TECHNICAL DOCUMENTATION

LECTURE HALL BOOKING SYSTEM

EC 6060 SOFTWARE ENGINEERING PROJECT

TEAM NEXUS

CONTENTS

INTRODUCTION

The Lecture Hall Booking System is designed to streamline the process of reserving lecture halls at the Faculty of Engineering, University of Jaffna. The system aims to reduce scheduling conflicts, enhance resource utilization, and provide a seamless experience for lecturers, instructors, and administrators.

ARCHITECTURE & DESIGN PRINCIPLES

- Client-Server Architecture: The system will use a client-server model where the front end (client) interacts with the back end (server) through RESTful APIs.
- Modularity: The system will be divided into modules such as space service, reservation service, waiting service, email service to enhance maintainability.
- Security: Implement secure authentication mechanisms using Spring OAuth 2.0 and HTTPS for secure communication.
- Scalability: Design the system to handle increasing numbers of users and bookings, with support for horizontal scaling.
- Reliability: Ensure high availability and robust error handling to maintain data integrity and system uptime.

USER STORY DESCRIPTION

As a Lecturer:

- I want to book a lecture hall, so that I can conduct my class.
- I want to view my booked and available time slots, so that I can manage my schedule.
- I want to cancel my reservations, so that I can free up time slots if my plans change.

As an MA:

- I want to add new lecture halls to the system, so that they are available for booking.
- I want to update the details of existing lecture halls, so that the information is accurate.
- I want to view and manage all bookings, so that I can resolve conflicts and optimize usage.

As a student:

- I want to view the schedule of lecture halls, so that I can attend classes accordingly.

SYSTEM DESIGN LANGAUGES

HTML (Hypertext Markup Language):

Standard language for creating the structure of web pages using tags and elements like headings, paragraphs, and images.

CSS (Cascading Style Sheets):

Language used to define the visual presentation of HTML documents, controlling layout, colors, and fonts.

Java:

Object-oriented programming language known for its portability and reliability, commonly used for building enterprise-scale applications.

JavaScript:

Scripting language used for client-side web development, adding interactivity and dynamic behavior to web pages.

SYSTEM TOOLS

VS Code:

Lightweight, open-source code editor with support for numerous programming languages and extensions, facilitating efficient coding workflows.

MySQL Workbench:

Visual database design and administration tool, providing features for SQL development, data modeling, and maintenance.

Jira:

Project management and issue tracking software used for agile software development, facilitating team collaboration and task management.

GitHub:

Web-based hosting service for version control using Git, enabling collaboration among developers through features like pull requests and code review.

Figma:

Collaborative interface design tool for creating and prototyping user interfaces, supporting real-time collaboration and design system management.

Draw.io:

Online diagramming tool for creating flowcharts and diagrams, offering a wide range of shapes and templates for visualizing ideas and processes.

IntelliJ IDEA:

Integrated development environment (IDE) for Java, offering advanced features for coding, debugging, testing, and profiling Java applications.

Swagger UI:

Interactive API Documentation: Swagger UI provides an interactive and user-friendly interface for exploring and testing API endpoints, simplifying API documentation and consumption for developers.

Postman:

Comprehensive API Testing and Development: Postman offers a comprehensive platform for testing, developing, and documenting APIs, streamlining the entire API lifecycle from creation to deployment.

SYSTEM FRAMEWORK

Spring Boot:

Java-based framework simplifying enterprise application development by providing out-of-the-box solutions for common tasks like dependency injection and MVC architecture.

React.js:

JavaScript library for building user interfaces, utilizing a component-based architecture for building reusable UI elements in single-page applications.

SCSS (Sassy CSS):

Extension of CSS with features like variables and nesting, aiding in writing more maintainable and modular CSS code.

SYSTEM DATABASE

MySQL:

Relational database management system offering features like ACID compliance and robust SQL support, widely used for web applications and content management systems.

DATA MODEL

ENTITIES AND ATTRIBUTES

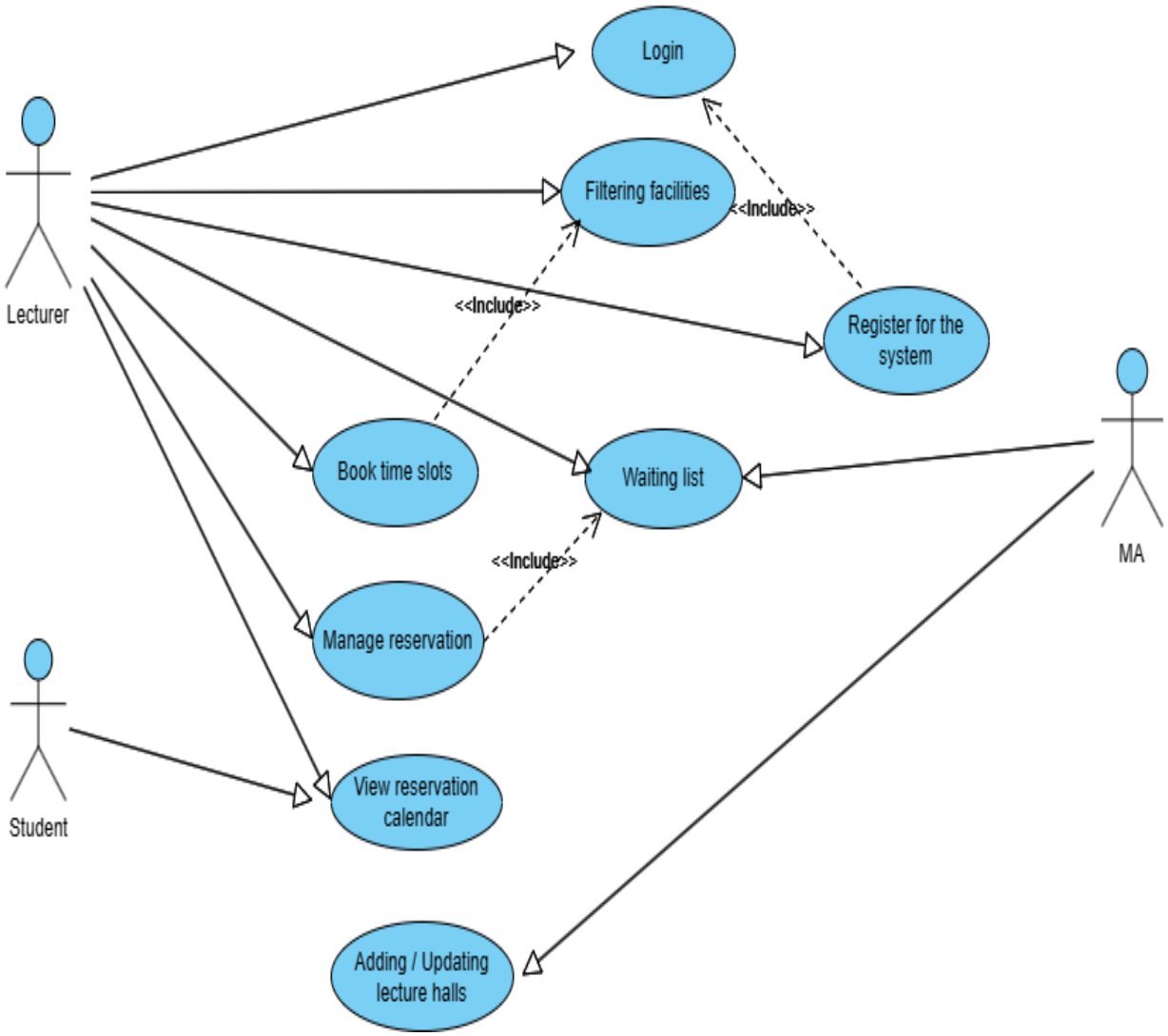
1. **User**
   - `id` (INT, Primary Key): Unique identifier for each user.
   - `firstName` (VARCHAR): First name of the user.
   - `lastName` (VARCHAR): Last name of the user.
   - `email` (VARCHAR, Unique): Email address of the user.
   - `password` (VARCHAR): Encrypted password of the user.
   - `role` (ENUM: `LECTURER`, `MA`, `STUDENT`): Role of the user in the system.
2. **Space (Lecture Hall)**
   - `id` (INT, Primary Key): Unique identifier for each lecture hall.
   - `name` (VARCHAR): Name of the lecture hall.
   - `location` (VARCHAR): Location of the lecture hall.
   - `capacity` (INT): Capacity of the lecture hall.
   - `description` (TEXT): Description of the lecture hall.
   - `facilities` (VARCHAR): List of facilities available in the lecture hall.
3. **Reservation**
   - `id` (INT, Primary Key): Unique identifier for each reservation.
   - `title` (VARCHAR): Title of the reservation.
   - `startTime` (TIME): Start time of the reservation.
   - `endTime` (TIME): End time of the reservation.
   - `reservationDate` (DATE): Date of the reservation.
   - `userId` (INT, Foreign Key): ID of the user who made the reservation.
   - `spaceId` (INT, Foreign Key): ID of the reserved space.
   - `batch` (VARCHAR): Batch of students (if applicable).
   - `waitingId` (INT, Foreign Key, Nullable): ID of the associated waiting list entry (if applicable).
4. **Waiting**
   - `id` (INT, Primary Key): Unique identifier for each waiting list entry.
   - `title` (VARCHAR): Title of the waiting list entry.
   - `startTime` (TIME): Start time for the desired reservation.
   - `endTime` (TIME): End time for the desired reservation.
   - `waitingForDate` (DATE): Date for the desired reservation.
   - `userId` (INT, Foreign Key): ID of the user who is waiting.
   - `spaceId` (INT, Foreign Key): ID of the desired space.
   - `batch` (VARCHAR): Batch of students (if applicable).
   - `isAvailable` (BOOLEAN): Indicates if the desired time slot has become available.

RELATIONSHIPS

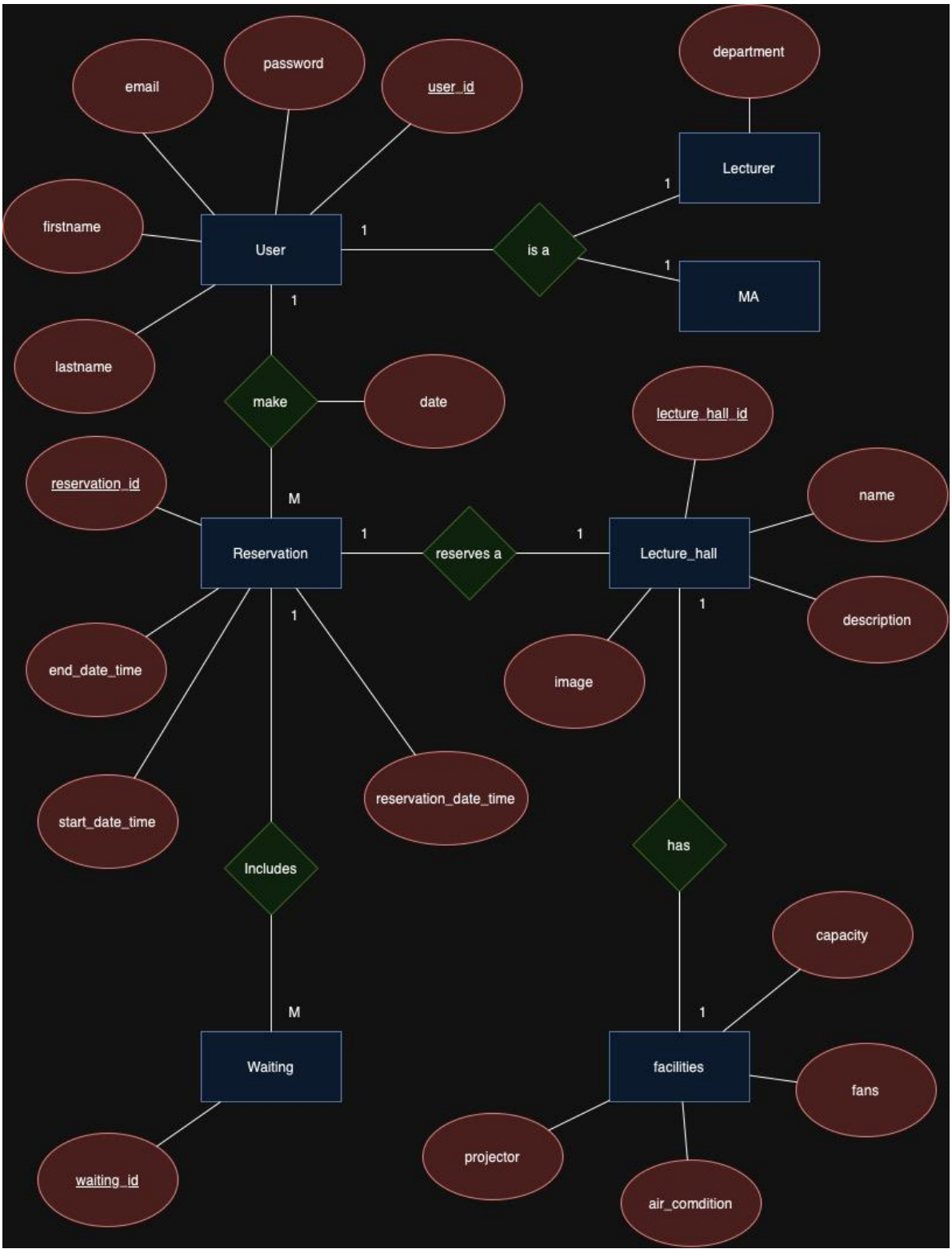
1. **User to Reservation**
   - One-to-Many: A user can make multiple reservations.
2. **Space to Reservation**
   - One-to-Many: A space can have multiple reservations.
3. **User to Waiting**
   - One-to-Many: A user can have multiple entries in the waiting list.
4. **Space to Waiting**
   - One-to-Many: A space can have multiple entries in the waiting list.
5. **Waiting to Reservation**
   - One-to-One: Each waiting list entry can be linked to one reservation if the slot becomes available.

DIAGRAMMATIC REPRESENTATION

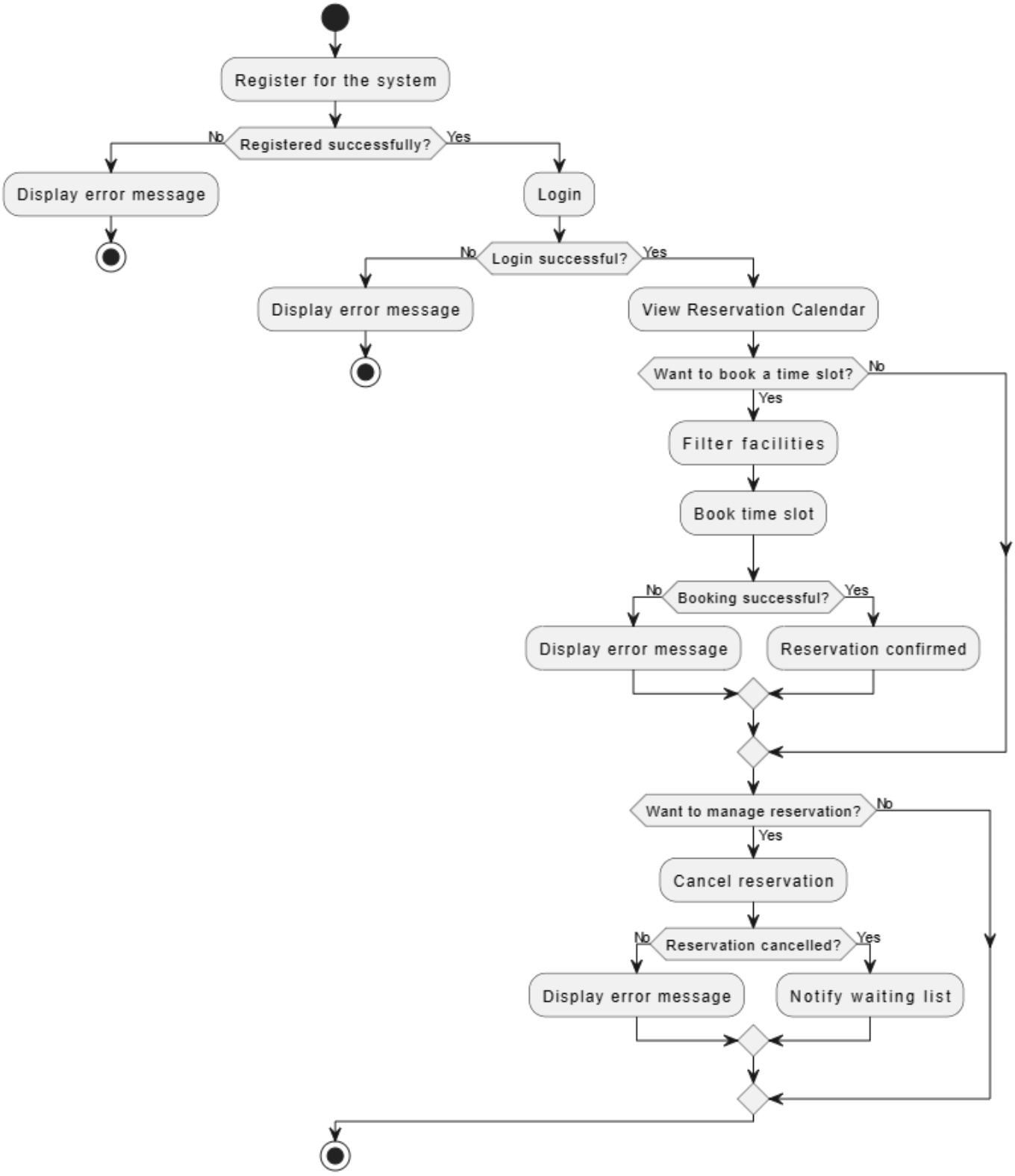
USE CASE DIAGRAM



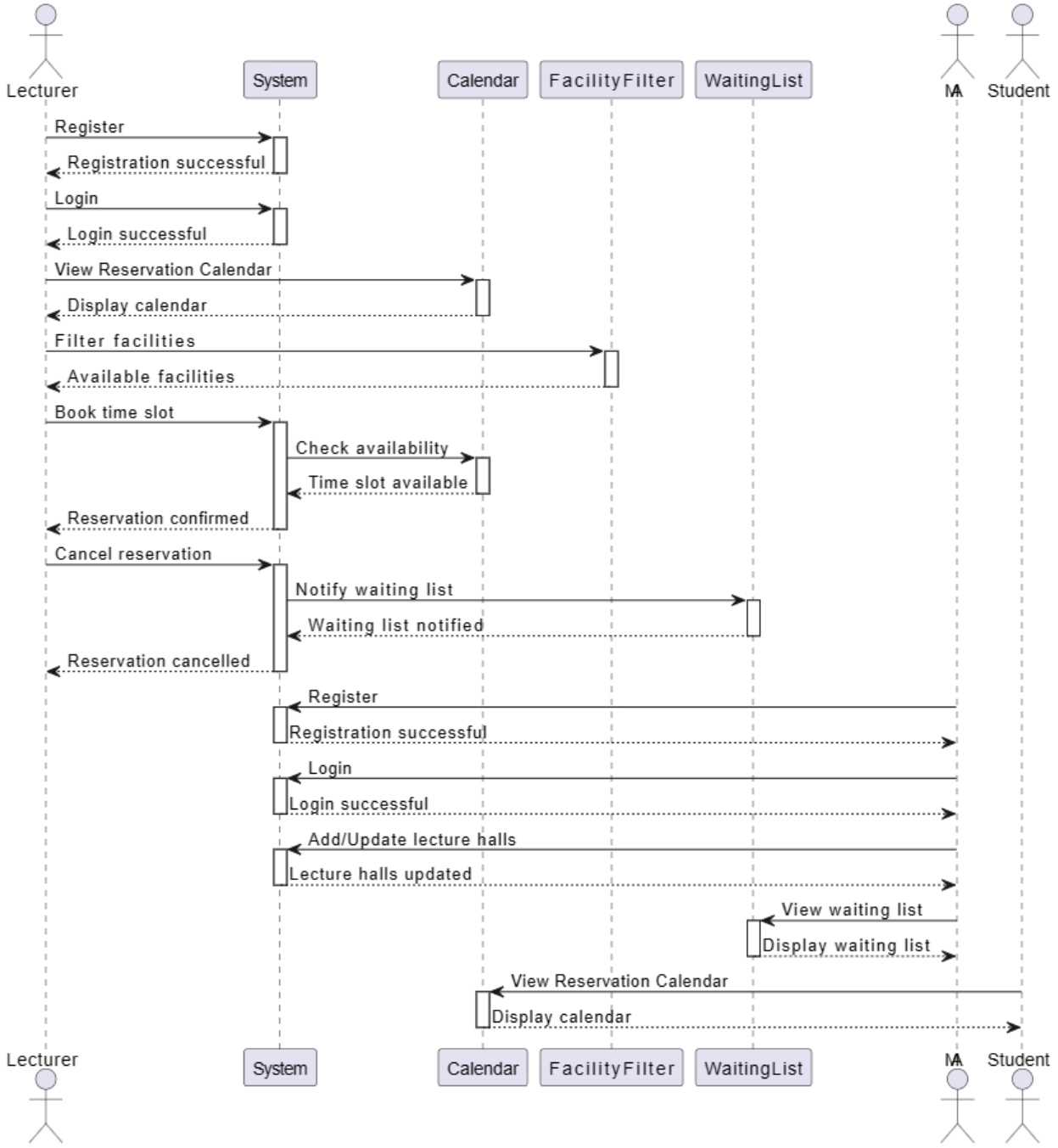
ER DIAGRAM


email
password
user_id
department
Lecturer
firstname
User
is a
MA
lastname
make
date
lecture_hall_id
reservation_id
name
Reservation
reserves a
Lecture_hall
description
end_date_time
image
start_date_time
reservation_date_time
Includes
has
capacity
Waiting
facilities
fans
projector
waiting_id
air_comdition
1
M

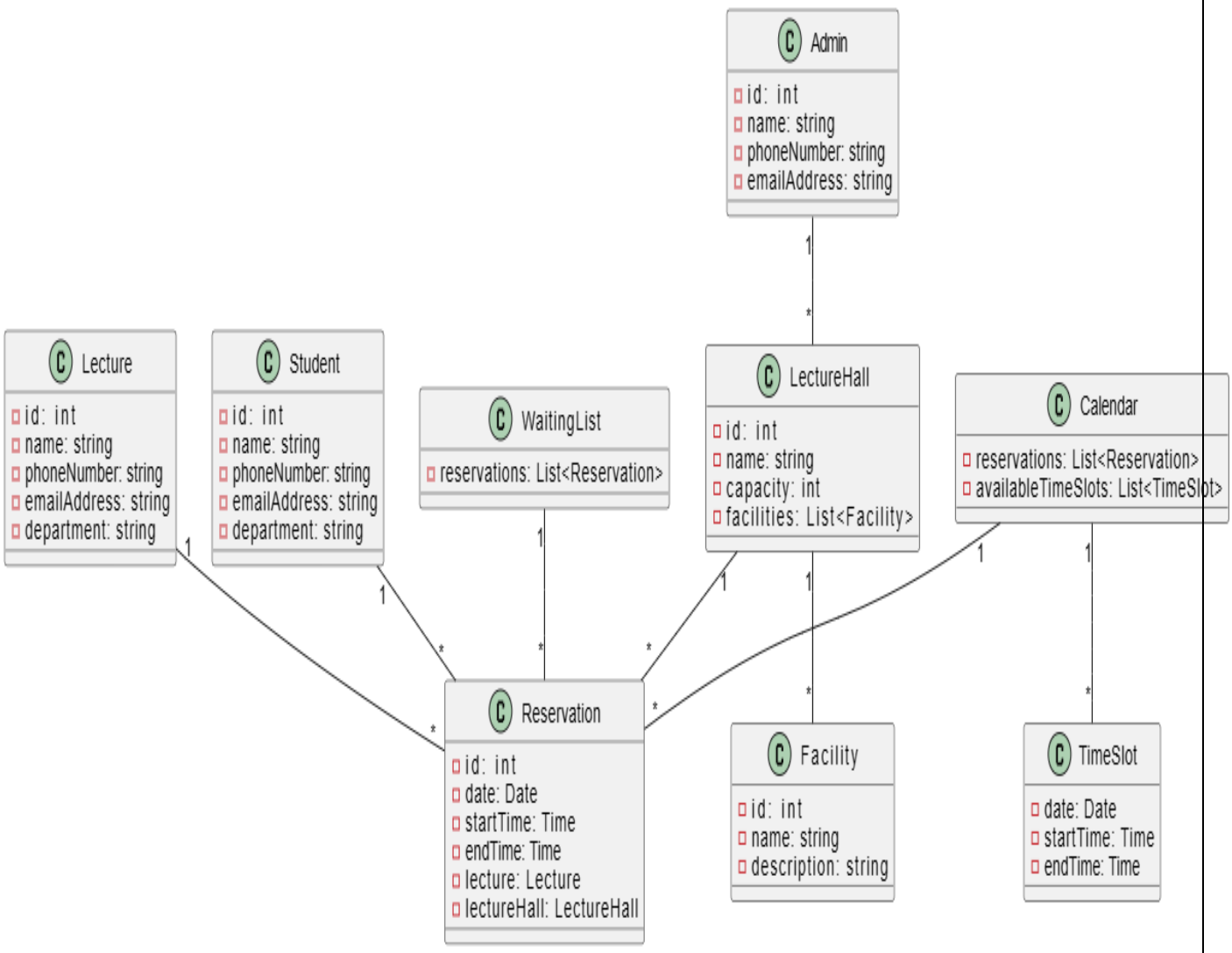
ACTIVITY DIAGRAM



SEQUENCE DIAGRAM



CONCEPTUAL DIAGRAM



MILESTONES

Planning and Requirements Gathering:

Define project scope, gather requirements, and identify stakeholders.

Design and Architecture Development:

Develop system architecture, design database schema, and create UI mockups.

Implementation of Core Features:

Develop and integrate user registration, login, and booking functionalities.

Testing and Quality Assurance:

Conduct unit, integration, and system testing to ensure functionality and performance.

Deployment and Maintenance:

Deploy the system to the production environment and provide ongoing maintenance and support.

API DOCUMENTATION

AUTHENTICATION SERVICE

REGISTER USER

- Endpoint : /api/v1/auth/register
- Method : POST
- Request :

```
{
 "firstName": "string",
 "lastName": "string",
 "email": "string",
 "password": "string",
 "userRole": "string"
}
```

- Response :

```
{
 "accessToken": "string",
 "refreshToken": "string"
}
```

- Description : Register a user for the system

LOGGING IN A USER FOR THE SYSTEM

- Endpoint : /api/v1/auth/authenticate
- Method : Method :
- Request :

```
{
 "email": "string",
 "password": "string"
}
```

- Response :

```
{
```

"accessToken": "string",

"refreshToken": "string"

}

- Descrption : loggin a user for the system

SPACE SERVICE

CREATE A SPACE

- Endpoint: /api/v1/spaces /createspaces
- Method : POST
- Request:

{

"name": "string",

"location": "string",

"capacity": 0,

"description": "string",

"facilities": [

"string"

]

}

- Response:

{

"id": 0,

"name": "string",

"location": "string",

"capacity": 0,

"description": "string",

"facilitiesList": [

"string"

]

}

- Description: create a new space(hall) in data base

GET ALL SPACES

- Endpoint: : /api/v1/spaces /getallspaces
- Method : GET
- Request: No parameters
- Response:


```
[
 {
  "id": 0,
  "name": "string",
  "location": "string",
  "capacity": 0,
  "description": "string",
  "facilitiesList": [
   "string"
  ]
 }
]
```


- Description: getting all the spaces to frontend

RESERVATION SERVICE

CREATE RESERVATION

- Endpoint: /api/v1/reservations/createreservations
- Method : POST
- Request:


```
{
 "title": "string",
 "startTime": 0,
 "endTime": 0,
 "spaceID": 0,
 "reservationDate": "string",
 "date": "string",
 "reservedByID": 0,
 "responsibleRole": "string",
 "batch": "string",
 "waitingId": 0
}
```


- Response:


```
{
 "id": 0,
 "title": "string",
 "startTime": 0,
 "endTime": 0,
 "spaceID": 0,
 "reservationDate": "string",
 "date": "string",
 "reservedByID": 0,
 "responsibleRole": "string",
 "batch": "string",
 "fullName": "string",
```

"waitingId": 0

}

- Description: create a reservation on database and sending it to the frontend

GET ALL RESERVATIONS

- Endpoint: /api/v1/reservations/getAllreservations
- Method : POST
- Request: No parameters
- Response:

[

{

"id": 0,

"title": "string",

"startTime": 0,

"endTime": 0,

"spaceID": 0,

"reservationDate": "string",

"date": "string",

"reservedByID": 0,

"responsibleRole": "string",

"batch": "string",

"fullName": "string",

"waitingId": 0

}

]

- Description: fetch all reservations from the database and sending them to frontend

GET USER RESERVATIONS

- Endpoint: /api/v1/reservations /user
- Method : GET
- Request: email - as a request param
- Response:


```
[
 {
  "id": 0,
  "title": "string",
  "startTime": 0,
  "endTime": 0,
  "spaceID": 0,
  "reservationDate": "string",
  "date": "string",
  "reservedByID": 0,
  "responsibleRole": "string",
  "batch": "string",
  "fullName": "string",
  "spaceName": "string",
  "waitingId": 0
 }
]
```


- Description: fetching user reservations from database

DELETE RESERVATIONS

- Endpoint: /api/v1/reservations /canceluserreservations
- Method : DELETE
- Request: reservationId – as arequest param
- Response: 200 stautus code
- Description: delete reservation from the database

WAITING SERVICE

CREATING A WAITING

- Endpoint: /api/v1/waiting /createrwaitings
- Method : POST
- Request:

```
{
 "title": "string",
 "startTime": 0,
 "endTime": 0,
 "spaceID": 0,
 "waitingForDate": "string",
 "date": "string",
 "waitingByID": 0,
 "responsibleRole": "string",
 "batch": "string",
 "waitingId": 0
}
```

- Response:

```
{
 "title": "string",
 "startTime": 0,
 "endTime": 0,
```

"spaceID": 0,

"reservationDate": "string",

"date": "string",

"waitingByID": 0,

"responsibleRole": "string",

"batch": "string",

"fullName": "string",

"waitingId": 0

}

- Description: create waiting and save it in database , getting it to database

GET USER WAITINGS

- Endpoint: /api/v1/waiting /user
- Method : GET
- Request: : email as a request param
- Response:

[

{

"id": 0,

"title": "string",

"startTime": 0,

"endTime": 0,

"spaceID": 0,

"reservationDate": "string",

"date": "string",

"waitingByID": 0,

"responsibleRole": "string",

"batch": "string",

"fullName": "string",

"spaceName": "string",

```
    "waitingId": 0,
    "available": true
  }
]
```


- Description: getting all user reservation from the database

DELETE USER WAITINGS

- Endpoint: /api/v1/waiting /deleteuserwaitings
- Method : DELETE
- Request: waitingId – as a request param
- Response: 200 status code
- Description: deleting waiting from the database

SECURITY MEASURES

- **Data Protection:** All sensitive data, such as passwords, are encrypted using SHA-256 before being stored in the database.
- **Access Control:** Implemented using Spring Security with role-based access control (RBAC) to ensure different permissions for lecturers, administrators, and students.
- **Audit Logging:** All user activities, including logins and bookings, are logged with timestamps for audit purposes.

PERFORMANCE CONSIDERATIONS

- **Scalability:** The system is designed to support horizontal scaling with load balancers distributing incoming requests across multiple servers.
- **Caching:** Redis is used to cache frequently accessed data, reducing load on the MySQL database and improving response times.
- **Performance Testing:** Load and stress testing have been conducted using JMeter, ensuring the system can handle up to 1000 concurrent users with an average response time of under 200ms.

DEPLOYMENT AND DEVOPS

- **Infrastructure as Code:** AWS CloudFormation is used for defining and provisioning the infrastructure, ensuring consistency across different environments.
- **Environment Configuration:** Separate configurations for development, staging, and production environments, managed through environment variables.
- **Containerization:** Docker is used for containerizing the application, allowing for consistent deployment across different environments.
- **Frontend Hosting:** The frontend is hosted on an S3 bucket, leveraging AWS CloudFront for content delivery and improved performance.
- **Backend Hosting:** The backend is deployed on an EC2 instance, ensuring scalability and reliability.
- **Database Management:** The MySQL database is hosted on Amazon RDS, providing a managed database service with automated backups and scaling.
- **CI/CD Pipeline:** Expected to be implemented using GitHub Actions, automating the process of building, testing, and deploying the application.

QUALITY ASSURANCE DOCUMENTATION

- **Automated Testing:** JUnit is used for unit tests, and Selenium for end-to-end tests. Test coverage is at 85%.
- **Manual Testing:** Detailed test cases have been documented, covering all functional aspects of the system. Each test case includes steps, expected results, and actual results.

USER GUIDE

Registering as a Lecturer

1. Go to the registration page.
2. Fill in the required details.
3. Click "Submit" to create your account.

Logging in to the System

1. Navigate to the login page.
2. Enter your username and password.
3. Click "Login" to access your dashboard.

Viewing the Reservation Calendar

1. Click on the "Calendar" tab.
2. Browse through the dates to see available and booked time slots.

Booking a Time Slot

1. Select an available time slot on the calendar.
2. Click "Book Now".
3. Confirm your booking details and click "Submit".

Canceling a Reservation

1. Go to "My Bookings".
2. Select the reservation you want to cancel.
3. Click "Cancel" and confirm your action.

Checking the Waiting List

1. Go to the "Waiting List" tab.
2. View the list of users waiting for a specific time slot.

Filtering Lecture Hall Facilities

1. Navigate to the "Lecture Halls" page.
2. Use the filters to search for lecture halls by capacity, equipment, etc.

(For MA) Adding and Updating Lecture Halls

1. Go to the "Admin" panel.
2. Click "Add Lecture Hall" or select an existing hall to update.
3. Fill in the required details and click "Save".