CPT202 Assignment 2
Group Report for Software Engineering Group Project

2023/2024 Semester 2
<Online Booking Service for a Sport Centre>
<2024.5.10>

URLs:
We provide you the URL for each sub-pages of our system, but each URls will jump to the login page because you need to login first. We prefer you to use the Google Chrome browser to visit our website because it is more stable.
http://121.43.129.89:9012/login/html/logtest.html
http://121.43.129.89:9012/home/html/homePage.html
http://121.43.129.89:9012/piao/html/football_booking.html
http://121.43.129.89:9012/community/html/community.html
http://121.43.129.89:9012/user/html/personal_information_display.html

We also give you two default account and please feel free to add more by signing up.

Default account	*Username*	*Password*
Normal user	Usertest	User123456
Administrator	Admintest	Admin123456

Group number: <07>
Submitted by: <Peizheng Zhao, 2144741>
Group Members:
1. <Peizheng Zhao, 2144741>
2. <Jingjie Qiu, 2141674>
3. <Kuo Guo, 2142141>
4. <Bowen Fang, 2144218>
5. <Ming Wang, 2143923>
6. <Jiayu Shen, 2033963>
7. <Bangxu Tian, 2144648>
8. <Gaoping Zhou, 2142077>

Contents

Introduction

This section will give a brief introduction of our project.

Problem Statement

There is one college gym lacks an online booking and management system. This cause that current ticket booking procedures at universities is difficult for the students. There is no doubt that this will discourage students from exercising and communicating with those who enjoy doing sports activity.

Aims of The Project

The aim of our project is to design a web-based online booking service system for sport center to help students and teachers book tickets more easily. In addition, our system also supports administrator directions to help better manage sports activity, tickets order, and user information to maintain the normal operation of the entire system.

Project Scope

Our project scope included the design and development of a web-based university stadium booking platform with integrated payment system, real-time event updates, and document queries for monitoring and management. The system will serve as a centralized booking hub for all events at the University stadium. Use case diagram is shown in Appendix A1 Use case diagram.

User Characteristics

Understanding the users of the online booking service system is important for its design and development. The following are the primarily users of our System. **1) Users(Students and staffs):** As a user, I want to book the ticket smoothly. **2) Administrator:** As an administrator, I want to manage the varies of information in the system.

Assumptions and Dependencies

The following assumptions have been made regarding users and their use of the web-based booking tickets online system. **1) Environment:** It is assumed that the university provides a stable and secure network environment suitable for hosting the online booking system. **2) Hardware and software requirements:** It is assumed that the necessary hardware and software requirements should be able to run the system effectively.
To ensure the reliability and stability of the booking system, dependencies must be considered in the following ways: **1) Server operation status:** It is assumed that the server is running normally and can support the system operation. **2) Network connection between the server and the client:** It is assumed that the network connection between the server and the client is stable and reliable. **3) Stability and maintenance updates of system hardware and software:** It is assumed that the hardware and software of the system are well maintained and regularly updated. 3) **A stable Internet connection from the user's perspective:** It is assumed that the user's Internet connection is stable and can support the booking process.

Project Risks

The development of the university ticket booking system presents a variety of risks that could affect the project's success. **1) Technical risks**: The integration of the new system with the existing university infrastructure can be more complex than expected, resulting in delays. **2) User resistance**: The system may be susceptible to cyber-attacks, which could compromise student data and ticketing information.

Architectural Design

In this section, both functional and non-functional requirements will be mentioned and their impact on the architecture will be discussed [1]. Then, the architectural design decisions made according to them will be explained.

Functional requirements:

- Users can get information from the database about the tickets they have booked as well as information about how they booked the tickets, such as the type of event and venue. Also, when a user books or deletes a ticket, their request is updated to the database.
- Administrators can view user information and user booking history through the data stored in the database, which can create a file to store daily order information. In addition, administrators can restrict the status of tickets for the sport and can also add some sport types.

Non-functional requirements:

- **Performance:** As long as there is no obvious lag during using this system when user use it, it is accepted.
- **Security:** It is a significant part for the actual online booking system, but for this part we mainly focus on the function requirements, which make us feel much easier in testing it.
- **Maintainability:** Maintainability may be important at this stage, but it is not necessary. Systems can be designed with components that are prepared for change where appropriate.

Caching or queuing server:

Caching or queuing servers aren't used due to functional requirements. Database updates wait until ticket booking is completed. Data size for booking is small. For non-critical data like user reviews, cookies can cache without a server.

System Architecture

1. Architecture pattern

This web-based system adopts the MVC architectural pattern (as shown in Appendix A2 MVC architecture diagram), chosen for its prominence and suitability for web applications. MVC divides the system into three components: Model, View, and Controller, facilitating clear and structured development, particularly beneficial for large and complex projects.[1] This pattern enables the separation of concerns, allowing independent management of data, user interface, and control logic.

2. Advantages

- **Enhanced maintainability:** Due to the separation of concerns, when changes need to be made to a certain part of the application, other parts remain unaffected, simplifying the maintenance and debugging process.
- **Increased reusability:** Independent models and views can be reused across multiple applications, reducing code duplication and enhancing efficient use of resources.
- **Promotes collaboration:** In team development, different developers can work simultaneously on the model, view and controller without interfering with each other, improving team productivity.

Frontend System

In this system, we use HTML (HyperText Markup Language), CSS (Cascading Style Sheets) and JS (JavaScript) technologies for building the structure and style of the web pages and interact the system .

1. HTML for layout

HTML is used to create the structure and element of a web page. In the fronted system, we separate this system into 4 parts. **1)** In the *login* processing, we made three interfaces including the login interface, the registration interface, and the password reset interface. **2)** In the *homepage*, we write a homepage for the system and an introduction interface for each sport. **3)** In *booking tickets* processing, we develop 4 pages for each sport activity involving booking page, ticket payment page, check page, payment page. **4)** In *administrator* processing, we build 3 extra interfaces involving activity page, user page, and order page.

2. CSS for style

CSS is used to design and modify the style and layout of web pages. Each interface has a CSS file corresponding to it. For example, in homePage.css, we mainly use the following layout technologies. **1) Flexbox (Flexible Box Layout):** It is one of the layout pattern which used to align and distribute child elements within a container, even if their size is unknown or changes dynamically. **2) Absolute Positioning:** Absolute positioning allows an element to be positioned relative to its nearest non-static positioning ancestor.

3. JS for dynamic

JS is used to add interactive features to web pages, including functionalities like booking tickets, with almost all of these features being written within the document. The requests sending to the backend is also written in the JS.

Backend System

In this system, we use the following backend technologies:
1. Spring Boot framework
Spring Boot is an open source Java infrastructure designed to simplify the initial setup and development of new Spring applications. This is the basic framework of our system. Used extensively, as indicated by the @RestController and @RequestMapping annotations in the controller classes.

2. MyBatis-Plus
Utilized for database operations, as evidenced by the BaseMapper interfaces in the DAO (Data Access Object) classes. Compared with JPA, MyBatis-Plus provides an auto-fill function, which can automatically fill fields, such as creation time, update time, etc. That's one of the reasons we chose it.

High Level Database Design

We create relative database to achieve these functions with 10 tables in a scheme. The ER diagram will be shown in Appendix A3 ER diagram. We have the following relationships:
A *UserEntity* can have multiple *OrderEntities* establishing a one-to-many relationship. A *UserEntity* can have multiple *CommunityEntities*, another one-to-many relationship. A *UserEntity* may be associated with *UserLoginHisEntity* records, indicating the login history, which is also a one-to-many relationship. *CommunityLikeEntity* establishes a many-to-one relationship with *CommunityEntity*, indicating which users liked a particular community post. *OrderEntity* may have a foreign key relationship with *DiscountCouponEntity* if an order uses a discount coupon.

Software Design

Software Modules

1. Class grouping
First, according to the fundamental requirements, core functions and data are grouped into these important classes: *UserAccount, TicketBooking, Administration, Login, Community, MasterFile, StatisticalReport.*
This grouping approach aligns with principles in object-oriented design, which emphasize encapsulation and modularity for improved system robustness and scalability [2].
To ensure each class operates as an independent unit with minimal dependency on others, cohesion and coupling are carefully considered. This approach creates robust class groupings with a high degree of internal cohesion and low external coupling, contributing to system modularity and ease of maintenance. Additionally, the likelihood of changes to specific functions is assessed to reduce the impact on the overall system structure, aiming to minimize future development costs.

2. Module grouping
Module grouping, similar to class grouping, considers factors such as cohesion, coupling, and team collaboration, but with differing weights assigned to each factor. Principles of low coupling and high cohesion are particularly emphasized in module formation [3].
Modules are organized to facilitate collaboration among team members, with functions requiring similar skill sets grouped together within modules. However, inter-module communication ensures that every team member is involved in all aspects of the project. Class and module diagram is shown in Appendix A4 Class and module diagram.

High-level Process Flow

The sports facility reservation system offers key functionalities for users and administrators. Users can register, log in, and book various sports facilities with ease. They can search venues based on specific criteria, make reservations, and manage their bookings, including cancellations and rescheduling. The system also provides a user community where users can also share their thoughts and recommendations. For administrators, the system includes a management center where they can oversee facility listings, monitor bookings, and manage user accounts[4]. A simple flow chart for high-level process flow is shown in the Appendix A5 High-level flow

A5 High-level flow.

Software Support Services

1. Database-related services

These services encompass the design, creation, maintenance, and backup of databases. MySQL is a commonly used database management system that supports these functions. It ensures that data is stored securely and can be retrieved efficiently, contributing to the smooth operation of the system.

2. Message queue related services

These services facilitate the asynchronous exchange of messages between different components of the system, enhancing its scalability and responsiveness.

3. Webpage navigation related services

These services encompass website design, layout, and user interface (UI) development, ensuring users can navigate the sports facility reservation system intuitively. It includes a well-designed UI facilitates smooth transitions between web pages.

4. Hosting-related services

Hosting-related services offer the infrastructure necessary for the system to operate on the web. These services include both cloud-based hosting and managed hosting providers, ensuring reliable system uptime and performance.

Coding Structure and Convention

This sub-section outlines the standard and coding conventions in the system’s development which helps the whole team to better understand other’s code and mire easily to combine the code.

The sport center booking system was developed using the Model-View-Controller (MVC) architecture, Repository pattern, Service Layer, Configuration Files, Static Resources.

1. MVC Architecture

The Model-View-Controller (MVC) pattern is adopted to separate concerns within the application. The Model encapsulates data-related operations, the View handles user interface elements, and the Controller manages business logic and communication between the Model and View [2].

2. Repository Pattern

The Repository pattern is implemented using My-Batis Plus. In the service reservation system, the mapper interface takes on the role of the Repository pattern, encapsulating data access logic to maintain separation of concerns with business logic.

3. Service Layer

The Service Layer acts as an intermediary between the Controller and the Repository. It contains the business logic, processing data and applying business rules before interacting with the Repository. This abstraction promotes code organization and testability.

4. Configuration Files

Configuration files define various configurations for the project, including database settings, application properties, and security configurations.

5. Static Resources
This component includes files that contribute to the user interface, such as HyperText Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript, and images.

For the convention, we strictly follow the naming convention and readable comments to clarify the codes.

Software Configuration and Production Environment

1. Software configuration

Table 1 Software configuration

OS/Software	*Version*
Microsoft Windows	10.0.19045
Java	1.8.0_202
MySQL	8.3.0

The website operates on Microsoft Windows 10.0.19045, providing a stable and widely adopted environment for seamless operations. The application is built using Java 1.8.0_202, ensuring stability and broad compatibility across various systems. The database engine used is MySQL 8.3.0, known for its robustness and scalability, supporting complex queries to manage bookings and user data effectively.
Git is used for version control, enabling developers to collaborate and manage code changes efficiently. Maven automates build processes and handles project dependencies, streamlining development workflows. Docker facilitates containerization, allowing for consistent application deployment across different environments. GitHub Actions automates the build and deployment pipeline, ensuring seamless transitions from development to production.

2. Sever configuration
The production environment is hosted on an Aliyun server, providing a reliable and scalable platform. The server applies automatic updates to maintain the latest security patches and stability. Host the MySQL database separately to enhance isolation and scalability, ensuring that database operations do not affect application performance. In addition, the database is backed up regularly to prevent data loss and ensure recovery in the event of system failure or damage.

Software testing

1. Testing procedure
Upon completion of any Product Backlog Item (PBI), the responsible developer must run their unit tests to ensure the code's correctness and reliability. When a branch is merged into the master branch or a new commit is made to the master, we conducts full suite of unit and integration tests to validate that the changes haven't caused any unexpected issues or regressions [5].
After each sprint, the development team conducts acceptance testing, where they collectively run and inspect the entire product. This comprehensive check ensures that the software meets the acceptance criteria for all the PBIs completed during the Sprint.
Unit and integration tests are written in JUnit and Spring Boot, allowing for automated test execution and seamless integration with continuous integration/continuous deployment (CI/CD) pipelines. Each module in the codebase contains a src/test directory, with subfolders for unit tests (unit test) and integration tests (integration test) to organize test cases by their purpose.

2. Unit testing
Unit testing is the first step in software testing by testing individual components or methods individually to ensure that they work as expected. It involves creating small test cases that validate specific functionality, covering a range of scenarios, including normal cases and edge cases. In the sports facility reservation system, unit testing checks

critical business logic within methods and classes. Tools like JUnit are used to automate these tests. Table 2 is a typical example of our unit test.

Table 2 Example of unit testing

Test name	testInvalidUserName()				
Precondition	When user is registering				
Inputs	(param username) Longer than 20	Shorter than 3	Not contain Capital letters	Not contain lowercase letters	Not contain numbers
Outputs	HTTP responses				
Expected	exception	exception	exception	exception	exception
Actual	exception	exception	exception	exception	exception
Test process	with JUnit 5				

2. Integration testing

Integration testing examines how different components or modules within a system interact with each other. This type of testing verifies that these components operate together as intended and that data flows smoothly and accurately between them. In the sports facility reservation system, integration testing includes database communication and HTTP request handling. The integration tests validate that the system can retrieve and store data correctly and that various services interact without issues. Table 3 shows an example of integration testing.

Table 3 Example of integrating testing

Test name	LoginControllerTest()
Precondition	When the URLs used in the integration testing are configured to require authentication. That is, not everyone can access them.
Inputs	HTTP GET requests for the URLs
Outputs	HTTP responses
Expected	HTTP 302 for all URLs accessed (which indicate that the user will be redirected to the somewhere else, namely the login page)
Actual	HTTP 302 for all URLs accessed (which indicate that the user will be redirected to the somewhere else, namely the login page)
Test process	with JUnit 5

3. Acceptance testing

Acceptance testing is the final step of the testing process and focuses on ensuring that the system meets business requirements and aligns with user expectations. This phase involves simulating real-world conditions and executing a complete product to verify that it meets the acceptance criteria established by the Product Backlog Item (PBI).

In the sports center booking system, acceptance testing is conducted after each Sprint to ensure that newly implemented features align with the user stories and business objectives defined by the PBIs. During acceptance testing, the team interacts with the system through a browser to simulate user experiences, validating key functionalities such as booking facilities, navigating the user center, exploring the community features, and managing reservations.

Conclusion and Future work

The development of the online sports venue booking system successfully fulfilled all specified user needs from the project description, while adding innovative features like real-time heatmaps and pie charts for booking statuses, and a community module for user interactions such as posting, rating, and liking. Despite these accomplishments, the project has identified several areas for future enhancement. For example, the UI could be refined to offer a more detailed and unified appearance and administrative functions need expansion. Additionally, incorporating user feedback continuously and upgrading system functionalities.

References

[1] De Lucia, A., & Ferrucci, F. (Eds.). (2009). Software Engineering: *International Summer Schools, ISSSE 2006-2008, Salerno, Italy*, Revised Tutorial Lectures. Springer.

[2] Gamma, E. (1995). Design patterns : elements of reusable object-oriented software. Addison-Wesley.

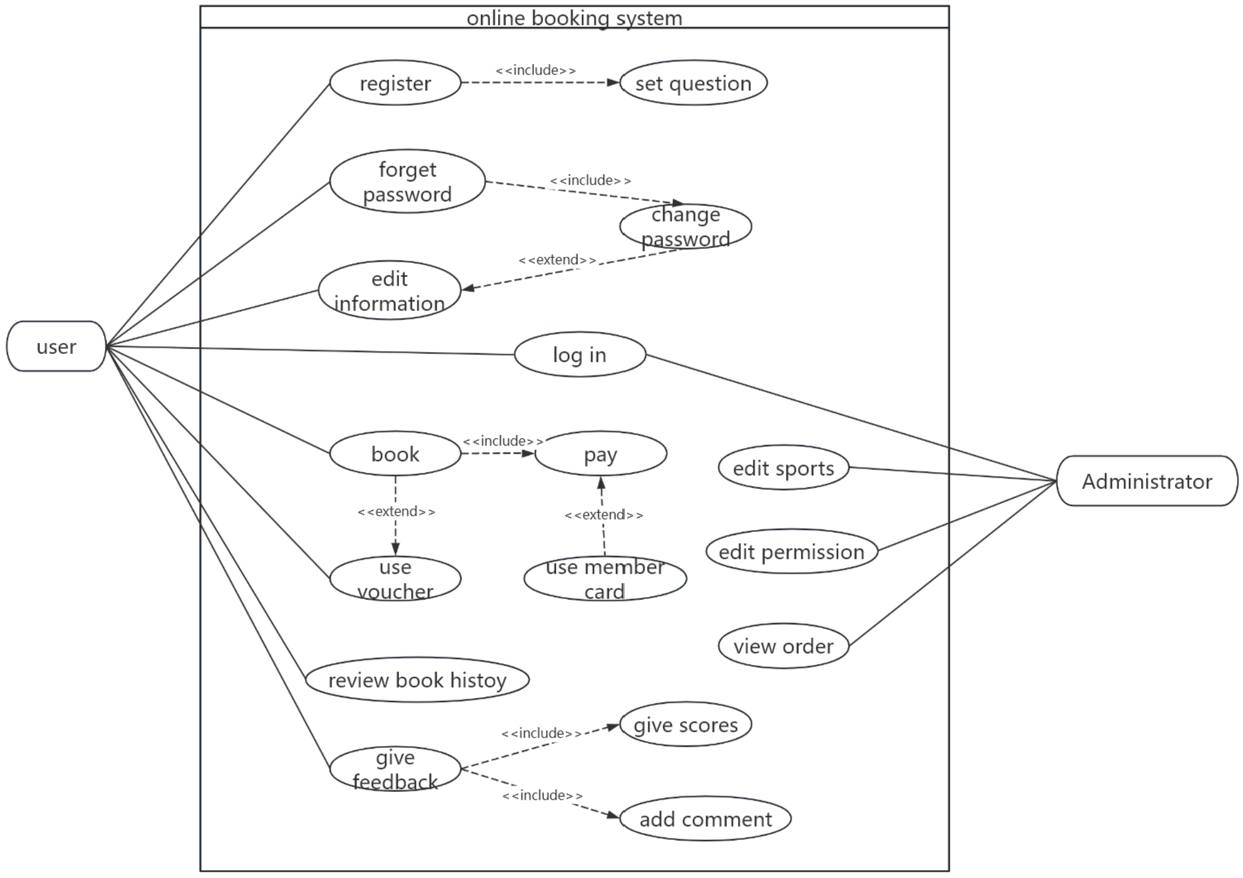
[3] Saca, M. A. (2017). Refactoring improving the design of existing code. 2017 IEEE 37th Central America and Panama Convention (CONCAPAN XXXVII), Central America and Panama Convention (CONCAPAN XXXVll), 2017 IEEE 37th, 1–3. https://doi.org/10.1109/CONCAPAN.2017.8278488.

[4] Evans, E. (2020). Domain-driven design : tackling complexity in the heart of software.

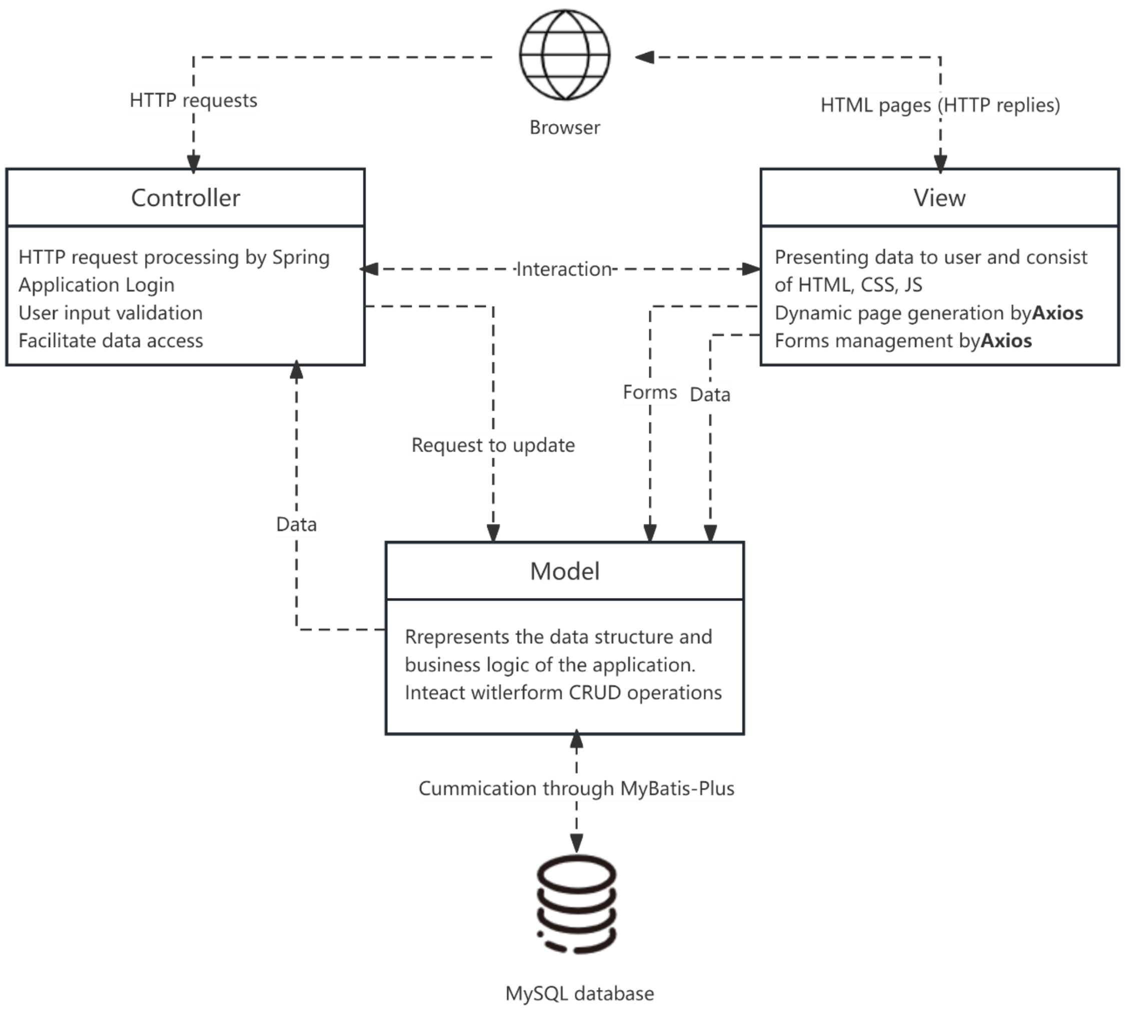
[5] Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.

Appendix

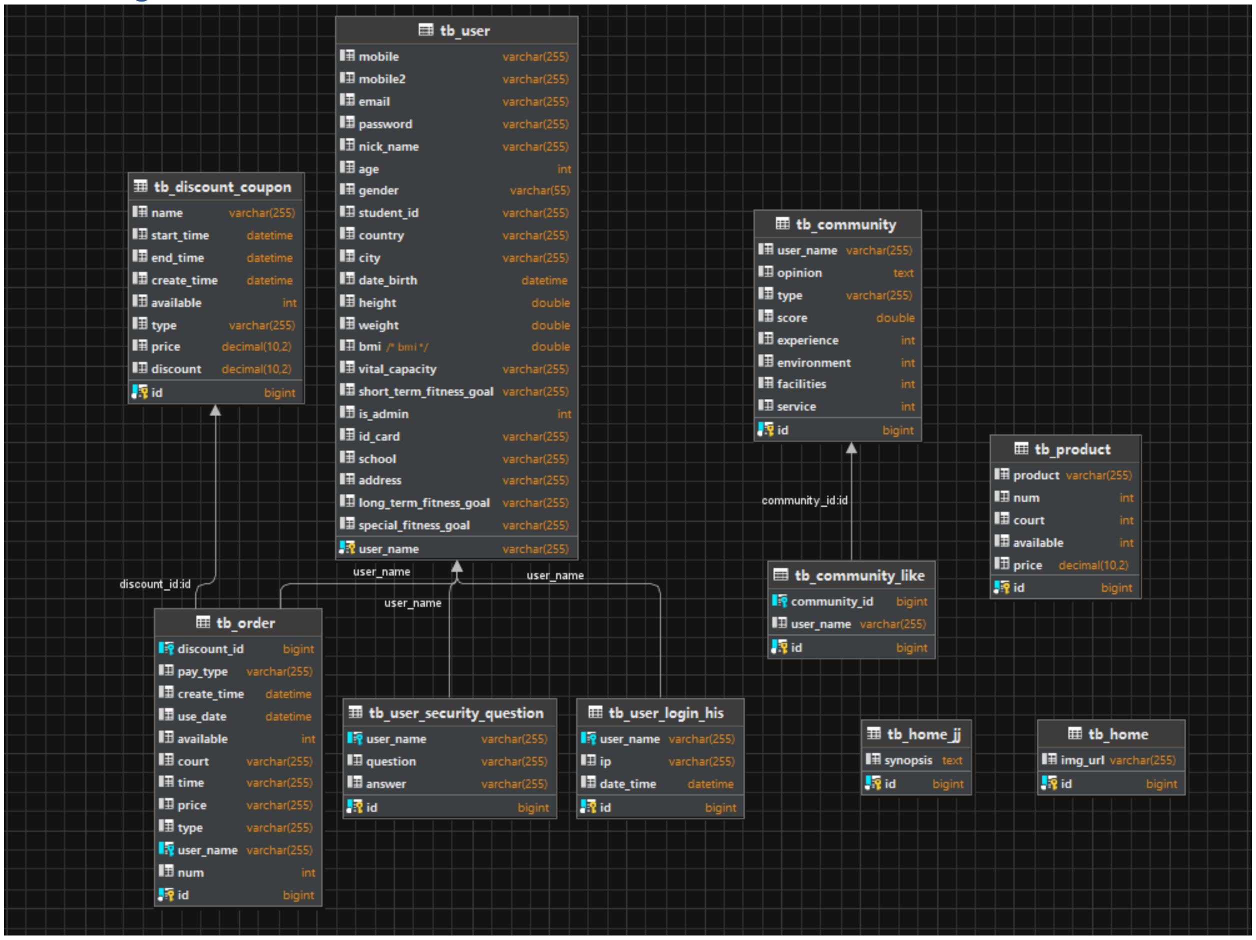
A1 Use case diagram



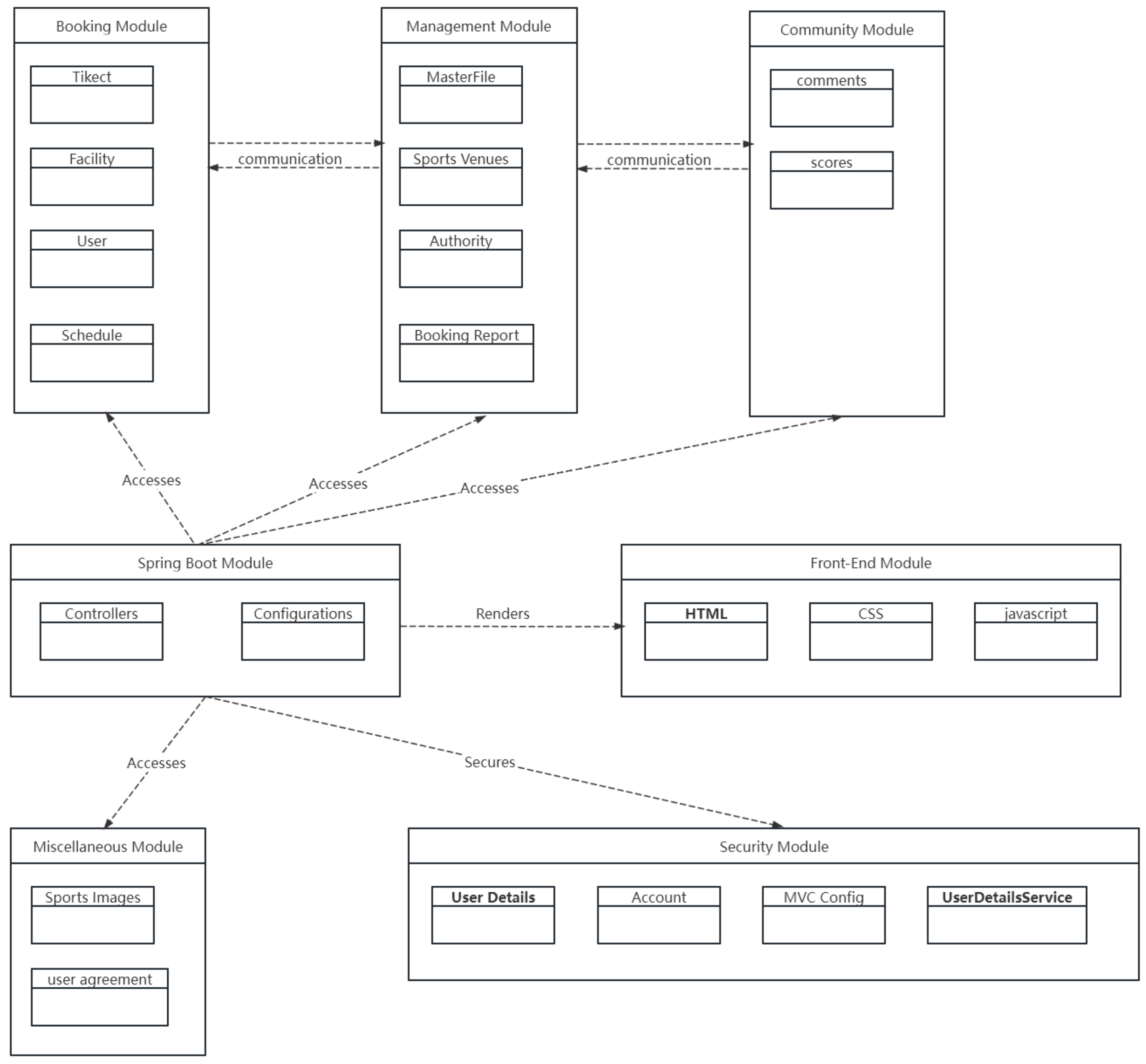
A2 MVC architecture diagram


Browser
HTTP requests
HTML pages (HTTP replies)
Controller
HTTP request processing by Spring
Application Login
User input validation
Facilitate data access
View
Presenting data to user and consist of HTML, CSS, JS
Dynamic page generation by**Axios**
Forms management by**Axios**
Interaction
Request to update
Forms
Data
Data
Model
Rrepresents the data structure and business logic of the application.
Inteact witlerform CRUD operations
Cummication through MyBatis-Plus
MySQL database

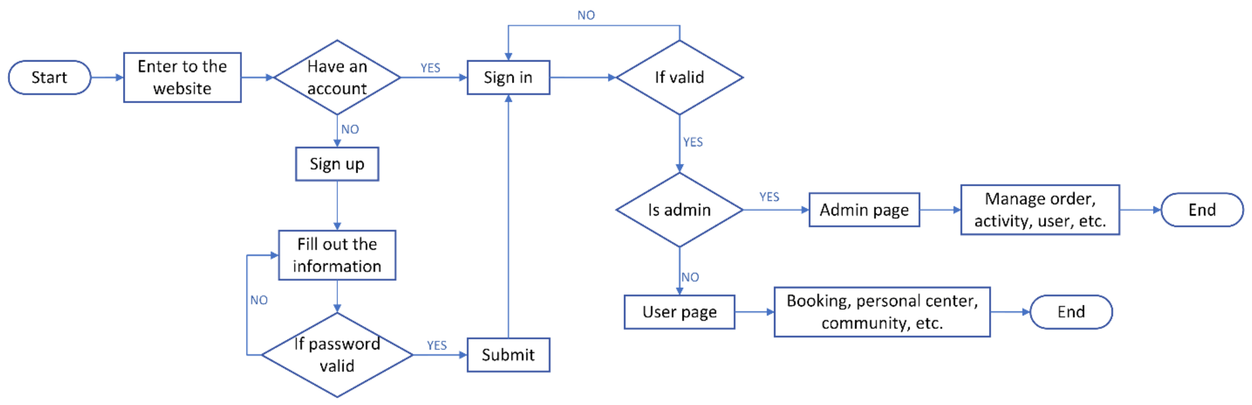
A3 ER diagram



A4 Class and module diagram



A5 High-level flow


Start
Enter to the website
Have an account
YES
NO
Sign up
Fill out the information
If password valid
NO
YES
Submit
Sign in
If valid
NO
YES
Is admin
YES
Admin page
Manage order, activity, user, etc.
End
NO
User page
Booking, personal center, community, etc.
End