**CPT202 Assignment 2**

**Group Report for Software Engineering Group Project**

2023/2024 Semester 2

<Food Ordering System>

*<2024.5.5>*

*<http://47.93.24.56:10000>*

Group number: <15>

Submitted by: <member name, member ID>

Group Members:

1. <member 1, member ID>
2. <member 2, member ID>
3. <member 3, member ID>
4. < member 4, member ID >

Contents

[Introduction (Suggest 1 pages) 3](#_Toc164178129)

[Problem Statement 3](#_Toc164178130)

[Aims of The Project 3](#_Toc164178131)

[Project Scope 3](#_Toc164178132)

[User Characteristics 3](#_Toc164178133)

[Assumptions and Dependencies 3](#_Toc164178134)

[Project Risks 3](#_Toc164178135)

[Architectural Design (Suggest 2.5 pages) 4](#_Toc164178136)

[System Architecture 4](#_Toc164178137)

[Frontend System 4](#_Toc164178138)

[Backend System 4](#_Toc164178139)

[High-level Database Design 4](#_Toc164178140)

[Software Design (Suggest 2.5 page) 5](#_Toc164178141)

[Software Modules 5](#_Toc164178142)

[High-level Process Flow 5](#_Toc164178143)

[Software Support Services 5](#_Toc164178144)

[Coding Structure and Convention 5](#_Toc164178145)

[Software Configuration and Production Environment 5](#_Toc164178146)

[Software testing (Suggest 1.5 page) 5](#_Toc164178147)

[Conclusion and future work (Suggest 0.5 pages) 5](#_Toc164178148)

[References (If needed, not count into the limit of 8 pages) 5](#_Toc164178149)

[Appendix (Not count into the limit of 8 pages) 6](#_Toc164178150)

# Introduction

The project at hand is the development of a web-based food ordering system for a newly established restaurant. The objective is to create a user-friendly, efficient, and secure platform that facilitates easy and customizable ordering of food items. The system is intended to cater to both the restaurant's operational needs and the customers’ convenience, enhancing the overall dining experience through digital means.

## Problem Statement

The current marketplace for food delivery services is highly competitive and technologically driven. A new restaurant must offer not only exceptional food but also a seamless ordering experience to establish and maintain a competitive edge. The lack of an efficient food ordering system can hinder a restaurant's ability to attract and retain customers, manage orders effectively, and scale operations. The proposed system aims to address these challenges by implementing a comprehensive solution that supports order customization, real-time order tracking, and user management.

## Aims of The Project

The project aims to achieve the following objectives:

1.Develop a web-based food ordering system that allows for detailed customization of orders including size and condiments.

2.Implement a pricing algorithm that calculates the cost based on the type, size, and customization of the food ordered.

3.Create a user registration and management system for new customers.

4.Design a delivery management system to efficiently manage the logistics of food delivery.

5.Enable customers to rate their ordering experience, thus providing feedback to the restaurant.

6.Ensure the system complies with relevant legal standards and includes necessary agreements for user data protection and service terms.

## Project Scope

The scope of this project includes:

1.Design and development of a web application interface for customers and restaurant managers.

2.Implementation of a backend system that handles order processing, customer management, and report generation.

3.Integration of payment processing capabilities.

4.Setting up a database to store user data, order history, and food item details.

5.Development of security measures to protect user information and transaction data.

Compliance with legal requirements for online businesses and food service providers.

## User Characteristics

1.Restaurant Manager: Tech-savvy individuals responsible for maintaining the system, updating food prices, and accessing business analytics.

2.Customers: Individuals of varying tech proficiency who require a simple, intuitive interface to place orders, track delivery, and manage their accounts.

## Assumptions and Dependencies

* The project assumes that the necessary infrastructure for hosting the web application, including servers and databases, is available and capable of handling the expected load.
* The project depends on third-party services for payment processing and may require integrations with existing delivery logistics providers.

## Project Risks

1.Technical Risks: Challenges in integrating various systems (e.g., payment gateways, delivery tracking).

2.Operational Risks: Potential delays in project milestones due to unforeseen technical difficulties.

3.Market Risks: The system might not be as readily accepted by users as anticipated, affecting the projected growth.

4.Security Risks: Risks associated with data breaches and security threats, necessitating robust security measures.

5.Compliance Risks: Failure to adhere to all legal requirements could lead to legal repercussions and damage to reputation.

# Architectural Design

*Describe the requirements that has impacts on your architectural design*

The architectural design of our web-based food ordering system is influenced by several key requirements:

1.Scalability: The system must handle varying loads, with the ability to scale during peak times.

2.Security: Protecting user data and transactions is critical, necessitating secure communications and data storage.

3.Reliability: The system should ensure high availability and fault tolerance to manage potential service disruptions effectively.

4.Usability: The front-end must provide a user-friendly interface that accommodates a diverse customer base.

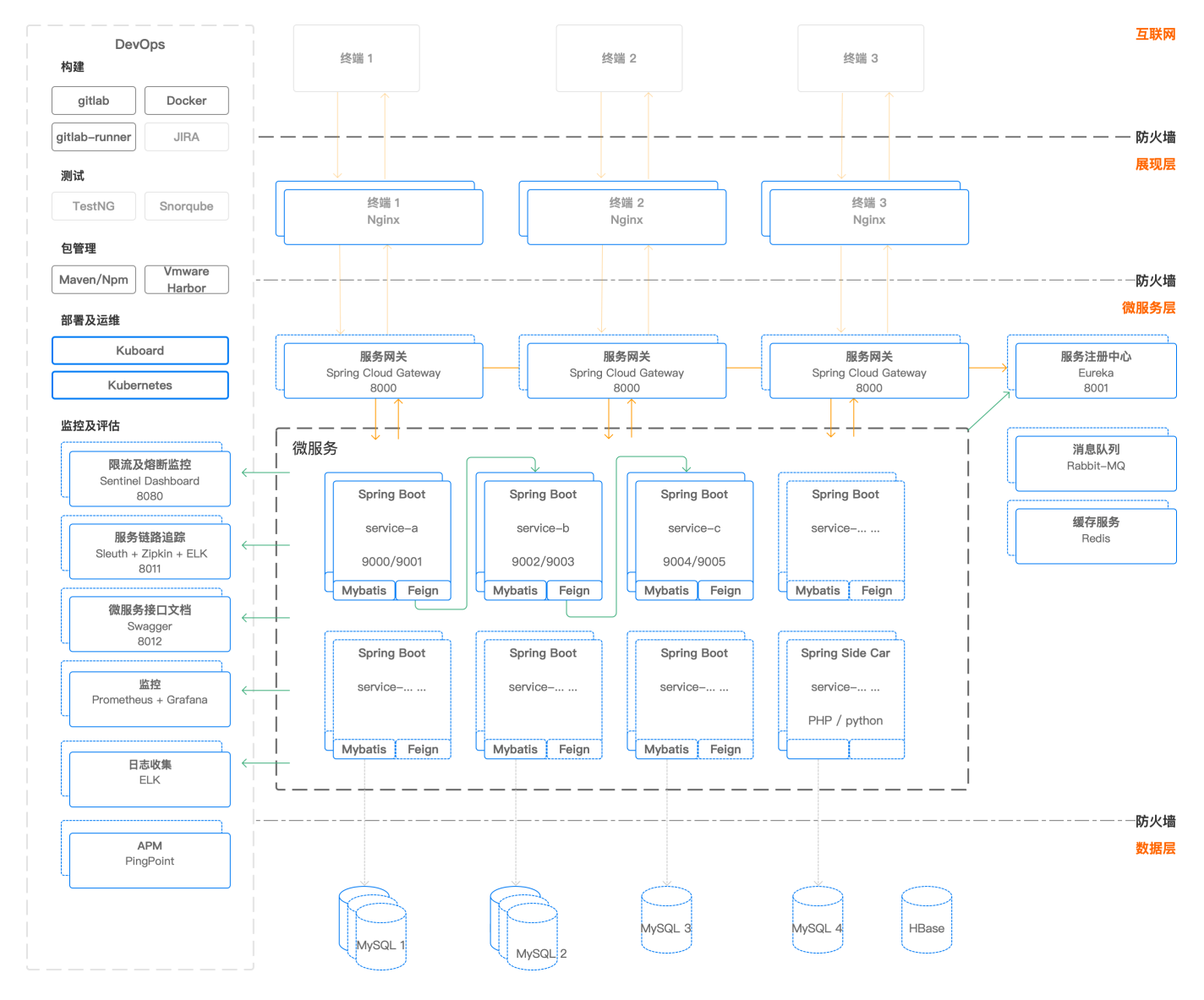
5.Performance: Response times should be minimized through efficient processing and data handling.

## System Architecture

**Overview**

The system architecture is designed to be robust, scalable, and maintainable, incorporating modern technologies and patterns to meet the diverse needs of the system. The architecture includes the following components:

* Frontend: HTML, CSS, JavaScript
* Backend: Spring Cloud
* Service Discovery: Nacos
* API Gateway: Spring Cloud Gateway
* Database: MySQL
* Message Queue: RabbitMQ
* Caching Server: Redis
* Rate Limiting and Circuit Breaker: Sentinel



The diagram illustrates the interactions between different system components. The frontend communicates with the backend through the API gateway, which routes requests to appropriate microservices managed by Spring Cloud. Nacos is used for service discovery, RabbitMQ for asynchronous message handling, Redis for caching, and Sentinel for rate limiting and fault tolerance.

**Justification**

* Spring Cloud offers a suite of tools for building some of the common patterns in distributed systems (e.g., configuration management, service discovery, circuit breakers).
* Nacos provides dynamic service discovery and configuration management, crucial for microservices architecture.
* Spring Cloud Gateway routes API requests and handles cross-cutting concerns like security, monitoring, and resilience.
* MySQL is a robust, relational database management system that supports complex queries and transactional data consistency.
* RabbitMQ facilitates asynchronous processing, reducing wait times for end users.
* Redis serves as a caching layer to enhance performance by minimizing database hits.
* Sentinel provides rate limiting and circuit breaking to ensure stability and resilience.

## 

## Frontend System

**Overview**

The frontend utilizes HTML, CSS, and JavaScript to deliver a responsive and accessible user interface. The choice of technologies ensures compatibility across different devices and browsers.

**Technologies**

* Jquery could be incorporated to manage the stateful user interactions dynamically and efficiently.
* Bootstrap for rapid, responsive UI development.
* Ajax for promise-based HTTP client to handle requests to the backend.

**Justification**

* Jquery offers a component-based architecture, making the frontend scalable and maintainable.
* Bootstrap facilitates quick development of aesthetically pleasing interfaces that are automatically responsive.
* Ajax enhances the handling of HTTP requests with features like intercepting requests and responses, which simplifies error handling and promotes cleaner code.

## Backend System

**Overview**

The backend is structured around Spring Cloud, leveraging microservices architecture to ensure scalability and flexibility.

**Technologies**

* Spring Boot for creating standalone, production-grade Spring based Applications easily.
* Spring Security for authentication and authorization.
* Spring Data JPA for database interactions.

**Justification**

* Spring Boot simplifies the development of new services by providing a wide range of out-of-the-box functionalities for web development.
* Spring Security offers comprehensive security and authentication mechanisms, crucial for protecting user data.
* Spring Data JPA abstracts boilerplate CRUD operations, accelerating development and reducing the likelihood of errors.

## High-level Database Design



Description

The database design includes tables for Users, Orders, Foods, Order\_Details, and Ratings. Relationships are as follows:

Users have many Orders.

Orders contain many Order\_Details, linked to Foods.

Orders can have one Rating.

Explanation

Users table stores customer information necessary for order tracking and marketing.

Orders table records each transaction, essential for financial reporting and customer service.

Foods table allows for dynamic menu management.

Order\_Details link food items to orders, supporting customization and pricing calculations.

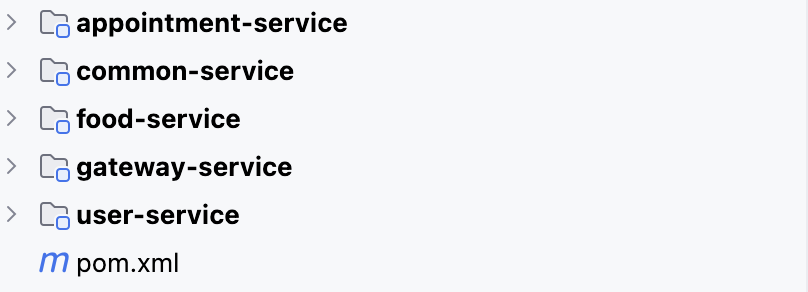
Ratings provide feedback per order, valuable for quality control and service improvement.

# Software Design

## Software Modules

**Modularization Overview**

Our software is modularized into distinct layers and components to promote separation of concerns, ease of maintenance, and scalability. The primary modules are:



1. User Management: Handles registration, login, and user profile management.
2. Order Management: Manages order creation, modification, and history tracking.
3. Product Management: Deals with menu items, pricing updates, and availability.
4. Delivery Management: Coordinates order delivery logistics.
5. Payment Processing: Integrates with third-party services for secure payment handling.
6. Rating System: Facilitates the collection and management of customer feedback.

Reporting: Generates statistical reports for business insights.

Software Design

Software Modules

**Modularization Overview**

Our software is modularized into distinct layers and components to promote separation of concerns, ease of maintenance, and scalability. The primary modules are:

User Management: Handles registration, login, and user profile management.

Order Management: Manages order creation, modification, and history tracking.

Product Management: Deals with menu items, pricing updates, and availability.

Delivery Management: Coordinates order delivery logistics.

Payment Processing: Integrates with third-party services for secure payment handling.

Rating System: Facilitates the collection and management of customer feedback.

Reporting: Generates statistical reports for business insights.

**Alternate Solution**

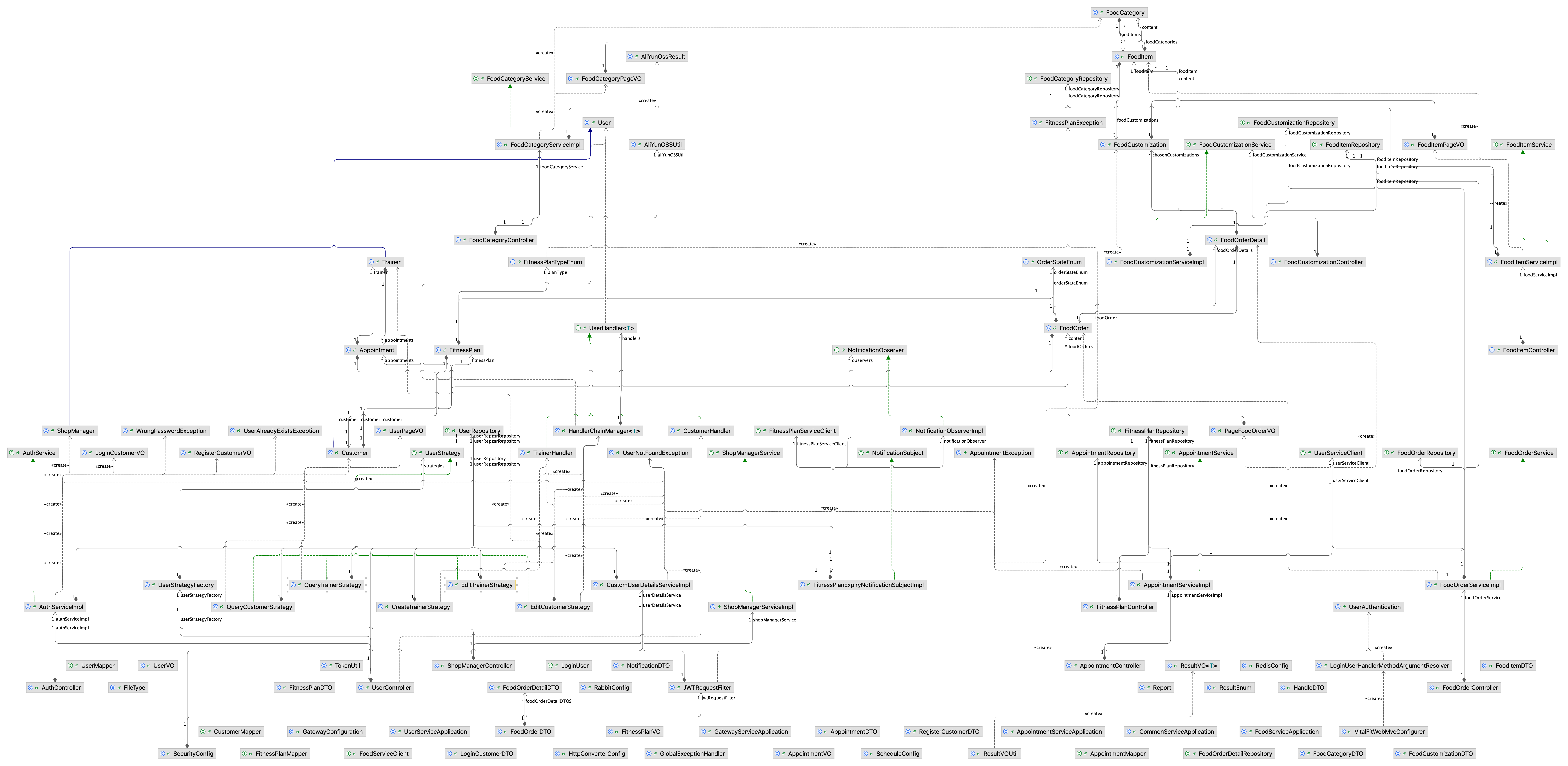
An alternative modularization approach could involve a more granular breakdown, such as separating the payment module into individual components for different payment methods. This would allow more focused scaling and updating but could complicate the system architecture.

**Justification**

The chosen modularization strikes a balance between complexity and maintainability, supporting independent development and testing of features without excessive overhead. It simplifies integration and allows for easier scaling of heavily used parts like order and user management.

## High-level Process Flow

The core functionalities include user registration and login, food ordering, order tracking, and feedback management. Each functionality is supported by its respective module, interacting through defined interfaces.



**Description**

* User Registration/Login: Users register or log in through the user management module, which interacts with the database to store/retrieve user data.
* Food Ordering: Customers select items, customize orders, and submit them via the order management module, which calculates prices and updates the database.
* Order Tracking: Customers can view the status of their orders through interfaces provided by the delivery management module.
* Feedback Management: After order completion, customers can rate their experience through the rating system module, influencing future business decisions.

## Software Support Services

**Database Related Services**

* ORM Tools: Use of Spring Data JPA to abstract database interactions, simplifying CRUD operations.
* Transaction Management: Ensuring data integrity and consistency across business operations.

**Security Related Services**

* Authentication and Authorization: Spring Security for managing secure access to various system parts.
* Data Encryption: Encrypt sensitive information in transit and at rest.

**Message Queue Related Services**

* Asynchronous Processing: RabbitMQ for handling non-immediate background tasks like email notifications and order status updates.

**Webpage Navigation Related Services**

* SPA Routing: Utilizing React Router for managing single-page application navigation without full page reloads.

## Coding Structure and Convention

**Structure**

* MVC Architecture: Organized into models, views, and controllers to separate the application's concerns.
* Repository Pattern: Used for data access layers to decouple the business logic from the data access code.

**Convention**

* Naming Conventions: Consistent and descriptive names for variables, functions, and classes.
* Commenting: Extensive use of comments to explain the purpose of code blocks, especially for complex logic.
* Error Handling: Standardized approach for managing exceptions and errors.

## Software Configuration and Production Environment

**Software Configuration**

* Version Control: Git for source code management with branching strategies to manage features, fixes, and releases.
* Dependency Management: Maven to handle library dependencies.

**Server Configuration**

* Microservices: Each module runs as a separate microservice, allowing individual scaling and deployment.
* Load Balancers: Used to distribute traffic among instances and improve system responsiveness.

**Containerization**

* Docker: Each component of the application is containerized, encapsulating its environment and dependencies.
* Kubernetes: Manages the deployment, scaling, and operations of these containers across clusters.

**Explanation**

Containerization simplifies deployment by ensuring that software runs reliably when moved from one computing environment to another. Kubernetes enhances this by automating scaling and management, making the system more resilient and easier to manage.

# Software testing

**Overview**

Software testing is crucial for ensuring that the food ordering system functions correctly, meets requirements, and provides a reliable user experience. The testing process is divided into three main categories: unit testing, integration testing, and acceptance testing. Each testing stage has its specific focus and methodologies.

**Unit Testing**

**Process**

Unit testing involves testing individual components or methods to ensure they perform as expected in isolation. Developers write these tests during the coding phase to catch bugs early and facilitate refactoring.

* Tools: Use of JUnit for Java backend services and Jest for JavaScript frontend components.
* Methodology: Test-Driven Development (TDD) approach where tests are written before the actual code.

**Test Data and Results**

**Example:**

* Module: User Registration
* Function: createUser(User user)
* Test Case: Ensure the user is saved with the correct details.
* Input: User object ({username: "testUser", password: "password123", email: "test@example.com"})
* Expected Result: Successful creation and storage of user details.
* Actual Result: As expected.

Results: All unit tests passed, confirming that individual components behave correctly under various scenarios.

**Integration Testing**

**Process**

Integration testing examines the interactions between integrated units/modules to expose faults in their interfaces and interactions. This is done after unit testing.

* Tools: Spring Boot Test for backend integration; React Testing Library for frontend.
* Methodology: Combines the modules that passed unit testing to verify that they function together as expected.

**Test Data and Results**

**Example:**

* Modules: User Management and Order Management
* Functionality: Placing an order as a logged-in user
* Test Case: Verify that the order is correctly linked to the user who placed it.
* Input: User ID and Order Details
* Expected Result: The order is placed under the correct user ID with accurate order details.
* Actual Result: As expected.

Results: Integration tests validate the correct interaction between user authentication and order processing, with no discrepancies found.

**Acceptance Testing**

**Process**

Acceptance testing is conducted with real-world scenarios to ensure the system meets the business requirements and is ready for deployment. This testing is typically done by the quality assurance team with input from stakeholders.

* Tools: Selenium for automated browser testing, Postman for API testing.
* Methodology: Based on user stories and use cases agreed upon during the system design phase.

**Test Data and Results**

**Example:**

* Scenario: A customer places an order and then checks the order status.
* Test Case: Ensure that the order reflects correctly in the user's order history and status updates are visible.
* Input: Login credentials, food selection, order confirmation.
* Expected Result: The user sees their order in the "Order History" with a status of "Pending."
* Actual Result: As expected.

Results: Acceptance testing confirms that the system fulfills the specified requirements and behaves predictably in simulated real-world usage conditions.

**Summary**

The comprehensive testing strategy encompassing unit, integration, and acceptance tests ensures that each component functions individually as intended, interacts correctly with other parts of the system, and meets the overall business requirements. Continuous integration tools automate running these tests during development to catch issues early. This rigorous testing process leads to a robust, reliable, and user-friendly food ordering system.

# Conclusion and future work

**Conclusion**

Throughout the project, our team has successfully designed and developed a comprehensive web-based food ordering system tailored to the needs of a modern restaurant. This system encompasses several significant modules such as user management, order management, product management, and delivery logistics. By adopting a microservices architecture, we ensured the system is scalable, maintainable, and robust enough to handle different operational scales and complexities.

**Achievements**

1.System Design and Implementation: We have completed the architectural and software design, ensuring all components work harmoniously. The frontend offers a user-friendly interface while the backend ensures efficient data processing and service delivery.

2.Development of Key Features: Features like customizable food ordering, real-time order tracking, and user registration and management are fully functional, providing a solid foundation for the restaurant’s operations.

3.Testing and Validation: Comprehensive testing phases, including unit, integration, and acceptance tests, have been conducted to ensure reliability and functionality, with all systems performing as expected.

**Current Problems**

1.User Experience (UX) Challenges: While functional, the user interface needs further refinement to enhance usability and aesthetic appeal, particularly on mobile devices.

2.Integration with External Services: Some issues have arisen in the seamless integration of third-party services, such as payment gateways and delivery tracking systems, which require additional debugging and testing.

3.Performance Optimization: As the system scales, we've noticed some performance bottlenecks, particularly during peak usage times, which necessitate further optimization.

**Future Work**

1.Enhancing User Interface and Experience: We plan to overhaul the frontend to make it more intuitive and responsive, ensuring a seamless experience across all devices and platforms.

2.Expanding Payment Options: To accommodate a broader range of customers, we aim to integrate more payment methods and enhance the security features associated with transactions.

3.Advanced Analytics and Reporting: Developing more sophisticated analytics capabilities to provide detailed insights into sales trends, customer behavior, and operational efficiency.

4.Incorporating AI and ML: We plan to implement AI-driven recommendations and machine learning models to predict order trends and manage inventory more effectively.

5.Continuous Performance Tuning: Ongoing efforts will be made to optimize the system's performance, focusing on load balancing, caching strategies, and efficient data handling techniques.

In conclusion, while we have made significant strides in developing a robust food ordering system, there is exciting potential for future enhancements that could further streamline operations, improve customer satisfaction, and expand the system’s capabilities. Our team is committed to continuous improvement and innovation to ensure the success and scalability of the system.

# References (If needed, not count into the limit of 8 pages)

**Books**

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Journal Articles

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine.

**Online Sources**

Spring Documentation. (2023). Spring Framework Documentation. Retrieved from [https://spring.io/projects/spring-framework](https://spring.io/projects/spring-framework" \t "/Users/youranxu/Documents\\x/_new)

React Documentation. (2023). React - A JavaScript library for building user interfaces. Retrieved from [https://reactjs.org/](https://reactjs.org/" \t "/Users/youranxu/Documents\\x/_new)

Docker Docs. (2023). Overview of Docker Compose. Retrieved from https://docs.docker.com/compose/

MySQL Documentation. (2023). MySQL Developer Documentation. Retrieved from [https://dev.mysql.com/doc/](https://dev.mysql.com/doc/" \t "/Users/youranxu/Documents\\x/_new)

RabbitMQ. (2023). RabbitMQ Official Website. Retrieved from [https://www.rabbitmq.com/](https://www.rabbitmq.com/" \t "/Users/youranxu/Documents\\x/_new)

Redis. (2023). Redis Documentation. Retrieved from https://redis.io/documentation

**Software and Tools**

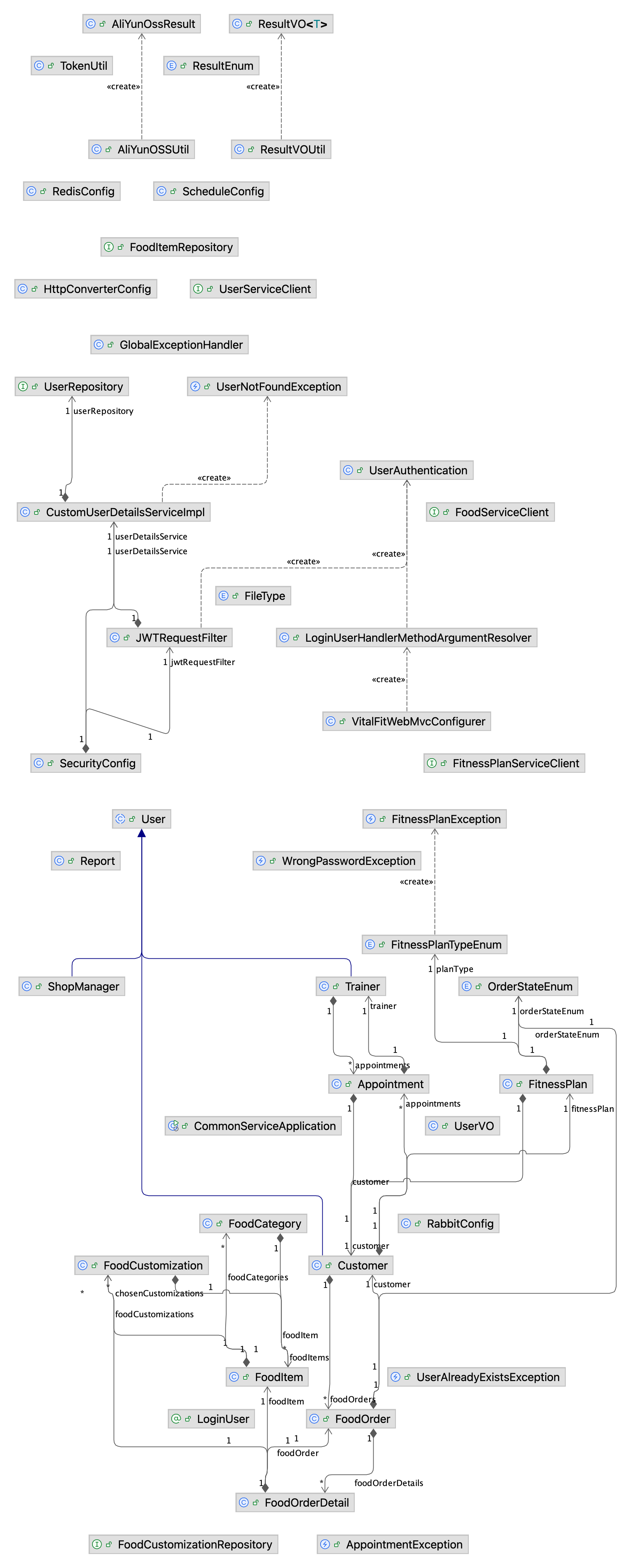
JUnit. (2023). JUnit 5 User Guide. Retrieved from https://junit.org/junit5/docs/current/user-guide/

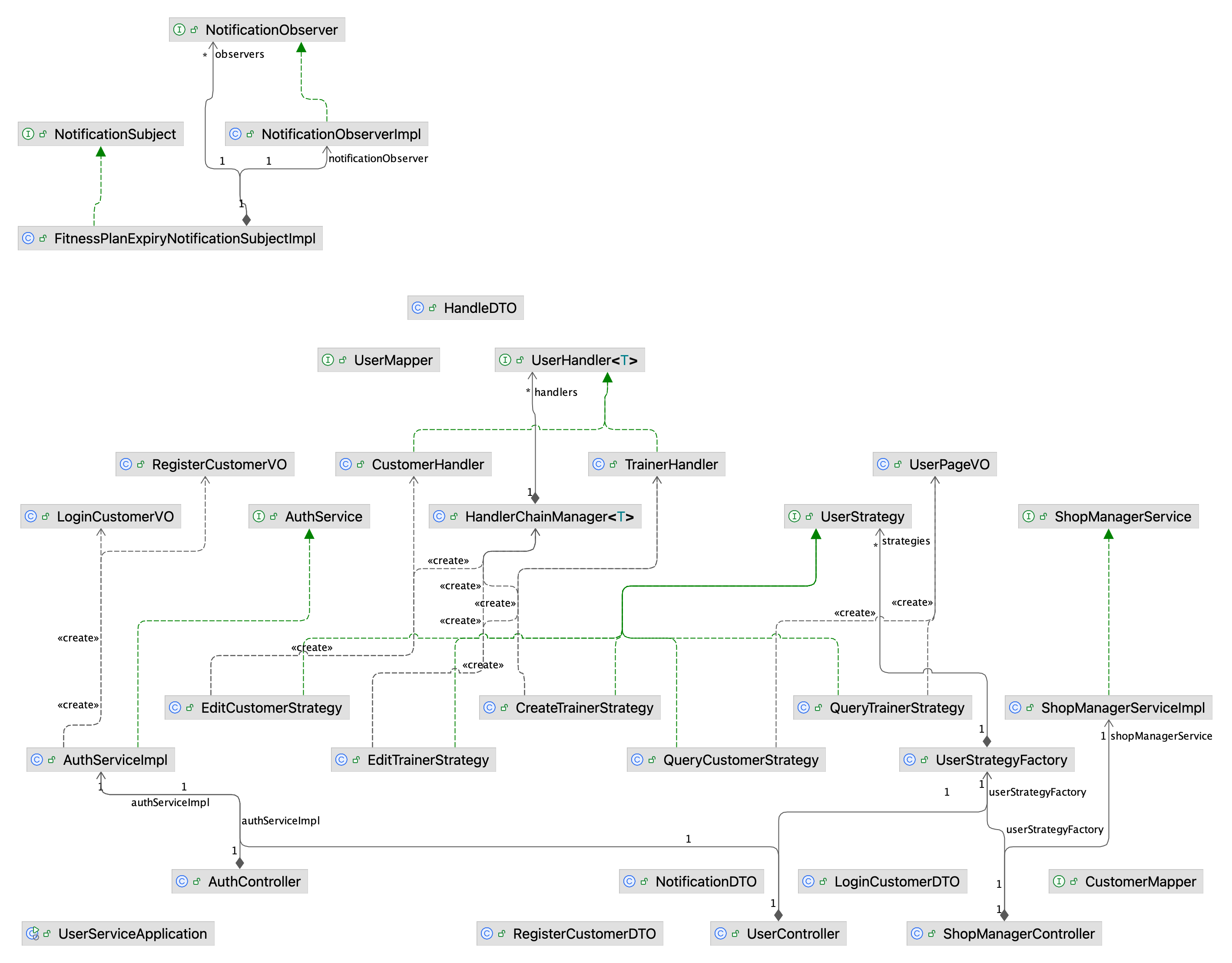
Selenium. (2023). Selenium Automated Testing. Retrieved from [https://www.selenium.dev/](https://www.selenium.dev/" \t "/Users/youranxu/Documents\\x/_new)

**Standards and Regulations**

International Organization for Standardization. (2023). ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

# Appendix (Not count into the limit of 8 pages)





*Sprint backlogs*

*Member contributions form*