Zewail City of Science, Technology and Innovation
University of Science and Technology
School of Computational Sciences and Artificial Intelligence

CSAI 203 – Fall 2025
Introduction to Software Engineering

Smart Pitch
Design Document (Phase 3 Deliverable)

Team Number:

Team Members:
Khaled Amr – 202400505
Ahmed Osama – 204200923
Ali Emam – 202401429
Zeyad Mossad – 202401628

Representative Contact Info:

s-ahmed.omohammed@zewailcity.edu.eg

Date:
22/11/2025

1. Introduction

1.1 Purpose of the Document

This document defines how the Smart Pitch web application will be designed and implemented. It translates the SRS into a technical blueprint that supports consistent development and architecture alignment.

1.2 Scope of the Design Phase

Covers:

• MVC-based system architecture
• Component design
• UI layout and wireframes
• Database schema
• UML diagrams (class + sequence)

Implementation, deployment, and testing are not included.

1.3 Intended Audience

This document is intended for:

Developers, team members, instructors, and future maintainers.

1.4 Overview of the Contents

This document begins with an introduction to the purpose, scope, and target audience. It then provides:

Includes the system overview, architecture, MVC structure, UML diagrams, UI wireframes, database schema, and a summary of design decisions.

2. System Overview

2.1 Brief Description of the System

Smart Pitch is a web application allowing users to search, filter, and book football pitches. Owners manage pitch listings, while admins oversee system operations. A relational database stores users, pitches, and bookings.

2.2 Key Design Goals and Constraints

Design Goals:

- Scalability:
  The system should support future expansion such as integrating online payments, adding mobile app versions, or increasing the number of users and pitches.
- Modularity:
  Following the MVC pattern ensures clear separation between system logic, user interface, and data handling, making the system easier to debug and extend.
- Usability:
  The interface should be simple, intuitive, and user-friendly for players, pitch owners, and administrators.
- Maintainability:
  A clean MVC structure, organized modules, and a well-defined database schema ensure that future updates and enhancements can be implemented efficiently.
- Security:
  Sensitive information (passwords, user data) must be protected using encrypted storage and secure authentication mechanisms.
  Communication through HTTPS should be enforced where possible.

Constraints:

Flask MVC pattern, HTML/CSS views, relational DB (SQLite/MySQL), owner-only pitch management, no online payment integration

3. Architectural Design

3.1 System Architecture Diagam

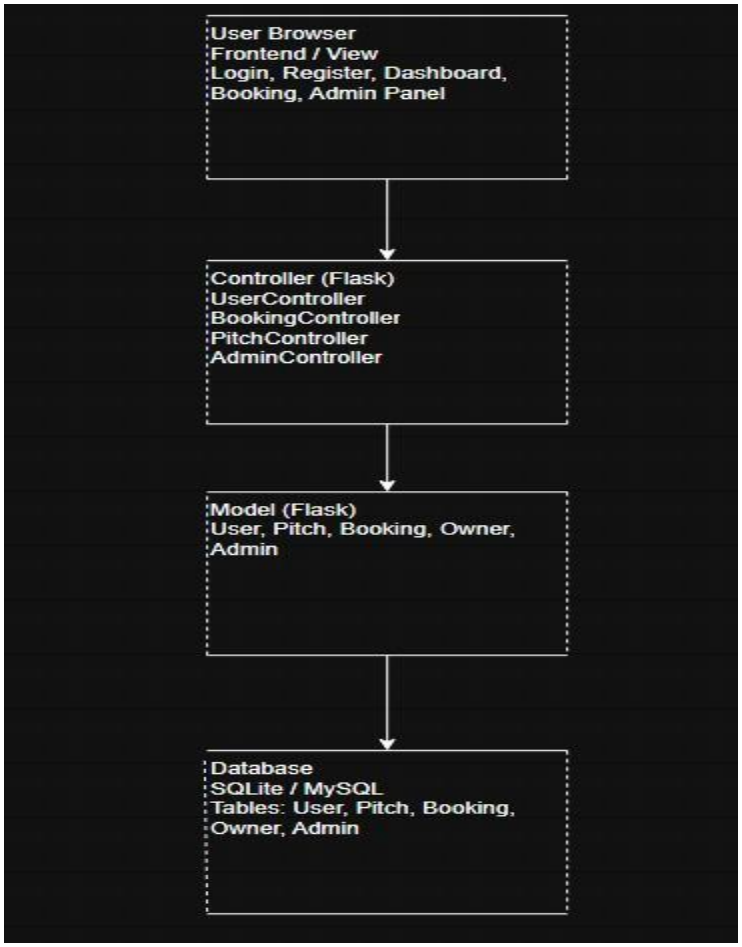
Layers:

• **View:** HTML/CSS templates

• **Controller:** Flask route logic

• **Model:** Data validation, business logic

• **Database:** Users, pitches, bookings, admins



3.2 Architecture Style & Components

Monolithic Flask MVC. Components: UI pages, Flask controllers, model classes, and DB handlers.

3.3 Technology Stack and Tools

HTML/CSS, Python Flask, SQLite/MySQL, Draw.io, VS Code, Git/GitHub.

4. Detailed Design

4.1 Model–View–Controller (MVC) Design Pattern

4.1.1 Description of MVC Pattern

The Smart Pitch system follows the Model–View–Controller (MVC) architectural pattern to ensure a clean separation between user interface components, business logic, and data management. This structure improves maintainability, modularity, and scalability across the project.

4.1.2 Mapping of Project Components to MVC

Classes + DB Layer → Model
HTML/CSS Templates → View
Flask Routes → Controller

4.1.3 Responsibilities of Model, View, and Controller

Model Responsibilities

- Manage business rules (booking logic, schedule conflicts, etc.)
- Interact with the database (read/write operations)
- Return processed data to controllers

View Responsibilities

- Display system output to users
- Provide forms, buttons, and UI components

- Show lists, details, error messages, and confirmations

Controller Responsibilities

- Call appropriate model functions
- Select the correct view to display results
- Maintain overall workflow and logic of the application

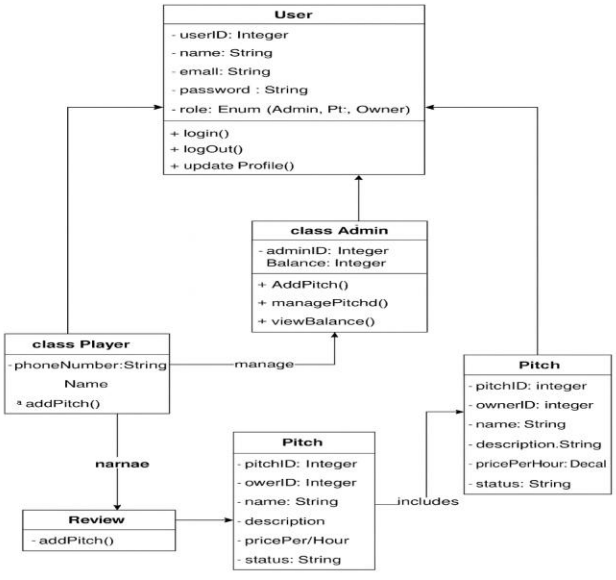
4.1.4 Interaction Between Components

Flow:

1. ***User interacts with the View***
2. ***Controller receives the request***
3. ***Controller communicates with the Model***
4. ***Model processes the data and sends back the result***
5. ***Controller selects the appropriate View***
6. ***View displays the final result to the user***

4.2 UML Diagrams

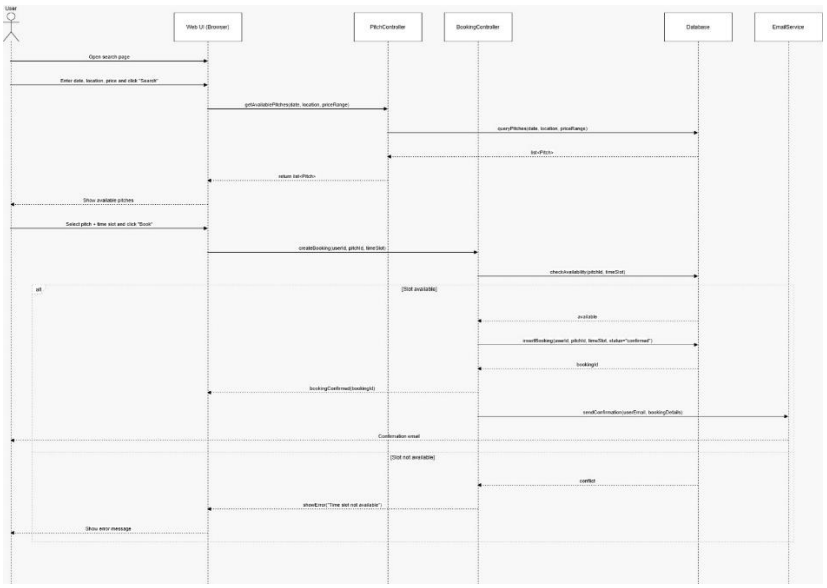
4.2.2 Detailed Class Diagram



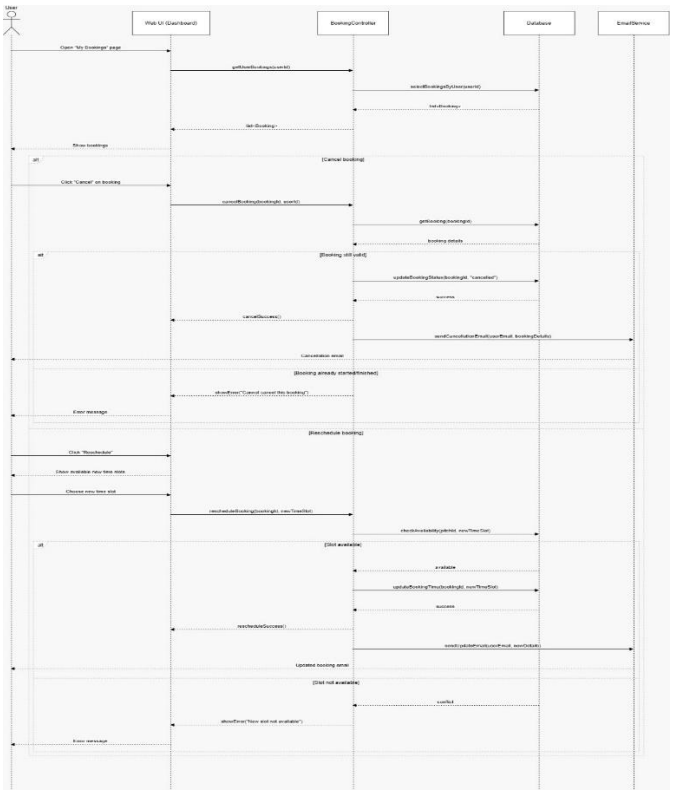
4.2.3 Sequence Diagrams

The system requires five sequence diagrams for core interactions.

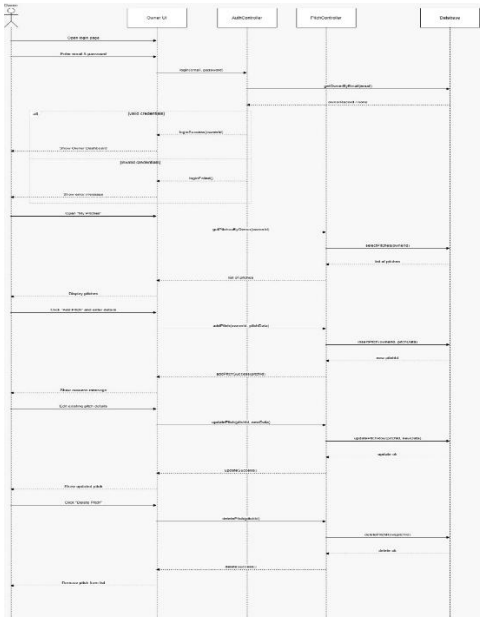
1-Search & Book Pitch



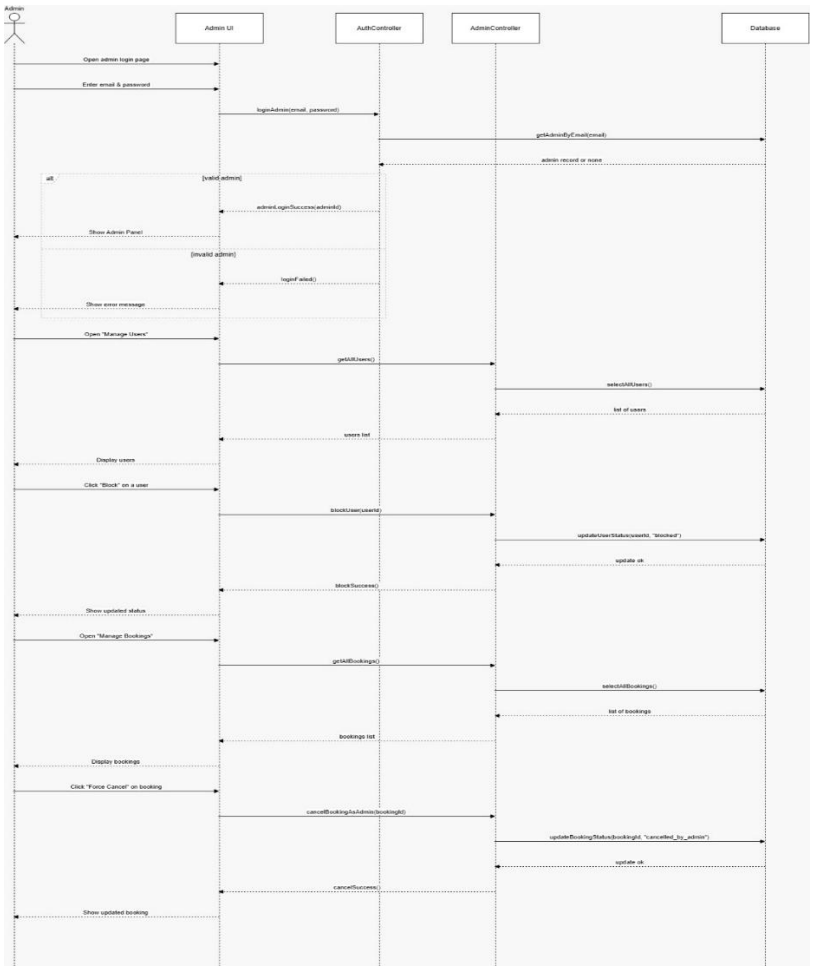
2- **Cancel / Reschedule Booking**



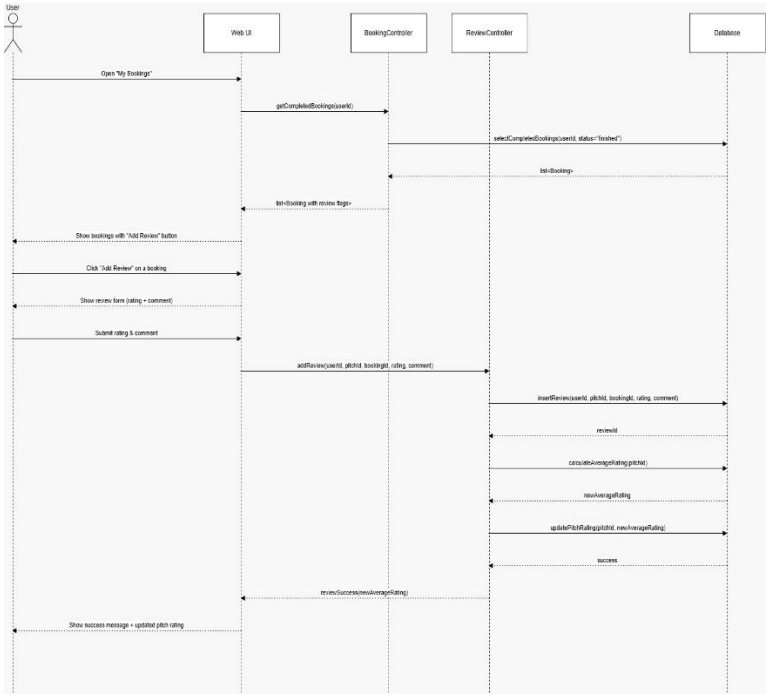
3- Pitch Management by Owner



4 – Admin Managing Users & Bookings



5 – Reviews & Ratings



4.3 Wireframes / Mockups

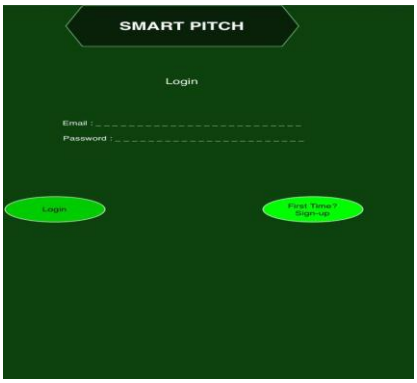
The Smart Pitch system UI consists of six major screens.

4.3.1 Wireframes / Mockups

The system UI consists of four major screens. Wireframes should illustrate layout, key elements, and user flow.

Insert wireframes in the spaces below:

1-Login page



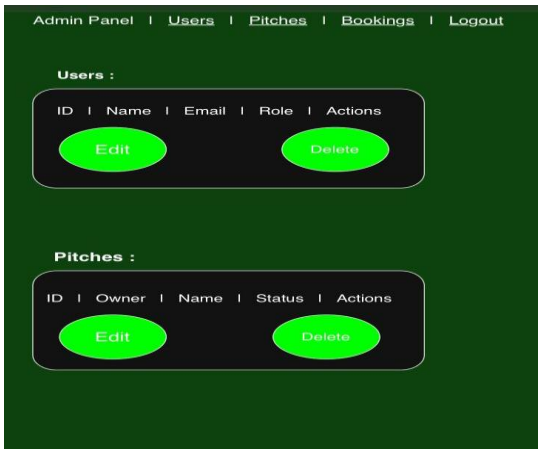
2-Register Page

3- Home page Search page

4-Booking Page

5-Owner Page

6-Admin Page



4.4.1 Database Schema / ER Diagram

The Smart Pitch system uses a relational database (SQLite or MySQL) to store all information related to users, pitch owners, pitches, and bookings. The database schema is designed to ensure data consistency, efficient data retrieval, and scalability as the system grows.

Main Entities

The database contains the following core entities:

- **Users** – stores player and owner accounts
- **Pitches** – stores football pitch information
- **Bookings** – stores reservation records
- **Admins** – stores administrator accounts

Entity Relationships

- One **User** can have **many Bookings** (1-to-many)
- One **Pitch** can have **many Bookings** (1-to-many)
- One **Owner** (user with role = "owner") can manage **many Pitches** (1-to-many)

These relationships link bookings to both the user and the pitch being reserved.

4.4.2 File Structure / Data Storage Model

USER TABLE

Field	**Type**	**Description**
user_id (PK)	INT	Unique user identifier
full_name	VARCHAR(100)	User’s full name
email	VARCHAR(120)	Unique login email
password	VARCHAR(255)	Hashed password
role	ENUM('user','owner')	Determines user type
created_at	DATETIME	Account creation timestamp

PITCHES TABLE

Field	**Type**	**Description**
pitch_id (PK)	INT	Unique pitch identifier

Field	Type	Description
owner_id (FK)	INT	References users.user_id
pitch_name	VARCHAR(100)	Name of the pitch
location	VARCHAR(150)	Pitch location
price_per_hour	DECIMAL(10,2)	Hourly cost
description	TEXT	Optional description
created_at	DATETIME	Timestamp of pitch creation

BOOKINGS TABLE

Field	Type	Description
booking_id (PK)	INT	Unique booking identifier
user_id (FK)	INT	References users.user_id
pitch_id (FK)	INT	References pitches.pitch_id
date	DATE	Booking date
time_slot	VARCHAR(20)	Reserved time range
status	ENUM('confirmed','cancelled')	Booking status
created_at	DATETIME	Timestamp of booking

ADMIN TABLE

Field	Type	Description
admin_id (PK)	INT	Unique admin identifier
username	VARCHAR(50)	Admin login name
password	VARCHAR(255)	Hashed password

Field	Type	Description
created_at	DATETIME	Timestamp of admin creation

4.4.3 Data Dictionary

Attribute	Description	Type
user_id	Unique identifier for each user	Integer
full_name	User’s full name	String
email	Login email	String
role	User type (user/owner)	String
pitch_id	Unique identifier for each pitch	Integer
location	Geographic location of the pitch	String
price_per_hour	Hourly booking cost	Float
booking_id	Unique identifier for each booking	Integer
date	Reservation date	Date
time_slot	Reserved time slot	String
status	Booking status	String
owner_id	Unique identifier for pitch owners	Integer

5. Conclusion

5.1 Summary of Design Phase

The design phase transformed the Smart Pitch system requirements into a complete technical plan using a clear MVC structure and a relational database. The architecture, UML diagrams, UI wireframes, and database schema together define how the system will function and how its components interact. This design provides a solid foundation for the implementation phase, ensuring that development will follow a consistent, organized, and scalable structure that meets the project’s goals.