0, sb.length() - 2) + ")  order by staffid ");
            while (rs.next()) {
                for (int i = j; i < staff.size(); i++) {
                    if (staff.get(i).getId() == rs.getInt(1)) {
                        staff.get(i).getFreeTime().addFreeTime(rs.getInt(2), rs.getInt(3), rs.getInt(4));
                        j = i;
                        break;
                    }
                }
            }
        } catch (Exception ex) {
            System.out.println(ex);
        } finally {
            conn.close();
        }
        return staff;
    }
```

Figure 4.8 (get the staff "freeTime" from the database)

- **Frontend:** create a form for get staff and manage how the user will add staff members with a friendly and easy user interface

```
getStaff() {
    this._staffService.getStaffList().subscribe({
      next: (res) => {
        const displayedData = res.map((data: {id:number;
          name: any;
          type: { name: any; id:number };
          branch: { name: any; id:number}; }) => ({
         id: data.id,
         jopname: data.name,
         tjop: data.type.name,
         bname: data.branch.name
         // 'action': This value is not present in the server response
       }));
        this.dataSource = new MatTableDataSource(displayedData);
        this.dataSource.sort = this.sort;
        this.dataSource.paginator = this.paginator;
      },
      error: console.log
    });
  }
```

Figure 4.9 (a form that get data about staff members)

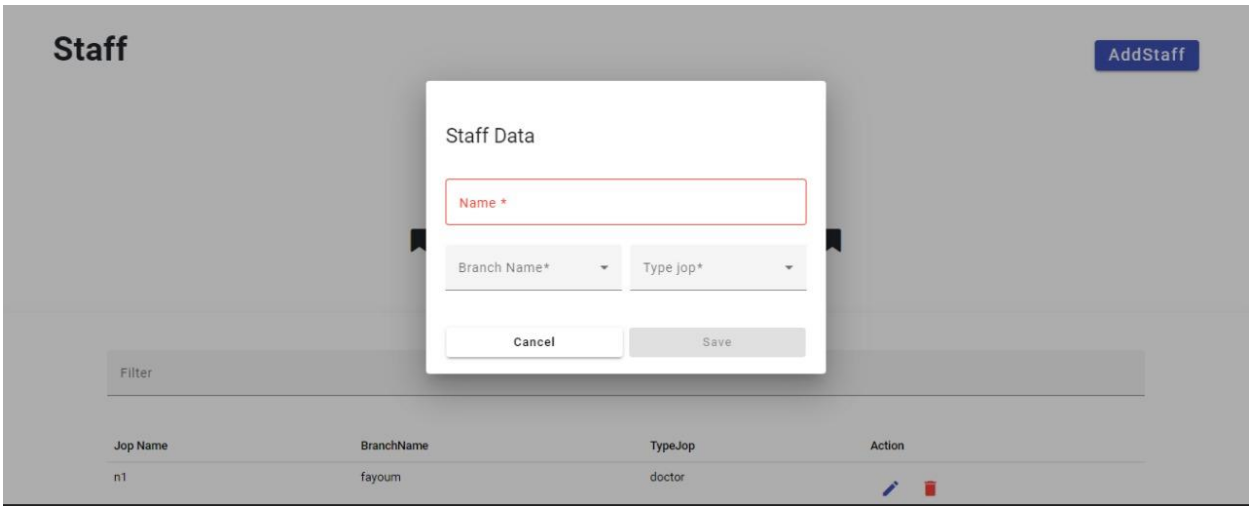

Figure 4.10 (a form to add staff members data)

4.2.3 Faculty and Courses component:

This component includes faculty and courses in the university.

- **Database:** create course and faculty tables and insert data collected to store them.

```sql
create table if not exists faculty(
    id int primary key auto_increment,
    name varchar(255) not null
);

create table if not exists courses(
    id int primary key auto_increment,
    name varchar(255) not null,
    code varchar(255) not null,
    lecHours int not null,
    labHours int not null,
    SemesterId int not null references semester(id),
    StudyplanId int not null references studyplan(id),
    facultyId int not null references faculty(id),
    lectureGroupId int not null references lecturegroups(id),
    SectionsNumberOfGroupsID int not null references SectionsNumberOfGroups(id)
);
```

Figure 4.11 (create faculty and courses table)

```sql
insert into faculty(name)
values('IT'),('BA');

INSERT INTO courses (name, code, lecHours, labHours, SemesterId, StudyplanId, facultyId, lectureGroupId, SectionsNumberOfGroupsId)
VALUES
('Advanced Software Engineering', 'CS344', 2, 1, 14, 2, 1, 1, 1),
('AI Systems Design and Implementation', 'AI343', 2, 1, 22, 3, 1, 2, 2),
('Algorithms and Data Structures', 'SWE206', 2, 1, 4, 1, 1, 6, 3),
('Big Data', 'AI361', 2, 1, 22, 3, 1, 2, 2),
('Computer Graphics', 'IT221', 2, 1, 5, 1, 1, 1, 4),
('Computer Networks (2)', 'NWE303', 2, 1, 6, 1, 1, 3, 5),
('Computer Organization (1)', 'CAS202', 2, 1, 4, 1, 1, 5, 3),
('Data Structures', 'SWE206', 2, 1, 12, 2, 1, 1, 6),
('Ethical Hacking-lab', 'LB313', 2, 2, 14, 2, 1, 1, 7),
('Information Assurance & Security', 'ITF404', 2, 1, 8, 1, 1, 1, 8),
('Information Computer Networks Security', 'IT333', 2, 1, 14, 2, 1, 1, 9),
('Integrated Information Systems', 'ITF405', 2, 1, 8, 1, 1, 1, 10),
('Intelligent Autonomous Robotics', 'AI351', 2, 1, 22, 3, 1, 2, 2),
('Intelligent Database', 'ITF303', 2, 1, 6, 1, 1, 3, 11),
('Introduction to Operation Research', 'IT217', 2, 1, 12, 2, 1, 1, 12),
('Introduction to Web Technology', 'NWE101', 2, 1, 2, 1, 1, 3, 13),
('Language Engineering', 'ITF306', 2, 1, 6, 1, 1, 4, 14),
('Logic Design', 'IT113', 2, 1, 10, 2, 1, 1, 15),
('Machine Learning Fundamentals', 'AI321', 2, 1, 13, 2, 1, 1, 12),
('Mathematics-2', 'MA113', 2, 1, 10, 2, 1, 1, 16),
('Microcontroller', 'IT343', 2, 1, 14, 2, 1, 1, 17),
('Microprocessors and Interfacing', 'CAS308', 2, 1, 6, 1, 1, 3, 18),
('Mobile and Sensor Network', 'NWE407', 2, 1, 8, 1, 1, 1, 10),
('Natural Languages Processing', 'ITF306', 2, 1, 14, 2, 1, 1, 19),
('Networking Fundamentals lab', 'LB211', 2, 2, 12, 2, 1, 1, 20),
('Neural Networks', 'ITF309', 2, 1, 8, 1, 1, 1, 21),
('Numerical methods', 'GEN207', 2, 1, 4, 1, 1, 3, 22),
('Operating Systems', 'CAS204', 2, 1, 4, 1, 1, 7, 23),
('Pattern Recognition', 'IT322', 2, 1, 14, 2, 1, 1, 24),
('Physics (2)', 'GEN109', 2, 1, 2, 1, 1, 6, 25),
('Probability and Statistics-1', 'ST121', 2, 1, 10, 2, 1, 1, 16),
('Programming Languages', 'CS112', 2, 1, 10, 2, 1, 1, 26),
('Programming Techniques 2', 'SWE102', 2, 1, 2, 1, 1, 5, 27),
('Reinforcement and Deep Learning', 'AI331', 2, 1, 22, 3, 1, 2, 2),
('Software Engineering(1)', 'SWE204', 2, 1, 4, 1, 1, 3, 28),
('Web Engineering (2)', 'NWE405', 2, 1, 7, 1, 1, 4, 29),
('Web Engineering (3)', 'NWE406', 2, 1, 8, 1, 1, 1, 30),
('Web Engineering 1', 'NWE304', 2, 1, 6, 1, 1, 3, 31),
('Web Technology', 'IT230', 2, 1, 12, 2, 1, 1, 32);
```

Figure 4.12 (insert into faculty and courses into database)

- **Backend:** collect all course data for creating lecture time table.

```
Course[] getAllCoursesDataForLectuer(List<Integer> coursesIds, Course[] courses, ArrayList<Bran
ch> allBranchesWithHostingRooms) {
     connection conn = new connection();
     ArrayList<Course> cours = new ArrayList<>();
     try {
         ResultSet rs = conn.select("select courses.id, courses.lecHours, courses.SemesterId,
 courses.lectureGroupId, coursesstaff.staffId, coursesstaff.BranchId, semester.number"
                 + " from courses "
                 + "inner join coursesstaff on coursesstaff.courseId=courses.id "
                 + "inner join semester on SemesterId=semester.id "
                 + "where  courses.id in "
                 + "(" + coursesIds.toString().substring(1, coursesIds.toString().length() - 1)
+ ")"
                 + " and coursesstaff.staffId in "
                 + "(select id from staff where JobTypeId=1)");
         int i = 0;
         while (rs.next()) {
             boolean x = false;
             for (int j = 0; j < i; j++) {
                 if (rs.getInt(1) == cours.get(j).getId()) {
                     x = true;
                 }
             }
             if (!x) {
                 courses[0] = new Course(rs.getInt(1));
                 cours.add(courses[0]);
                 courses[0].setLectureHours(rs.getInt(2));
                 courses[0].setSemester(new Semester(rs.getInt(3), rs.getInt(7)));
                 courses[0].setLectuerGroup(new LectureGroup(rs.getInt(4)));
                 courses[0].getLectuerGroup().getTheLecGroups();
                 courses[0].setLecGroups(courses[0].getLectuerGroup().getLecGroups());
```

Figure 4.13 (insert into faculty and courses into database)

- **Frontend:** implementation of course and faculty interface and the functions created to handle the operations.

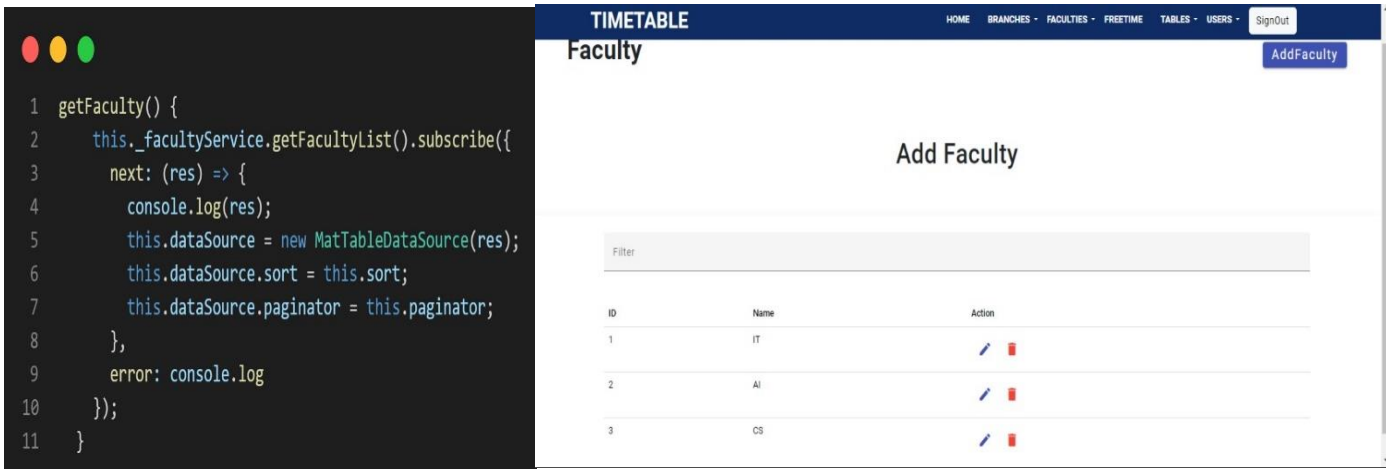

Figure 4.14 (get faculty form implementation and get faculty form)

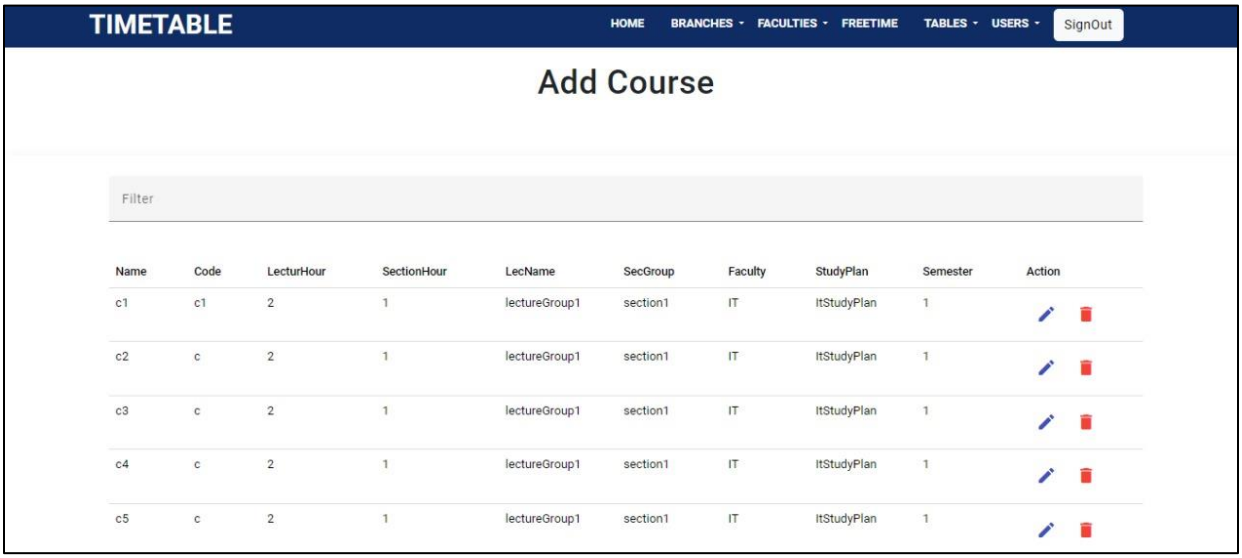

Figure 4.15 (add course form interface)

4.2.4 Lectures and sections component:

- **Database:** lectures and sections tables data were inserted with a high degree of focus because they are two of the most important tables we worked on.

```
insert into lecturegroups(name)
values('LG1'),('LG2'),('LG3'),('LG4'),('LG5'),('LG6'),('LG7');

insert into lecgroup(name,lecturegroupId)
values('A&B',1),('C&D',1),
('AI',2),('VCR1',3),('VCR2',4),('VCR3',5),('VCR4',6),('VCR5',7);

insert into lecgroupbranches(lecGroupId,branchId)
values(1,5),(1,13),(1,7),(1,3),(1,10),(1,8),
(2,4),(2,9),(2,2),(2,6),(2,11),
(3,11),(3,6),(3,2),(3,3),(3,13),(3,5),
(4,11),(4,6),(4,2),(4,9),(4,4),(4,8),(4,3),(4,13),(4,5),
(5,6),(5,2),(5,5),
(6,5),(6,8),
(7,8),
(8,5);

insert into sectionsnumberofgroups(name)
values('SG1'),('SG2'),('SG3'),('SG4'),('SG5'),('SG6'),('SG7'),('SG8'),('SG9'),('SG10'),
('SG11'),('SG12'),('SG13'),('SG14'),('SG15'),('SG16'),('SG17'),('SG18'),('SG19'),('SG20'),
('SG21'),('SG22'),('SG23'),('SG24'),('SG25'),('SG26'),('SG27'),('SG28'),('SG29'),('SG30'),('SG31');

```

Figure 4.16 (lectures and sections groups insertion)

- **Backend:** Here are the functions to handle operations for lectures and sections timetables creation.

```
public void createLectureTimetable(Course... courses) {
    ArrayList<Branch> allBranchesWithHostingRooms = Branch.getAllBranchesWithHostingRooms();
    courses = getAllCoursesDataForLectuer(courses, allBranchesWithHostingRooms);
    ArrayList<Branch> branchsWithRooms = Branch.getAllBranchesWithAllRooms();
    makeTheLecGroupBranchesRefrence(courses, branchsWithRooms);
    makeTheStaffRefrance(courses);
    ArrayList<Staff> staff = getStaffFreeTime(courses);
    setHostingRoomsInDaysForStaff(staff);
    makeHostingRoomsReference(courses);
    getRoomsFreeTime(branchsWithRooms);
    ArrayList<Semester> semesters = getSemesters(courses);
    makeBranchesForEachSemester(semesters, branchsWithRooms);
    assineValuesToStaffIsSymmetricAndFreeTimeInDays(staff);
    RemoveHostingBranchesFromLecGroups(courses);
    makeTheLecFreeTimeRefranceInEachSemester(semesters);
    splitedSemestersWithDays s = splitTheSemesterIntoDays(semesters);
    if (!cheekIfallSemestersWereAddAndRemoveTheAddedSemesters(s, semesters)) {
        System.out.println("not all the semesters were added");
    }
    addTheLectureCoursesToTheTimeTable(s);
    System.out.println(timesInTimetable.size());
    System.out.println(courses.length);
    System.out.println("            ----------                    ");
}
```

Figure 4.17 (create lecture function)

```
public void createSectionTimetable(Timetable timetable, Integer branchId, Integer... coursesIds) {
        Course[] courses = new Course[coursesIds.length];
        List<Integer> a = Arrays.asList(coursesIds);
        timetable.setTimesInTimetable(getThePreTimesInTimetable(timetable.getId()));
        courses = getAllCoursesDataForSections(a, courses, branchId);
        courses = getTheSectionsWithMoreThanOneStaff(a, courses, branchId);
        Branch branch = new Branch(branchId);
        branch.getAllRooms();
        getRoomsFreeTime(branch);
        ArrayList<Semester> semesters = getSemesters(courses);
        subtractTheSemesterFreeTimeFromTheLectureTimetable(timetable, semesters);
        makeTheStaffRefrance(courses);
        ArrayList<Staff> staff = getStaffFreeTime(courses);
        makeSectionGroupsFreeTimeRefranceInASemster(semesters);
        makeTheBranchRefranceToStaff(branch, courses);
        //      SubtractFreeTimeForRoomsFromTheTimetable(branch, timetable);
        getTheRoomTypeForCourses(a, courses);
        timesInTimetable = new ArrayList<>();
        getTheSemesterWorkingDaysAndTryToAddThem(semesters);
        addTheRemainingCoursesToSectionTimeTable(semesters);
        timesInTimetable.sort((o1, o2) -> {
            return o1.getDay() - o2.getDay();
        });
        System.out.println(timesInTimetable.size());
        System.out.println(courses.length);
        System.out.println("           ----------                ");
    }
```

Figure 4.18 (create section function)

- **Frontend:** here we select the courses and semester also the name.

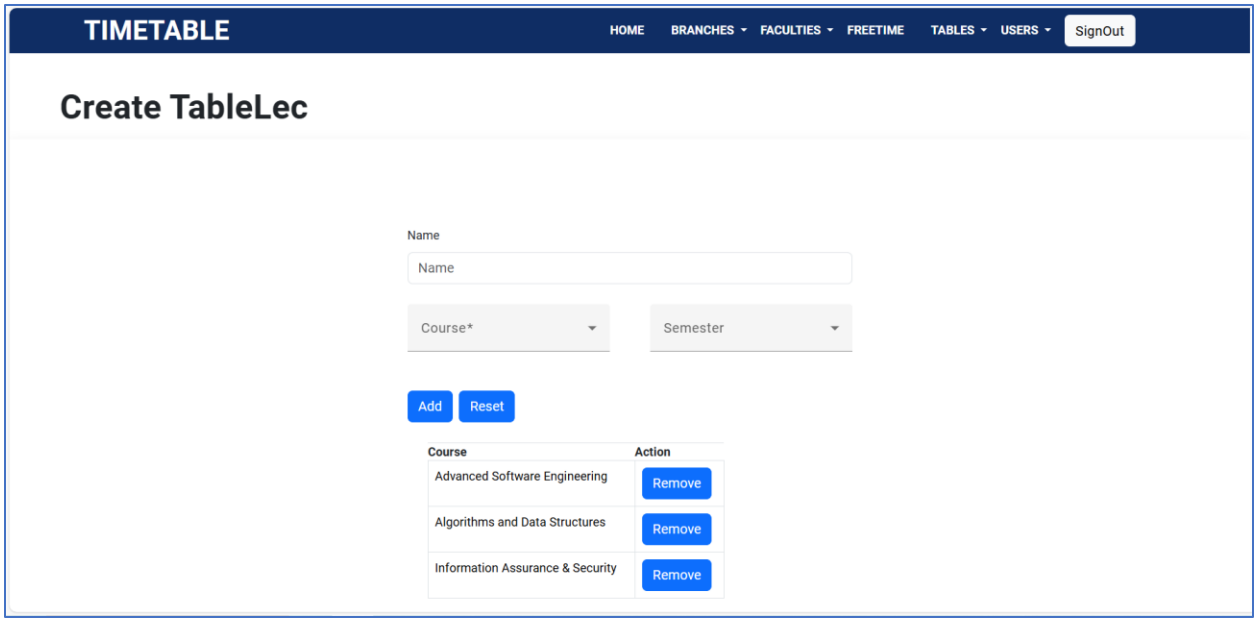

Figure 4.19 (create lecture table)

4.2.5 Study plan and Semester component:

In study plan component it was implemented to handle different study plans, Semester component implemented to handle courses in the semester.

- **Database:** we created semester and study plan tables to store and insert data collected, and this is the creation and a sample of insertion of data collected.

```sql
create table if not exists studyplan(
    id int primary key auto_increment,
    name varchar(255) not null,
    facultyId int references faculty(id)
);

create table if not exists semester(
    id int primary key auto_increment,
    number int not null,
    StudyPlanId int not null references studyplan(id)
);
```

Figure 4.20 (create semester and study plan tables)

```sql
insert into studyplan(name, facultyId)
values('IT_old', 1),('IT',1),('AI',1);

insert into semester(number, StudyPlanId)
values(1, 1),(2, 1),(3, 1),(4, 1),(5, 1),(6, 1),(7, 1),(8, 1),
(1, 2),(2, 2),(3, 2),(4, 2),(5, 2),(6, 2),(7, 2),(8, 2),
(1, 3),(2, 3),(3, 3),(4, 3),(5, 3),(6, 3),(7, 3),(8, 3);
```

Figure 4.21 (insert data into semester and study plan tables)

- **Backend: "**getSemester" function is implemented to set semester courses with reference to each other, and study plan class that contains study plan function used in the operation of creating timetable.

```
ArrayList<Semester> getSemesters(Course[] courses) {
        ArrayList<Semester> s = new ArrayList<>();
        for (Course course : courses) {
            for (int j = 0; j <= s.size(); j++) {
                if (j == s.size()) {
                    Semester sem = new Semester(course.getSemester().getId());
                    s.add(sem);
                    sem.getCourses().add(course);
                    course.setSemester(sem);
                    break;
                } else if (course.getSemester().getId() == s.get(j).getId()) {
                    s.get(j).getCourses().add(course);
                    course.setSemester(s.get(j));
                    break;
                }
            }
        }

        for (Course course : courses) {
            for (int i = 0; i < course.getOtherSemster().size(); i++) {
                for (int j = 0; j < s.size(); j++) {
                    if (course.getOtherSemster().get(i).getId() == s.get(j).getId()) {
                        course.getOtherSemster().set(i, s.get(j));
                    }
                }
            }
        }
        return s;
    }
```

Figure 4.22 (get semester function)

```java
public class StudyPlan {

    private int id;
    private String name;
    private Faculty faculty;

    public StudyPlan(int id, String name, int facultyId,String facultyName) {
        this.id = id;
        this.name = name;
        this.faculty = new Faculty(facultyId, facultyName);
    }
```

Figure 4.23 (study plan class)

- **Frontend:** allow user to add, edit and remove the study plan and semester with a friendly user interface.

```
getStudy() {
    this._studyervice.getStudyitList().subscribe({
      next: (res) => {
        console.log(res);
        const displayedData = res.map((data: {
          id: number;
          name: any;
          faculty: { id: number, name: any }
        }) => ({
          sname: data.name,
          fname: data.faculty.name,
          sId:data.id,
          fId:data.faculty.id
          // 'action': This value is not present in the server response
        }));
        this.dataSource = new MatTableDataSource(displayedData);
        this.dataSource.sort = this.sort;
        this.dataSource.paginator = this.paginator;
      },
      error: console.log
    })
  }
```

```
getSemster() {
   this._semservice.getSemsteritList().subscribe({
     next: (res) => {
       console.log(res);
       const displayedData = res.map((data: {
         id: number;
         number: number;
         studyPlan: { name: any; id: number; faculty: { name: any; id: number; } };
       }) => ({
         sId: data.id,
         snumber: data.number,
         studyname: data.studyPlan.name,
         ssId: data.studyPlan.id
       }));

       this.dataSource = new MatTableDataSource(displayedData);
       this.dataSource.sort = this.sort;
       this.dataSource.paginator = this.paginator;
     },
     error: console.log
   })
 }
```

Figure 4.24 (get study-plan data and get semester data implementation)

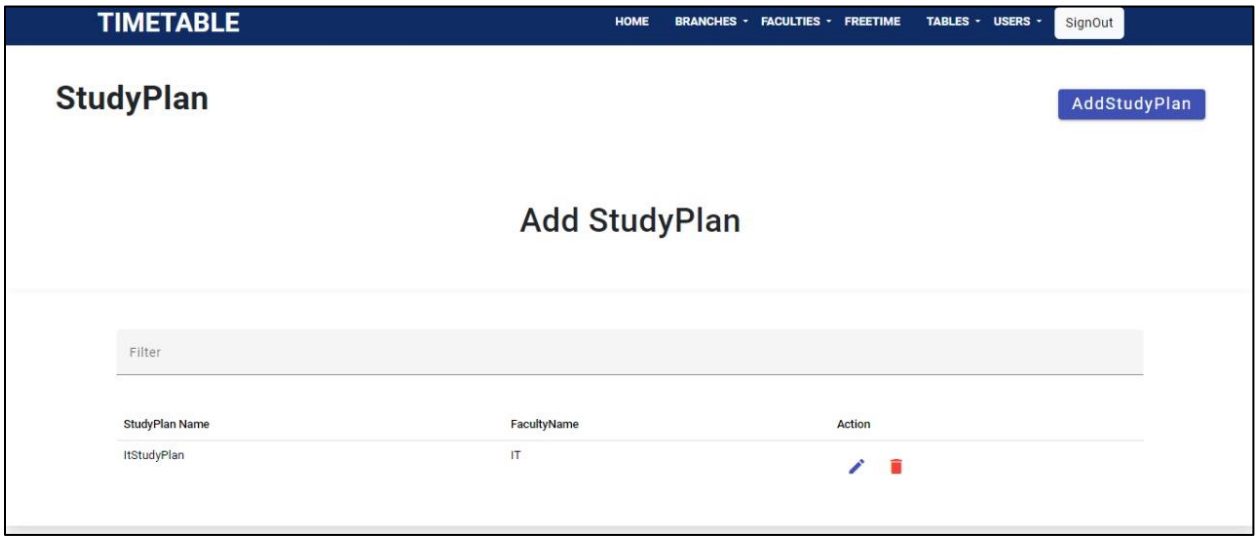

Figure 4.25 (study-plan form)

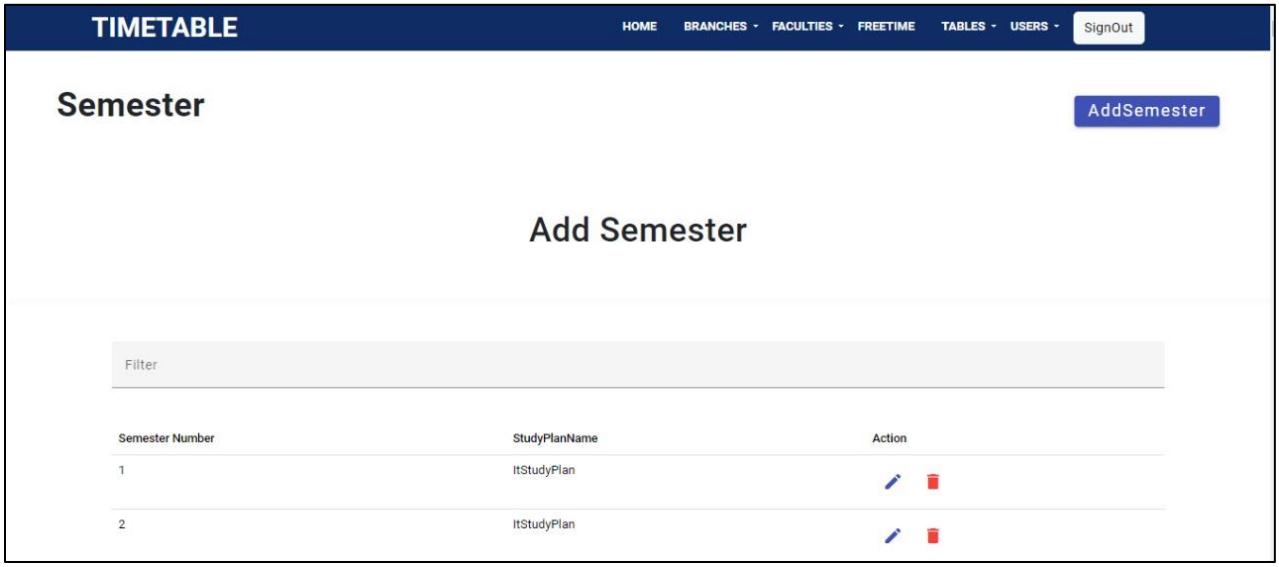

Figure 4.26 (semester form)

4.2.6 Rooms and branches components:

Rooms in each branch not having the same type as The type of room varies according to the academic content.

- **Database:** Stores information about the rooms available for scheduling, branches are in the university, and insertion of them.

```sql
create table if not exists branch(
    id int primary key auto_increment,
    name varchar(255)not null
);

create table if not exists rooms(
    id int primary key auto_increment ,
    name varchar(255) not null,
    capacity int not null,
    TypeId int not null references roomsTypes,
    branchId int not null references branch(id)
);
```

Figure 4.27 (create rooms and branches tables)

```sql
insert into branch(name)
values('Dokki'),('Fayum'),('Assuit'),('Alex'),('Ain Shams'),('Menoufia'),('Qena'),
('suhag'),('Beni Suef'),('Hurgada'),('Asmalia'),('sadat'), ('aswan'),('Tanta');

insert into rooms(name, capacity, TypeId, branchId)
values('Room 2', 60, 3, 1),
('Room 4', 100, 3, 1),('Room 5', 60, 3, 1),('Room 6', 50, 3, 1),('Room 7', 50, 3, 1),('Room 8', 200, 3, 1),
('Lab 1', 40, 2, 1),('Lab 2', 30, 2, 1);

insert into rooms(name, capacity, TypeId, branchId)
values('Room 4', 100, 3, 2),('Room 8', 250, 3, 2),('Room 16', 100, 3, 2),
('Lab 1', 30, 2, 2),('Lab 2', 30, 2, 2),('Lab 3', 30, 2, 2),('Lab 4', 30, 2, 2),('Lab 5', 30, 2, 2);
```

Figure 4.28 (rooms and branches tables insert)

- **Backend:** each branch contains a number of rooms with different types, therefor it takes a number of functions here is a sample of this functions.

```
1  static void getRoomsFreeTime(ArrayList<Branch> branchs) {
2          ArrayList<Room> rooms = new ArrayList<>();
3          StringBuilder sb = new StringBuilder();
4          for (int i = 0; i < branchs.size(); i++) {
5              for (int j = 0; j < branchs.get(i).getRooms().size(); j++) {
6                  rooms.add(branchs.get(i).getRooms().get(j));
7                  sb.append(branchs.get(i).getRooms().get(j).getId()).append(", ");
8              }
9          }
10         rooms.sort((o1, o2) -> {
11             return o1.getId() - o2.getId();
12         });
13         connection conn = new connection();
14         try {
15             int j = 0;
16             ResultSet rs = conn.select("select * from freetimeforrooms where roomid in (" +
   sb.substring(0, sb.length() - 2) + ") order by roomId");
17             while (rs.next()) {
18                 for (int i = j; i < rooms.size(); i++) {
19                     if (rooms.get(i).getId() == rs.getInt(1)) {
20                         rooms.get(i).getFreeTime().addFreeTime(rs.getInt(2), rs.getInt(3),
   rs.getInt(4));
21                         j = i;
22                         break;
23                     }
24                 }
25             }
26         } catch (Exception ex) {
27             System.out.println(ex);
28         } finally {
29             conn.close();
30         }
31     }
```

Figure 4.29 (get the “freeTime” for rooms in one branch)

Frontend: Enables the management of rooms information, and branches.

```
getBranch(){
  this._brnService.getBranchList().subscribe({
      next: (res) => {
        this.dataSource = new MatTableDataSource(res);
        this.dataSource.sort = this.sort;
        this.dataSource.paginator = this.paginator;
      },
      error: console.log
    });}
applyFilter(event: Event) {
  const filterValue = (event.target as HTMLInputElement).value;
  this.dataSource.filter = filterValue.trim().toLowerCase();

  if (this.dataSource.paginator) {
    this.dataSource.paginator.firstPage();
  }}
deleteBranch(id: number){
      const body = { id };
      this._brnService.removeBranch( body).subscribe({
        next: (res) => {
          alert('done');
        },
        error: console.log
      }); }
openEditBranch(data: any){
 const dialogRef = this._dialog.open(FormbranchComponent,{
    data,
  });
  dialogRef.afterClosed().subscribe({
    next: (val ) =>{
      if(val){
        this.getBranch();
      }}}}}
```

Figure 4.30 (get branch data implementation)

```
getRoom() {
     this._roomService.getRoomList().subscribe({
       next: (res) => {
         const displayedData = res.map((rooom: {id:number;
            name: any;
             capacity: any;
              roomtype: { name: any;id:number };
               branch: { name: any; id:number}; }) => ({
           id:rooom.id,
           roomname: rooom.name,
           capacity: rooom.capacity,
           troom: rooom.roomtype.name,
           bname: rooom.branch.name
         }));
         this.dataSource = new MatTableDataSource(displayedData);
         this.dataSource.sort = this.sort;
         this.dataSource.paginator = this.paginator;
       },
       error: console.log
     });
   }

```

Figure 4.31 (get room data implementation)

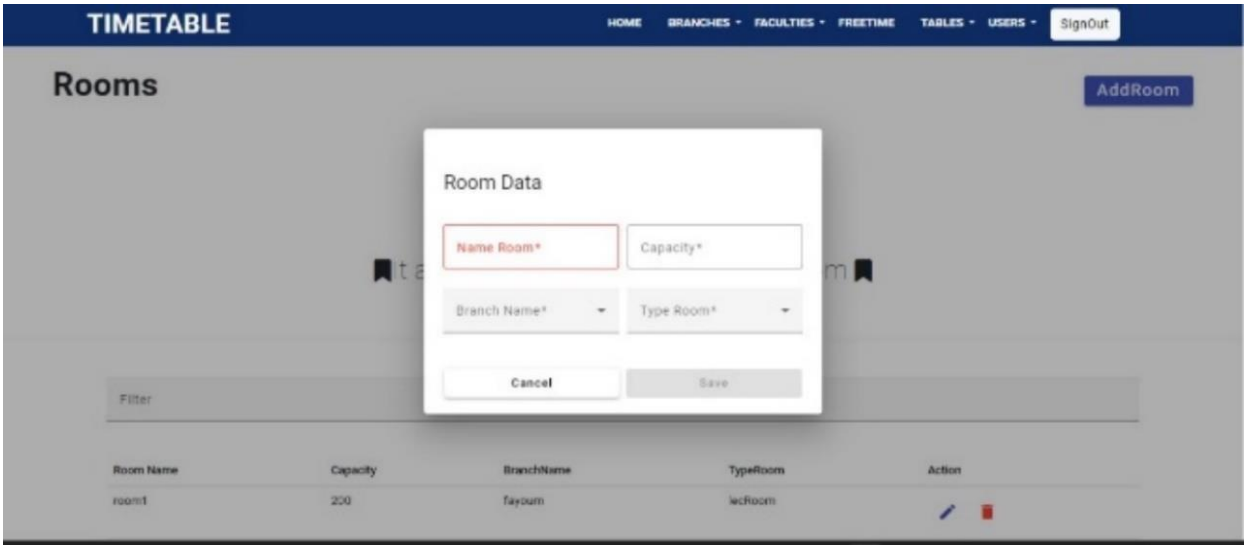

Figure 4.32 (room data form)

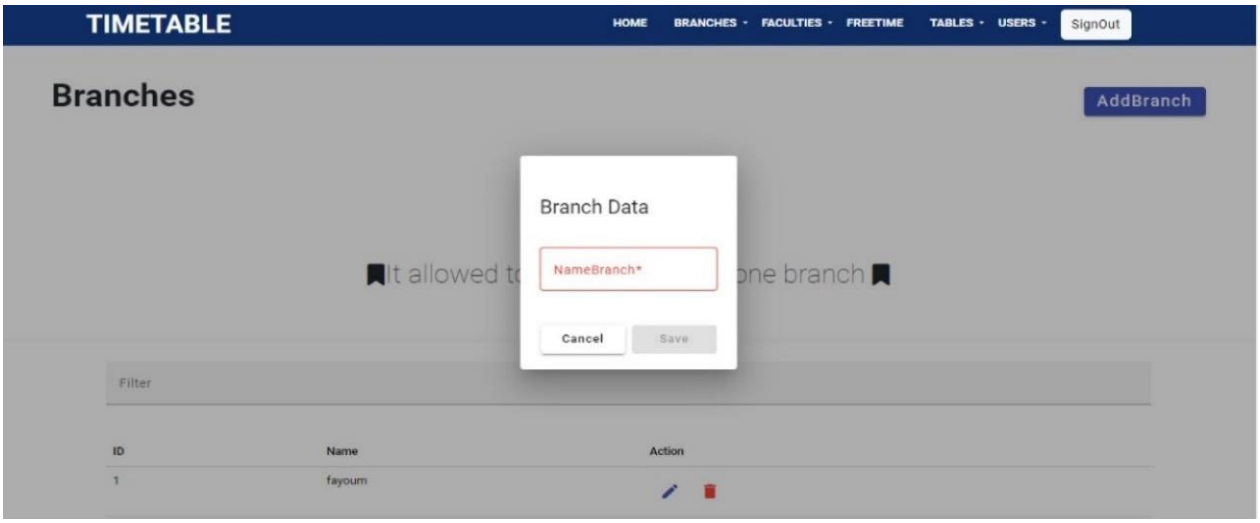

Figure 4.33 (branch data form)

4.2.7 Timetable components:

Here we collect all previous tables, functions to make the final time table to view it.

- **Database:** create table “timesintimetable” and “timetable” with reference from other tables in database and insert the data collected.

```sql
create database if not exists timeTableManagementSystem;
use timeTableManagementSystem;

create table if not exists branch(
    id int primary key auto_increment,
    name varchar(255)not null
);

create table if not exists login(
    userId int not null primary key references staff(id),
    username varchar(255) not null,
    password varchar(255) not null
);

```

Figure 4.34 (create table “timesintimetable” and timetable)

- **Backend:** function to get all data for timetable from database.

```
public ArrayList<timeInTimetable> getATimetable(int timetableId) {
        ArrayList<timeInTimetable> tims = new ArrayList<>();
        StringBuilder sb = new StringBuilder();
        ResultSet rs = conn.select("select timesintimetable.id,"
                + "timesintimetable.staffId,"
                + " staff.name,"
                + " timesintimetable.courseId,"
                + " courses.name,"
                + "courses.code,"
                + "timesintimetable.hostingBranchId,"
                + "branch.name,"
                + "timesintimetable.hostingRoomId,"
                + "rooms.name,"
                + "timesintimetable.dayid,"
                + "timesintimetable.startingTime,"
                + "timesintimetable.enddingTime,"
                + "timesintimetable.lecGroupId, "
                + "lecgroup.name "
                + "from timesintimetable "
                + "LEFT JOIN staff on staff.id=timesintimetable.staffId "
                + "left join courses on courses.id=timesintimetable.courseId "
                + "left join branch on branch.id=timesintimetable.hostingbranchId "
                + "left join rooms on rooms.id=timesintimetable.hostingRoomId "
                + "left join lecgroup on lecgroup.id=timesintimetable.lecgroupId "
                + "order by timesintimetable.id");
        try {
            while (rs.next()) {
                sb.append(rs.getInt(1)).append(",");
                timeInTimetable t = new timeInTimetable(rs.getInt(1),
                        new Staff(rs.getInt(2), rs.getString(3)),
                        new course(rs.getInt(4), rs.getString(5), rs.getString(6), 0, 0, 0, 0, 0, null, 0,
null, 0, null, 0, null),
                        new ArrayList<Branch>(),
                        new Branch(rs.getInt(7), rs.getString(8)),
                        new ArrayList<room>(),
                        new room(rs.getInt(9), rs.getString(10)),
                        rs.getInt(11), rs.getInt(12), rs.getInt(13), null,
                        new LecGroup(rs.getInt(14), rs.getString(15)), null);
                tims.add(t);
            }
```

Figure 4.35 (get all data for timetable from database)

```
rs = conn.select("select timeInTimetableId, branchId ,branch.name from  branchesintimeintimetable left join branch on branch.id=branchId where timeInTimetableId in (" + sb.substring(0, sb.length() - 1) + ") order by timeInTimetableId");
            int i = 0;
            while (rs.next()) {
                for (int j = i; j < tims.size(); j++) {
                    if (rs.getInt(1) == tims.get(j).getId()) {
                        i = j;
                        tims.get(j).getBranchs().add(new Branch(rs.getInt(2), rs.getString(3)));
                        break;
                    }
                }
            }

            rs = conn.select("select timeInTimetableId, RoomId ,rooms.name,rooms.branchId from  roomsintimeintimetable left join rooms on rooms.id=roomid where timeInTimetableId in (" + sb.substring(0, sb.length() - 1) + ") order by timeInTimetableId");
            i = 0;
            while (rs.next()) {
                for (int j = i; j < tims.size(); j++) {
                    if (rs.getInt(1) == tims.get(j).getId()) {
                        i = j;
                        tims.get(j).getRooms().add(new room(rs.getInt(2), rs.getString(3), 0, 0, null, rs.getInt(4), null));
                        break;
                    }
                }
            }

            rs = conn.select("select * from sectiongroupnamefortimeintimetable order by timeInTimetableId");
            i = 0;
            while (rs.next()) {
                for (int j = i; j < tims.size(); j++) {
                    if (rs.getInt(1) == tims.get(j).getId()) {
                        i = j;
                        tims.get(j).setSectionGroupName(rs.getString(2));
                        break;
                    }
                }
            }
            return tims;
        } catch (Exception ex) {
            System.out.println(ex);
        }
        return null;
    }
```

Figure 4.36 (get all data for timetable from database)

- **Frontend :** this is the final table that view the timetable.

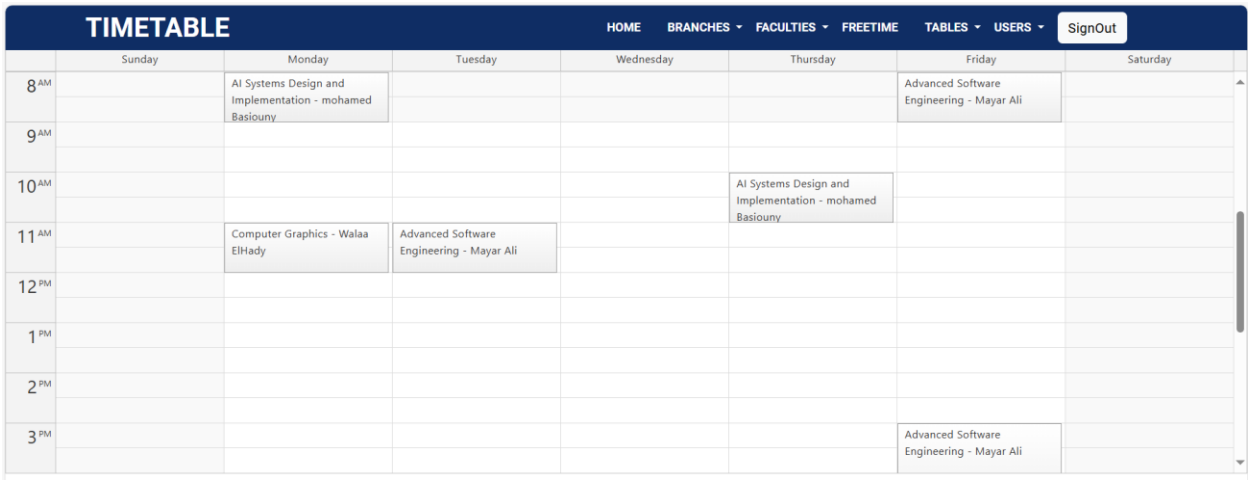

Figure 4.37 (final view of time table)

CHAPTER 5

Testing & Evaluation

5.1 Introduction

Testing is done to verify the results by testing each build, including both internal and intermediate builds as well as final versions of the system to be released to external parties" (Booch, Jacobson and Rumbaugh, 2003).

The steps involved are:

- Build the test plans required in each iteration.
- Perform integration and system tests.
- Create test cases to design and implement tests. Test cases define what to test.
- how to test and create executable test components Systematically handle and record test results of each test so that the significant defects and be fixed.

5.2 Test Plan

A software project test plan is a written document that outlines the goals, scope, strategy, and emphasis of a software testing activity. Preparing a test plan is a good approach to think through the steps required to confirm the acceptance of a software product. The finished test plan will assist consumers outside of the test group in comprehending the "why" and "how" of product validation.

• Requirements Analysis: collecting and assessing customer requirements.

• Software Test Plan: defining the scope and goals, developing acceptable testing techniques, developing a software testing strategy, assigning roles and duties, defining resource needs, and establishing start and completion criteria.

• Test Environment: establishing the test infrastructure, determining the testing environment and tools, and installing and configuring the product.

• Test Metrics: a description of the areas to be measured, as well as the creation and collection of metrics

• Test execution is the static and dynamic performance of testing that is given for the use of human and automatic test cases as needed by STP and STS.

• Defect Management (Bug Tracking), documenting testing results, defect description (Problem Reports, Change Requests); defect review and testing results analysis, faults repair, and defect resolution verification

• Status reports, weekly reports, milestone reports, and a closing report are all examples of reporting

5.7 Test Cases and Results:

A test case is a series of actions used to verify the correct behavior of an application's functionality or feature. A test case is a document that defines Input, Action, Event, and Expected Response in order to assess whether or not a feature of an application is operating properly

The following tables include observations and summaries of carefully selected test cases and outcomes.

Name	**User Login**		
Test case	**Input**	**Expected result**	**Status**
Testing Connection to the Server	Server name, user name, password and database name	Display connection confirmation page	Pass
Login – Registered User	Username and password	Home page	Pass

Table 5.1 (User Login test)

Name	**Generate Timetables**		
Test case	Input	Expected result	Status
Testing generate timetable according to the free define constrains	timetable according to the free define constrains Constraints, Subjects, resources	Display newly created timetable.	Pass

Table 5.2 (Generate Timetables test)

Name	**New Lecturers Insertion**		
Test case	Input	Expected result	Status
Testing Insert New staff member with all inputs	Staff member name, Faculty, Department, Name of subjects	Display newly inserted member in staff members page	Pass
Testing Insert New lecturer with Some blank details	With few input missing	error, the form save bottom will not be able to save	Pass
User tries to delete a staff member	Clicks Delete	Particular lecturer will be deleted from the database.	Failed

Table 5.3 (Generate Timetables test)

5.8 System Testing Process

To guarantee that all parts of the system have been evaluated, test cases are created using a wide range of test data. The inputs designed to test the system are referred to as test data. These test cases are created using the system requirements created during the project's design phase (i.e., knowledge of the program's structure and implementation).

5.9 Testing Data

The current manual System collected the system's test data, which was saved in Excel and Word formats.

5.6 User Evaluation

The user evaluation is done by using a questioner. The questioner compost of 10 questions to obtain user feedback. Regarding requirement fulfillness, user friendliness, report generation, supporting to decision making functionality, authentication, interface and response time.

CHAPTER 6

CONCLUSION & FUTURE WORK

6.1 Problems Encountered

The total project completion time needs more time than the original project schedule, because of the excessive development time. Most of the concepts and technologies were new and comprehensive initial knowledge was required to develop an appropriate website.

6.2 Critical Assessment

By the end of the project, almost all the primary objectives had been largely accomplished. The main objective was to provide a web-based facility for staff and the technical coordinator. As it was observed in the analysis phase, the main areas included lecture timetable, section timetable, and management reports. Separate User Management module is provided for the admin apart from system backup facility. They all were proved to be of the acceptable standard during the acceptance testing.

More effort is spent on system analysis and design throughout the project life cycle. Various methods of data collection were utilized for system analysis. They were meetings, past reports observation. As each requirement was found, it was recorded in the Requirements Catalogue and due care was taken to maintain the catalogue correctly as it serves as the main reference for the other stages of the project. IEEE standards were used to document the requirements catalogue. Regular requirement reviews were carried out to ensure the accuracy of the requirements gathered.

User interfaces proved to be friendly and easy to use and consist with a good color combination that mixed and kept the identity for EELU.

6.3 Conclusion

EELUTTMS was developed to manage Lecture & section schedules. It is a system created to give college staff members better support. To forecast, the timetable module provides a stable foundation. It offers a feature that allows users to create timetables Automatically. Additionally, this system includes a database that stores information related to the staff and Branches subjects.

Timetabling problem is NP-complete, it is being the hard combinational problem that would take more than just the application of only one principle. The timetable problem may only be solved when the constraints and the allocations are clearly defined and simplified thoroughly, and more than one principle is applied to it.

6.4 Future Work

Identified some further work to be done on implemented system. Since in the development process followed incremental model for software development which allows developing system incrementally module by module.

- **Integrate system to SIS**

  University students and staff will have access to timetables that improve the efficiency of organization functionality.

- **Knowledge sharing**

  Since most of the modules are developed with the requirement gathered from Stakeholders, others are not aware the exact functionality of other system, therefore planning to do a hands-on practical session of all chosen staff who operate with system.

- **Improve performance**

  can also develop predictive features that anticipate scheduling conflicts or suggest time management improvements.

- **Improve UX**

  Focus on simplifying data entry and editing of timetables.

References

[1] "Timetable Management System Software Report," SlideShare, https://www.slideshare.net/AdityaJain335/time-table-management-system-software-report (accessed Feb. 10, 2024).

[2] MUHAMMED TISHAN, "ALIGARH MUSLIM UNIVERSITY ALIGARH (INDIA)." 2010

[3] © 2007 - 2024 UniTime Tomáš Müller, "University timetabling," UniTime, https://www.unitime.org/ (accessed Feb. 10, 2024).

[4] Unitime, https://www.unitime.org/present/apereo17-courses.pdf (accessed Feb. 10, 2024).

[5] R.D.P.Indika Priyadarshana, "Web Based Timetable Management System for University of Vocational Technology (UNIVOTEC)," 2018 =

[6] T. S. Pang, "System architecture for B2B and SAAS," Medium, https://medium.com/@tatsean1977/system-architecture-for-b2b-and-saas-bc0da33e0d1e (accessed Feb. 10, 2023).

[7] "Bullet TimeTabler Education – System demonstration." Pedro Fernandes · Armando Barbosa · Luis Moreira