**Final Project Report**

**Project: Pickleball Court Rental System**

| **……** | |
| --- | --- |
| **Group Members** | Dương Việt Hùng |
| **Supervisor/Lecturer** | … |

- Hanoi, January 2025 -

**Table of Contents**

**Definition and Acronyms 7**

**I. Project Introduction 9**

1. Overview 9

1.1 Project Information 9

1.2 Project Team 9

1.3 Stakeholders (End-users) 9

2. Product Background 9

3. Existing Systems 9

4. Business Opportunity 10

5. Software Product Vision 10

**II. Project Management Plan 11**

1. Overview 11

1.1 Scope & Estimation 11

1.2 Project Objectives 12

1.2.1. Project management 12

1.2.2. Allocated effort 12

1.3 Project Risks 13

2. Management Approach 14

2.1 Project Process 14

2.2 Quality Management 15

2.3 Training Plan 15

3. Project Deliverables 15

4. Responsibility Assignments 16

4.1.Team & Structures 16

4.2 Roles & Responsibilities 17

5. Communications Plan 17

6. Configuration Management 18

6.1 Document Management 18

6.2 Source Code Management 18

6.3 Tools & Infrastructures 18

**III. Software Requirement Specification 19**

1. Product Overview 19

2. User Requirements 19

2.1 Actors 19

2.2 Use Cases 19

2.2.1 Diagram 19

2.2.2 Descriptions 20

3. Functional Requirements 20

3.1 System Functional Overview 20

3.1.0 Business Flow 20

3.1.1 Screens Flow 21

3.1.1.1 Guest Flow 21

3.1.1.2 Admin Flow 21

3.1.1.3 User Flow 23

3.1.2 Screen Descriptions 23

3.1.3 Screen Authorization 24

3.1.4 Non-Screen Functions 25

3.1.5 In/Out of scope 25

3.2 Feature description 26

3.2.1. Login 26

3.2.2. Logout 26

3.2.3. View Profile 27

3.2.4. Edit Profile 28

3.2.5. View facility list 28

3.2.6. View facilities detail 29

3.2.7. Evaluate 30

3.2.8. Create booking 30

3.2.9. View notification list 31

3.2.10. Chat 32

3.2.11. View booking list 32

3.2.12. Booking processing 33

3.2.13. Block/Unblock Account 34

3.2.14. View Statistic 34

3.2.15. Create Facility 35

3.2.16. Update Facility 36

3.2.17. View facility updating history 37

3.2.18. Active/Inactive facility 37

3.2.19. Create category 38

3.2.20. Update category 39

3.2.21. View category updating history 40

4. Non-Functional Requirements 41

4.1 External Interfaces 41

4.1.1. User interfaces 41

4.2.2 Security interfaces 41

4.2 Quality Attributes 41

4.2.1 Usability 41

4.2.2 Reliability 41

4.2.3 Performance 41

4.2.4 Security 41

5. Requirement Appendix 42

5.1 Business Rules 42

5.2 Application Messages List 42

**IV. Software Design Description 44**

1. System Design 44

1.1 System Architecture 44

1.2 Package Diagram 44

2. Database Design 47

3. Detailed Design 49

3.1 Login 50

3.1.1 Class Diagram 50

3.1.2. Sequence Diagram 51

3.2 Manage User 52

3.2.1 View User 52

3.2.1.1 Class Diagram 52

3.2.1.2 Sequence Diagram 52

3.2.2 Update User 53

3.2.2.1 Class Diagram 53

3.2.2.2 Sequence Diagram 54

3.2.3 View user profile 55

3.2.3.1 Class Diagram 55

3.2.3.2 Sequence Diagram 55

3.2.4 Search Account 56

3.2.4.1 Class Diagram 56

3.2.4.2 Sequence Diagram 56

3.2 Manage Booking 58

3.2.1 View Booking 58

3.2.1.1 Class Diagram 58

3.3.1.2 Sequence Diagram 58

3.2.2 Find booking by user and facility 59

3.2.2.1 Class Diagram 59

3.3.2.3 Sequence Diagram 60

3.2.3 Update Booking 60

3.2.3.1 Class Diagram 60

3.2.3.2 Sequence Diagram 61

3.2.4 Create Booking 61

3.2.4.1 Class Diagram 61

3.2.4.2 Sequence Diagram 62

3.2.5 Status Booking 63

3.2.5.1 Class Diagram 63

3.2.5.2 Sequence Diagram 64

3.2.6 Dashboard by year 64

3.2.6.1 Class Diagram 64

3.2.7 Dashboard by week 66

3.2.7.1 Class Diagram 66

3.3 Manage Facility 67

3.3.1 View Facility 67

3.3.1.1 Class Diagram 67

3.3.1.2 Sequence Diagram 68

3.3.2 Create Facility 68

3.3.2.1 Class Diagram 68

3.3.2.2 Sequence Diagram 69

3.3.3 Update Facility 70

3.3.3.1 Class Diagram 70

3.3.3.2 Sequence Diagram 71

3.3.4 Change Status Facility 72

3.3.4.1 Class Diagram 72

3.3.4.2 Sequence Diagram 73

3.3.5 View Detail Facility 74

3.3.5.1 Class Diagram 74

3.3.5.2 Sequence Diagram 74

3.4 Manage Category 75

3.4.1 View Category 75

3.4.1.1 Class Diagram 75

3.4.1.2 Sequence Diagram 75

3.4.2 Create Category 76

3.4.2.1 Class Diagram 76

3.4.2.2 Sequence Diagram 76

3.4.3 Update Category 77

3.4.3.1 Class Diagram 77

3.4.3.2 Sequence Diagram 77

3.5 Manage Comments 78

3.5.1 View Comment 78

3.5.1.1 Class Diagram 78

3.5.1.2 Sequence Diagram 78

3.5.2 Create Comment 79

3.5.2.1 Class Diagram 79

3.5.2.2 Sequence Diagram 79

3.5.3 Check Comment Permission 79

3.5.3.1 Class Diagram 79

3.5.3.2 Sequence Diagram 80

3.6 Manage Notification 81

3.6.1 View notification 81

3.6.1.1 Class Diagram 81

3.6.1.2 Sequence Diagram 81

3.6.2 Update Notification 82

3.6.2.1 Class Diagram 82

3.6.2.2 Sequence Diagram 82

**V. Software Testing Documentation 83**

1. Scope of Testing 83

1.1. Testing Stages 83

1.2. Range of Testing 84

2. Test Strategy 84

2.1 Testing Types 84

2.2 Test Levels 84

2.3 Supporting Tools 85

3. Test Plan 86

3.1 Human Resources 86

3.2 Test Environment 86

3.3 Test Milestones 86

**VI. Release Package & User Guides 88**

1. Deliverable Package 88

2. Installation Guides 88

2.1 System Requirements 88

2.2 Hosting requirement 88

3. User Manual 88

3.1 Overview 88

3.2 Role User 88

3.3 Admin Role 95

# **Definition and Acronyms**

| **Acronym** | **Definition** |
| --- | --- |
| BA | Business Analysis |
| BR | Business Rule |
| ERD | Entity Relationship Diagram |
| UI | User Interface |
| PM | Project Manager |
| SDD | Software Design Description |
| SPMP | Software Project Management Plan |
| SRS | Software Requirement Specification |
| UC | Use Case |
| FMS | Facilities Management System |
| ST | System Testing |
| UT | Unit Test |
| IT | Integration Testing |
| AT | Acceptance Testing |

# **I. Project Introduction**

## **1. Overview**

### **1.1 Project Information**

● Project name: Pickleball Court Rental System

● Project code: PCRS

● Group name: SE1763\_NJ\_G1

● Software type: Web-based Application

● Timeline: 07/01/2025 - 30/03/2025

### **1.2 Project Team**

| Full **Name** | **Role** | **Email** |
| --- | --- | --- |
| Nguyễn Thanh Tú | Lecturer | TuNT57@fe.edu.vn |
| Dương Việt Hùng | Leader | hungdvhe173437@fpt.edu.vn |
| Lê Hoàng Lâm | Member | lamlhhe173007@fpt.edu.vn |

***Table I.1****: Team member*

### **1.3 Stakeholders (End-users)**

| ***Full Name*** | ***Email*** | **Title** |
| --- | --- | --- |
| …. |  | Technical support, Product Owner, User |

***Table 1.2:*** *Stakeholders*

## **2. Product Background**

The Pickleball Court Rental System was developed to streamline court rental operation. Managing daily bookings with traditional methods has become inefficient and error-prone as demand grows. This system optimizes court availability, customer bookings, and payment tracking, ensuring efficient operations and improved customer satisfaction. Developed in nine weeks using SCRUM, the system meets the needs of modern pickleball court management.

## **3. Commercial Possibility**

A digital platform called the Pickleball Court Rental System was created to update and simplify the court rental procedure at PicklePlay. With plans for future scaling, the technology will be initially deployed at the primary court location.

Features like online booking, real-time court availability tracking, and integrated payment options are all offered by the system. It guarantees effective rental management, reduces operational bottlenecks, and enhances the client experience by automating processes. Online court booking is convenient for customers, cutting down on wait times and improving satisfaction while giving the management team useful information to improve operations.

## 

## **4. Software Product Vision**

The Pickleball Court Rental System was developed to address the operational challenges faced. It is a comprehensive solution designed to simplify and enhance the court rental process while improving resource utilization and customer service.

With its user-friendly interface and robust features, the system empowers administrators to manage court availability, bookings, payments, and maintenance schedules efficiently. For the management team, it eliminates the need for manual paperwork, enabling real-time monitoring of bookings and customer accounts.

For customers, the system offers a seamless experience by allowing them to browse available courts, make bookings, and track rental details online. This shift from manual to digital operations ensures that PicklePlay remains competitive in the modern court rental market while delivering exceptional service to its customers.

# **II. Project Management Plan**

## **1. Overview**

### **1.1 Scope & Estimation**

Table II.1 The WBS (Work Breakdown Structure) items are categorized into three levels of complexity (Simple, Medium, Complex) and estimate the total effort to complete each item in man-day (1 man-day = 8 working hours)

| **WBS ID** | **WBS Item** | **Complexity** | **Est. Effort (man-days)** |
| --- | --- | --- | --- |
| **1. Project Management** | | | |
| 1.1 | Project Initiation | Normal | 1 |
| 1.2 | Project Planning | Complex | 2 |
| Total : | | | 3 |
| **2. Project Designing** | | | |
| 2.1 | Gathering and collecting requirement with Team member and Lecture | Normal | 4 |
| 2.2 | Create Project Introduction document | Simple | 1.5 |
| 2.3 | Define project scope and objectives | Complex | 1 |
| 2.4 | Learn about similar Facilities Management System | Normal | 2 |
| 2.5 | Create Project Timeline and Schedule | Normal | 5 |
| 2.6 | Designing Screenflow (Guest, User, Admin) | Normal | 5 |
| 2.7 | Designing basis of layout (Concept, Feature, Requirement) | Complex | 5 |
| 2.8 | Identify and mitigate project risks | Normal | 3 |
| 2.9 | Creating Test Plan | Normal | 2 |
| Total : | | | 28.5 |
| **3. Project Development** | | | |
| 3.1 | Prototyping Project | Complex | 5 |
| 3.2 | Develop product prototype | Complex | 7 |
| 3.3 | Conduct product testing and validation | Complex | 5 |
| 3.4 | Develop front-end design | Complex | 10 |
| 3.5 | Develop back-end functionality | Complex | 35 |
| 3.6 | Implement security measures | Complex | 5 |
| 3.7 | Create Test (IT, UT, ST) along coding | Normal | 10 |
| Total : | | | 77 |
| **4. Quality Assurance & Testing** | | | |
| 4.1 | Conduct functional testing | Normal | 5 |
| 4.2 | Conduct usability testing | Normal | 5 |
| 4.3 | Testing and Fixing bugs | Normal | 5 |
| Total : | | | 15 |
| **Total Estimated Effort (man-days)** | | | **123.5** |

***Table II.1****: Scope & Estimation*

### **1.2 Project Objectives**

#### ***1.2.1. Project management***

Table II.2: describes the project management .

| **No** | **Objectives** |
| --- | --- |
| 1 | Project will release before 30/03/2025 |
| 2 | Deliver screen design after 2 first sprint |
| 3 | Sprint goals are fully completed |
| 4 | Release product reduce 80% manual process |

***Table II.2****: Project management*

#### ***1.2.2. Allocated effort***

Table II.3: describes the allocated effort

| **#** | **Member** | **Weekdays** | **Weekends** |
| --- | --- | --- | --- |
| 1 | Dương Việt Hùng | 2 hours | 4 hours |
| 2 | Lê Hoàng Lâm | 2 hours | 4 hours |

***Table II.3****: Allocated effort*

### **1.3 Project Risks**

Table II.4 Describe the possible risks of the project

**People Risk** is the Human Factor of the Risks. Risk like a teammate is in sickness, accident, late arrival, Any communication hardness between remember or even stakeholder with team is considered to be a People Risk.

**Process Risk** is the Risk that emerges when working on Project Management. Project Risk like budgeting cost more than its planned, inadequate quality assurance and other related to Project Management.

**System Risk**  associated with technical issues, such as the failure of hardware, software, or network infrastructure that could impact project performance, data integrity, or security.

**External Event Risk** is External events such as natural disasters, pandemics, economic downturns, or geopolitics upheavals can significantly impact project success.

**Legal and Compliance Risk** often associated with legal or regulatory requirements that could impact project outcomes. Legal and compliance risks can arise from issues such as intellectual property infringement, breach of contract, regulatory violations, or environmental risks.

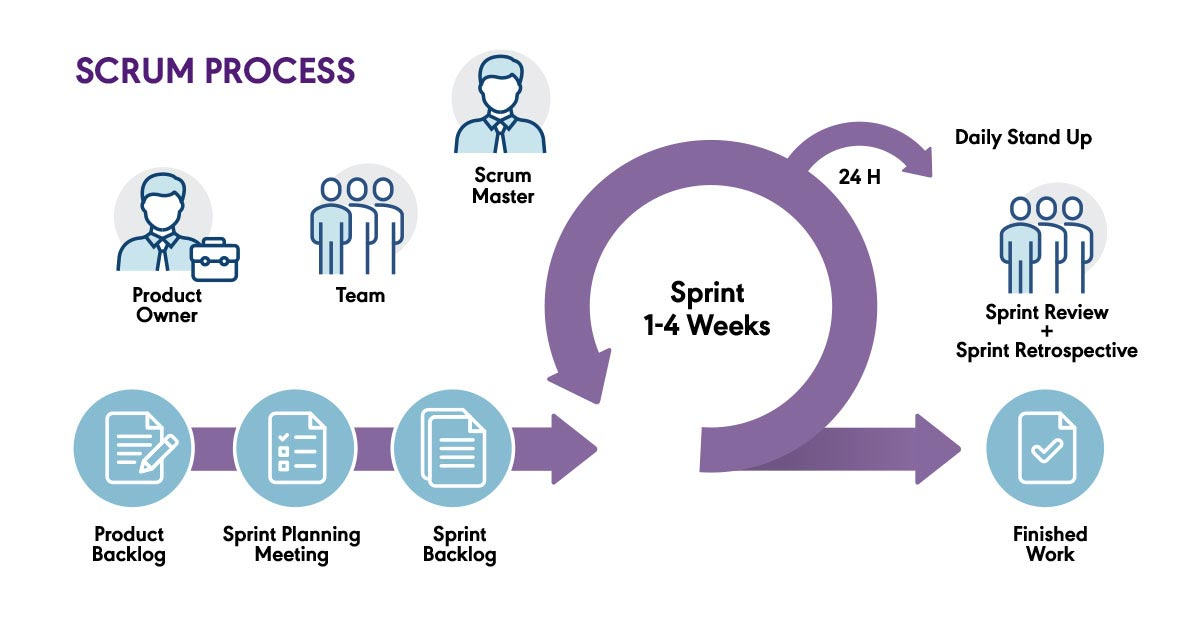
| # | Risk Description | Impact | Possibility | Response Plans |
| --- | --- | --- | --- | --- |
| 1 | Requirement  changes. | Medium | Medium | All members discuss carefully the project requirements before starting  implementation. |
| 2 | Members have argued,conflicted with others, leads to stressful working environments. | Medium | High | Define clear tasks for each member and agree on ideas before starting work. |
| 3 | Illness or absence of team members so that they cannot complete tasks under deadline | Low | Medium | Members have to notify the team about illness or absence periods and the plan of how to keep up with the work process. |
| 4 | Members lack the knowledge and skills to complete a particular task | High | Medium | Training all members before starting the project. |
| 5 | The library used in the project is no longer supported | Low | Low | Choose a reputable library with active maintenance on GitHub. |

***Table II.4****: Project Risks*

## **2. Management Approach**

### **2.1 Project Process**

Figure II.5 Describe the project process of the project

******

***Figure II.5****: Project Process*

After researching the software development model carefully, the project will use the Agile - Scrum Software Process Model. In an Iterative & Incremental model, Scrum software process model is an agile approach to project management that emphasizes iterative development, flexibility, and collaboration. Scrum is a widely adopted framework because it offers numerous benefits over traditional software development methodologies.

### 

### **2.2 Quality Management**

***2.2.1 Defect Prevention:***

- If any defect is found, the related person must be notified immediately at that time.

- Defects must be carefully evaluated such as "How bad is the defect and can it damage the system?", "How long is the time to fix that defect?".

- The deadline for fixing the defect must be specified clearly.

- There is always a plan to prepare for what could happen at any time.

***2.2.2 Reviewing:***

- The curator must be honest and not biased towards any of the project members. If there is an error, the person must immediately notify the person responsible for the defect.

- Defects should be recorded on the Bug Tracking software with details such as priority.

- The person responsible for defects found must-have solutions to fix the defect as quickly as possible.

### **2.3 Training Plan**

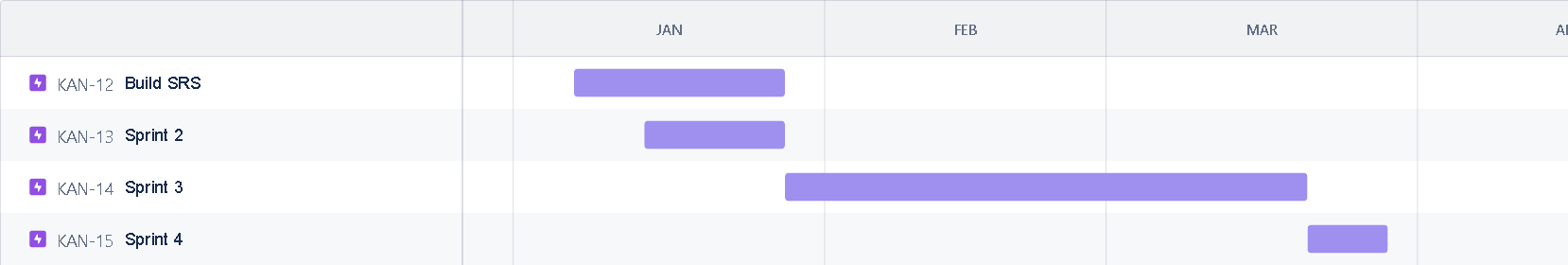
Table II.6 Describe the training plan of the project

| **Training Area** | **Participants** | **When, Duration** | **Waiver Criteria** |
| --- | --- | --- | --- |
| NodeJS | All members | 1 days | Mandatory |
| Meeting Note | All members | 15 minutes | Mandatory |
| Github | All members | 30 minutes | Mandatory |
| MongoDB | All members | 30 minutes | Mandatory |
| NextJS | All members | 1 days | Mandatory |
| Postman | All members | 15 minutes | Mandatory |

***Figure II.6****: Training plan*

## **3. Project Deliverables**

Figure II.7. Describe the project delivery of the project



***Figure II.7****: Project Deliver*

Project used Sprint Plan to manage task and estimate for new sprint every week

For more details, click to this [Jira](https://d10nysus.atlassian.net/jira/software/projects/KAN/boards/1?atlOrigin=eyJwIjoiaiIsImkiOiIxMjlhNGZjMWM5YmQ0NzRkODQ3NTIxYmRiYzg4ZjNmMiJ9&cloudId=c375c48b-9f55-43be-856e-623a13c3eeb4&fbclid=IwY2xjawH1uzlleHRuA2FlbQIxMAABHek3GcY2qjXGCd1AvQVtNvCSEhIouREtoHtLzrFLv9pKrx2L6Kc7m4loAA_aem_gvL8pvf1ovv-cqaSkf5yFw)

## **4. Responsibility Assignments**

### **4.1.Team & Structures**

Figure II.8 Describe the team & structures of project

### 

***Figure II.8****: Team & Structures*

### **4.2 Roles & Responsibilities**

Table II.9 Describe the roles & responsibility of the project

| **Role** | **Responsibility** |
| --- | --- |
| Project Manager | - Planning, developing schedules, coordinating communication, responsible for keeping the team’s focus on the main goal.  - Research and implement UX design for the front-end. |
| BA | - Analyze requirements.  - Define scope and create an SRS document.  - Design entity relationship diagram.  - Define business process flow and object state.  - Capture and specifically describe the use case. |
| Technical Leader | - Define high-level architecture based on SRS.  - Implement configuration and web server. |
| Front-End / Back-End  Developer | - Involve in coding the product and reviewing the code of other developers. |
| Tester | - Create a template testing document.  - Define test strategy, create test case  - Implement test cases and log defects. |

***Table II.9****: Roles & Responsibilities*

## **5. Communications Plan**

Figure II.10 Describe the communication plan of the project

| **Communication Item** | **Who/ Target** | **Purpose** | **When, Frequency** | **Type, Tool, Method(s)** |
| --- | --- | --- | --- | --- |
| Team meeting | All team members | Review plan, schedule, code, design, test and document | Once every 3 days | Online - Google meet |
| Weekly meeting with mentor/lecturer | All team members | Gather requirements, review project,  log issue, define next week plan | Tuesday and Friday every week | Offline |
| On-going Assessment Meeting | All team members | Evaluate project after every iteration | Once every 3 weeks | Offline |
| Unscheduled meeting | All team members | Discuss and solve important problems | When someone has important problems | Online - Google meet |

***Table II.10****: Communications Plan*

## **6. Configuration Management**

### **6.1 Document Management**

Document tools: Microsoft Office 2016, Microsoft Excel 2016, Google Document, Google Sheet

File management: Google Drive

### **6.2 Source Code Management**

Source code is managed by Git onhttps://github.com/hungduongg1903/WDP301\_Group1

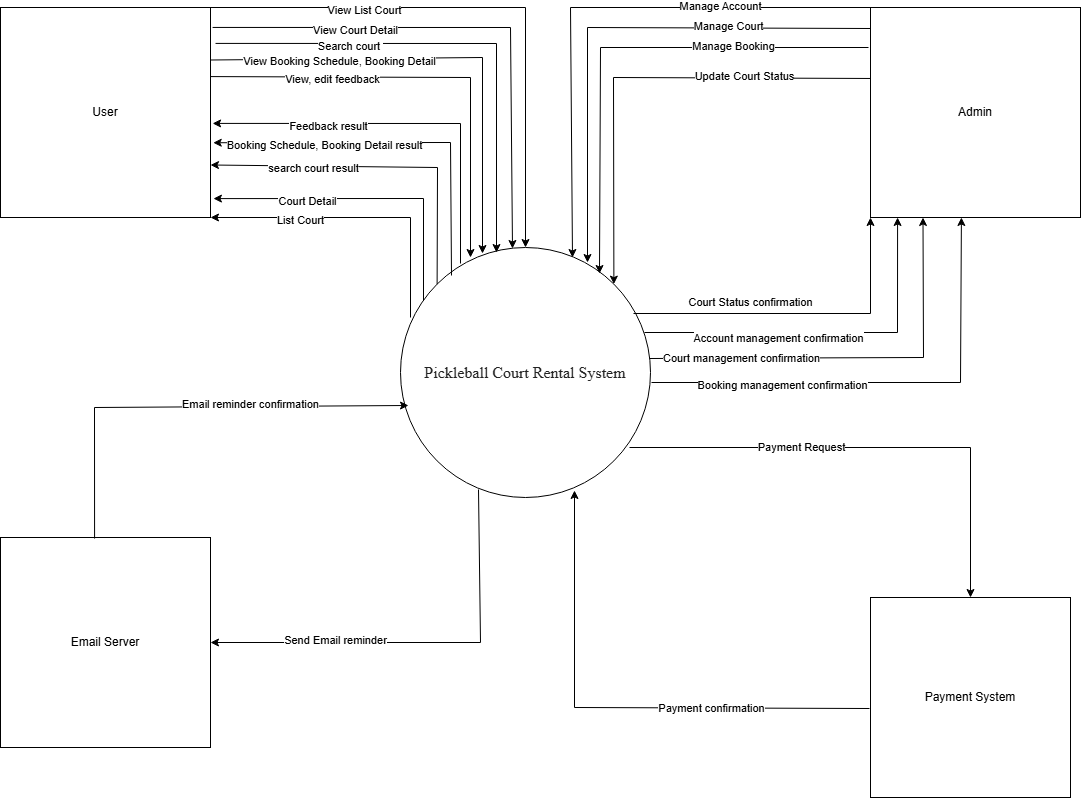
### **6.3 Tools & Infrastructures**

Table II.11 Describe the tool & infrastructures of the project

| **Category** | **Tools / Infrastructure** |
| --- | --- |
| **Technology** | NodeJS |
| **Database** | MongoDB |
| **IDEs/Editors** | Visual Studio Code |
| **Diagramming** | DrawIO |
| **Documentation** | Microsoft Office, Google Docs/Sheets/Slides |
| **Version Control** | GitHub(Source Codes), Google Drive (Documents) |
| **Project management** | Jira (Schedule), GitHub (Tasks, Defects) |

***Table II.11****: Tools & Infrastructures*

# **III. Software Requirement Specification**



## **1. Product Overview**

Pickleball Court Rental System is an online platform that allows users to search, reserve, and pay for Pickleball court rentals. The system focuses on two main games:

User (Renter): Register an account, find courts, book, pay, and view court booking history.

Administrator (Administrator): Manage court information (add/delete/edit), process court booking requests, manage users, and report.

Force command:

The system relies on the administrator to accurately update court information (quiet hours, including prices).

Integrate with third-party payment systems (e.g. VNPay, Momo) to process transactions.

Assume the user has a stable internet connection to book and pay.

## **2. User Requirements**

### **2.1 Actors**

Table III.1 Describe the project delivery of the project

| **#** | **Actor** | **Description** |
| --- | --- | --- |
| 1 | Admin | Manage all court and user information |
| 2 | User | Search for court, payment, view court history |
| 3 | Guest | Guest user |

### 

### 

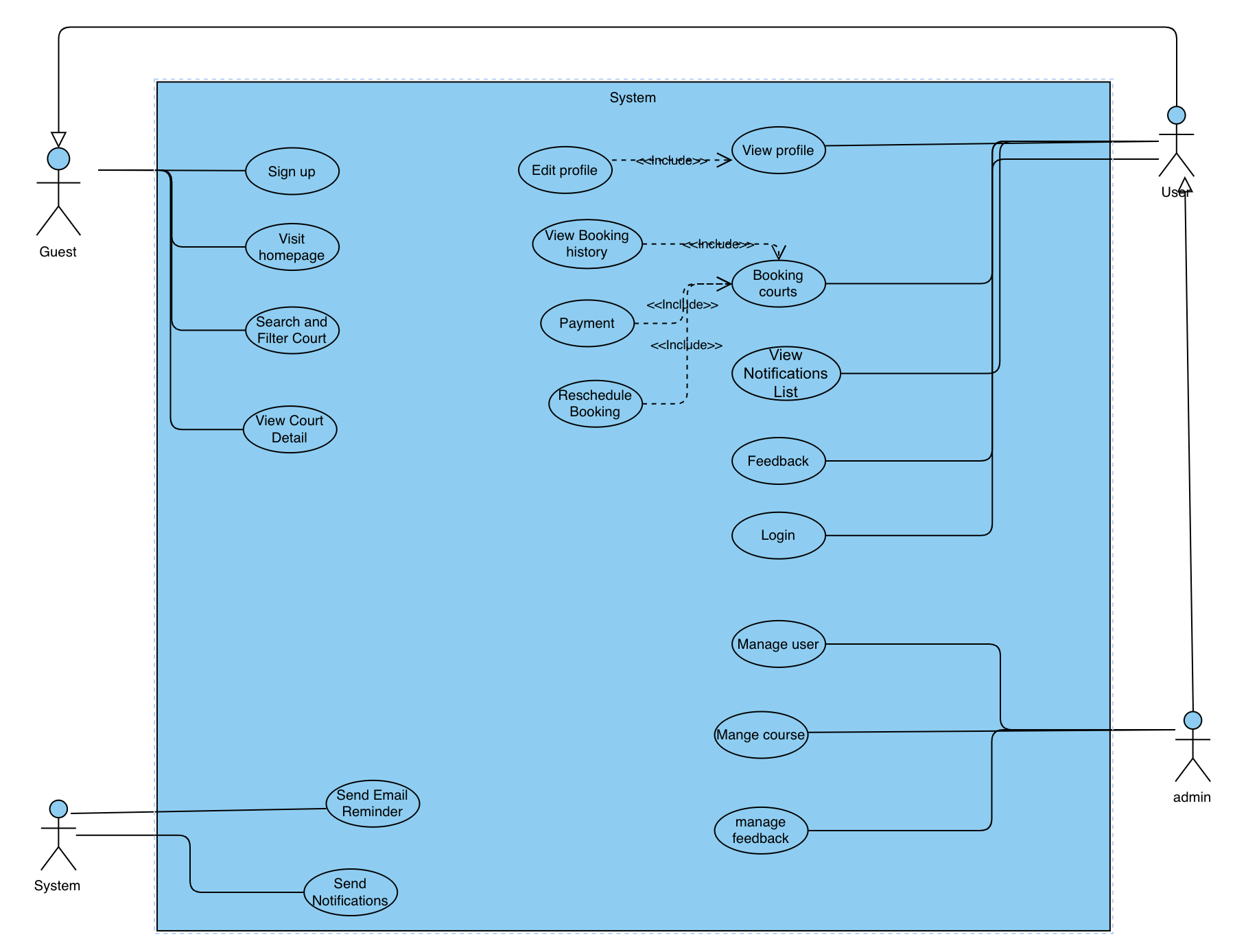
### 

***Table III.1****: Actors*

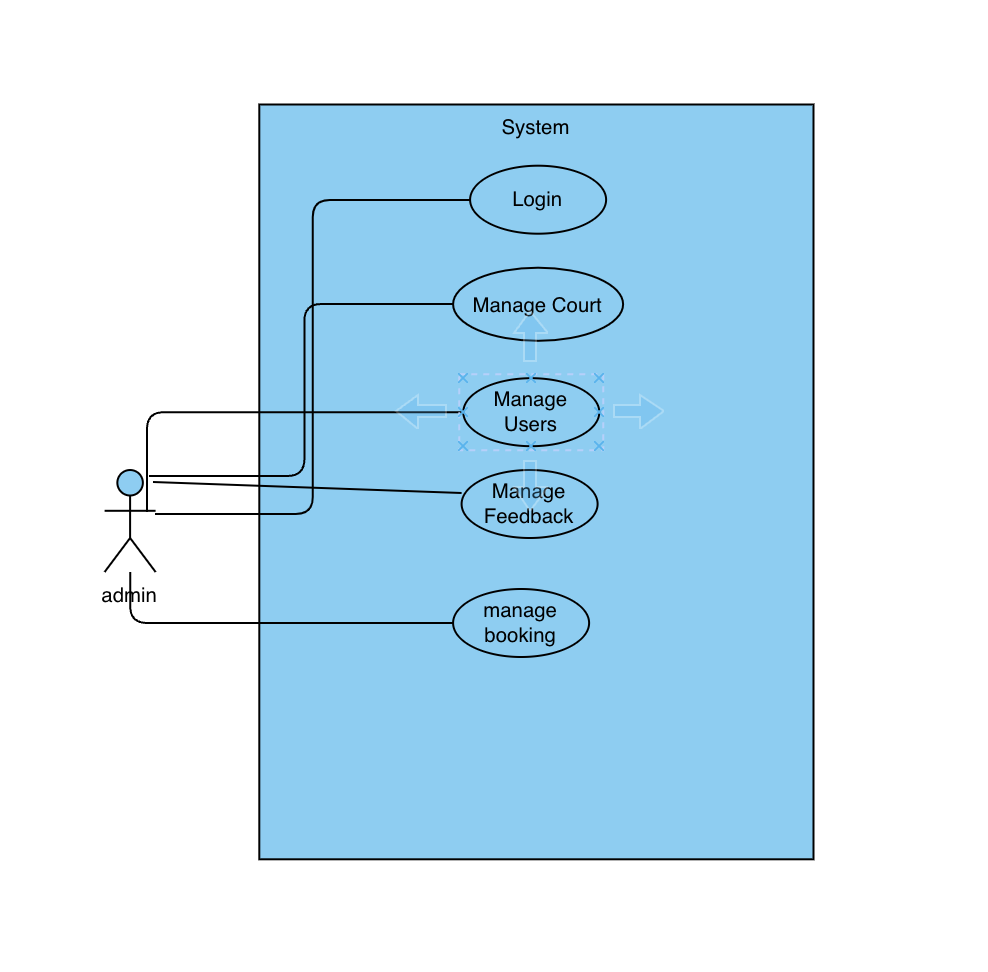
### **2.2 Use Cases**

### **2.2.1 Diagram**

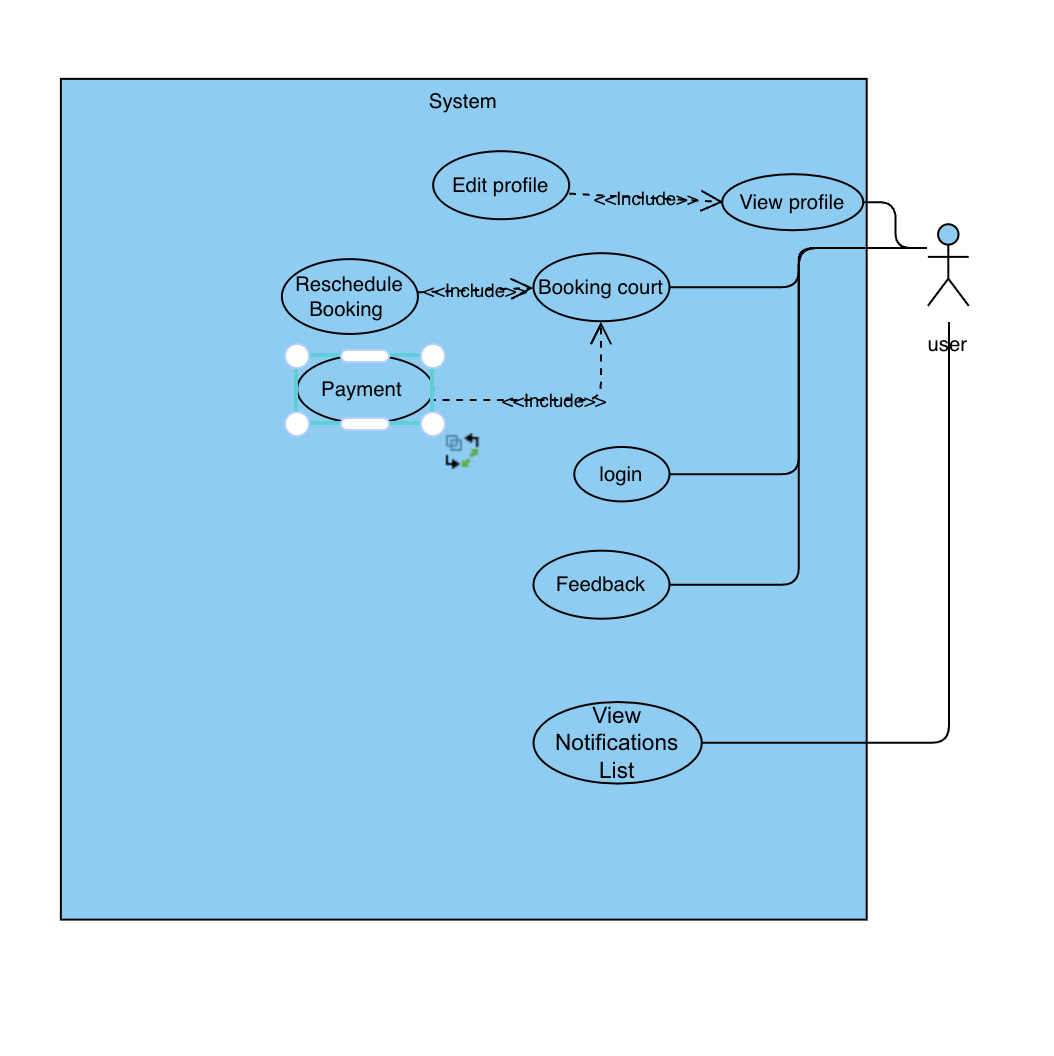
Figure III.2 Describe the project delivery of the project

****

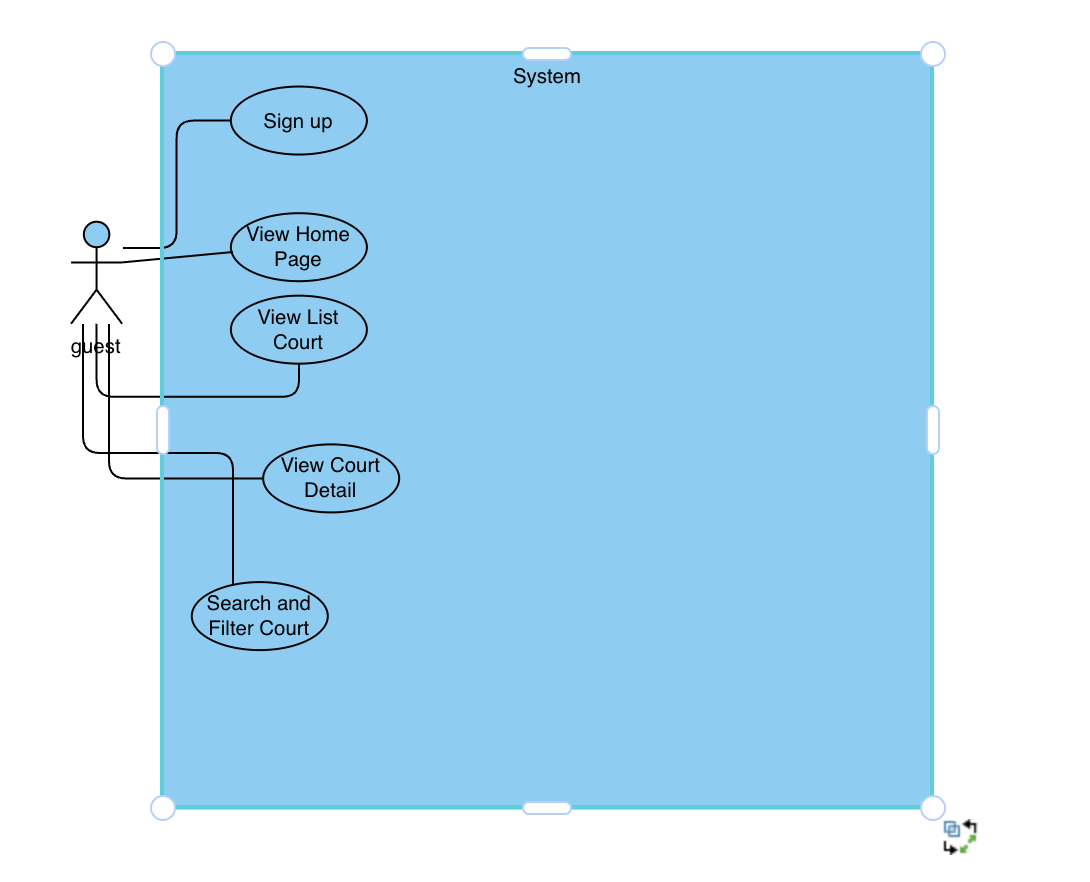
***Figure III.2****: Usecase diagram*

**

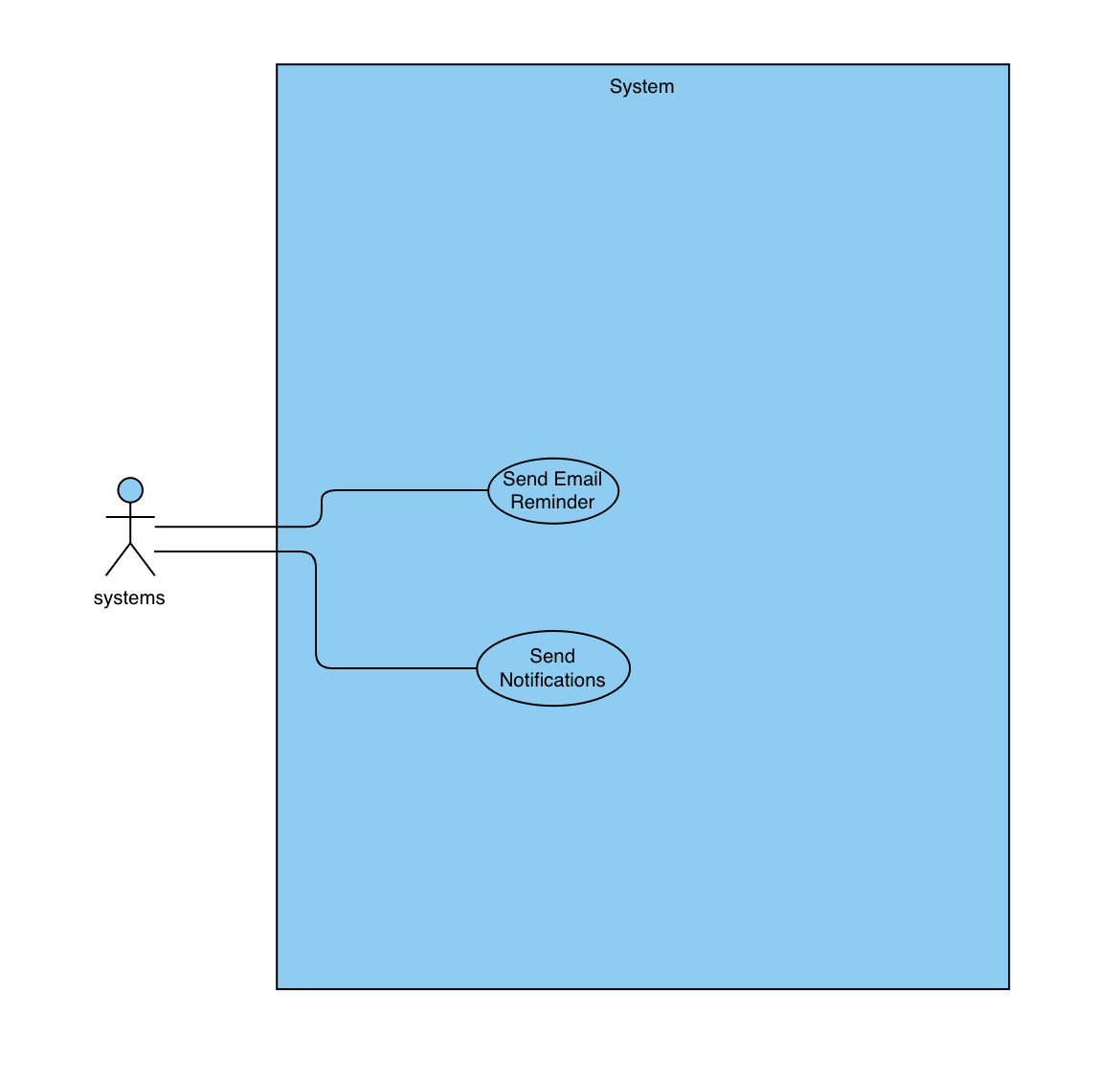
***Figure III.3****: Admin Use Case diagram*

**

***Figure III.4****: User Use Case diagram*

**

***Figure III.5****: Guest Use Case diagram*

**

***Figure III.6****: System Use Case diagram*

### **Nma 2.2.2 Descriptions**

[Figure](https://docs.google.com/document/d/1DDuli0VTmQbUii5wbItkVfbrL2yOgHRSqDVQJc8W8Aw/edit#heading=h.vfr36oj0vfd6) III.6 Describe the functions present in the system and describe the roles used for each function.

| **ID** | **Use Case** | **Actors** | **Use Case Description** |
| --- | --- | --- | --- |
| 01 | Login | User, Admin | Enables users to log in to their account |
| 02 | Logout | User, Admin | Logs users out of the system |
| 03 | Sign Up | User, Admin,Guest | Allows users to create an account |
| 04 | View profile | User, Admin | Enables users to view personal details |
| 05 | Edit Profile | User, Admin | Allow users change their information |
| 06 | View Home Page | User, Admin, Guest | Displays key features and introductory information |
| 07 | View List Court | User, Admin, Guest | Enables users to view list court |
| 08 | View Court Detail | User, Admin | Shows details of a specific booking |
| 09 | Book Court | User | Enables users to book a court |
| 10 | Forgot Password | User, Admin | Provides password recovery options |
| 11 | Change Password | User, Admin | Allows users to update their password |
| 12 | View Booking Detail | User | Shows details of a specific booking |
| 13 | View Booking History | User, Admin | Displays previous bookings |
| 14 | Manage Court | Admin | Allows owners to add/edit/delete court listings |
| 15 | Search and Filter Court | User, Admin, Guest | Admin can view history detail of category updating |
| 16 | Manage Users | Admin | Allows admin to manage user accounts |
| 17 | Set Availability Status | Admin | Allow manager change court status |
| 18 | Reschedule Booking | User, Admin | Enables users to update booking schedule |
| 19 | Cancel Booking | User, Admin | Allows users to cancel a booking |
| 20 | Payment integration | User | Handles court booking payments |
| 21 | Add feedback | User | Allows users to add a feedback for court |
| 22 | View Booking Schedule | User | Show inforation about booking schedule |
| 23 | View Court Feedback | User, Admin | Allows users to view court’s feedback |
| 24 | Manage Bill | Admin | Allows admin to view and confirm bill listings |
| 25 | Long Term Booking | User | Allows users to book long term court |
| 26 | Manage Borrower | User, Admin | Allows users and admins to manage borrower schedule |

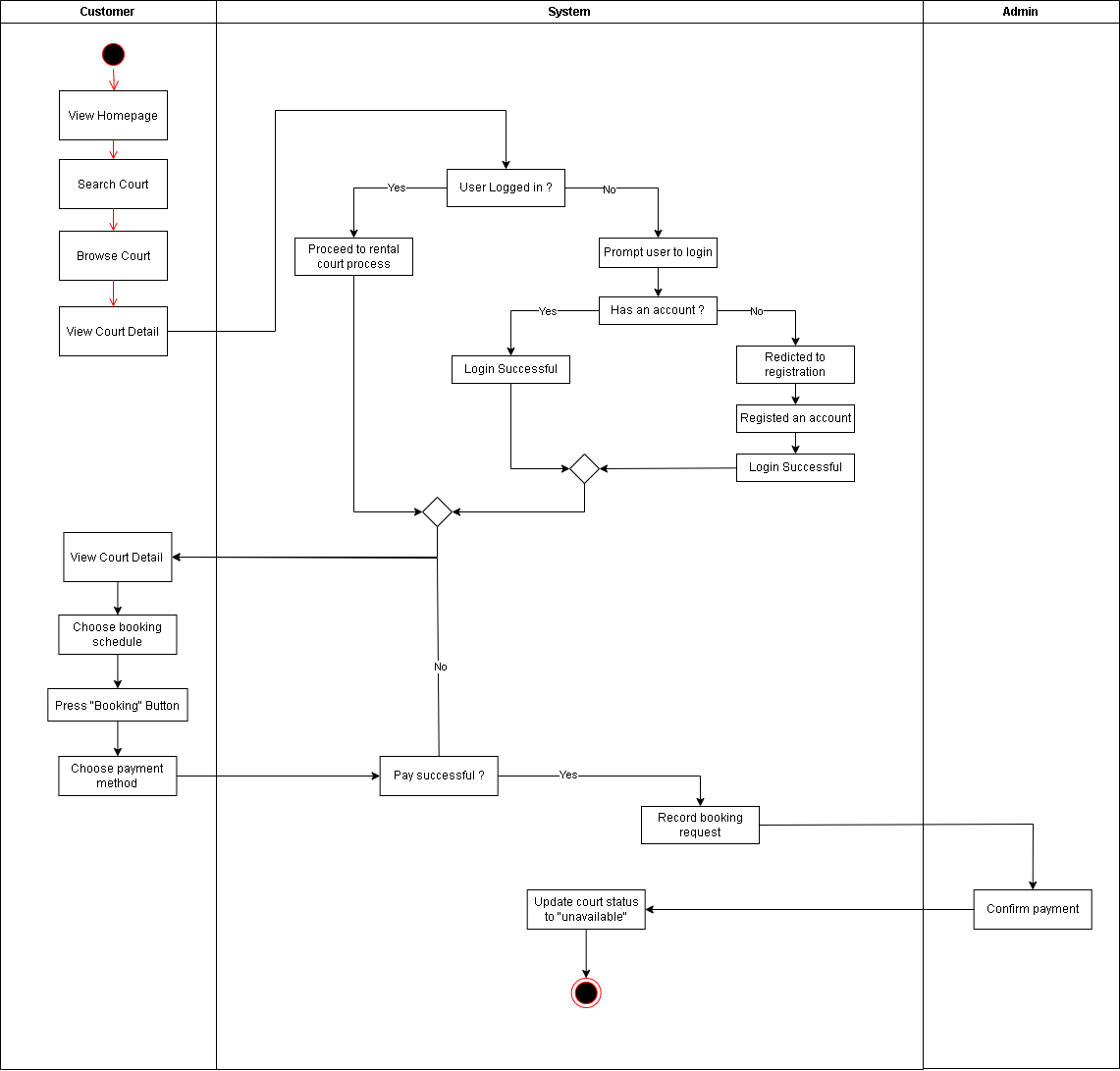
***Table III.6****: Description*

## **3. Functional Requirements**

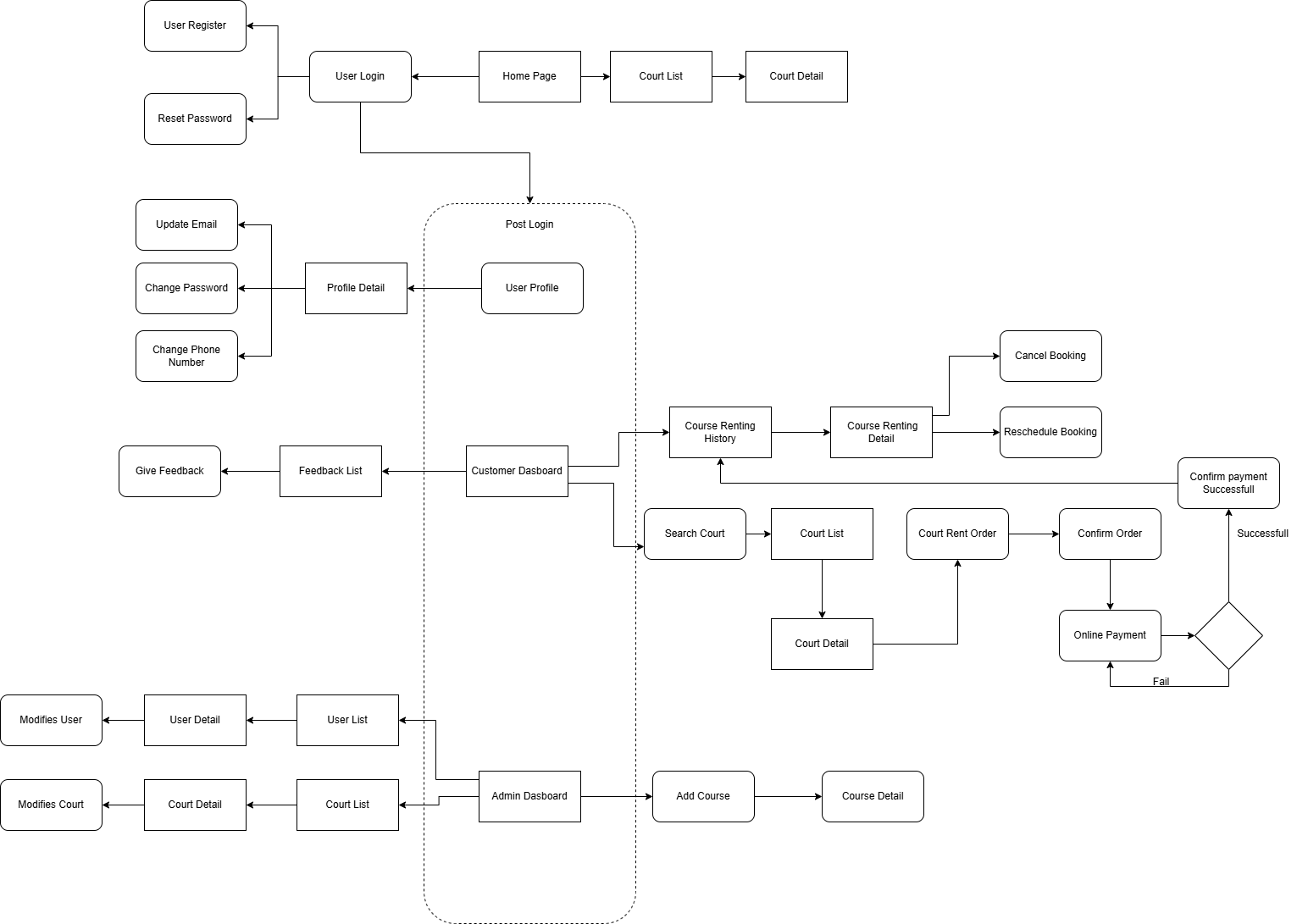
### **3.1 System Functional Overview**

#### ***3.1.1 Business Flow***

* Court Rental Stream:



#### ***3.1.2 Screens Flow***



#### ***3.1.3 Screen Descriptions***

Table III.11: Shows the screen descriptions of the system

| **#** | **Feature** | **Screen** | **Description** |
| --- | --- | --- | --- |
| 1 | Signup | User signup | Allows users to create an account |
| 2 | Login | User Login | Enables users to log in to their account |
| 3 | Logout | User logout | Enables users to log out of their account |
| 4 | View Profile | User Profile | Enables users to view personal details |
| 5 | Edit Profile | User profile | Enables users to update personal details |
| 6 | View Home Page | Home Page | Displays key features and introductory info |
| 7 | View List Court | List Court | Enables users to view list court |
| 8 | View Court Detail | Court Detail | Shows details of a specific booking |
| 9 | Book Court | Book Court | Enables users to book a court |
| 10 | Forgot Password | Forgot Password | Provides password recovery options |
| 11 | Change Password | Change Password | Allows users to update their password |
| 12 | View Booking Detail | Booking Detail | Shows details of a specific booking |
| 13 | View Booking History | Booking History | Displays previous bookings |
| 14 | Manage Court | Manage Court | Allows owners to add/edit/delete court listings |
| 15 | Search and Filter Court | Search and Filter Court | Allows user to search and filter for court |
| 16 | Manage Users | Manage Users | Allows admin to manage user accounts |
| 17 | Set Availability Status | Set Availability Status | Allow manager change court status |
| 18 | Reschedule Booking | Reschedule Booking | Enables users to update booking schedule |
| 19 | Cancel Booking | Cancel Booking | Allows users to cancel a booking |
| 20 | Payment Integration | Payment Integration | Handles court booking payments |
| 21 | Add Feedback | Add Feedback | Allows user to add feedback |
| 22 | View Booking Schedule | Schedule booking | Show inforation about booking schedule |
| 23 | View Court Feedback | Court Feedback | Allows users to view feedback of Court |
| 24 | Manage Bill | Manage Bill | Allows admin to view and confirm bill listings |
| 25 | Long Term Booking | Long Term Booking | Allows users to book court for long term |
| 26 | Booking many hours | Booking many hours | Allows users to book court for long time |

***Table.III.11****: Screen description*

#### ***3.1.4 Screen Authorization***

Table.III.12: Shows the screen authorizations of the system

| **#** | **Screen** | **Guest** | **User** | **Admin** |
| --- | --- | --- | --- | --- |
| 1 | Home page | x | x | x |
| 2 | User Login |  | x | x |
| 3 | User Register | x |  |  |
| 4 | Reset Password |  | x | x |
| 5 | User Profile |  | x | x |
| 6 | Court List | x | x | x |
| 7 | Court Detail |  | x | x |
| 8 | User List |  |  | x |
| 9 | Booking History |  | x |  |
| 10 | Book Court |  | x | x |
| 11 | Reschedule Booking |  | x | x |
| 12 | Feedback List |  | x | x |
| 13 | Customer Dashboard |  | x |  |
| 14 | Admin Dashboard |  |  | x |
| 15 | Booking Schedule |  | x |  |

***Table III.12****: Screen authorization*

#### ***3.1.5 Non-Screen Functions***

Table III.13: Shows the screen authorizations of the system

| **#** | **Feature** | **System Function** | **Description** |
| --- | --- | --- | --- |
| 1 | User management | Logout | User logout to system |
| 2 | Management | Search | User search for court |

***Table III.13****: Non-Screen functions*

#### 

#### ***3.1.6 In/Out of scope***

Table III.14: Shows the goals set to upgrade the system after completing the functions outlined above.

| **#** | **In of Scope** | **Out of Scope** |
| --- | --- | --- |
| 1 | Manage Booking Schedule | HR Management |
| 2 | Manage Users | Financial Management |
| 3 | Manage Bookings Court | Mobile Application Development |
| 4 |  | Integration with Other Systems,... |

***Table III.14****: In/Out of scope*

### **3.2 Feature description**

#### ***3.2.1. LogIn***

Table III.15: Shows the Login feature description

| **USE CASE-01** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Use-case No.** | UC01 | **Use-case Version** | | | 1.0 |
| **Use-case Name** | Login | | | | |
| **Author** | ThanhLM | | | | |
| **Date** | 07/01/2025 | **Priority** |  | Medium | |
| **Actor** | User, Admin | | | | |
| **Summary** | User login to use actor features on the system | | | | |
| **Goal** | Allow guest become user/admin of system | | | | |
| **Triggers** | Guest click on button “Đăng nhập“ on navigation bar | | | | |
| **Preconditions** | User not login and have FPT email | | | | |
| **Post Conditions** | User join system with their role and use all feature with permission | | | | |
| **Main Success Scenario** | | | | | |
| **Step** | **Actor Event** | |  | **System response** | |
| 1 | User click button “Đăng nhập” in navigation bar | |  | Redirect to google authentication service | |
| 2 | Login to google with FPT email | |  | Allow user join on the system and log the message | |
| **Alternative Scenario** | | | | | |
| **Step** | **Action** | | | | **Use case** |
| **Exceptions** | **Exception code** | **Message** |  | **Caught** | |
| 403 | Forbidden |  | User do not have permission on this page | |
| **Business Rules** | **Code** | **Rule** | | | |
| BR-01 | In first time login, account detail and user’s detail is added into database | | | |
|  |  | | | |
|  |  |  |  |  |  |

#### 

***Table III.15****: Login feature*

#### ***3.2.2. Logout***

Table III.16: Shows the Logout feature description

| **USE CASE-02** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC02 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Logout | | | |
| **Author** | ThanhLM | | | |
| **Date** | 07/01/2025 | **Priority** | High | |
| **Actor** | User, Admin | | | |
| **Summary** | User logout system and back to guest's role | | | |
| **Goal** | Set role user to guest | | | |
| **Triggers** | User click on button “Đăng xuất“ | | | |
| **Preconditions** | User already logged in the system | | | |
| **Post Conditions** | User become guest | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User click button “Đăng xuất” in | |  | |
| 2 |  | | Logout of the system | |
| **Alternative Scenario** | | | | |
| **Step** | **Action** | | | **Use case** |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| N/A |  |  | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

***Table III.16****: Logout feature*

#### ***3.2.3. View Profile***

Table III.17: Shows the view profile feature description

| **USE CASE-03** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC03 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View Profile | | | |
| **Author** | … | | | |
| **Date** | 09/01/2025 | **Priority** | High | |
| **Actor** | User, Admin | | | |
| **Summary** | Actor can view their information in this page | | | |
| **Goal** | Display user’s profile successfully | | | |
| **Triggers** | Click in “My Profile” in avatar icon on navbar | | | |
| **Preconditions** | User has been login | | | |
| **Post Conditions** | Display all details of user | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User click in “My Profile” in avatar icon on navbar | | Redirect to user profile page | |
| 2 |  | | Display user’s detail | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 401 | Unauthorised | User is not login yet | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

***Table III.17****: View profile feature*

#### ***3.2.4. Edit Profile***

Table III.18: Shows the edit profile feature description

| **USE CASE-04** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC04 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Edit Profile | | | |
| **Author** | … | | | |
| **Date** | 09/01/2025 | **Priority** | Medium | |
| **Actor** | User, Admin | | | |
| **Summary** | Actor can update their information in this page | | | |
| **Goal** | Update user’s profile successfully | | | |
| **Triggers** | Fill in field | | | |
| **Preconditions** | Profile screen is available | | | |
| **Post Conditions** | New information is update into database | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | Fill your info in field | |  | |
| 2 | Click in button “Lưu” | | Validate user’s input | |
| 3 |  | | Save to database and log message | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **System response** |
| 3 | Update new user detail | | | User detail is invalid, log message |
| 4 | Click in button “Hủy” | | | Clear inputted data |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 401 | Unauthorised | User is not login yet | |
| **Business Rules** | **Code** | **Rule** | | |
| BR-02 | User’s detail must be valid | | |
|  |  |  |  |  |

***Table III.18*** *Edit profile feature*

#### ***3.2.5. SingUp***

Table III.19: Shows the view facility feature description

| **USE CASE-01** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Use-case No.** | UC05 | **Use-case Version** | | | 1.0 |
| **Use-case Name** | Sign Up | | | | |
| **Author** | ThanhLM | | | | |
| **Date** | 07/01/2025 | **Priority** |  | Medium | |
| **Actor** | User, Admin | | | | |
| **Summary** | Allows users to create an account on the system. | | | | |
| **Goal** | Enable new users to register and access features. | | | | |
| **Triggers** | Guest clicks on the "Sign Up" button. | | | | |
| **Preconditions** | Guest is not registered. | | | | |
| **Post Conditions** | User account is created, and the user can log in. | | | | |

***Table III.19****: View facility feature*

#### ***3.2.6. View homepage***

Table III.20: Shows the view facilities detail feature description

| **USE CASE-06** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC06 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View homepage | | | |
| **Author** |  | | | |
| **Date** | 15/01/2025 | **Priority** | Complex | |
| **Actor** | Guest, User, Admin | | | |
| **Summary** | Displays key features and introductory information. | | | |
| **Goal** | Provide an overview of the system to users. | | | |
| **Triggers** | User visits the website homepage. | | | |
| **Preconditions** | None | | | |
| **Post Conditions** | Home page is displayed with relevant information. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User opens the website. | | System displays the home page with key features and introductory content. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| N/A |  |  | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

***Table III.20****: View facilities detail feature*

#### ***3.2.7. View List Court***

Table III.21: Shows the evaluate feature description

| **USE CASE-07** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC07 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View List Court | | | |
| **Author** |  | | | |
| **Date** | 20/01/2025 | **Priority** | Simple | |
| **Actor** | User, Admin | | | |
| **Summary** | Enables users to view a list of courts. | | | |
| **Goal** | Display all available courts. | | | |
| **Triggers** | User clicks on "View Courts" in the menu. | | | |
| **Preconditions** | None | | | |
| **Post Conditions** | List of courts is displayed. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User clicks on "View Courts." | | System retrieves and displays the list of available courts. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 404 | Courts not found. | System fails to retrieve court data. | |
| **Business Rules** | **Code** | **Rule** | | |
|  |  | | |
|  |  |  |  |  |

***Table III.21****: Evaluate feature*

#### ***3.2.8. View Court Detail***

Table III.22: Shows the create booking feature description

| **USE CASE-08** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC08 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View Court Detail | | | |
| **Author** | HungDV | | | |
| **Date** | 25/01/2025 | **Priority** | Simple | |
| **Actor** | User, Admin | | | |
| **Summary** | Shows details of a specific court. | | | |
| **Goal** | Provide comprehensive details about a selected court. | | | |
| **Triggers** | User clicks on a court from the court list. | | | |
| **Preconditions** | Court is listed in the system. | | | |
| **Post Conditions** | Detailed information about the selected court is displayed. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User selects a court from the list. | | System displays detailed information about the court. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 404 | Court not found. | Selected court details are missing in the system | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

***Table III.22*** *Create booking feature*

#### ***3.2.9. Book Court***

Table III.23: Shows the view notification list feature description

| **USE CASE-09** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC09 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Book Court | | | |
| **Author** | HungDV | | | |
| **Date** | 30/01/2025 | **Priority** | Complex | |
| **Actor** | User, Admin | | | |
| **Summary** | Enables users to book a court. | | | |
| **Goal** | Allow users to reserve a court successfully. | | | |
| **Triggers** | User clicks on the "Book Now" button for a specific court. | | | |
| **Preconditions** | User is logged in, and the court is available. | | | |
| **Post Conditions** | Court booking is successfully created and stored in the system. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User selects a court and fills in booking details. | | System validates availability and confirms the booking. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  | If the court is unavailable, the system displays an error message. | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 409 | Court already booked. | User attempts to book a court that is not available. | |
| **Business Rules** | **Code** | **Rule** | | |
| BR-03 | Notification of user and admin have to mapping | | |
|  |  |  |  |  |

***Table III.23****: View notification feature*

#### ***3.2.10. Forgot Password***

#### 

Table III.24 : Shows the chat feature description

| **USE CASE-10** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC10 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Forgot Password | | | |
| **Author** | Minhlvh | | | |
| **Date** | 02/03/2025 | **Priority** | Medium | |
| **Actor** | User, Admin | | | |
| **Summary** | Provides password recovery options. | | | |
| **Goal** | Help users regain access to their account. | | | |
| **Triggers** | User clicks the "Forgot Password" link on the login page. | | | |
| **Preconditions** | User account exists in the system. | | | |
| **Post Conditions** | Password recovery email or method is initiated. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User enters their registered email. | | System sends a password recovery email. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 404 | Email not found | User enters an email that is not registered. | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

***Table III.24****: Chat feature*

#### ***3.2.11. Change Password***

Table III.25: Shows the view booking list feature description

| **USE CASE-11** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC11 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Change Password | | | |
| **Author** | Minhlvh | | | |
| **Date** | 03/03/2025 | **Priority** | Simple | |
| **Actor** | Admin, User | | | |
| **Summary** | Allows users to update their password. | | | |
| **Goal** | Enable users to change their password successfully. | | | |
| **Triggers** | User accesses the "Change Password" feature in settings. | | | |
| **Preconditions** | User is logged in. | | | |
| **Post Conditions** | New password is saved in the system. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User enters their current password and a new password. | | System validates the input and updates the password. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  | If the current password is incorrect, the system displays an error message. | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 401 | Unauthorised | User is not login yet | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

***Table III.25****: View booking feature*

#### ***3.2.12. Send Notifications***

Table III.26: Shows the booking processing feature description

| **USE CASE-12** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC12 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Send Notifications | | | |
| **Author** |  | | | |
| **Date** | 05/03/2025 | **Priority** | Medium | |
| **Actor** | System | | | |
| **Summary** | System sends notifications to users. | | | |
| **Goal** | Notify users about important updates or events. | | | |
| **Triggers** | System event triggers the notification. | | | |
| **Preconditions** | Relevant events occur in the system. | | | |
| **Post Conditions** | Notifications are sent to users. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | System generates a notification based on an event. | | Notification is sent to the user. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **System Response** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 500 | Notification failed. | System encounters an error while sending notifications. | |
| **Business Rules** | **Code** | **Rule** | | |
|  |  | | |
|  |  |  |  |  |

***Table III.26****: Booking processing feature*

#### ***3.2.13. View Notifications List***

#### 

Table III.27: Shows the Block/Unblock account feature description

| **USE CASE-13** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC13 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View Notifications List | | | |
| **Author** |  | | | |
| **Date** | 07/03/2025 | **Priority** | Simple | |
| **Actor** | User | | | |
| **Summary** | Enables users to view notifications. | | | |
| **Goal** | Display a list of user notifications. | | | |
| **Triggers** | User clicks on the "Notifications" icon. | | | |
| **Preconditions** | User is logged in. | | | |
| **Post Conditions** | Notifications list is displayed. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User opens the notifications tab. | | System displays a list of notifications. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 500 | Failed to load notifications. | System encounters an error while retrieving notifications. | |
| **Business Rules** | **Code** | **Rule** | | |
|  |  | | |
|  |  |  |  |  |

***Table III.27****: Block/ Unblock feature*

#### ***3.2.14. View Booking Schedule***

Table III.28: Shows the view statistic feature description

| **USE CASE-14** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC14 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View Booking Schedule | | | |
| **Author** |  | | | |
| **Date** | 10/01/2025 | **Priority** | Simple | |
| **Actor** | Admin, User | | | |
| **Summary** | Enables users to view booking schedules. | | | |
| **Goal** | Provide a schedule of court bookings. | | | |
| **Triggers** | User clicks on "View Schedule" in the menu. | | | |
| **Preconditions** | User is logged in. | | | |
| **Post Conditions** | Booking schedule is displayed. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User selects the "View Schedule" option. | | System retrieves and displays the booking schedule. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 404 | Schedule not found. | System fails to retrieve booking schedule. | |
| **Business Rules** | **Code** | **Rule** | | |
|  |  | | |
|  |  |  |  |  |

##### 

***Table III.28****: View statistic feature*

#### ***3.2.15. View Booking Detail***

Table III.29: Shows the create facility feature description

| **USE CASE-15** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC15 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View Booking Detail | | | |
| **Author** |  | | | |
| **Date** | 13/01/2025 | **Priority** | Simple | |
| **Actor** | User | | | |
| **Summary** | Shows details of a specific booking. | | | |
| **Goal** | Provide detailed information about a selected booking. | | | |
| **Triggers** | User clicks on a specific booking from the schedule. | | | |
| **Preconditions** | Booking exists in the system. | | | |
| **Post Conditions** | Booking details are displayed. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User selects a booking. | | System retrieves and displays the booking details. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **System response** |
|  |  | |  | |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 404 | Booking not found. | System cannot find the requested booking. | |
| **Business Rules** | **Code** | **Rule** | | |
|  |  | | |
|  |  |  |  |  |

***Table III.29****: Create facility feature*

#### ***3.2.16. View Booking History***

Table III.30: Shows the update facility feature description

| **USE CASE-16** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC16 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View Booking History | | | |
| **Author** | HungDV | | | |
| **Date** | 15/02/2025 | **Priority** | Simple | |
| **Actor** | Admin, User | | | |
| **Summary** | Displays previous bookings | | | |
| **Goal** | Allow users to view their past bookings | | | |
| **Triggers** | User clicks on "Booking History" in the menu | | | |
| **Preconditions** | User is logged in | | | |
| **Post Conditions** | Booking history is displayed | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User accesses "Booking History." | | System retrieves and displays the user's booking history. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **System Response** |
| 3 | Click “Hủy” | | | Back to facilities list |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 500 | Failed to retrieve booking history. | System encounters an error while fetching booking history. | |
| **Business Rules** | **Code** | **Rule** | | |
|  |  | | |
|  |  |  |  |  |

***Table III.30****: Update facility feature*

#### ***3.2.17. Manage Court***

Table III.31: Shows the view facility updating history feature description

| **USE CASE-17** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC17 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Manage Court | | | |
| **Author** | HungDV | | | |
| **Date** | 20/02/2025 | **Priority** | Simple | |
| **Actor** | Admin | | | |
| **Summary** | Allows administrators to manage courts. | | | |
| **Goal** | Enable administrators to add, edit, or delete court listings. | | | |
| **Triggers** | Admin accesses the "Manage Court" option. | | | |
| **Preconditions** | Admin is logged in. | | | |
| **Post Conditions** | Court data is updated in the system. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | Admin modifies court information and saves changes. | | System validates and updates the court data. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 400 | Invalid input. | Admin provides invalid court data. | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

***Table III.31****: View facility feature*

#### ***3.2.18. Search and Filter Court***

Table III.32: Shows the active/inactive feature description

| **USE CASE-18** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC18 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Search and Filter Court | | | |
| **Author** |  | | | |
| **Date** | 23/01/2025 | **Priority** | Simple | |
| **Actor** | Admin, User, Guest | | | |
| **Summary** | Allows users to search and filter courts. | | | |
| **Goal** | Enable users to find courts based on specific criteria. | | | |
| **Triggers** | User enters search criteria or applies filters. | | | |
| **Preconditions** | Courts are listed in the system. | | | |
| **Post Conditions** | Search results are displayed. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User enters search criteria or selects filters. | | System retrieves and displays matching courts. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **System Response** |
| 2 |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 404 | No courts found. | System cannot find courts matching the search criteria. | |
| **Business Rules** | **Code** | **Rule** | | |
|  |  | | |
|  |  |  |  |  |

***Table III.32****: Active/Inactive facility feature*

#### ***3.2.19. Manage Users***

Table III.33: Shows the create category feature description

| **USE CASE-19** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC19 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Manage Users | | | |
| **Author** |  | | | |
| **Date** | 25/02/2025 | **Priority** | Simple | |
| **Actor** | Admin | | | |
| **Summary** | Allows administrators to manage user accounts. | | | |
| **Goal** | Enable admins to add, edit, or delete user accounts | | | |
| **Triggers** | Admin accesses the "Manage Users" option | | | |
| **Preconditions** | Admin is logged in | | | |
| **Post Conditions** | User account data is updated in the system | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | Admin selects the "Manage Users" option. | |  | |
| 2 | Admin adds, edits, or deletes user account information. | | System validates and applies the changes to the user account data. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **System Response** |
| 3 | Select “Thêm” | | Validate inputted data | |
| 4 |  | | Return error and require input data again | |
| 5 | Re-input data and select “Thêm” | | Validate inputted data | |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 403 | Unauthorized access. | Admin does not have sufficient permissions to manage user accounts. | |
| **Business Rules** | **Code** | **Rule** | | |
|  |  |  |  |  |

***Table III.33****: Create category feature*

#### ***3.2.20. Set Availability Status***

Table III.34: Shows the update category feature description

| **USE CASE-20** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC20 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Set Availability Status | | | |
| **Author** |  | | | |
| **Date** | 26/02/2025 | **Priority** | Simple | |
| **Actor** | Admin | | | |
| **Summary** | Allows administrators to change the availability status of courts. | | | |
| **Goal** | Enable admins to set courts as available or unavailable. | | | |
| **Triggers** | Admin selects a court and updates its status. | | | |
| **Preconditions** | Admin is logged in. | | | |
| **Post Conditions** | Court availability status is updated in the system. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | Admin selects a court and changes its availability status. | | System updates the court’s status accordingly. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **System Response** |
| 3 | Click “Hủy” | | | Back to facilities list |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 500 | Failed to update court status. | System encounters an error while updating the status. | |
| **Business Rules** | **Code** | **Rule** | | |
|  |  | | |
|  |  |  |  |  |

***Table III.34****: Update category feature*

#### ***3.2.21. Send Email Reminder***

Table III.35: Shows the view category feature description

##### 

| **USE CASE-21** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC21 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Send Email Reminder | | | |
| **Author** |  | | | |
| **Date** | 27/02/2025 | **Priority** | Medium | |
| **Actor** | Admin, User | | | |
| **Summary** | Sends reminders for upcoming bookings to users. | | | |
| **Goal** | Notify users about their upcoming bookings. | | | |
| **Triggers** | Booking reminders are triggered by the system or admin manually initiates it. | | | |
| **Preconditions** | Booking exists in the system. | | | |
| **Post Conditions** | Reminder email is sent successfully. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | System identifies upcoming bookings. | | System sends reminder emails to users. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
| 1 | Admin manually triggers a reminder email. | | |  |
| 2 | System sends the reminder email successfully. | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 500 | Failed to send email. | System encounters an error while sending the email. | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

***Table III.35****: View category feature*

#### ***3.2.22. Cancel Booking***

Table III.35: Shows the view category feature description

##### 

| **USE CASE-22** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC22 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Cancel Booking | | | |
| **Author** | HungDV | | | |
| **Date** | 27/02/2025 | **Priority** | Simple | |
| **Actor** | Admin, User | | | |
| **Summary** | Allows users or admins to cancel an existing booking. | | | |
| **Goal** | Enable cancellation of bookings. | | | |
| **Triggers** | User or admin selects a booking and chooses the "Cancel" option. | | | |
| **Preconditions** | Booking exists in the system. | | | |
| **Post Conditions** | Booking is removed from the system. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User selects a booking and confirms cancellation. | | System removes the booking from the database. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 404 | Booking not found. | The selected booking does not exist in the system. | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

#### ***3.2.23. Payment Integration***

Table III.35: Shows the view category feature description

##### 

| **USE CASE-23** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC23 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Payment Integration | | | |
| **Author** |  | | | |
| **Date** | 27/02/2025 | **Priority** | Medium | |
| **Actor** | User | | | |
| **Summary** | Handles payments for court bookings. | | | |
| **Goal** | Allow users to pay for their bookings securely. | | | |
| **Triggers** | User proceeds to payment after booking a court. | | | |
| **Preconditions** | Booking details are confirmed. | | | |
| **Post Conditions** | Payment is processed successfully, and booking is confirmed. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User proceeds to payment gateway. | | System sends reminder emails to users. | |
| 2 | User enters payment details and completes the transaction. | | System confirms payment and finalizes the booking. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
| 1 | User cancels the payment process. | | |  |
| 2 | System returns the user to the booking page without confirming the booking | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 402 | Payment required. | User fails to complete the payment process. | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

#### ***3.2.24. Create Feedback***

Table III.35: Shows the view category feature description

##### 

| **USE CASE-24** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC24 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Create Feedback | | | |
| **Author** | HungDV | | | |
| **Date** | 27/02/2025 | **Priority** | Medium | |
| **Actor** | User | | | |
| **Summary** | Allows users to submit feedback about courts. | | | |
| **Goal** | Enable users to share their experiences. | | | |
| **Triggers** | User clicks on the "Submit Feedback" option for a court. | | | |
| **Preconditions** | User is logged in. | | | |
| **Post Conditions** | Feedback is saved in the system. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User submits feedback through the feedback form. | | System validates and saves the feedback. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 400 | Invalid input. | User provides invalid feedback details. | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

#### ***3.2.25. View Feedback***

Table III.35: Shows the view category feature description

##### 

| **USE CASE-25** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC25 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View Feedback | | | |
| **Author** |  | | | |
| **Date** | 27/02/2025 | **Priority** | Simple | |
| **Actor** | Admin, User | | | |
| **Summary** | Allows users or admins to view feedback for courts. | | | |
| **Goal** | Display feedback submitted by users. | | | |
| **Triggers** | User or admin selects the "View Feedback" option. | | | |
| **Preconditions** | Feedback exists in the system. | | | |
| **Post Conditions** | Feedback details are displayed. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User selects a court and views feedback. | | System retrieves and displays feedback for the selected court. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 404 | Feedback not found. | No feedback exists for the selected court. | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

#### ***3.2.26. Edit Feedback***

Table III.35: Shows the view category feature description

##### 

| **USE CASE-26** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC26 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Edit Feedback | | | |
| **Author** |  | | | |
| **Date** | 27/02/2025 | **Priority** | Medium | |
| **Actor** | User | | | |
| **Summary** | Allows users to edit their submitted feedback. | | | |
| **Goal** | Enable users to modify their feedback. | | | |
| **Triggers** | User selects a feedback entry and clicks "Edit." | | | |
| **Preconditions** | Feedback exists, and the user is the author. | | | |
| **Post Conditions** | Feedback is updated successfully. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User modifies their feedback and submits the changes. | | System validates and updates the feedback. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
| 1 | Admin manually triggers a reminder email. | | |  |
| 2 | System sends the reminder email successfully. | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 403 | Unauthorized access. | User attempts to edit feedback they did not create. | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

#### ***3.2.27. Delete Feedback***

Table III.35: Shows the view category feature description

##### 

| **USE CASE-27** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC27 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Delete Feedback | | | |
| **Author** | HungDV | | | |
| **Date** | 27/02/2025 | **Priority** | Medium | |
| **Actor** | Admin, User | | | |
| **Summary** | Allows users or admins to delete feedback. | | | |
| **Goal** | Remove feedback from the system. | | | |
| **Triggers** | User or admin selects a feedback entry and clicks "Delete." | | | |
| **Preconditions** | Feedback exists. | | | |
| **Post Conditions** | Feedback is removed from the system. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User or admin confirms deletion of feedback. | | System removes the feedback from the database. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 404 | Feedback not found. | Selected feedback does not exist. | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

#### ***3.2.28. Court Rating System***

| **USE CASE-21** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC21 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Court Rating System | | | |
| **Author** |  | | | |
| **Date** | 27/02/2025 | **Priority** | Medium | |
| **Actor** | Admin, User | | | |
| **Summary** | Allows users to rate courts. | | | |
| **Goal** | Collect and display court ratings. | | | |
| **Triggers** | User selects a court and provides a rating. | | | |
| **Preconditions** | User is logged in. | | | |
| **Post Conditions** | Rating is saved and aggregated in the system. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | User submits a rating for a court. | | System validates and saves the rating. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 400 | Invalid input. | User provides an invalid rating (e.g., out of range). | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

Table III.35: Shows the view category feature description

#### ***3.2.29. Send mail reminder***

##### 

| **USE CASE-21** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC21 | **Use-case Version** | | 1.0 |
| **Use-case Name** | Send Email Reminder | | | |
| **Author** |  | | | |
| **Date** | 27/02/2025 | **Priority** | Medium | |
| **Actor** | Admin, User | | | |
| **Summary** | Allows system send mail to user | | | |
| **Goal** | Alert user keep track of time. | | | |
| **Triggers** | System auto send mail when time up | | | |
| **Preconditions** | User already booking court. | | | |
| **Post Conditions** | Recipient sees the email in their inbox. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | Reminder time is reached. | | System retrieves recipient details and email content. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 408 | RequestTimeout | System over load | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

#### ***3.2.30. View booking schedule***

##### 

| **USE CASE-21** | | | | |
| --- | --- | --- | --- | --- |
| **Use-case No.** | UC30 | **Use-case Version** | | 1.0 |
| **Use-case Name** | View booking schedule | | | |
| **Author** | Minhlvh | | | |
| **Date** | 25/3/2025 | **Priority** | Medium | |
| **Actor** | Admin, User | | | |
| **Summary** | Display to users booking list from current day | | | |
| **Goal** | Alert user keep track of time. | | | |
| **Triggers** | User press “đặt sân ngay” button | | | |
| **Preconditions** | User already booking court. | | | |
| **Post Conditions** | Recipient sees the email in their inbox. | | | |
| **Main Success Scenario** | | | | |
| **Step** | **Actor Event** | | **System response** | |
| 1 | Reminder time is reached. | | System retrieves recipient details and email content. | |
| **Alternative Scenario** | | | | |
| **#** | **Action** | | | **Use case** |
|  |  | | |  |
| **Exceptions** | **Exception code** | **Message** | **Caught** | |
| 408 | RequestTimeout | System over load | |
| **Business Rules** | **Code** | **Rule** | | |
| N/A |  | | |
|  |  |  |  |  |

## **4. Non-Functional Requirements**

### **4.1 External Interfaces**

#### ***4.1.1. User interfaces***

- UI-01: The website shall display correctly without glitches on popular browsers including Chrome, Firefox, Safari and Edge.

- UI-02: FMS screen displays shall conform to the Process Impact Internet Application User Interface Standard.

- UI-03: Audit form interface: An interface that enables auditors to input data related to safety inspections, including checklist items, observations, photographs, and comments.

- UI-04: The system should responsively in mobile device

#### ***4.2.2 Security interfaces***

- SI-01: Access control interface: An interface that enforces role-based access control, defining user permissions and restrictions based on their roles within the system.

### **4.2 Quality Attributes**

#### ***4.2.1 Usability***

- US-01: The system will be designed with simple and friendly user interfaces.

- US-02: Users shall become familiar and productive at main operations after a short time using the application.

- US-03: Clear and informative error messages should be provided to guide users in resolving any issues or mistakes encountered during system use.

#### ***4.2.2 Reliability***

- RE-01: The system should be highly available, ensuring minimal downtime to support continuous access and use by auditors and stakeholders.

- RE-02: All data entered by the user is validated before sending it to the server

#### ***4.2.3 Performance***

- PE-01: The system should respond promptly to user actions, ensuring minimal delay when loading pages, saving data, or generating reports.

- PE-02: The system should be able to handle an increasing number of concurrent users and data volumes without significant degradation in performance.

- PE-03: The system should utilize system resources, such as memory and processing power, efficiently to deliver optimal performance.

#### ***4.2.4 Security***

- SE-01: The system should ensure that sensitive data, such as audit reports and user information, is protected from unauthorised access or disclosure.

- SE-02: The system should prevent unauthorized modification or tampering of data, ensuring that audit records remain accurate and reliable.

- SE-03: Robust mechanisms should be implemented to verify the identity of users and grant appropriate access privileges based on their roles and responsibilities.

## **5. Requirement Appendix**

### **5.1 Business Rules**

Table III.36: Description rule definition of the system

| **ID** | **Rule Definition** |
| --- | --- |
| BR-01 | In first time login, account detail and user’s detail is added into database |
| BR-02 | User’s detail must be valid |
| BR-03 | Notification of user and admin have to mapping |
| BR-04 | Statistic and chart have to mapping with database |
| BR-05 | New facility can’t be duplicate with existed facilities |
| BR-06 | New facility’s detail can’t duplicate with other facilities |
| BR-07 | When inactive a facilities, all the pending request will be decline with reason “Cơ sở này không còn khả dụng” |
| BR-08 | New category can’t be duplicate with existed categories |
| BR-09 | New facility’s detail can’t duplicate with other facilities |

***Table III.36****: Business rules*

### **5.2 Application Messages List**

Table III.37: Shows the application messages list of the system

| **#** | **Message code** | **Message Type** | **Context** | **Content** |
| --- | --- | --- | --- | --- |
| 1 | MSG01 | In red, above the button sign-in google | You have to you FPT email | Bạn phải sử dụng email FPT |
| 2 | MSG02 | In line, inside table | There is not any search result | Không có dữ liệu |
| 3 | MSG03 | Popup message | Successfully | Thành công |
| 4 | MSG04 | Popup message | Error | Lỗi |
| 5 | MSG05 | Popup message | Notice | Hành động này sẽ từ chối hết tất cả các pending request của facility này |
| 6 | MSG07 | Inline, under the text box | Not fill in required field | This field is required |

***Table III.37****: Messages List*

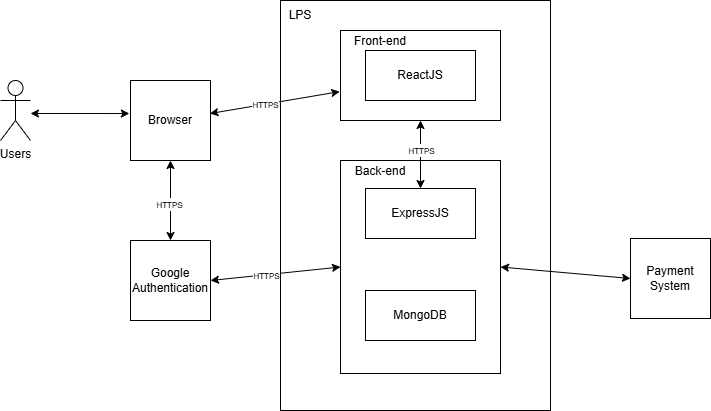
# 

# 

# **IV. Software Design Description**

## **1. System Design**

### **1.1 System Architecture**



- Figure IV.1 shows the software system architecture of the Facilities Management system.

***Figure IV.1****: System architecture*

### **1.2 Package Diagram**

### 

***Package Descriptions***

Table IV.4 Shows the Packages contained in the project and describes what those packages do in the project

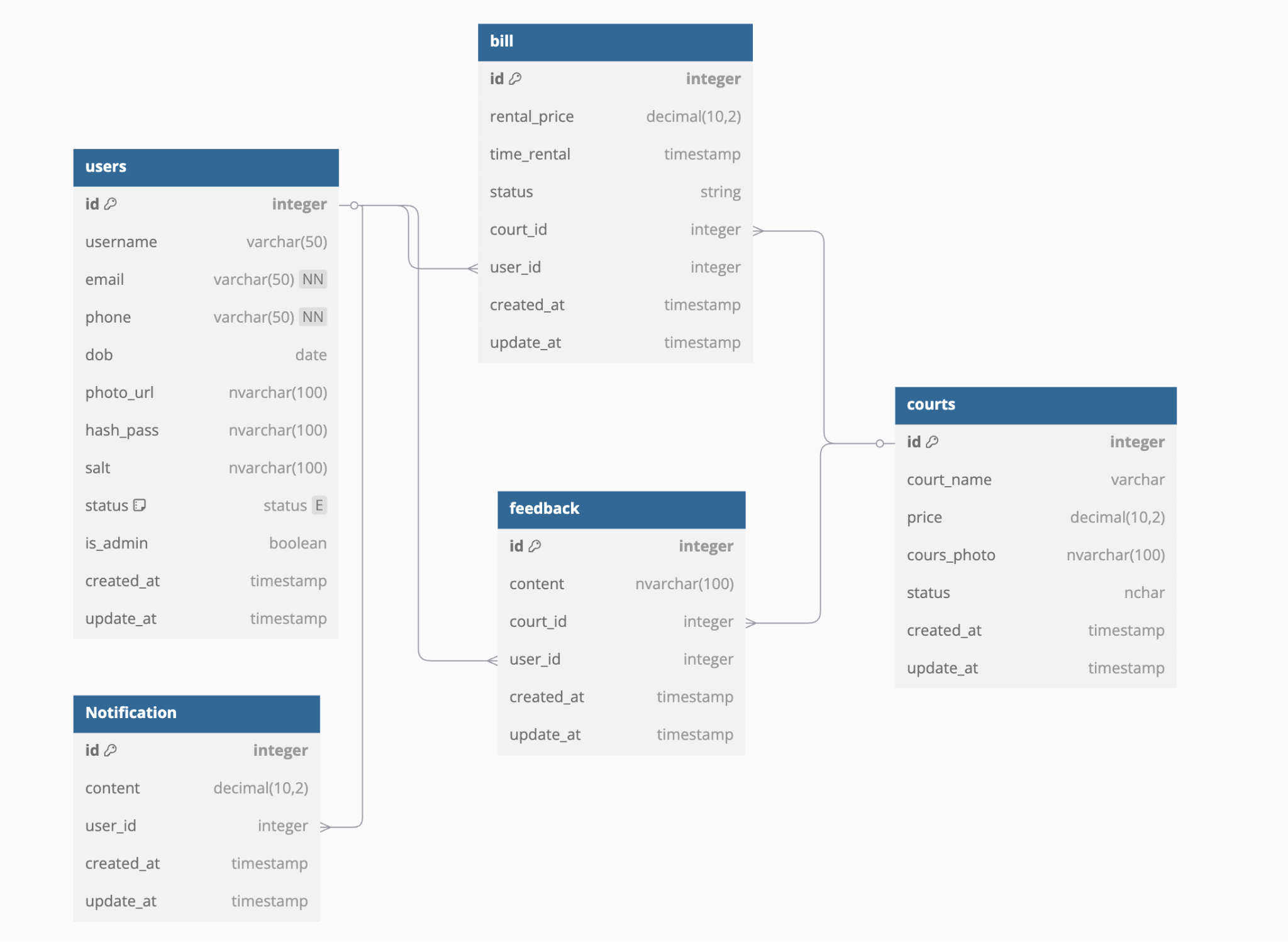
| **No** | **Package** | **Description** |
| --- | --- | --- |
| 01 | **Docker & CI/CD**: | **Docker**: Packages the application into containers to ensure a consistent environment across servers.  **CI/CD Pipeline**: Automates the processes of testing, building, and deploying the application.   * **Build**: Generates an application build from the source code. * **Test**: Runs automated tests to ensure the quality of the source code. * **Deploy**: Deploys the application to the production or staging environment. |
| 02 | **Express Application**: | **AppModule**: The root module of the application, containing main configurations and initializing other components like controllers, services, and middleware.  **Middleware**: Handles tasks before requests reach the controllers, such as logging or security checks. |
| 03 | **Application** | **Controller:** Handles HTTP requests from users and calls services to execute business logic.  **Service:** Executes the business logic of the application, processes data, and interacts with repositories.  **Repository:** Retrieves and stores data from the database (MongoDB) as needed by the application.  **DTO (Data Transfer Object):** Used to transfer data between layers of the application, ensuring data validity and structure.  **Interface:** Defines data types and structures to ensure consistency across the components**.** |
| 04 | **Common** | **ExceptionFilter:** Handles errors during request processing, returning appropriate error responses to the user.  **AuthGuard:** Protects routes that require access control, ensuring users are authenticated and authorized to access resources. |
| 05 | **Config** | **AppConfig:** Holds general application settings such as port, environment, and security configurations.  **Database Config:** Manages database connection settings (MongoDB), providing necessary credentials and access info.  **MongoDB:** The database used to store the application's data. |
| 06 | **External Server:** | **Payment:** A third-party payment service (Payment Gateway), invoked by services within the application when processing financial transactions.  **Gmail:** Gmail service used to send email notifications, called from the Service when sending email alerts to users. |

## 

***Table IV.4****: Package description*

## **2. Database Design**

Figure IV.5 is a database schema design for a Facilities Management system (Classrooms/Fields)



***Figure IV.5****: Database schema*

***Table Descriptions***

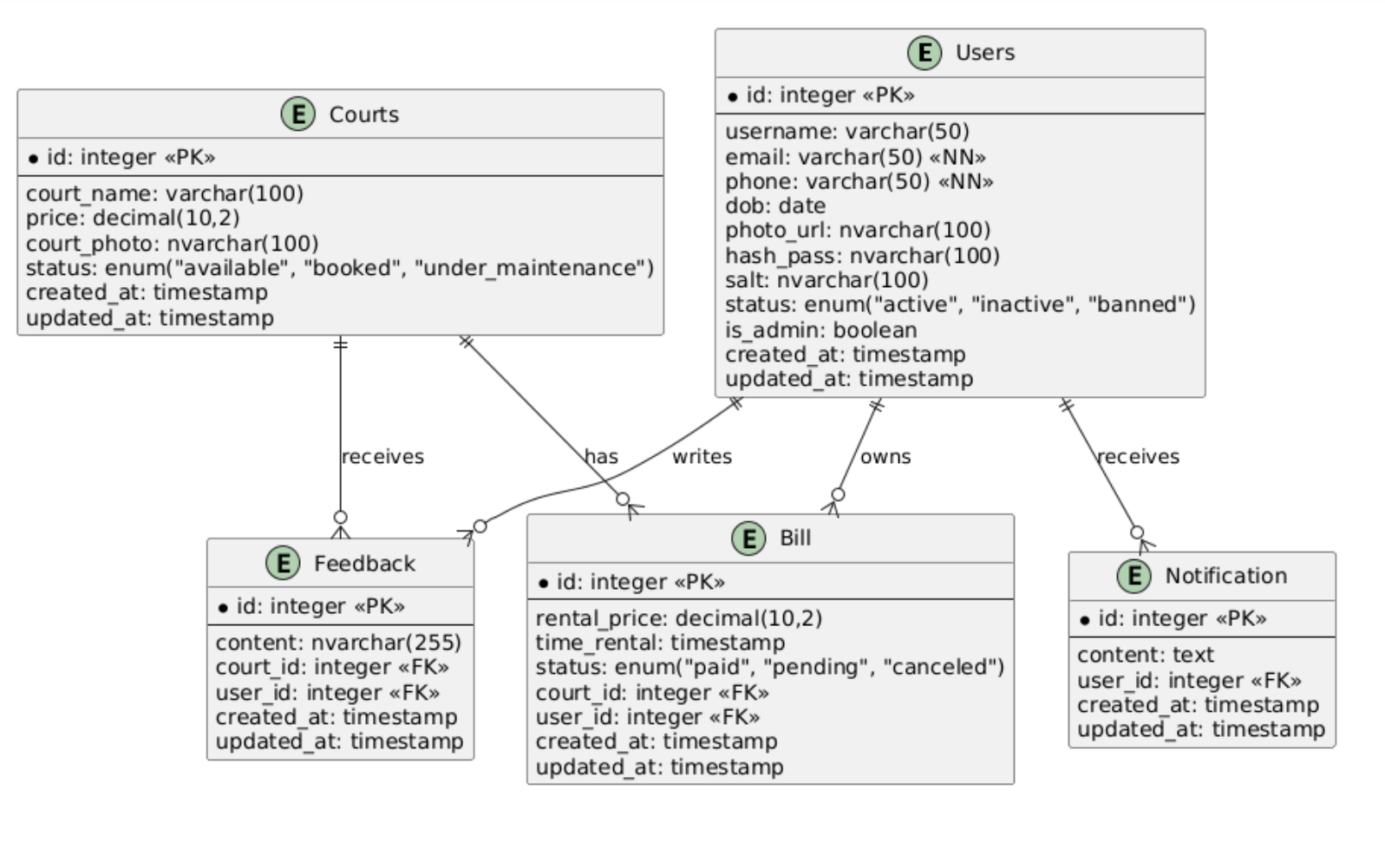
******

Table IV.6 Represents Entity entities and descriptions..

| **No** | **Table** | **Description** |
| --- | --- | --- |
| 1 | User | Store user’s detail such as role, email, name,... |
| 2 | Feedback | Store user’s feedback about court |
| 3 | Bill | Store bill’s detail such as price, userId, courtId,... And store time schedule |
| 4 | Notification | Contain notification detail |
| 5 | Court | Store court’s detail such as role, email, name |
| 6 | Borrower | Store all booking of user |

***Table IV.6****: Describe entity database*

### **2.1 User**

| **No.** | **Attribute** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- |
| 1 | **id** | ObjectId | Primary Key | Unique ID for each user. |
| 2 | username | string |  | Name of the user |
| 3 | email | String | Required | Email address of the user. |
| 4 | hash\_password | String | Required | Hash Password for the user account. |
| 5 | salt | string |  | Use to encrypt and decrypt password |
| 6 | phone | String | Required | Contact phone number of the user. |
| 7 | dob | Date |  | Birthdate of the user. |
| 8 | photo\_url | String |  | URL to the user's profile or avatar image. |
| 9 | status | string |  | Current status of the User (e.g., active, deactive). |
| 10 | is\_admin | boolean | Required | Indicates if the user is admin (true or false). |
| 11 | created\_at | Datetime |  | Timestamp for when the user account was created. |
| 12 | updated\_at | Datetime |  | Timestamp for the last update of the user record. |

### 

### **2.3 Court**

| **No.** | **Attribute** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- |
| 1 | **id** | ObjectId | Primary Key | Unique ID for each book. |
| 2 | court\_name | string | Foreign Key | Reference to the title or information about the book. |
| 3 | price | interger | Required | Unique identifier for the book (e.g., ISBN or library code). |
| 4 | court\_photo | Number | Required | Section where the book is located in the library. |
| 5 | status | String |  | Current status of the book (e.g., active, maintain, deactive). |
| 6 | created\_at | Datetime |  | Timestamp for when the book record was created. |
| 7 | updated\_at | Datetime |  | Timestamp for the last update of the book record. |

### 

### 

### **2.4Bills**

| **No.** | **Attribute** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- |
| 1 | **id** | ObjectId | Primary Key | Unique ID for each payment record. |
| 2 | user\_id | ObjectId | Foreign Key | Reference to the user making the payment. |
| 3 | court\_id | ObjectId | Foreign Key | Reference to the court details |
| 4 | retal\_price | Integer | Required | Total amount paid. |
| 5 | status | String | Required | Status of the payment (e.g., completed, pending, failed). |
| 6 | details | String |  | Additional details or notes regarding the payment. |
| 7 | created\_at | Datetime | Required | Timestamp for when the payment record was created. |
| 8 | updated\_at | Datetime |  | Timestamp for the last update of the book record. |

### 

### 

### **2.5 Feedback**

| **No.** | **Attribute** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- |
| 1 | **id** | ObjectId | Primary Key | Unique ID for each feedback record. |
| 2 | user\_id | ObjectId | Foreign Key | Reference to the user who made the feedback. |
| 3 | court\_id | ObjectId | Foreign Key | Reference to the court that the feedback is about. |
| 4 | content | String | Required | Content of the feedback or review text. |
| 6 | created\_at | Date |  | Timestamp for when the feedback was created. |
| 7 | updated\_at | Date |  | Timestamp for when the feedback was last updated. |

### 

### **2.6 Notification**

| **No.** | **Attribute** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- |
| 1 | **id** | ObjectId | Primary Key | Unique ID for each notification record. |
| 2 | user\_id | ObjectId | Foreign Key | Reference to the user who will receive the notification. |
| 3 | content | String | Required | Content or text of the notification message. |
| 6 | created\_at | Date |  | Timestamp for when the feedback was created. |
| 7 | updated\_at | Date |  | Timestamp for when the feedback was last updated. |

### **2.6 Borrower**

| **No.** | **Attribute** | **Type** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- |
| 1 | **id** | ObjectId | Primary Key | Unique ID for each notification record. |
| 2 | user\_id | ObjectId | Foreign Key | Reference to the user who will receive the notification. |
| 3 | court\_id | ObjectId | Foreign Key | Reference to the court which user rental |
| 4 | time\_rental | String | Required | Time rental of booking |
| 5 | end\_time \_rental | string | Required | End Time of booking |
| 6 | created\_at | Date |  | Timestamp for when the feedback was created. |
| 7 | updated\_at | Date |  | Timestamp for when the feedback was last updated. |

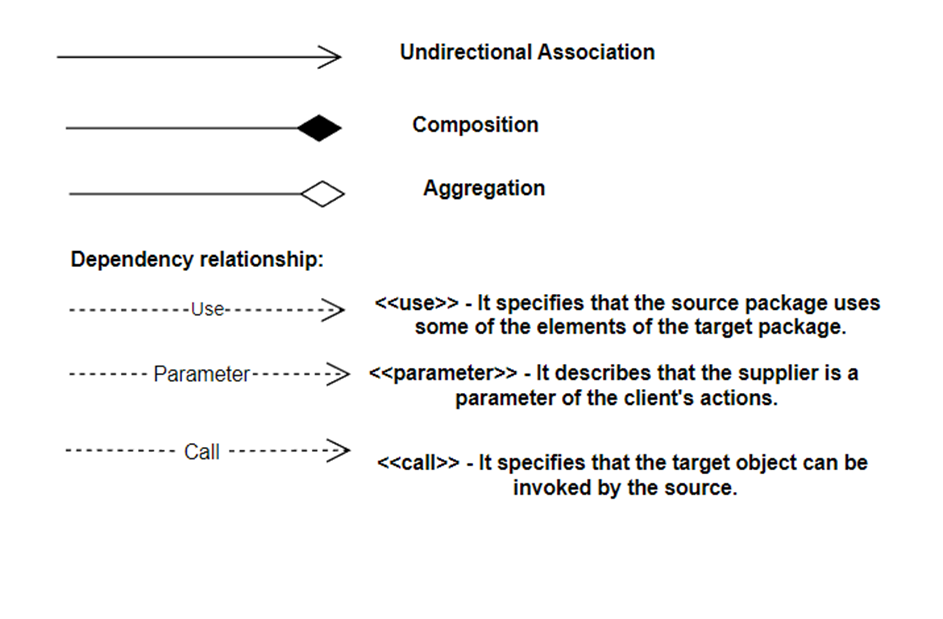
### 

### 

### 

## **3. Detailed Design**

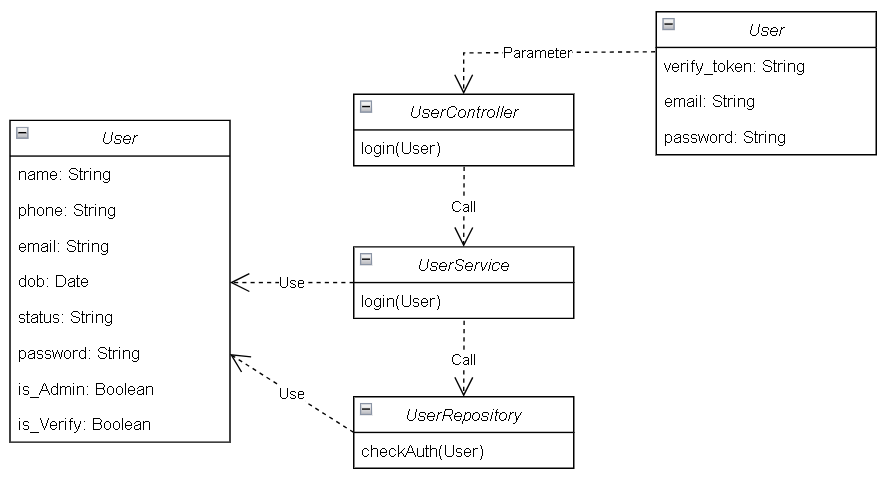
Figure IV.7 Notation for UML diagram



***Figure IV.7****: Notation UML diagram*

**1. Login**

a. Class Diagram



b. Class Specifications

#### **UserController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | login(User) | Handles the login request from the user. It calls the login method in UserService.  Inputs: User object containing login credentials.  Outputs: Authentication result (success/failure).  Internal Processing: Passes the User object to UserService for authentication. |

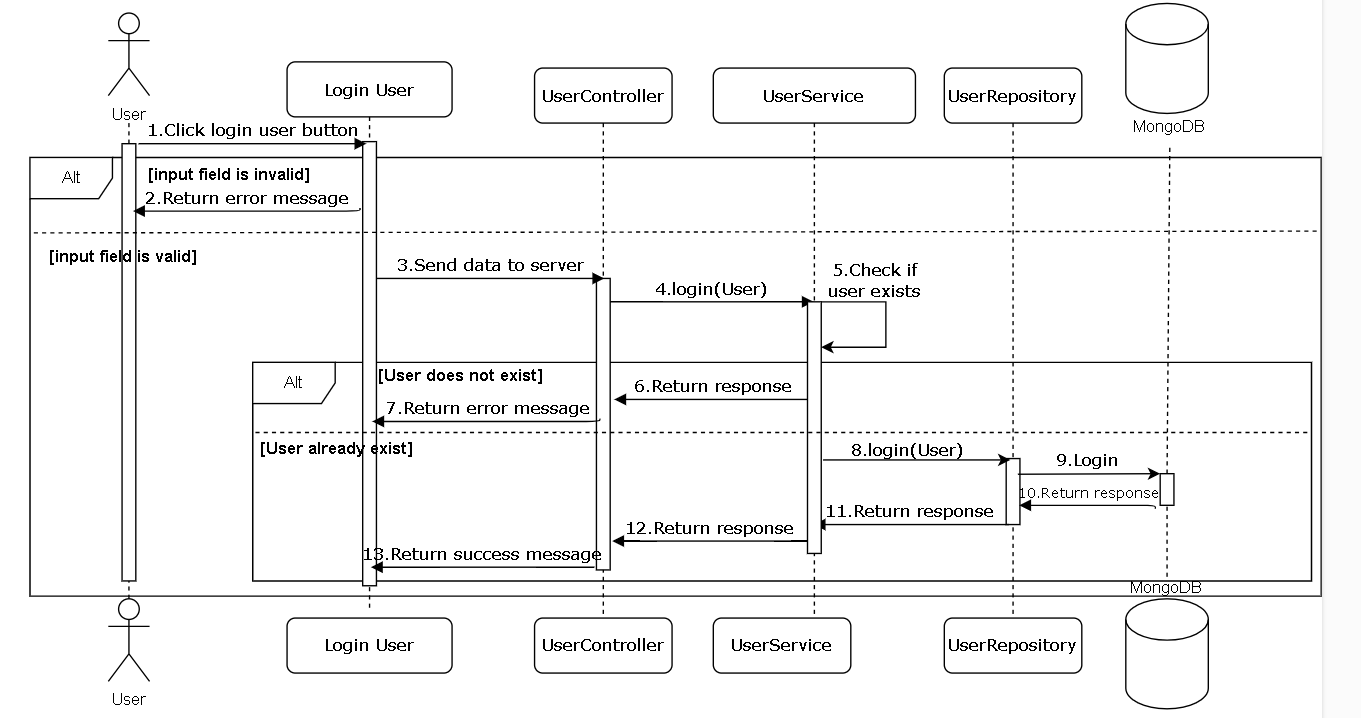
#### **UserService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | login(User) | Processes the login request by calling the checkAuth method in UserRepository.  Inputs: User object containing login credentials.  Outputs: Authentication result (success/failure).  Internal Processing: Validates the user credentials by interacting with UserRepository. |

#### **UserRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | login(User) | Checks the authentication of the user against the database.  Inputs: User object containing login credentials.  Outputs: Boolean indicating if the user is authenticated.  Internal Processing: Queries the database to verify the user's credentials. |

c. Sequence Diagram



d. Database Queries

Find Query for Authentication:

db.users.findOne({

username: "input\_username",

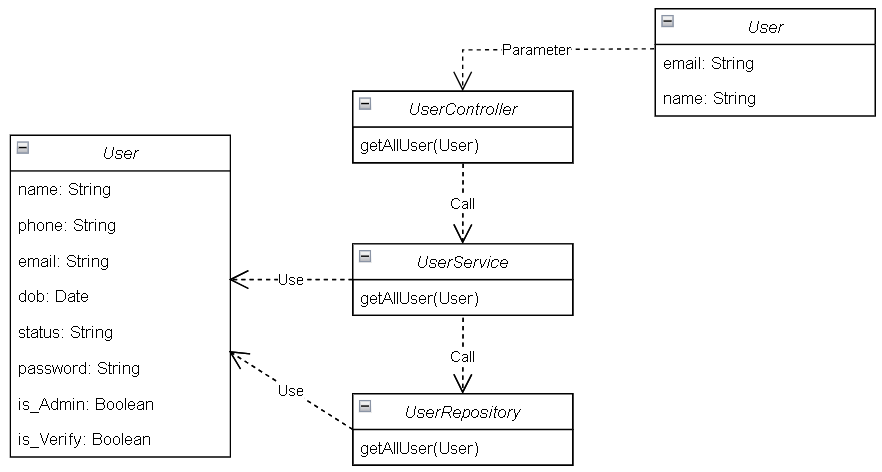
hash\_pass: "input\_hashed\_password"

});

**2. Manage User**

**2.1 Get User**

a. Class Diagram



b. Class Specifications

#### **UserController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllUser(User) | Handles the request to get all users. It calls the getAllUser method in UserService.  Inputs: User object containing user information.  Outputs: Result of getting all users (success/failure).  Internal Processing: Passes the User object to UserService for processing. |

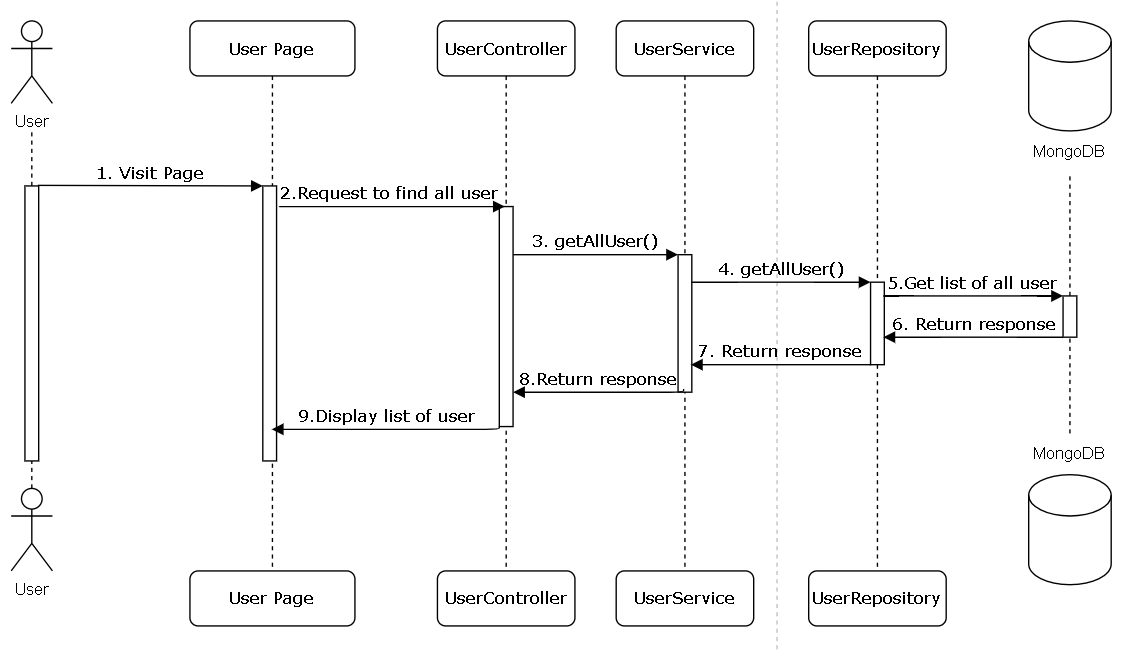
#### **UserService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllUser(User) | Processes the request to get all users by calling the getAllUser method in UserRepository.  Inputs: User object containing user information.  Outputs: Result of getting all users (success/failure).  Internal Processing: Interacts with UserRepository to get all users. |

#### **UserRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllUser(User) | Retrieves all the users from the database.  Inputs: User object containing user identifier (e.g., userId).  Outputs: All users.  Internal Processing: Executes the query to fetch all users from the database. |

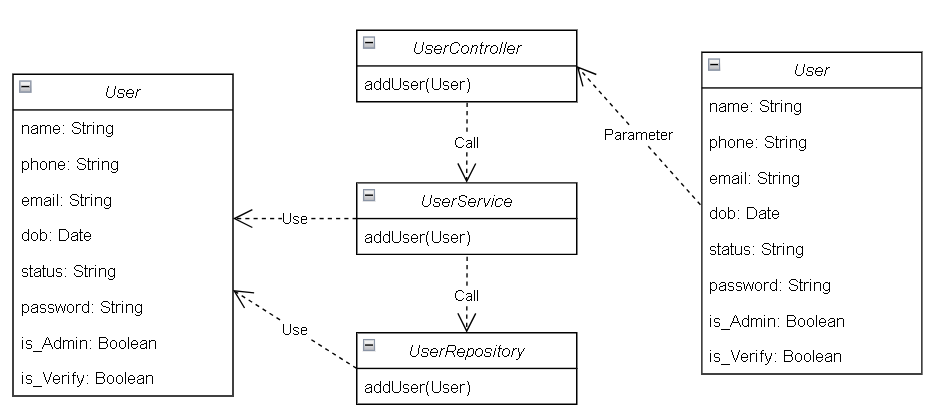
c. Sequence Diagram



d. Database Queries

**2.2 Add User**

a. Class Diagram



b. Class Specifications

#### **UserController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | add(User) | Handles the request to add a new user. It calls the addUser method in UserService.  Inputs: User object containing user information.  Outputs: Result of adding the user (success/failure).  Internal Processing: Passes the User object to UserService for processing. |

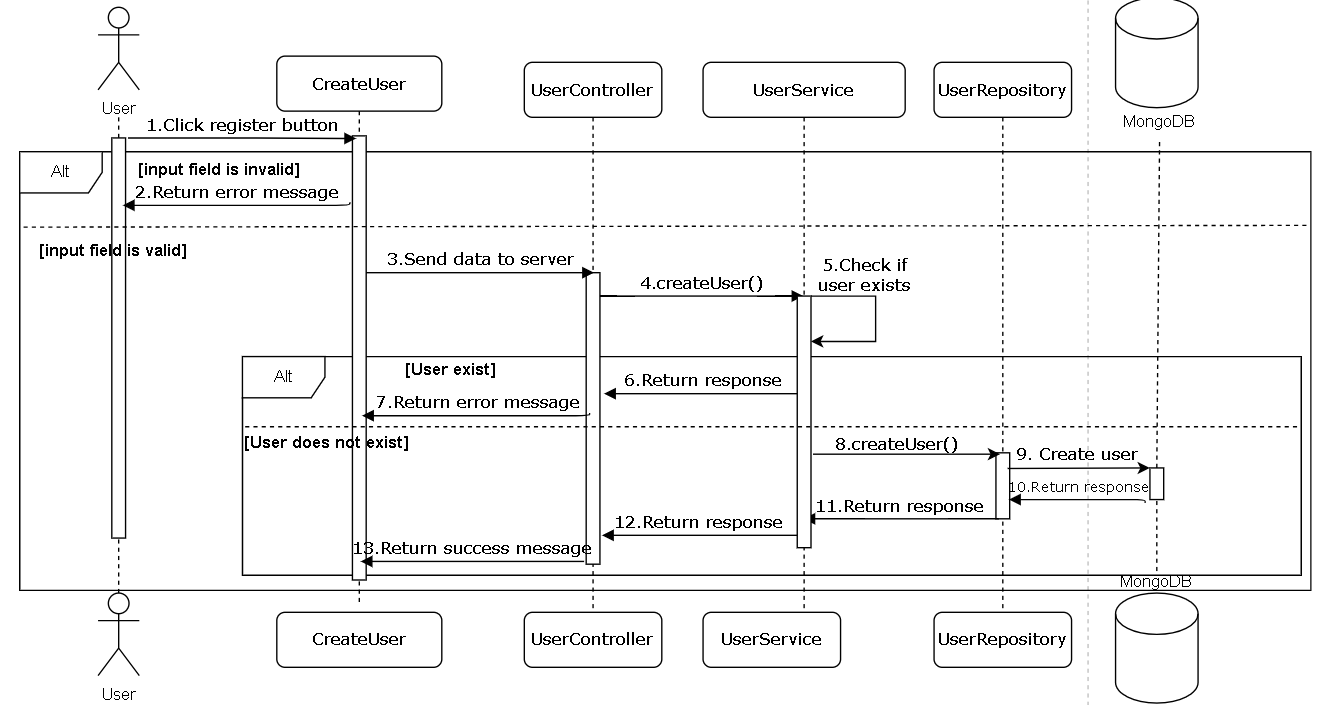
#### **UserService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | add(User) | Processes the request to add a user by calling the addUser method in UserRepository.  Inputs: User object containing user information.  Outputs: Result of adding the user (success/failure).  Internal Processing: Interacts with UserRepository to add the user to the database. |

#### **UserRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | add(User) | Adds a new user to the database.  Inputs: User object containing user information.  Outputs: Boolean indicating if the user was added successfully.  Internal Processing: Executes the query to add the user to the database. |

c. Sequence Diagram



d. Database Queries

Insert Query for New User:

db.users.insertOne({

username: "new\_username",

phone: "new\_phone",

email: "new\_email",

dob: ISODate("YYYY-MM-DD"),

verify\_token: "new\_verify\_token",

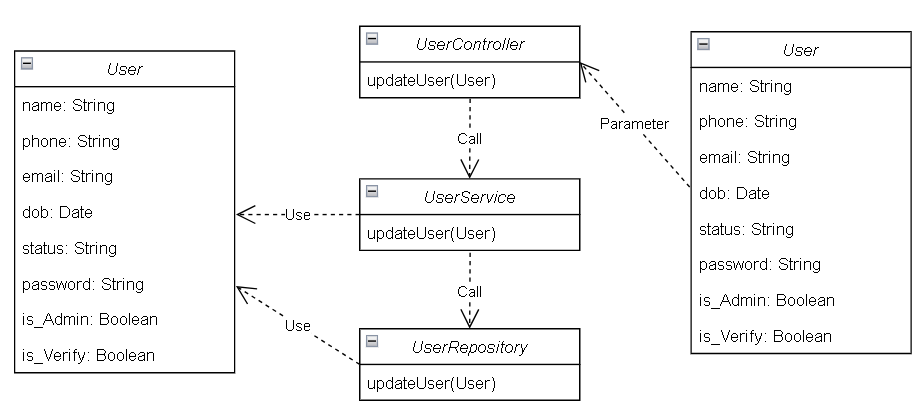
hash\_pass: "new\_hashed\_password",

is\_admin: false

});

**2.3 Update User**

a. Class Diagram



b. Class Specifications

#### **UserController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | update(User) | Handles the request to update user information. It calls the updateUser method in UserService.  Inputs: User object containing updated user information.  Outputs: Result of updating the user (success/failure).  Internal Processing: Passes the User object to UserService for processing. |

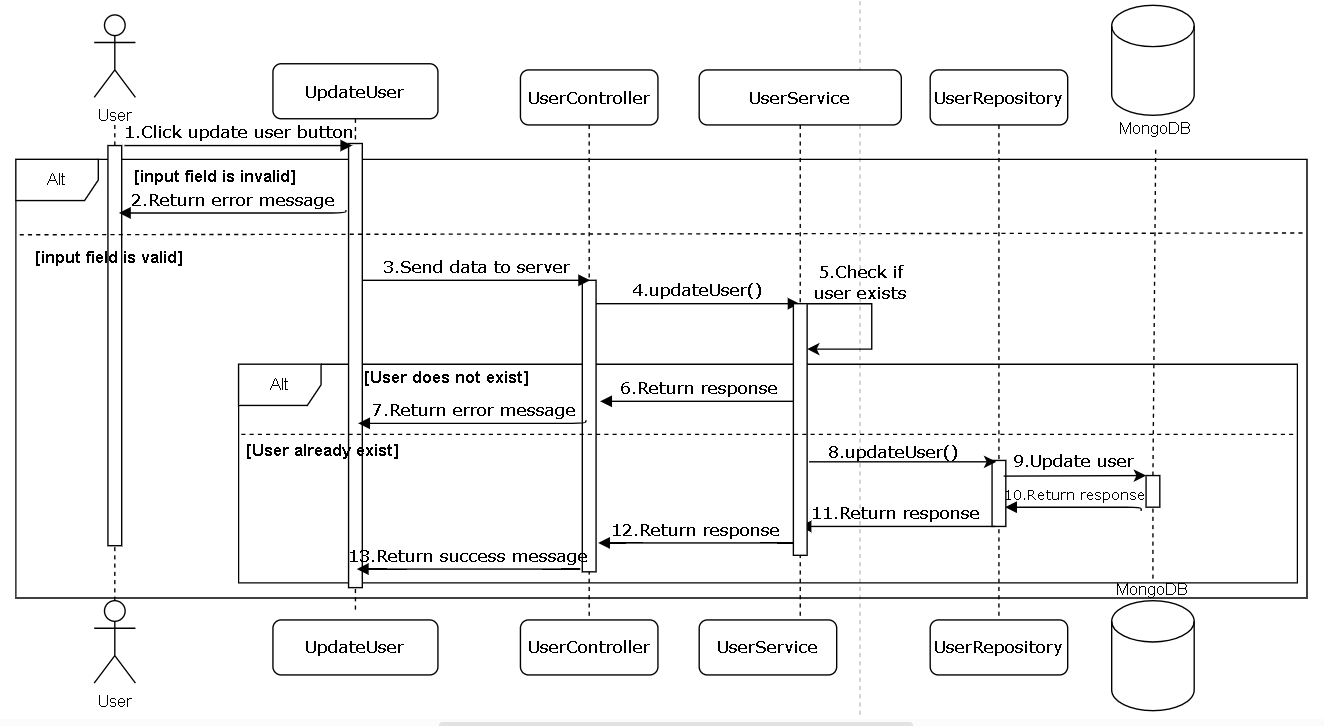
#### **UserService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | update(User) | Processes the request to update user information by calling the updateUser method in UserRepository.  Inputs: User object containing updated user information.  Outputs: Result of updating the user (success/failure).  Internal Processing: Interacts with UserRepository to update the user in the database. |

#### **UserRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | update(User) | Updates the user information in the database.  Inputs: User object containing updated user information.  Outputs: Boolean indicating if the user was updated successfully.  Internal Processing: Executes the query to update the user in the database. |

c. Sequence Diagram



d. Database Queries

Update Query for User Information:

db.users.updateOne(

{ username: "existing\_username" },

{

$set: {

phone: "updated\_phone",

email: "updated\_email",

dob: ISODate("updated\_YYYY-MM-DD"),

verify\_token: "updated\_verify\_token",

hash\_pass: "updated\_hashed\_password",

is\_admin: updated\_is\_admin

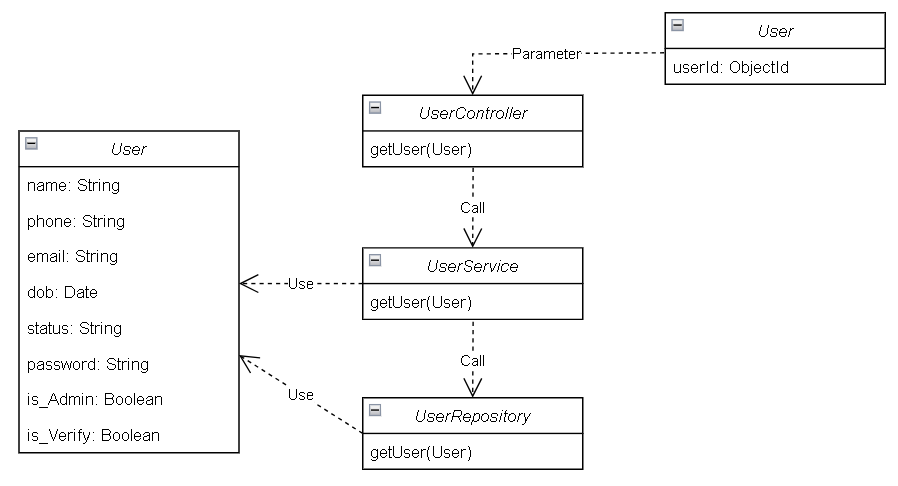
}

}

);

**2.3 Get User Profile**

a. Class Diagram



b. Class Specifications

#### **UserController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getUser(User) | Handles the request to retrieve user profile information. It calls the getUser method in UserService.  Inputs: User object containing user identifier (e.g., userId).  Outputs: User profile information.  Internal Processing: Passes the User object to UserService for processing.  Internal Processing: Passes the User object to UserService for processing. |

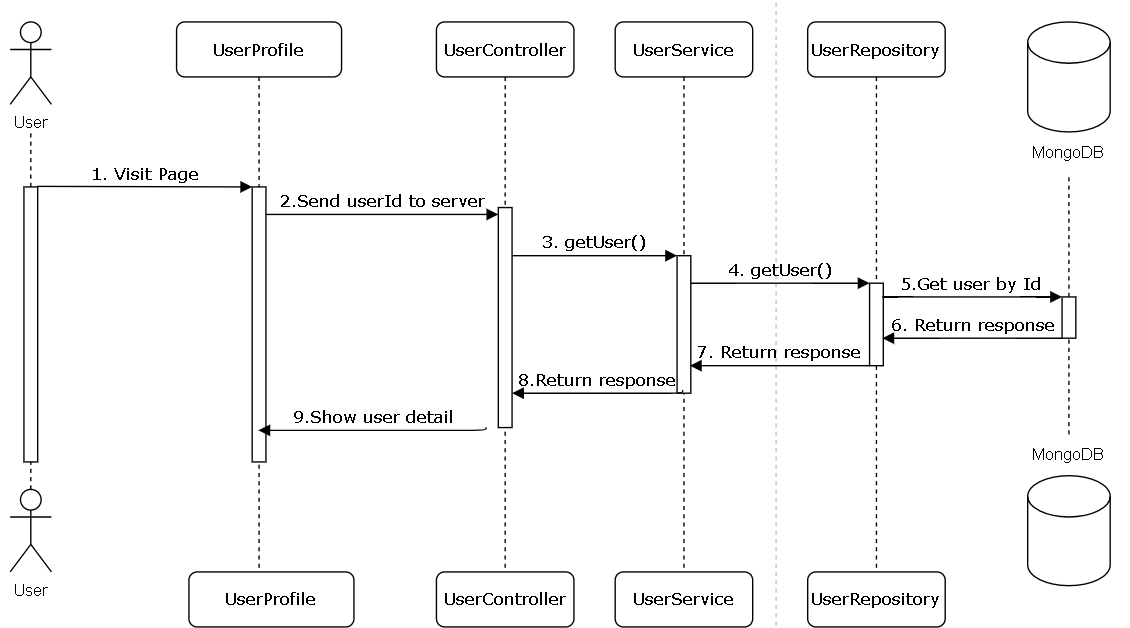
#### **UserService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getUser(User) | Processes the request to retrieve user profile information by calling the getUser method in UserRepository.  Inputs: User object containing user identifier (e.g., userId).  Outputs: User profile information.  Internal Processing: Interacts with UserRepository to fetch the user profile from the database. |

#### **UserRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getUser(User) | Retrieves the user profile information from the database.  Inputs: User object containing user identifier (e.g., userId).  Outputs: User profile information.  Internal Processing: Executes the query to fetch the user profile from the database. |

c. Sequence Diagram



d. Database Queries

Find Query for User Profile:

db.users.findOne({

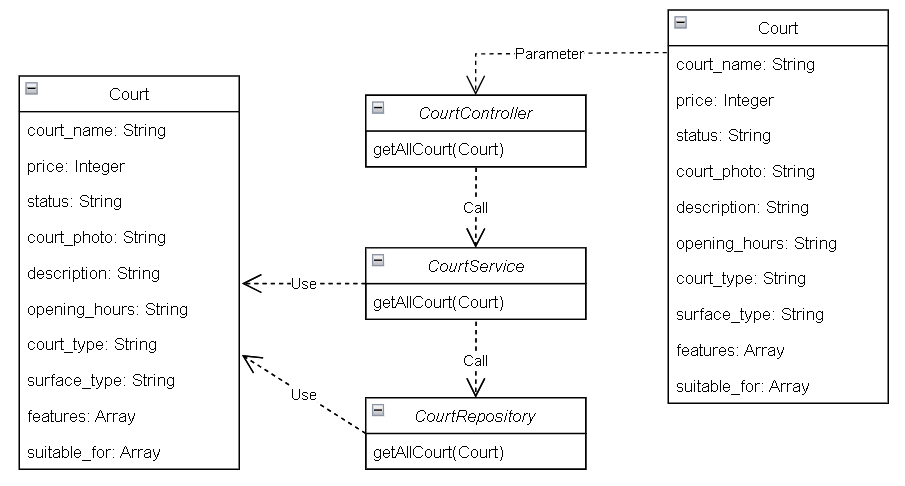
\_id: ObjectId("user\_id")

});

**3 Manage Court**

**3.1 Get Court**

a. Class Diagram



b. Class Specifications

#### **CourtController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllCourt(Court) | Handles the request to retrieve all counts (courts). It calls the getAllCount method in CountService.  Inputs: None (or optional parameters for filtering).  Outputs: List of all counts (courts).  Internal Processing: Passes the request to CountService for processing. |

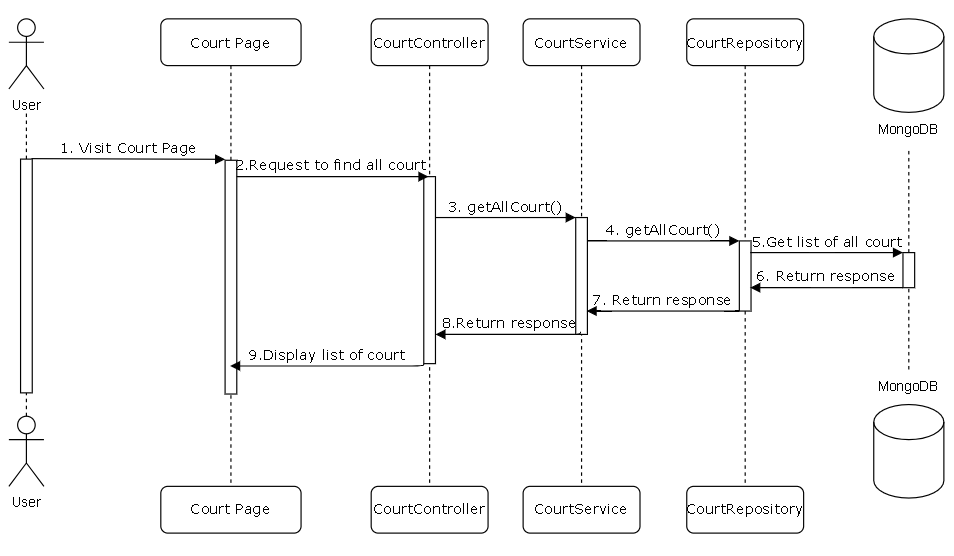
#### **CourtService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllCourt(Court) | Processes the request to retrieve all counts (courts) by calling the getAllCount method in CountRepository.  Inputs: None (or optional parameters for filtering).  Outputs: List of all counts (courts).  Internal Processing: Interacts with CountRepository to fetch all counts from the database. |

#### **CourtRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllCourt(Court) | Retrieves all counts (courts) from the database.  Inputs: None (or optional parameters for filtering).  Outputs: List of all counts (courts).  Internal Processing: Executes the query to fetch all counts from the database. |

c. Sequence Diagram



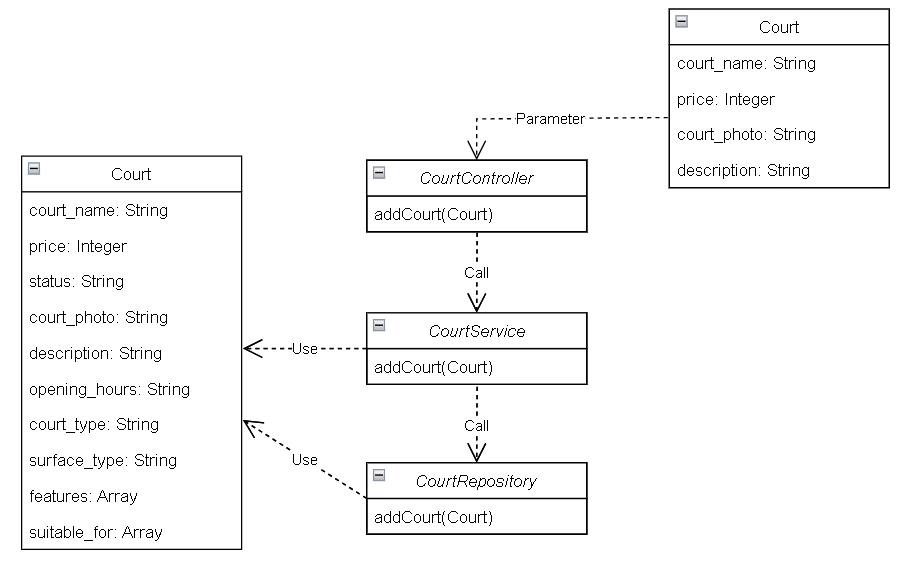
d. Database Queries

Find All Counts Query:

db.counts.find({});

**3.2 Add Court**

a. Class Diagram



b. Class Specifications

#### **CourtController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addCourt(Court) | Handles the request to add a new count (court). It calls the addCount method in CountService.  Inputs: Count object containing count information.  Outputs: Result of adding the count (success/failure).  Internal Processing: Passes the Count object to CountService for processing. |

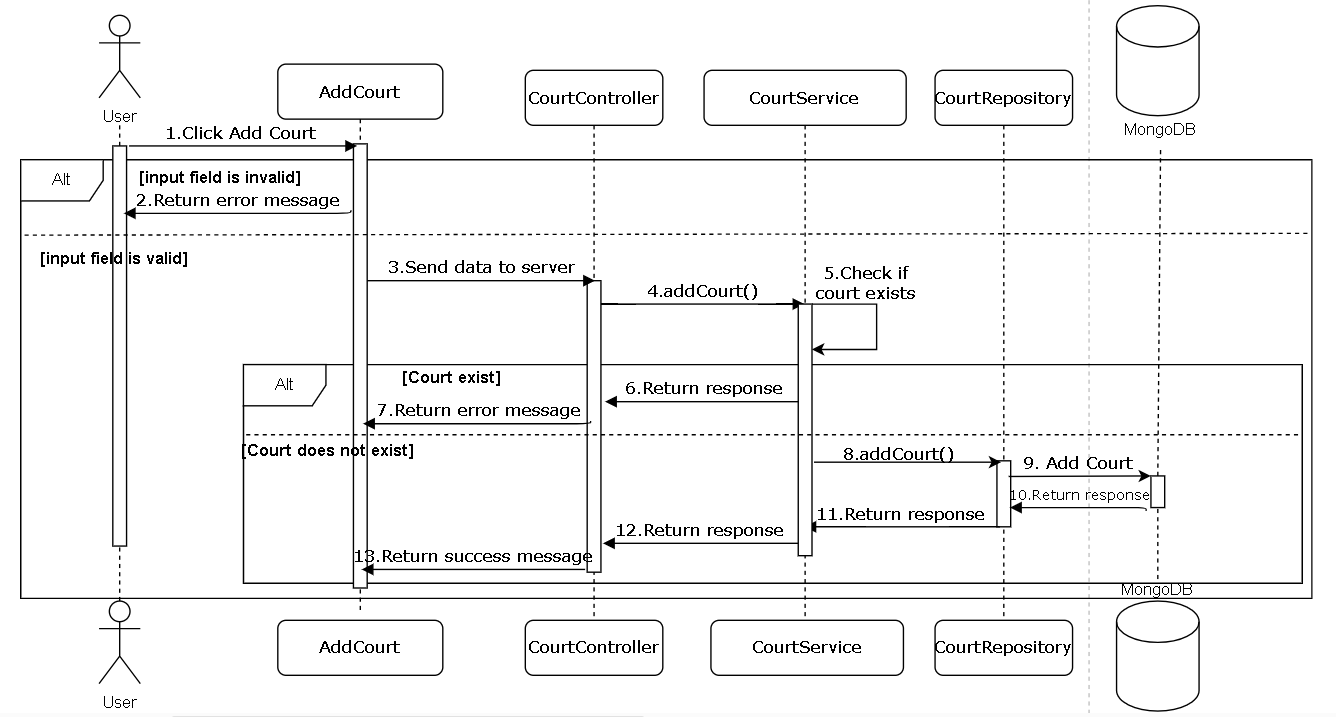
#### **CourtService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addCourt(Court) | Processes the request to add a new count by calling the addCount method in CountRepository.  Inputs: Count object containing count information.  Outputs: Result of adding the count (success/failure).  Internal Processing: Interacts with CountRepository to add the count to the database. |

#### **CourtRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addCourt(Court) | Adds a new count to the database.  Inputs: Count object containing count information.  Outputs: Boolean indicating if the count was added successfully.  Internal Processing: Executes the query to add the count to the database. |

c. Sequence Diagram



d. Database Queries

Insert Query for New Count:

db.counts.insertOne({

Count\_name: "new\_count\_name",

price: new\_price,

status: "new\_status",

count\_photo: "new\_count\_photo",

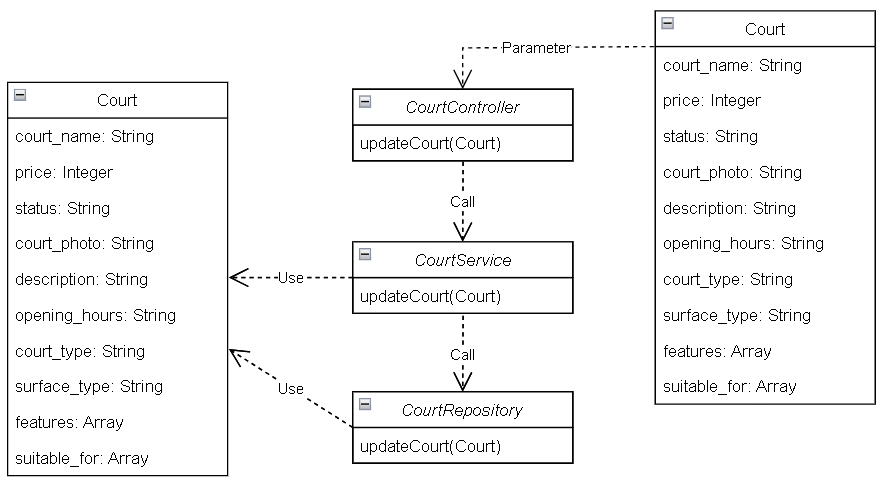
description: "new\_description",

opening\_hours: "new\_opening\_hours"

});

**3.3 Update Court**

a. Class Diagram



b. Class Specifications

#### **CourtController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | updateCourt(Court) | Handles the request to update count information. It calls the updateCount method in CountService.  Inputs: Count object containing updated count information.  Outputs: Result of updating the count (success/failure).  Internal Processing: Passes the Count object to CountService for processing. |

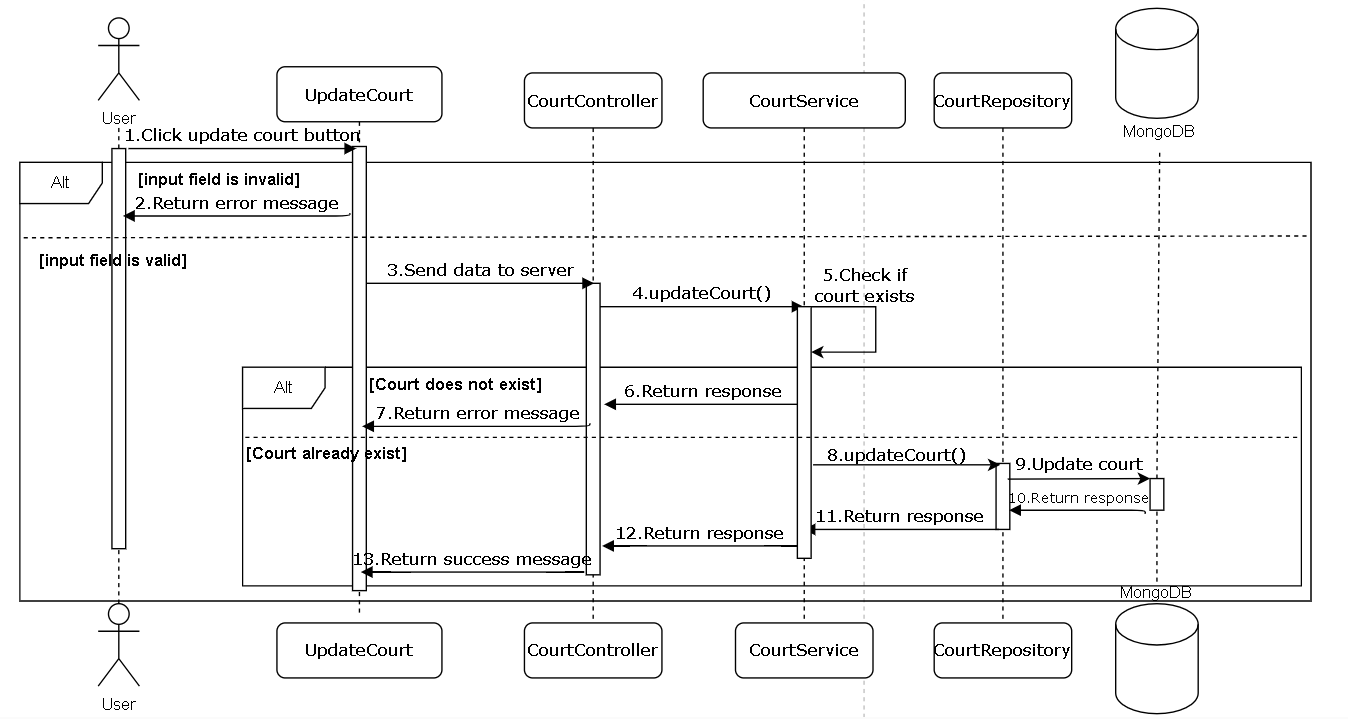
#### **CourtService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | updateCourt(Court) | Processes the request to update count information by calling the updateCount method in CountRepository.  Inputs: Count object containing updated count information.  Outputs: Result of updating the count (success/failure).  Internal Processing: Interacts with CountRepository to update the count in the database. |

#### **CourtRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | updateCourt(Court) | Updates the count information in the database.  Inputs: Count object containing updated count information.  Outputs: Boolean indicating if the count was updated successfully.  Internal Processing: Executes the query to update the count in the database. |

c. Sequence Diagram



d. Database Queries

Update Query for Count Information:

db.counts.updateOne(

{ count\_name: "existing\_count\_name" },

{

$set: {

price: updated\_price,

status: "updated\_status",

count\_photo: "updated\_count\_photo",

description: "updated\_description",

opening\_hours: "updated\_opening\_hours"

}

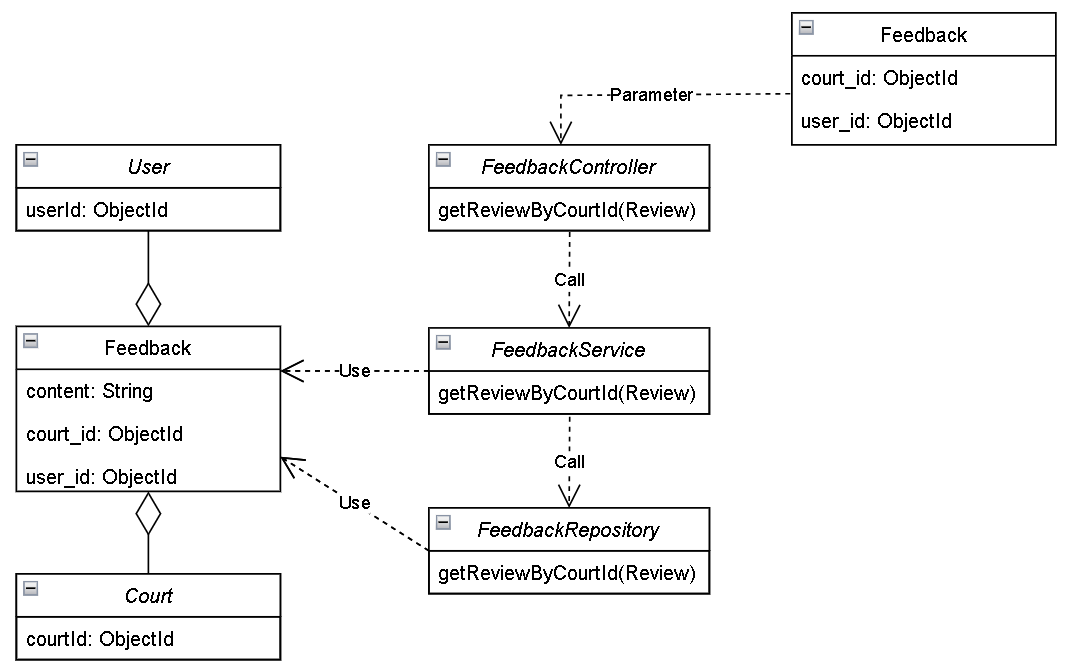
}

);

**4 Manage Feedback**

**4.1 Get Feedback**

a. Class Diagram



b. Class Specifications

#### **FeedbackController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getReviewByCourtId(Review) | Handles the request to retrieve reviews by court ID. It calls the getReviewByCourtId method in FeedbackService.  Inputs: Review object containing court ID.  Outputs: List of reviews for the specified court.  Internal Processing: Passes the Review object to FeedbackService for processing. |

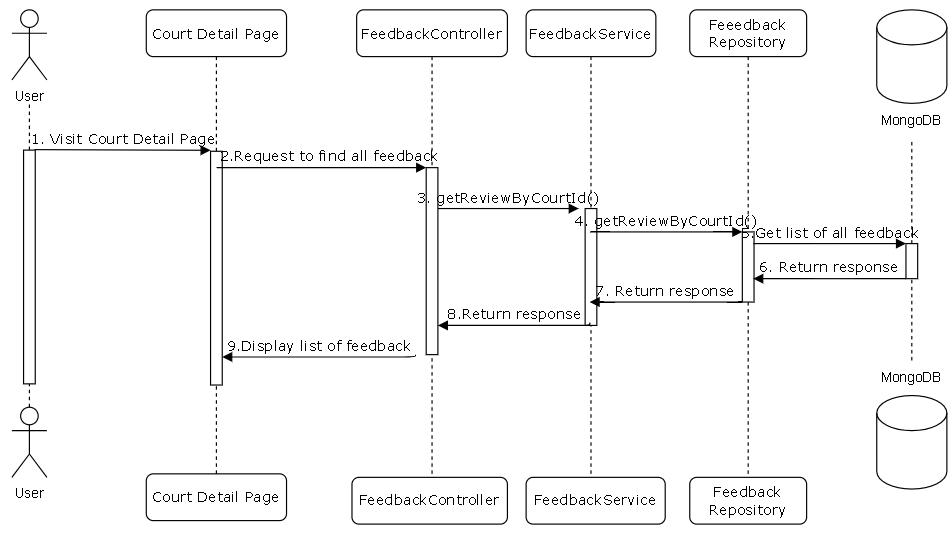
#### **FeedbackService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getReviewByCourtId(Review) | Processes the request to retrieve reviews by court ID by calling the getReviewByCourtId method in FeedbackRepository.  Inputs: Review object containing court ID.  Outputs: List of reviews for the specified court.  Internal Processing: Interacts with FeedbackRepository to fetch the reviews from the database. |

#### **FeedbackRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getReviewByCourtId(Review) | Retrieves reviews from the database based on the court ID.  Inputs: Review object containing court ID.  Outputs: List of reviews for the specified court.  Internal Processing: Executes the query to fetch the reviews from the database. |

c. Sequence Diagram



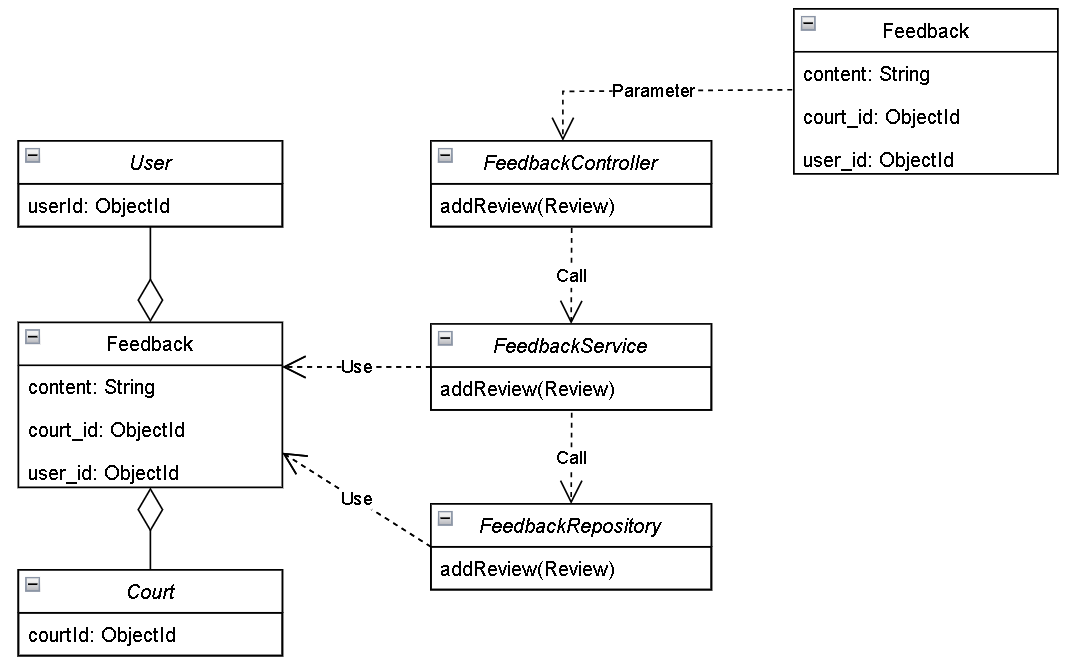
d. Database Queries

Find Reviews by Court ID Query:

db.feedbacks.find({ count\_id: ObjectId("court\_id") });

**4.2 Add Feedback**

a. Class Diagram



b. Class Specifications

#### **FeedbackController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addReview(Review) | Handles the request to add a new review. It calls the addReview method in FeedbackService.  Inputs: Review object containing review content, court ID, and user ID.  Outputs: Result of adding the review (success/failure).  Internal Processing: Passes the Review object to FeedbackService for processing. |

#### **FeedbackService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addReview(Review) | Processes the request to add a new review by calling the addReview method in FeedbackRepository.  Inputs: Review object containing review content, court ID, and user ID.  Outputs: Result of adding the review (success/failure).  Internal Processing: Interacts with FeedbackRepository to add the review to the database. |

#### **FeedbackRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addReview(Review) | Adds a new review to the database.  Inputs: Review object containing review content, court ID, and user ID.  Outputs: Boolean indicating if the review was added successfully.  Internal Processing: Executes the query to add the review to the database. |

c. Sequence Diagram

d. Database Queries

Insert Query for New Review:

db.feedbacks.insertOne({

content: "review\_content",

count\_id: ObjectId("court\_id"),

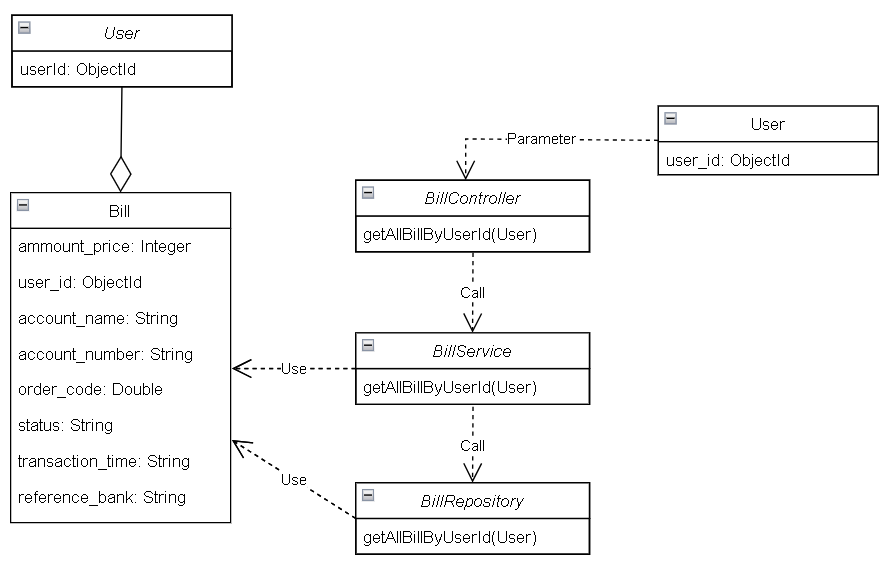
user\_id: ObjectId("user\_id")

});

**5 Manage Bill**

**5.1 Get Bill**

a. Class Diagram



b. Class Specifications

#### **BillController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllBillByUserId (User) | Handles the request to retrieve all bills by user ID. It calls the getAllBillByUserId method in BillService.  Inputs: User object containing user ID.  Outputs: List of bills for the specified user.  Internal Processing: Passes the User object to BillService for processing. |

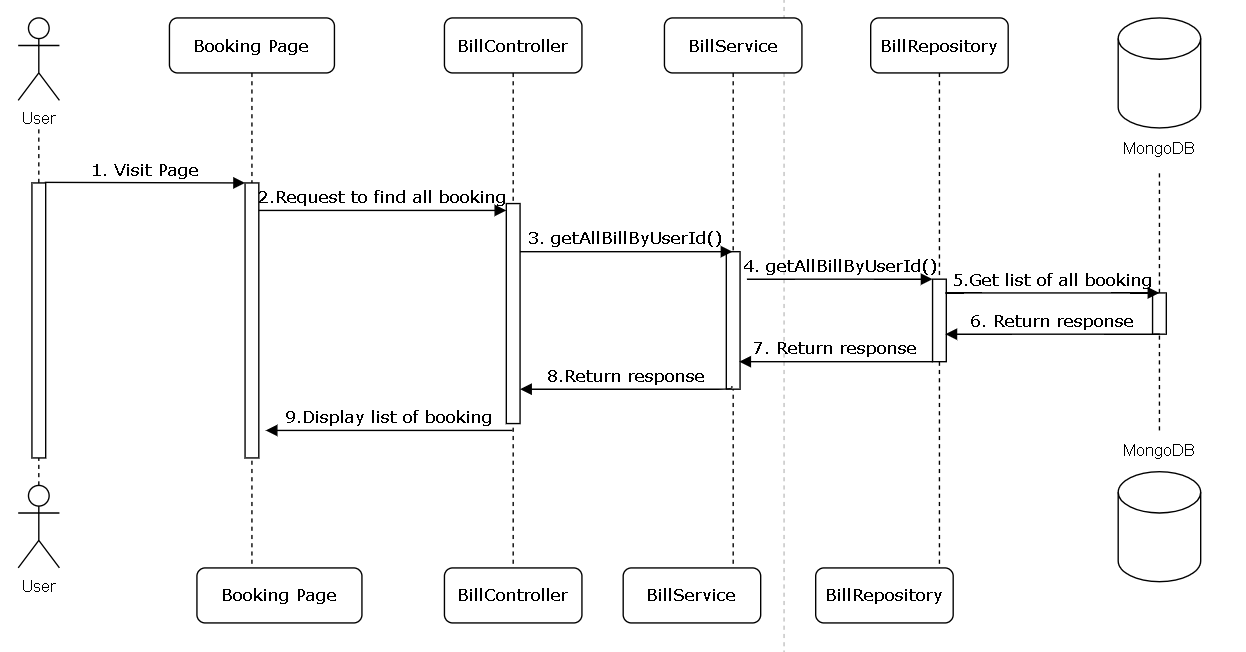
#### **BillService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllBillByUserId (User) | Processes the request to retrieve all bills by user ID by calling the getAllBillByUserId method in BillRepository.  Inputs: User object containing user ID.  Outputs: List of bills for the specified user.  Internal Processing: Interacts with BillRepository to fetch the bills from the database. |

#### **BillRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | getAllBillByUserId (User) | Retrieves bills from the database based on the user ID.  Inputs: User object containing user ID.  Outputs: List of bills for the specified user.  Internal Processing: Executes the query to fetch the bills from the database. |

c. Sequence Diagram



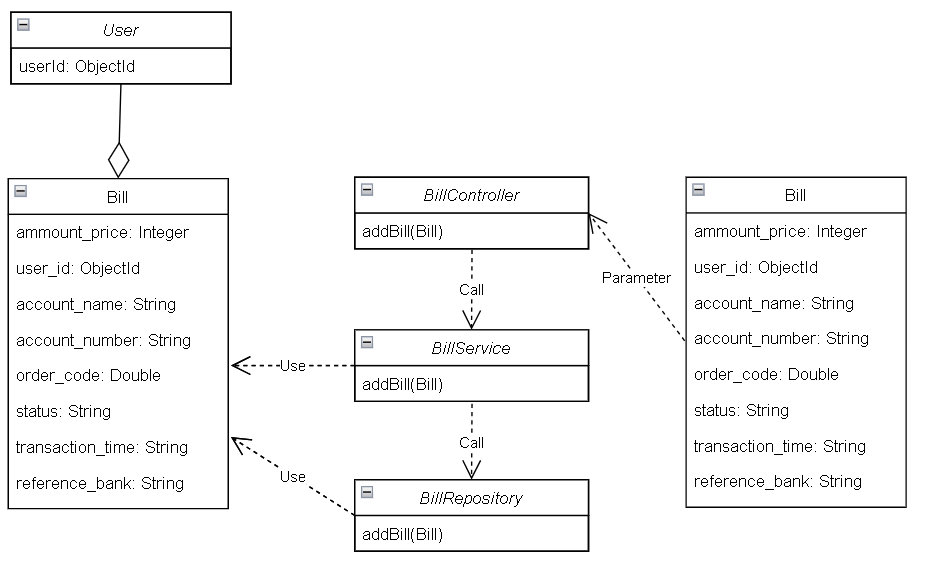
d. Database Queries

Find Bills by User ID Query:

db.bills.find({ user\_id: ObjectId("user\_id") });

**5.2 Add Bill**

a. Class Diagram



b. Class Specifications

#### **BillController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addBill(Bill) | Handles the request to add a new bill. It calls the addBill method in BillService.  Inputs: Bill object containing bill details.  Outputs: Result of adding the bill (success/failure).  Internal Processing: Passes the Bill object to BillService for processing. |

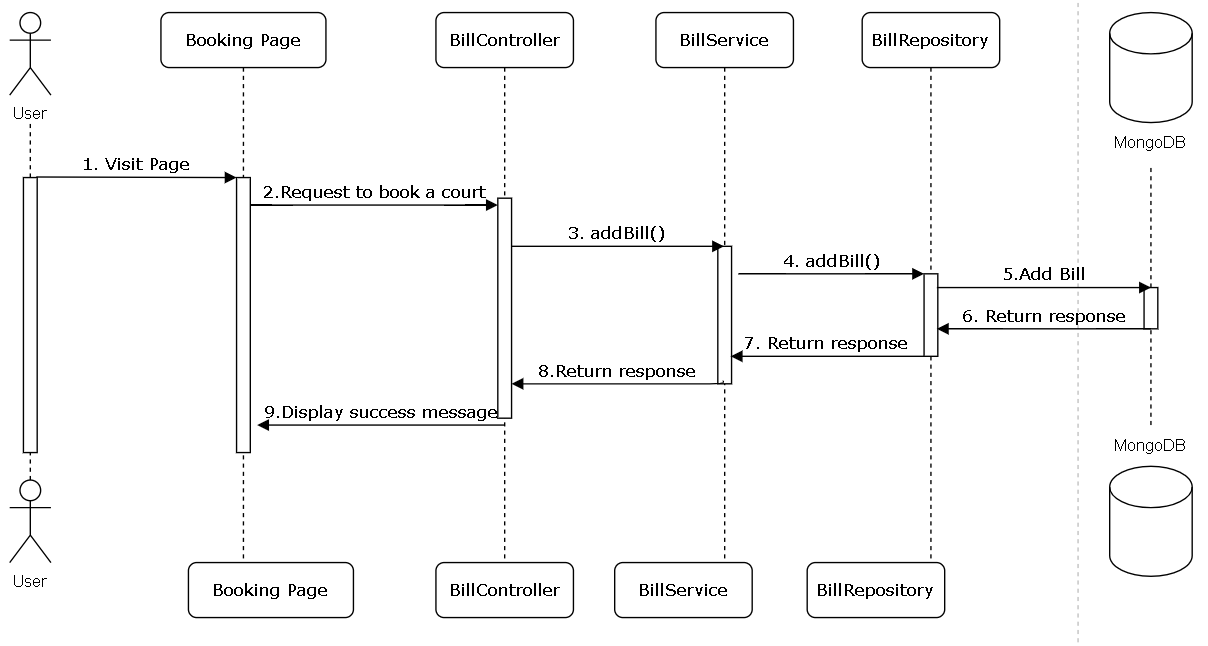
#### **BillService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addBill(Bill) | Processes the request to add a new bill by calling the addBill method in BillRepository.  Inputs: Bill object containing bill details.  Outputs: Result of adding the bill (success/failure).  Internal Processing: Interacts with BillRepository to add the bill to the database. |

#### **BillRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | addBill(Bill) | Adds a new bill to the database.  Inputs: Bill object containing bill details.  Outputs: Boolean indicating if the bill was added successfully.  Internal Processing: Executes the query to add the bill to the database. |

c. Sequence Diagram



d. Database Queries

Insert Query for New Bill:

db.bills.insertOne({

rental\_price: rental\_price\_value,

time\_rental: "time\_rental\_value",

end\_time\_rental: "end\_time\_rental\_value",

account\_name: "account\_name\_value",

account\_number: "account\_number\_value",

order\_code: order\_code\_value,

status: "status\_value",

count\_id: ObjectId("count\_id\_value"),

user\_id: ObjectId("user\_id\_value"),

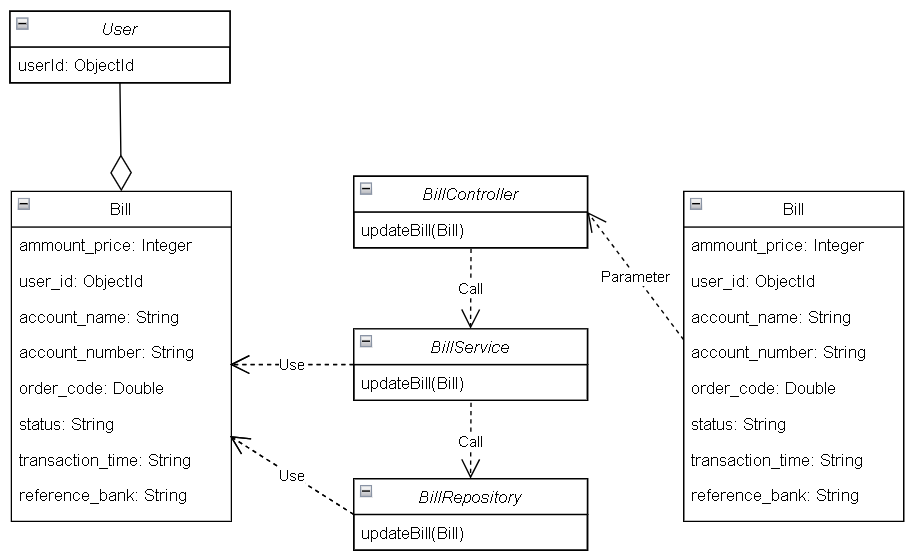
transaction\_time: "transaction\_time\_value",

reference\_bank: "reference\_bank\_value"

});

**5.3 Update Bill**

a. Class Diagram



b. Class Specifications

#### 

#### **BillController Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | updateBill(Bill) | Handles the request to update a bill. It calls the updateBill method in BillService.  Inputs: Bill object containing updated bill details.  Outputs: Result of updating the bill (success/failure).  Internal Processing: Passes the Bill object to BillService for processing. |

#### 

#### 

#### 

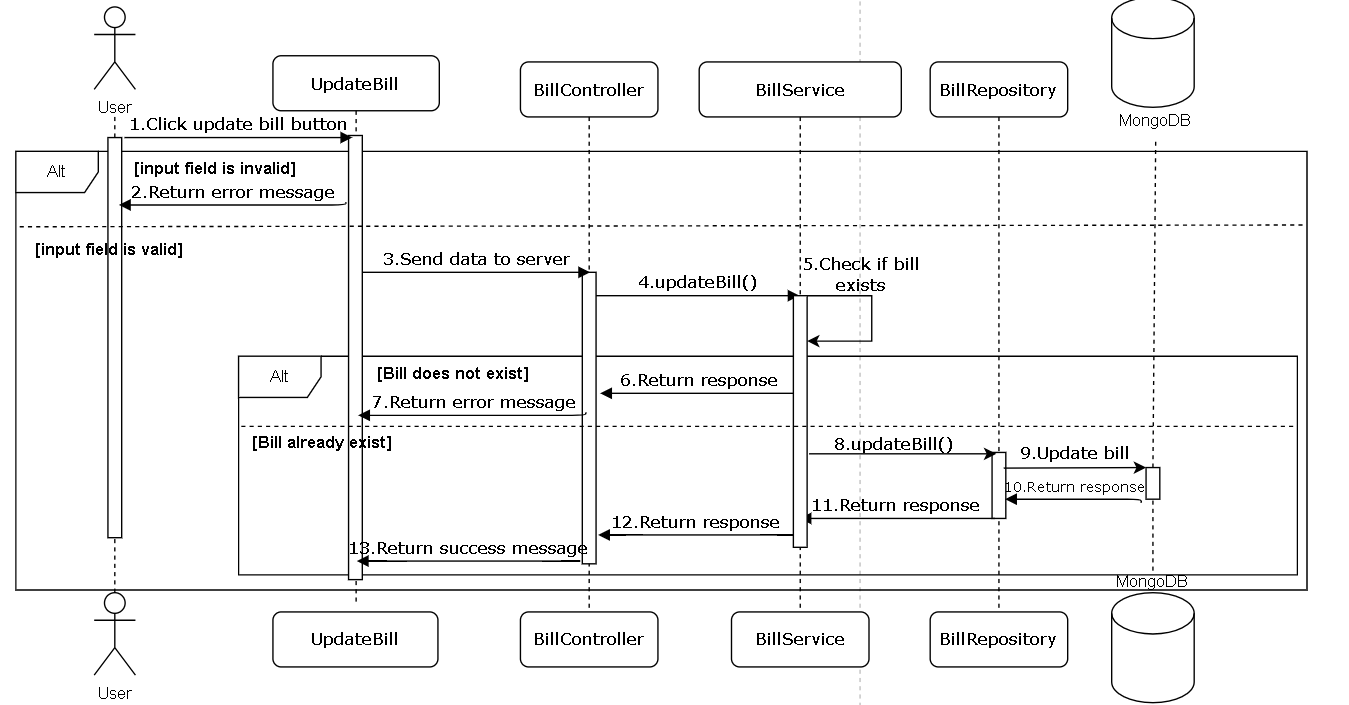
#### **BillService Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | updateBill(Bill) | Processes the request to update a bill by calling the updateBill method in BillRepository.  Inputs: Bill object containing updated bill details.  Outputs: Result of updating the bill (success/failure).  Internal Processing: Interacts with BillRepository to update the bill in the database. |

#### **BillRepository Class**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | updateBill(Bill) | Updates the bill information in the database.  Inputs: Bill object containing updated bill details.  Outputs: Boolean indicating if the bill was updated successfully.  Internal Processing: Executes the query to update the bill in the database. |

c. Sequence Diagram



d. Database Queries

Update Query for Bill Information:

db.bills.updateOne(

{ \_id: ObjectId("bill\_id") },

{

$set: {

rental\_price: updated\_rental\_price,

time\_rental: "updated\_time\_rental",

end\_time\_rental: "updated\_end\_time\_rental",

account\_name: "updated\_account\_name",

account\_number: "updated\_account\_number",

order\_code: updated\_order\_code,

status: "updated\_status",

court\_id: ObjectId("updated\_court\_id"),

user\_id: ObjectId("updated\_user\_id"),

transaction\_time: "updated\_transaction\_time",

reference\_bank: "updated\_reference\_bank"

}

}

);

# 

# **V. Software Testing Documentation**

#### **1. Scope of Testing**

**System Under Test:** Pickleball Court Rental System

**Scope of Testing:** This testing scope encompasses all core functionalities of the pickleball court rental system, including court booking, cancellation, payment processing, and user account management. The testing will verify that both functional and non-functional requirements are met, with a focus on user experience, system reliability, and transaction accuracy.

**Functional Features to be Tested:**

* Court booking process (e.g., availability check, time slot selection, booking confirmation)
* Booking cancellation and refund processing
* User account creation and management (e.g., profile updates, rental history)
* Payment processing for court rentals and subscription plans (if applicable)
* Notifications and reminders for upcoming bookings or payment deadlines
* Viewing rental history and receipts

**Non-Functional Requirements:**

* **Performance:** Ensure quick response times during peak booking periods.
* **Security:** Protect user data and payment information with encryption and access controls.
* **Usability:** Provide an intuitive interface for booking and managing rentals.
* **Reliability:** Maintain consistent availability and accurate booking records.

**Stages/Levels of Testing:**

* **Unit Testing:**
  + **In-charge:** Development team
  + **Focus:** Verify individual components, such as court availability checks, payment calculations, and notification triggers.
  + **Input/Time:** Unit test data including court schedules, user profiles, and payment scenarios.
  + **Acceptance Criteria:** Each component functions correctly under normal and edge cases with no errors.
* **Integration Testing:**
  + **In-charge:** Development and QA team
  + **Focus:** Ensure seamless interaction between components, such as the booking system, payment gateway, and notification module.
  + **Input/Time:** Simulated user actions like booking a court and receiving a confirmation notification.
  + **Acceptance Criteria:** All integrated processes execute smoothly, with accurate data flow between modules.
* **System Testing:**
  + **In-charge:** QA team
  + **Focus:** Test the complete workflow, including booking a court, making payments, canceling bookings, and receiving notifications.
  + **Input/Time:** Realistic scenarios with various user types (e.g., casual renters, admins) and booking patterns.
  + **Acceptance Criteria:** The system meets all functional requirements, delivering a flawless end-to-end rental experience.

**Constraints and Assumptions:**

* **Constraints:**
  + Testing conducted in a simulated environment with limited real-time court availability data.
  + Restricted to predefined user roles (renter and admin).
* **Assumptions:**
  + Stable internet connectivity during performance testing.
  + Mock court schedules and user data are sufficient for comprehensive testing.

#### **2. Test Strategy**

##### **2.1 Testing Types**

* **Unit Testing**
  + **Objective:** Validate the accuracy of individual components, such as court slot availability logic, payment processing, and notification generation.
  + **Technique:** Automated unit tests with assertions to check input-output behavior, error handling, and boundary conditions.
  + **Completion Criteria:** All unit tests pass, achieving at least 90% coverage of core functionalities like booking limits and payment calculations.
* **Integration Testing**
  + **Objective:** Confirm that interconnected modules (e.g., booking system, payment processing, and notifications) work cohesively without data discrepancies.
  + **Technique:** Simulate user actions like booking a court and processing a payment to test module interactions.
  + **Completion Criteria:** All integrated workflows complete successfully with no data loss or errors.
* **System Testing**
  + **Objective:** Ensure the entire system meets functional and non-functional requirements under real-world conditions.
  + **Technique:** Execute end-to-end test cases, such as booking a court, canceling it, and receiving a refund, across various user scenarios.
  + **Completion Criteria:** All features perform as expected, meeting performance, security, and usability standards.
* **Performance Testing**
  + **Objective:** Verify system efficiency under normal and high-demand booking scenarios.
  + **Technique:** Perform load tests with multiple simultaneous users and stress tests to determine breaking points.
  + **Completion Criteria:** Response times remain below 2 seconds for key actions under normal load, with no degradation during peak usage.
* **User Acceptance Testing (UAT)**
  + **Objective:** Confirm the system meets user expectations and supports real-life rental needs.
  + **Technique:** Conduct testing with end-users (e.g., pickleball players and court admins) performing tasks like booking and managing rentals.
  + **Completion Criteria:** Users report satisfactory experiences, and the system is deemed ready for deployment.
* **Regression Testing**
  + **Objective:** Ensure updates or fixes do not disrupt existing functionalities.
  + **Technique:** Run automated regression tests on critical features (e.g., booking and payment) after changes.
  + **Completion Criteria:** All affected areas pass tests without introducing new issues.

##### **2.2 Test Levels**

* **Unit Testing Level**
  + **Description:** Tests individual functions or methods, such as availability checks or payment validations.
  + **Test Types:** Unit Testing
* **Integration Testing Level**
  + **Description:** Validates interactions between combined modules, such as booking and notification systems.
  + **Test Types:** Integration Testing
* **System Testing Level**
  + **Description:** Ensures the fully integrated system meets all specified requirements.
  + **Test Types:** System Testing, Performance Testing
* **Acceptance Testing Level**
  + **Description:** Confirms the system is user-ready and meets operational needs.
  + **Test Types:** User Acceptance Testing, Regression Testing (as needed)

# **VI. Release Package & User Guides**

## **1. Deliverable Package**

Figure VI.1: Description of deliverables

| **No.** | **Deliverable Item** | **Description** |
| --- | --- | --- |
| 1 | Source Codes | Facility-Management-System.zip |
| 2 | Database Script(s) | Data.zip |
| 3 | SRS | Software Requirement Specification.docx |
| 4 | SDS | Software Design Document.docx |
| 5 | Final Report | Final Project Report.docx |
| 6 | Unit Test | G3\_Unit\_Test Case.xlsx |

*Table VI.1: Deliverable package*

## **2. Installation Guides**

### **2.1 System Requirements**

Below are system requirements necessary to support the web application and help us to deploy the code easily:

- CPU: 2.5GHZ

- RAM: 8GB

- Minimum disk space: 10GB

- Network: Internet connection

### **2.2 Hosting requirement**

1. Extract Facility-Management-System.zip

2. Open folder Client and Server in VS Code

3. Open terminal and run “npm i” for both folder

4. Extract Data.zip

5. Open MongoDB and import json file into Collection

6. Ìn folder Server, run “npm start”

7. Ìn folder Client. run “npm run dev”

8. Open your browse with link “http://localhost:3000/”

## **3. User Manual**

### **3.1 Overview**

- The Facilities Management System was born based on the need of facilities management in FPT Education. FPT Education is very large with 10 campuses and nearly 150,000 students. Facilities in FPT are used and booked everyday which is really hard to manage and follow by using paper. With high demands to manage facilities more effectively, the campuses within the FPT education system require an efficient management system to support their daily operations.

### **3.2 Role User**

- Access the home page.

- Users can login by clicking on **“Đăng nhập”** at the header.

- Users can login to the system by clicking on **“Đăng nhập”**, users will go to the login page and login to the system by google account with FPT email.

- After login, user will be navigated to homepage

- In the home page, users will see the list of categories, top 8 of facilities

- When click in a category, it will facilities list of the category

- When you click on a specific facilities, you will be navigated to the facilities detail page

- Click in “Đặt phòng” to create a booking request for a facilities. After click in “Đặt phòng”, a schedule will be displayed with different status and notation for each status. Then click “Đặt” to create booking request for specific slot of facility (notice that you can book the slot before current time at least 2 hours and you can not booking slot in the pass)

- Click the avatar in the header, you can see a drop-down list. Select “Lịch sử đặt phòng” to view all of booking request and their status. You can search facilities by name in here

- You can see notification about your booking request when you click in bell icon in the header

- In homepage, you can click the icon to open chat box and send your message to admin

- After you use a facility, you can evaluate for that facility by voting and comment in facility detail page

- Click the avatar in the header and choose “Hồ sơ của tôi” to open your Profile and update if you want. You can choose “Đăng xuất” also to log out

### **3.3 Admin Role**

- Users login to the system with an account with the role **“Admin”**. Users can see the Dashboard for **“Admin”**.

- Click “Quản lý phòng, sân bóng” to view facilities list

- From facilities list page:

+, Click in + button to create a new facilities

+, Click in “Cập nhật” to update facility

+,Click in “Xem” to view record of change of facility

+, Change Active/Inactive to change status of facility

- Similar to Category List

- Click in “Duyệt yêu cầu đặt sân, phòng học” to view booking request list

Then admin can choose “Chấp nhận” to approve or “Hủy” to disapprove with reason

- “Các yêu cầu được duyệt” display all accepted request

- “Các yêu cầu không được duyệt” display all rejected request

- “Các yêu cầu quá hạn” display all overdue request

- “Quản lý tài khoản” displays an account list in the system. Admin can also change account status Active/Inactive of Student only

- Click the message icon in the header to open the chat screen. You can chat with user in this page