Keep Fit

Team Member:

Weicheng Zheng 2154286

Juekai Lin 2253744

Lixin Ma 2153085

Junhao Yang 2154284

Instructor: Ping Sun

LIFE LIES IN MOVEMENT

ENJOY YOUR ROMANTIC TIME WAKE UP THE FRAGRANT WINE THE SECRET OF BEAUTIFUL MEETING OURS JOIN

Content

1. Introducion

1.1. Project Purpose

This project KeepFit aims to streamline the fitness journey with an integrated solution that combines diet plans, exercise routines, and equipment management. By providing personalized diet and workout programs, which dynamically adjust to each user's fitness schedule, it helps users pursue their goals more intelligently and efficiently. The equipment management feature guides users in selecting and optimizing fitness equipment usage. Through smart recommendations and real-time feedback, the project encourages healthy habits, supports the nationwide fitness movement, and fosters the adoption of a healthier lifestyle.

This System Design Model Document (SDM document) aims to comprehensively explain the design model and architecture of the KeepFit system. The document details the overall structure of the system, including the functional modules of each subsystem, the interface relationship, and the data flow process. In addition, the document explores in depth the specific mechanisms of system prototyping, covering all stages, from requirements analysis to system implementation. In order to help to better understand and apply the design models, the document also provides some practical examples, showing how to flexibly configure and expand according to the specific business requirements. These contents will provide clear guidance for development, implementation and subsequent maintenance, and ensure the efficient and stable operation of the system.

1.2. Project Scope

Keep-Fit is a Web-based smart fitness software, and the goal is to help users achieve their fitness goals smarter and more effectively. The main scenarios of the system include the selection of fitness tutorial, punching schedule, fitness posture guidance, food equipment recommendation and AI robot chat, etc. VIP members and ordinary users can browse information and participate in the activity after registering and logging in, and choose whether to become uploader. In addition, the platform administrators can maintain the platform order and keep the platform fitness content up to date.

In addition, Keep-Fit also has the following characteristics:

- The platform system operates in a network environment.
- The system has an accurate recommended fitness tutorial algorithm.
- The intelligent system allows users to chat and consult with the AI.
- Have a safe and reliable database to ensure information security.

1.3. Glossary of Terms

English terms	Terminology interpretation
Keep-Fit	The name of our project, the intelligent fitness platform, is designed to help users achieve their fitness goals smarter and more effectively.

English terms	Terminology interpretation
User	The most basic users of the intelligent fitness platform can access the platform through registration, and use the most basic functions of the platform.
VIP Member	Ordinary users can become VIP members by recharging, and VIP members can use the richer functions in the platform, such as some paid tutorials, more intelligent AI chat systems and more accurate recommendation systems.
Uploader	Each kind of user(Users and VIP Members) can become a Uploaders by uploading exercise tutorials and sharing fitness experiences, and the Uploaders can also gain revenue and fans by selling paid fitness tutorials.
Administrator	The administrator needs to maintain the order of the platform and the stability of the environment, and can ban the illegal users. You can also make announcements, upload new fitness tutorials, diet combinations and fitness equipment, and so on.
Fitness Tutorial	Fitness tutorials are uploaded by uploaders or administrators, including paid and free, and cover a variety of fitness categories.
Check-in	After the user selects a fitness tutorial, the system automatically generates daily tasks and plans for the tutorial (the user can also plan the tutorial by himself/herself). Users need to complete the tasks every day to check in.
AI Motion Detection	Users can upload photos of themselves during their workout process, and the system's AI will automatically check whether the exercise movements meet the specifications and give appropriate suggestions.
Nutritional Composition	After the user selects a fitness tutorial, the system will automatically generate the recommended diet package of the tutorial, and give the nutritional composition of the package. Users can also take photos and upload daily diet pictures, and the AI system will automatically identify and give the nutritional content list of the diet.
Reward Mechanism	Users need to complete the corresponding tasks every day according to the daily plan generated by the system(or tasks planning by himself/heself). After completing the tasks, they can get virtual rewards (such as level growth and platform transaction currency).
Report	Administrators regularly make announcements to various users of the platform, including some important notifications, new tutorials, and the latest feature points on the platform.
Fitness Equipment	Each fitness course is accompanied by a series of required fitness equipment, the system will give the corresponding purchase links and size size recommendations.
Chat Platform	In the chat platform, users can chat not only with their friends, but also with the system's AI robot, which can answer anything about fitness intelligently.
Multi-device Access	Users can access and use the functions of the platform through different devices (such as personal computers, tablets, smartphones, etc.). This flexible access approach can improve the user experience and engagement.
Intelligent Recommendation	The system automatically generates personalized suggestions based on the users' personal information, fitness goals, history, and preferences.

1.4. Progress on System Design

In the previous report, we have completed the analytical model design of the KeepFit intelligent Fitness system, and presented in detail the user interface, system architecture and class diagram in this section. Based on the results of these preliminary work, this document further clarifies and improves the micro-service architecture design of the platform. We will provide more specific technical architecture and logical architecture models, and design detailed interfaces for the different modules of the project. The paper will also thoroughly describe the use of some third-party API, the design of an interface of the sub-system, the implementation of the analysis mechanism, the specific operation of the use case and the design of the system prototype. We focused on the specific implementation design of the project, elaborated on the main technology stack needed to achieve the project, and defined the specific way in which the project was implemented.

The improvement and enhancement of the platform design are mainly reflected in the following aspects:

- **Reconfigure the logical structure and functional mechanisms**

Through in-depth analysis of the mechanism and functions of the platform, we redesigned the logical structure of the system, and optimized and adjusted the various functions of the system to meet the actual needs of the project.

- **Add technical and physical architecture design**

In addition to the original logical architecture, we also further expand the design of the technical architecture and the physical architecture. For each function of the layer, we detail the required technology stack and provide the deployment scheme of the system at the physical level to ensure the efficient operation of the system.

- **Optimize the interface design and detailed description**

We have carried out more detailed planning in interface design, providing new subsystems and various modules with more specific interface design, including the type of interface parameters, return value type and details of data transmission. For the interfaces of third-party services and some subsystems, we have also made detailed instructions to ensure the smooth and stable interface call.

- **Improve the safety and reliability of the system**

To improve the security and reliability of the platform, we have adopted the API gateway, the data persistence mechanism, and a range of design solutions. These designs effectively enhance the fault tolerance, flexibility, and scalability of the system, ensuring that the platform operates stably in a high-concurrency, high-load environment.

- **Analyze the mechanism and select the appropriate design mode**

We have made an in-depth analysis of the mechanism in the platform and selected the most suitable design mode to improve the performance and scalability of the system and ensure the scientific and high efficiency of the system design.

- **System analysis and design by using the design mode**

In the analysis and design process of the system, we have widely used a variety of design modes to ensure that the system has good maintainability, scalability and high efficiency.

- **Practice part of the project development work and propose the prototype design**

In the actual development process, we personally participated in the construction of some project modules, carried out the development and implementation of some functions, and proposed the system prototype design, so as to better verify the feasibility and user experience of the system.

- **Combine the user use case and the system structure analysis and implementation mode**

By combining user use cases with the system structure of the platform, we deeply analyzed the implementation process of the system to ensure that each use case can be efficiently connected with the system architecture, so as to improve the overall performance and user experience of the system.

Through these improvements and enhancements, this design document not only provides a more detailed and clear diagram of the system architecture, but also ensures that the platform can operate efficiently during the implementation process to meet the needs of future expansion and innovation.

2. The Platform and Architecture Styles

2.1. Platform and Architecture

In the actual development process, we will use a microservice architecture to develop our KeepFit intelligent fitness system. The architecture enables each microservice in the system to be deployed independently and loosely coupled to each other. Each microservice focuses on completing a single function and implementing it in an optimum way. This architecture design brings low coupling, high flexibility, specific solutions to specific problems, and is conducive to independent development. In addition, the microservice architecture also enhances the availability and stability of the system. Under this architecture, we will use a series of mature technologies and frameworks to build the system. The implementation of each subsystem and component is summarized as follows:

- **Web application**

We will use Flutter, Material-UI, Panache, Supernova, Sylph, Codemagic and other tools to build the front-end framework, and use AJAX technology to dynamically update the page content without refreshing the page to improve the user experience.

- **API gateway**

The API gateway will be implemented through Spring Cloud Alibaba, as a unified external interface to provide microservice support. Spring Cloud Alibaba provides key components for building distributed applications, allowing developers to easily develop and manage various services in the microservice architecture through the Spring Cloud programming model.

- **Security**

The security certification of the system will use Sa-Token, a lightweight licensing framework that can meet the security requirements of the platform. Certification and authorization through this framework can effectively guarantee the security of the platform and support security testing.

- **Data storage**

In terms of data storage, we will combine a variety of database technologies. The relational database MySQL is used to store structured data, the document database MongoDB is used to store unstructured data, MinIO will be used for object storage, and Redis provides efficient cache and data consistency support in high concurrency scenarios to ensure the security and high availability of data.

- **Service framework**

Backend development will be based on Spring Boot, using its strong scalability and excellent performance to build server systems. We will develop using the Java language, and Spring Boot provides the ideal environment for developing high-performance, scalable network applications.

Through the integrated application of these technologies and frameworks, we are able to build an efficient, stable and highly scalable microservice architecture system to meet the various needs of the KeepFit intelligent fitness platform.

2.2. Architecture Style

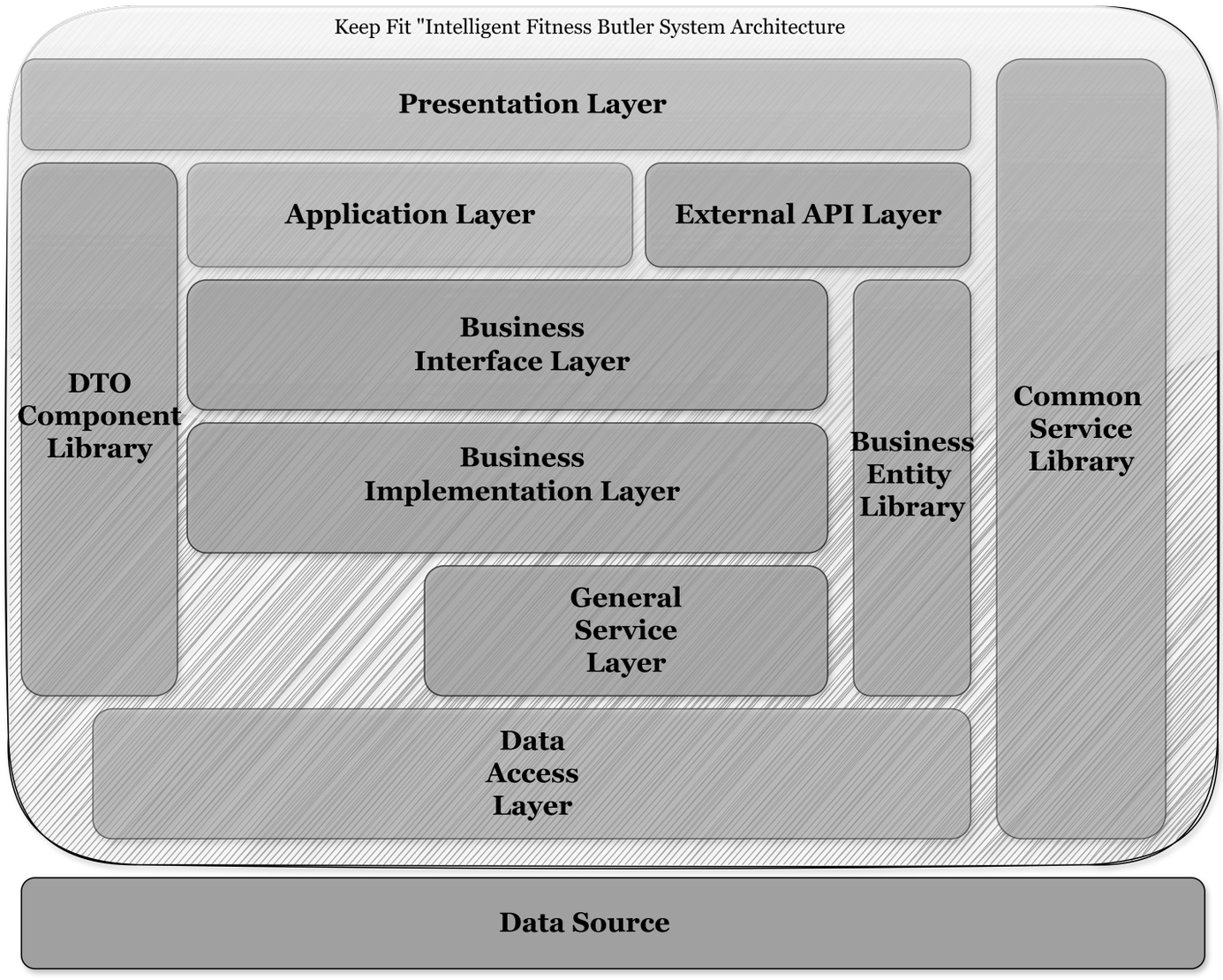
In the modern software architecture design, we abandon the traditional single architecture and instead adopt a more flexible and efficient microservice architecture. The core idea of this architecture style is to deconstruct complex systems into multiple lightweight, independent and autonomous service units, with each micro-service focused on performing specific business functions. With this decentralized design approach, we significantly improve the overall performance and resilience of the system. Unlike traditional single large applications, the micro-service architecture allows development teams to respond more quickly to changing business requirements, enabling rapid iteration of technology and business through loosely coupled service design. Each micro-service is a relatively independent business boundary, with its own data storage, business logic and communication interface, thus forming a highly decoupled and portfolio system ecosystem.

Due to the high completeness and independence of the services provided by HW, it can be conveniently packaged into different microservice clusters and applied to other systems. Therefore, the system adopts the micro-service architecture design, with the characteristics of high cohesion and low coupling, and each service can be deployed independently. The microservice architecture also has the advantages of flexible development, support for rapid iteration and update, which can well meet the functional requirements of the current system. In the microservice architecture, we conduct an in-depth analysis of the design of individual applications, as described below.

According to the classification of software architecture styles by Garlan and Shaw, software architecture styles are mainly divided into the following five categories:

- **Data flow style** including batch sequence and pipeline / filter mode.
- **Call / return style** including the master program / subprogram, object-oriented style, and hierarchy.
- **Independent component style** including inter-process communication and event-driven systems.
- **Virtual machine style** including interpreter and rule-based systems.
- **Warehouse style** including database system, hypertext system and blackboard system.

In designing the system, the overall architecture adopts a call / return style, with a focus on data-based abstraction and object-oriented design concepts. In the hierarchical design, we gradually expand from local to whole to ensure that the system has clear hierarchy and good scalability while meeting the requirements.



3. Design Patterns and Decisions

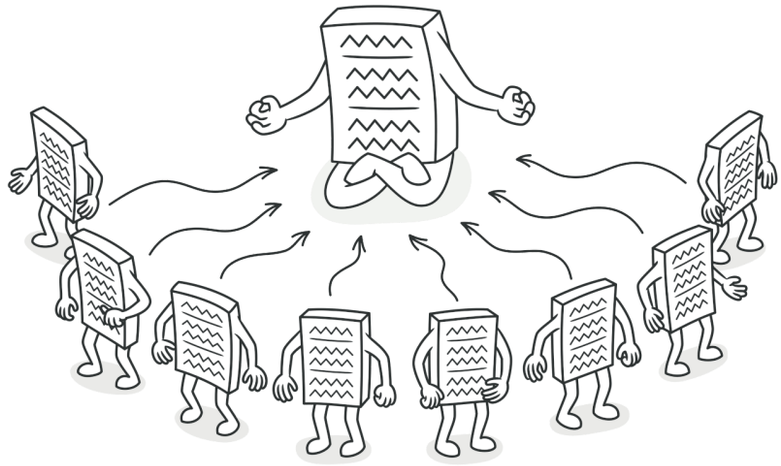
3.1. Design Patterns

In the development process of the intelligent fitness platform, we follow the following design patterns:

3.1.1. Singleton Pattern

The singleton mode is a common design mode, designed to ensure that a class has only one instance, and that it provides a global access point to obtain that instance. By using the singleton mode, you can avoid resource waste and management problems caused by creating multiple instances, especially for data or services that require global sharing.

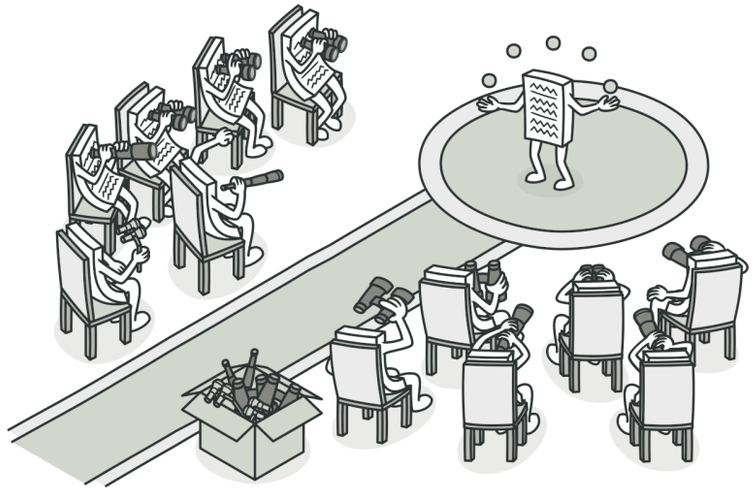
In our smart fitness system, we use the singleton mode to manage the data of the current users. After the user logs in, the current user information is saved in a singleton class to ensure that this unique user data is shared by other modules in the application without repeating user objects. At the same time, the default Settings of the fitness plan or the global system configuration (such as the reward system, AI detection rules, etc.) can be managed in the singleton mode to ensure the consistency and global accessibility of the configuration items.



3.1.2. Observer Pattern

The observer pattern is a behavioral design pattern that defines a one-to-many dependency that allows multiple observers to listen on the state changes of a single subject object. When the state of a subject object changes, all observers who depend on it are notified and automatically updated.

In our intelligent fitness system, the observer mode can be used for multiple scenarios, such as the user subscribing to their own target progress update, the fitness plan management module as the theme, and the user interface or the notification module as the observer, and the user can view the progress of the task in real time. When the user's fitness progress changes, multiple observers will receive notification and automatically update relevant content, such as reward distribution, data statistics, etc.



3.1.3. Strategy Pattern

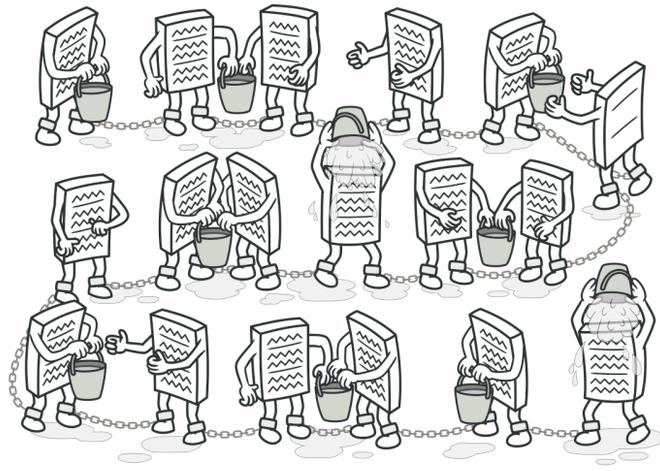
The strategy mode allows for the selection of the behavior of the algorithm at runtime. In strategy patterns, a series of algorithms is defined and each algorithm is encapsulated into a separate class so that they are interchangeable. The strategy mode makes the algorithm change independent of the customer using the algorithm.

In our intelligent fitness system, different recommendation algorithm strategies are selected according to users' needs and preferences. For example, users prefer to learn through a video tutorial and can choose a content recommendation strategy. The recommendation system can flexibly switch between different recommendation algorithms through the strategy pattern to improve the user experience.

3.1.4. Responsibility Chain Pattern

The responsibility chain mode is a behavioral design mode that forms a chain by connecting multiple processing objects in series to form a chain in which a client request passes along the chain until an object processes the request. The main purpose of this pattern is to give multiple objects the opportunity to process the request, thus avoiding the coupling between the request sender and the receiver.

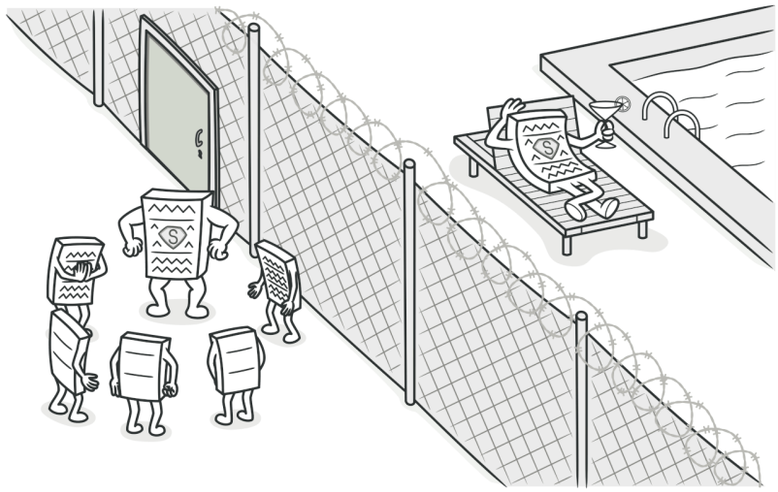
In the intelligent fitness system, the responsibility chain mode can be used to process a series of fitness tasks, user operation requests, data processing processes, etc. Especially when a request or operation needs to go through multiple processing steps, such as the training task audit process, the user feedback processing process, and the user training plan update process.



3.1.5. Proxy Pattern

The proxy mode provides a proxy for other objects to control the access to that object. The agent mode accesses the real objects indirectly by introducing the agent objects. The agent objects can perform some additional operations before and after accessing the real objects, such as permission control, delayed loading, caching, etc.

In smart fitness systems, video tutorials may require permission control. Only users who have purchased or unlocked the relevant tutorial can access the video content. Using proxy mode, proxy objects can control access to video tutorials, ensuring that only authorized users can watch it. At the same time, the proxy object can create an AI object on the first request and call the object directly in later requests, rather than creating every time, thus improving performance.



3.2. Critical Design Decisions

- In our system, all images and video resources uploaded by users are stored in Alibaba Cloud OSS and returned to the front-end for display through generated URLs. This approach not only provides high reliability and availability, ensuring secure and stable access to data, but also supports high concurrency processing and low latency, enhancing the user experience.
- The microservice architecture we use is extremely flexible, can adapt well to the agile development process, and has advantages beyond traditional architectures in terms of system reconstruction and continuous integration. In addition, the deployment and expansion of the micro-service architecture enable us to make full use of the advantages of the cloud platform in the operation and maintenance process, effectively control costs and improve the maintainability of the system.
- In every link of the system design, we always put the user privacy protection in the first place. In addition to the strict permission control based on sa-token, we also plan to implement encryption at the database level and adopt disk-level hardware encryption technology to ensure the security of user data. In the process of data collection and analysis, we will use desensitized data, so as to effectively protect user privacy.
- Using ElPlus and Vue to build our UI: Vue as a popular front-end framework provides an efficient development experience and a flexible component structure that can quickly respond to user needs. The introduction of ElPlus further improves the standardization and consistency of UI design, making the front-end interface more modern and user-friendly, thus bringing a smoother user experience.
- Based on the concept of object-oriented programming, combined with the domain-specific technology stack: our projects mainly use the Springboot technology stack for development and process most of the business logic. However, in some specific services, thanks to the flexibility of the micro-service architecture, we adopt a dedicated technology stack in the corresponding field to improve the development efficiency and system performance.

4. Architecture Refinement

4.1. Platform-dependent Architecture

In the previous logical architecture design of our Keep Fit Platform, we focused heavily on the completeness of internal system functions and the security of data transmission. However, considering the increasing number of users and business complexity, the original layered architecture has gradually exposed some problems, such as poor system scalability, low development and deployment efficiency, and wide impact of failures. Therefore, we have decided to adopt a microservices architecture based on Domain Driven Design (DDD) to improve and upgrade the system logic architecture.

- **Microservices Segmentation**

Our platform can be divided into multiple microservices that are both interrelated and relatively independent:

1. **Login and Registration Microservice**: Handles user login and registration functions.
2. **Fitness Tutorial Microservice**: Includes features such as fitness tutorial recommendations, learning, and fitness check-ins.
3. **AI Fitness Coach Microservice**: Includes features such as using AI to analyze whether users' fitness movements are standard, recommending fitness equipment, intelligent Q&A with fitness equipment, and intelligent price comparison for fitness equipment.
4. **Healthy Diet Microservice**: Includes features such as users setting meal plans based on system recommendations, recording dietary intake, and AI analysis of dietary habits.

- **Microservices Integration**

Previously, we had separated AI fitness movement analysis and fitness equipment into two subsystems. However, this time we have merged them into a single microservice. This is because these two functions are highly related in actual use. Merging them allows for better coordination and optimization of user workflows, reduces inter-service communication overhead, and improves overall performance.

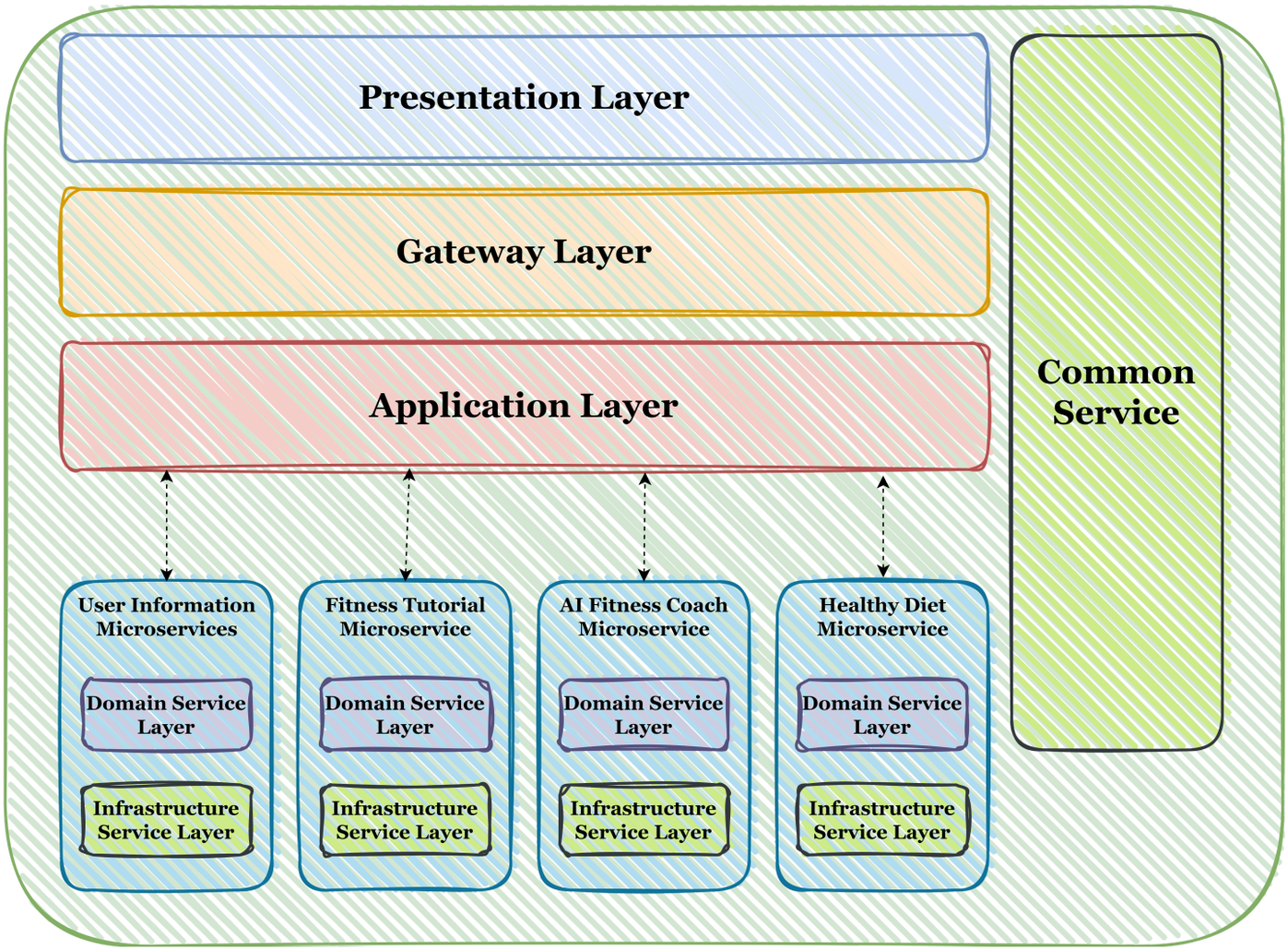
- **Interdependencies Between Microservices**

1. **Fitness Tutorial Microservice** influences the fitness equipment recommendation function in the **AI Fitness Coach Microservice**. When users learn specific fitness tutorials, the system recommends appropriate fitness equipment based on the tutorial content.
2. **Fitness Check-in Function** in the **Fitness Coach Microservice** must be validated by the **AI Fitness Coach Microservice** to ensure that the user's fitness movements are standard. Only when the user's movements meet the standards will the system record a successful check-in.
3. **Healthy Diet Microservice** recommends meal plans based on the ongoing fitness tutorials in the **Fitness Tutorial Microservice**. After AI analysis of the user's dietary habits, it intelligently adjusts the fitness plan in the **Fitness Tutorial Microservice**. For example, if a user is undergoing high-intensity training, the system might suggest increasing protein intake.

- **Advantages of Adopting a Microservices Architecture**

1. **Improved Development Efficiency**: Teams can develop different microservices in parallel, accelerating development and delivery speeds.
2. **Independent Deployment and Scaling**: Each microservice can be independently deployed and scaled without affecting other services, enhancing the system's flexibility and maintainability.
3. **Fault Isolation**: A failure in one service does not affect others, improving the system's stability and reliability.
4. **Continuous Delivery and Support**: Supports Continuous Integration and Continuous Delivery (CI/CD), facilitating rapid iteration and feature releases.

Based on the above content, the logical architecture of this system is as follows:

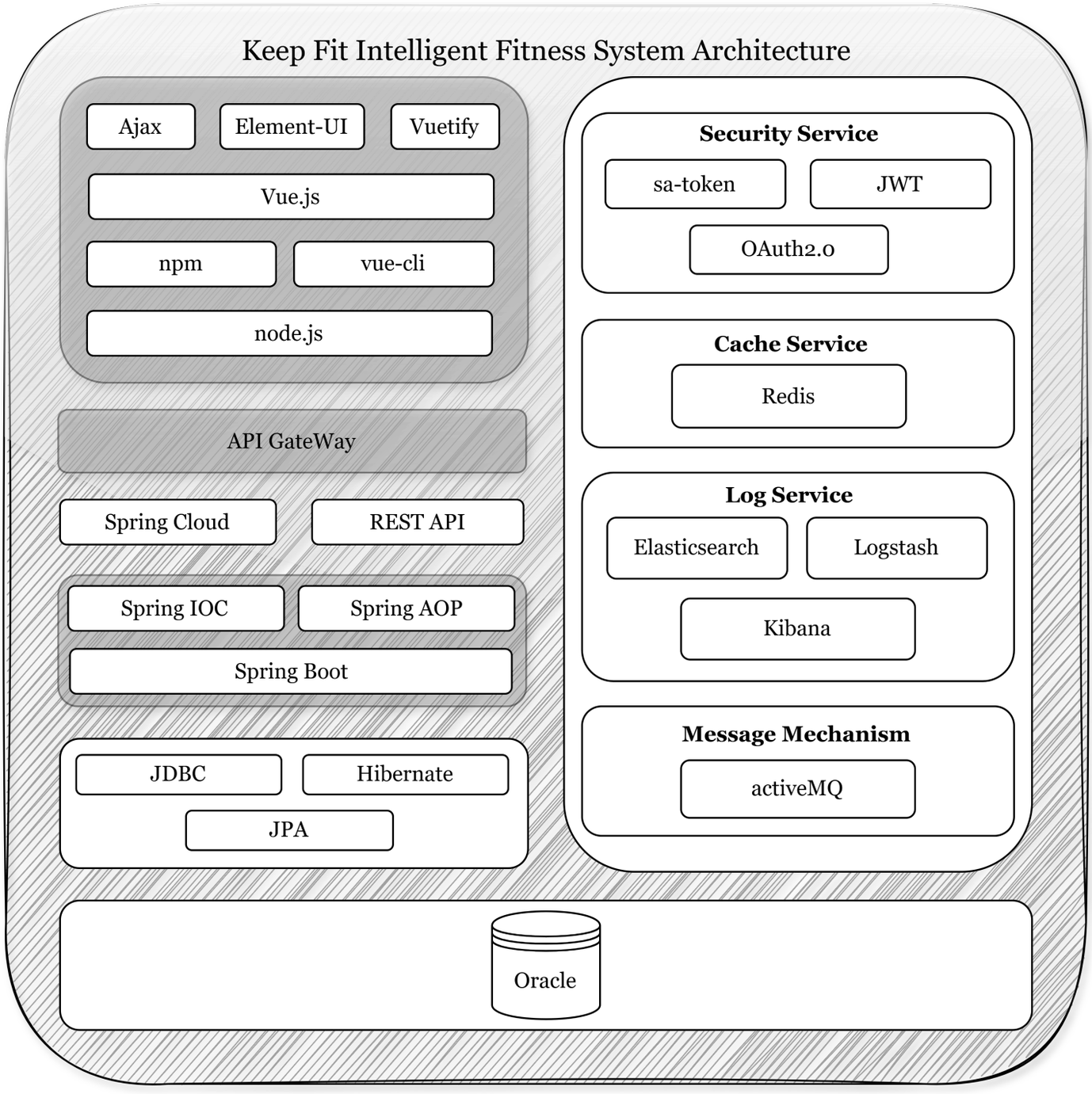


- **Technical Stack for the Microservices Architecture**

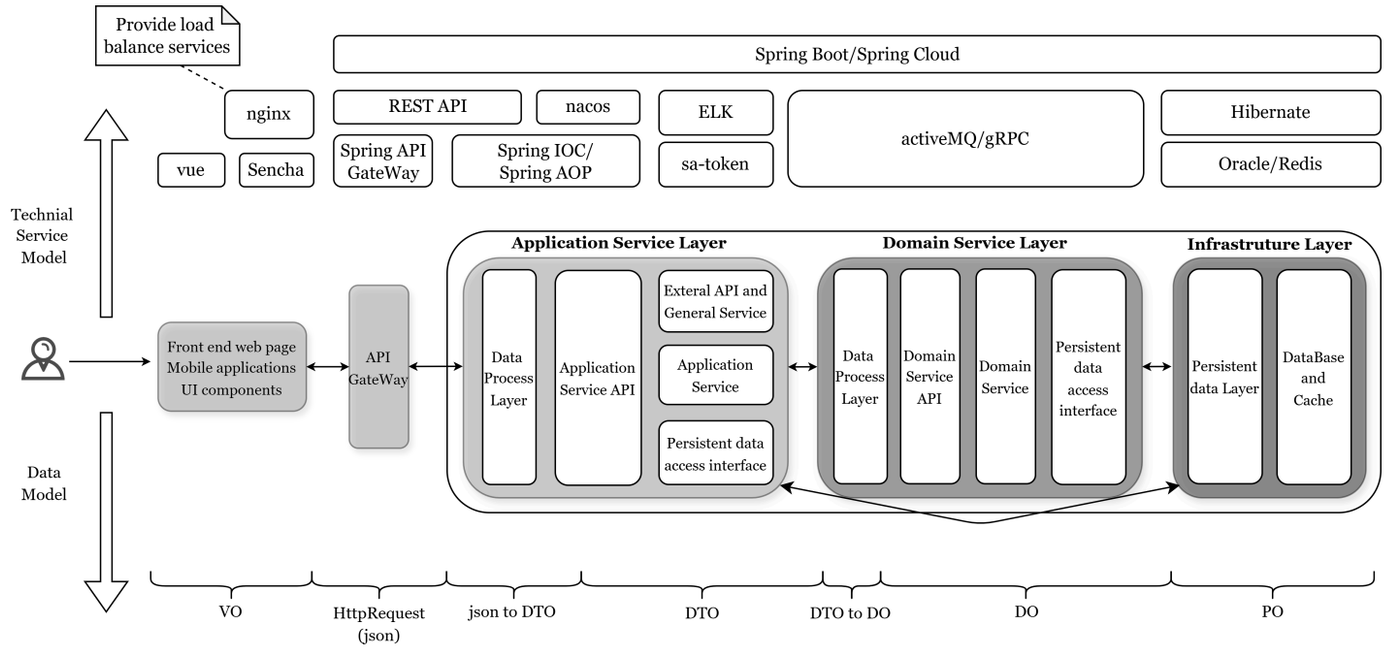
To ensure the microservices architecture scales effectively with the project size, we continue to use separate front-end and back-end development approaches. The front-end is developed using frameworks and libraries such as Vue and Sencha, while the back-end is built using Spring Boot and Spring Cloud. Communication between the front-end and back-end occurs through RESTful APIs and JSON data files. Data layers communicate and transform data using different models, such as Value Objects (VO), Data Transfer Objects (DTO), and Domain Objects (DO).

For authorization and authentication, the system uses sa-token. Comprehensive log collection and data analysis visualization are provided by the ELK stack. To efficiently handle asynchronous messages and traffic management, ActiveMQ is used for message passing and storage. The back-end persists data to Oracle databases or Redis caches using the Hibernate persistence mechanism.

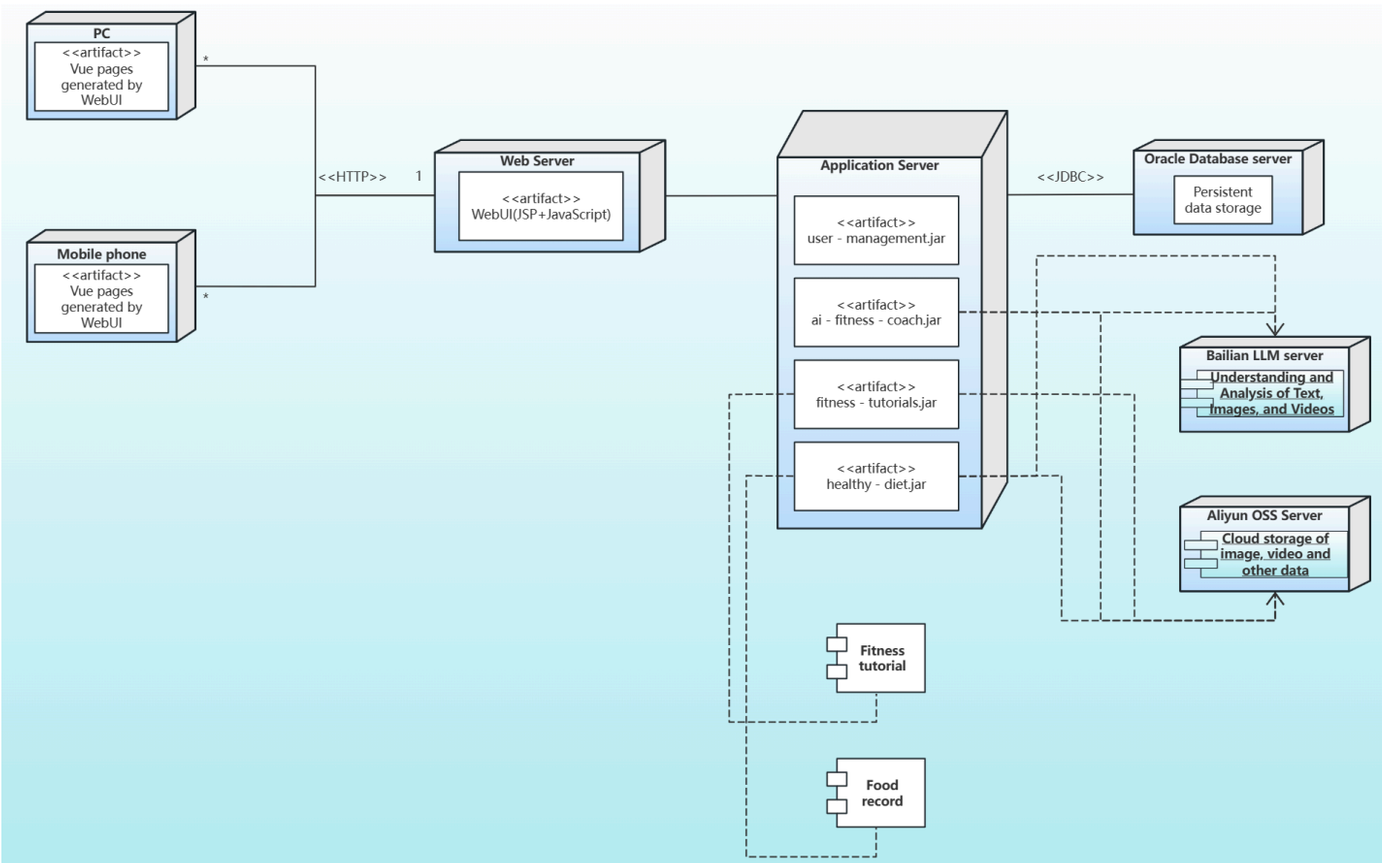
Based on these technologies, the technical stack diagram for the system is as follows:



Combining the technical architecture and the microservice architecture, we ensure the integrity and correctness of the system design, and improve the scalability and flexibility of the system by dividing the independent service modules. In the design process, we focus on the interface between the services, the data flow and the dependencies, and finally form the data and technical service model diagram:



The following shows the deployment diagram of the Keep Fit platform: It presents a multi - layer architecture. The top layer is the user access layer, including PCs and mobile phones, which communicate with the intermediate - layer Web Server through the HTTP protocol. The Web Server contains content generated by WebUI. The Web Server is then connected to the Application Server through the HTTP protocol. The Application Server is designed with multiple artifacts based on the division of microservices, namely user - management.jar, ai - fitness - coach.jar, fitness - tutorials.jar, and healthy - diet.jar. These artifacts are responsible for user management, AI fitness coaching, fitness tutorials, and healthy diet functions respectively. The Application Server conducts persistent data storage through JDBC with the Oracle Database. In addition, there is the Bailian LLM server for the understanding and analysis of texts, images, and videos, and the Aliyun OSS Server for cloud - based storage of images, videos, and other data. There are also two modules, Fitness tutorial and Food record, which interact with these servers.



4.2. Subsystems and Interfaces

According to the principle of domain-driven design, to ensure that the business logic of the system is implemented efficiently and accurately, we divide the individual business functions in the system into five main areas. This division not only helps to clarify the scope of responsibility in each field, but also ensures that the degree of coupling between services in each field is minimized, thus optimizing the overall structure and maintainability of the system.

- **Login and Registration Subsystem**
- **Fitness Tutorial Subsystem**
- **Fitness Action Coaching SubSystem**
- **Fitness Equipment SubSystem**
- **Healthy Diet Subsystem**

Based on the above domain design, in order to meet the demand for moderately granular services in the system design, we further divide the microservices into the following specific subsystems. This division is functionally oriented and aims to ensure that each subsystem can independently assume specific responsibilities, while maintaining low-coupling relationships with other subsystems.

- **Login and Registration Service Subsystem**
- **Fitness Tutorial Service Subsystem**
- **Fitness Action Coaching Servrice SubSystem**
- **Fitness Equipment Service SubSystem**
- **Healthy Diet Service Subsystem**
    - Dietary Record Module
    - Dietary Plan Module

4.2.1. Login and Registration Subsystem

API Interface	Method	Parameters	Description
/api/ua/login/passwd	POST	username, password	Log in using username and password, include user type (system).
/api/ua/logOut	POST	token	Log out, clear token and clean up permission cache.
/api/ua/login/wechatQR	POST	wechat_code	Log in using information from WeChat QR login.
/api/ua/user/info	GET	token	Get user information.
/api/ua/user/info	POST	token, info	Set user information.
/api/ua/user/setPasswd	POST	token, password	Set user password.
/api/ua/user/del	POST	token	Delete user.
/api/ua/user/register/new	POST	None	Create a temporary user during registration, return token information.
/api/ua/user/register/check	POST	toke	Check if the temporary user information is correct and if contact information is verified.

API Interface	Method	Parameters	Description
/api/ua/user/confirm	POST	verification	Call correctly to confirm the contact information is correct.
/api/ua/token/refresh	POST	token	Refresh token.
/api/ua/token/check	POST	token	Check if the token is valid.

4.2.2. Fitness Tutorial Subsystem

API Interface	Method	Parameters	Description
/api/tutorial/tutorialSearch	GET	keyword	Through this interface, users find the relevant fitness tutorial through keyword search and return a list of matching tutorials.
/api/tutorial/{tutorialID}/tutorialInfo	GET	tutorialId	The interface uses the user to obtain the details of the specified fitness course, including the course name, introduction, duration, difficulty, etc.
/api/tutorial/{tutorialID}/tutorialPurchase	POST	token, tutorialId	The user buys the specified unlocked tutorial through this interface to return whether the purchase is successful.
/api/tutorial/{tutorialID}/tutorialPlay	GET	token	Users can watch the purchased or unlocked fitness tutorial videos through this interface.
/api/tutorial/{tutorialID}/planGenerate	POST	token, tutorialId	This interface can generate a personalized fitness plan and return the generated plan content.
/api/tutorial/{tutorialID}/planAIadjust	PUT	token, tutorialId	The user adjusts the fitness plan through this interface, and changes it according to the suggestions of AI assistance.
/api/tutorial/{tutorialID}/fitProgress	POST	token, tutorialId	The user uploads the fitness training data through this interface, and the system tracks the user's fitness progress.
/api/tutorial/{tutorialID}/dataAnalysis	GET	token, tutorialId	The system provides the data analysis function, show the user's fitness performance and trend, and generate the analysis report.
/api/tutorial/progressReport	GET	token, tutorialId, reportType	Users can export detailed fitness progress reports in different formats, view data, trends, and suggestions for improvement.
/api/tutorial/{tutorialID}/tutorialCheckin	POST	token, actionId,imgUrl/videoUrl	Users upload photos or videos during the fitness process through the interface to punch in, and the system detects the standardization of AI movements.
/api/tutorial/rewardGet	GET	token, tutorialId	The corresponding virtual reward obtained after the user completes the task and clocks in is recorded in the database.
/api/tutorial/tutorialRecommendation	POST	token	The system recommends suitable tutorials and plans based on the user's fitness history and progress, and returns a personalized recommendation list.

API Interface	Method	Parameters	Description
/api/tutorial/newTutorialUpload	POST	token, newTutorialContents	The Uploader uploads a new fitness tutorial through this interface, and submits the basic information of the tutorial (such as the tutorial name, brief introduction, length, difficulty, etc.).
/api/tutorial/tutorialApprove	PUT	token, newTutorialContents	The administrator reviews the newly uploaded tutorial through this interface. After the audit, the tutorial becomes available, and the failure information is returned.
/api/tutorial/releaseComment	POST	token, tutorialId, comment	The user evaluates the specified tutorial through this interface.
/api/tutorial/commentApprove	PUT	token, tutorialId, comment	The administrator reviews the user's tutorial evaluation through this interface, and the approved evaluation will be displayed on the tutorial page.

4.2.3. Fitness Action Coaching SubSystem

API INTERFACE	METHOD	PARAMETERS	INTERFACE INTRODUCTION
/api/AICoaching/upload	POST	token, videoUrl, actionName	Users upload fitness related images or videos and return the review results and legality check results.
/api/AICoaching/getAllNoDetails	GET	token	Obtain abbreviated information of all historical analysis records of users
/api/AICoaching/getOneDetail	GET	token, analysisID	Retrieve detailed information of a user's historical record
/api/AICoaching/	DELETE	token, analysisID	Delete the specified historical analysis records.
/api/AICoaching/AIanalysis	POST	token, videoUrl, actionName	Call the Qwen VL large model for analysis, return action analysis and corrective suggestions.
/api/AICoaching/planCheckIn	POST	token, actionName, imgUrl/videoUrl	The AI analysis interface reviews whether the user's fitness plan check-in content meets the standards and returns a check-in success or failure status.

4.2.4. Fitness Equipment SubSystem

API INTERFACE	METHOD	PARAMETERS	INTERFACE INTRODUCTION
/api/equipment/recommend	GET	token	Recommend fitness equipment based on user information and return the list in card form.

API INTERFACE	METHOD	PARAMETERS	INTERFACE INTRODUCTION
/api/equipment/search	GET	token, query, page, limit	Retrieve relevant fitness equipment based on search keywords
/api/equipment/getOneDetail	GET	equipmentID	Obtain detailed information about designated equipment, including ratings, experience analysis, historical evaluations, etc.
/api/equipment/{equipmentID}/review	POST	token, equipmentID, rating, comment	Users evaluate and rate the equipment.
/api/equipment/{equipmentID}/compare	GET	token, equipmentID	Obtain the prices, product introductions, and user reviews of designated equipment on different e-commerce platforms, and recommend the most affordable options
/api/equipment/{equipmentID}/chat	POST	token, equipmentID, imgUrl, message	Provide intelligent answers for VIP users to help them further understand equipment usage methods
/api/equipment/{equipmentID}/accept	POST	token, equipmentID, responseID	Users can directly insert intelligent answers as comments into equipment evaluations.

4.2.5. Healthy Diet Subsystem

- **Dietary Record Module**

API Interface	Method	Parameters	Description
api/dietaryRecord/createDietaryRecord	POST	token,DietRecord	Create a new dietary record information and return whether the operation was successful and the results of the AI analysis.
api/dietaryRecord/updateDietaryRecord	PUT	token,recordID,DietRecord	Update an existing dietary record and return whether the operation is successful.
api/dietaryRecord/deleteDietaryRecord	DELETE	token,recordID	Delete a dietary record and return whether the operation is successful.
api/dietaryRecord/getDietaryRecord	POST	token	Return the dietary records of the user.
api/dietaryRecord/getAIanalysis	POST	token, recordIDs(List of recordIDs of records to be analyzed)	Send dietary records for analysis and return one, single, AI analysis result.
api/dietaryRecord/saveAnalysis	POST	token, Analysis	Save AI analysis results to the database and return whether the operation is successful.

Dietary Plan Module

API Interface	Method	Parameters	Description
api/dietaryPlan/getAllPlan	GET	None	Return summarized information of all dietary plans.
api/dietaryPlan/getPlanDetail	GET	planID	Return detailed information about a specific dietary plan.
api/dietaryPlan/setupComplete	POST	token, ClockinRecord	Mark a dietary plan as completed.
api/dietaryPlan/setupSkip	POST	token, ClockinRecord	Mark a dietary plan as skipped and log the reason for skipping.
api/dietaryPlan/startPlan	POST	token, planID	User selects current plan.
api/dietaryPlan/getAllDish	GET	None	Return summarized information of all available dishes.
api/dietaryPlan/getDishDetail	GET	dishID	Return detailed information about a specific dish.
api/dietaryPlan/createPlan	POST	token, DietPlan	Create a new dietary plan and return whether the creation was successful.
api/dietaryPlan/updatePlan	PUT	token, planID, DietPlan	Update an existing dietary plan and return whether the creation was successful.
api/dietaryPlan/aduitPlan	POST	token, aduitResult, planID(Approval or rejection status)	Audit the dietary plan and return the audit results.
api/dietaryPlan/getTodayPlan	POST	token	Get the details of today's corresponding plan in the user's current execution plan.

4.3. Interface Specification

Since the intelligent fitness system platform involves different roles and authority management, in order to ensure the privacy security of data and the efficiency of information transmission in the process of system operation, the project needs to include a special security service module, responsible for user login authentication and authority control. Considering that the front and back end architecture of the system is separated, and there are a series of requirements, such as authentication and security access, we chose sa-token as the security framework. The sa-token isa lightweight and powerful framework capable of providing efficient, flexible authentication and licensing capabilities. By using sa-token, we can effectively manage the permissions of different user roles, ensure the data security of the system, while ensuring the fluency of user operation and the efficiency of the authentication process, so as to support the stable operation and rapid iteration of the system.

4.3.1. Internal Interface Description

- **Login Authentication**

Before users can access the various features of our project, they must first authenticate their login credentials. This is necessary because access to certain operational interfaces is restricted to authorized users only. Login authentication serves as a prerequisite for performing any operations that require authenticated user permissions.

To ensure smooth and secure user interactions, the sa-token framework is utilized to manage the login process. It provides a set of API interfaces that allow the system to check the current login status, as well as retrieve token-related information such as validity, expiration, and user identity. These interfaces seamlessly integrate with other parts of the system, enabling efficient authentication management and facilitating subsequent user operations that require authentication.

By leveraging the sa-token framework, our project can ensure that all operations are performed securely and only by authenticated users, enhancing both functionality and security across the platform.

API	Method	Parameters	Description
`/api/ua/login/passwd`	POST	`username`, `password`	Log in using username and password, include user type (system)
`/api/ua/logOut`	POST	`token`	Log out, clear token and clean up permission cache
`/api/ua/login/wechatQR`	POST	`wechat_code`	Log in using information from WeChat QR login
`/api/ua/token/refresh`	POST	`token`	Refresh token
`/api/ua/token/check`	POST	`token`	Check if the token is valid

4.3.2. External Interface Description

To enhance development efficiency and reduce costs, we have selected a range of third-party APIs to provide users with intelligent, personalized, and convenient functionality and services.

Alibaba Cloud Bailian Platform's Large Model Service API:
The Alibaba Cloud Bailian platform provides large model services based on multimodal technology, suitable for intelligent analysis and generation tasks in various scenarios. Its features include powerful performance, support for multimodal interaction, high customizability, and mature API service interfaces.
This API supports the following functionalities in the system:

- **Fitness Action Analysis**:
  Analyze the standardization and accuracy of users' fitness movements through visual capabilities and provide targeted improvement suggestions.
  Combine action recognition to help users track training progress and health data.
- **Smart Answering Functionality**:
  Fitness equipment assistants use the NLP capabilities of the large model to answer users' questions, provide personalized suggestions, and offer equipment guidance.
- **Customized Fitness Plans**:
  Generate personalized fitness plans based on users' physical data, exercise history, and goals, increasing user engagement.

- **Diet Analysis and Suggestions**:
  Provide optimization suggestions by analyzing users' dietary images or records, helping them manage diet and health.

WeChat Login API:
The WeChat Login API is a convenient third-party login service provided by WeChat, allowing users to quickly log in to other platforms via their WeChat accounts while supporting access to basic user information (such as nickname, avatar, etc.). Its features include simplicity, high efficiency, wide user coverage, and secure authentication.

In our project, we implement WeChat QR Code login to provide users with a quick and secure login method. Users can scan the WeChat QR code on the website, grant authorization, and log in directly. This method utilizes WeChat's Open Platform API.

This API supports the following functionalities in the system:

- **Convenient User Login**:
  Users can log in with one click via WeChat, reducing the complexity of registration processes and enhancing user experience.
- **User Data Support**:
  Access to basic user information (e.g., nickname, avatar) enhances the system's personalized display and interaction capabilities.
- **Retention Rate Improvement**:
  Reduces the burden of account management, lowering user attrition rates.
- **Wechat**
- **Simple Flow**
  1. **Generate QR Code**: The website generates a WeChat QR code.
  2. **User Scans QR Code**: The user scans the QR code with their WeChat app.
  3. **Authorization**: After scanning, the user authorizes the login.
  4. **Exchange Code for Tokens**: The backend exchanges the authorization code for an access token and openid.
  5. **Retrieve User Info**: With the access token and openid, the backend fetches the user's basic information.
  6. **Login Complete**: The user is logged in and can proceed with using the system.

Alibaba Cloud Object Storage Service (OSS) API:
Alibaba Cloud OSS (Object Storage Service) is a cloud storage service designed for massive data storage, supporting distributed storage and high-speed access. Its features include high scalability, high reliability, and cost-effectiveness. This API supports the following functionalities in the system:

- **User Data Storage**:
  Store multimedia files uploaded by users, such as images and videos.
- **System Resource Storage**:
  Store static system resources such as fitness equipment images and product icons to optimize system loading speed.
- **Distributed Storage Support**:
  Ensure stability for massive data under high-concurrency access, meeting the system's fast response requirements.

Taobao, Tmall, and Pinduoduo Product APIs:
The product APIs of Taobao, Tmall, and Pinduoduo provide access to product data from these platforms, including information on prices, inventory, promotions, reviews, and more. These APIs feature strong real-time capabilities, extensive coverage, and support for quick access to massive product information.
This API supports the following functionalities in the system:

- **Price Comparison Support**:
  Retrieve fitness equipment prices across different e-commerce platforms, helping users select the most cost-effective products.
- **Product Information Retrieval**:
  Provide product descriptions, reviews, and promotional information to enhance users' purchasing experience.
- **E-commerce Redirection**:
  Allow users to directly jump to corresponding e-commerce platforms to complete the purchase loop after selecting desired products.
- **Recommendation Functionality**:
  Recommend related products based on users' search behavior, enhancing the system's personalized service capabilities.

Example:Alibaba Cloud Large Model Service API Interface Specification

Aliyun's Large Model Service enables developers to leverage advanced natural language processing and visual understanding capabilities through API calls. This document provides specific methods for invoking Aliyun's Large Model Service, including request formats, response formats, and exception handling.

Request Format:

Requests should follow the HTTP POST protocol and include necessary header information and request body.

Request URL:

```
https://dashscope.aliyuncs.com/api/v1/services/aigc/multimodal-generation/generation
```

Request Headers

- `Authorization`: Required, format is `Bearer <api_key>`
- `Content-Type`: Required, value is `application/json`.

Request Body The request body is in JSON format and mainly includes the model name and a list of messages. Each message in the message list can contain text or multimedia data (such as image URLs, video URLs).

- **Example Request**

```
{
  "model": "qwen-vl-max-latest",
  "input": {
    "messages": [
      {
        "role": "user",
        "content": [
          {
            "type": "video",
            "video": [
              "example1.jpg",
```

```
            "example2.jpg",
            "example3.jpg",
            "example4.jpg"
          ]
        },
        {
          "type": "text",
          "text": "Describe the specific fitness action shown in this video."
        }
      ]
    }
  ]
 }
}
```

Response Format

A successful response will return data in JSON format, containing the generated content, token usage, etc.

- **Example Response**

```
{
  "output": {
    "choices": [
      {
        "finish_reason": "stop",
        "message": {
          "role": "assistant",
          "content": [
            {
              "text": "This video shows a moment from a football match. The video is shot from behind the goal, showing players running and playing on the field. The specific process is as follows:..."
            }
          ]
        }
      }
    ]
  },
  "usage": {
    "output_tokens": 151,
    "input_tokens": 1465,
    "total_tokens": 1616
  },
  "request_id": "c728d1e0-79ad-9076-8589-7f072e96bccf"
}
```

Exception Handling

When a request fails, the API returns an error code and error message to help developers identify the issue.

- **Example Exception Response**

```
{
  "status_code": 400,
  "code": "InvalidParameter",
  "message": "The provided parameter is invalid.",
  "request_id": "7574ee8f-38a3-4b1e-9280-11c33ab46e51"
}
```

Details

- `model`: The name of the large model being used.
- `messages`: A list of messages sent by the user, each message can be text or multimedia data.
- `role`: The role of the message, usually `user` or `assistant`.
- `type`: The type of content, such as `text`, `image_url`, or `video`.
- `content`: The specific content of the message, for text type it is a string, for multimedia type it is a URL.

Supported Image Formats

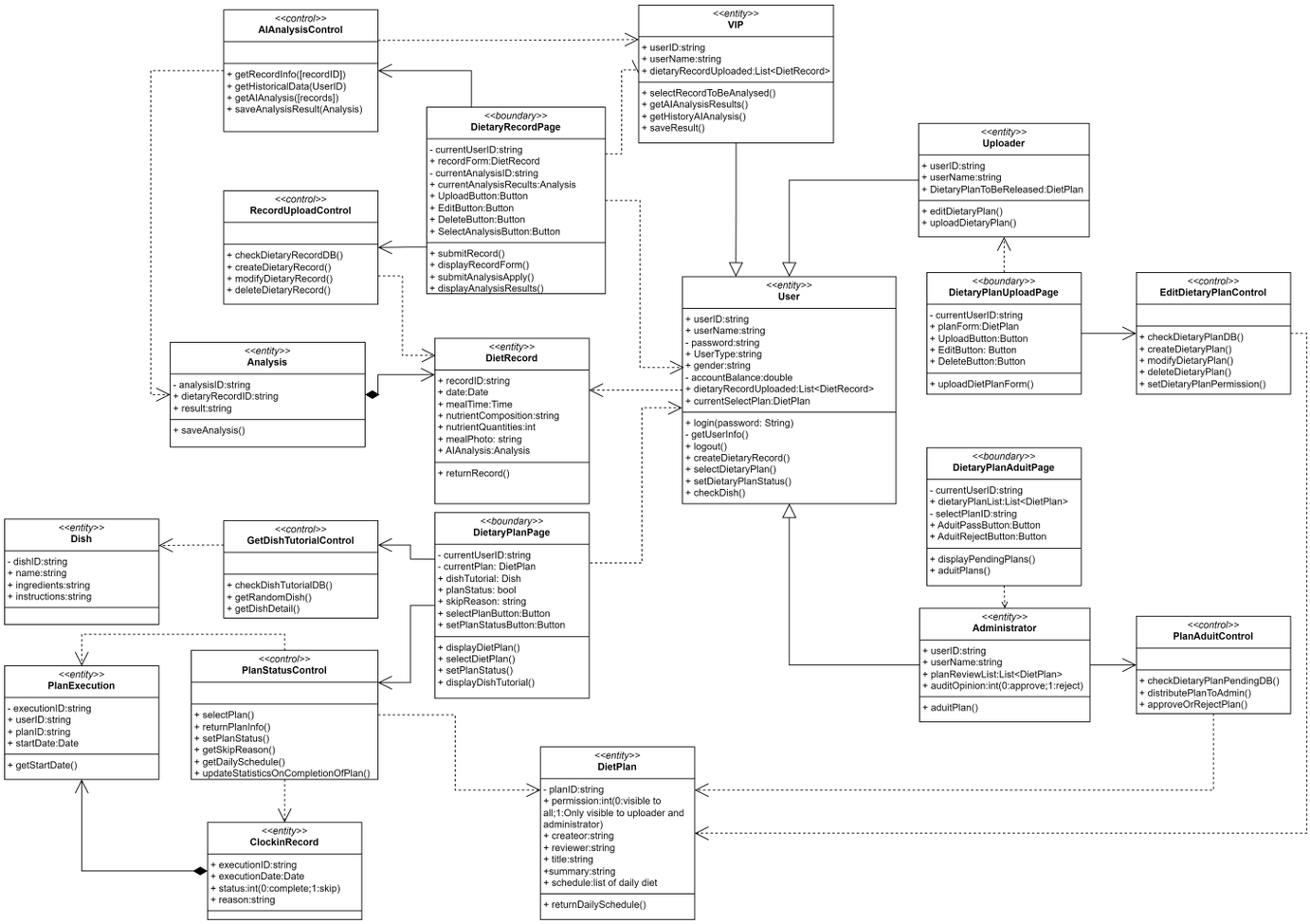
- **BMP**: `image/bmp` (.bmp, .dib)
- **DIB**: `image/bmp` (.dib)
- **ICNS**: `image/icns` (.icns)
- **ICO**: `image/x-icon` (.ico)
- **JPEG**: `image/jpeg` (.jfif, .jpe, .jpeg, .jpg)
- **JPEG2000**: `image/jp2` (.j2c, .j2k, .jp2, .jpc, .jpf, .jpx)
- **PNG**: `image/png` (.apng, .png)
- **SGI**: `image/sgi` (.bw, .rgb, .rgba, .sgi)
- **TIFF**: `image/tiff` (.tif, .tiff)
- **WEBP**: `image/webp` (.webp)

Supported Models

Model Name	Description
qwen-vl-max	A high-capacity model supporting a context length of 32k tokens, enhanced for image understanding, and better at recognizing multilingual text and handwritten text in images.
qwen-vl-max-latest	The latest version of the qwen-vl-max model, incorporating the most recent improvements and updates.
qwen-vl-plus	A versatile model designed for both text and image processing, suitable for a wide range of applications.
qwen-vl-plus-latest	The latest version of the qwen-vl-plus model, featuring the most recent enhancements and optimizations.

4.4. Example

The applications of the Smart Fitness Platform are not limited to a single web-based program or application. In order to better meet the diverse needs of users, we have designed multiple subsystems including the healthy diet subsystem. In the specific implementation process, we designed a series of API interface use cases for the healthy diet subsystem to facilitate users to interact with the system and obtain real-time dietary advice. In the process of further design, we first further improved the classes of the diet system.



The specific application program interface design for the healthy diet system is shown below:

4.4.1. Create Dietary Record

- **Request Type**: POST
- **REST API**: `/api/dietaryRecord/createDietaryRecord`
- Parameters:
  - `token`: User token for authentication.
  - `DietRecord`: JSON object containing the new dietary record details.
- **Steps on the server side:**
  1. Authenticate user identity.
  2. Generate a globally unique `recordID`.
  3. Store the record in the database.
  4. Return the result including `recordID` and AI analysis.

- **Request Example:**

```
{
  "token": "userToken12345",
  "DietRecord": {
    "userID": "user001",
    "date": "2024-11-28",
    "mealTime": "12:18",
    "mealPhoto": "base64EncodedPhotoString", // Optional, Base64 encoded string if present
    "items": [
      {
        "nutrientComposition": "Carbohydrates",
        "quantity": 70
      },
      {
        "nutrientComposition": "Protein",
        "quantity": 50
      },
      {
        "nutrientComposition": "Fat",
        "quantity": 40
      }
    ]
  }
}
```

- **Response Example:**

```
{
  "status": "success",
  "message": "Dietary record created successfully.",
  "recordID": "record_id_1",
  "AIAnalysis": "Here is a brief dietary suggestion: \n 1. Protein: Include a moderate amount of shrimp, tofu, and beef tendon balls, but avoid excess.\n 2. Fat: Limit fried foods and opt for steaming or boiling methods instead.\n 3. Carbohydrates: Reduce white rice intake"
}
```

4.4.2. Get Dietary Record

- **Request Type**: POST
- **REST API**: `/api/dietaryRecord/getDietaryRecord`
- Parameters:
  - `token`: User token for authentication.
- **Steps on the server side:**
  1. Authenticate user identity.
  2. Retrieve the dietary record(s) from the database using the provided `recordID`.
  3. Return the dietary record(s) to the client.
- **Response Example:**

```
{
  "status": "success",
  "message": "Dietary records retrieved successfully.",
  "dietaryRecords": [
    {
      "recordID": "record12345",
      "userID": "user001",
      "date": "2024-11-28",
      "mealTime": "12:18",
      "mealPhoto": "base64EncodedPhotoString", // Base64 encoded string, optional
      "items": [
        {
          "nutrientComposition": "Carbohydrates",
          "quantity": 70
        },
        {
          "nutrientComposition": "Protein",
          "quantity": 50
        },
        {
          "nutrientComposition": "Fat",
          "quantity": 40
        }
      ]
    },
    {
      "recordID": "record67890",
      "userID": "user001",
      "date": "2024-11-29",
      "mealTime": "19:00",
      "mealPhoto": "",
      "items": [
        {
          "nutrientComposition": "Carbohydrates",
          "quantity": 60
        },
        {
          "nutrientComposition": "Protein",
          "quantity": 45
        },
        {
          "nutrientComposition": "Fat",
          "quantity": 35
        }
      ]
    }
  ]
}
```

4.4.3. Get AI Analysis

- **Request Type**: POST
- **REST API**: `/api/dietaryRecord/getAIanalysis`
- Parameters:
  - `token`: User token for authentication.
  - `recordIDs`: An array of dietary record IDs to analyze.
- **Steps on the server side:**
  1. Authenticate user identity.
  2. Read each dietary record based on the provided `recordIDs`.
  3. Analyze using an AI model.
  4. Package and return the analysis results to the client.
- **Request Example:**

```
{
  "token": "userToken12345",
  "recordIDs": [
    "record_id_1",
    "record_id_2",
    "record_id_3"
  ]
}
```

- **Response Example:**

```
{
  "status": "success",
  "message": "Request succeed.",
  "results": [
    {
      "analysisID": "analysis_id_1",
      "recordIDs": [
        "record_id_1",
        "record_id_2",
        "record_id_3"
      ],
      "result": "Here is a brief dietary suggestion: \n 1. Protein: Include a moderate amount of shrimp, tofu, and beef tendon balls, but avoid excess.\n 2. Fat: Limit fried foods and opt for steaming or boiling methods instead.\n 3. Carbohydrates: Reduce white rice intake"
    }
  ]
}
```

4.4.4. Get All plan

- **Request Type**: GET
- **REST API**: `/api/dietaryPlan/getAllPlan`
- **Steps on the server side:**
    1. Authenticate user identity and read the corresponding UserID and role.
    2. Based on the user's role, determine the content of the plans to return:
        - For regular users or VIPs, return all plans with permission 0 (public plans).
        - For content creators, return all permission 0 plans and permission 1 plans where the creator is the current UserID.
        - For administrators, return all plans.
    3. Retrieve the relevant plan IDs, titles, and summaries from the database.
    4. Return the dietary plan information.
- **Response Example:**

```
{
  "status": "success",
  "message": "All dietary plans retrieved successfully.",
  "plans": [
    {
      "planID": "plan_id_123",
      "title": "Low Carb Plan",
      "summary": "A low carbohydrate diet plan designed for weight loss."
    },
    {
      "planID": "plan_id_456",
      "title": "High Protein Plan",
      "summary": "A high protein diet plan designed for muscle gain."
    }
  ]
}
```

4.4.5. Setup current plan

- **Request Type**: POST
- **REST API**: `/api/dietaryPlan/setupCurrentPlan`
- Parameters:
    - `token`: User token for authentication.
    - `planID`: The ID of the plan to set as the current plan.
- **Steps on the server side:**
    1. Authenticate user identity and read the corresponding UserID.
    2. Generate the current request time as `startDate` and a globally unique `executionID`.
    3. Store the `PlanExecution` in the database.
    4. Return the result including the `executionID`.

- **Request Example:**

```
{
  "token": "userToken12345",
  "planID": "plan_id_123"
}
```

- **Response Example:**

```
{
  "status": "success",
  "message": "Current plan set successfully.",
  "executionID": "execution_id_01"
}
```

4.4.6. Get today plan

- **Request Type**: POST
- **REST API**: `/api/dietaryPlan/getTodayPlan`
- Parameters:
  - `token`: User token for authentication.
- **Steps on the server side:**
  1. Authenticate user identity and read the corresponding UserID.
  2. Retrieve the latest record for the UserID to get the current plan and its start date.
  3. Retrieve the schedule of the current plan from the database.
  4. Calculate today's plan based on the start date, today's date, and the schedule.
  5. Return the content of today's plan from the schedule.
- **Response Example:**

```
{
  "status": "success",
  "message": "Today's plan retrieved successfully.",
  "todayPlan": {
    "Breakfast": {
      "calories": "293kcal",
      "content": "Skimmed milk (1 box), ..."
    },
    "Lunch": {
      "calories": "547kcal",
      "content": "Sweet potato rice (1 bowl)"
    },
    "Dinner": {
      "calories": "421kcal",
      "content": "Rice (1 bowl)"
    }
  }
}
```

4.4.7. Setup complete

- **Request Type**: POST
- **REST API**: `/api/dietaryPlan/setupComplete`
- **Parameters**:
  - `token`: User token for authentication.
  - `clockinRecord`: JSON object containing the completion details.
- **Steps on the server side:**
  1. Authenticate user identity.
  2. Store the completion status in the database.
  3. Return the success message and `executionID`.
- **Request Example:**

```
{
  "token": "userToken12345",
  "clockinRecord": {
    "executionID": "execution_id_01",
    "executionDate": "2024-12-03T08:00:00Z",
    "status": 0, // 0: complete, 1: skip
    "reason": "" // Optional field, empty on completion status
  }
}
```

- **Response Example:**

```
{
  "status": "success",
  "message": "Plan completion recorded successfully.",
  "executionID": "execution_id_01"
}
```

4.4.8. Update plan

- **Request Type**: PUT
- **REST API**: `/api/dietaryPlan/updatePlan`
- Parameters:
  - `token`: User token for authentication.
  - `planID`: The ID of the plan to update.
  - `DietPlan`: JSON object containing the new details of the diet plan.
- **Steps on the server side:**
  1. Authenticate user identity and verify the user's authority to update the plan.
  2. Read from the backend database to check if the `planID` corresponds to an existing plan.
  3. Update the corresponding plan in the database with the new `DietPlan` content.
- **Request Example:**

```
{
  "token": "userToken12345",
  "planID": "plan_id_1",
  "DietPlan": {
    "title": "Updated Low Carb Plan",
    "summary": "An updated low carbohydrate diet plan designed for weight loss.",
    "details": {
      "meals": [
        {
          "mealType": "Breakfast",
          "items": [
            {
              "name": "Eggs",
              "quantity": "2 eggs"
            },
            {
              "name": "Avocado",
              "quantity": "1/2 avocado"
            }
          ]
        },
        {
          "mealType": "Lunch",
          "items": [
            {
              "name": "Grilled Chicken",
              "quantity": "150g"
            },
            {
              "name": "Mixed Salad",
              "quantity": "1 bowl"
            }
          ]
        }
      ]
    }
  }
}
```

- **Response Example:**

```
{
  "status": "success",
  "message": "Dietary plan updated successfully."
}
```

4.4.9. Audit plan

- **Request Type**: POST
- **REST API**: `/api/dietaryPlan/auditPlan`
- **Parameters**:
  - `token`: User token for authentication.

- `auditResult`: Boolean indicating whether the plan is approved or not.
- `planID`: The ID of the plan to audit.

- **Steps on the server side:**
  1. Authenticate the user's administrator identity.
  2. Update the plan's audit status (permission field) in the backend database.

- **Request Example:**

```
{
  "token": "adminToken12345",
  "auditResult": true,
  "planID": "plan_id_1"
}
```

- **Response Example:**

```
{
  "status": "success",
  "message": "Plan audited successfully."
}
```

5. Design Mechanism

5.1. Data Persistence Mechanism

In our Keep Fit Platform, data persistence is a core component to ensure the efficient operation of the system. With the expansion of platform features and the growth of user demand, we need to perform data persistence operations across various modules, including but not limited to: diet plans, execution statuses, diet records, and analysis results in the health diet system; fitness videos or images uploaded by users and their corresponding analysis results; users' fitness plans, execution statuses, and fitness preferences. Moreover, in order to support AI recommendation engines and intelligent analysis, we also need to store personalized fitness data, health history, and progress trends.

5.1.1. Data Persistence Requirements

The data in our platform is diverse, including both static data (e.g., personal information, diet plans) and dynamic data (e.g., fitness videos, execution statuses, analysis results). To efficiently and securely store this data, choosing an appropriate persistence solution is crucial.

Given the system's needs, a relational database is the most suitable choice, as it can flexibly store this structured and related data. To ensure the platform can interact efficiently with the database, we have chosen to use the Hibernate framework for data persistence management.

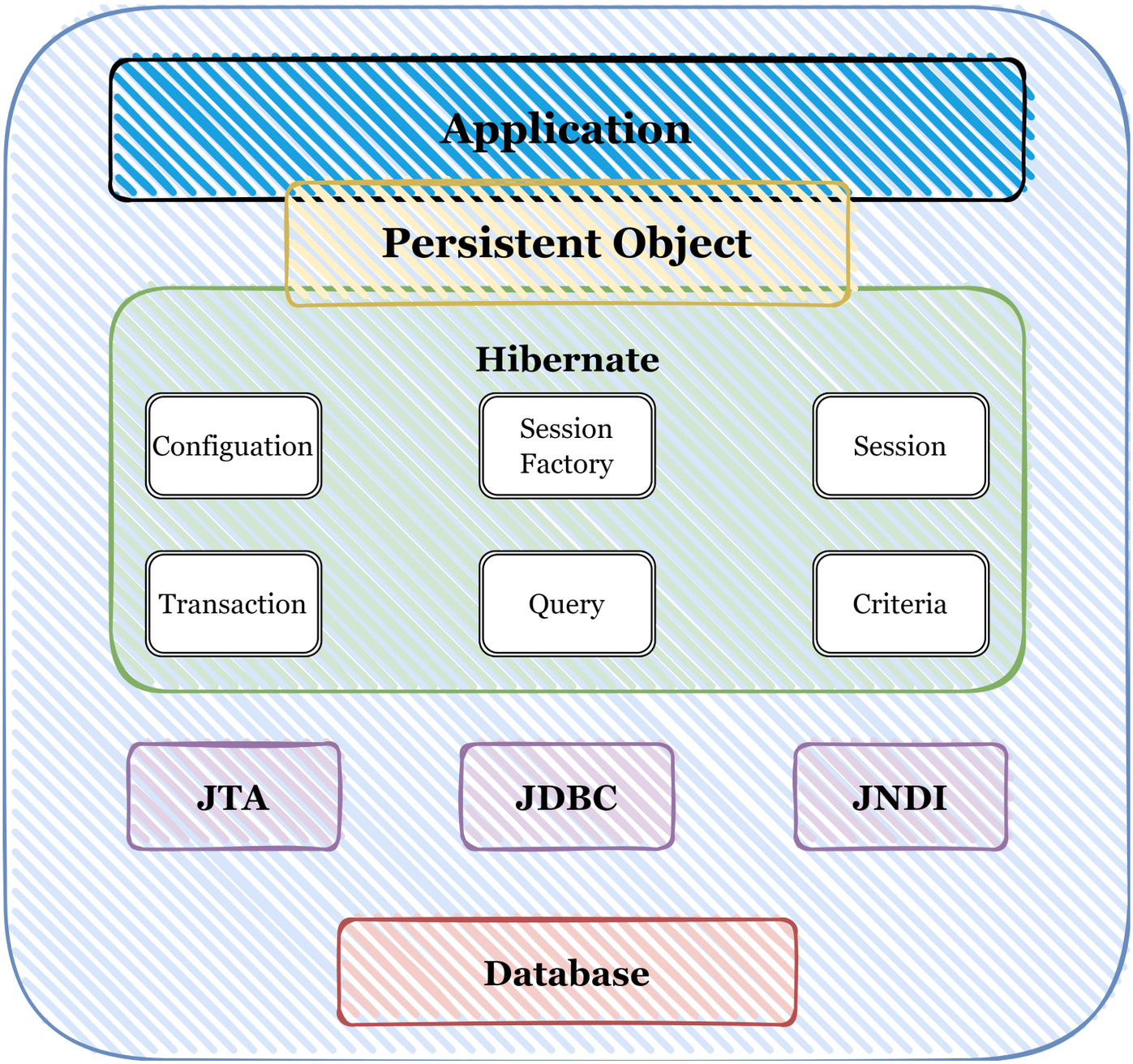
5.1.2. Persistence Architecture and ORM Technology

In our platform architecture, Java objects typically reside in memory, and these objects need to interact with data stored in the relational database at different points in time. Thus, we employ Object-Relational Mapping (ORM) technology through the Hibernate framework to establish a mapping between Java objects in memory and data tables in the relational database.

The core advantage of the Hibernate framework is that it simplifies database operations and significantly reduces code redundancy. It can automatically generate SQL statements, handle database transactions, and integrate closely with JDBC to ensure support for most relational databases. For our smart fitness platform, Hibernate easily implements bidirectional mapping between objects and database tables.

5.1.3. Persistence Layer Design

In the design of our platform, we have created a persistence layer that sits between the business layer and the database, acting as an intermediary. This layer is responsible for data persistence operations, providing interfaces through which the business layer can easily interact with the database for data retrieval, updates, and deletions.



- **Hibernate Framework Workflow**

1. **Configuration Object**: First, we need to configure Hibernate through configuration files such as `hibernate.cfg.xml` and `hibernate.properties`. These files include database connection information (e.g., database URL, username, password) and the mappings between Java classes and database tables. By using these configuration files, Hibernate can identify and initialize all the necessary information for connecting to the database.

2. **SessionFactory Object**: When the application starts, we create a `SessionFactory` object. This is a heavyweight object responsible for managing all database sessions. The `SessionFactory` creates a database connection pool based on the configuration files, ensuring efficient sharing of database connection resources across the entire system. For different database configurations, multiple `SessionFactory` objects may be required.
3. **Session Object**: The `Session` object is the basic unit of interaction with the database in Hibernate. Each time a database operation (such as inserting or querying data) occurs, it is done through a `Session`. The `Session` object is lightweight and is typically created dynamically for each operation, then destroyed after the operation is completed. To ensure performance and thread safety, the `Session` object is generally created and destroyed per request rather than being kept open for long periods.
4. **Transaction Object**: Transaction management is crucial for interacting with databases. The `Transaction` object in Hibernate ensures atomicity and consistency of operations. For example, when updating a user's fitness record, it is essential to synchronize data across multiple related tables. The `Transaction` object provides methods for beginning, committing, and rolling back transactions.
5. **Query Object and Criteria Object**: To retrieve data from the database, Hibernate provides the `Query` object and `Criteria` object. The `Query` object allows data retrieval using either native SQL statements or Hibernate Query Language (HQL). The `Criteria` object offers a more object-oriented approach to querying, enabling dynamic construction of SQL statements based on query conditions. In our fitness platform, retrieving AI analysis results and fitness preferences typically relies on these query tools to perform flexible queries.

5.1.4. Examples of Data Persistence Scenarios

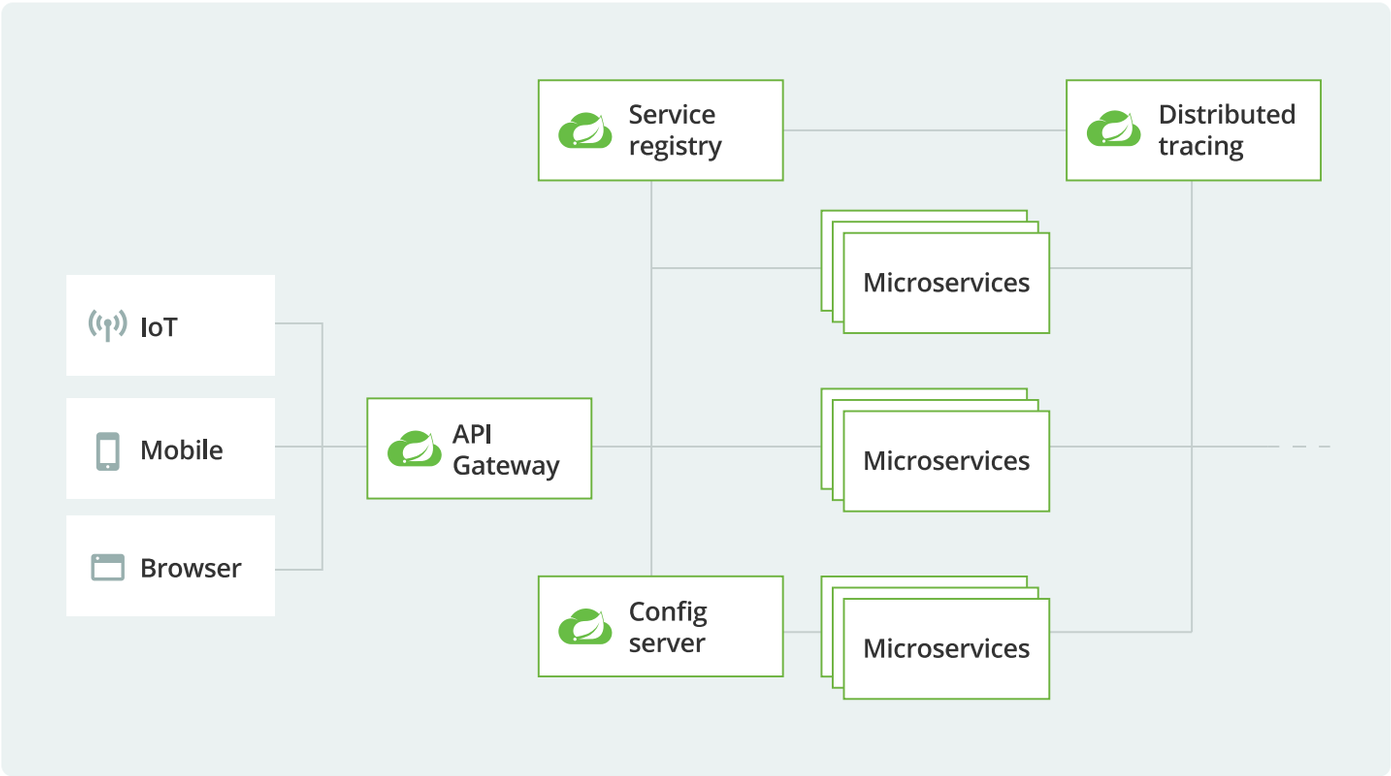
1. **Diet Plans and Records**: Each user's diet plan, execution status, diet records, and analysis results need to be persistently stored. With Hibernate, we map these details to Java objects and store them in the corresponding database tables. When users update their diet records, the system automatically synchronizes the data with the database, ensuring that the user’s data is up-to-date.
2. **Fitness Videos and Analysis Results**: The fitness videos or images uploaded by users, along with their analysis results, are typically large files stored in the file system. However, related metadata (such as filenames, user IDs, upload time) is persistently stored in the database using Hibernate.
3. **Personalized Fitness Plans and AI Recommendations**: Based on user preferences and historical data, the platform provides personalized fitness plan recommendations. These recommendations, along with the user's fitness preferences and historical training records, are persistently stored for the AI engine to analyze and optimize.

In the Keep Fit Platform, data persistence involves not only storing basic user information and fitness records but also handling complex data models such as diet plans, video analysis results, and AI recommendation information. By adopting the Hibernate framework, we can efficiently map between objects and relational database tables, simplifying the implementation of the data access layer and improving the system's maintainability and scalability. The decoupling of the persistence layer from the business layer allows the system to be more flexible and extensible, providing users with a more personalized and intelligent fitness management experience.

5.2. Distribution Mechanism

As we choose to use **Microservices-based Architecture** during our project design and implementation, it is natural for us to consider distributing our microservices to make our service more scalable, resilient, and able to handle a large volume of users and transactions. And we hope to do so by utilizing a matured, automated framework that can significantly reduce the development and maintenance complexity.

In our system, we have adopted **Spring Cloud** to construct a robust, scalable, and fault-tolerant distributed architecture. **Spring Cloud** provides a comprehensive suite of tools and frameworks designed to tackle the common challenges faced when developing microservices, such as service discovery, load balancing, configuration management, and fault tolerance. These tools help to ensure that our system can scale efficiently, remain highly available, and maintain smooth communication across a large number of distributed services.



5.2.1. Microservices Architecture

As mentioned before, to enhance the system's modularity, maintainability, and resilience, we have implemented a **Microservices-based Architecture**. The entire platform is divided into a set of small, independently deployable services, each responsible for a specific functionality within the system. For example, we have distinct services for user management, order processing, inventory management, and notifications, allowing each service to evolve independently while still working together seamlessly.

Each microservice in the system is loosely coupled, which means that they can be developed, deployed, and scaled independently. This architectural style enables rapid development cycles, as teams can work on different services simultaneously without causing dependency conflicts. Moreover, microservices can be written in different programming languages or frameworks based on the specific requirements of the service, providing flexibility in choosing the best tools for each task.

In this architecture, **Spring Cloud Netflix Eureka** is used to manage service discovery. Eureka acts as the service registry where every microservice registers itself upon startup, and other services can discover and interact with them through a simple API. This eliminates the need for static service URLs, allowing the system to scale dynamically without having to manually update configurations whenever new instances are added or removed.

Furthermore, **Spring Cloud Gateway** is employed as the single entry point for all external requests. It handles routing, load balancing, and API management. The gateway uses predefined routes to direct traffic to the appropriate microservices. This pattern improves the overall system’s security and performance, as traffic is centralized and can be controlled more efficiently.

5.2.2. Service Management and Fault Tolerance

In a distributed system, service failures are inevitable, so ensuring resilience is a key consideration in our implementation. We use **Spring Cloud Circuit Breaker**, specifically **Hystrix** (or **Resilience4J** as a more modern alternative), to implement fault tolerance in our services. This allows us to define fallback methods for critical service calls, preventing cascading failures when one service becomes unavailable.

For example, when a microservice fails, Hystrix will "trip" the circuit breaker and return a predefined fallback response, thus preventing further attempts to contact the failing service. This mechanism ensures that the rest of the system remains functional even when one or more services are down. Additionally, this pattern helps avoid overloading the failing service, providing time for recovery without affecting the user experience significantly.

We also utilize **Spring Cloud Config** for centralized configuration management. Instead of hard-coding configuration settings in individual services, we store them in a centralized repository (such as Git), which can be dynamically loaded by the services at runtime. This centralization simplifies the management of application settings across multiple environments (development, staging, production) and makes it easier to update configurations without redeploying the services. **Spring Cloud Bus** is used to propagate configuration changes across the entire system in real-time, ensuring that all services receive the latest configuration updates immediately.

5.2.3. Load Balancing and High Availability

Spring Cloud Netflix Ribbon provides client-side load balancing, ensuring that requests are evenly distributed across available service instances. Ribbon works alongside **Eureka** to automatically retrieve the list of available service instances and balance the load among them. This reduces the risk of overloading any single instance and ensures a more efficient use of resources.

For example, when a user requests data from a particular service, Ribbon will select one of the available instances based on the chosen load balancing algorithm (such as round-robin, weighted response time, etc.). This mechanism helps improve the performance and responsiveness of the system, as each service can scale independently based on demand.

In addition, **Spring Cloud Sleuth** provides distributed tracing, allowing us to trace requests as they flow across different microservices. This is crucial for debugging and monitoring the performance of the system, especially in complex distributed systems where requests may pass through multiple services before reaching their final destination. By integrating **Zipkin** or **ELK Stack (Elasticsearch, Logstash, Kibana)** with Sleuth, we can visualize the flow of requests, identify bottlenecks, and ensure that the system is operating efficiently.

5.2.4. Scalability and Auto-Scaling

The microservices architecture allows for flexible and elastic scaling. Each microservice can be scaled independently, which helps to accommodate increasing traffic without affecting other parts of the system. We use **Kubernetes** (or other container orchestration platforms) to manage the deployment and scaling of microservices. With Kubernetes, we can automatically scale the number of replicas for a given microservice based on CPU utilization, memory usage, or custom metrics.

For example, if the order service experiences a high volume of requests, Kubernetes can automatically spin up additional instances to handle the load, ensuring that the service remains responsive. Once the load decreases, the extra instances are terminated to save resources. This dynamic scaling is vital in a cloud-native environment, where workloads can fluctuate frequently, and the system must be able to respond quickly to changes in demand.

5.2.5. Monitoring and Logging

To ensure the health of our system, we integrate **Spring Boot Actuator** to expose operational metrics and health checks, such as system performance, database connection status, and memory usage. These metrics are collected and monitored using **Prometheus** and **Grafana**, providing real-time insights into the system's health.

For logging, we use **ELK Stack (Elasticsearch, Logstash, Kibana)** or **Splunk** to collect and analyze logs from all microservices. By centralizing logs, we can quickly identify and troubleshoot issues in the system. The logs are also essential for debugging, especially when tracing requests across distributed services.

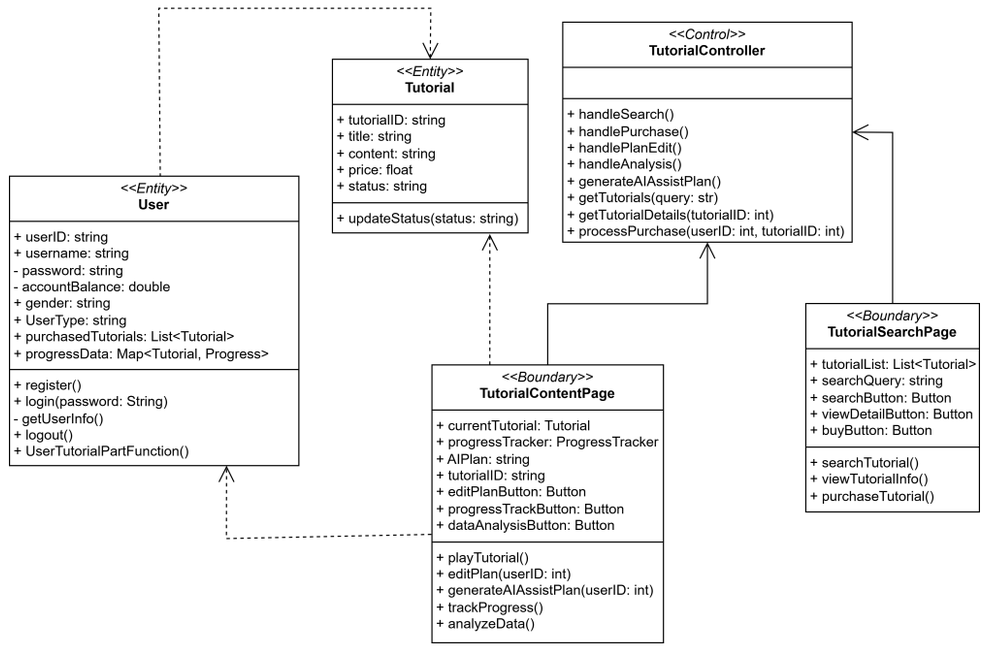
5.2.6. Conclusion

Through the adoption of **Spring Cloud**, we have successfully built a microservices architecture that is flexible, resilient, and capable of handling the complex demands of our platform. With features such as dynamic service discovery, centralized configuration, fault tolerance, and scalable deployment, we ensure that our system is both robust and easy to maintain. Furthermore, by integrating monitoring, logging, and auto-scaling capabilities, we guarantee that our system remains highly available and responsive even under heavy load. This architecture allows us to continuously improve and scale our services while maintaining high performance and reliability.

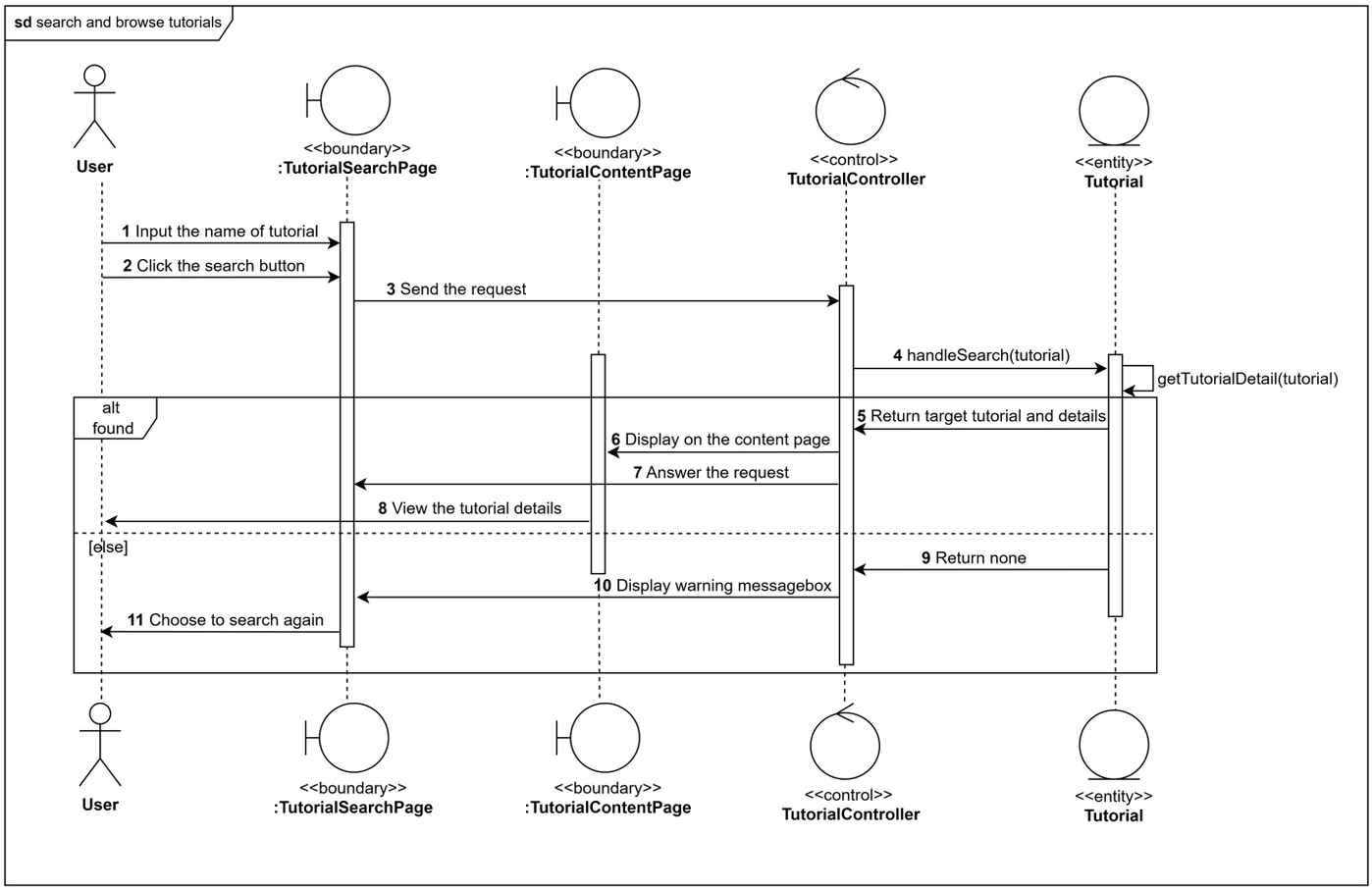
6. UseCase Realization

6.1. Search and Browse Tutorials

The class diagram for searching and browsing the tutorial use case is shown below:



The interaction diagram for the search and browse tutorial use cases is shown below:



According to the interactive diagram of search and browse tutorial of the intelligent fitness system, we can get the corresponding implementation steps, which are as follows:

1. Users enter the tutorial name in the tutorial search interface and click the "Search" button. This operation is usually implemented with front-end technologies, such as Vue.js. With these frameworks, the front end is able to respond to user input in real time and bind data to HTML elements. Once the user clicks "Search" and triggers the AJAX request through the JavaScript code, the background will receive the search keyword (tutorial name) sent by the front end. The front end page uses AJAX technology so that you can interact with the back end without refreshing the entire page to improve the user experience.
2. When a user submits a search request, the front end page sends the tutorial name (or keyword) entered by the user to the back end. This request is sent to the controller (Controller) in the back-end Spring MVC framework via the AJAX or Fetch technology. The front-end request uses the HTTP protocol, usually a GET or POST request, passing the query parameters containing user input. After receiving the request, the controller can extract the search keyword entered by the user through the parameter resolution technology (such as the @RequestParam of Spring) and pass it to the business layer for subsequent processing.
3. After receiving a search request from the front end, the Spring MVC's controller invokes the service layer (Service) to process the business logic. The service layer interacts with the database through the data access layer (DAO) to query for eligible tutorial data. The data access layer usually uses frameworks such as JPA (Java Persistence API) to perform SQL queries or uses ORM (Object relationship mapping) technology to query the database. At this point, an external database may be called through REST API to get information about the tutorial. If you use REST API, the returned data format is usually a JSON, and the controller parses it and passes it to the view layer.
4. The database returns eligible tutorial data (e. g., tutorial name, profile, duration, difficulty, etc.). This data is returned to the front end via Spring MVC's controller. The controller uses the @ResponseBody annotation to encapsulate the data in the JSON format and return it to the front-end application. Here, Spring MVC's view parser passes this data to the front-end interface, where the front-end framework (such as Vue.js) presents the tutorial information to the user through data binding technology.

5. The search controller encapsulates the tutorial information returned by the database and returns it to the front end via REST API. At the front end, the updated view is dynamically displayed to the user interface based on the JavaScript framework (such as Vue.js). At this point, the front end can update the UI using Vue.js or React status management (such as Vuex) and display the tutorial information through the corresponding components. Users can see a list containing the tutorial name, profile, duration, etc., and click to enter the detailed page.
6. If a match tutorial is not found in the database, the controller will receive an empty result or error message. At this point, the controller captures this result through the Spring MVC's exception handling mechanism and generates an error response ("No relevant tutorial found"). This response is returned to the front-end in the JSON format, which displays the corresponding prompt message box using a front-end framework such as Bootstrap, Vue.js or React. The Bootstrap pop-up prompt component or custom message box shows the user the "not found tutorial" prompt message to ensure that the user gets clear feedback.
7. When the user clicks on a tutorial in the search results, the front end implements the page jump through Vue Router or React Router. At this point, the route is dynamically loaded, and the system requests the details of the tutorial from the background via the REST API. The controller of Spring MVC queries the details in the database based on the passed tutorial ID, possibly using the Thymeleaf or JSP to render the tutorial details page. During rendering, the data is passed to the Thymeleaf Template engine (or JSP) to generate the final HTML page. The page includes tutorial details, previews, and price reviews, which users can browse in detail.

Through the Spring MVC framework for front-and rear-end interaction, with REST API, database query, AJAX and other technologies, the system can efficiently process user search requests, and return the corresponding results. Thymeleaf and JSP serve as view template engines to help show back-end data to front-end users, improving the response speed and user experience of the system. On the front end, using the Vue.js or React framework at the front end makes the interface more dynamic and interactive, providing users with a good operating experience.

The corresponding pseudo-code is as follows:

- Users enter the tutorial name in the tutorial search interface and click the search button:

```
// Vue.js: Handle user input and call the search function
searchTutorial() {
  const searchQuery = this.searchInput;
  fetchTutorialData(searchQuery);
}
```

- The search interface sends the user input to the tutorial search controller:

```
// Vue.js: Send search request to the backend using async/await and AJAX
async function fetchTutorialData(query) {
  try {
    const response = await $.ajax({
      url: `/api/tutorial/tutorialSearch`,
      // Use GET request method
      method: 'GET',
      data: { query: query },
    });
    // On success, call function to update the UI
    updateTutorialList(response);
  } catch (error) {
    // Show error message if the request fails
    showError("Search failed");
  }
```

```
}
```

- After receiving the request, the tutorial search controller queries the tutorial database:

```
// Spring MVC Controller: Receive request from the frontend and query the database in the service layer
@RequestMapping("/api/tutorial/tutorialSearch")
public ResponseEntity<List<Tutorial>> tutorialSearch(@RequestParam String query) {
    // Call service layer to query the database
    List<Tutorial> tutorials = tutorialService.searchTutorials(query);
    if (tutorials.isEmpty()) {
        // Throw exception if no tutorials are found
        throw new TutorialNotFoundException("No relevant tutorials found: " + query);
    }
    // Return the list of tutorials to the frontend
    return ResponseEntity.ok(tutorials);
}
```

- The database returns the tutorial information that matches the criteria:

```
// Service Layer: Query the tutorial data in the database
public List<Tutorial> tutorialSearch(String query) {
    // Use JPA to query tutorial data
    return tutorialRepository.findByNameContaining(query);
}
```

- The search controller returns the matching tutorial information to the frontend:

```
// Spring MVC Controller: Return matching tutorial data to the frontend
@RequestMapping("/api/tutorial/tutorialSearch")
public ResponseEntity<List<Tutorial>> tutorialSearch(@RequestParam String query) {
    // Get matching tutorials from the service layer
    List<Tutorial> tutorials = tutorialService.tutorialSearch(query);
    // Return the tutorial data in JSON format to the frontend
    return ResponseEntity.ok(tutorials);
}
```

- If no matching tutorial is found, the controller will display a "No tutorial found" message on the frontend:
  - Backend handles the exception when the requested tutorial is not found:

    ```
    // Spring MVC: Exception handling for when no tutorial is found
    @ExceptionHandler(TutorialNotFoundException.class)
    public ResponseEntity<String> handleTutorialNotFound(TutorialNotFoundException ex) {
        // Return 404 error and the message
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }
    ```

  - The front end displays error messages to alert the user of a problem:

```
// Vue.js: Display error message
function showError(message) {
  // Show an error alert
  alert(message);
}
```

- When the user clicks on a tutorial, the page navigates to the tutorial detail browse page:
  - Front-end processing The user clicks on a tutorial to see the details:

```
// Vue.js: Handle user click on a tutorial to view details
async function viewTutorialDetails(tutorialId) {
  try {
    const response = await $.ajax({
      url: `/api/tutorial/tutorialInfo`,
      // Use GET request to fetch tutorial details
      method: 'GET',
      data: { id: tutorialId },
    });
    // On success, render the tutorial details on the page
    displayTutorialDetails(response);
  } catch (error) {
    // On error, show error message
    showError("Failed to load tutorial details");
  }
}
```

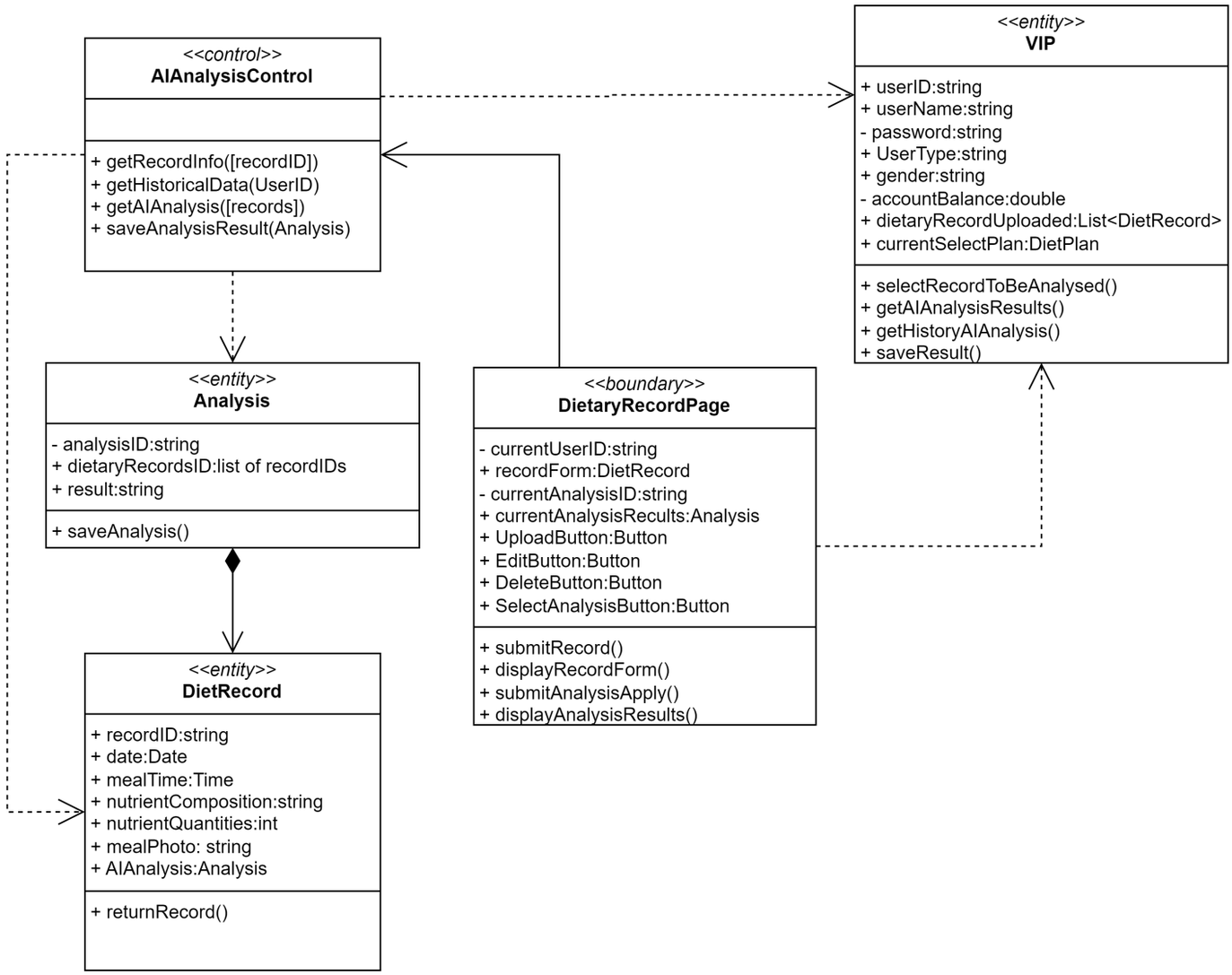
  - The backend uses the Spring MVC controller method to query the details of the tutorial against the incoming tutorialId:

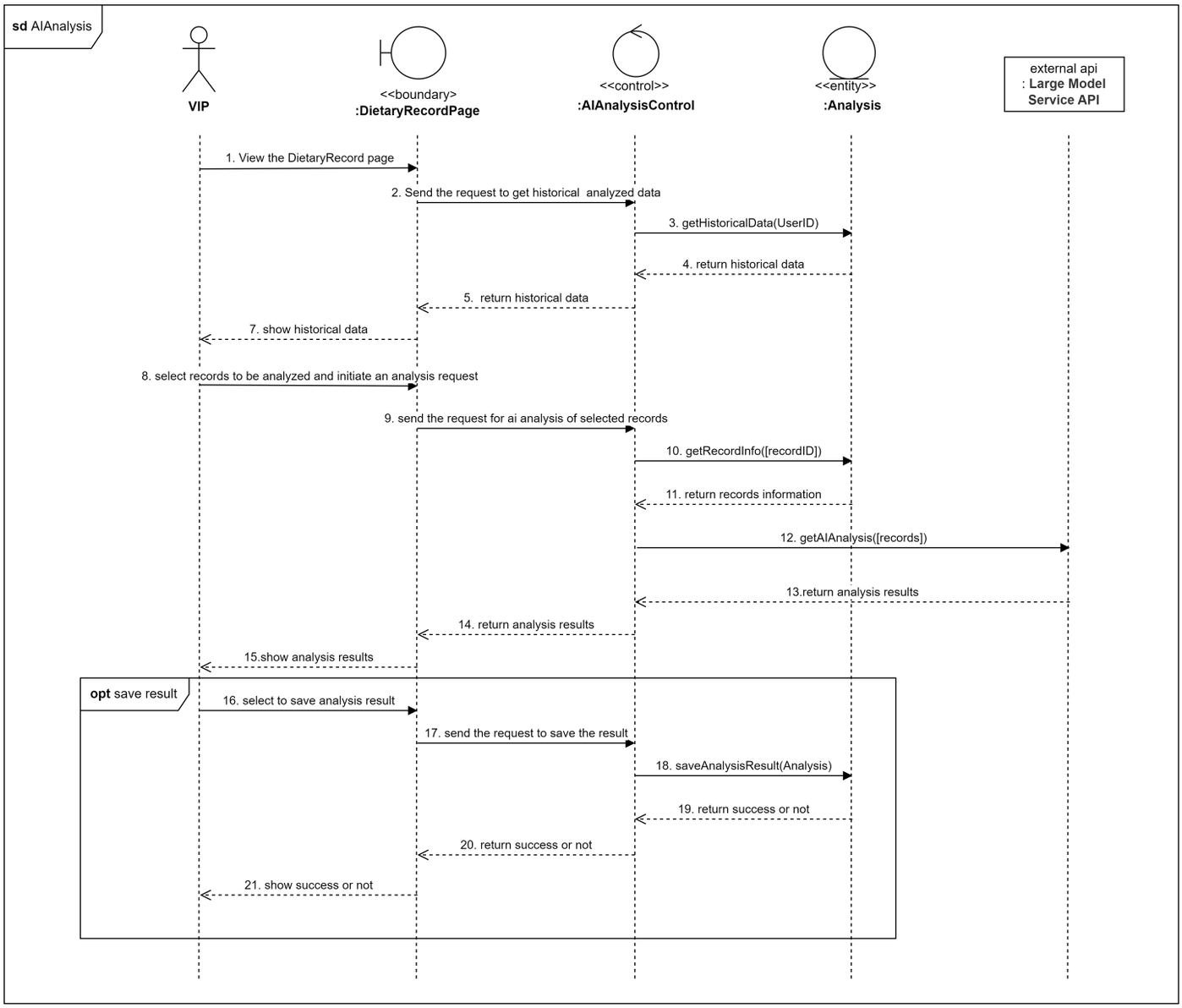
```
// Spring MVC Controller: Query tutorial details based on tutorial ID
@RequestMapping("/api/tutorial/tutorialInfo")
public ResponseEntity<Tutorial> getTutorialDetails(@RequestParam Long id) {
    // Get tutorial details from the database
    Tutorial tutorial = tutorialService.getTutorialDetails(id);
    if (tutorial == null) {
        // Throw exception if no tutorial found
        throw new TutorialNotFoundException("No tutorial found with ID " + id);
    }
    // Return the tutorial details
    return ResponseEntity.ok(tutorial);
}
```

The above code illustrates the process from front-end to back-end in searching tutorials and viewing tutorial details. The front end uses Vue.js to interact with the backend, which is built using Spring MVC. The data is transmitted between the front end and backend using REST APIs, and error handling is implemented to provide feedback to users.

6.2. AI analyzes dietary records

In our system, we use AJAX for asynchronous communication between the frontend and backend. Spring MVC is responsible for handling backend requests, business logic, data validation, and generating response data. Additionally, we use Alibaba Cloud's large model API to implement the AI analysis functionality.


<<control>>
AIAnalysisControl
+ getRecordInfo([recordID])
+ getHistoricalData(UserID)
+ getAIAnalysis([records])
+ saveAnalysisResult(Analysis)
<<entity>>
VIP
+ userID:string
+ userName:string
- password:string
+ UserType:string
+ gender:string
- accountBalance:double
+ dietaryRecordUploaded:List<DietRecord>
+ currentSelectPlan:DietPlan
+ selectRecordToBeAnalysed()
+ getAIAnalysisResults()
+ getHistoryAIAnalysis()
+ saveResult()
<<entity>>
Analysis
- analysisID:string
+ dietaryRecordsID:list of recordIDs
+ result:string
+ saveAnalysis()
<<boundary>>
DietaryRecordPage
- currentUserID:string
+ recordForm:DietRecord
- currentAnalysisID:string
+ currentAnalysisReculs:Analysis
+ UploadButton:Button
+ EditButton:Button
+ DeleteButton:Button
+ SelectAnalysisButton:Button
+ submitRecord()
+ displayRecordForm()
+ submitAnalysisApply()
+ displayAnalysisResults()
<<entity>>
DietRecord
+ recordID:string
+ date:Date
+ mealTime:Time
+ nutrientComposition:string
+ nutrientQuantities:int
+ mealPhoto: string
+ AIAnalysis:Analysis
+ returnRecord()



The use case class diagram and sequence diagram are shown above.

1. The user accesses the dietary records page to view historical analysis data

The user accesses the "Dietary Records" page, which is dynamically loaded by the frontend framework (Vue.js) and displays the user's historical analysis records. The frontend page sends an HTTP request to the backend via AJAX, submitting the user's authentication token to retrieve the relevant historical data.

2. The dietary records page sends the request to the AI analysis controller class

When the user initiates a request to view historical records, the frontend encapsulates the user's token into the request body and sends it to the backend controller (`AIAnalysisControl` class) via the Spring MVC framework. The request uses the POST method. The backend controller parses the request data and passes it to the service layer for database query operations.

3. The AI analysis controller class receives the request and queries the database for historical records

Upon receiving the query request from the frontend, the Spring MVC controller invokes the service layer to process the business logic. The service layer interacts with the database using an ORM framework (Hibernate) to query the user's corresponding historical analysis records and returns the results to the controller class.

4. Returning historical records data and displaying it on the frontend

The historical records data in the database (including analysis IDs, related record IDs, analysis content, etc.) is returned to the frontend through the Spring MVC controller. The controller uses the `@ResponseBody` annotation to package the data into JSON format, which is then sent to the frontend application. The frontend framework parses this data and dynamically displays the results on the "Dietary Records" page using data binding techniques (e.g., Vue.js's `v-bind`). Users can browse the relevant information.

5. The user selects records and initiates an AI analysis request

After reviewing the historical records, the user can select certain records for AI analysis. The user selects the records on the frontend page and clicks the "Submit Analysis" button, triggering an AJAX request that sends the selected record IDs and the user's token to the backend controller class.

6. The AI analysis controller class retrieves record information and calls an external API

After receiving the analysis request from the frontend, the controller class retrieves the detailed information of the corresponding records from the database by invoking the `getRecordInfo(recordID)` method. Subsequently, it calls the external API (Alibaba Cloud Large Model API) to complete the AI analysis request. The external API is implemented using a REST API interface, with data exchanged in JSON format.

7. The external API returns AI analysis results

The external API processes the request, completes the analysis, and returns the results in JSON format to the backend controller class. The controller class parses the returned data and packages it into a frontend-compatible format, which is sent back to the frontend through Spring MVC.

8. The frontend displays the analysis results

After receiving the AI analysis results, the frontend uses dynamic rendering techniques to display the analysis results on the page, helping the user intuitively understand the analysis information.

9. The user chooses to save the analysis results

The user can choose to save the AI analysis results in the system. When the user clicks the "Save" button, the frontend triggers another AJAX request, sending the analysis results and the user's token to the backend controller class.

10. The AI analysis controller class saves the analysis results

After receiving the save request, the backend controller class interacts with the database through the ORM framework to store the analysis results in the database.

11. The database returns the save result and provides feedback to the frontend

After the database executes the save operation, it returns the status (success or failure) to the controller class. The controller class packages the result into JSON format and sends it back to the frontend via Spring MVC. The frontend, upon receiving the result, updates the page state using a state management tool, displaying a success or failure message (e.g., using a Bootstrap modal or a custom message box).

The corresponding pseudo-code is as follows:

- User goes to page, sends request to view history

```
async loadHistrical() {
  try {
    const response = await axios.post("api/dietaryRecord/loadHistrical", {
      token: this.token
    });
    this.records = response.data.records;
  } catch (error) {
    console.error("Error loading records:", error);
  }
},
```

- Back-end querying the database to get historical analysis results
  - Controller Layer

```
@PostMapping("api/dietaryRecord/loadHistrical")
public ResponseEntity<List<Analysis>> loadRecords(@RequestBody TokenRequest request) {
    String userID = userService.getUserIdByToken(request.getToken());
    if (userID == null) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
    }
    List<Analysis> results = analysisService.getAllAnalysis(request.getToken());
    return ResponseEntity.ok(results);
}
```

  - Service Layer

```
 public List<Analysis> getAllAnalysisByUserId(String userID) {
        return analysisRepository.findByUserID(userID);
    }
```

- The front-end submits a request for AI analysis

```
async analyzeRecord(recordId) {
    try {
      const response = await axios.post("api/dietaryRecord/getAIanalysis
        token: this.token,
        recordId: [record_id_01,
                   record_id_02,
                   record_id_03]
      });
      this.analysisResult = response.data.analysisResult;
    } catch (error) {
      console.error("Failed to analyze record:", error);
    }
  }
```

- Backend Processing Analysis Request
  - Controller Layer

```
@PostMapping("api/dietaryRecord/getAIanalysis")
public ResponseEntity<Analysis> analyzeRecord(@RequestBody AnalysisRequest request) {
    String recordContent = recordService.getRecordContent(request.getRecordId());
    String analysisResult = analysisService.analyze(recordContent);
    return ResponseEntity.ok(new Analysis(analysisResult));
}
```

- Service Layer

```
public String getRecordContent(String recordId) {
    Record record = recordRepository.findById(recordId).orElseThrow(() -> new
RuntimeException("Record not found"));
    return record.getContent();
}
public String analyze(String recordContent) {
    // Create a RestTemplate instance
    RestTemplate restTemplate = new RestTemplate();

    // Construct the request header
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    headers.set("Authorization", "Bearer " + API_KEY);

    // Construct the request body
    Map<String, Object> requestBody = new HashMap<>();
    requestBody.put("content", recordContent);

    HttpEntity<Map<String, Object>> requestEntity = new HttpEntity<>(requestBody, headers);

    try {
        // Calling external APIs
        ResponseEntity<String> response = restTemplate.exchange(
            AI_API_URL,
            HttpMethod.POST,
            requestEntity,
            String.class
        );

        // Parses and returns the API response
        return parseResponse(response.getBody());
    } catch (Exception e) {
        // Handle failed calls
        e.printStackTrace();
        return "Error: Unable to analyze the content.";
    }
}
```

- The front-end saves the results of the analysis

```
  async saveAnalysis() {
    try {
      const response = await axios.post("/api/saveAnalysis", {
        token: this.token,
        Analysis: this.Analysis
```

```
      });
      if (response.data.success) {
        alert("Analysis result saved successfully!");
      } else {
        alert("Failed to save analysis result.");
      }
    } catch (error) {
      console.error("Failed to save analysis result:", error);
    }
  }
```

- Back-end saving of analysis results
  - Controller Layer

```
@PostMapping("/saveAnalysis")
public ResponseEntity<String> saveAnalysis(@RequestBody SaveAnalysisRequest request) {
    boolean success =
analysisService.saveAnalysisResult(userService.getUserIdByToken(request.getToken()),
request.getAnalysisResult());
    return ResponseEntity.ok(new SaveResponse(success));
}
```

  - Service Layer

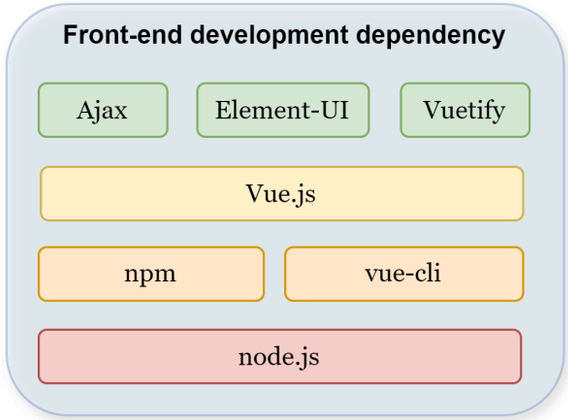
```
public boolean saveAnalysisResult(String userID, String analysisResult) {
    Analysis analysis = new Analysis();
    analysis.setUserID(userID);
    analysis.setResult(analysisResult);
    analysisRepository.save(analysis);
    return true;
}
```

7. Progress on Prototyping

7.1. Front-end Prototyping

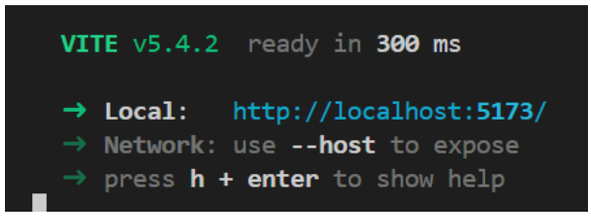
In this project, we use VUE3 framework to develop the front-end so as to do the web design of `Keep Fit`. At the same time, we still use `npm` as the front-end project package management tool, and use `Vite` as the front-end build tool for rapid development and build.

The dependency graph is as follows:



The way to run the front end:

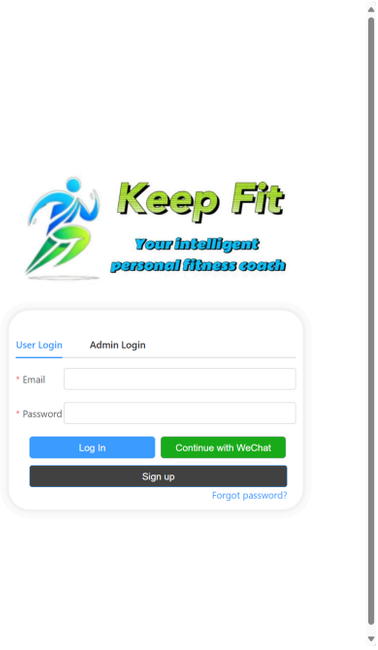
We type the `npm run dev` command in the terminal, and we can access our page on port 5137 on localhost.



Taking the login page as an example, our front-end code is as follows:

```
<div class="login-container">
    <div class="content">
        <div class="title-img"></div>
        <el-card class="login-card">
            <el-tabs v-model="activeName" class="tabs">
                <el-tab-pane label="User Login" name="user">
                    <el-form :model="LogInForm" :rules="rules" label-position="left" label-width="70px">
                        <el-form-item label="Email" prop="email">
                            <el-input v-model="LogInForm.email" />
                        </el-form-item>
                        <el-form-item label="Password" prop="password">
                            <el-input v-model="LogInForm.password" type="password" show-password />
                        </el-form-item>
                        <el-row :gutter="8">
                            <el-col :span="11">
                                <el-button type="primary" class="login-button" @click="submitForm">Log
                                    In</el-button>
                            </el-col>
                            <el-col :span="11">
                                <el-button type="info" class="wechat-login-button" @click="wechatLogin">Continue
                                    with WeChat</el-button>
                            </el-col>
                        </el-row>
                        <el-row>
                            <el-button type="primary" class="signup-button" @click="submitForm">Sign
                                up</el-button>
                        </el-row>
                        <el-link type="primary" @click="signUp" class="forgot-password-row">Forgot
                            password?</el-link>
```

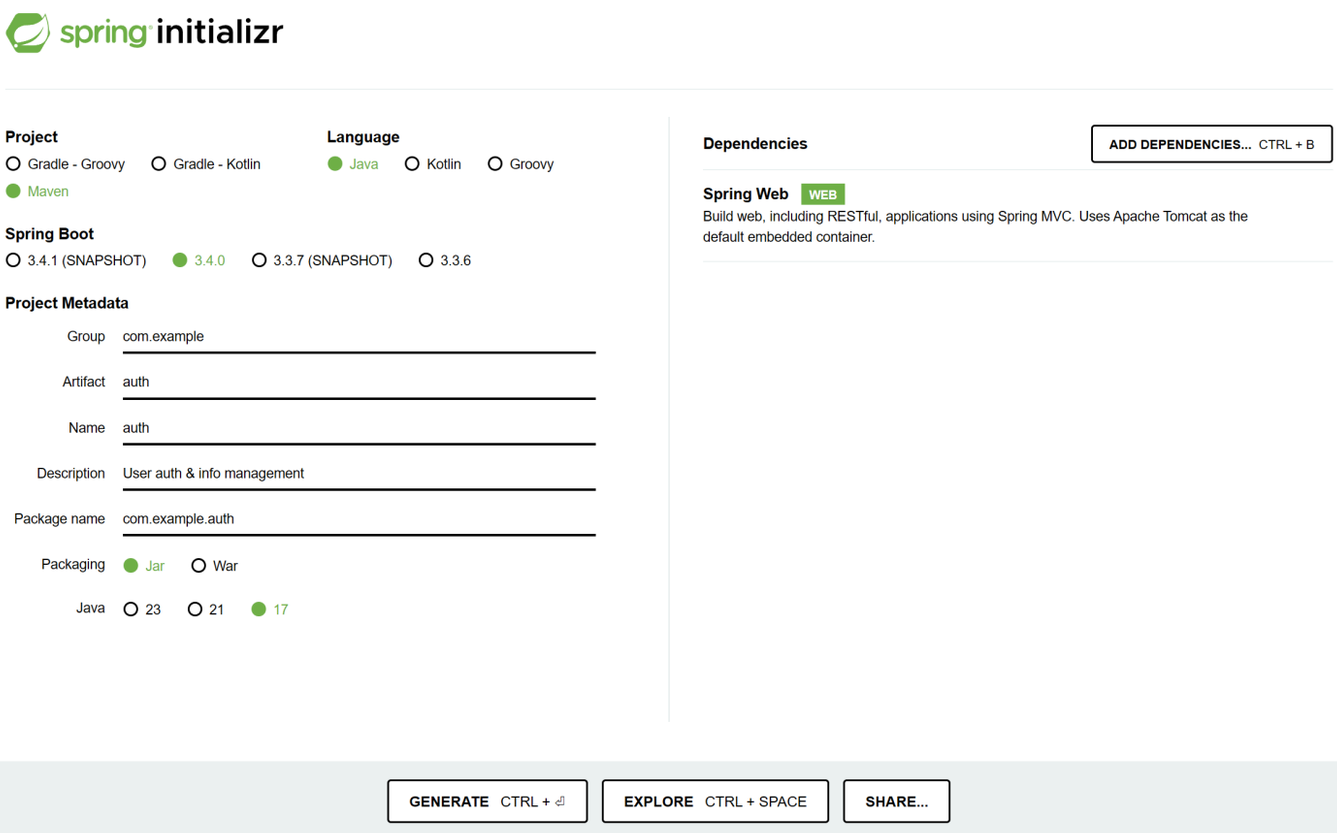
Our login page was designed based on a snapshot of the system, adding the administrator login method, and our project's logo. The page is shown in the figure:



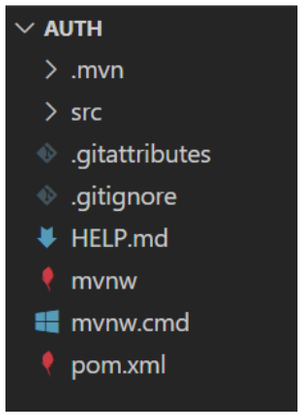
7.2. Back-end Prototyping

For the back-end of our system, we use Java and the Spring Boot framework for development. At this prototyping stage, we choose to ignore the distribution problem related to Spring Cloud first and focus on the implementation of the functionality of each microservice. We use REST API to communicate between the front-end and the back-end. In the rest of this section, we will demonstrate our progress on the back-end prototyping with a focus on the user authentication microservice responsible for the user login, sign-up, and so on. Be aware that we are not going to cover every detail of this prototyping, rather than giving a sense of how it is implemented.

We begin our work by creating a Spring Boot microservice called `auth`:



which is structured as follows:

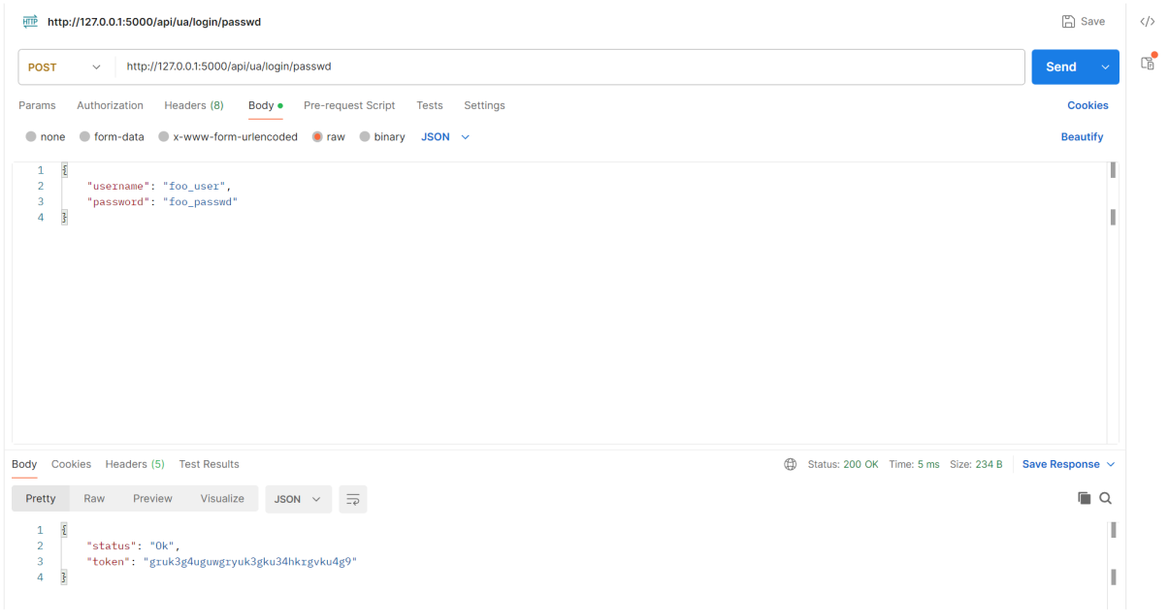


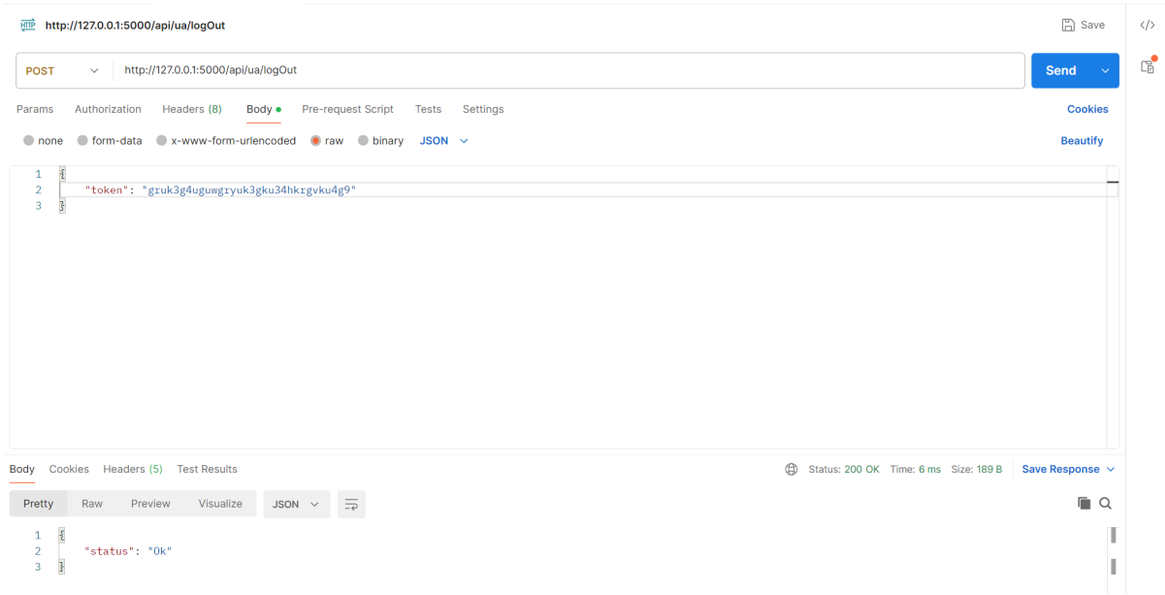
Then we start to add our implementation. Take the function `login` as an example, we create a class called `LoginController`, and add our APIs like `/api/ua/login/passwd` or `/api/ua/logOut`:

```
@RestController
public class LoginController {

    @Autowired
    private TokenStore tokenStore;

    @Autowired
    private AuthAccountService authAccountService;

    @Autowired
    private PermissionFeignClient permissionFeignClient;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @PostMapping("/api/ua/login/passwd")
    public ServerResponseEntity<TokenInfoVO> login(
            @Valid @RequestBody AuthenticationDTO authenticationDTO) {

        ServerResponseEntity<UserInfoInTokenBO> userInfoInTokenResponse =
authAccountService.getUserInfoInTokenByInputUserNameAndPassword(authenticationDTO.getPrincipal(),a
uthenticationDTO.getCredentials(), authenticationDTO.getSysType());

        if (!userInfoInTokenResponse.isSuccess()) {
            return ServerResponseEntity.transform(userInfoInTokenResponse);
        }

        UserInfoInTokenBO data = userInfoInTokenResponse.getData();

        ClearUserPermissionsCacheDTO clearUserPermissionsCacheDTO = new
ClearUserPermissionsCacheDTO();
        clearUserPermissionsCacheDTO.setSysType(data.getSysType());
        clearUserPermissionsCacheDTO.setUserId(data.getUserId());

        ServerResponseEntity<Void> clearResponseEntity =
permissionFeignClient.clearUserPermissionsCache(clearUserPermissionsCacheDTO);

        if (!clearResponseEntity.isSuccess()) {
            return ServerResponseEntity.fail(ResponseEnum.UNAUTHORIZED);
```

```
        }

        return ServerResponseEntity.success(tokenStore.storeAndGetVo(data));
    }

    @PostMapping("/loginOut")
    public ServerResponseEntity<TokenInfoVO> loginOut() {
        UserInfoInTokenBO userInfoInToken = AuthUserContext.get();
        ClearUserPermissionsCacheDTO clearUserPermissionsCacheDTO = new
ClearUserPermissionsCacheDTO();
        clearUserPermissionsCacheDTO.setSysType(userInfoInToken.getSysType());
        clearUserPermissionsCacheDTO.setUserId(userInfoInToken.getUserId());

        permissionFeignClient.clearUserPermissionsCache(clearUserPermissionsCacheDTO);

        tokenStore.deleteAllToken(userInfoInToken.getSysType().toString(),
userInfoInToken.getUid());
        return ServerResponseEntity.success();
    }
}
```

So now, we can call the APIs in tools like ***Postman*** for testing.





8. Open Issues

8.1. Data consistency and transaction management

- **Problem Description**

How to ensure data consistency and transaction management during data operations between multiple microservices? For example, when a user is recharged, and multiple microservices (such as payment services, user services, and reward systems) are involved, how to ensure data consistency?

- **Challenge**

Distributed transactions (such as using Saga mode) or event-driven architecture need to be considered to ensure data consistency between microservices.

8.2. Optimization of the dynamic recommendation algorithm

- **Problem Description**

Can the recommendation system in the platform (such as fitness tutorial recommendation, diet plan recommendation, etc.) automatically adjust the recommendation strategy according to users' real-time behavior and feedback? Is the current design flexible enough to handle complex transactions?

- **Challenge**

We need to consider how to introduce more machine learning algorithms (such as deep learning, collaborative filtering, etc.) to optimize the recommendation algorithms, and to ensure that the recommendation system can update the recommendation content in real time and accurately according to user behavior.

8.3. Integration and security of payment systems

- **Problem Description**

Is the payment system designed to meet the modern payment security standards? For example, the platform integrates third-party payment interfaces (such as Alipay, WeChat payment, etc.), how to ensure the security of transactions and prevent the disclosure or tampering of payment information?

- **Challenge**

Further evaluation of the security strategy of the payment system is needed to ensure that all payment processes are encrypted and meet security standards such as PCI-DSS.

8.4. User behavior analysis and data privacy

- **Problem Description**

How can user behavior data (such as fitness records, eating habits, etc.) be used for data analysis and personalized recommendation on the platform? How to ensure that users' data privacy is guaranteed and avoid data abuse?

- **Challenge**

GDPR (Privacy protection regulations) in data collection, analysis and storage, and provide users with data privacy control options.

8.5. Platform performance with high concurrent processing

- **Problem Description**

As the number of users increases, how can you ensure that the platform maintains high performance despite high concurrency, especially the response time of user query tutorials, submit comments, punch in, and so on?

- **Challenge**

The performance of database query and API calls needs to be further optimized, using caching and load balancing techniques to ensure that the system can handle a large number of concurrent requests.

9. Project self-reflection

- **WeiCheng Zheng**

After a semester of study, I deeply realized the importance of the pre-design process of the project. From the initial idea to the release and operation and maintenance of the final product, every link is indispensable. At first, I had no concept of project modeling, but now I am able to skillfully draw case maps, class maps and other models. I have a macro global understanding of project development and understand the specific implementation steps.

System analysis and design are not fought alone, but the crystallization of teamwork. Thanks to my teammates for their efforts and mutual support during this period, we learned new knowledge and solved problems together. We not only accumulated deep friendship, but also made me make significant progress in teamwork. We feel very honored to have the advice and guidance given by the teacher after the mid-term defense, which will be the wealth of our life.

- **Juekai Lin**

Through the study of system analysis and design, I deeply realize that software development does not write code at the beginning, but requires sufficient preliminary design and analysis. From user needs to UML modeling, prototype design, to function and API design, and finally to achieve the function, every step is closely related. Especially in the early stage, some designs have a profound impact on the subsequent development, which also makes the later development more clear and smooth.

It was a valuable experience to cooperate with our excellent teammates, and we were honored to get guidance from our teachers. The clear division of labor and smooth communication not only exercised my teamwork ability, but also gained me rich cooperation experience and example strength. I have benefited a lot from this experience.

- **Lixin Ma**

The system analysis and design course in this semester has put me through the whole process of the system from conception to implementation, and I have a clearer understanding of the development process. I learned to visually present the system use case with the use case diagram and the use case instruction manual, and to deeply analyze the functional details with the activity map and the order diagram. These design tools allow me to better understand the system design, and will be of great use in the future.

At the same time, I have a deeper understanding of the key mechanisms in the system design, which provides a clear guidance for the future project development. Now, I know how to standardize the development process, avoid common beginner mistakes, and no longer feel confused about the design of complex systems.

- **Junhao Yang**

System analysis and design is both a practical and impressive course. In the process of agile analysis and architecture design of the project, I learned the relevant knowledge of requirements analysis and architecture design. From how to determine the target users, to how to draw UML maps, and then to analyze the rationality of the system architecture, this fruitful process has given me a more systematic understanding of the software development.

Also, the cooperation with my teammates allowed me not only to accumulated teamwork experience precious in my future career, but also to gain new ideas from others. I feel lucky to have my skills in both engineering and teamwork improved, while gaining a deeper understanding of the overall process of software development along the way.

10. Contributions

During the implementation of the project, the team members actively participate in the discussion and put into the task according to their respective interests and expertise, to ensure that the work is completed on time under the premise of high quality. The atmosphere in the team is harmonious, smooth and efficient communication, which provides an important guarantee for the smooth progress of the project. Each member performed his own duties, and made outstanding contributions to the success of the project.

Members	Part 1	Part 2	Part 3	Part 4	Part 5	Part 6	Part 7	Part 8	Part9	SCORE WEIGHT
2154286 Weicheng Zheng	√	√		√		√	√	√	√	25%
2253744 Juekai Lin	√		√	√	√	√		√	√	25%
2153085 Lixin Ma	√	√		√		√	√	√	√	25%
2154284 Junhao Yang	√		√	√		√		√	√	25%

Specifically, Weicheng Zheng is mainly responsible for the Fitness Action Coaching and Equipment SubSystem. Juekai Lin is mainly responsible for the Fitness Tutorial Subsystem. Lixin Ma is mainly responsible for the Healthy Diet Subsystem. Junhao Yang is mainly responsible for the Login and Registration Subsystem.