StadiumPro-stadium reservation management system-test

V4.0

Design Document

Catalogue

[1. Introduction 4](#_Toc134328403)

[2. Composition Purpose 4](#_Toc134328404)

[3. Project Risk 5](#_Toc134328405)

[3.1 Task originator (sports centre): 5](#_Toc134328406)

[3.2 Software developer: 5](#_Toc134328407)

[3.3 Product users: 7](#_Toc134328408)

[4. Expected users and reading suggestions 7](#_Toc134328409)

[4.1 Customers: 7](#_Toc134328410)

[4.2 Software developers: 7](#_Toc134328411)

[4.3 Project managers: 8](#_Toc134328412)

[4.4 Sales: 9](#_Toc134328413)

[4.5 Testers: 9](#_Toc134328414)

[4.6 Document writers: 9](#_Toc134328415)

[5. Design Brief 10](#_Toc134328416)

[6. Limitations and Constraints 10](#_Toc134328417)

[6.1 Technique: 10](#_Toc134328418)

[6.2 Budget: 11](#_Toc134328419)

[6.3 Developing environment: 11](#_Toc134328420)

[6.4 Time: 12](#_Toc134328421)

[6.5 Policy: 12](#_Toc134328422)

[7. Design principles and requirements 13](#_Toc134328423)

[7.1 Name principle: 13](#_Toc134328424)

[7.2 Module independence principle: 13](#_Toc134328425)

[7.3 Boundary design principle: 14](#_Toc134328426)

[7.4 Database design principle: 14](#_Toc134328427)

[7.5 Necessary safety measures: 15](#_Toc134328428)

[7.6 Security and confidentiality principles: 15](#_Toc134328429)

[7.7 System flexibility requirement: 16](#_Toc134328430)

[7.8 System operability requirements: 16](#_Toc134328431)

[7.9 System maintainability requirements: 16](#_Toc134328432)

[8. UI Design 17](#_Toc134328433)

[8.1 Design style and color scheme 17](#_Toc134328434)

[8.2 Page design 17](#_Toc134328435)

[1. Login and regisiter 17](#_Toc134328436)

[2. Homepage 18](#_Toc134328437)

[3. Facility page 18](#_Toc134328438)

[4. Timetablepage 19](#_Toc134328439)

[5. User homepage 19](#_Toc134328440)

[6. Employee homepage 20](#_Toc134328441)

[7. Manager homepage 20](#_Toc134328442)

[9. Module Design 20](#_Toc134328443)

[9.1 User 20](#_Toc134328444)

[9.2 Employee 24](#_Toc134328445)

[9.3 manager 26](#_Toc134328446)

[10. Interface design 28](#_Toc134328447)

[10.1 ActivityService/ AppointmentService/ MembershipService/ OrderService/ StadiumService/ SysUserService/ TimetableService 28](#_Toc134328448)

[8. Page 28](#_Toc134328449)

[9. listall 28](#_Toc134328450)

[10. getActivityById 28](#_Toc134328451)

[11. deleteActivityById 29](#_Toc134328452)

[12. addActivityById 29](#_Toc134328453)

[13. updateActivityById 30](#_Toc134328454)

[10.2 MembershipService 30](#_Toc134328455)

[14. getMemberByUserName 30](#_Toc134328456)

[10.3 OrderService 30](#_Toc134328457)

[15. staticsPrice 31](#_Toc134328458)

[16. staticsStadium 31](#_Toc134328459)

[17. sendPdfToEmail 31](#_Toc134328460)

[18. confirmOrder 31](#_Toc134328461)

[10.4 TimetableService 32](#_Toc134328462)

[19. getTimeTableByActivityId 32](#_Toc134328463)

[10.5 SysUserFaceController 32](#_Toc134328464)

[20. login 32](#_Toc134328465)

[21. captcha 33](#_Toc134328466)

[22. roles 33](#_Toc134328467)

[23. currentUser 33](#_Toc134328468)

[24. logout 34](#_Toc134328469)

[25. register 34](#_Toc134328470)

[10.6 FilesServiceFaceController 34](#_Toc134328471)

[26. upload 34](#_Toc134328472)

[27. uploadSeccurity 35](#_Toc134328473)

[28. download 35](#_Toc134328474)

[29. delete 35](#_Toc134328475)

[11. System error handling design 35](#_Toc134328476)

[11.1 Error classification: 36](#_Toc134328477)

[11.2 Error handling process: 36](#_Toc134328478)

[11.3 Error prompt design: 37](#_Toc134328479)

[11.4 Logging: 37](#_Toc134328480)

[12. Database design 38](#_Toc134328481)

[12.1 Introduction 38](#_Toc134328482)

[12.2 Database name principle 38](#_Toc134328483)

[12.3 Database design principle 38](#_Toc134328484)

[12.4 Database logical design 39](#_Toc134328485)

[12.5 Database physical design 40](#_Toc134328486)

[1. DBMS system selection: Mysql 40](#_Toc134328487)

[2. Storage structure design 41](#_Toc134328488)

[3. Database distribution 42](#_Toc134328489)

[4. Table design 42](#_Toc134328490)

# Introduction

The introduction is an overview of the software system overview design report. It is intended to help the reader understand how the document was written and how it should be read, understood, and interpreted.

# Composition Purpose

This is a software system for the management of a sports centre. It is made for the digital transformation of the sports centre, where involves functions towards customers as well as employees and managers. This system will promote the operation efficiency of the sports centre thus benefiting its management and profit. Through this software system design report detailed description of the software structure of the software product, including database structure and error handling are presented, so as to describe the structure of the software product.

# Project Risk

In the system development process, several risk might appear to the corresponding risk bearers, including:

## 3.1 Task originator (sports centre):

* Cost risk: The risk of overspend in project expense (including human cost, project management cost, maintaining cost, hardware cost, etc)
* Scope risk: The risk of the change in the scale of the product, which is derived mainly by 1) measures in scope estimation 2) confidence in product size estimates 3) number of users 4) potential change in requirements
* Budget overruns: If the project exceeds the allocated budget, it can impact the profitability of the task originator.
* Delays in project delivery: If the project is delayed, it can impact the ability of the task originator to meet customer needs and expectations.
* Quality issues: If the software does not meet quality standards, it can impact the reputation of the task originator and lead to customer dissatisfaction.
* Scope creep: If the project scope expands beyond the initial requirements, it can impact the timeline and budget of the project and lead to customer dissatisfaction.
* Inadequate resources: If the development team lacks the necessary resources, it can impact the success of the project and the reputation of the task originator.

## 3.2 Software developer:

* Schedule risk: Schedule risk is a potential threat that can impact the timely completion of a software development project. It refers to the uncertainty in the timeline of the project, which can cause delays in the delivery of the final product. There are several factors that can contribute to schedule risk in software development, including: 1) Requirements changes: Changes in project requirements during the development process can impact the timeline of the project, especially if the changes are significant. 2) Technical complexity: Technical challenges such as integrating with legacy systems, dealing with data migration, or developing complex algorithms can cause delays in the project. 3) Resource availability: Availability of resources, including development teams, hardware, and software, can impact the timeline of the project. 4) Dependencies: Delays in deliverables from vendors, partners, or other teams can cause delays in the project. 5) Testing and quality assurance: Testing and quality assurance can take longer than expected, especially if there are issues that need to be resolved.
* Quality risk: Quality risk in software development refers to the potential risk of delivering a product that does not meet quality standards. This can result in customer dissatisfaction, loss of revenue, and damage to the reputation of the development team or organization. There are several factors that can contribute to quality risk in software development, including: 1) Poor requirements: If the requirements for the software are not clear or are incomplete, it can lead to a product that does not meet the needs of the end-users. 2) Lack of testing: If the software is not tested thoroughly, it can lead to bugs and other issues that impact the quality of the product. 3) Time constraints: If there are tight deadlines for the project, it can lead to rushed development and testing, which can compromise the quality of the product. 4) Inadequate resources: If the development team lacks the necessary resources, such as skilled personnel, tools, or hardware, it can impact the quality of the product. 5) Poor communication: If there is poor communication between the development team and the stakeholders, it can lead to misunderstandings and errors that impact the quality of the product.
* Technical challenges: If the software is technically complex, it can require specialized knowledge and skills that the developer may not possess. This can lead to delays in the project and impact the quality of the software.
* Poorly defined requirements: If the requirements for the software are unclear or incomplete, it can lead to misunderstandings and errors that impact the quality of the software.
* Inadequate testing: If the software is not tested thoroughly, it can lead to bugs and other issues that impact the quality of the software.
* Tight deadlines: If there are tight deadlines for the project, it can lead to rushed development and testing, which can compromise the quality of the software.
* Burnout and stress: Software development can be stressful and demanding, which can lead to burnout and impact the mental and physical health of the developer.

## 3.3 Product users:

* Security vulnerabilities: If the software has security vulnerabilities, it can expose user data to unauthorized access or compromise the integrity of the system.
* Performance issues: If the software is slow or unresponsive, it can impact user experience and lead to frustration.
* Compatibility issues: If the software is not compatible with other software or hardware, it can lead to issues with integration and impact user productivity.
* Data loss or corruption: If the software is not designed to handle data properly, it can lead to data loss or corruption, which can impact user productivity and result in data breaches.
* Usability issues: If the software is difficult to use or has a poor user interface, it can impact user experience and lead to user frustration.

# Expected users and reading suggestions

## 4.1 Customers:

* Review the design document carefully: As a customer, it is important to review the design document carefully to ensure that it meets your requirements and expectations. Make note of any areas that are unclear or that require further clarification.
* Provide feedback: If you have any concerns or questions about the design document, provide feedback to the development team. This will help ensure that the final product meets your needs and expectations.
* Understand the technical details: While you may not be a technical expert, it is important to have a basic understanding of the technical details outlined in the design document. This will make it easier to communicate with the development team and understand the progress of the project.
* Be open to changes: Software development is an iterative process, and changes may need to be made to the design document as the project progresses. Be open to these changes and work collaboratively with the development team to ensure that the final product meets your needs.
* Ask for updates: It is important to stay up-to-date on the progress of the project. Ask the development team for regular updates on the status of the project and any changes that have been made to the design document.

## 4.2 Software developers:

* Understand the requirements: It is important to understand the requirements outlined in the design document. If there are any areas that are unclear, ask the customer for clarification.
* Use a standardized format: Use a standardized format for the design document to ensure that it is organized and easy to read. This will help ensure that all team members understand the requirements and are working towards a common goal.
* Make it comprehensive: The design document should be comprehensive, covering all aspects of the software development project. This will help ensure that the final product meets the customer's needs and expectations.
* Be specific: Use specific language in the design document to avoid ambiguity. This will help ensure that everyone on the development team understands what is expected of them.
* Keep it up-to-date: The design document should be kept up-to-date throughout the software development process. If changes are made to the requirements, update the design document to reflect these changes.
* Collaborate with the customer: Work collaboratively with the customer to ensure that the design document accurately reflects their needs and expectations. This will help ensure that the final product meets their requirements.

## 4.3 Project managers:

* Ensure that the design document is clear and comprehensive: As a project manager, it is important to ensure that the design document is clear and comprehensive and covers all aspects of the software development project. This will help ensure that the final product meets the customer's needs and expectations.
* Review the design document with the development team: Review the design document with the development team to ensure that everyone understands the requirements and is working towards a common goal. This will help ensure that the project stays on track and is completed on time and within budget.
* Monitor progress: Monitor the progress of the project and ensure that the development team is following the design document. If there are any deviations from the design document, address them promptly to ensure that the project stays on track.
* Communicate with the customer: Communicate regularly with the customer to ensure that the design document accurately reflects their needs and expectations. If there are any changes to the requirements, update the design document to reflect these changes.
* Use the design document as a reference: Use the design document as a reference throughout the software development project to ensure that the project stays on track and that everyone is working towards a common goal.
* Encourage collaboration: Encourage collaboration between the development team and the customer to ensure that the final product meets the customer's needs and expectations.

## 4.4 Sales:

* Understand the customer's requirements: It is important to understand the customer's requirements before creating the design document. Ask the customer questions to clarify their needs and expectations.
* Collaborate with the customer: Work collaboratively with the customer to ensure that the design document accurately reflects their needs and expectations. This will help ensure that the final product meets their requirements.

## Testers:

* Understand the requirements: It is essential to understand the requirements outlined in the design document to ensure that the software being tested meets the customer's needs and expectations.
* Verify that the requirements are testable: The requirements outlined in the design document should be specific and testable. If the requirements are ambiguous or unclear, work with the development team to clarify them.
* Create test cases based on the requirements: Based on the requirements outlined in the design document, create test cases that cover all possible scenarios. This will ensure that all requirements are tested thoroughly.
* Use the design document as a reference: Use the design document as a reference throughout the testing process to ensure that the software being tested meets the requirements outlined in the document.
* Collaborate with the development team: Work collaboratively with the development team to ensure that any issues identified during testing are addressed promptly and effectively.
* Provide feedback: If there are any issues or concerns with the design document, provide feedback to the development team. This will help ensure that the final product meets the customer's needs and expectations.

## 4.6 Document writers:

* Understand the customer's requirements: It is essential to understand the customer's requirements before creating the design document. Work collaboratively with the customer to ensure that their needs and expectations are captured accurately.
* Use a standardized format: Use a standardized format for the design document to ensure that it is organized and easy to read. This will help ensure that all team members understand the requirements and are working towards a common goal.
* Make it comprehensive: The design document should be comprehensive, covering all aspects of the software development project. This will help ensure that the final product meets the customer's needs and expectations.
* Be specific: Use specific language in the design document to avoid ambiguity. This will help ensure that everyone on the development team understands what is expected of them.
* Keep it up-to-date: The design document should be kept up-to-date throughout the software development process. If changes are made to the requirements, update the design document to reflect these changes.
* Use visuals: Use visuals such as diagrams or flowcharts to illustrate complex concepts. This will help ensure that everyone on the development team understands the requirements.
* Collaborate with the development team: Work collaboratively with the development team to ensure that the design document accurately reflects the requirements and is useful for the development process.

# Design Brief

This section describes the existing development conditions and the objectives to be achieved, and explains the design principles that should be followed and the design methods that must be adopted when doing the outline design.

# Limitations and Constraints

## 6.1 Technique:

* Technology platforms: The choice of programming languages, development frameworks, and software libraries can constrain the development team's approach to software development.
* Development methodologies: The choice of development methodology, such as agile or waterfall, can constrain the development team's approach to software development.
* Development tools: The choice of development tools, such as integrated development environments (IDEs) or version control systems, can constrain the development team's approach to software development.
* Skillset: The available skillset of the development team can limit the use of certain software development techniques or tools.

## 6.2 Budget:

* Money constraints: Money constraints can limit the development team's ability to use certain software development techniques or tools.
* Resource allocation: Budget constraints may limit the resources, such as personnel or technology, that can be allocated to the development team. This can impact the team's ability to complete the project within the desired timeframe or to the desired level of quality.
* Choice of technology: Budget constraints may limit the choice of technology platforms and tools that can be used in the development process. This can impact the team's ability to use the most advanced or appropriate technology for the project.
* Development methodology: Budget constraints may also impact the choice of development methodology. For example, an iterative approach, such as Agile, may be more appropriate for projects with budget constraints as it allows for more flexibility in scope and budget.
* Testing and quality assurance: Budget constraints may limit the resources available for testing and quality assurance, which can impact the quality of the final product.
* Scalability: Budget constraints may impact the scalability of the software developed. If a project has a limited budget, it may be difficult to develop software that can scale to meet future needs.

## 6.3 Developing environment:

* Hardware limitations: Developing software may require specific hardware resources, such as memory or processing power, which may not be available in the development environment.
* Software dependencies: Developing software may require specific software dependencies, such as libraries or frameworks, which may not be available in the development environment.
* Network connectivity: Developing software may require network connectivity to access resources or services, which may not be available in the development environment.
* Security restrictions: Developing software may require specific security measures, such as firewalls or access controls, which may restrict the development environment.
* Development policies and procedures: Developing software may require adherence to specific development policies and procedures, such as version control or testing procedures, which may not be available in the development environment.

## 6.4 Time:

* Scope prioritization: When time is limited, it may not be possible to develop all the features or requirements that were initially planned. Development teams may need to prioritize the scope of the project and focus on the most important features or requirements first.
* Agile approach: An agile approach to software development can be helpful when time is limited. Agile methodology allows for more flexibility in scope and budget, which can help development teams to deliver software within the limited time frame.
* Resource allocation: Time constraints may limit the resources, such as personnel or technology, that can be allocated to the development team. This can impact the team's ability to complete the project within the desired timeframe or to the desired level of quality.
* Testing and quality assurance: Time constraints may limit the resources available for testing and quality assurance, which can impact the quality of the final product.
* Project management: Time constraints may require more rigorous project management to ensure that the project stays on track and is completed within the desired timeframe.

## 6.5 Policy:

* Compliance regulations: Organizations in certain industries, such as finance or healthcare, may be subject to compliance regulations that impact the software development process. For example, the software may need to meet specific security or privacy requirements.
* Intellectual property: Organizations may have intellectual property policies that impact the software development process. For example, the organization may require that all software developed be owned by the organization.
* Open-source software: Organizations may have policies regarding the use of open-source software in software development. For example, the organization may require that all open-source software used meet specific licensing requirements.
* Version control: Organizations may have policies regarding version control, such as requiring that all code changes be reviewed and approved before being merged into the main codebase.
* Testing and quality assurance: Organizations may have policies regarding testing and quality assurance, such as requiring that all software be rigorously tested before being released.

# Design principles and requirements

## 7.1 Name principle:

* SOLID: SOLID is an acronym that stands for Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion. This principle is designed to help create software that is easy to maintain, extend, and modify.
* DRY: DRY stands for Don't Repeat Yourself. This principle emphasizes the importance of avoiding duplicating code or functionality in software development.
* KISS: KISS stands for Keep It Simple, Stupid. This principle emphasizes the importance of keeping software development simple and avoiding unnecessary complexity.
* YAGNI: YAGNI stands for You Ain't Gonna Need It. This principle emphasizes the importance of avoiding developing functionality that is not currently needed in the software.
* Agile Manifesto: The Agile Manifesto is a set of principles for agile software development. This manifesto emphasizes the importance of individuals and interactions, working software, customer collaboration, and responding to change.

## 7.2 Module independence principle:

* The Module Independence Principle is a software development principle that emphasizes the importance of designing software modules that are independent of one another. This principle is also known as the Single Responsibility Principle.
* The Module Independence Principle states that a module should have only one reason to change. This means that a module should be responsible for one specific function or behavior, and changes to that function or behavior should only impact that module.
* By designing modules that are independent of one another, developers can create software that is easier to maintain, test, and modify. It also helps to reduce the risk of unintended side effects when making changes to one module, as changes are less likely to impact other modules.
* In practice, achieving module independence can involve breaking down larger modules into smaller, more focused modules. It can also involve creating interfaces between modules, which can help to define clear boundaries and ensure that each module is responsible for its own behavior.

## 7.3 Boundary design principle:

* The Boundary Design Principle is a software development principle that emphasizes the importance of clearly defining the boundaries between different components or modules of a software system.
* The Boundary Design Principle states that components or modules should have well-defined interfaces that define how they interact with other components or modules. This helps to ensure that components or modules can be developed and tested independently, and can be easily replaced or modified without impacting other parts of the system.
* In practice, achieving boundary design can involve creating clear and concise documentation of interfaces between components or modules. It can also involve using design patterns, such as the Adapter or Facade patterns, to help manage interactions between components or modules.
* By using the Boundary Design Principle, developers can create software systems that are modular, flexible, and easy to maintain over time. It also helps to reduce the risk of unexpected side effects or errors when making changes to one part of the system, as changes are less likely to impact other parts of the system due to the well-defined boundaries.

## 7.4 Database design principle:

* Normalization: Normalization is the process of organizing data in a database so that it is free from redundancy and dependencies. Normalization helps to ensure that data is stored in a way that is efficient and easy to maintain.
* Data integrity: Data integrity is the accuracy and consistency of data in a database. Database designers should ensure that data is validated and that constraints are put in place to prevent invalid or inconsistent data from being stored.
* Scalability: Database designers should design databases to be scalable, so that they can handle increasing amounts of data and traffic without performance degradation.
* Indexing: Indexing is the process of creating indexes on database columns to improve query performance. Database designers should create indexes on columns that are frequently queried.
* Security: Database designers should ensure that databases are secure, with appropriate access controls and encryption.
* Backup and recovery: Database designers should create backup and recovery procedures to ensure that data can be recovered in the event of a system failure or data loss.

## 7.5 Necessary safety measures:

* Secure coding practices: Developers should follow secure coding practices, such as avoiding hardcoded passwords or using input validation to prevent SQL injection attacks.
* Encryption: Sensitive data should be encrypted to prevent unauthorized access.
* Authentication and access control: User authentication and access control mechanisms should be implemented to ensure that only authorized users have access to sensitive data.
* Testing and quality assurance: Rigorous testing and quality assurance processes should be implemented to ensure that the software is free from vulnerabilities and defects.
* Regular updates and patches: Regular software updates and patches should be released to address security vulnerabilities and fix bugs.
* Compliance with regulations: Software should be developed in compliance with relevant regulations, such as data protection regulations.
* Disaster recovery planning: Disaster recovery plans should be put in place to ensure that data can be recovered in the event of a system failure or data loss.

## 7.6 Security and confidentiality principles:

* Authentication and access control: Implement authentication and access control mechanisms to ensure that only authorized users have access to sensitive data.
* Encryption: Sensitive data should be encrypted to prevent unauthorized access.
* Input validation: Validate user input to prevent attacks such as SQL injection or cross-site scripting attacks.
* Least privilege: Follow the principle of least privilege, which means that users and processes should only have the minimum access necessary to do their job.
* Secure coding practices: Follow secure coding practices to avoid common vulnerabilities such as buffer overflows or hardcoded passwords.
* Confidentiality: Ensure that sensitive data is kept confidential and is not accessible to unauthorized users.
* Data protection: Implement data protection mechanisms such as backups, redundancy, and disaster recovery to protect against data loss.
* Compliance with regulations: Develop software in compliance with relevant regulations such as data protection regulations like GDPR, HIPAA, or PCI-DSS.

## 7.7 System flexibility requirement:

* Modularity: Develop software components that can be easily replaced or modified without impacting the rest of the system.
* Extensibility: Develop software components that can be easily extended to support new features or requirements.
* Scalability: Develop software that can scale to meet changing demands, whether in terms of data volume, user traffic, or other factors.
* Reusability: Develop software components that can be reused across multiple projects or applications, reducing development time and effort.
* Agility: Develop software using agile methodologies that enable the development team to respond quickly to changing requirements or conditions.
* Open standards and interfaces: Implement open standards and interfaces to allow for integration with other systems or technologies.

## 7.8 System operability requirements:

* User-friendly interface: Develop a user interface that is intuitive and easy to use, minimizing the need for training or support.
* Error handling and logging: Implement error handling and logging mechanisms that enable system administrators to quickly identify and resolve issues.
* Monitoring and alerting: Implement monitoring and alerting mechanisms that enable system administrators to proactively identify and resolve issues before they impact users.
* Documentation: Develop documentation that accurately describes the system architecture, configuration, and operation, enabling system administrators to quickly understand and troubleshoot the system.
* Scalability: Develop software that can scale to meet changing demands, ensuring that the system can handle increasing user traffic or data volume without performance degradation.
* Compatibility: Ensure that the system is compatible with a wide range of hardware and software configurations, reducing the need for specialized infrastructure or software.

## 7.9 System maintainability requirements:

* Modularity: Develop software components that are modular and can be easily replaced or modified without impacting the rest of the system.
* Documentation: Develop documentation that accurately describes the system architecture, configuration, and operation, enabling developers to quickly understand and modify the system.
* Version control: Implement version control mechanisms that enable developers to track changes to the system over time and roll back changes if necessary.
* Testing and quality assurance: Implement rigorous testing and quality assurance processes to ensure that changes to the system do not introduce defects or vulnerabilities.
* Code maintainability: Develop code that is maintainable and easy to understand, using clear and consistent coding standards and practices.
* Scalability: Develop software that can scale to meet changing demands, ensuring that the system can handle increasing user traffic or data volume without performance degradation.

# UI Design

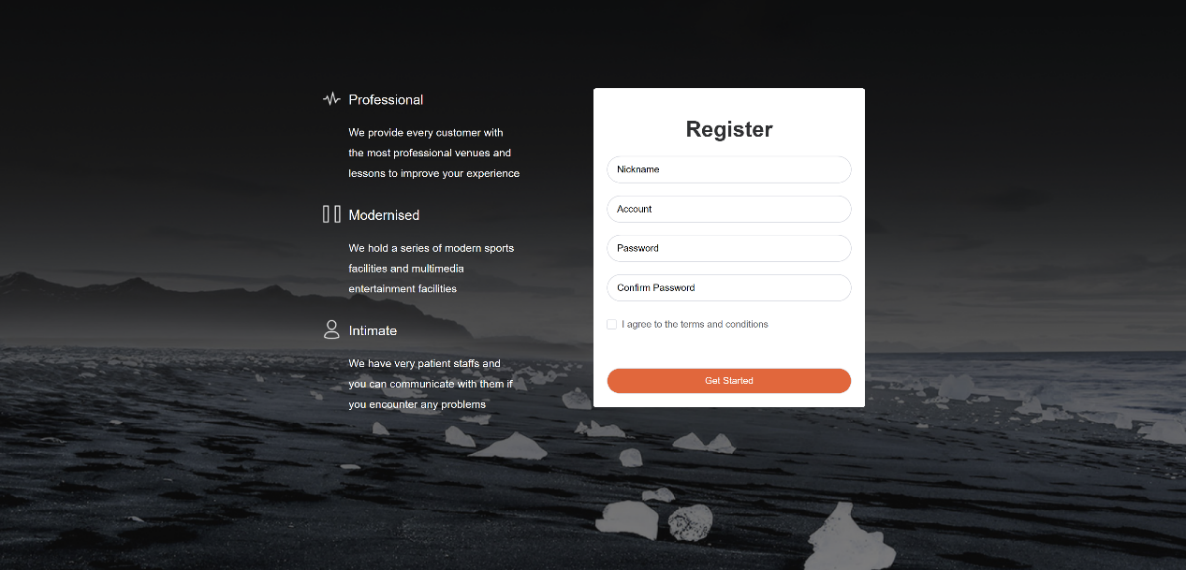
## **8.1 Design style and color scheme**

The design style of the system should be modern and concise, helping users quickly find the information and functions they need. The color scheme should use sports related colors, we choose orange.

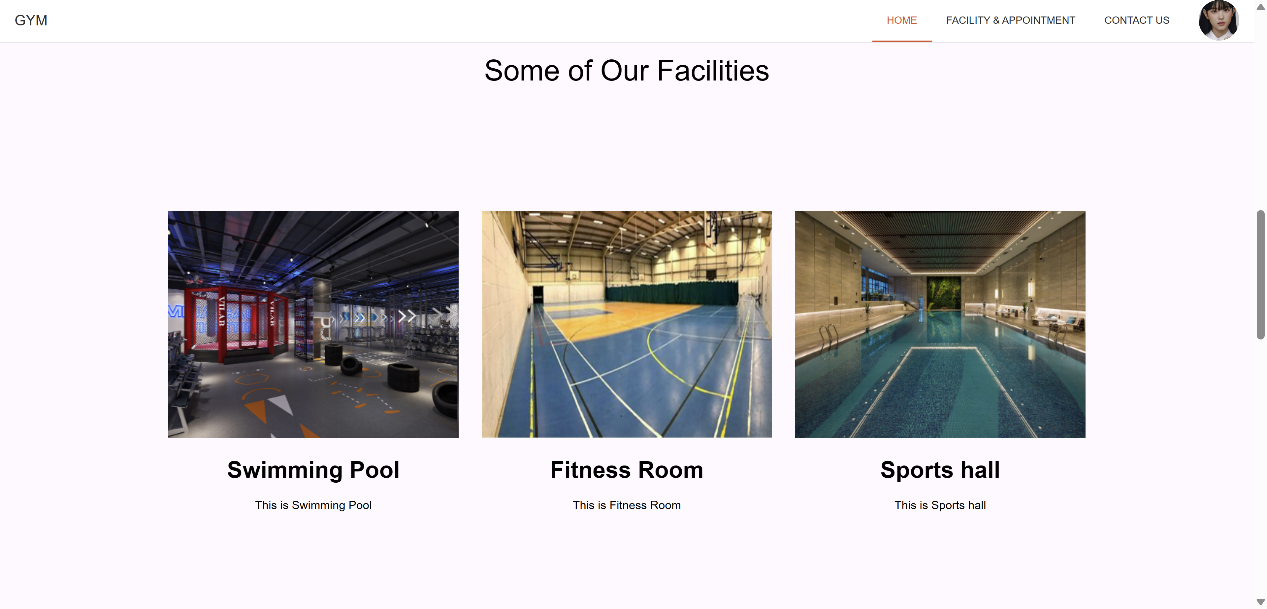
## 8.2 Page design

1. Login and regisiter

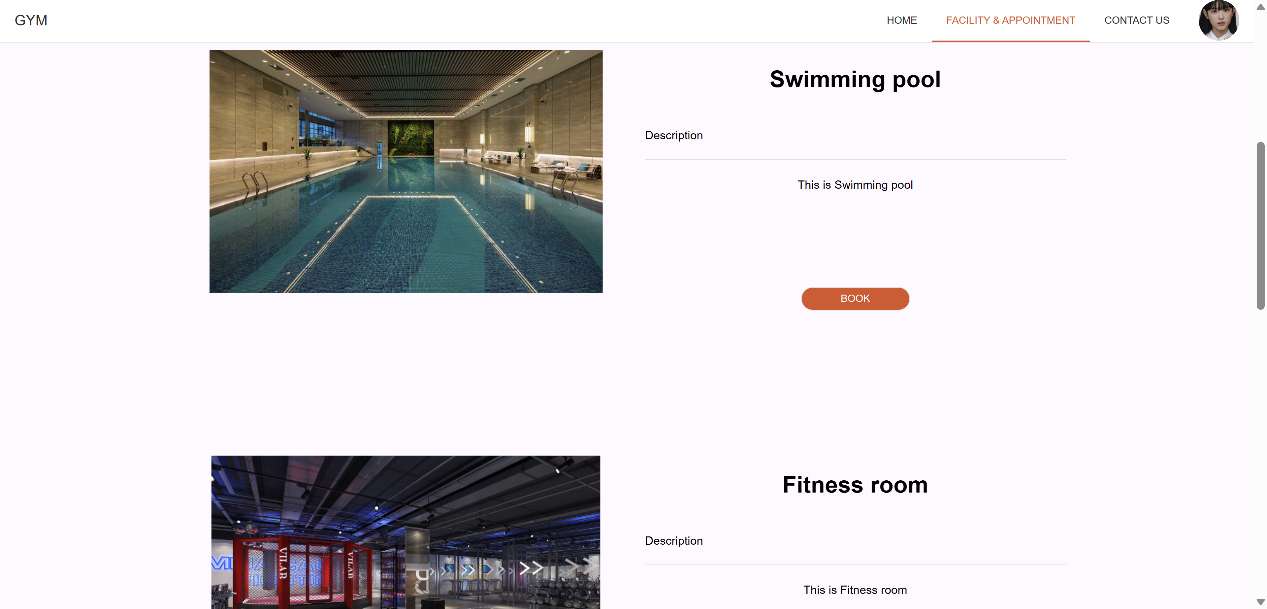




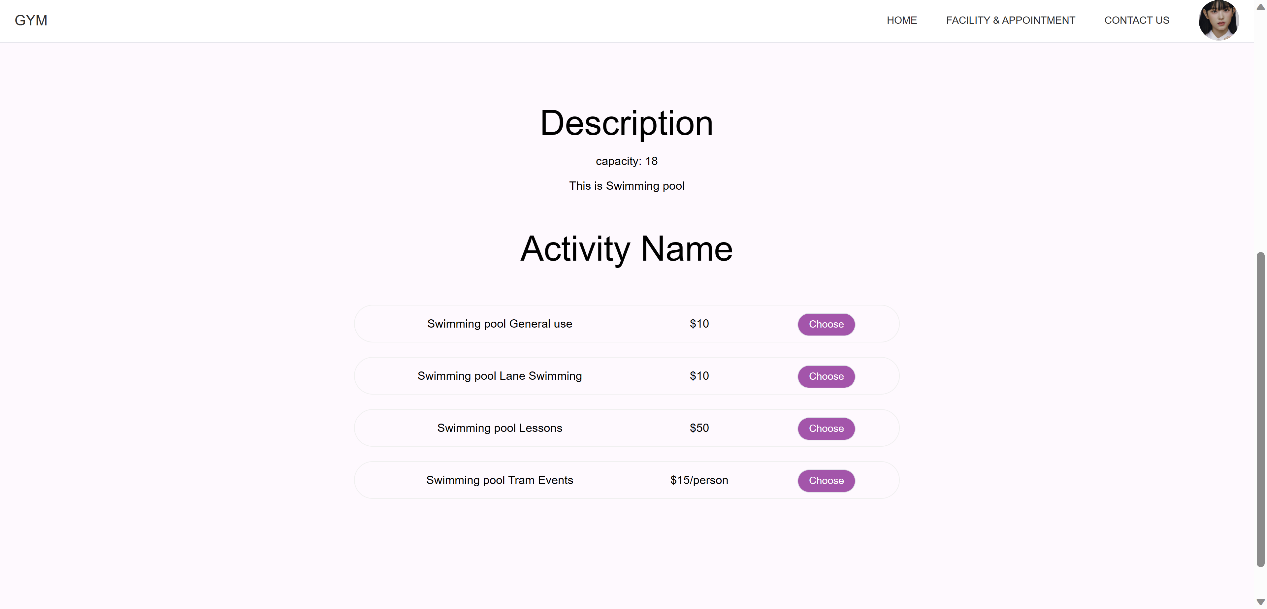
1. Homepage

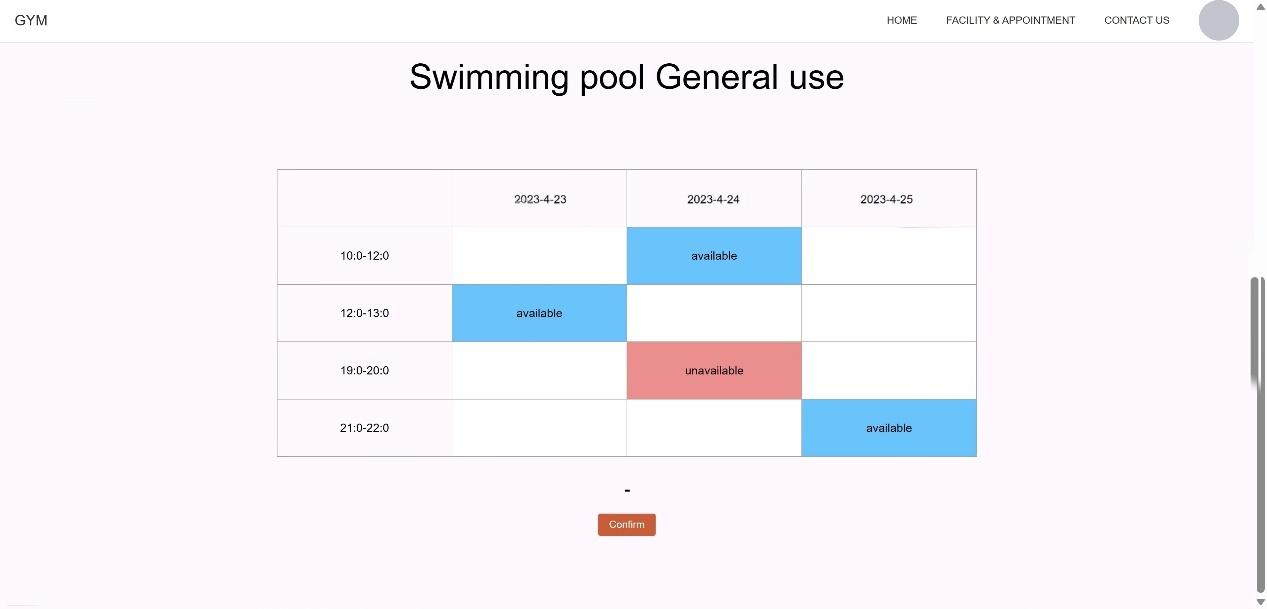


1. Facility page

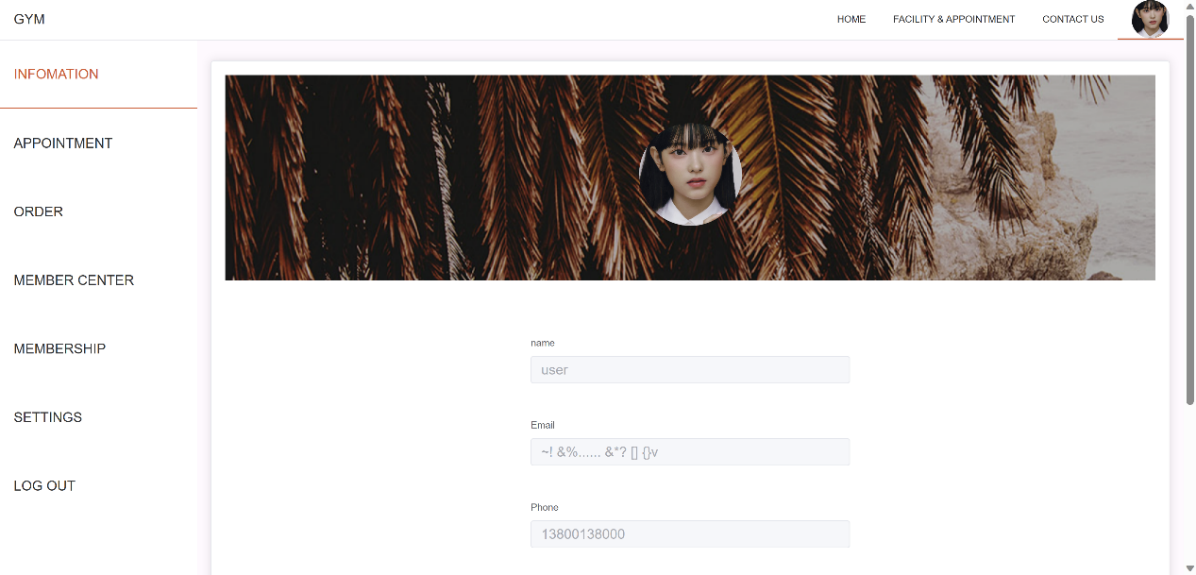


1. Timetablepage

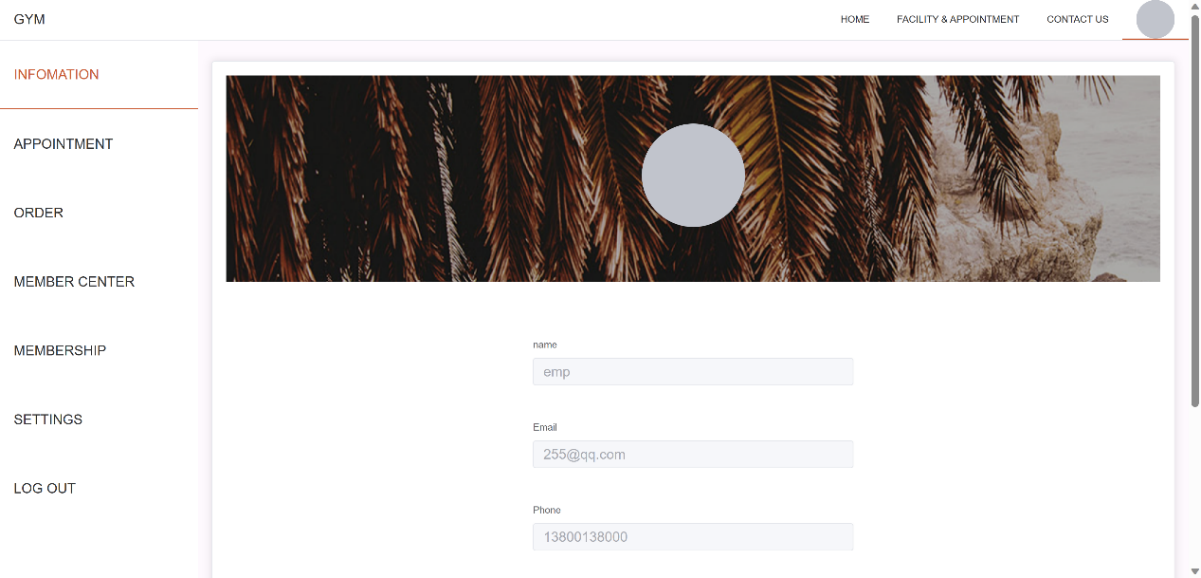




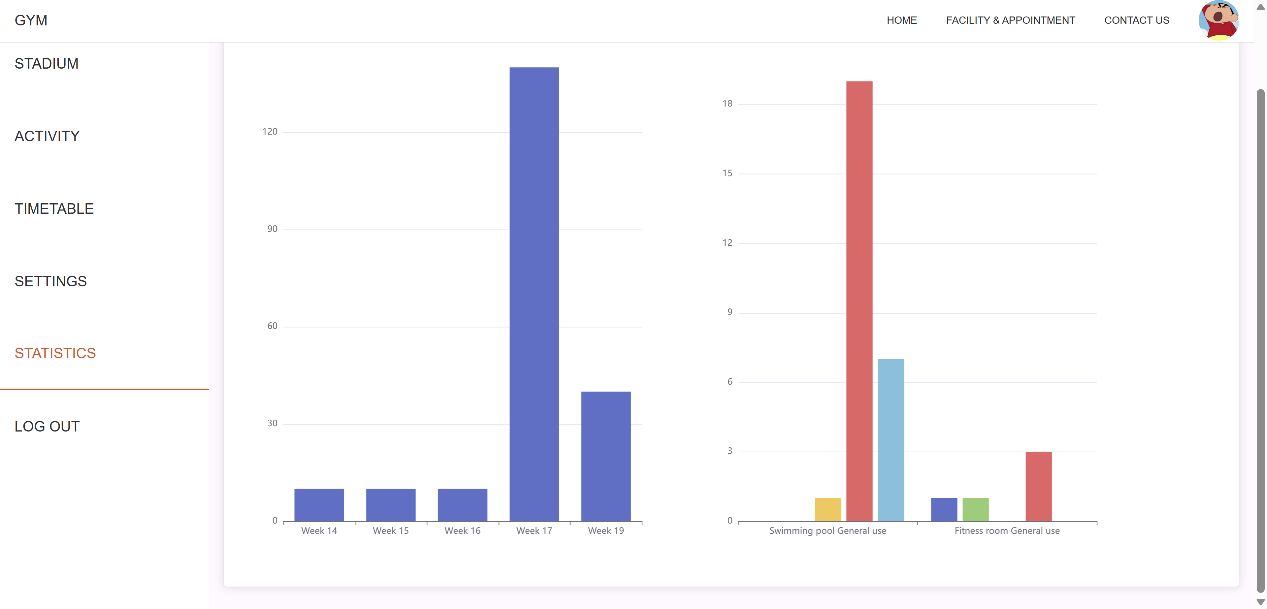
1. User homepage



1. Employee homepage



1. Manager homepage



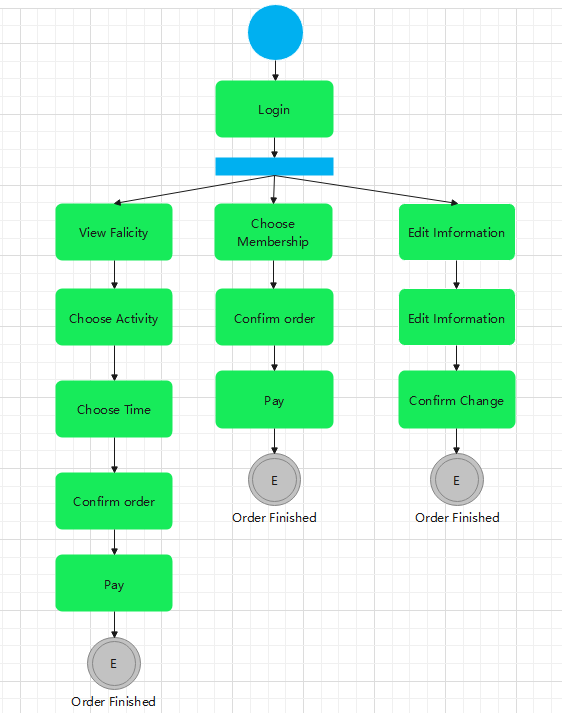
1. Module Design

## 9.1 User

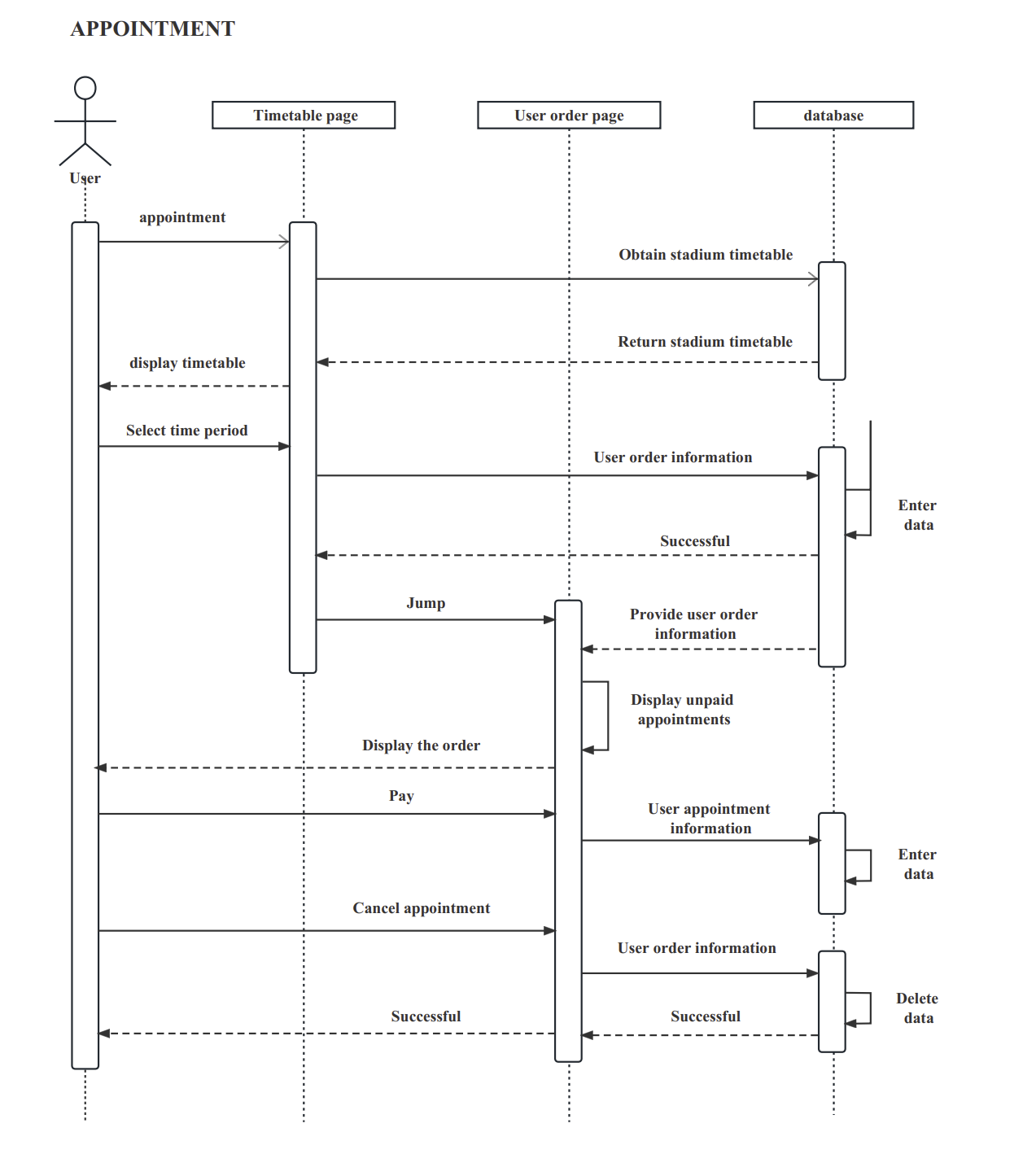
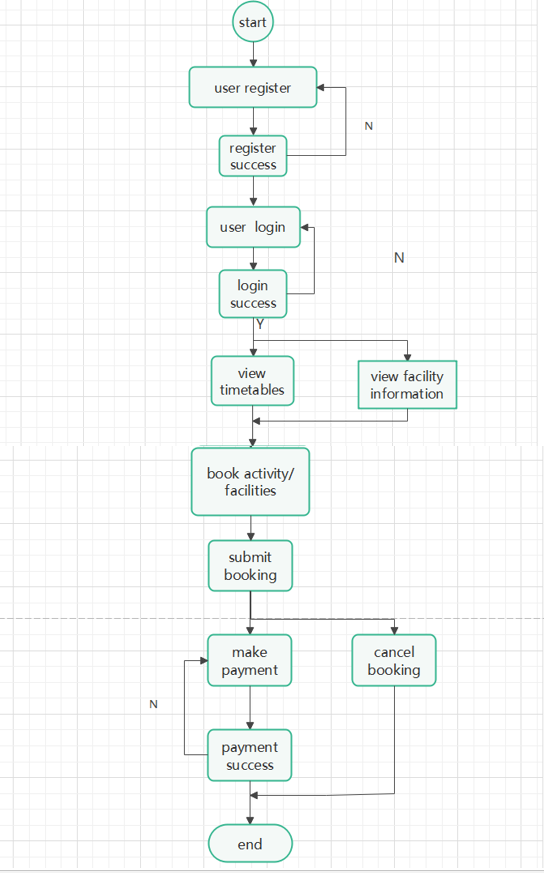
Through requirement analysis, we have learned that the user's behavior in the software can be divided into three parts:

1. Browse venues and make appointments and pay
2. Recharge membership and pay
3. Modify personal information and save it

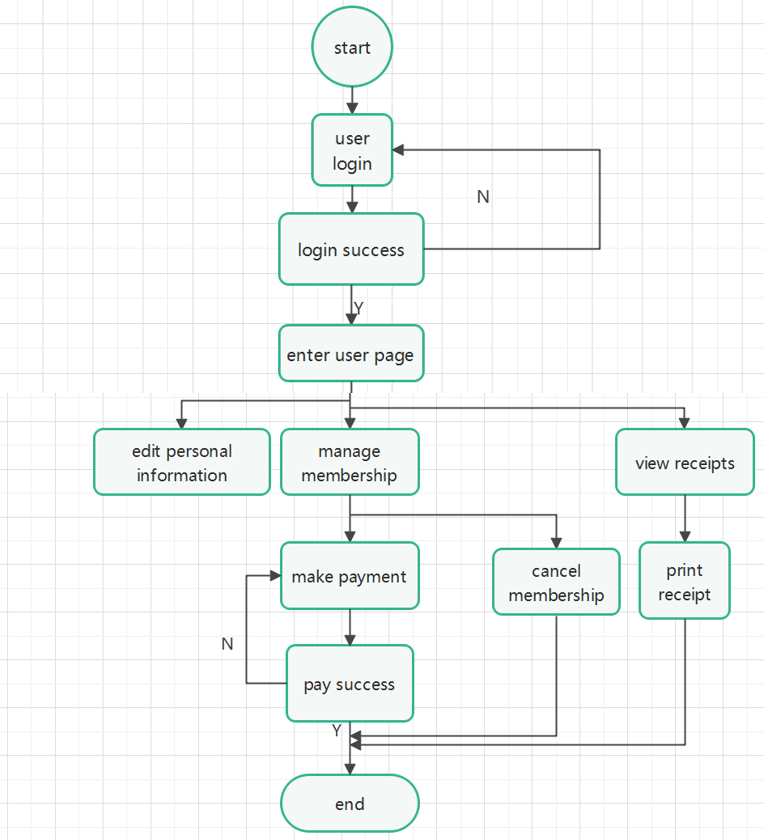
The following diagram shows the user's activity diagram:

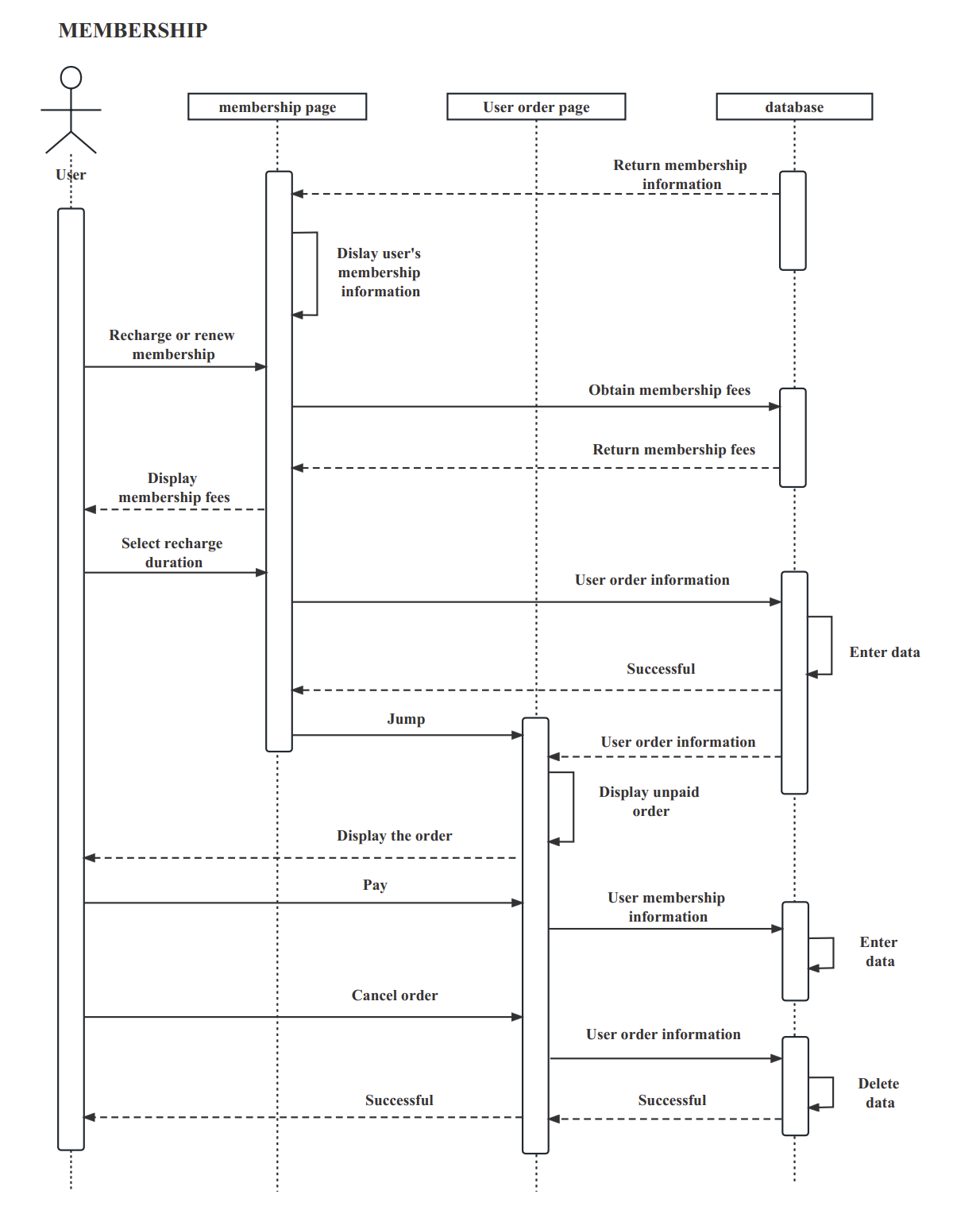


User Appointment Flowchart and Sequence Diagram：

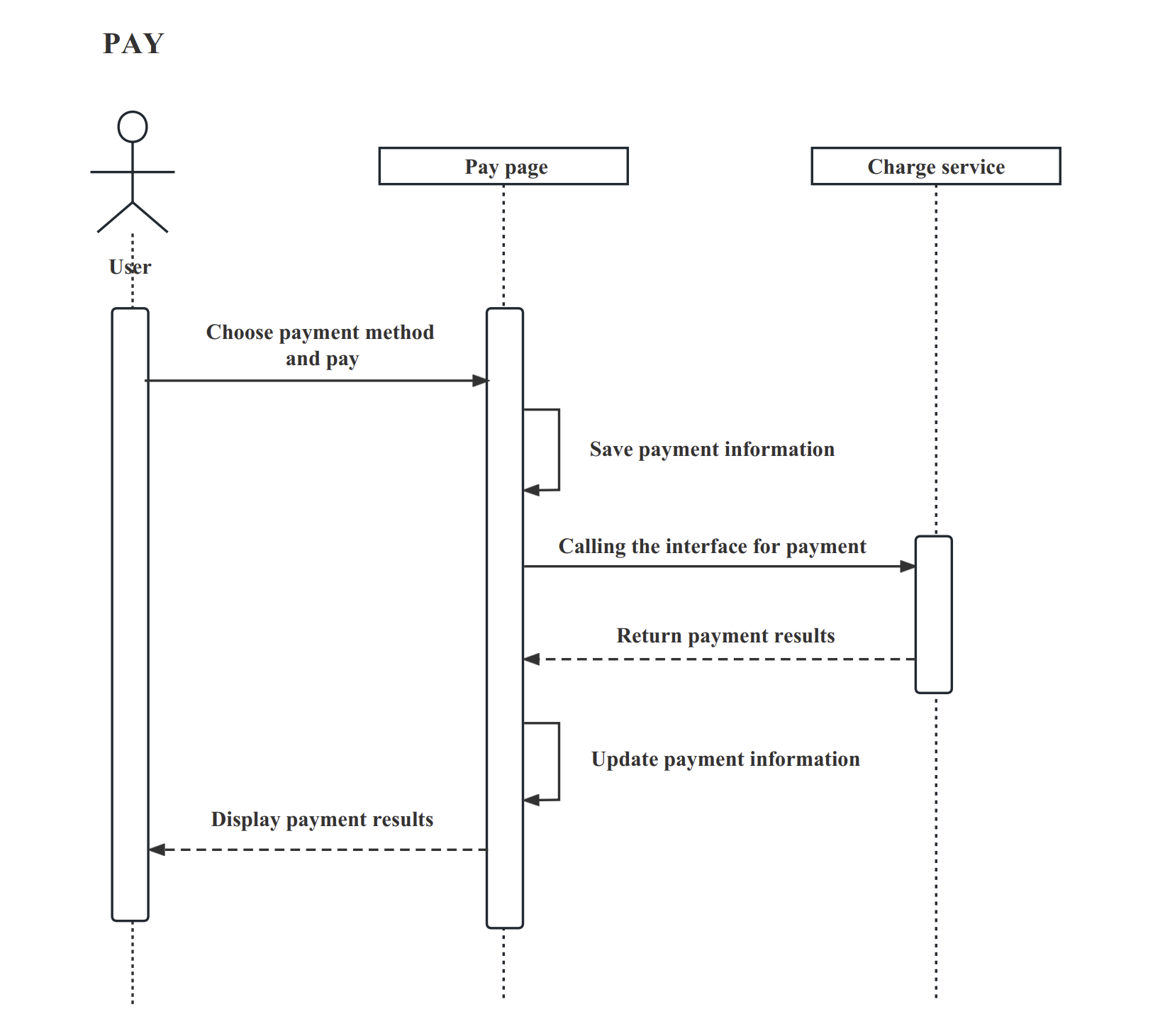


User purchase membership flowchart and Sequence Diagram:





User payment Sequence Diagram:

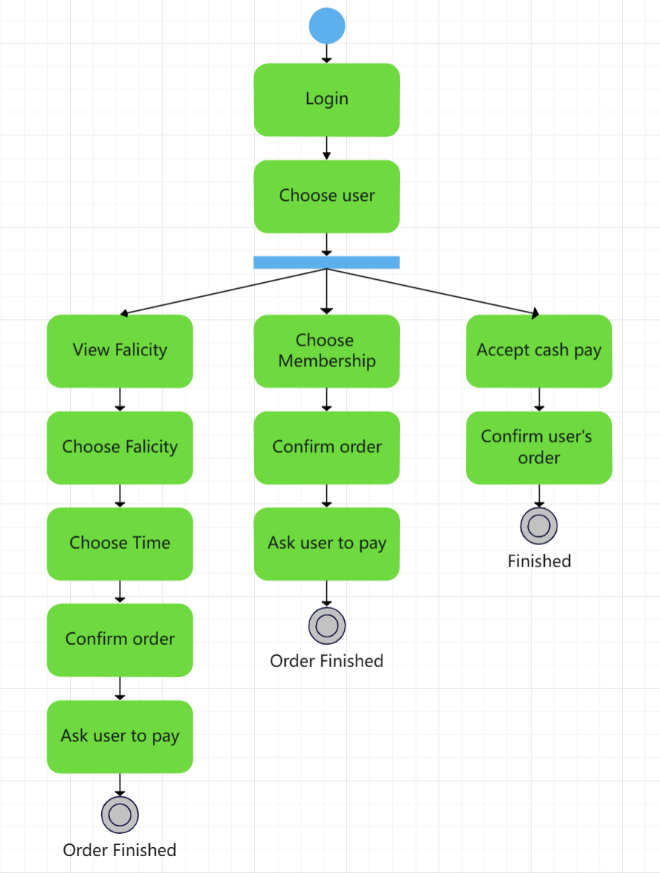


## 9.2 Employee

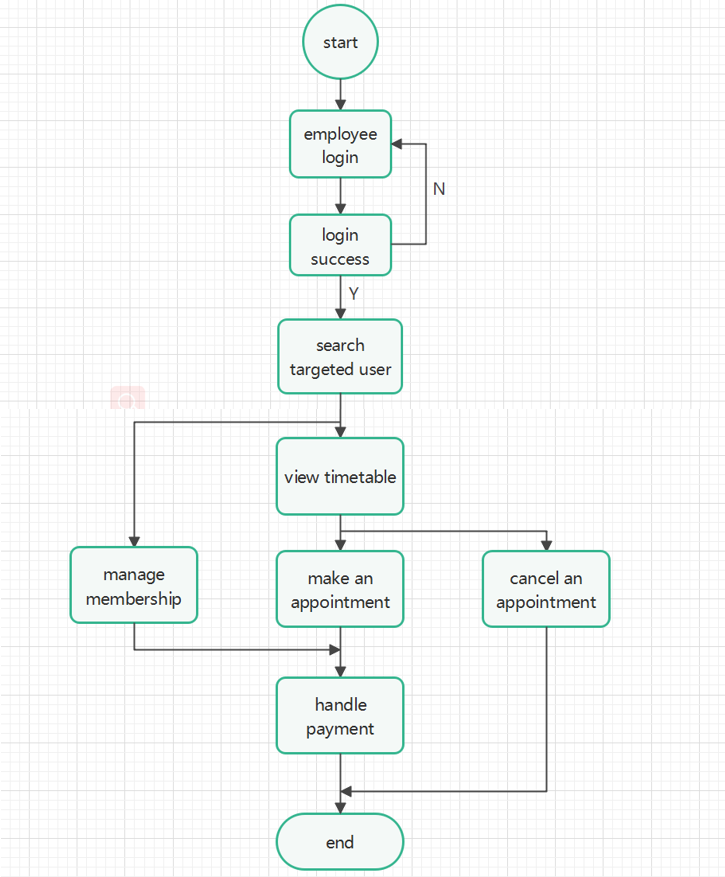
Based on the needs analysis, we have learned that employees should be able to

1. view activities and schedules to help users make appointments
2. assist users in membership processing
3. process cash payment activities,
4. send receipts

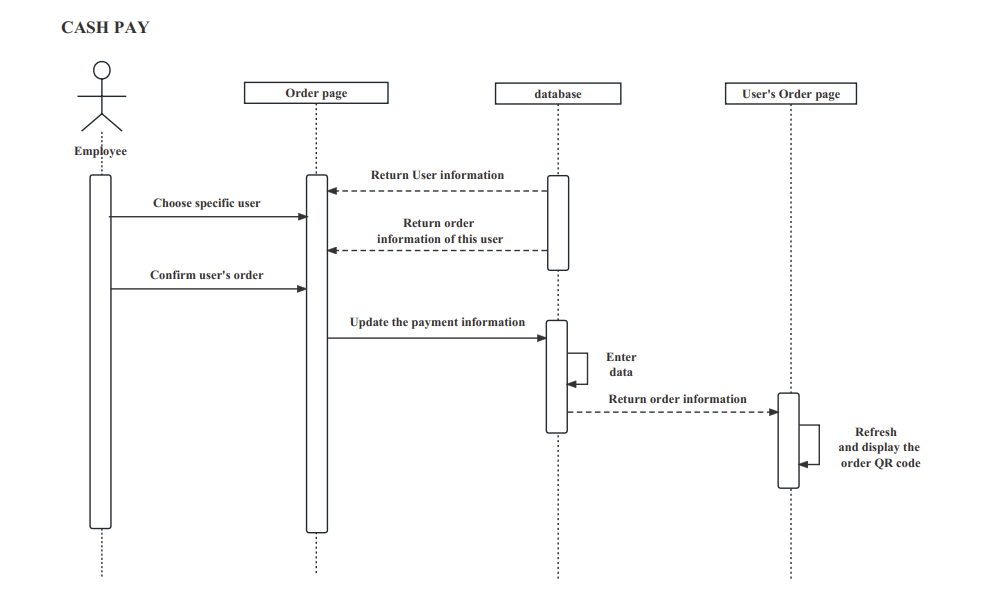
The following diagram shows the manager’s activity diagram:



Employee activities flowchart:



Employee accept cash pay Sequence Diagram:

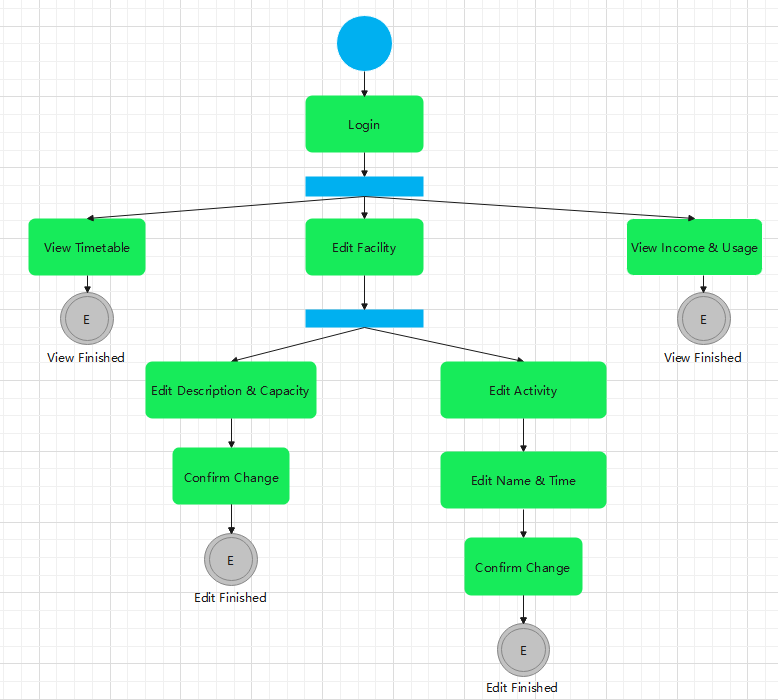


## 9.3 manager

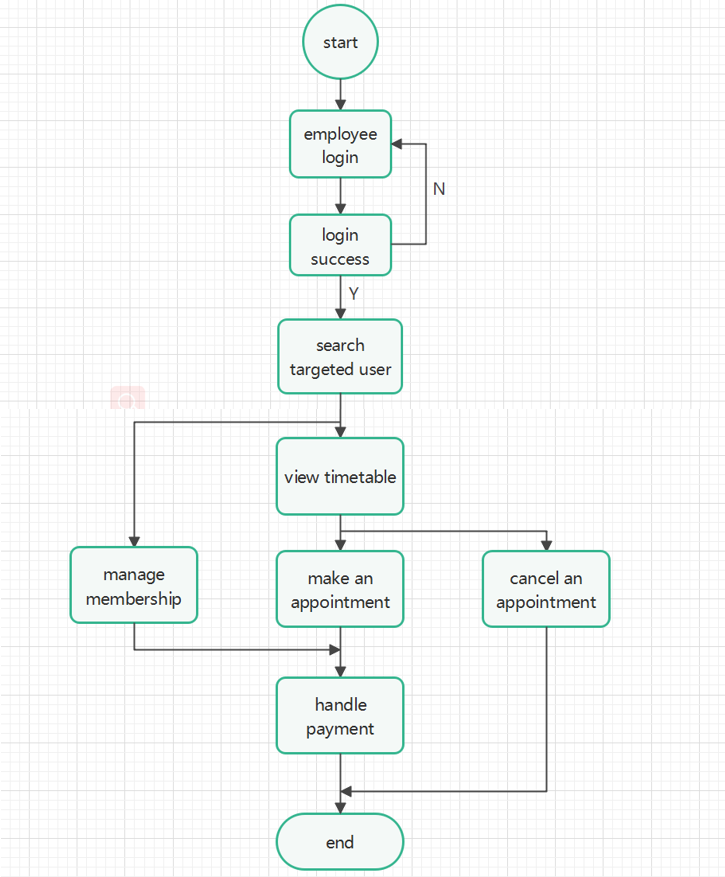
Based on demand analysis, we have learned that manager should be able to:

1. view timetable
2. manage stadiums and activities
3. view summary data on weekly stadiums usage and revenue

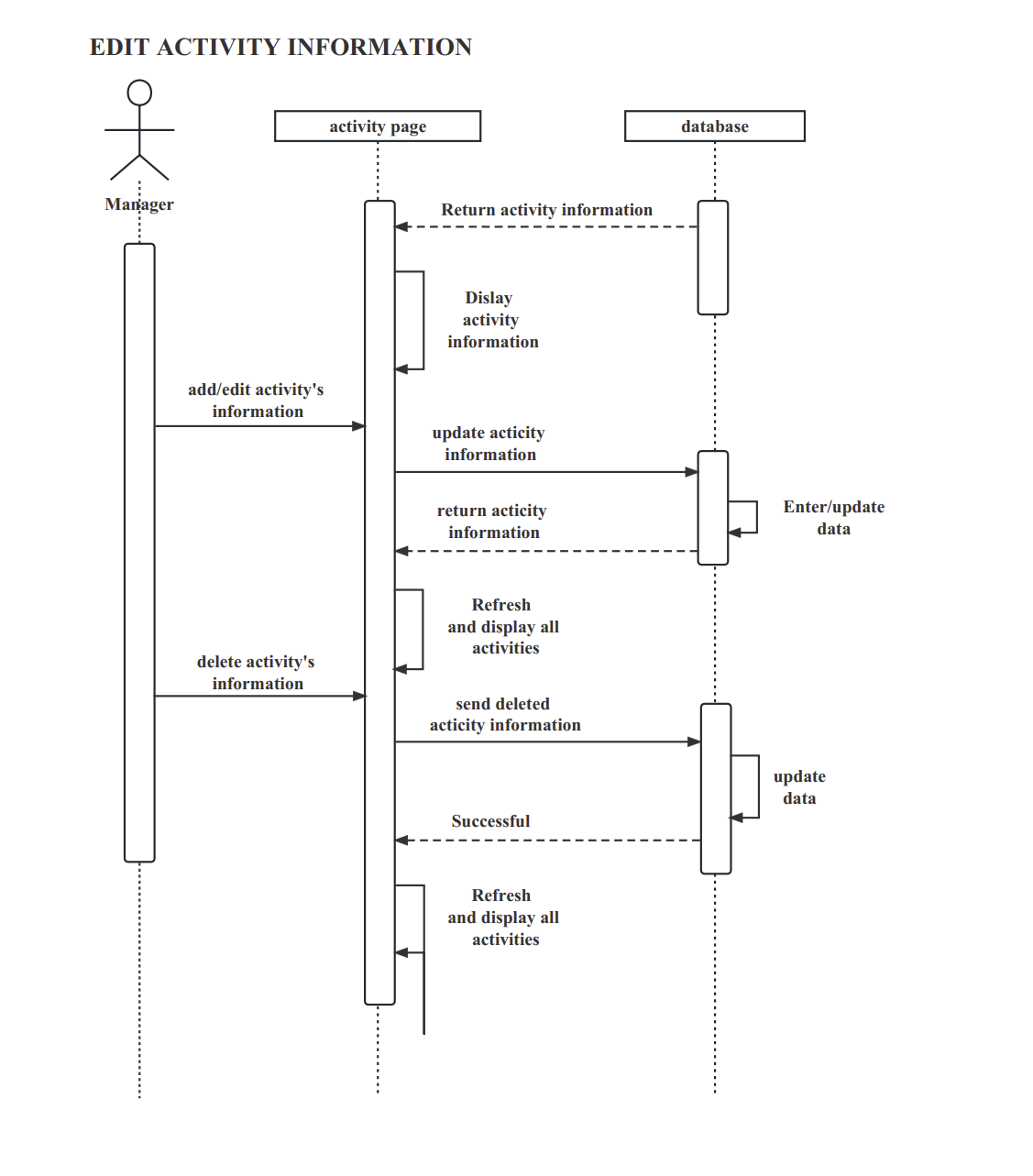
The following diagram shows the manager’s activity diagram:



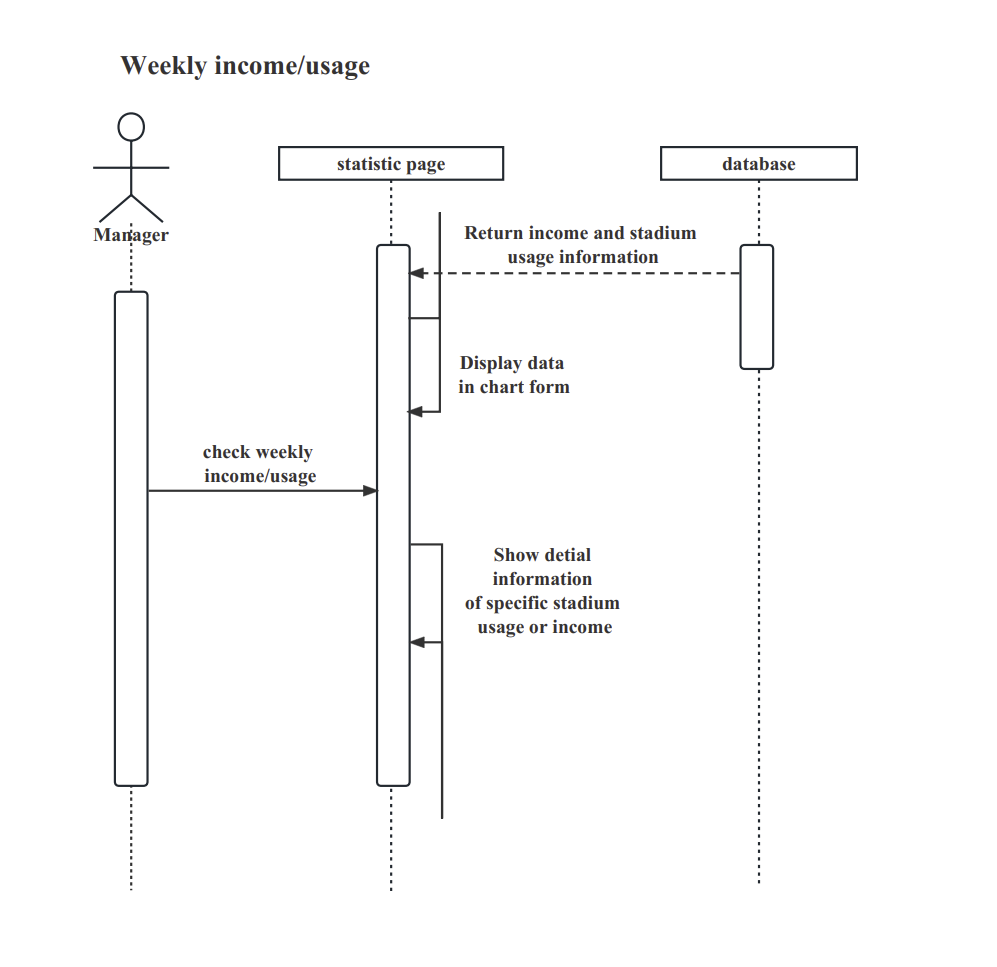
Manager activities flowchart:

****

Manager edit activity Sequence Diagram:



Manager check weekly income/usage Sequence Diagram:



# Interface design

## 10.1 ActivityService/ AppointmentService/ MembershipService/ OrderService/ StadiumService/ SysUserService/ TimetableService

The following section is the interface for controlling activities/ appointments/ memberships/ stadiums/ users/ timetables, including paging, showing, get details, adding, deleting, and modifying. The following content takes the interface for controlling activities as an example, and the interface for controlling other content is the same.

1. Page

*Request type:* POST

*Function description:* Paging Activities.

*Input parameter:* pageDTO(PageDTO)

*Output parameters:* pageDTO(PageDTO)

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. listall

*Request type:* GET

*Function description:* List all Activities.

*Input parameter:* null

*Output parameters:* temperature: list(List<Activity>)

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. getActivityById

*Request type:* GET

*Function description:* Query specific activity details.

*Input parameter:* id(String)

*Output parameters:* *e.*g.:

{ “id”: 1649985909376151444,

“Stadium\_id”: 1649985546757599234,

“name”: Squash courts General use,

“price”: 10,

“unit”: ,

“create\_time”: 2023-04-23 11:57:51,

“create\_by”: admin}

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. deleteActivityById

*Request type:* DELETE

*Function description:* Delete an Activity.

*Input parameter:* id(String)

*Output parameters:* "Delete Success" /error

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. addActivityById

*Request type:* POST

*Function description:* Add an Activity.

*Input parameter:* obj(Activity)

*Output parameters:* "Add Success" /error

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. updateActivityById

*Request type:* PUT

*Function description:* Modify an Activity.

*Input parameter:* obj(Activity)

*Output parameters:* "Update Success" /errpr

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

## 10.2 MembershipService

The following content is the remaining function in member management except the above six functions: obtaining member status through username.

1. getMemberByUserName

*Request type:* GET

*Function description:* Obtain member information through name.

*Input parameter:* null

*Output parameters:* membership

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

## 10.3 OrderService

The following content is the remaining function in order management except the above six functions, including weekly revenue summary, weekly gym usage summary, sending receipt emails, and confirming orders.

1. staticsPrice

*Request type:* GET

*Function description:* Calculate weekly income.

*Input parameter:* null

*Output parameters:* dataMap(Map)

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. staticsStadium

*Request type:* GET

*Function description:* Calculate the weekly usage time of stadium.

*Input parameter:* null

*Output parameters:* dataMap(Map)

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. sendPdfToEmail

*Request type:* GET

*Function description:* Send email.

*Input parameter:* id(String)

*Output parameters:* “Send Email Success”/error

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. confirmOrder

*Request type:* GET

*Function description:* The staff confirms the order for cash payment.

*Input parameter:* id(String)

*Output parameters:* “confirm Success”/error

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

## 10.4 TimetableService

The following content is the remaining function in timetable management except the above six functions: obtaining timetable by activity’s id.

1. getTimeTableByActivityId

*Request type:* GET

*Function description:* Obtain activity information through id.

*Input parameter:* id(String)

*Output parameters:* tableVO(ActivityTimeTableVO)

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

## 10.5 SysUserFaceController

The following content is about the functions of user login and logout, user registration, etc. in the software.

1. login

*Request type:* POST

*Function description:* Log in.

*Input parameter:* loginDTO(LoginDTO)

*Output parameters:* Login Success/error

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. captcha

*Request type:* POST

*Function description:* Generate captcha.

*Input parameter:* null

*Output parameters:* an image with captcha

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. roles

*Request type:* POST

*Function description:* Get Role List.

*Input parameter:* null

*Output parameters:* role\_list(Array)

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. currentUser

*Request type:* GET

*Function description:* Get current user

*Input parameter:* null

*Output parameters:* username(String)

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. logout

*Request type:* GET

*Function description:* Logout.

*Input parameter:* null

*Output parameters:* “Logout Success”/error

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

1. register

*Request type:* POST

*Function description:* Register.

*Input parameter:* registerDTO(RegisterDTO)

*Output parameters:* “Register Success”/error

*Return value:*

* SUCCESS(200, "success")
* FAIL(201, "fail")
* TOKEN\_ERROR(202, "fail")

*Exception handling:*

If the interface call fails, a status code and error message will be returned.

## 10.6 FilesServiceFaceController

Control the upload, download, and deletion of files in the software.

1. upload

*Request type:* POST

*Function description:* Upload file.

*Input parameter:* files(MultipartFile[]), fileTypeEnum(FileTypeEnum)

*Output parameters:* null/error message

*Return value:* filesInfo(List<SysFile>)

*Exception handling:*

If the interface call fails, an error will be threw and an error message will be returned.

1. uploadSeccurity

*Request type:* POST

*Function description:* Upload file.

*Input parameter:* fileDTO(FileDTO)

*Output parameters:* null/error message

*Return value:* filesInfo(List<SysFile>)

*Exception handling:*

If the interface call fails, an error will be threw and an error message will be returned.

1. download

*Request type:* GET

*Function description:* Download file.

*Input parameter:* fileID(String)

*Output parameters:* null/error message

*Return value:* null

*Exception handling:*

If the interface call fails, an error will be threw and an error message will be returned.

1. delete

*Request type:* DELETE

*Function description:* delete file.

*Input parameter:* fileID(String)

*Output parameters:* null

*Return value:* T/F

*Exception handling:*

If the interface call fails, an error will be threw and an error message will be returned.

# System error handling design

This software is an application used to help users make reservations for gym stadiums. In this application, errors are inevitable. In order to enable users to better use the software, it is necessary to design the system's error handling to ensure timely, accurate, and effective response when errors occur. This article will introduce the error handling design of the gymnasium reservation software system from the following aspects.

## 11.1 Error classification:

When designing system error handling, it is first necessary to classify the errors. Errors can be classified into the following categories based on their severity:

1. Blocker: The system cannot function properly. Issues that hinder development or testing work; Causing system crashes, crashes, loops, database data loss, incorrect connection to the database, loss of main functions, and missing basic modules. For example, code errors, dead loops, database deadlocks, and the inability to use important first level menu functions.
2. Critical: An obvious erroneous bug. The main functions of the system are partially lost, database save calls are incorrect, user data is lost, and the primary function menu cannot be used, but it does not affect the testing of other functions. The module cannot be started or called due to serious discrepancies between functional design and requirements, program restart, automatic exit, conflicting calls between related programs, security issues, stability, etc. For example, after saving data in the software, there are errors displayed in the database, missing functions required by the user, program interface errors, numerical calculation and statistics errors, etc.
3. General: Common bugs. The function is not fully implemented but does not affect usage. The function menu has defects but does not affect system stability. For example, long operation time, long query time, formatting errors, boundary condition errors, deletion without confirmation box, excessive fields in database tables, etc.
4. Interface and performance defects, suggested issues that do not affect the execution of operational functions, and solutions that can optimize performance. For example, typographical errors, non-standard interface format, overlapping page displays, hiding inappropriate content, unclear descriptions, missing prompts, untidy text arrangement, incorrect cursor position, poor user experience, and solutions that can optimize performance.

## 11.2 Error handling process:

When dealing with errors, it is necessary to define a specific processing flow for each type of error. The following is the error handling process that may be involved in the software:

1. Input parameter error handling: If the user enters incorrect parameters, such as incorrect date format, invalid address, etc., a clear prompt should be provided on the page and the user should be asked to re-enter.
2. Network connection error handling: If there is a problem with the network connection, an error prompt should be provided on the page and the user should be asked to try the operation again.
3. Database error handling: If a database error occurs, a log should be logged and an error report should be sent to the administrator. At the same time, when dealing with errors, data consistency and integrity should be taken into account to avoid data corruption in the database.
4. Unauthorized error: If there is an authentication error, the user should be prohibited from accessing and an error message should be displayed to inform the user that they do not have permission.
5. Http Request Method Not Supported error: If there is an unsupported request error, an error message should be displayed, informing the user that the login has expired and requesting a new login.

## 11.3 Error prompt design:

When dealing with errors, it is necessary to provide appropriate error prompt information for different types of errors. The following are error messages:

1. Input parameter error prompt: The input parameter error prompt should be simple and clear, and indicate which input parameter is incorrect, so that users can quickly locate and solve the problem.
2. Network connection error prompt: The network connection error prompt information should include error codes and error messages, so that users can obtain sufficient information to solve the problem.
3. Database error prompt: The database error prompt should indicate which operation caused the error, provide recovery suggestions, and contact the administrator.
4. Other error: The current error situation should be clearly pointed out, and the user should be informed of how to handle the error or wait for the error to be fixed.

## 11.4 Logging:

When the system encounters an error, it is necessary to log and send an error report to the administrator. The following are the log records and report contents:

1. Record error information: It is necessary to record detailed information such as error type, error time, error code, error message, etc., so that administrators can quickly locate and solve problems.
2. Sending error report: It is necessary to send the error report to the administrator and resolve the issue as soon as possible to ensure stable system operation.
3. Database design

## 12.1 Introduction

This document typically provides an overview of the purpose and scope of the database, as well as a summary of the key features and functionality. It may also describe the intended users of the database and any relevant business or technical requirements.

It describes the design of a database, including its structure, data model, and relationships between tables. It also provides a clear and detailed description of the data

base design, enabling developers, stakeholders, and other interested parties to understand how the database works and how it can be used.

## 12.2 Database name principle

* Use descriptive names: Use names that accurately describe the purpose and function of the database or database object.
* Be consistent: Use a consistent naming convention for all databases and database objects to make it easier to understand and manage the database.
* Avoid abbreviations: Use full words or phrases instead of abbreviations to avoid confusion and make it easier to understand the purpose of the database or database object.
* Use CamelCase or underscores: Use CamelCase or underscores to separate words in a name, making it easier to read and understand.
* Avoid reserved words: Avoid using reserved words in database and object names, as this can cause conflicts and errors.
* Keep names simple and concise: Use short, simple names that are easy to understand and remember.

## 12.3 Database design principle

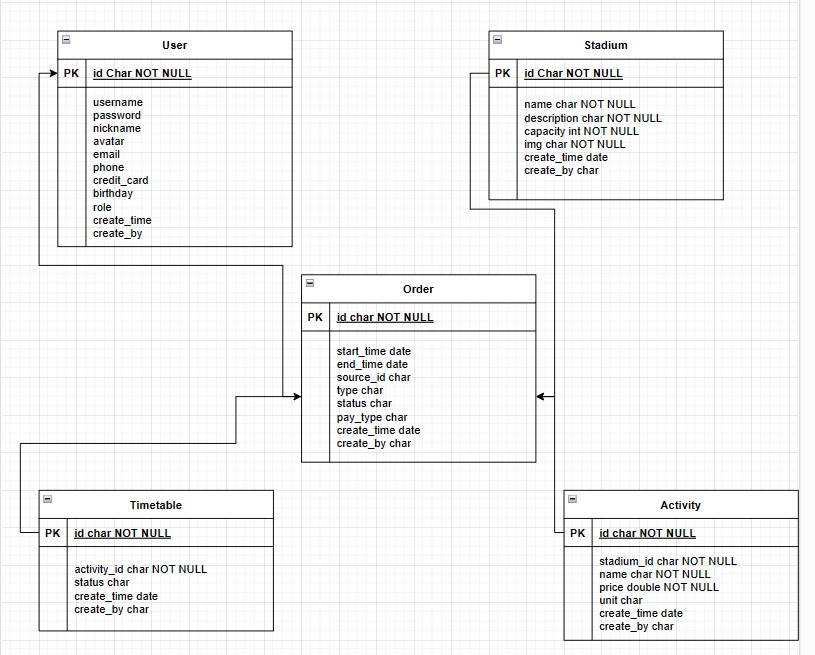
* Normalization: This principle involves organizing data in a way that minimizes redundancy and dependency. Normalization helps to ensure that data is stored in a consistent and efficient manner, and that updates and modifications to the data are easier to manage.
* Entity-Relationship Modeling: This principle involves identifying the entities (objects or concepts) that are relevant to the application and defining the relationships between them. This model helps to ensure that data is organized in a logical and consistent way.
* Data Integrity: This principle involves ensuring the accuracy and consistency of data by implementing constraints and rules that prevent invalid data from being entered into the database.
* Performance: This principle involves designing the database in a way that optimizes performance. This includes considerations such as indexing, partitioning, and caching.
* Security: This principle involves implementing measures to protect the database from unauthorized access, including authentication, authorization, and encryption.
* Scalability: This principle involves designing the database in a way that allows it to handle increasing amounts of data and traffic over time. This includes considerations such as replication and load balancing.
* Flexibility: This principle involves designing the database in a way that allows it to adapt to changing requirements and data structures over time. This includes considerations such as schema evolution and versioning.

## 12.4 Database logical design

It is the process of creating a conceptual representation of a database that reflects the business requirements and data relationships. The logical design focuses on the logical organization of data elements and their relationships, and is independent of any specific database management system or physical implementation.

It is an important step in the database design process, as it provides a blueprint for the database that reflects the business requirements and data relationships. The logical design serves as the basis for the physical design

There are 5 tables in database totally, User table, Stadium table, Activity table, Timetable table, Order table. With its own ids and characteristics. The ER diagram is shown below:



## Database physical design

1. DBMS system selection: Mysql

MySQL is a popular open-source relational database management system that provides a powerful and flexible way to store and manage data. Here are some reasons why you might choose to use MySQL:

* Open source: MySQL is open-source software, which means that it is free to use and can be modified by anyone to suit their needs.
* Scalability: MySQL can scale to handle large amounts of data and high levels of traffic, making it suitable for use in enterprise-level applications.
* Compatibility: MySQL is compatible with many different operating systems, programming languages, and development frameworks, making it a versatile choice for developers.
* Performance: MySQL is designed to be fast and efficient, with features like caching and indexing that can help improve query performance.
* Security: MySQL provides robust security features to protect your data, including encryption, access controls, and user authentication.
* Reliability: MySQL is a reliable database system, with features like automatic backups and recovery that help ensure that your data is always available when you need it.

1. Storage structure design

The storage design of each table is displayed in the following forms:

|  |  |  |  |
| --- | --- | --- | --- |
| **User** | Name | Type | Length |
| id | varchar | 64 |
| username | varchar | 64 |
| password | varchar | 64 |
| nickname | varchar | 64 |
| avatar | varchar | 255 |
| email | varchar | 64 |
| phone | varchar | 64 |
| credit\_card | varchar | 64 |
| birthday |  |  |
| role | varchar | 64 |
| Create\_time |  |  |
| Create\_by | varchar | 64 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Stadium** | Name | Type | Length |
| id | varchar | 64 |
| name | varchar | 64 |
| description | longtext |  |
| capacity | int |  |
| img | varchar | 255 |
| create\_time | datetime |  |
| create\_by | varchar | 64 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Activity** | Name | Type | Length |
| id | varchar | 64 |
| stadium\_id | varchar | 64 |
| name | varchar | 64 |
| price | double |  |
| unit | varchar | 64 |
| create\_time | datetime |  |
| create\_by | varchar | 64 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Timetable** | Name | Type | Length |
| id | varchar | 64 |
| activity\_id | varchar | 64 |
| status | varchar | 64 |
| create\_time | datatime |  |
| create\_by | varchar | 64 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Order** | Name | Type | Length |
| id | varchar | 64 |
| start\_time | datetime |  |
| end\_time | datetime |  |
| source\_id | varchar | 64 |
| type | varchar | 64 |
| status | varchar | 64 |
| pay\_type | varchar | 64 |
| create\_time | datetime |  |
| create\_by | varchar | 64 |

1. Database distribution

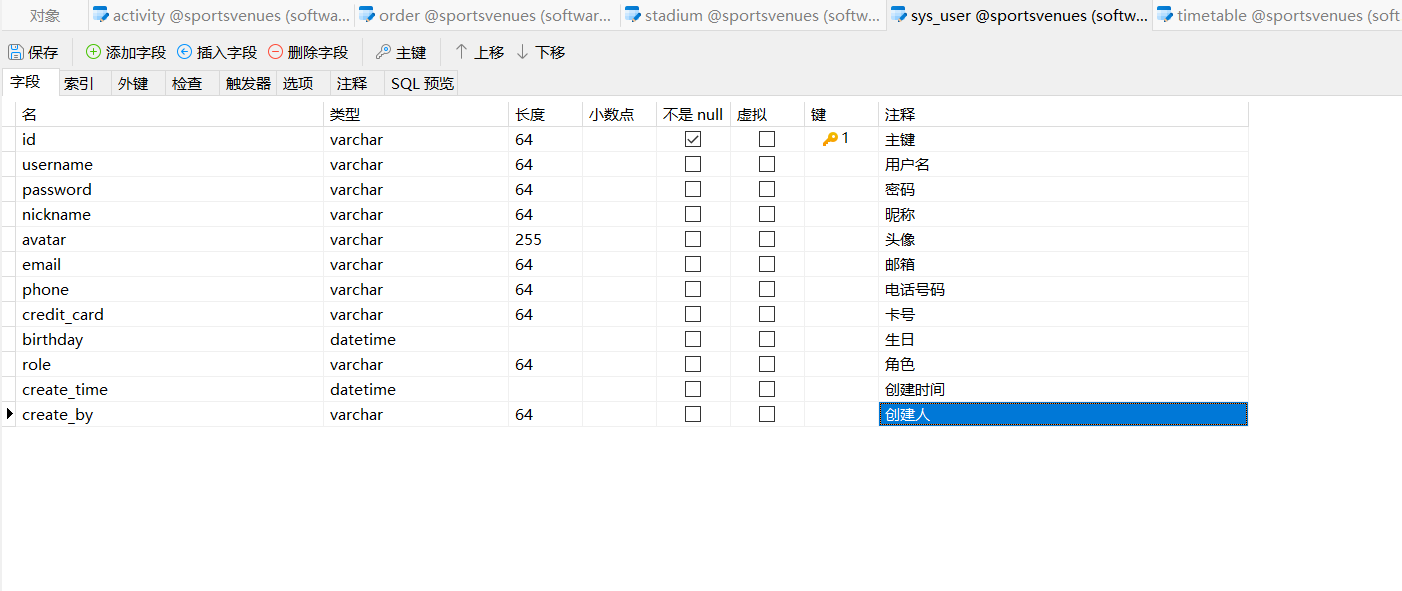
|  |  |
| --- | --- |
| Database number | Database name |
| 01 | User |
| 02 | Stadium |
| 03 | Order |
| 04 | Timetable |
| 05 | Activity |

1. Table design

**01 User**

This database table appears to represent a user account system, for a web or mobile application. The table contains several fields which store information about each user, including their unique ID, username, password, nickname, avatar image URL, email address, phone number, and credit card information. The "birthday" field is intended to store the user's date of birth, although the data type is not specified. The "role" field suggests that this system may have different levels of user permissions or privileges, with each user assigned a specific role. The "Create\_time" field stores the timestamp of when the user account was created, and the "Create\_by" field may store the ID or username of the user who created the account (such as an administrator or employees).

Overall, this table provides a comprehensive set of fields for storing user data, although it is worth noting that sensitive information such as passwords and credit card details should be stored securely and encrypted, and appropriate measures are taken to ensure the privacy and security of user data.

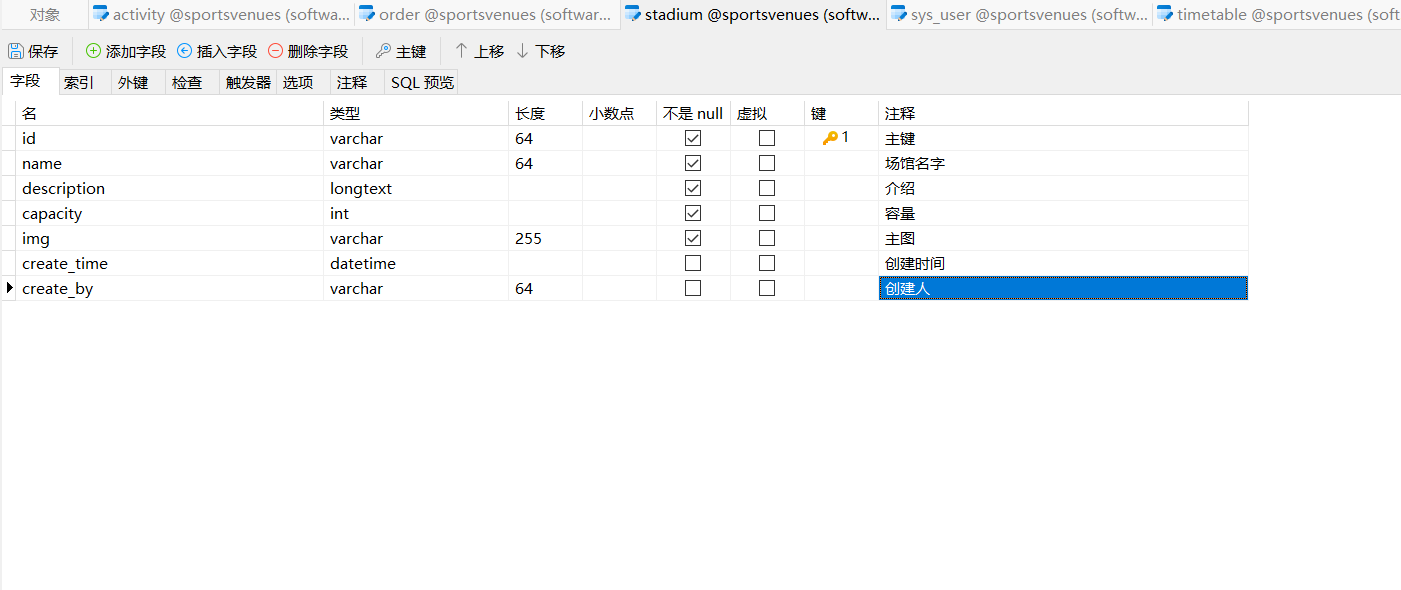


**02 Stadium**

This database table represent a collection of stadiums for the sports centre system. The table contains several fields which store information about each stadium, including a unique ID, the stadium's name, a description field which allows for longer text, the stadium's seating capacity, and an image field which likely contains a URL or file path to an image of the stadium.

The "create\_time" field stores the timestamp of when the stadium record was created, and the "create\_by" field store the ID or username of the user who created the record (such as an administrator or event manager).

Overall, this table provides a basic but useful set of fields for storing information about stadiums. Depending on the specific use case, additional fields could be added to the table to store more detailed information about each stadium, such as location, contact information, or event scheduling data.

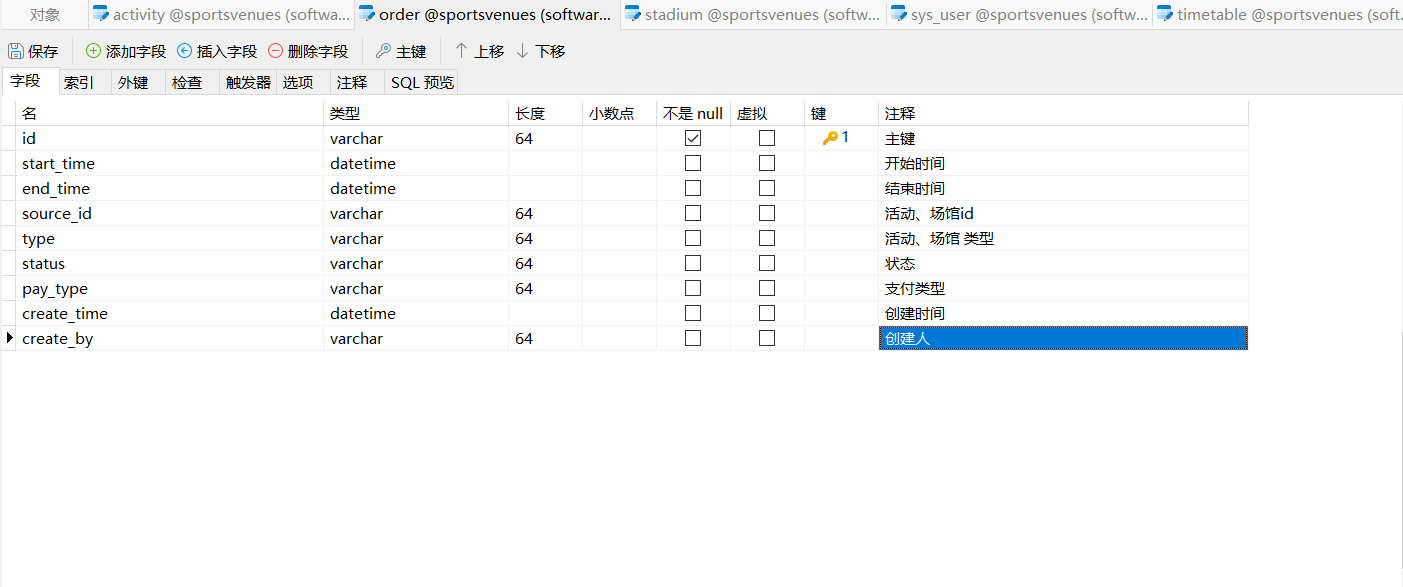


**03 Order**

This database table represents a collection of orders for the sports centre system. The table contains several fields which store information about each order, including a unique ID, the start and end times of the order, the ID of the source (such as a customer or booking agent), the type of order (such as a product or service), the status of the order, and the payment type used for the order.

The "create\_time" field stores the timestamp of when the order record was created, and the "create\_by" field stores the ID or username of the user who created the record (such as a customer service representative or booking agent).

Overall, this table provides a basic but useful set of fields for storing information about orders. Depending on the specific use case, additional fields could be added to the table to store more detailed information about each order, such as shipping or delivery information, customer contact details, or inventory tracking information.

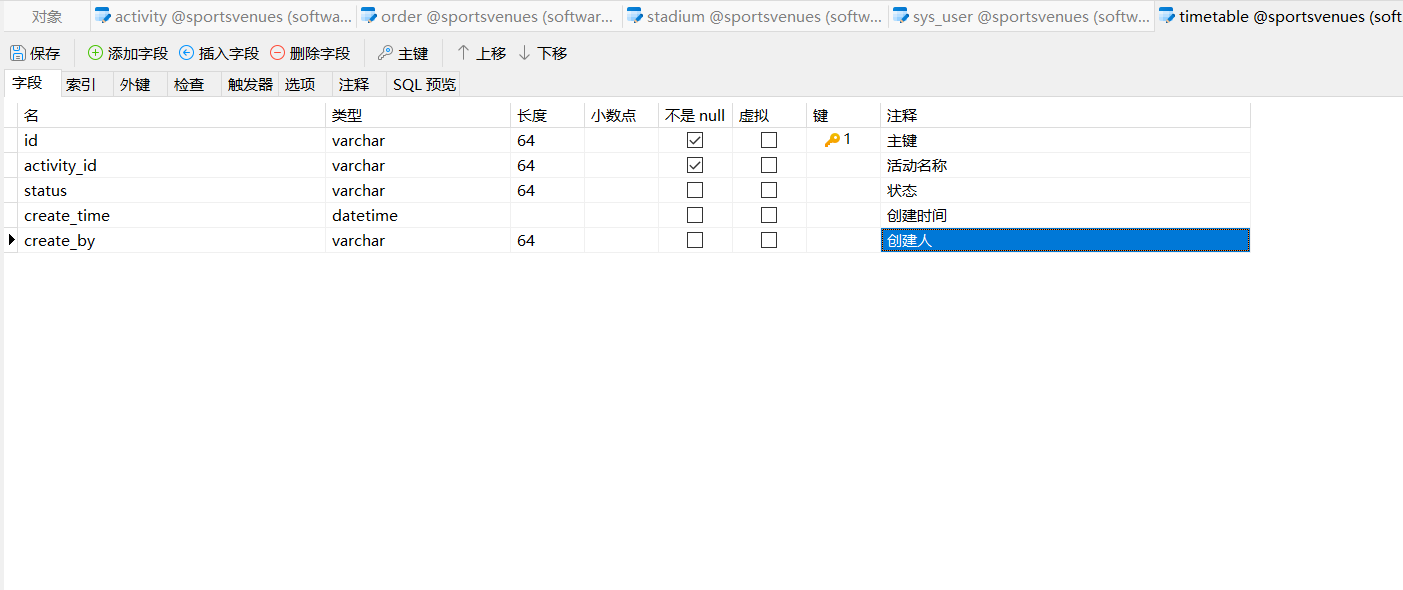


**04 Timetable**

This database table represents a collection of timetables for the sports centre system. The table contains several fields which store information about each timetable, including a unique ID, the ID of the activity being scheduled (referencing the "activity\_id" field in the Activity table), the status of the timetable (such as "confirmed", "pending", or "cancelled"), and the timestamp of when the timetable record was created.

The "create\_by" field store the ID or username of the user who created the timetable record (such as an administrator or event manager).

Overall, this table provides a basic set of fields for storing information about timetables or schedules for activities at a venue. Depending on the specific use case, additional fields could be added to the table to store more detailed information about each timetable, such as start and end times for each activity. Additionally, there may be a need for a separate table to store information about bookings or reservations for each activity, which could be linked to the timetable via the activity ID.



**05 Activity**

This database table represents a collection of activities for the sports centre system. The table contains several fields which store information about each activity, including a unique ID, the ID of the stadium where the activity is taking place (referencing the "stadium\_id" field in the Stadium table), the name of the activity, the price of the activity, and the unit of measurement for the price (such as "per hour" or "per person").

The "create\_time" field stores the timestamp of when the activity record was created, and the "create\_by" field store the ID or username of the user who created the record (such as an administrator or event manager).

Overall, this table provides a basic but useful set of fields for storing information about activities taking place at a venue. Depending on the specific use case, additional fields could be added to the table to store more detailed information about each activity, such as start and end times, age restrictions, or equipment requirements. Additionally, there may be a need for a separate table to store information about bookings or reservations for each activity.

