Software Design Document (SDD)

Campus Room Schedule and Management System

Course: CSAI 203 – Introduction to Software Engineering

Instructor: Mohamed Rakha

TAs: Kholoud Osman, Maram Alazab, Mohamed Nabil, Shorouq Odaiba, Nada Mostafa

Team Members:

Ahmed Ayman Mostafa – 202401612

Mohamed Ahmed Fouad – 202401032

Yousef Hossameldin Mohamed – 202402592

Prepared for CSAI 203 Project – Zewail City University of Science and Technology

Table of Contents

1. Introduction

1.1 Purpose of the Document

The purpose of this Design Document is to provide a comprehensive architectural and detailed design for the **Campus Room Schedule and Management System**. It translates the requirements defined in the SRS into a complete technical blueprint for implementation. The document describes the system architecture, data design, user interface design, and explains how the MVC design pattern is applied to ensure modularity and maintainability.

1.2 Scope of the Design Phase

This document covers both the architectural and detailed component design of the entire web application.
The scope includes:

- **Architectural Design:** Defining the high-level structure using the Monolithic architecture style and MVC pattern.
- **Data Design:** Defining the database schema, entities, and relationships for users, rooms, and schedules.
- **Component Design:** Detailed mapping of the Flask application (Models, Views, Controllers).
- **User Interface Design:** Wireframes representing the layout and flow of HTML-based web pages.

1.3 Intended Audience

This document is intended for:

- **Development Team:** To understand the system structure, database schema, and coding standards required for implementation.
- **Course Evaluators (Professor & TAs):** To verify that the design meets course requirements, especially the correct application of the MVC pattern and architectural constraints.

1.4 Overview of the Contents

The document is organized as follows:

- **Section 2 – System Overview:** Summarizes the system functionality and highlights key design goals and constraints.
- **Section 3 – Architectural Design:** Illustrates the high-level architecture and technology stack.
- **Section 4 – Detailed Design:** Presents MVC implementation details, class & sequence diagrams, UI wireframes, and the database schema.
- **Section 5 – Conclusion:** Summarizes the design decisions.

2. System Overview

2.1 Brief Description of the System

The **Campus Room Schedule and Management System** is a web-based platform designed to streamline the management of university hall schedules. Staff currently rely on manual Excel sheets, which leads to inefficiencies.
This system automates the workflow through:

- Uploading Excel schedules which are parsed and saved to the database.
- A live Dashboard showing real-time room status (Open/Closed) with countdown timers.
- Role-based access:
    - **Admins:** Manage rooms & users.
    - **Staff:** Perform manual room overrides.
- An integrated issue-tracking system enabling TAs and Professors to report maintenance problems.

2.2 Key Design Goals and Constraints

Design Goals

- **Simplicity:** A straightforward architecture for easy development within the course timeline.
- **Maintainability:** Strict use of MVC ensures separation of concerns and simplified updates.

- **Usability:** An intuitive, lightweight interface enabling quick actions (e.g., opening a room).

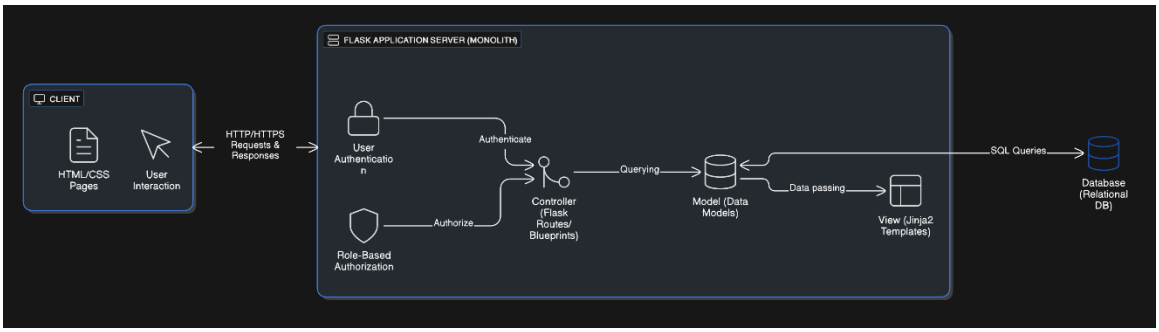
Design Constraints

The system adheres to CSAI 203 course technical requirements:

- **Architectural Pattern:** Full implementation of the MVC design pattern.
- **Backend Technology:** Python Flask for all application logic.
- **Frontend Technology:** Plain HTML and CSS only—no advanced UI frameworks.
- **Deployment:** Monolithic architecture deployed on a single server.
- **Data Persistence:** Relational DB using SQLAlchemy/SQLite for schedules and user logs.

1. **Architectural Design**

3.1 System Architecture Diagram



3.2 Discussion of Architectural Style and Components

The Campus Room Schedule and Management System utilizes a **Monolithic Architecture**. In this architectural style, the server-side application is designed as a single, unified unit where the user interface, business logic, and data access layers reside within the same codebase and are deployed together.

This architecture was chosen to satisfy the project constraints of being a "standalone web application" and to minimize deployment complexity for the demonstration server. The system is composed of the following core components:

Presentation Layer (View): Handles the rendering of the user interface using HTML and CSS. It receives data from the controller and presents it to the Hall Staff, TAs, and Professors.

Application Logic Layer (Controller): Implemented using the Python Flask framework. This component processes incoming HTTP requests, enforces role-based access control (RBAC), and orchestrates the flow of data between the Model and the View.

Data Access Layer (Model): Manages the business rules and interaction with the persistent storage. It creates an abstraction over the database, allowing

the controller to manipulate entities like Users, Rooms, and Schedules without writing raw SQL.

3.3 Technology Stack and Tools

The following technologies and tools have been selected to implement the system, adhering to the design constraints:

Programming Language: Python 3.x (Backend logic).

Web Framework: Flask (Implements the MVC structure and routing).

Frontend Technologies: HTML5, CSS3 (No external JavaScript frameworks, adhering to constraints).

Database: MySQL (for production persistence).

Containerization: Docker (Ensures consistent environments across development and deployment).

Version Control: Git (Source code management).

CI/CD: GitHub Actions (Automated testing and deployment pipelines).

2. **Detailed Design**

4.1 Model–View–Controller (MVC) Design Pattern

4.1.1 Description of MVC Pattern

The system follows the **Model-View-Controller (MVC)** design pattern. This pattern separates the application into three interconnected components, ensuring a clean separation of concerns:

1. **Model:** Manages the data, logic, and rules of the application.
2. **View:** Outputs the representation of the information (the UI).
3. **Controller:** Accepts input and converts it to commands for the model or view.

This pattern was chosen to meet the project's design constraints and to ensure the codebase remains modular and testable.

4.1.2 Mapping of Project Components to MVC

The project components map to the MVC layers as follows:

- **Model:** Represented by Python classes (SQLAlchemy models) that define the structure of the database tables. Key classes include User, Room, Schedule, Issue, and AuditLog.
- **View:** Represented by **Jinja2 templates** (HTML files) and static CSS assets. These templates dynamically render data provided by the backend.
- **Controller:** Represented by **Flask View Functions (Routes)**. These functions (e.g., login(), dashboard(), upload_schedule()) handle the HTTP requests, validate input, and determine which template to render.

4.1.3 Responsibilities of Model, View, and Controller

Model Responsibilities:

- Defines the data schema and relationships (e.g., a Room *has many* Schedules).
- Handles database interactions (Create, Read, Update, Delete).
- Validates business logic, such as checking for schedule conflicts during import.

View Responsibilities:

- Presents the User Interface to the user (e.g., Dashboard, Login Form).
- Displays dynamic data such as real-time countdown timers and room statuses.
- Provides forms for user input (e.g., Issue Reporting form, Excel upload button).

Controller Responsibilities:

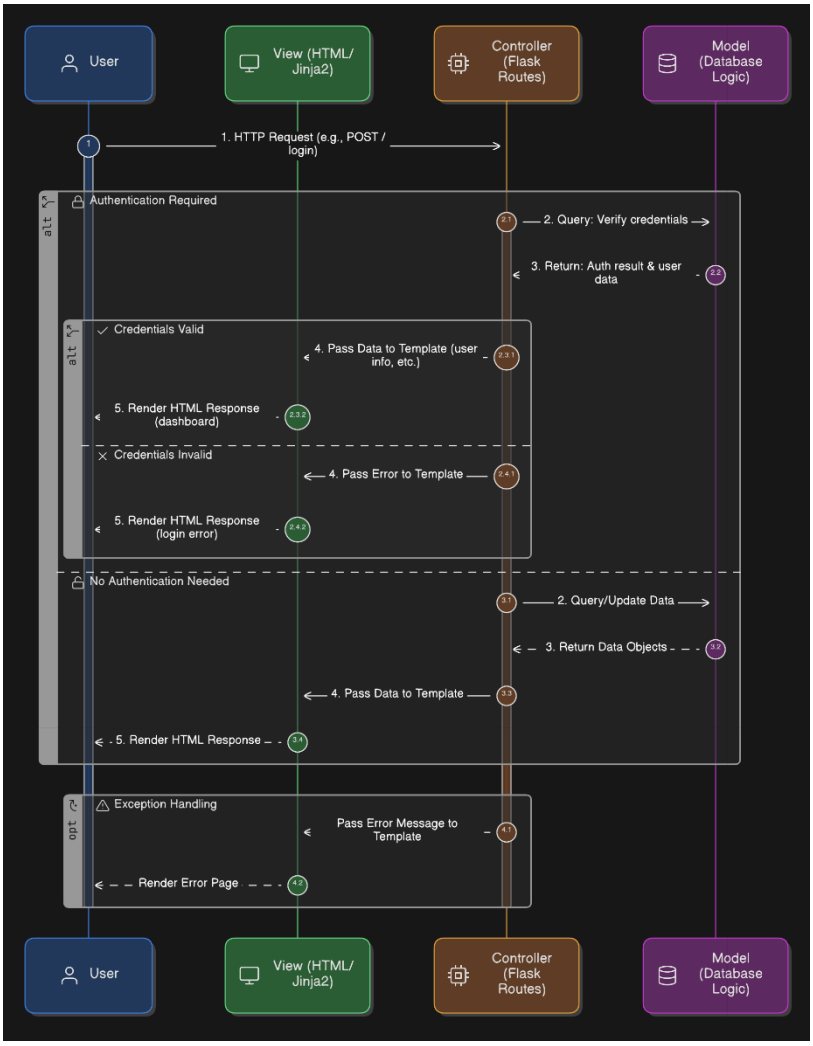
- Receives HTTP requests (GET/POST) from the browser.
- Authenticates users and enforces role-based authorization.
- Calls Model methods to retrieve or update data.
- Selects the appropriate View to return to the user (e.g., redirecting to the Dashboard after a successful login).

4.1.4 Interaction Between Components

The data flow within the system operates as follows:

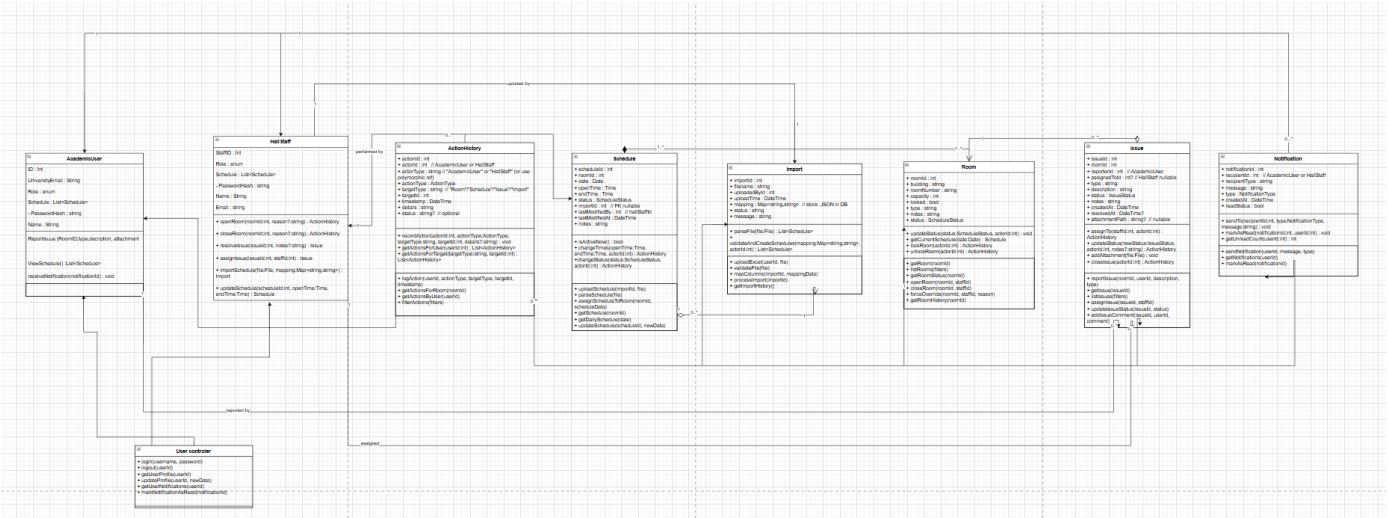
1. **Request:** The user interacts with the **View** (e.g., clicks "Login"), sending a request to the **Controller**.
2. **Processing:** The **Controller** processes the request. It may ask the **Model** to verify credentials or retrieve data.
3. **Data Retrieval:** The **Model** queries the database and returns the requested objects to the **Controller**.
4. **Response:** The **Controller** injects this data into a specific **View** template and returns the rendered HTML to the user's browser.

MVC Interaction diagram:



4.2 UML Diagrams

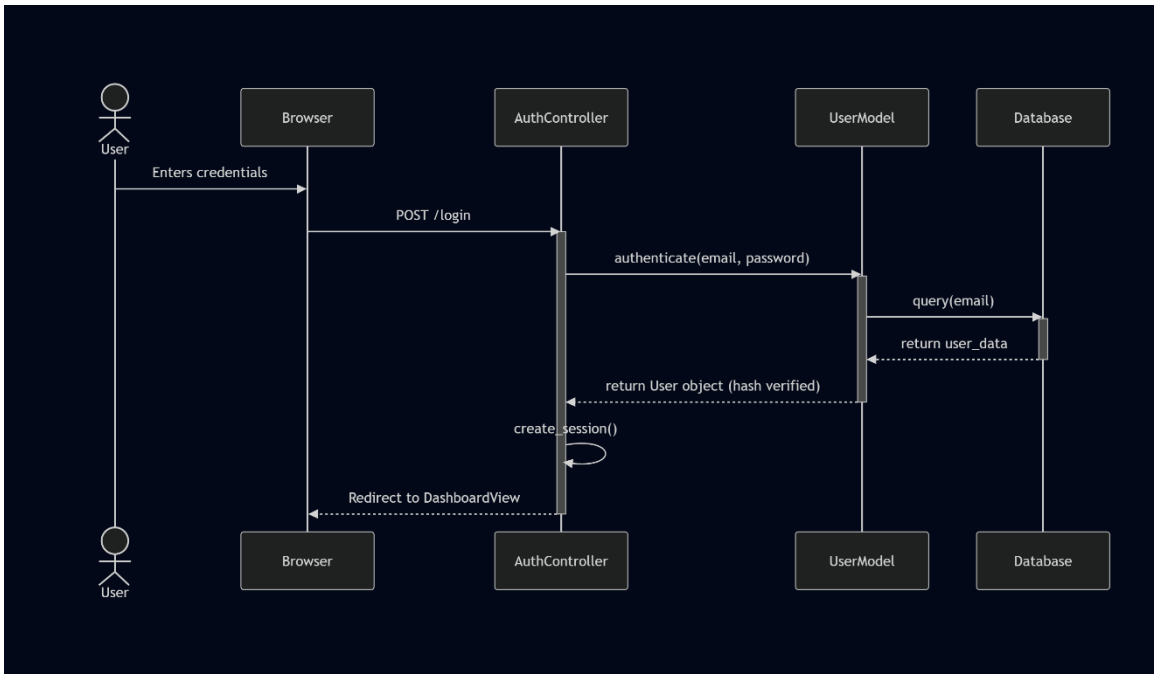
4.2.2 Detailed Class Diagram



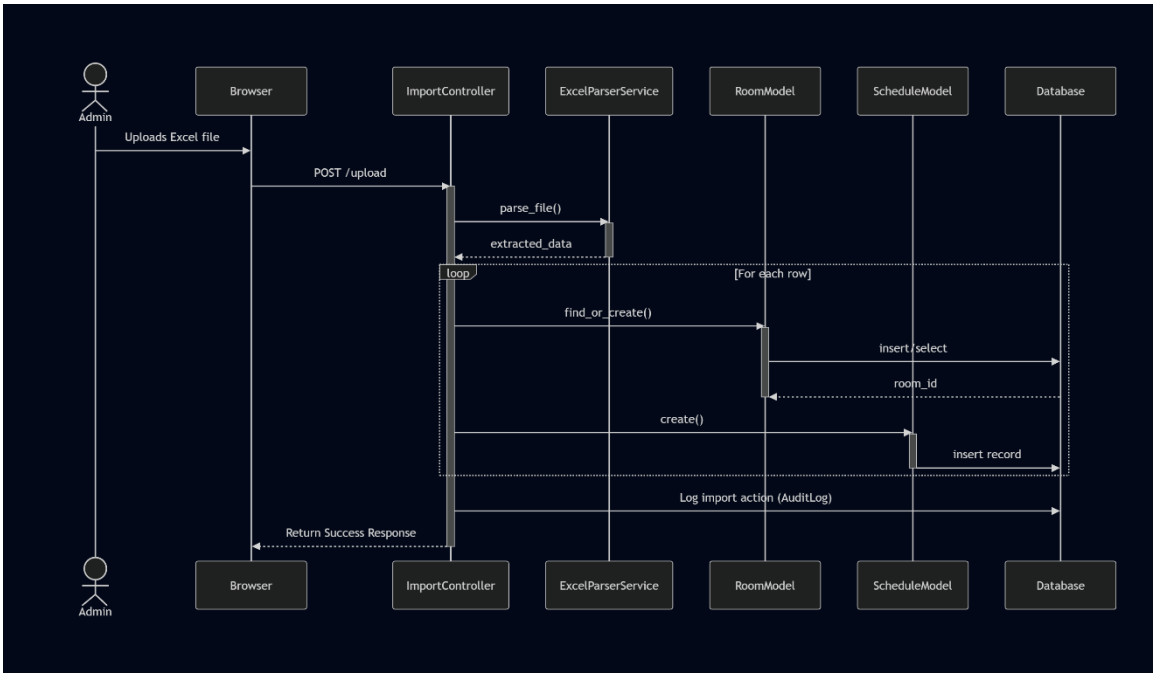
* the controlers is the third row of each class, to make it simple as possible

4.2.3 Sequence Diagrams

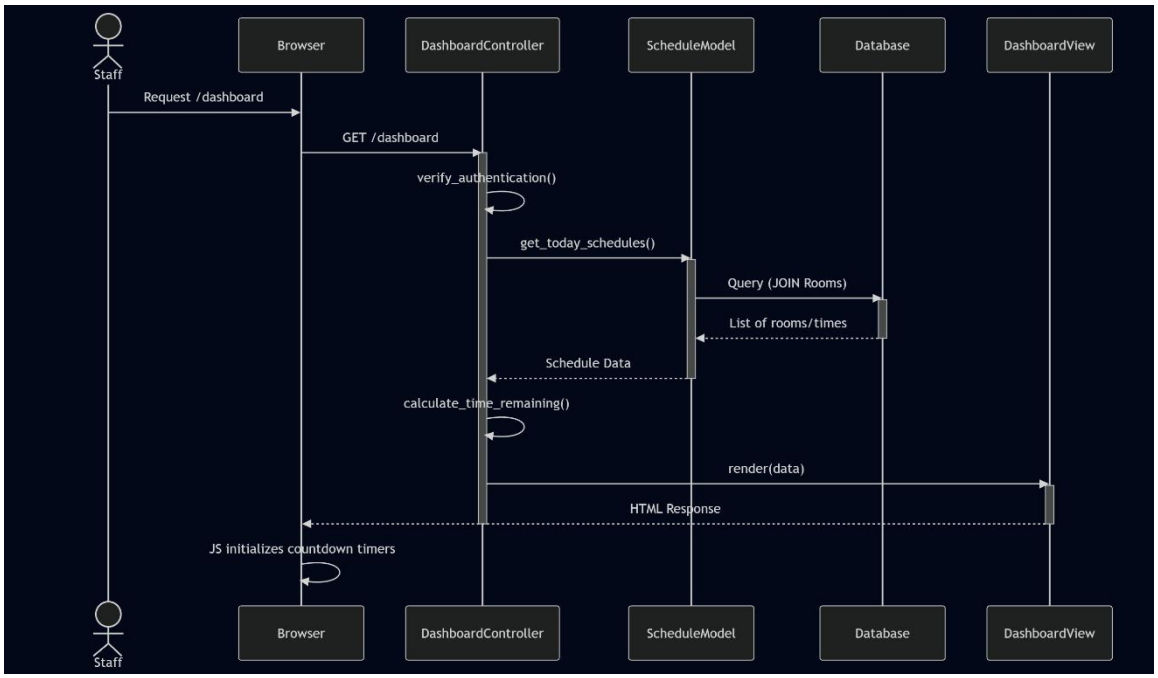
Sequence diagram1:



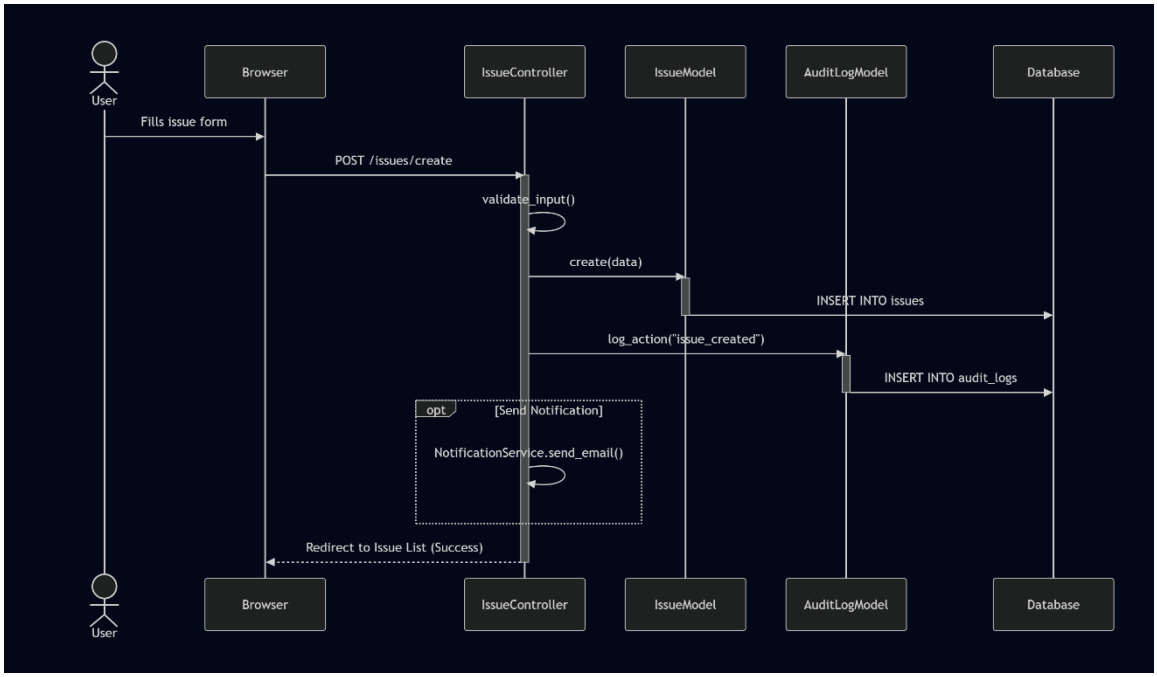
Sequence diagram2:



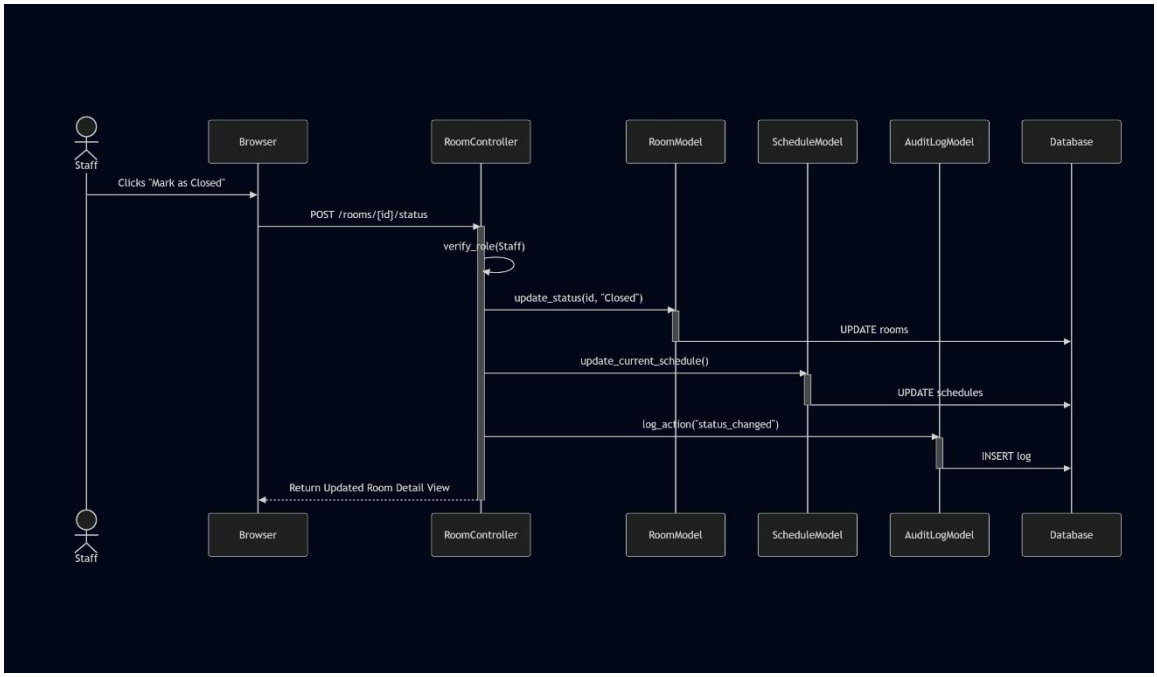
Sequence diagram3:



Sequence diagram4:



Sequence diagram5:



4.3 UI/UX Design

4.3.1 Wireframes / Mockups

1. landing page

2.login page

3. main page of hallstaff

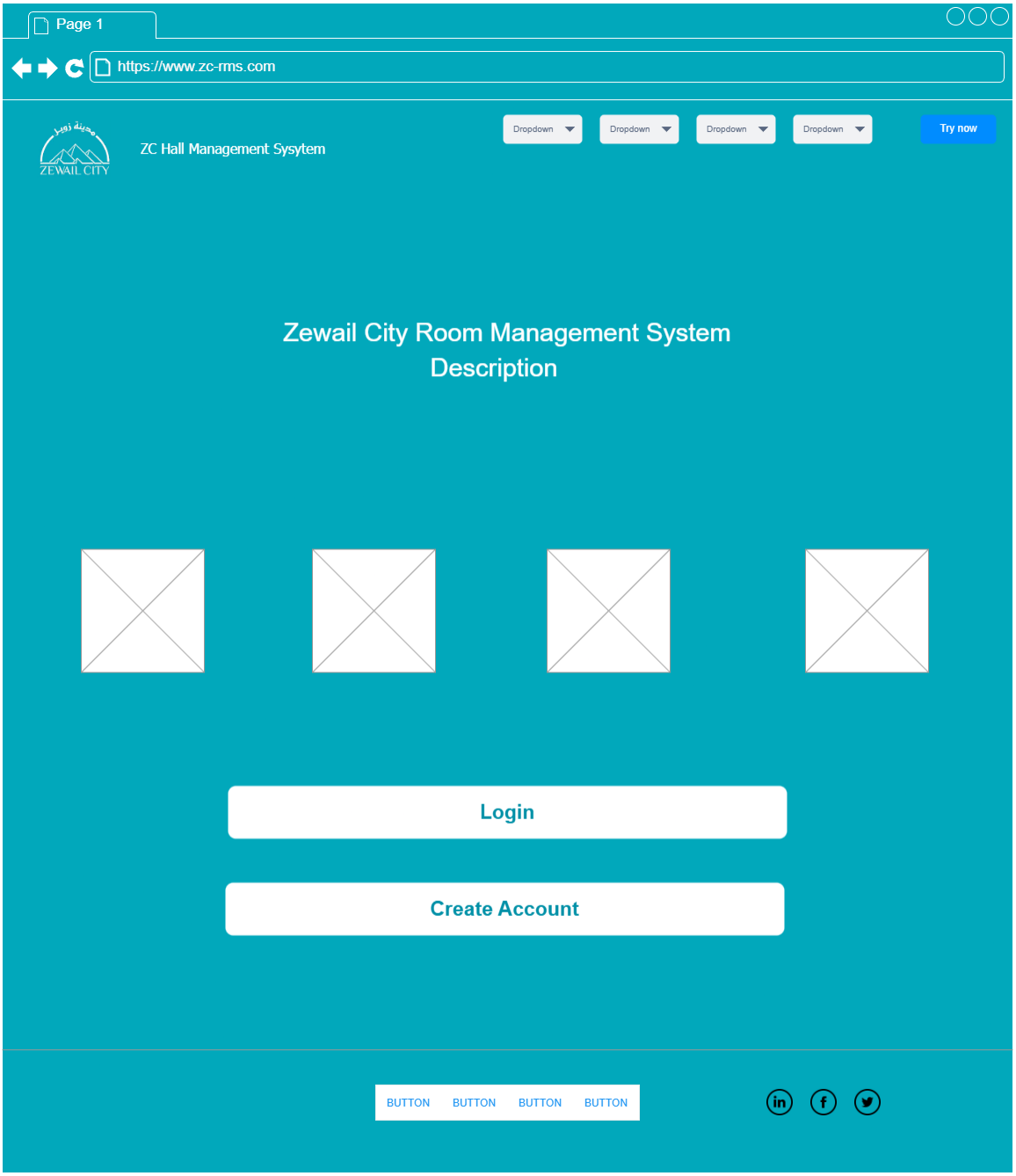
4.create account page

5.import page (hall staff)

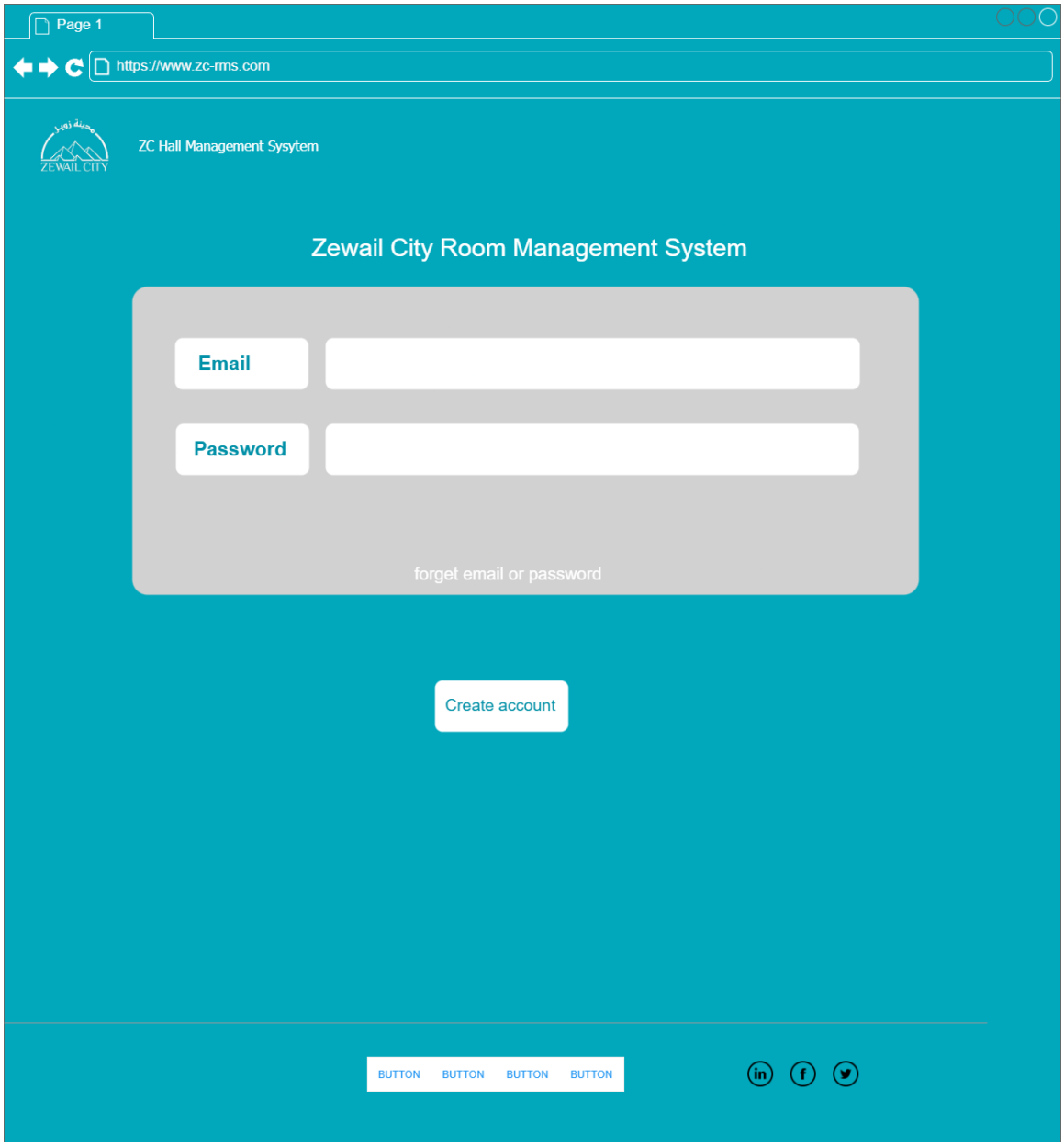
6.schedule (Academic user)

7 academic user main page

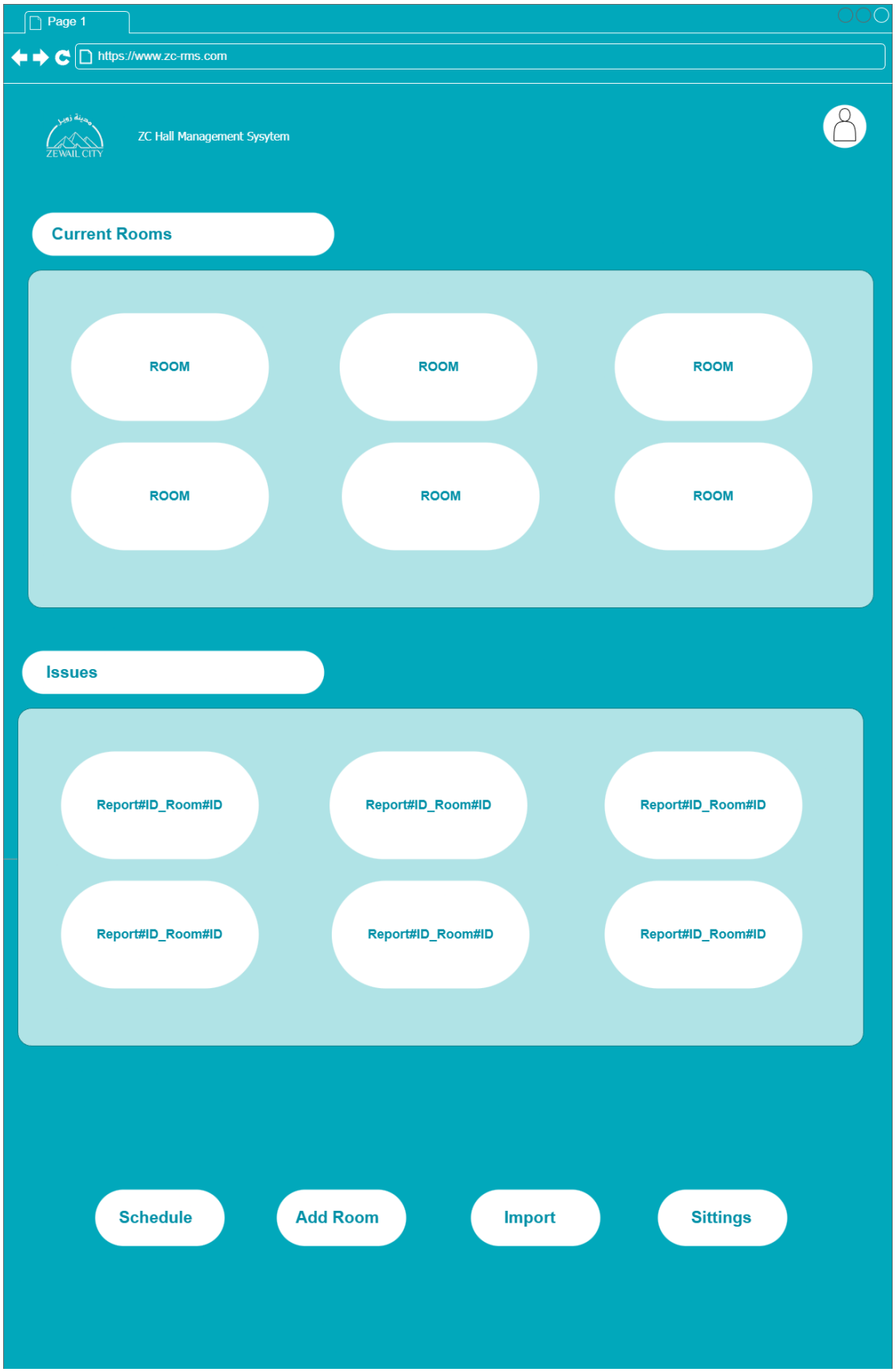
3.

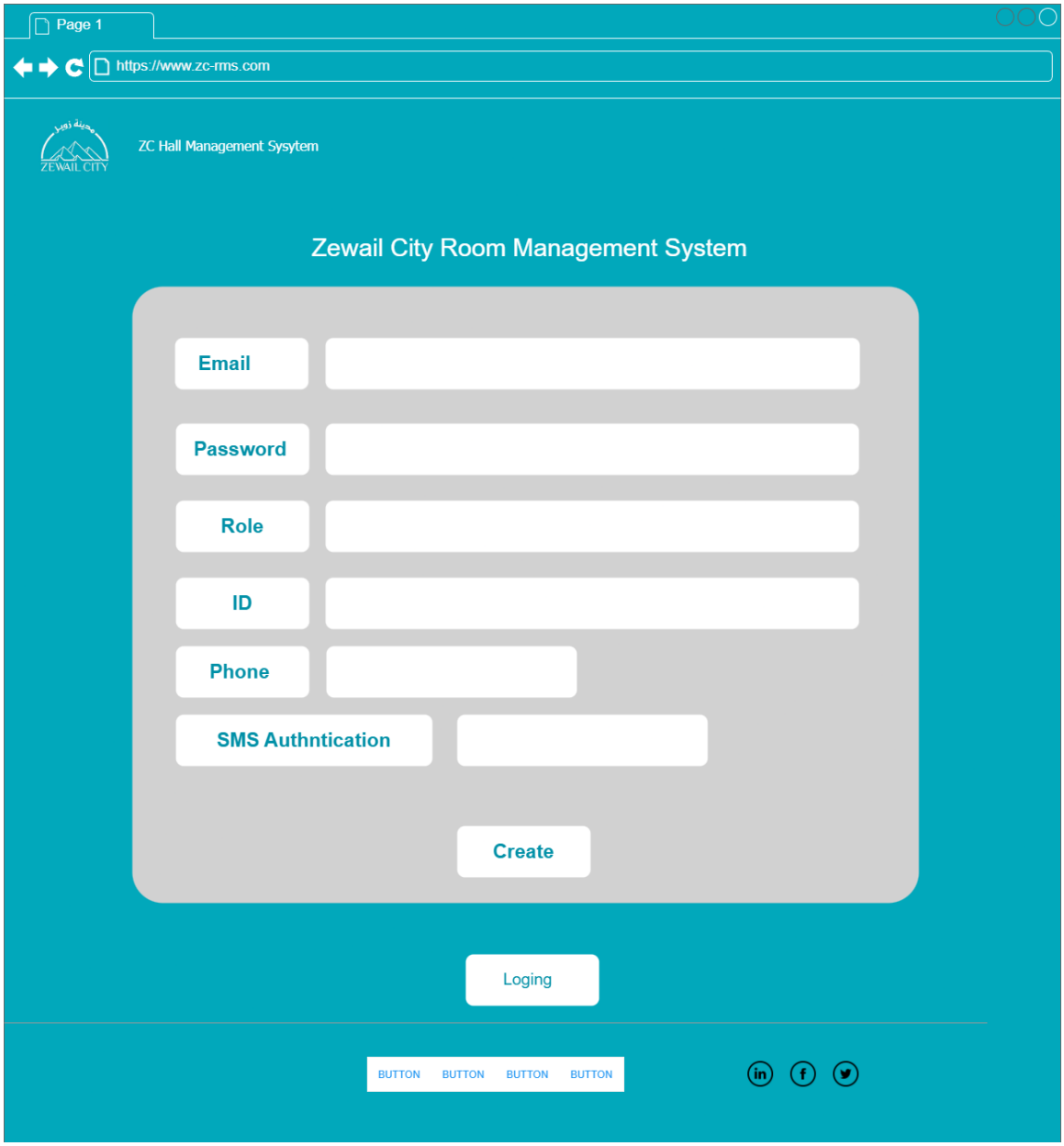


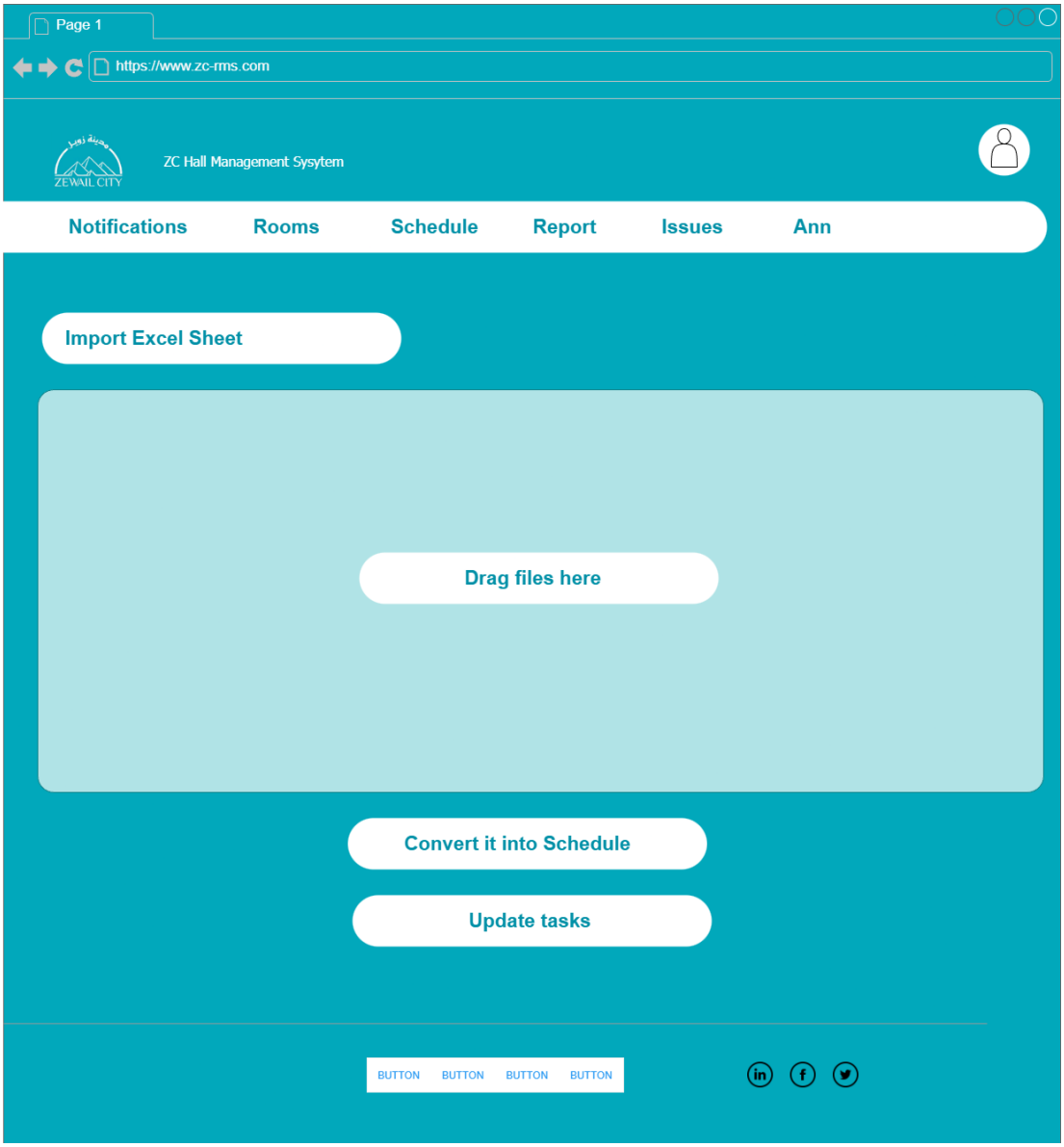
* this Is the wireframe of the landing page of the website

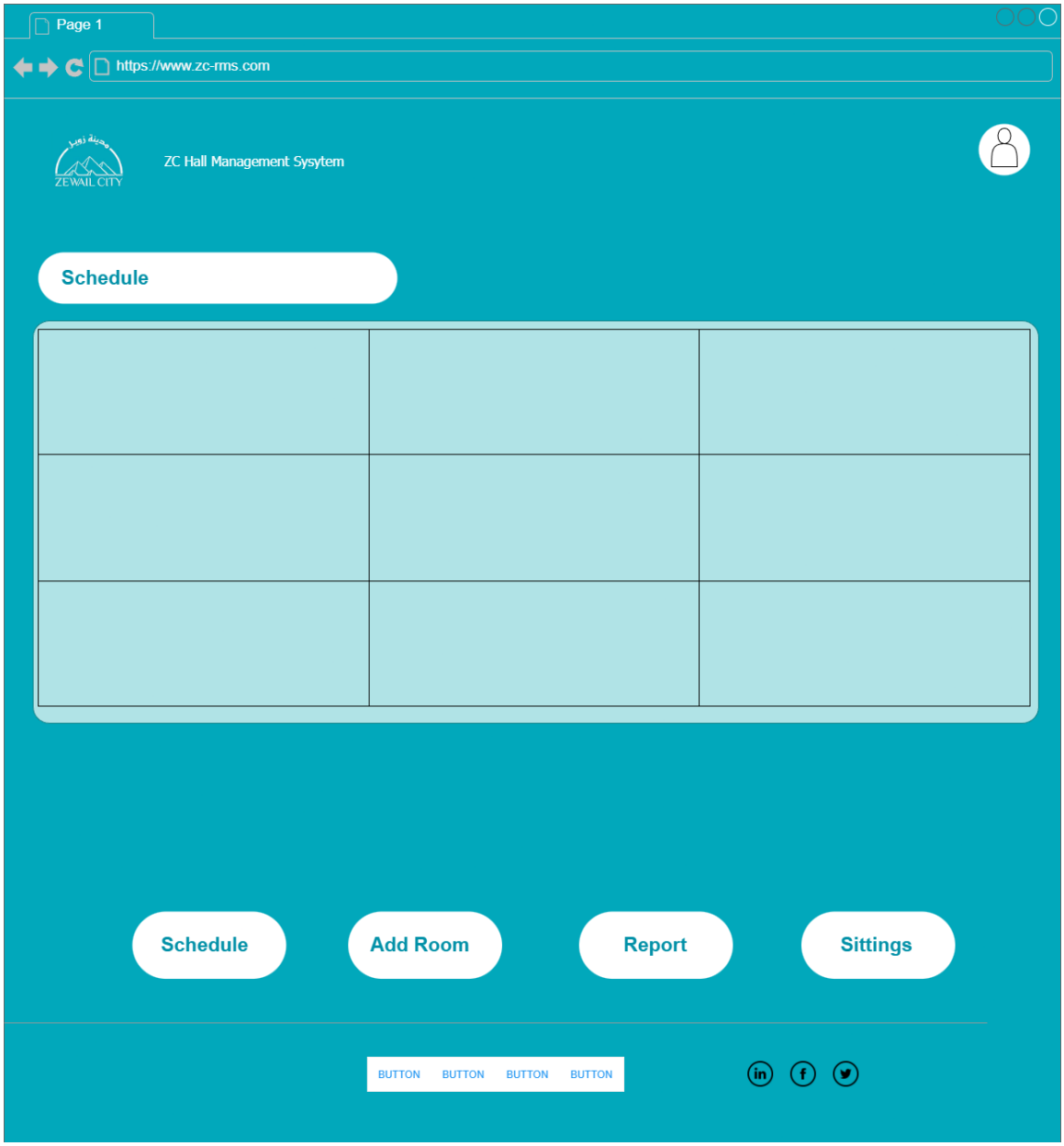

Page 1
https://www.zc-rms.com
ZEWAIL CITY
ZC Hall Management Sysytem
Zewail City Room Management System
Email
Password
forget email or password
Create account
BUTTON
BUTTON
BUTTON
BUTTON

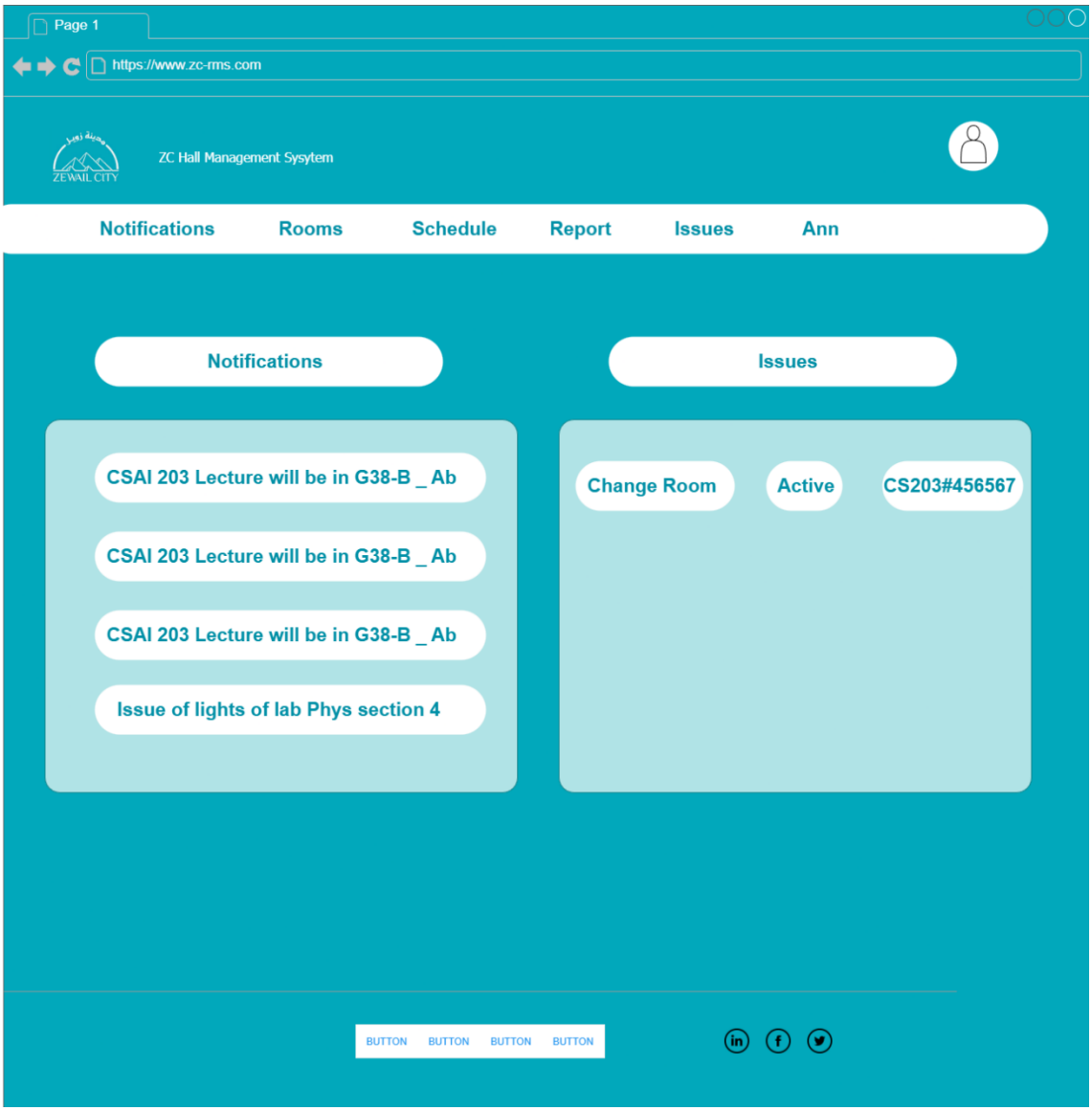
- Login pag


Page 1
https://www.zc-rms.com
ZEWAIL CITY
ZC Hall Management Sysytem
Current Rooms
ROOM
ROOM
ROOM
ROOM
ROOM
ROOM
Issues
Report#ID_Room#ID
Report#ID_Room#ID
Report#ID_Room#ID
Report#ID_Room#ID
Report#ID_Room#ID
Report#ID_Room#ID
Schedule
Add Room
Import
Sittings


Page 1
https://www.zc-rms.com
ZEWAIL CITY
ZC Hall Management Sysytem
Zewail City Room Management System
Email
Password
Role
ID
Phone
SMS Authntication
Create
Loging
BUTTON
BUTTON
BUTTON
BUTTON

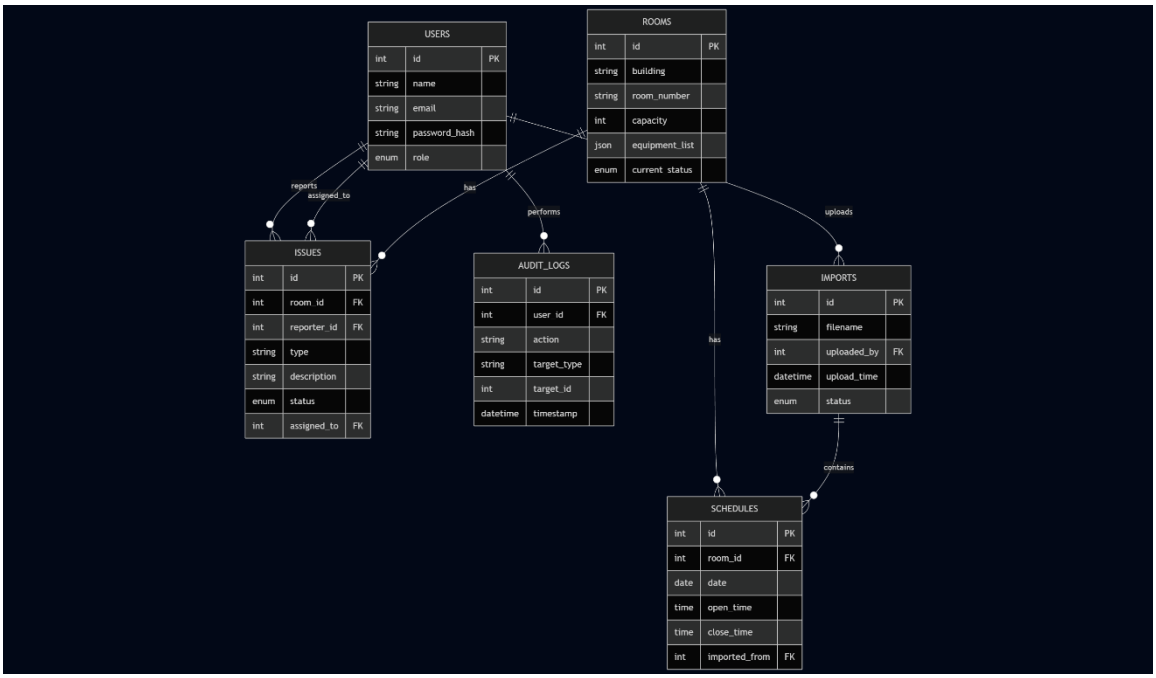

Page 1
https://www.zc-rms.com
ZEWAIL CITY
ZC Hall Management Sysytem
Notifications
Rooms
Schedule
Report
Issues
Ann
Import Excel Sheet
Drag files here
Convert it into Schedule
Update tasks
BUTTON
BUTTON
BUTTON
BUTTON


Page 1
https://www.zc-rms.com
ZEWAIL CITY
ZC Hall Management Sysytem
Schedule
Schedule
Add Room
Report
Sittings
BUTTON
BUTTON
BUTTON
BUTTON


Page 1
https://www.zc-rms.com
ZEWAIL CITY
ZC Hall Management Sysytem
Notifications
Rooms
Schedule
Report
Issues
Ann
Notifications
Issues
CSAI 203 Lecture will be in G38-B _ Ab
CSAI 203 Lecture will be in G38-B _ Ab
CSAI 203 Lecture will be in G38-B _ Ab
Issue of lights of lab Phys section 4
Change Room
Active
CS203#456567
BUTTON
BUTTON
BUTTON
BUTTON

4.4 Data Design

4.4.1 Database Schema / ER Diagram



4.4.2 File Structure / Data Storage Model

The system utilizes a **Relational Database Management System (RDBMS)** for persistent storage.

- **Production:** MySQL is recommended for robust data handling and concurrency.
- **Development:** SQLite may be used for local testing.
- **Structure:** Data is organized into normalized tables with Primary Keys (PK) and Foreign Keys (FK) to maintain referential integrity.

4.4.3 Data Dictionary

1. Table: Users

Column	Type	Constraints	Description
id	INTEGER	PK, AUTO_INCREMENT	Unique identifier for the user.
name	VARCHAR (100)	NOT NULL	Full name of the user.
email	VARCHAR (100)	UNIQUE, NOT NULL	User's email address (used for login).
password_hash	VARCHAR (255)	NOT NULL	Hashed password for security.
role	ENUM	'admin', 'staff', 'ta_prof'	Role determining access permissions.

2. Table: Rooms

Column	Type	Constraints	Description
id	INTEGER	PK	Unique identifier for the room.
building	VARCHAR(50)	NOT NULL	Building name/code.
room_number	VARCHAR(20)	NOT NULL	Room identifier (e.g., 301).
capacity	INTEGER	-	Max student capacity.
equipment_list	TEXT	JSON Array	List of assets (Projector, AC, etc.).
current_status	ENUM	Default 'closed'	'open', 'closed', 'scheduled'.

3. Table: Schedules

Column	Type	Constraints	Description
id	INTEGER	PK	Unique identifier for the schedule slot.
room_id	INTEGER	FK -> rooms(id)	The room being scheduled.
date	DATE	NOT NULL	The specific date of the booking.
open_time	TIME	NOT NULL	Start time of the slot.
close_time	TIME	NOT NULL	End time of the slot.
imported_from	INTEGER	FK -> imports(id)	Reference to the source Excel file.

4. Table: Issues

Column	Type	Constraints	Description
id	INTEGER	PK	Unique issue identifier.
room_id	INTEGER	FK -> rooms(id)	The room having the issue.
reporter_id	INTEGER	FK -> users(id)	Who reported the problem.
type	VARCHAR(50)	-	Type of issue (AC, Lights, etc.).
description	TEXT	NOT NULL	Detailed description of the fault.
status	ENUM	Default 'new'	'new', 'in_progress', 'resolved'.
assigned_to	INTEGER	FK -> users(id), NULL	Staff member fixing the issue.

5. Table: Audit_Logs

Column	Type	Constraints	Description
id	INTEGER	PK	Unique log identifier.
user_id	INTEGER	FK -> users(id)	Who performed the action.
action	VARCHAR(50)	NOT NULL	Action name (e.g., 'login').
target_type	VARCHAR(50)	-	'room', 'issue', etc.
details	TEXT	JSON	Additional context info.

6. Table: Imports

Column	Type	Constraints	Description
id	INTEGER	PK	Unique import identifier.
filename	VARCHAR(255)	NOT NULL	Name of the uploaded Excel file.
uploaded_by	INTEGER	FK -> users(id)	Admin who uploaded the file.
status	ENUM	-	'pending', 'completed', 'failed'.