FOOD ORDERING AND EVALUATION APPLICATIONS

Li Xiang Lin，Kok Ven Jyn

Faculty of Information Science & Technology

Universiti Kebangsaan Malaysia

43600 Bangi, Selangor

Abstract

Projek ini melibatkan penciptaan dan pelaksanaan sistem pesanan makanan dan ulasan yang kompleks yang merangkumi kedua-dua dimensi fungsional pesanan makanan dan interaksi sosial. Sistem ini menawarkan kemudahan kepada pengguna untuk melihat menu restoran, memesan makanan, membuat pembayaran, dan memberi ulasan dengan cara yang berkemungkinan meningkatkan pengalaman makan. Selain daripada fungsi utama ini, sistem ini juga termasuk fungsi sosial di mana pengguna dapat berkongsi pengalaman makan mereka, membaca ulasan orang lain, dan terlibat dalam perbincangan dua hala yang interaktif, dengan itu membina kepercayaan dan meningkatkan pembuatan keputusan mengenai tempat dan jenis makanan yang hendak dimakan.

Semua fungsi yang paling penting telah diuji secara menyeluruh dalam sistem. Setiap modul juga diuji untuk memastikan bahawa sistem berfungsi seperti yang diperlukan. Walau bagaimanapun, terdapat isu prestasi yang didokumenkan untuk modul pembayaran, dengan kelembapan yang didokumenkan bagi kebanyakan pengguna. Isu prestasi ini akan dipertingkatkan dalam iterasi masa depan untuk membolehkan pemprosesan yang lebih pantas walaupun semasa beban penuh. Proses pentadbiran pengendalian data pengguna, sejarah pesanan, dan maklum balas juga diuruskan oleh modul pengurusan backend.

Kemajuan masa depan dirancang, contohnya, meningkatkan modul pembayaran untuk pengendalian transaksi dengan lebih banyak keserentakan, meningkatkan cadangan peribadi dengan teknik pembelajaran mesin yang lebih maju, dan menyederhanakan antara muka pengguna untuk aksesibiliti dan kebolehgunaan yang lebih mudah. Selain itu, aspek keselamatan akan diperkukuhkan, terutamanya dalam mempertahankan maklumat pengguna dan keperluan privasi keseluruhan sistem.

Abstract

The project involves the creation and implementation of a complex food ordering and reviewing system that includes both the functional dimensions of food ordering and social interaction. The system offers the convenience to users of viewing restaurant menus, ordering food, paying, and reviewing in a manner most likely to enhance the dining experience. In addition to these core functionalities, the system includes social functionality where users are able to share their eating experience, read others' reviews, and engage in interactive two-way discussions, thereby building trust and improving decision-making on where and what to eat.

All the most important functionalities have been thoroughly tested in the system. Each of the modules was also tested to guarantee that the system performs as required. There were issues of performance, however, documented for the payment module, with slowdowns being documented for most of the users. The performance issues are to be enhanced in future iterations to enable faster processing even during full load. Administrative processes of user data handling, order history, and feedback are also managed by the backend management module.

Future progress is planned, e.g., enhancing the payment module for additional concurrency transaction handling, enhancing personal recommendation with advanced machine learning techniques, and simplifying the user interface for easier accessibility and usability. In addition, security aspects will be fortified, particularly in defending user information and the overall privacy needs of the system.

1.1 INTRODUCTION

Project Introduction: This project aims to develop an online food ordering and review mobile application with social functions, which helps users not only order quickly when selecting a restaurant, but also get the dining experience and advice of others through social interaction.

Background and why it matters: With the expansion of the food delivery market, users increasingly rely on others' reviews to make decisions. Traditional food ordering applications are limited to the display of menus and prices, which lack interaction and social functions, and cannot meet users' needs for friend recommendation and interaction. Through the application in order to join the social function, can enhance the user's trust and share, improving the user experience, user not only can quickly order, also can see others evaluation, thus make more informed.

1.2 CURRENT RESEARCH AND RELATED TECHNOLOGIES

Review of past research: The research field of food ordering and evaluation applications originated in the early days of e-commerce systems. These systems mainly focused on providing an online platform where users could browse menus and place orders. With the popularity of the Internet and the rapid development of mobile technology, research has gradually turned to the optimization of mobile applications, aiming to improve the user experience and convenience. Past studies have emphasized user experience design, integration of payment systems, and efficiency of order management. At the same time, the research on recommender systems has also laid a foundation for the development of food ordering platform.

Analysis of existing systems and technologies: Currently, there are many mature food ordering applications in the market, such as Uber Eats, DoorDash, and Ele. me. These systems often integrate restaurant search, order management, payment systems, and user ratings. These apps commonly use mobile development frameworks such as React Native and Flutter to support cross-platform development needs. In terms of server-side technologies, frameworks such as Spring Boot and Django are widely used for efficient back-end processing and database management. Using collaborative filtering recommendation system, many existing platform and content filtering algorithm, combined with the historical data and user preferences, providing personalized recommendation service.

Current trends and development: at present, food order and evaluate the application trend of the development of mainly concentrated in two aspects. The first is the further optimization of intelligent recommendation technology. Along with the progress of the artificial intelligence and machine learning, the application of deep learning models are used more and more, such as neural network, the real time and to improve the accuracy of recommendation. Secondly, social interaction functions are becoming a new development direction. Users not only want to be able to order food, but also want to share their dining experience and interact with other users through the platform. Therefore, the integration of food community and instant messaging functions has become particularly important, which improves user stickability and engagement. At the same time, the rising trend of healthy eating has also prompted apps to integrate health data to provide users with nutrition information and personalized health advice. Through the combination of these trends and technology, the food order and evaluate applications are moving in the direction of more intelligent and social development.

1.3 METHODS FROM PAST STUDIES

Research methods: In the research of food ordering and rating applications, traditional research methods mainly include user behavior analysis, collaborative filtering, content filtering, and hybrid recommendation systems. User behavior analysis predicts users' interests and needs by collecting and analyzing users' click, browse and purchase records. Collaborative filtering methods make recommendations according to the historical behavior of similar users, which can be divided into two types: user-based and item-based. Content filtering methods match similar content according to users' past preferences or attributes, and recommend items that are consistent with users' interests. Hybrid recommender systems combine collaborative filtering and content filtering to improve the diversity and accuracy of recommendations.

Effectiveness and drawbacks of the methods: These methods improve the relevance of recommendations and user satisfaction to some extent. Collaborative filtering can effectively use the data of user groups to predict the preferences of target users through the choices of similar users. However, this method is vulnerable to the cold start problem, that is, when new users or new items lack sufficient historical data, the recommendation effect will be significantly reduced. Content filtering through the analysis of the characteristics of the content, can provide the basic recommended in cold start phase, but it is in use for a long time may produce "fitting", lead to the lack of novelty and diversity recommendation results. Although hybrid recommender systems can integrate the advantages of the two methods, they are more complex and require more stringent computing resources.

1.4 COMPARISON AND CRITICISM OF PAST RESEARCHES

Technology comparison: In past studies, different recommender system technologies have shown their respective advantages and limitations in food ordering and evaluation applications. Collaborative filtering technology provides effective personalized recommendation by analyzing the similarity between users. However, its sensitivity to data sparsity and cold start problems significantly reduce the accuracy and coverage of recommendations. In contrast, content filtering technology can provide basic recommendation services in the early stage by analyzing the characteristics of items and users, but it is easy to produce information cocoon effect in the long-term use, which limits users' exposure to new content.

Critical analysis: Despite the importance of collaborative filtering and content filtering techniques in traditional recommender systems, their limitations when dealing with large-scale user and item data are laid bare. Collaborative filtering relies too much on the similarity between users and cannot cope with the recommendation needs of new users and new items. In addition, content filtering is prone to deviation in feature selection and lacks dynamic adaptability. Although hybrid recommender systems improve the coverage and accuracy of recommendations, their complexity increases the cost of system development and maintenance, and may not be suitable for all application scenarios.

1.5 OBJETIVES

When comparing food ordering and evaluation applications with other similar software, one can start from several aspects such as functionality, user experience, technical architecture, and unique selling points.

Table 1.1 Comparison

Comparison Aspect	**Meituan Waimai**	**Taobao (Food Category)**	**Ele.me**	**This Software**

Focus	Restaurant connections and food delivery	Comprehensive e-commerce with food category	Instant food delivery	Personalized food shopping experience
Shopping Cart	Limited deep integration	Generalized cart for all product types	Requires separate orders for different merchants	Allows multiple merchants' products in one cart
Evaluation System	General food reviews	Suitable for bulk commodity trading	Basic restaurant reviews	Fine-grained food-specific evaluation
Bulk Purchasing	Primarily designed for individual meal orders	Suitable for large-scale product purchases	Limited bulk order capabilities	Optimized for bulk food purchasing
User Experience	Focus on instant food delivery	Broad shopping experience, not food-specific	Streamlined for quick restaurant orders	Complete food shopping and ordering experience
Key Advantage	Extensive restaurant network	Massive product selection	Fast delivery services	Integrated, flexible, and optimized food shopping experience

Comparison Aspect Meituan Waimai Taobao (Food Category) Ele.me This Software

Focus Restaurant connections and food delivery Comprehensive e-commerce with food category Instant food delivery Personalized food shopping experience

Shopping Cart Limited deep integration Generalized cart for all product types Requires separate orders for different merchants Allows multiple merchants' products in one cart

Evaluation System General food reviews Suitable for bulk commodity trading Basic restaurant reviews Fine-grained food-specific evaluation

Bulk Purchasing Primarily designed for individual meal orders Suitable for large-scale product purchases Limited bulk order capabilities Optimized for bulk food purchasing

User Experience Focus on instant food delivery Broad shopping experience, not food-

specific   Streamlined for quick restaurant orders   Complete food shopping and ordering experience

Key Advantage   Extensive restaurant network   Massive product selection   Fast delivery services   Integrated, flexible, and optimized food shopping experience

Compared to Meituan Waimai, this software focuses more on personalized shopping experience rather than just delivery. The core of Meituan lies in connecting restaurants and users, providing a wide range of dining options, but there is relatively little deep integration of functions such as shopping carts and event management. However, Meituan's discounts are mostly focused on platform subsidies and merchant discounts, lacking flexible customized promotional activities. In addition, the Shopping Car design of this software allows users to purchase multiple types of food on the same platform, not limited to instant delivery of meals.

Compared to the food category on Taobao, this software focuses more on the vertical field of food ordering. Although Taobao's food category has a massive number of products, as a comprehensive e-commerce platform, its shopping cart, order management, and promotion strategies need to adapt to all categories of products, resulting in a lack of targeted purchasing processes for food products. In addition, Taobao's evaluation system is more suitable for bulk commodity trading, while this software's evaluation system is more in line with the scenario of food shopping, supporting finer grained evaluation and feedback.

Compared to Ele.me, the advantage of this software is a more complete shopping experience. Ele.me's strength lies in its instant delivery service, but if users want to purchase food in bulk or combine products from different merchants for shopping, Ele.me's shopping cart and promotion mechanism are relatively limited. The Shopping Car provided by this software allows users to add products from different merchants to the cart at once, forming a complete order, while Ele.me's model usually requires users to place separate orders.

Overall, the advantages of this software in the field of food ordering are mainly reflected in promotional flexibility, shopping cart optimization, and targeted food evaluation. Compared with Meituan, Taobao, and Ele.me, it not only has good food management functions, but also provides a more diverse shopping experience, allowing users to integrate food from different merchants in one purchase.

1.6 SCOPE

1.6.1 User Functional Scope

1.6.1.1 Browse and view categorized product listings with basic details (name, price, image).

1.6.1.2 Add products to the shopping cart and manage item quantity.

1.6.1.3 Proceed to checkout with a summary of selected items.

1.6.1.4 Complete orders via a confirmation interface.

1.6.1.5 View previous orders if logged in (optional user login).

1.6.2 Admin Functional Scope

1.6.2.1 Add, update, and delete product entries with name, price, and image.

1.6.2.2 Monitor stock levels and update availability.

1.6.2.3 Access transaction records for tracking completed orders.

1.6.2.4 Generate sales reports for internal review and analysis.

1.6.2.5 Maintain overall system content via a simple admin interface.

1.7 RESEARCH GAP

Identify research gaps: Although a great deal of research has been conducted in the area of food ordering and evaluation applications, there are still some key research gaps. Although many existing systems have made breakthroughs in recommendation and search functions, they still fall short in integrating multiple functions to enhance the user experience. Most of the current recommender systems mainly rely on a single technical method, such as collaborative filtering or content filtering, which fails to make full use of users' multi-dimensional information such as social data and geographical location. In addition, these systems have deficiencies in the diversity and accuracy of personalized recommendation, especially in the problem of data sparsity and cold start.

In addition, the research of this project provides new ideas for solving the shortcomings of existing systems, especially in data integration and user experience optimization. The results of this project will provide an important reference for the development of similar applications in the future, and promote the technological progress in the field of food ordering and evaluation.

1.8 INNOVATIVE SOLUTIONS DERIVED FROM PAST RESEARCH

Innovative approaches: Past research has proposed many innovative approaches in the area of food ordering and evaluation, especially in user interaction. By combining collaborative filtering, content filtering, and location-based recommendation techniques, several studies have achieved more accurate and diverse recommendation systems. These methods not only improve the accuracy of recommendation but also solve the problems of cold start and data sparsity to some extent. In addition, the introduction of social recommendation provides users with a more personalized experience, making it more in line with users' actual needs and preferences.

Suggestions for improvement: Although the existing methods have achieved certain results, there is still room for further improvement. Another improvement proposal is to strengthen the privacy protection of user data to ensure that users' privacy is not violated while providing personalized services. In addition, integrating multi-source data, such as users' historical purchase records， will help build a more comprehensive and accurate user profile, thus improving the overall effect of the recommendation system.

Improving user Interface design to improve user interaction: In terms of user interface design, past research has shown that simple and intuitive interfaces can significantly improve user interaction experience. By optimizing the navigation structure, adding visual data display and providing personalized interface Settings options, users can more easily find the information they need and enjoy the convenience brought by the application.

1.9 CONCLUSION

Summary of Key Findings: This paper reviews the development of online food ordering and evaluation applications, and analyzes the application of related technologies and methodologies. It is found that the combination of geolocation features can significantly improve the user's food exploration experience. In addition, apps that integrate social interaction functions can not only meet users' basic needs, but also enhance interaction between users. The study also shows that personalized recommender systems play an important role in improving user satisfaction and app usage frequency.

Research reason: To develop a food ordering and evaluation application that integrates multiple functions, aiming to provide users with a more convenient and comprehensive food exploration platform. This kind of multi-functional mobile application can effectively fill the shortage of existing single-function applications in the market, provide users with one-stop service, and increase user loyalty and frequency of use.

The impact of social media on food ordering behavior: With the widespread use of social media, users' interaction and sharing behavior on social platforms has a significant impact on their food ordering decisions. Users are more likely to trust recommendations from friends and online communities than traditional advertisements. This trend presents new opportunities for food ordering apps. By tightly integrating social features with the ordering system, users can more easily obtain trusted dining suggestions and reviews to make more satisfying choices.

2.0 LITERATURE REVIEW

2.1. Functional requirements

The system should meet the needs of different user groups. Visitors can browse the platform content, post comments on this basis, users can complete account registration, bind mobile phone and modify avatar, post content (including pictures or text up to 150 words), and carry out a variety of interactive operations, such as liking, commenting.

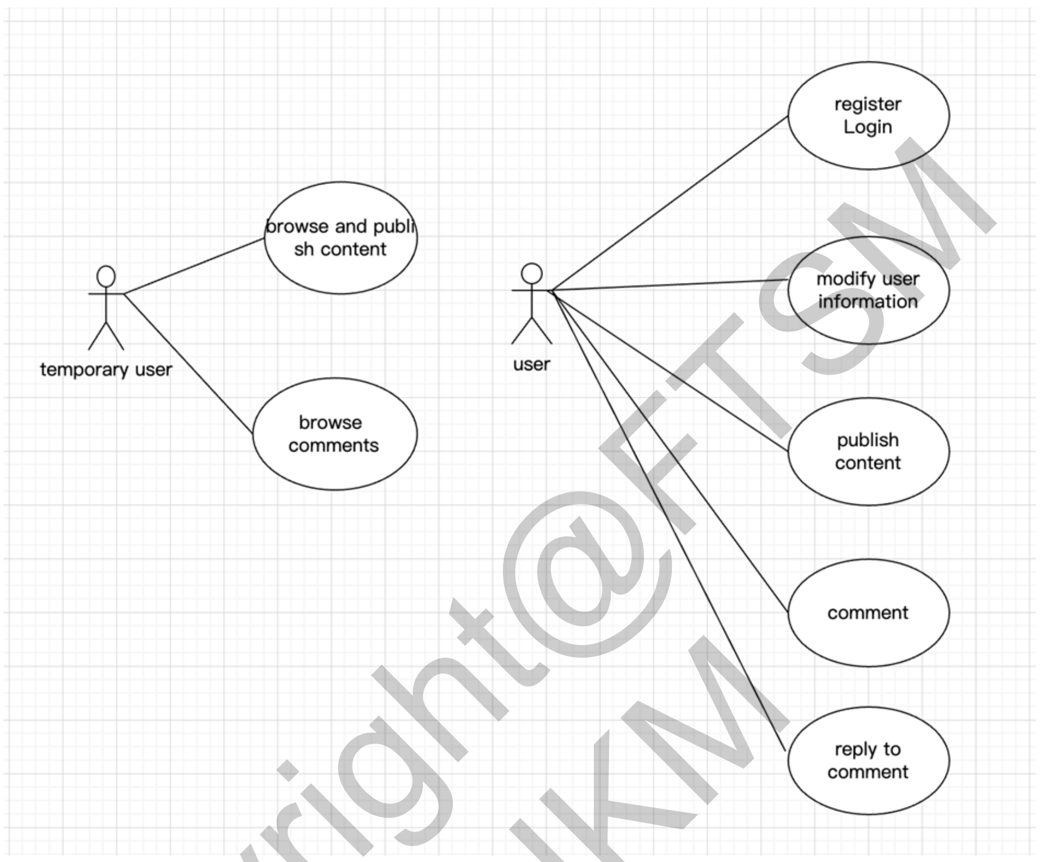


Figure 2.1 Use Case Diagram

Background management function is an important part of the system, the administrator can perform the following tasks: manage user information, review user registration and content publishing, perform content classification management, review user comments, and so on. These functions ensure the compliance of the platform content and the health of the user environment.

2.2 OVERALL SYSTEM ARCHITECTURE

The system design of online food ordering and review application adopts the design pattern of hierarchical architecture to ensure the modularity, maintainability and expansibility of the system. The architecture is mainly divided into front-end display layer, application logic layer, data processing and data storage layer. The specific functions and implementation methods of each layer are as follows.

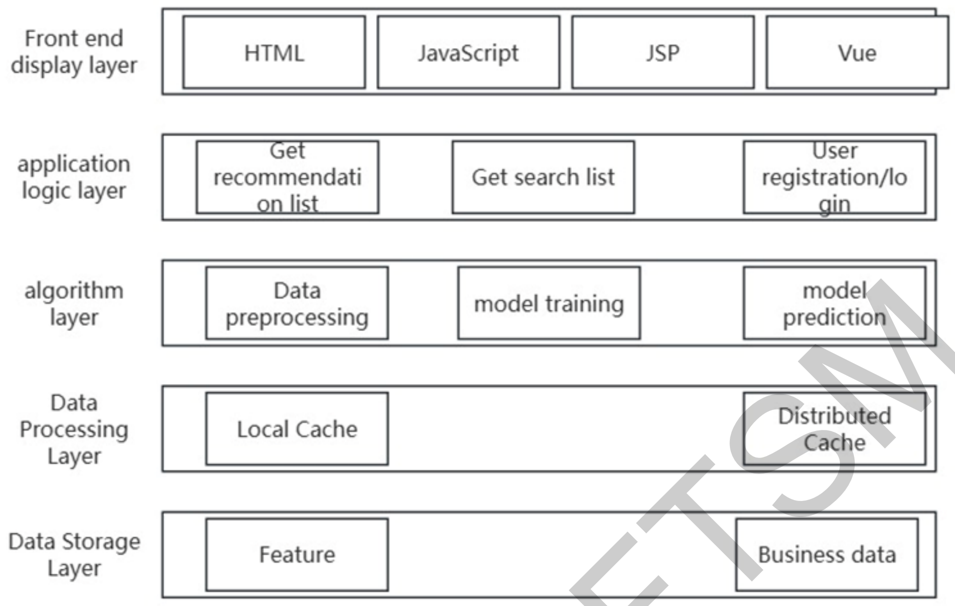


Figure 2.2 System architecture

As shown in Figure 2.2 , the presentation layer is responsible for interacting with the user and providing a friendly user interface and experience. Figure 3.2 System architecture This layer uses modern Web technology frameworks and tools such as React, Vue, or Angular to achieve responsive design and efficient user interfaces. Users access the application through a browser or mobile device, and are able to perform member login, registration, content browsing, comment interaction and other operations. The front layer is also responsible for to show the user list, notifications, and to provide graphics processing capabilities, to ensure that content in best visual effects.

The application logic layer is the core of the system, which is responsible for handling user requests, implementing business logic and coordinating the interaction between the front-end and the data layer. This layer realizes user registration, login and information management functions, including binding mobile phone, modifying user profile picture. Business logic layer also manage content, reviews, thumb up, and other functions. Using server-side programming framework such as Node. Js, Spring Boot or Django development, to ensure the processing of efficient and flexible business development.

The data processing layer acts as a bridge between business logic and data storage. It is responsible for receiving requests from the front-end and interacting with the data storage layer.

The data storage layer includes relational databases and non-relational database, which is used to store user data, published content, comments and user activity records. Relational databases such as MySQL or PostgreSQL is used to store structured data, such as user account information, reviews and integral record. Non-relational databases such as MongoDB are used to store dynamic content and large volumes of published content. Data storage layer also includes support for data backup and recovery, to ensure the data security and reliability of the system.

In addition, the system also includes a background management function to review and manage content and users. The background management system can realize the functions of release review, new release, release category management and user management. In general, through clear hierarchical design, reasonable technology selection and efficient processing mechanism, the system architecture ensures that the online food ordering and review application performs well in terms of user experience, performance, scalability and security.

2.3 FUNCTIONAL MODULE DESIGN

The online order food and reviews the application of the system design consists of several functional modules, to ensure that the user can be efficient and diversified operation, meet the needs of users and visitors. These function modules cover all the key functions of the application, from user interaction to back-office management and data processing.

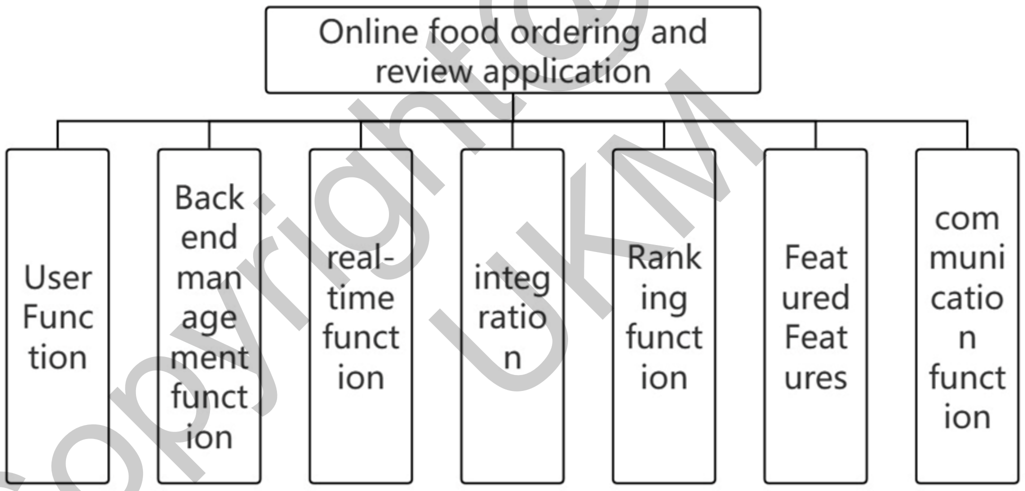


Figure 2.3 Feature architecture

In the user management module, it includes functions such as login, registration, mobile phone binding, and avatar modification.

Content distribution and interaction module allows users to publish pictures, text (up to 150 words). Users can be on the platform for content, enrich the interaction of the user. In order to enhance the user experience. Content and comments posted by users can to the list.

The background management module provides administrators with the function of reviewing and managing content and users. The administrator can review the content published by users, manage the published categories and approve the published. The background user information review function ensures the compliance of user registration and behavior. The comment review function helps ensure that the content on the platform is healthy and civilized, and prevents the spread of malicious comments and illegal content.

In a word, the functional module design of the system comprehensively covers all aspects from user registration, content publishing to background management and real-time interaction. Modular design is not only to ensure the system run efficiently, also facilitate the function of the subsequent development and maintenance, to ensure that applications can meet the needs of users is growing.

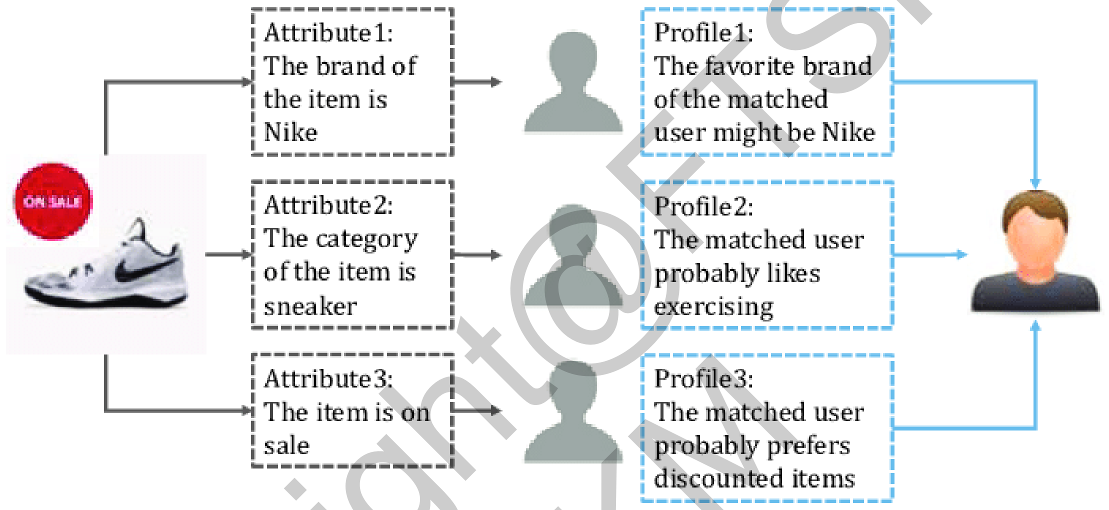

ON SALE
Attribute1: The brand of the item is Nike
Attribute2: The category of the item is sneaker
Attribute3: The item is on sale
Profile1: The favorite brand of the matched user might be Nike
Profile2: The matched user probably likes exercising
Profile3: The matched user probably prefers discounted items

Figure 2.4 User profile relationship structure diagram

The user profile building blocks are the foundation of our system. The module not only the basic attribute information of the product, such as price, category, etc., but also includes the dynamic data such as user evaluation .

2.4 DATABASE DESIGN

In today's information age, effective management and application of data has become the key to improve the quality of service. Especially in the commodity recommendation system, the quality of the database design directly affects the performance of the system and user experience. This study aims to design a commodity recommendation system based on Java environment and collaborative filtering algorithm, in which the design of database plays a crucial role.

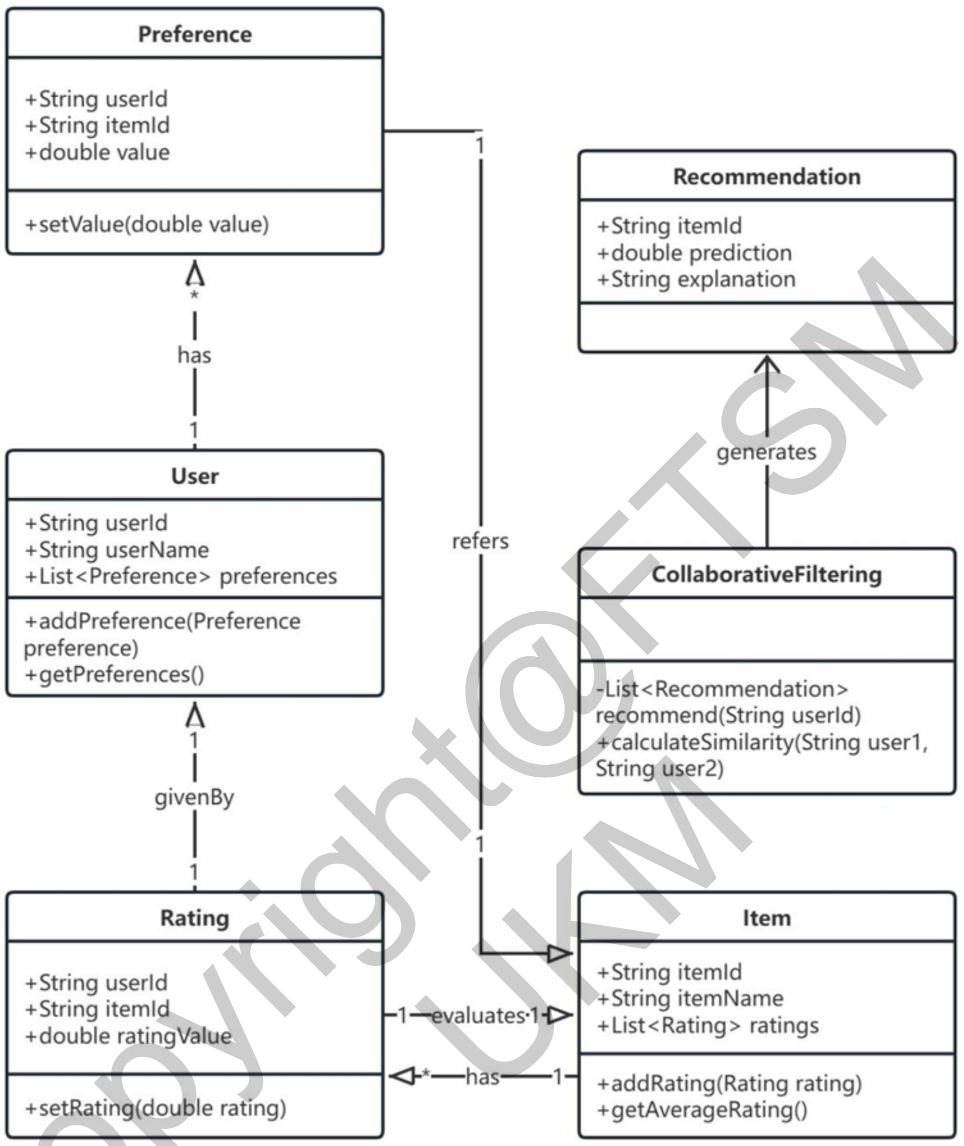


Figure 2.5 Class diagram

As shown in Figure 2.6, given the large amount of user data and product information that the system needs to process, the database design must ensure efficient data access and good scalability. In this system, the relational database management system (RDBMS) is used to ensure the consistency and integrity of data by utilizing its mature technology stack and wide community support. The database architecture is organized around three core modules: user information management, commodity information management, and user behaviour record.

Column Name	Data Type	Length	Primary Key	Auto Increment	Allow Null	Default Value	Description

id	bigint		√	√			User ID
login_name	varchar	-255					Login name
admin_name	varchar	-255					Username
password	varchar	-255					Password
mobile	char	-11					Mobile phone
age	tinyint						Age
sex	char	-1					Gender (Male)
address	varchar	-255					User address
status	tinyint					0	User status: 0 = Active, 1 = Frozen
gmt_create	datetime						Time of user creation
gmt_modified	datetime						Time of last modification
is_deleted	tinyint					0	Logical delete flag

Table 2.1 shopping sys admin

As shown in Table 2.1, for the user information management module, a table structure containing multi-dimensional information is designed. Table 1This table not only includes basic personal information such as username, password and contact information, but also covers the user's browsing history, purchase record and evaluation feedback, so as to facilitate in-depth mining of user preferences and achieve accurate recommendation. To improve the efficiency of data query, the index optimization strategy is used to index the frequently queried fields such as user ID.

Column Name	**Data Type**	**Length**	**Primary Key**	**Auto Increment**	**Allow Null**	**Default Value**	**Description**

product_id	bigint		√	√		Product ID
product_name	varchar	-40				Product Name
category1_id	bigint					Primary Category ID of the product
category2_id	bigint					Secondary Category ID of the product
purchase_price	decimal	(30, 0)				Purchase Price
sale_price	decimal	(30, 0)				Sale Price
inventory	int					Inventory
hits	bigint				0	Click count for the product
img_src	varchar	-255				Image URL
description	varchar	-255				Product Description
gmt_create	datetime					Time when the record was created
gmt_modified	datetime					Date when the record was last modified
is_show	tinyint				1	Whether the product is displayed: 1 = Display, 0 = Hide
is_deleted	tinyint				0	Logical delete flag

Table 2.2 Shopping productThe product information management module, shown in Table 2.2, focuses on structured storage of product data. Each product entry contains a detailed product description, category information, inventory status, and price fields. The design of this module

pays attention to the consistency and update efficiency of data. The reference integrity of data is ensured by foreign key constraints, and the relevant data is updated by triggers to ensure the real-time and accuracy of commodity information.

Column Name	Data Type	Length	Primary Key	Auto Increment	Allow Null	Default Value	Description
user_id	bigint		√				User ID
category2_id	bigint		√				Secondary Category ID
hits	bigint						The number of clicks the user made in this secondary category

Table 2.2 Shopping cart

Column Name	Data Type	Length	Primary Key	Auto Increment	Allow Null	Default Value	Description
user_id	bigint		√				User ID
product_id	bigint		√				Product ID
product_name	varchar	-255					Product Name
gmt_create	datetime						Record creation time
gmt_modified	datetime						Record modification time
is_deleted	tinyint					0	Logical deletion indicator

Table 2.3 Shopping activeThe database design of the system takes full account of the efficiency, security and scalability of data processing, aiming to provide a stable and reliable data support platform, and lay a solid foundation for the commodity recommendation system based on Java environment and collaborative filtering algorithm. Through the carefully designed database architecture, massive data can be effectively managed, the response speed of the system and the quality of recommendation can be improved, and finally the high-quality user experience

can be achieved.

2.5 SYSTEM PROCESS DESIGN

When designing a commodity recommendation system based on collaborative filtering algorithm, the process design of the system is a crucial part. This process involves the whole process of the system from data collection, processing to final generation of recommendation results. The design of the system process should not only ensure efficiency and accuracy, but also consider the user experience and the scalability of the system.

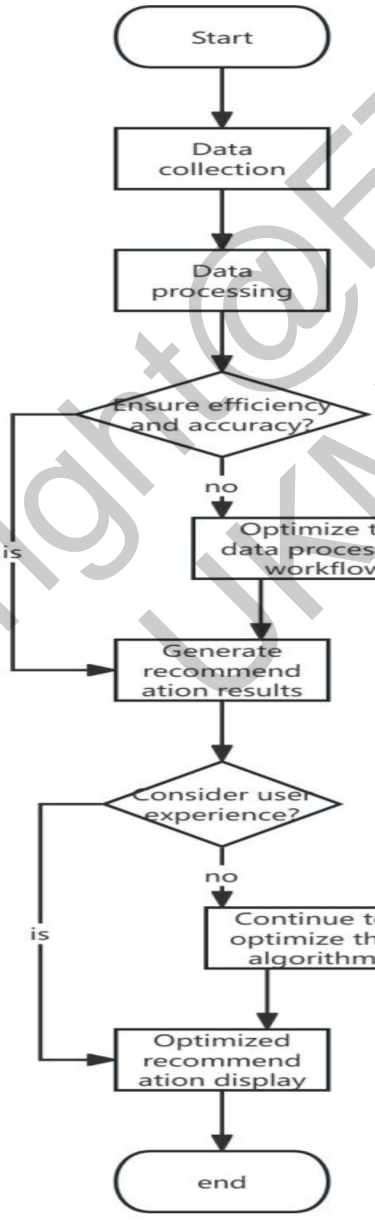

Start
Data collection
Data processing
Ensure efficiency and accuracy?
no
is
Optimize the data processing workflow
Generate recommend ation results
Consider user experience?
no
is
Continue to optimize the algorithm
Optimized recommend ation display
end

Figure 2.6 Flow chart

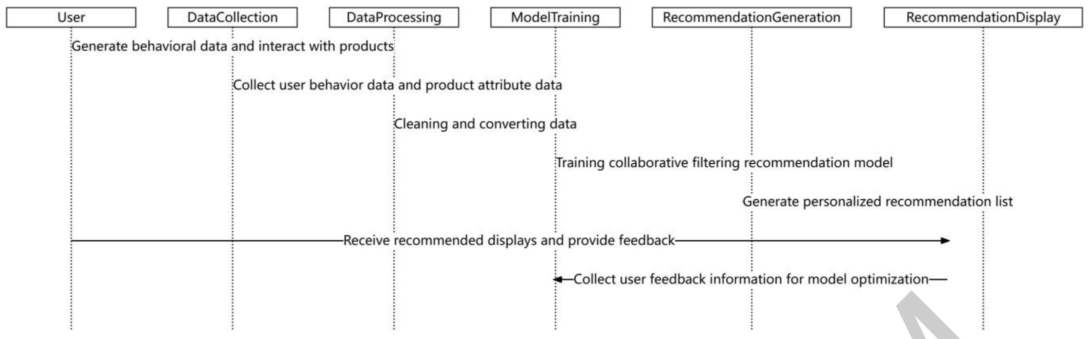

User
DataCollection
DataProcessing
ModelTraining
RecommendationGeneration
RecommendationDisplay
Generate behavioral data and interact with products
Collect user behavior data and product attribute data
Cleaning and converting data
Training collaborative filtering recommendation model
Generate personalized recommendation list
Receive recommended displays and provide feedback
Collect user feedback information for model optimization

Figure 2.7 System timing

As shown in Figure 2.7, in the data collection phase of the system, the main focus is on the behavior data of users and the attribute data of items. The user behavior data includes but is not limited to the user's purchase history, browsing history, and rating information, while the product attribute data includes the description, category, and price of the product. The comprehensive collection of these data provides rich materials for subsequent data analysis and model training.

3.0 METHODOLOGY

3.1 IMPLEMENTATION OF FOOD SEARCH MODULE

The development of the food search module aims to provide users with a fast and accurate search experience, so as to meet their needs for timeliness and convenience when exploring food. In the system design, the backend is built with the SpringBoot framework, and the efficient data serialization mechanism ensures that users can quickly respond to their search requests. Vue technology is used in the front-end to create a simple and intuitive interactive interface, so that users can easily find their favorite food and restaurants.

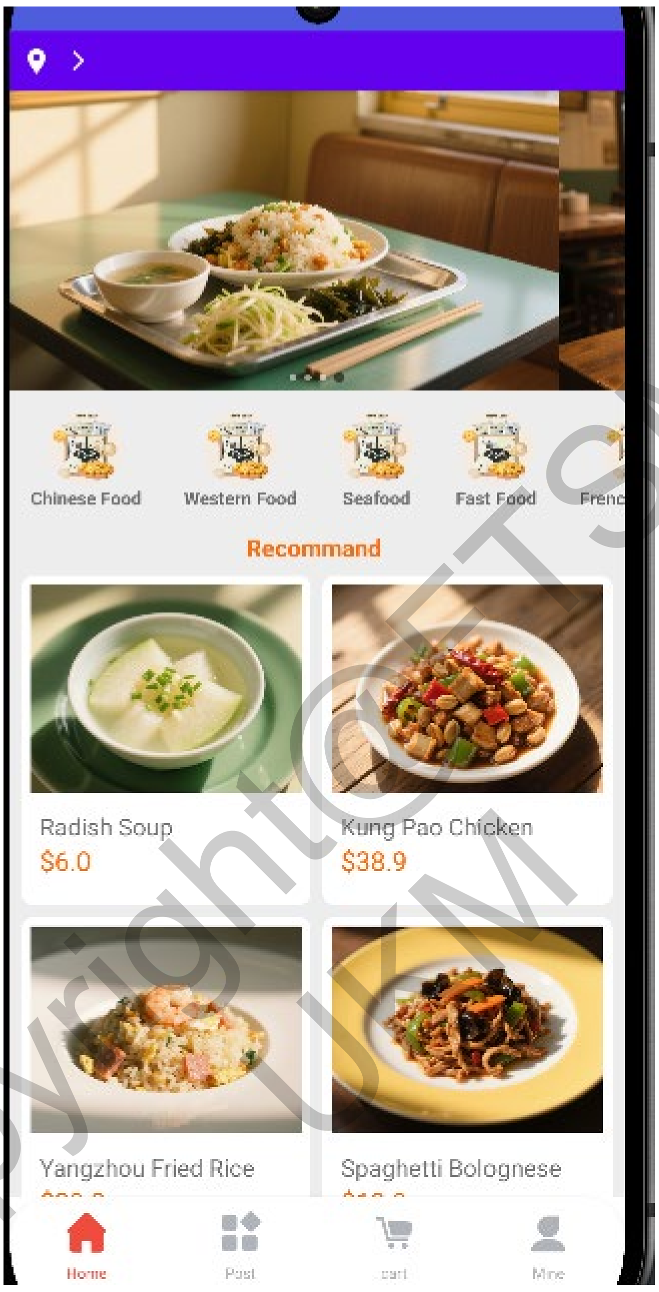


Figure 3.1 Interface diagram

The interface design thoroughly considers user needs, aiming to create an intuitive and seamless experience for ordering dishes. Users who wish to view specific dishes can simply click on the category bar, enabling them to quickly locate their preferred food categories without spending excessive time searching for desired items.

There's a small icon shaped like a trash can. Tapping on it instantly clears the list of past searches. This is handy, especially if you're using a shared phone or just like keeping things tidy and private.

The screen's layout doesn't try to impress with fancy visuals—it sticks to what works. Everything is where you'd expect it to be, with plenty of space and no visual clutter. That makes it simple to use, even for someone opening the app for the first time.Prominent buttons and icons are not just decorative elements—they are strategically designed with color schemes that guide user attention, highlight interactive options, and add visual vibrancy to the interface without compromising focus.

3.2 IMPLEMENTATION OF FOOD STRATEGY MODULE

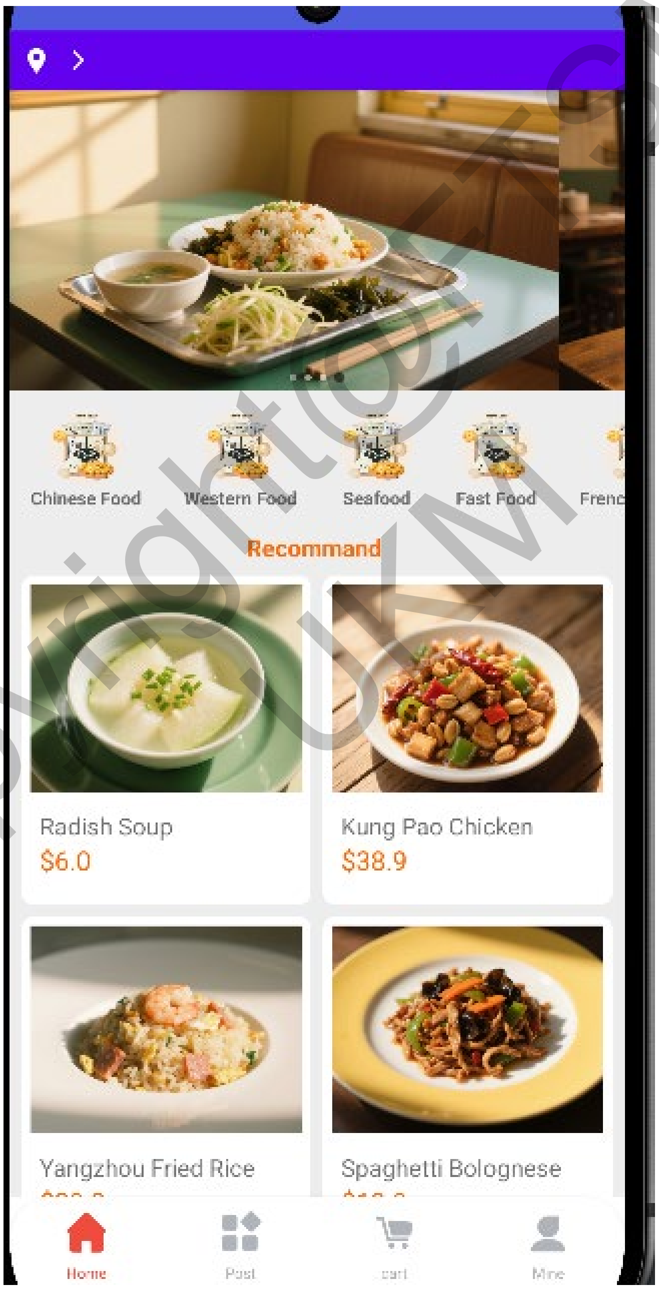


Figure 3.2 Interface diagram

The system incorporates a comment feature designed to improve user engagement. It enables users to provide feedback on dish content , thereby enhancing the richness and variety of information within the module. Furthermore, additional users can join the interaction through likes or comments, which significantly boosts community activity. This interactive framework transforms the food information platform into a collaborative environment where users can communicate and learn from one another.

In conclusion, the food module delivers extensive food-related services across various levels through the integration of diverse content and robust interactive features. It serves not only as a means for users to access quality meals but also as a platform that encourages healthier living by fostering user engagement. As an essential part of the application, this module effectively addresses user demands while simultaneously advancing the promotion of food culture and strengthening community connections.

3.3 FOOD RECOMMENDATION

The food module represents a core component of the application, . Its design not only improves user convenience but also facilitates the discovery of diverse culinary options, thereby more effectively aligning with users' taste preferences and enhancing the overall experience of exploring and ordering food.

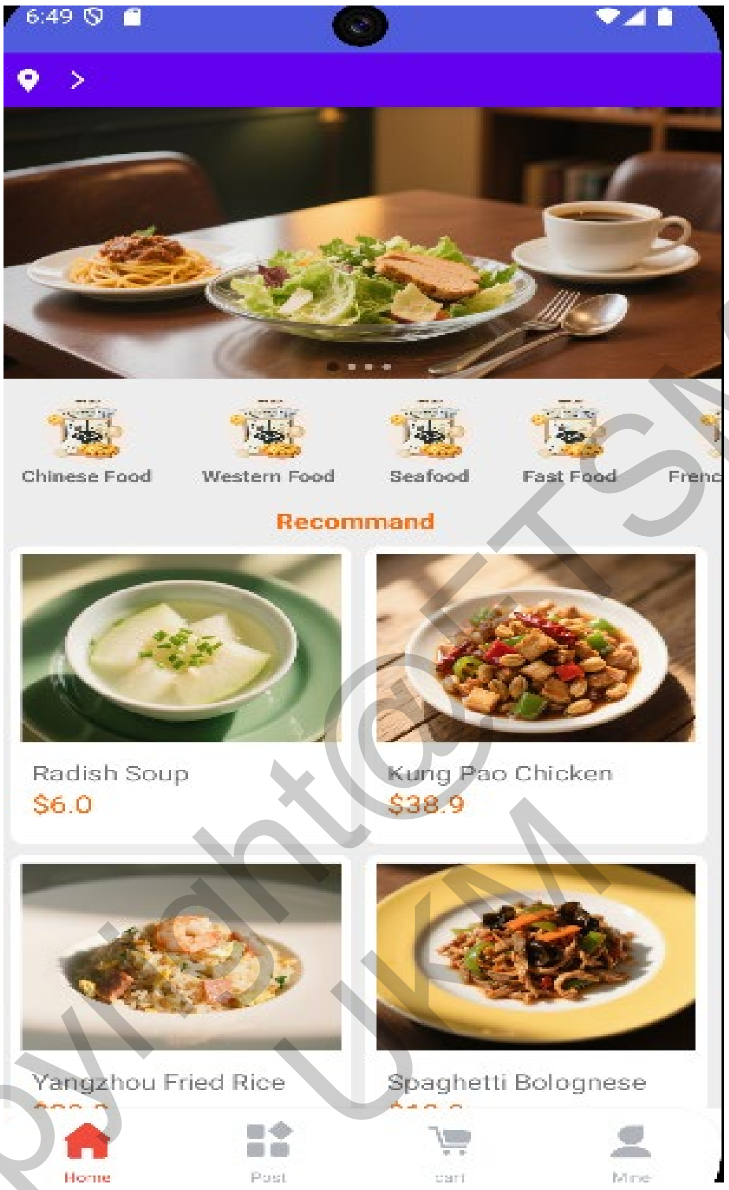


Figure 3.3 Interface diagram

The user interface presents a wide variety of food choices, featuring dish suggestions and categorized food classifications. This enables the system to deliver more customized content to users. In cases where users are uncertain about their selection, they can explore the recommended dishes section. Additionally, when making a choice, users have the option to browse by specific food types. Selecting a particular category allows for quick access to similar food items, significantly minimizing the effort and time typically associated with deciding on a meal.

In conclusion, the module offers personalized food recommendation services to users. This enhancement not only improves overall user satisfaction but also substantially increases the application's interactivity and user retention, enabling individuals to conveniently discover dining choices that align with their preferences and enjoy tailored culinary experiences.

3.4 IMPLEMENTATION OF FOOD COMMUNITY MODULE

The food community module is an important part of the application, which focuses on enhancing the interaction and social experience between users. By building an open platform, the module encourages users to share food experiences, publish restaurant reviews, and exchange cooking skills, so as to build a vivid community ecology. In this community, users can not only obtain diverse food information, but also establish connections with like-minded fans to further enrich their own food exploration experience.

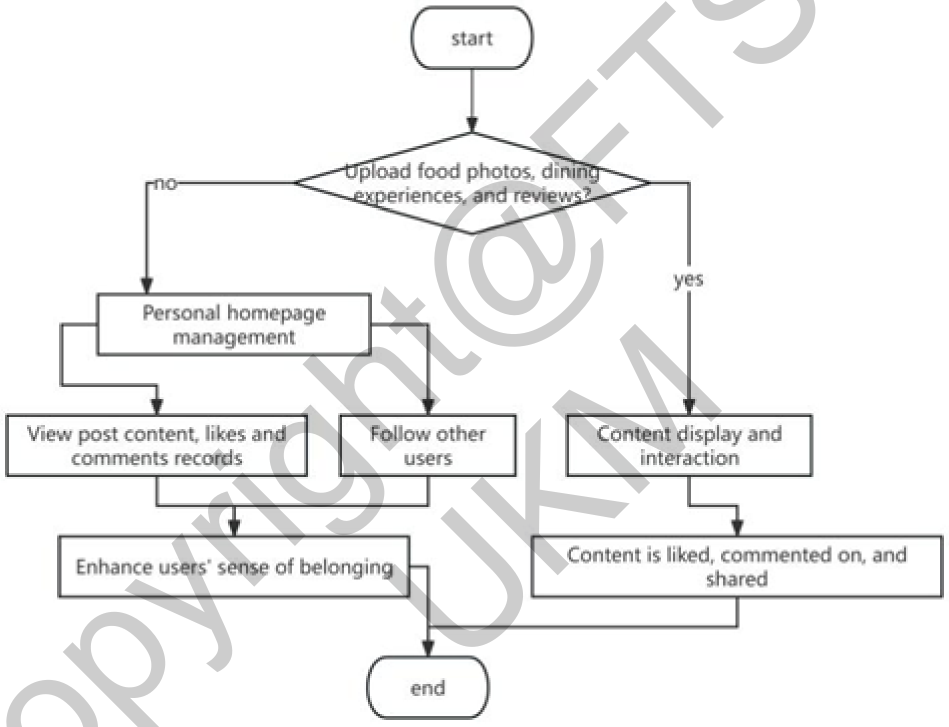

start
Upload food photos, dining experiences, and reviews?
no
yes
Personal homepage management
View post content, likes and comments records
Follow other users
Content display and interaction
Enhance users' sense of belonging
Content is liked, commented on, and shared
end

Figure 3.4 Flow chart

The core of the module lies in the creation and sharing of User-Generated Content (UGC). Users can upload food photos, write about the dining experience, and rate and evaluate the restaurant. These real user contents not only provide valuable reference information for others but also show various food cultures and personal styles to a certain extent. Other users can interact with the content creators through liking, commenting , thereby promoting the dissemination of content and the activity of the community. Excellent food photos and wonderful reviews are more helpful for content creators to accumulate attention and enhance their influence and sense of participation in the community.

In order to increase users' sense of belonging and convenience, the system designs a personal homepage function for each user. Users can view their posts, record their interaction history, and manage other users they follow through their personal homepage.

To maintain a healthy and harmonious community environment, the module incorporates a rigorous content review and management system. All user-submitted content undergoes platform moderation to prevent the dissemination of inappropriate or unlawful material, while also discouraging uncivilized behavior and fostering a welcoming community atmosphere. This mechanism ensures constructive interactions and upholds the overall quality of the community, thereby enhancing the value and satisfaction of user participation.

In summary, the food community module successfully builds a diversified food communication platform by integrating user-generated content, This module not only enhances the connection between users, but also provides rich creative inspiration and communication opportunities for users, which significantly improves the social value and user engagement of the app, and becomes an indispensable part of the app ecosystem.

4.0 RESULTS

4.1.1 Document Purpose

This document mainly records some specific processes of the entire food delivery ordering system during the development and implementation stage, including the implementation of key modules, encountered problems and solutions, etc

4.1.2 Document Scope

This project is a mobile endpoint catering application. The user end is developed using Java and runs on the Android platform. The merchant backend is implemented using Vue, and the backend is constructed by combining Spring Boot with MyBatis and MySQL. The entire system supports functions such as user registration and login, browsing merchants and goods, online ordering, payment, confirmation of receipt, and evaluation, basically covering a complete online food ordering process.

Users can browse various merchants through the homepage. After entering the merchants, they can view the details of the dishes and place orders. The order supports online payment. After the payment is completed, the order status can be tracked until the goods are confirmed upon receipt. After receiving the goods, users can evaluate the products. The system also features a social-like food recommendation function. Users can post their own food recommendation posts to interact and communicate with other users.

In the personal center, users can view the collection list, comment records, and the recommended posts they have published themselves. In addition, multiple delivery addresses can be maintained to facilitate quick selection when placing an order. The entire system has a clear architecture from the front end to the back end, with a wide range of functions, making it suitable for small catering merchants to go online and use.

4.1.3 Relevant Documents

1. JWT: Json Web Token
2. Project Proposal (T_4172)
3. Requirements Specification (D3)
4. Design Specification (D4)
5. User Manual (to be submitted later)
6. Test plan (to be provided in another document)

4.1.4 Terms and Abbreviations

1. JWT: Json Web Token
2. UI: User Interface
3. DB: Database
4. API: Application Programming Interface.

4.1.5 Development Process

This ordering system is mainly developed for the Android mobile platform, using Android Studio as the development environment and adopting a front-end and back-end separated architecture as a whole. The front end is the user-end App, written in Java, and the merchant back-end management system is implemented with Vue. The back-end service is constructed by Spring Boot and combined with MyBatis and MySQL to implement business logic and data management.

To enhance development efficiency and user experience, the project integrates multiple mainstream third-party libraries and services, covering functional modules such as network requests, UI display, map positioning, and image processing.

4.1.6 Technology Stack

a. Android client (food ordering App)

1. Language and Structure: Written in Java, the project enables ViewBinding to simplify view binding and adopts Material Design to enhance interface consistency and interaction experience.

2. Interface layout: Rely on ConstraintLayout to build a responsive interface and combine the Banner component to achieve the display of the home page carousel.

3. Network communication: Retrofit + Gson is used to achieve HTTP communication with the backend, ensuring the stability and efficiency of data interaction.

4. Image loading: Integrate Glide for asynchronous image loading and caching, and CircleImageView for displaying circular avatars.

5. Map positioning:Through the Autonavi Map SDK (2D map and positioning library), it supports positioning and other functions, providing support for address maintenance and merchant distance calculation.

b. Web Backend (Merchant Management)

Build a single-page application (SPA) with Vue, providing functions such as merchant information management, dish maintenance, and order viewing. The interface is simple and intuitive, and it supports responsive layout.

c. Back-end service Back-end service

The RESTful API is constructed using Spring Boot. MyBatis is responsible for database operations, and MySQL is used as the main data storage. The entire back-end system supports core businesses such as user registration and login, order processing, comment management, recommended content publication, and address management.

The system also implements a Token authentication mechanism to ensure the security of user operations.

4.2 SYSTEM COMPONENTS

The application contains multiple modules, such as home page merchant browsing, product details, shopping cart and payment, order tracking, comment section, as well as the Posting and display of recommended posts. The Personal center module supports viewing collections, comments, and post records, and provides address management functions. Combined with map positioning, it improves the efficiency of filling in.

The project ensures the smoothness, maintainability and user experience of the entire system through clear module division and stable technical selection.

a. JWT Login Verification (Backend)

The following code snippet uses a custom JWTUtil utility class on the server side to handle token generation and verification operations after user login, ensuring the security and uniqueness of the user session.

When the user successfully verifies through the login interface, the system will call the getToken(UserBean user) method. This method sets an expiration time of 30 days based on the current time and uses the HS256 algorithm combined with the preset key to sign the token. The generated JWT token contains the user's username, user ID and role information. The front end can complete subsequent authentication requests through this token.

In the interface where the user's identity needs to be verified, the system will call the verify(String token) method to determine whether the token is legal and valid. If an exception is thrown during the parsing process, such as token expiration or signature error, the verification fails.

Furthermore, JWTUtil also provides multiple static methods to extract user information from the token, including getUsername(String token) and getUserId(String token). It also supports directly reading the token field in the request header from HttpServletRequest to extract the user ID, which is convenient for use at the interface layer.

Through this JWt-based authentication method, the system achieves stateless session management, ensuring the security of the interface and reducing the storage pressure on the server.

```
public class JWTUtil {

    private static final long EXPIRE_TIME = 30L * 24 * 60 * 60 * 1000;

    private static final String SECRET_KEY = "weilong";

    public static String getToken(UserBean user) {
        try {
            Date date = new Date(System.currentTimeMillis() + EXPIRE_TIME);
            JwtBuilder jwtBuilder = Jwts.builder();
            String token = jwtBuilder
                    .setHeaderParam("typ", "JWT")
                    .setHeaderParam("alg", "HS256")
                    .claim("username", user.getUsername())
                    .claim("userId", user.getUid())
                    .claim("role", user.getRole())
                    .setExpiration(date)
                    .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                    .compact();
            return token;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public static boolean verify(String token) {
        try {
            if (token == null) {
                return false;
            }
            JwtParser jwtParser = Jwts.parser();
            Jws<Claims> claimsJws = jwtParser.setSigningKey(SECRET_KEY).parseClaimsJws(token);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    //   获得用户信息
    public static String getUsername(String token) {
        try {
            JwtParser jwtParser = Jwts.parser();
```

Figure 4.1 Code

```
 Jws<Claims> claimsJws = jwtParser.setSigningKey(SECRET_KEY).parseClaimsJws(token);
            Claims claims = claimsJws.getBody();
            String username = claims.get("username").toString();
            return username;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public static Integer getUserId(String token) {
        if (token == "") {
            return -1;
        }
        try {
            JwtParser jwtParser = Jwts.parser();
            Jws<Claims> claimsJws = jwtParser.setSigningKey(SECRET_KEY).parseClaimsJws(token);
            Claims claims = claimsJws.getBody();
            String userId = claims.get("userId").toString();
            return Integer.parseInt(userId);
        } catch (Exception e) {
            e.printStackTrace();
            return -1;
        }
    }

    public static Integer getUserId(HttpServletRequest request) {
        try {
            JwtParser jwtParser = Jwts.parser();
            Jws<Claims> claimsJws =
jwtParser.setSigningKey(SECRET_KEY).parseClaimsJws(request.getHeader("token"));
            Claims claims = claimsJws.getBody();
            String userId = claims.get("userId").toString();
            return Integer.parseInt(userId);
        } catch (Exception e) {
            e.printStackTrace();
            return -1;
        }
    }

}
```

Figure 4.2 Code

b. The Android order placement function has been realized

The client uniformly builds Retrofit instances by encapsulating the getRetrofit() method and adds interceptor functionality in the OkHttpClient.

Before each network request is initiated, the system automatically reads the user's login token from SharedPreferences and adds it to the request header (the field name is "token"). In this way, all requests sent through this Retrofit instance will carry tokens by default, eliminating the need to pass parameters separately for each interface, thereby enhancing the simplicity and maintainability of the code.

Furthermore, the Gson converter is configured in the code, and the date format yyyy-MM-dd HH:mm:ss is set so that Retrofit can handle the date field correctly when serializing and deserializing JSON data.

This design enables the Retrofit network layer to have a unified authentication mechanism and data parsing capability, which is a key link for secure communication between the client and the back end.

```
public static Retrofit getRetrofit() {
    if (retrofit == null) {
        OkHttpClient httpClient = new OkHttpClient.Builder()
                .addInterceptor(chain -> {
                    SharedPreferences loginInfo =
context.getSharedPreferences("loginInfo", MODE_PRIVATE);
                    Request request = chain.request().newBuilder()
                            .addHeader("token",
loginInfo.getString("token", ""))
                            .build();
                    return chain.proceed(request);
                }).build();

        Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd
HH:mm:ss")
                .serializeNulls()
                .create();
        retrofit = new Retrofit.Builder()
                .client(httpClient)
                .baseUrl(BaseURL)
                .addConverterFactory(GsonConverterFactory.create(gson
))
                .build();
    }
    return retrofit;
}
```

Figure 4.3 Code

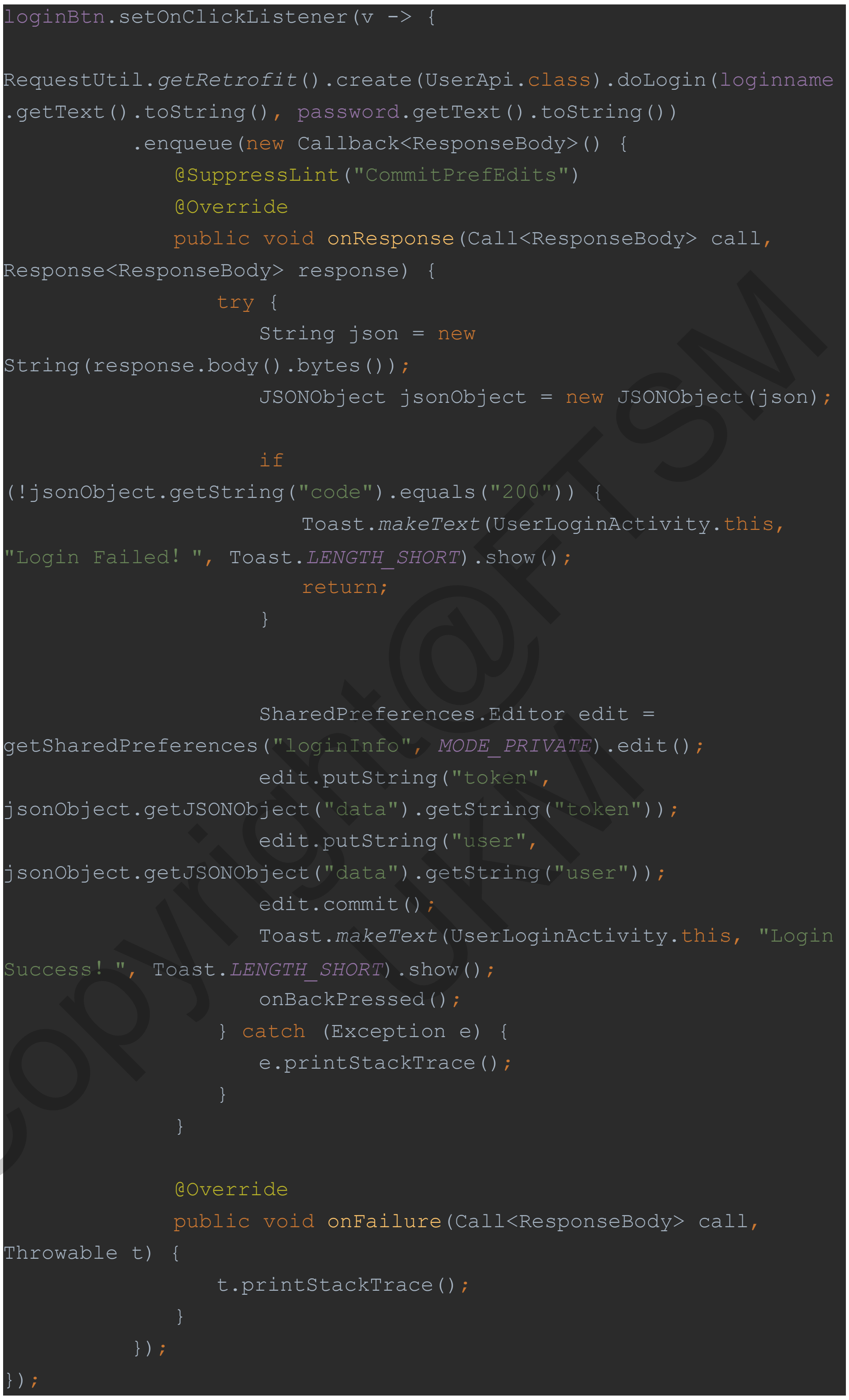
When the user clicks the login button loginBtn, the system will call RequestUtil.getRetrofit() to obtain the Retrofit instance and send a login request through the UserApi interface, passing the input username and password to the login interface doLogin() at the back end.

After the request is executed, the program processes the returned result in the onResponse() callback. The returned body is converted to a string and then parsed through JSONObject. If the response status code is not "200", the system will pop up a prompt of "Login failed".

When the login is successful, the program will extract the token and user strings from the returned JSON data and persistently store them locally through SharedPreferences (the file name is loginInfo). In subsequent network requests, this token will be automatically read through the interceptor and added to the request header to achieve user identity authentication.

After the storage is completed, the system pops up a prompt saying "Login successful" and calls onBackPressed() to return to the previous page, thus completing the entire login process.

If the request fails, such as a network error or server exception, the onFailure() callback will be entered and the error stack will be printed to facilitate debugging and troubleshooting.



```
loginBtn.setOnClickListener(v -> {

RequestUtil.getRetrofit().create(UserApi.class).doLogin(loginname
.getText().toString(), password.getText().toString())
          .enqueue(new Callback<ResponseBody>() {
             @SuppressLint("CommitPrefEdits")
             @Override
             public void onResponse(Call<ResponseBody> call,
Response<ResponseBody> response) {
                 try {
                     String json = new
String(response.body().bytes());
                     JSONObject jsonObject = new JSONObject(json);

                     if
(!jsonObject.getString("code").equals("200")) {
                         Toast.makeText(UserLoginActivity.this,
"Login Failed! ", Toast.LENGTH_SHORT).show();
                         return;
                     }


                     SharedPreferences.Editor edit =
getSharedPreferences("loginInfo", MODE_PRIVATE).edit();
                     edit.putString("token",
jsonObject.getJSONObject("data").getString("token"));
                     edit.putString("user",
jsonObject.getJSONObject("data").getString("user"));
                     edit.commit();
                     Toast.makeText(UserLoginActivity.this, "Login
Success! ", Toast.LENGTH_SHORT).show();
                     onBackPressed();
                 } catch (Exception e) {
                     e.printStackTrace();
                 }
             }

             @Override
             public void onFailure(Call<ResponseBody> call,
Throwable t) {
                 t.printStackTrace();
             }
          });
});
```

Figure 4.4 Code

When the payment button is clicked, the system will call the doPayment() method to start the payment process.

```
binding.idPayment.setOnClickListener(v -> doPayment());
```

Figure 4.5 Code

The filtering function of the order page has been implemented. By returning a View.OnClickListener listener, responses to different order status buttons have been achieved.

When a user clicks on different status buttons (such as All Orders, Pending Payment, Pending Shipment, Pending Receipt, Pending Evaluation), the system will first uncheck the selected status of all buttons (setSelectedToFalse()), and then set the currently clicked button to selected.

Next, based on the order status code (ostatus) corresponding to the button, filter the order list dataList to generate a subset of orders with the corresponding status, and refresh the interface display through the adapter.setData() method.

In this way, users can conveniently click different buttons to view the order list of the corresponding status, improving the user experience of order management.

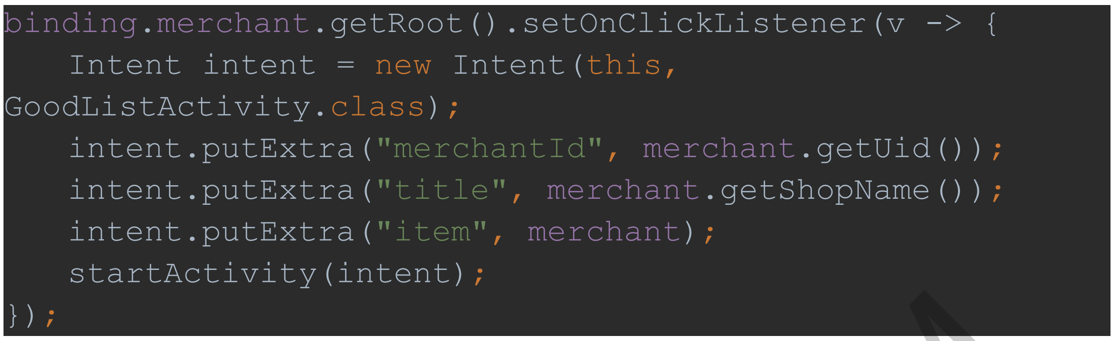
```
private View.OnClickListener getClickEvent() {
    return v -> {
        setSelectedToFalse();
        int id = v.getId();
        if (id == R.id.idTextBtn1) {
            idTextBtn1.setSelected(true);
            adapter.setData(dataList);
        } else if (id == R.id.idTextBtn2) {
            idTextBtn2.setSelected(true);
            List<OrderBrief> temp = new ArrayList<>();
            for (OrderBrief o : dataList) {
                if (o.getOstatus() == 1)
                    temp.add(o);
            }
            adapter.setData(temp);
        } else if (id == R.id.idTextBtn3) {
            idTextBtn3.setSelected(true);
            List<OrderBrief> temp = new ArrayList<>();
            for (OrderBrief o : dataList) {
                if (o.getOstatus() == 2)
                    temp.add(o);
            }
            adapter.setData(temp);
        } else if (id == R.id.idTextBtn4) {
            idTextBtn4.setSelected(true);
            List<OrderBrief> temp = new ArrayList<>();
            for (OrderBrief o : dataList) {
                if (o.getOstatus() == 3)
                    temp.add(o);
            }
            adapter.setData(temp);
        } else if (id == R.id.idTextBtn5) {
            idTextBtn5.setSelected(true);
            List<OrderBrief> temp = new ArrayList<>();
            for (OrderBrief o : dataList) {
                if (o.getOstatus() == 4)
                    temp.add(o);
            }
            adapter.setData(temp);
        }
    };
}
```

Figure 4.6 Code

When a user clicks on a merchant entry in the merchant details, a click event is triggered, initiating a jump to the GoodListActivity page.

When jumping, the merchant's ID (merchantId), merchant name (title), and the entire merchant object (item) will be passed through Intent, so that the target page can load the corresponding merchant's product data and title display based on the incoming information.

This realizes the navigation from the merchant list to the product detail page, enhancing the user experience.



```
binding.merchant.getRoot().setOnClickListener(v -> {
    Intent intent = new Intent(this,
GoodListActivity.class);
    intent.putExtra("merchantId", merchant.getUid());
    intent.putExtra("title", merchant.getShopName());
    intent.putExtra("item", merchant);
    startActivity(intent);
});
```

Figure 4.7 Code

When the user clicks the "Submit Comment" button, the system first acquires the comment content in the input box, removes the blank Spaces before and after, and then performs a non-empty check. If the Comment is empty, a prompt "Comment cannot be empty" pops up to prevent submission.

If the comment is valid, the system will hide the soft keyboard and then encapsulate a CommentBean object, which contains the ID (postId) of the current post, the user ID, and the comment content.

Then, the back-end interface addPostComment is called through Retrofit to send the comment data. After the request is successful, clear the comment input box and refresh the comment list (by calling the getData() method).

```
binding.commentSubmit.setOnClickListener(v -> {
    String                          comment                          =
binding.commentInput.getText().toString().trim();
    if (comment.length() == 0) {
        showToast("Comment cannot be empty");
        return;
    }

    hideSoftInput();
    CommentBean commentBean = new CommentBean();
    commentBean.setGid(postId);
    commentBean.setUid(user.getUid());
    commentBean.setCcontent(comment);

getRetrofit().create(CommentApi.class).addPostComment(com
mentBean).enqueue(new Callback<ResponseBody>() {
        @Override
        public   void   onResponse(Call<ResponseBody>   call,
Response<ResponseBody> response) {
            if (response.isSuccessful()) {
                binding.commentInput.setText("");
                getData();
            }
        }

        @Override
        public   void   onFailure(Call<ResponseBody>   call,
Throwable t) {

        }
    });
});
```

Figure 4.8 Code

c. Vue management backend permission control

The merchant management background part of this system adopts Vue-3 combined with vue-router to achieve front-end routing management. The routing configuration file defines multiple routing rules, mainly including:

The root path/automatically redirects to the Login page /user/Login.

The main framework page /main contains multiple sub-routes, such as the home page, user management, product management, order management, carousel management, etc. Each module corresponds to an independent component.

Independent routing for the login page and registration page.

Wildcard routing /:pathMatch(.*) is used to match all undefined paths and redirect to the 404 page.

Furthermore, a simple permission control logic is implemented through the router.beforeEach navigation guard:

Read the token of the current user's login status through the Vuex store.

A whitelist route (login and registration pages) has been defined, allowing unlogged-in users to access it.

When a user is not logged in and attempts to access a page outside the whitelist, it will automatically redirect back to the login page.

This routing design ensures that users cannot access other pages of the management backend when they are not logged in, guaranteeing system security.

At the same time, roles were designed, separating the different functions of administrators and merchants.

```
import { createRouter, createWebHistory } from 'vue-router'

import Main from '../views/Main.vue'

import HomePage from '../views/childViews/HomePage.vue'

import store from '../store'


const routes = [

  {

    path: '/',

    redirect: '/user/Login'

  },{

    path: '/main',

    name: 'Main',

    component: Main,

    children: [{

      path: '',

      name: 'default',

      component: HomePage

    }, {

      path: 'homePage',

      name: 'HomePage',

      component: HomePage

    }, {

      path: 'userManage',

      name: 'UserManage',
```

Figure 4.9 Code

```
component: () => import('@/views/childViews/GoodManage.vue')
    }, {
      path: 'commentManage',
      name: 'CommentManage',
      component: () => import('@/views/childViews/CommentManage.vue')
    }, {
      path: 'homePage',
      name: 'HomePage',
      component: () => import('@/views/childViews/HomePage.vue')
    }, {
      path: 'orderManage',
      name: 'OrderManage',
      component: () => import('@/views/childViews/OrderManage.vue')
    }, {
      path: 'orderDeliveryManage',
      name: 'OrderDeliveryManage',
      component: () =>
import('@/views/childViews/OrderDeliveryManage.vue')
    }, {
      path: 'bannerManage',
      name: 'BannerManage',
```

Figure 4.10   Code

```
component: () => import('@/views/childViews/BannerManage.vue')
    }, {
      path: 'selectedManage',
      name: 'SelectedManage',
      component: () =>
import('@/views/childViews/SelectedManage.vue')
    }, {
      path: 'categoryManage',
      name: 'CategoryManage',
      component: () =>
import('@/views/childViews/CategoryManage.vue')
    }, {
      path: 'typeManage',
      name: 'TypeManage',
      component: () => import('@/views/childViews/TypeManage.vue')
    }, {
      path: 'merchantInfo',
      name: 'MerchantInfo',
      component: () => import('@/views/MerchantInfo.vue')
    }]
  }, {
    path: '/user/Login',
    name: 'Login',
    component: () => import('@/views/Login.vue')
```

Figure 4.11 Code

```
{
    path: '/:pathMatch(.*)',
    component: () => import('@/views/NotFound.vue')
  }
]


const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
})





router.beforeEach((to, from, next) => {
  const token = store.getters.token;

  const whiteList = ['Login', 'Register'];

  if (token == null && !whiteList.includes(to.name)) {
    next({ name: 'Login' });
  } else {
    next();
  }
})
```

Figure 4.12 Code

4.3 MAIN MODULES AND COMPONENTS OF THE SYSTEM

1. Back-end: user module, product module, order module, comment module, etc
2. Android: Login and registration, home page menu, Order placement, My Orders, post

module

3. Management backend: Login, menu management, order list, user list, comment list

4. Database: MySQL, with 12 main tables created. The er diagram is as follows

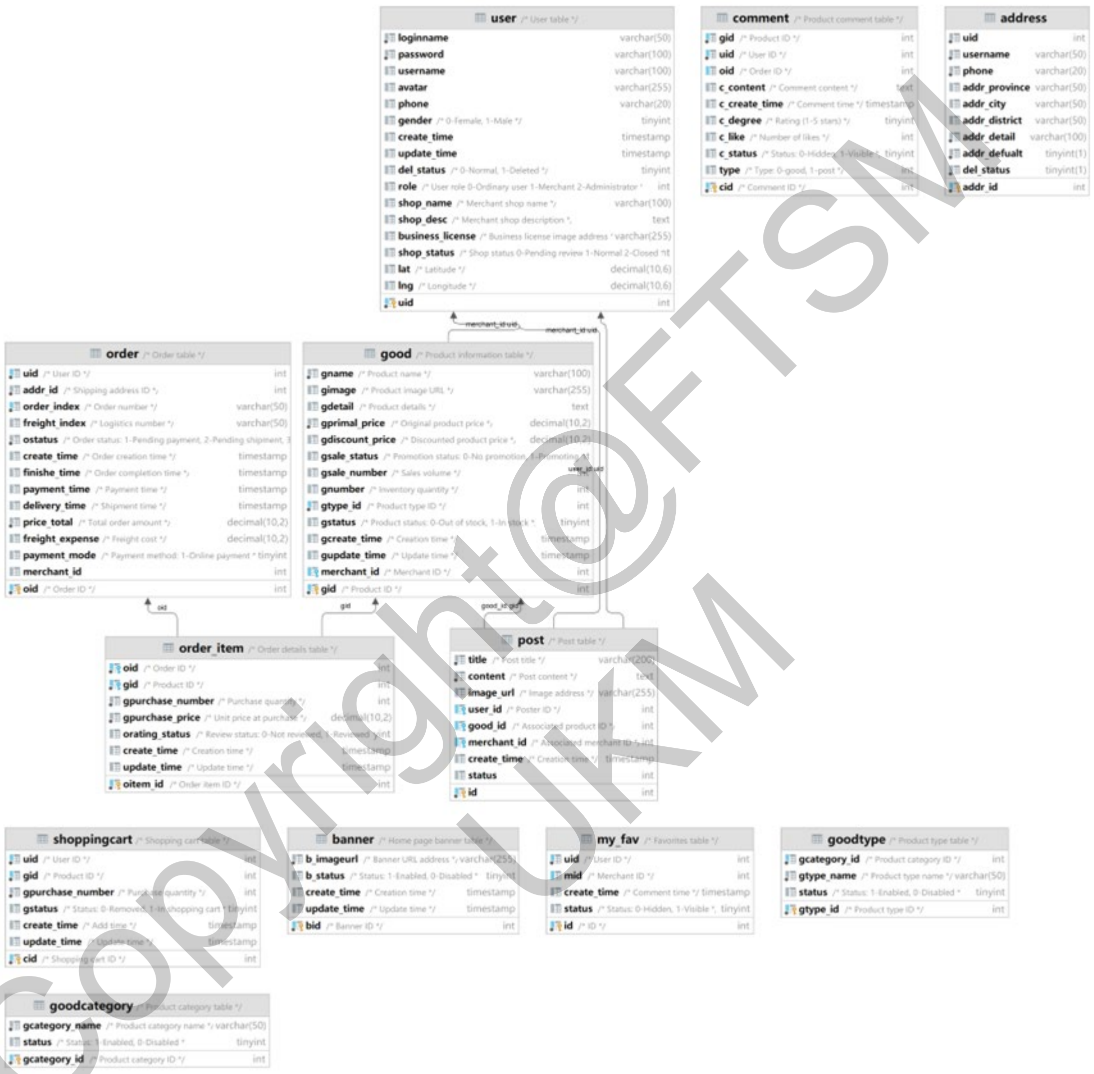


Figure 4.13 Main Modules and Components

4.4 USER INTERFACE DISPLAY

Screenshot description of the App operation interface: login page, menu page, order page, personal centre page, etc.

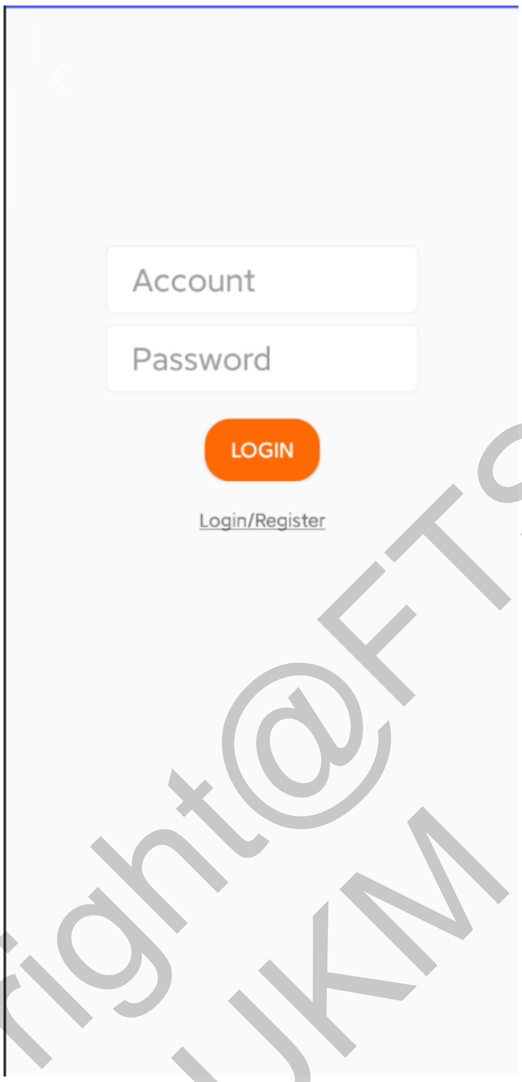


Figure 4.14 User Interface

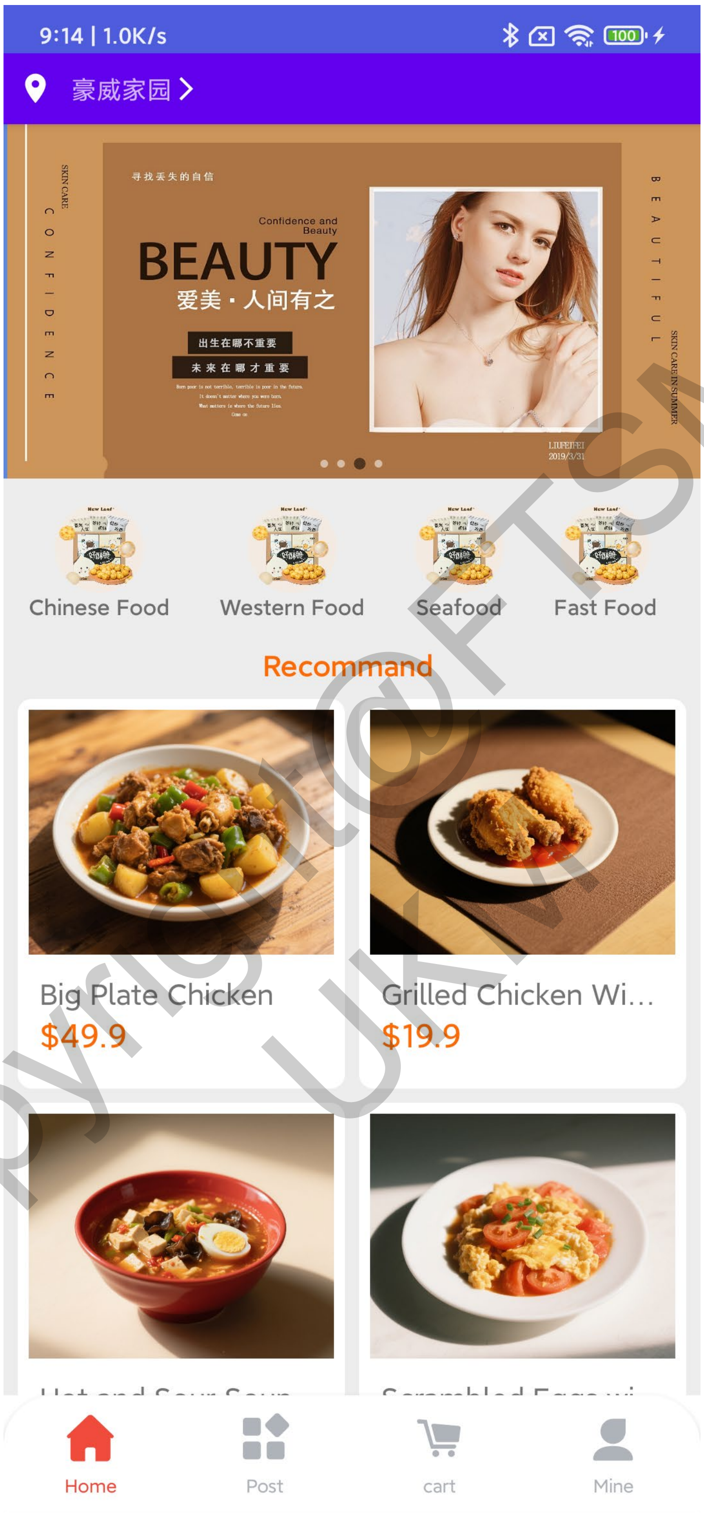


Figure 4.15 User Interface

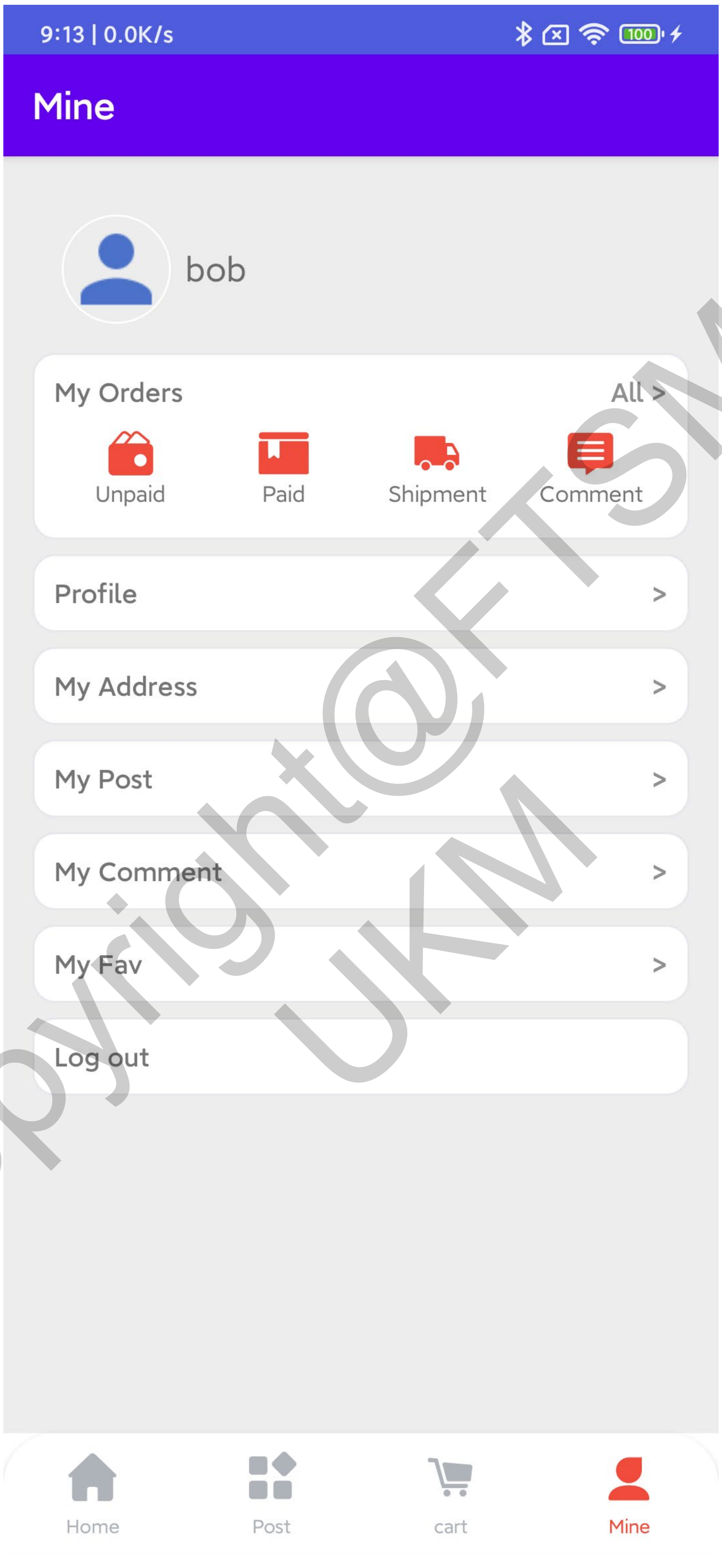
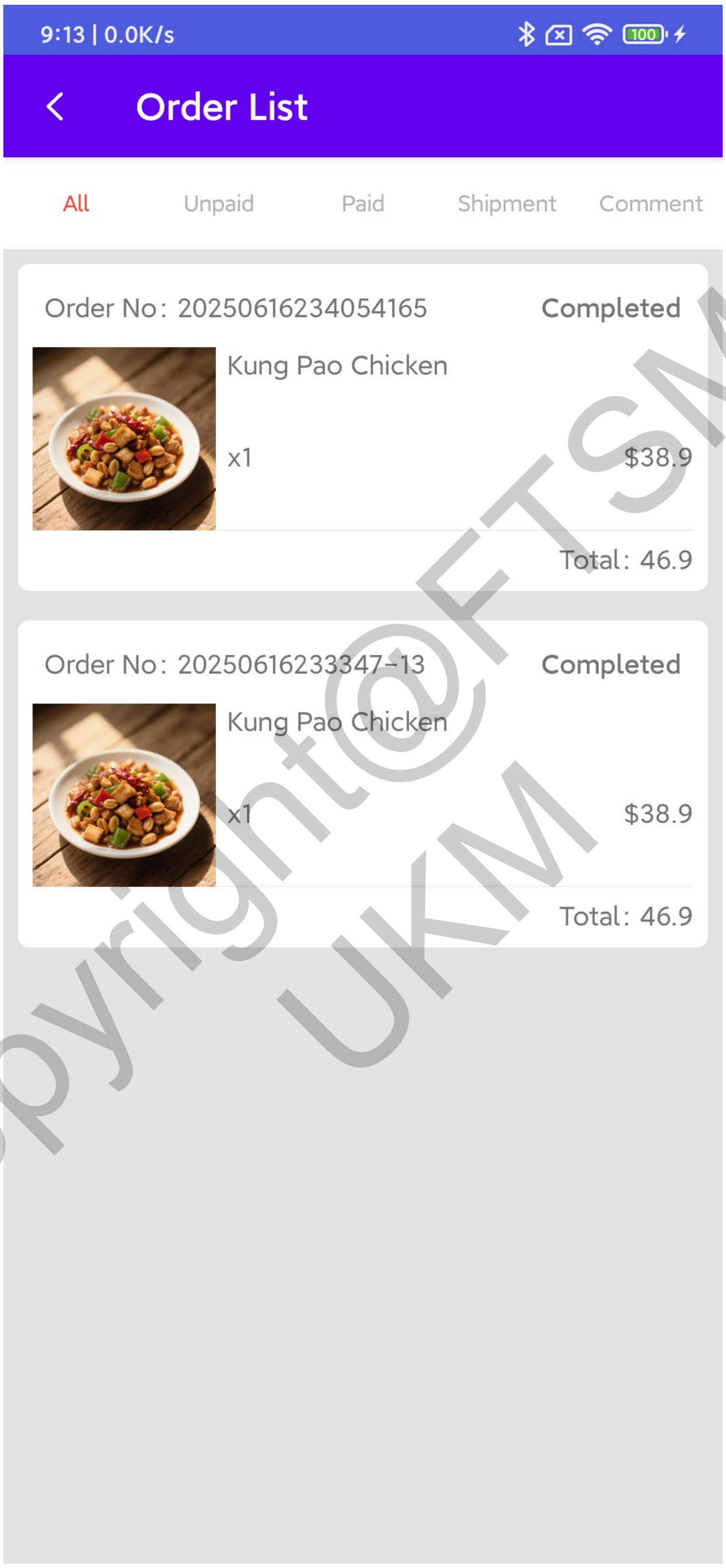


Figure 4.16 User Interface


9:13 | 0.0K/s

Order List

All | Unpaid | Paid | Shipment | Comment

Order No: 20250616234054165 Completed

Kung Pao Chicken

x1 $38.9

Total: 46.9

Order No: 20250616233347-13 Completed

Kung Pao Chicken

x1 $38.9

Total: 46.9

Figure 4.17 User Interface

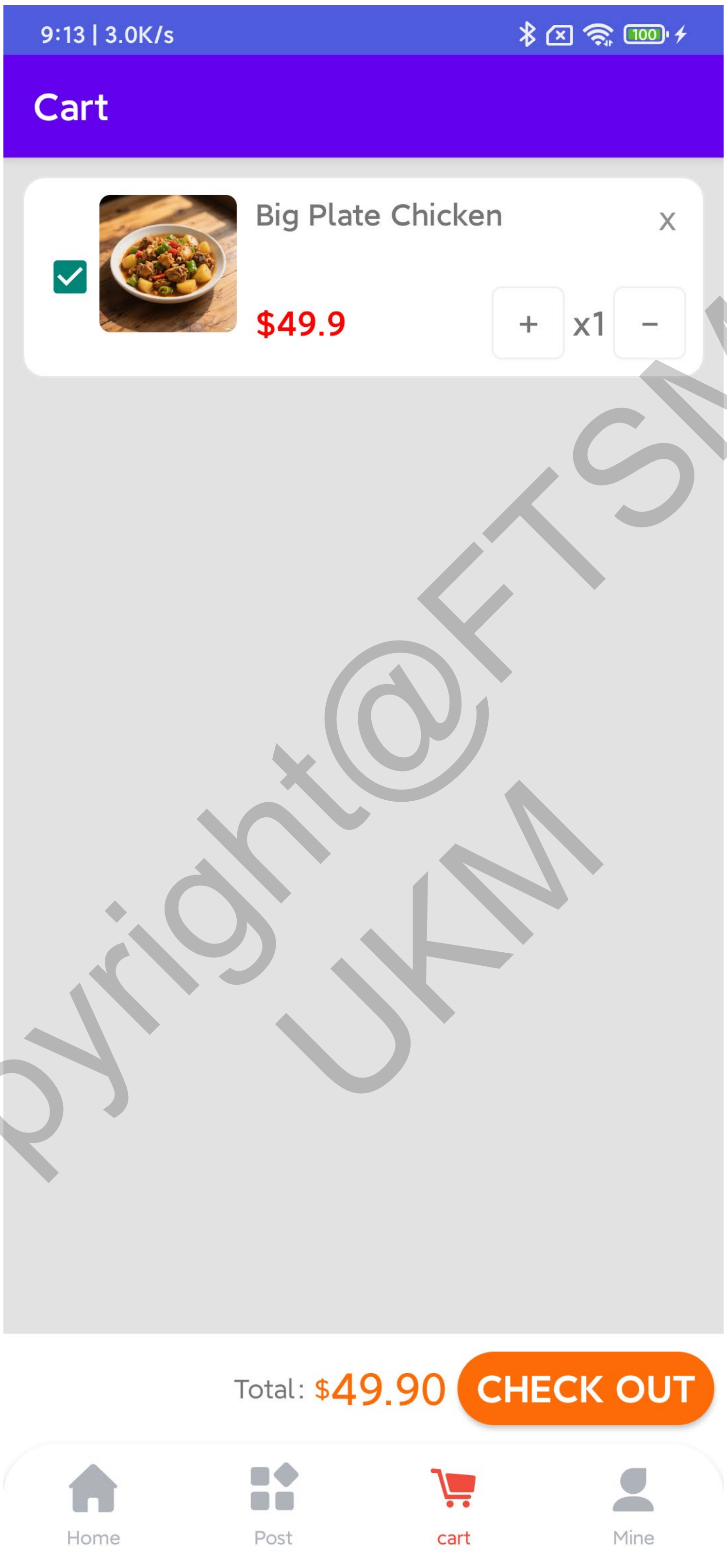
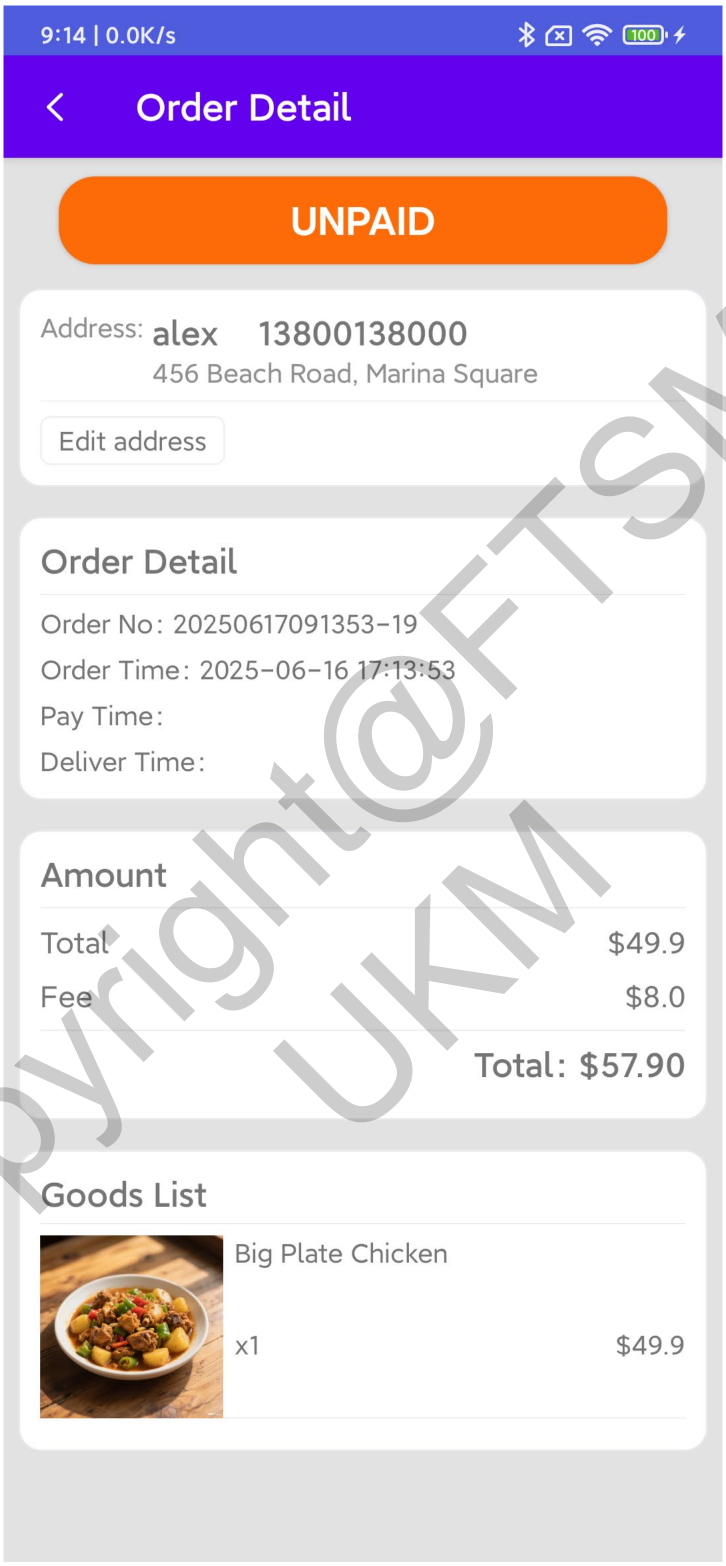


Figure 4.18 User Interface



Address: alex 13800138000
456 Beach Road, Marina Square

Order No: 20250617091353-19
Order Time: 2025-06-16 17:13:53
Pay Time:
Deliver Time:

Total	$49.9
Fee	$8.0
	Total: $57.90

Big Plate Chicken
x1 $49.9

Figure 4.19 User Interface

Screenshots of the management backend: login page, order placement page, order details page

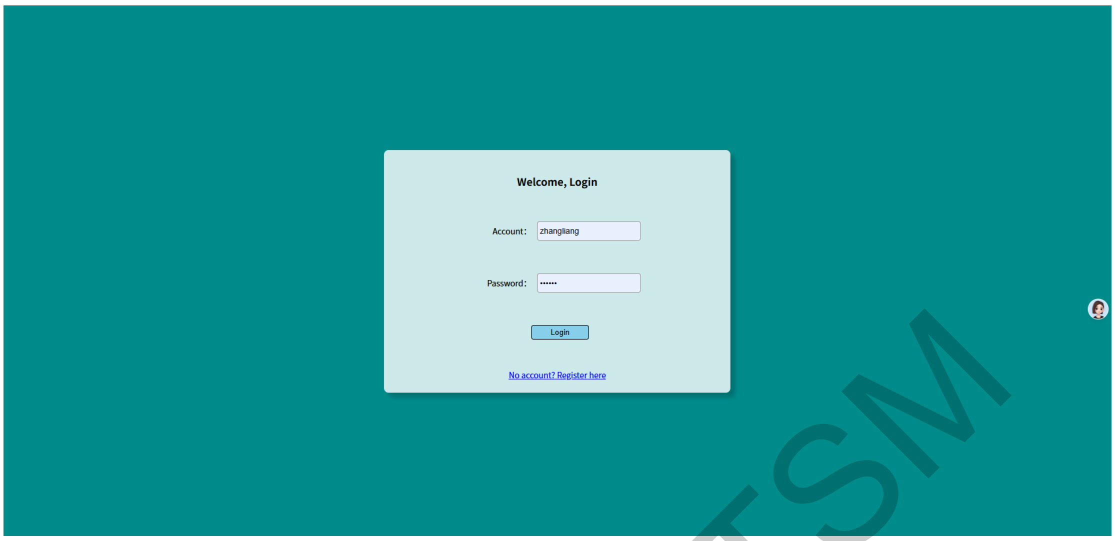


Figure 4.20 creenshots of the management backend

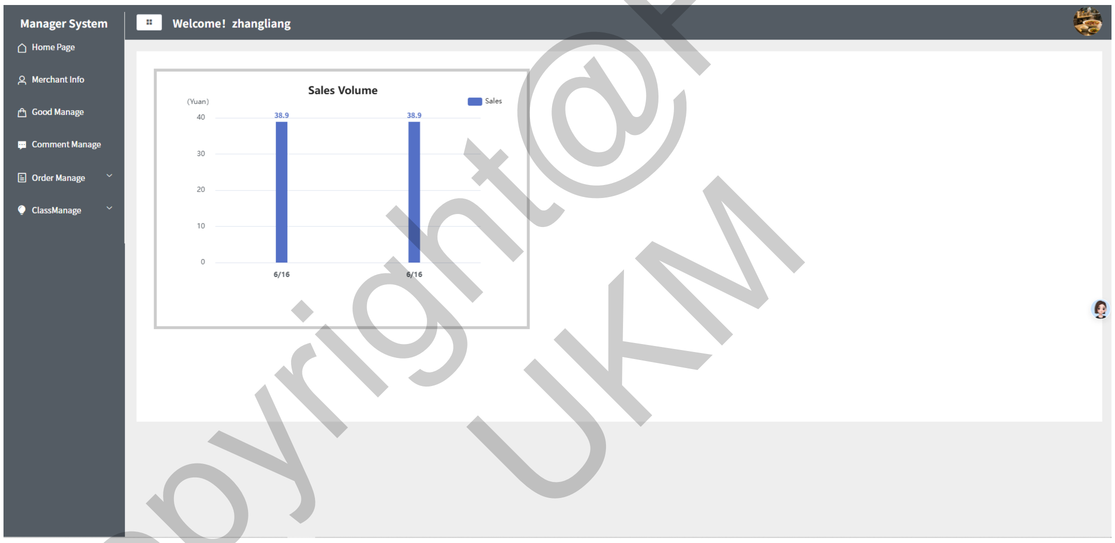
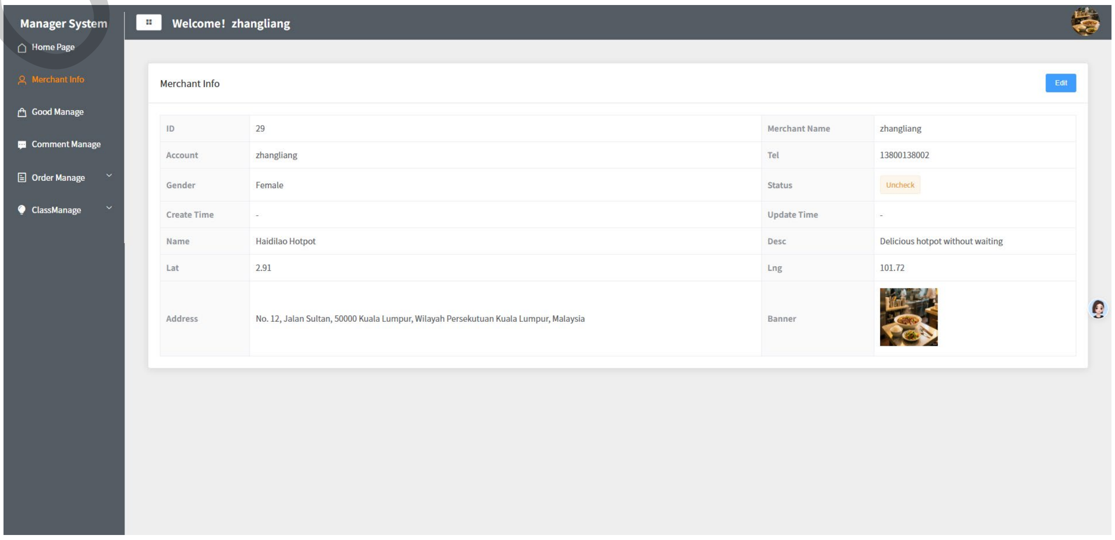


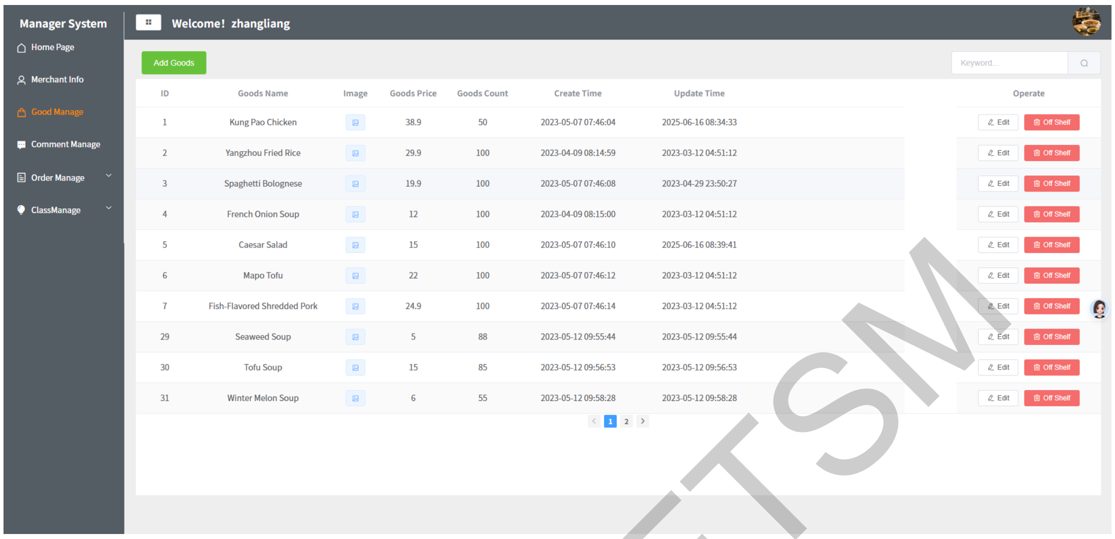
Figure 4.21 creenshots of the management backend



Merchant Info

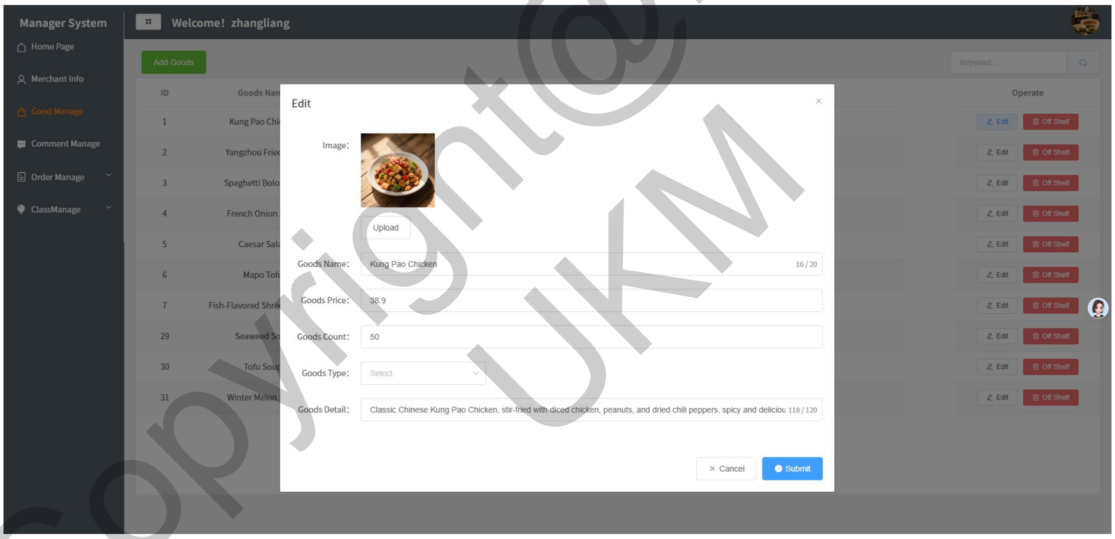
ID	29	Merchant Name	zhangliang
Account	zhangliang	Tel	13800138002
Gender	Female	Status	Uncheck
Create Time	-	Update Time	-
Name	Haidilao Hotpot	Desc	Delicious hotpot without waiting
Lat	2.91	Lng	101.72
Address	No. 12, Jalan Sultan, 50000 Kuala Lumpur, Wilayah Persekutuan Kuala Lumpur, Malaysia	Banner	

Figure 4.22 creenshots of the management backend



ID	Goods Name	Image	Goods Price	Goods Count	Create Time	Update Time	Operate
1	Kung Pao Chicken		38.9	50	2023-05-07 07:46:04	2025-06-16 08:34:33	Edit Off Shelf
2	Yangzhou Fried Rice		29.9	100	2023-04-09 08:14:59	2023-03-12 04:51:12	Edit Off Shelf
3	Spaghetti Bolognese		19.9	100	2023-05-07 07:46:08	2023-04-29 23:50:27	Edit Off Shelf
4	French Onion Soup		12	100	2023-04-09 08:15:00	2023-03-12 04:51:12	Edit Off Shelf
5	Caesar Salad		15	100	2023-05-07 07:46:10	2025-06-16 08:39:41	Edit Off Shelf
6	Mapo Tofu		22	100	2023-05-07 07:46:12	2023-03-12 04:51:12	Edit Off Shelf
7	Fish-Flavored Shredded Pork		24.9	100	2023-05-07 07:46:14	2023-03-12 04:51:12	Edit Off Shelf
29	Seaweed Soup		5	88	2023-05-12 09:55:44	2023-05-12 09:55:44	Edit Off Shelf
30	Tofu Soup		15	85	2023-05-12 09:56:53	2023-05-12 09:56:53	Edit Off Shelf
31	Winter Melon Soup		6	55	2023-05-12 09:58:28	2023-05-12 09:58:28	Edit Off Shelf

Figure 4.23 creenshots of the management backend



Edit

Image:

Goods Name: Kung Pao Chicken 16 / 20

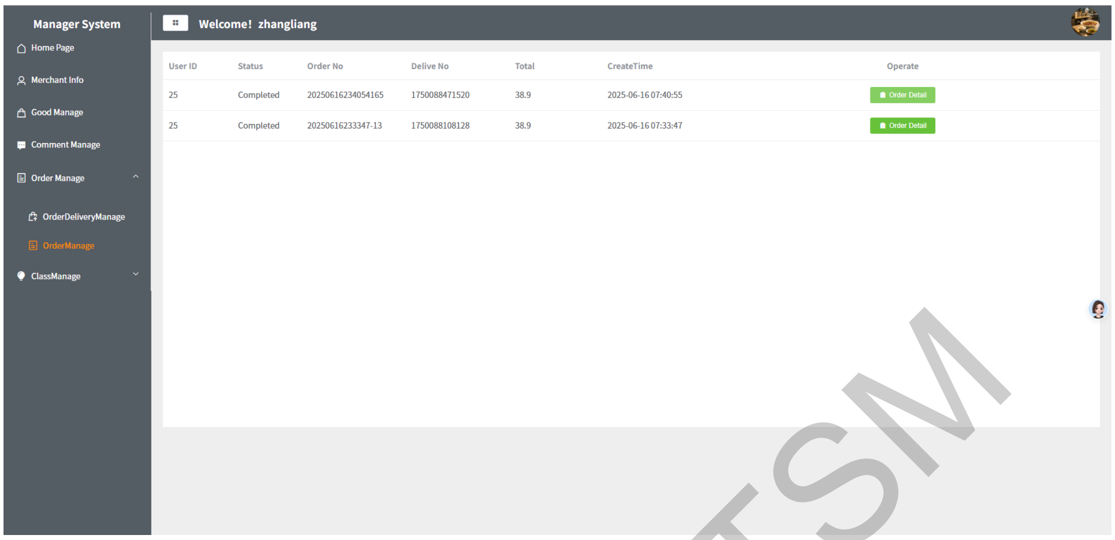
Goods Price: 38.9

Goods Count: 50

Goods Type: Select

Goods Detail: Classic Chinese Kung Pao Chicken, stir-fried with diced chicken, peanuts, and dried chili peppers, spicy and deliciou 118 / 120

Figure 4.24 creenshots of the management backend



User ID	Status	Order No	Delive No	Total	CreateTime	Operate
25	Completed	20250616234054165	1750088471520	38.9	2025-06-16 07:40:55	Order Detail
25	Completed	20250616233347-13	1750088108128	38.9	2025-06-16 07:33:47	Order Detail

Figure 4.25 creenshots of the management backend

4.5 SUMMARY

4.5.1 Problems Encountered and Solutions

When configuring Spring Boot with JWT, I encountered many pitfalls. The online tutorials were not quite the same. Finally, I referred to the official documentation. When doing interception verification, I encountered some problems. Finally, with the help of Google, they were solved.

For Android network requests, Retrofit was used. Some issues were encountered in data format and data parsing. Initially, the original method was to convert strings to json and parse them manually. Later, it was upgraded to directly parsing data objects.

Some issues have also been encountered with the adaptation of different versions of Android. The adaptation varies among different systems and versions, especially in the file reading and writing section, which has taken a lot of effort.

Due to the introduction of android's advanced viewmodel and viewbinding components, I encountered a problem where child components could not be obtained when nesting Views. It was only after referring to the official documentation and excellent code on github that the issue was resolved

Vue's permission routing control is not particularly strict. Front-end restrictions are easily bypassed, and interceptors are also added at the back end for secondary verification

In addition, in terms of database design, the correlation between tables reflects the design of the data model. During the business adjustment process, several modifications were made.

4.5.2 Content Summary

The entire system's functions have been basically completed. Although there aren't many features, all three ends can run smoothly. The App can place orders, make payments, comment, and post. The back-end can manage, and the interface at the back end is also relatively clear. There are still many areas that can be optimized, such as the UI design being too simple and the interface security not being high, etc. We plan to continue to improve them in the later stage.

5.0 CONCLUSION

5.1 INTRODUCTION

The subsequent section presents background and project scope, prominent system features, usage patterns, and why a system is undertaken. The project is to create an online ordering and review system for food with social features to increase user interest and trust in their decision-making.

The system not only allows users to select restaurants in a convenient manner but also allows them to gain others' experience and opinion at dining through social interactions.

5.2 TEST PLAN

5.2.1 Test Objectives

The aim of this testing is to verify the fundamental functionality of the food order and review system, i.e., the user registration, login, product browsing, order generation, payment, review posting, and social interaction modules. The test will establish whether the modules are behaving as anticipated, and whether the system is stable, responsive, and operational.

5.2.2 Test Scope

This test guarantees the general functional modules like user registration and login, restaurant discovery and view menu, ordering and payment, posting review and administration, and backend administration module. The test will focus on integrity and stability testing of the modules to allow users to conduct registration, login, order process, and review posting effortlessly. The backend management feature will also be tested to verify whether the administrators are able to manage user, order, and review data efficiently. The test will be performed to verify whether the features of the system work properly with different setups, meeting user demands and providing a simple operation experience.

5.2.3 Test Procedures

This test follows the black-box testing strategy, which is not interested in anything except verifying whether the system's functionalities are as desired, without bothering about the internal implementation or structure. By the simulation of user actions, this test verifies whether primitive modules such as user registration, log in, ordering, payment, and posting reviews behave as desired. Black-box testing primarily involves inputting and analyzing the output responses to verify the correctness, stability, and usability of the system under different conditions.

5.3 TEST CASE DESIGN

5.3.1 User Registration and Login Module

Use Case TC-001	**User Registration**
Test Objective	Verify that the user registration function works correctly
Pre-conditions	The user is not registered
Test steps	Enter a valid username and password, submit the registration request
Input data	Username, password.
Expected result	Registration successful, redirect to the login page.
Test Status	Passed

Table 5.1 Use Case TC-001

Use Case TC-003	**Login Failure**
Test Objective	Verify the behavior when the user logs in with an incorrect password.
Pre-conditions	The user is already registered.
Test steps	Enter the username and incorrect password, click the login button.
Input data	Incorrect password.
Expected result	Error message displayed, login failed.
Test Status	Passed

Table 5.2 Use Case TC-002

Use Case TC-003	**Login Failure**
Test Objective	Verify the behavior when the user logs in with an incorrect password.
Pre-conditions	The user is already registered.
Test steps	Enter the username and incorrect password, click the login button.
Input data	Incorrect password.
Expected result	Error message displayed, login failed.
Test Status	Passed

Table 5.3 Use Case TC-003

5.3.2 Food Ordering and Review Module

Use Case TC-004	**Restaurant Selection and Menu Browsing**
Test Objective	Verify that the restaurant selection and menu browsing functions work correctly.
Pre-conditions	The user is logged in.
Test steps	Select a restaurant and view the menu.
Input data	Restaurant name.
Expected result	Display restaurant menu.
Test Status	Passed

Table 5.4 Use Case TC-004

Use Case TC-005	**Order Generation and Payment**
Test Objective	Verify that the order generation and payment functions work correctly.
Pre-conditions	The user has selected a menu and added items to the shopping cart.
Test steps	Complete order generation and choose a payment method.
Input data	Product information, payment method.
Expected result	Order generated and payment successful.
Test Status	Passed

Table 5.5 Use Case TC-005

Use Case TC-006	Review Posting
Test Objective	Verify that the review posting function works correctly.
Pre-conditions	The user has completed an order and is on the review page.
Test steps	Enter the review content, submit the review.
Input data	Review content.
Expected result	Review successfully posted and displayed on the restaurant page.
Test Status	Passed

Table 5.6 Use Case TC-006

5.4 TEST EXECUTION AND RESULT

5.4.1 Test Environment

	Specification
Operating System	Windows 11
Testing Device	iPhone 15
Browser	Google Chrome
Database	Google Chrome

Table 5.7 Test Environment

5.4.2 Test Results Summary

All system working modules were confirmed to be working fine following testing. Core functionalities such as user registration, login, payment, order generation, and review posting passed functionality tests. Delays in responses were experienced in some functionalities, i.e., payment, when under high load and will need to be optimized. The entire system performed well in stability, and all functionalities passed design requirements. All issues faced in testing were addressed through optimization measures so that the system can run efficiently on diverse environments.

5.5 ISSUE SUMMARY AND FIXES

No.	Problem Description	Discovery Stage	Remediation Method	Result
BUG01	Payment operation responds slowly under high concurrency.	Order Generation and Payment Module	Implement asynchronous processing and use message queues to optimize the payment interface, reducing delays.	Fixed

BUG02	User reviews are not immediately displayed after submission.	Review Posting Module	Resolve data synchronization issues and use frontend auto-refresh or WebSocket for real-time updates of reviews.	Fixed

Table 5.8 BUG

5.6 CONCLUSION

Every major functional module of the food ordering and review system, i.e., user registration and login, selection of restaurant and browsing of menu, creation of order and payment, posting and management of reviews, and administration in the backend, is addressed by this test. Through rigorous system testing, every function, stability, and performance of each module were verified. All major functions, e.g., user operations, payment process, and posting of reviews, passed with success in the tests, ensuring the system operates in a normal manner in different usage scenarios.

In the test period, there were some performance issues discovered, such as delayed execution of the payment feature under heavy concurrent usage and real-time rendering issues with reviews. The corresponding factor optimization measures were proposed and implemented successfully. Overall, the core functions of the system are in line, providing users with a smooth operation experience and efficient backend management features.

This test successfully validated the quality of the system and served as a basis for later optimization and release.

6.0 APPRECIATION

Faculty of Technology and Information Science Grant FTSM and LI XIANG LIN A191788

7.0 REFERENCES

Gupta S ,Mishra A .Design and implementation of adaptive filtering-based recommendation systems for maximizing publisher-side revenue[J].Multimedia Tools and Applications,2025,(prepublish):1-28.

Li X ,Hu G .Design and Implementation of an Electronic Journal Resource Recommendation System Integrating User Profile and Knowledge Graph[J].Frontiers in Computing and Intelligent Systems,2024,10(2):74-78. (in Chinese)

Zhou J ,Jin J .Design and Implementation of the Internet of Things Virtual Art Exhibition Hall for Intelligent Computing[J].Art and Performance Letters,2024,5(4):

Pinon S ,Burnay C ,Linden I .Business-Driven Data Recommender System: Design and Implementation of data recommender system [J].The Journal of Computer Information Systems,2024,64(5):593-606.

Shao W ,Liu K.Design and Implementation of Online Ordering System Based on SpringBoot[J].Journal of Big Data and Computing, 2024, 2 (3) :

Li B ,Cuison L .Design and implementation of a personalized course recommendation system for MOOCs based on deep learning[J].Academic Journal of Computing &amp; Information Science,2024,7(7):

Chi C ,Zeng X ,Bruniaux P , et al.An intelligent recommendation system for personalised parametric garment patterns by integrating designer's knowledge and 3D body measurements.[J].Ergonomics,2024,21-21.

Alwedyan S .Optimal location selection of a casual-dining restaurant using a multi-criteria decision-making (MCDM) approach:Planning and Design Implementation[J].International Review for Spatial Planning and Sustainable Development, 2024, 12 (1) : 156-172.

Chibuzor U ,Robert Z ,Farzaneh D , et al.Design and Implementation of a Product Recommendation System with Association and Clustering Algorithms[J].Procedia Computer Science, 2020,219512-520.

Yang Y .Design and Implementation of Online Food Ordering System Based on Springcloud[J].Information Systems and Economics, 2022, 3 (4) :

Xiang N ,Xiaolan K K .Design and Implementation of a Personalized Tourism Recommendation System Based on the Data Mining, and Collaborative

Filtering Algorithm [J]. Journal of Computational Intelligence and Neuroscience, 2022202142097-1424097.

Suk H C ,Qian P ,Seop W R .Design and Implementation of the Machine Learning-based Restaurant Recommendation [J].Journal of Digital Contents Society,2020,21(2):259-268.

Wen Z ,Hu S ,Clercq D D , et al.Design, implementation, and evaluation of an Internet of Things (IoT) network system for restaurant food waste management[J].Waste Management, 2018732, 6 to 38.

Fang J S ,Mao J K ,Shen J .The Design and Implementation of Online Meal Ordering System[J].Advanced Materials Research, 2012191, 7 (562-564) : 1630-1633.

Acosta L, Gonzalez J E, Rodriguez N J, et al.Design and Implementation of a Service robot for a Restaurant.[J].I. J. Robotics and Automation, 2006, 21 (4) : 273-281.

LI XIANG LIN(A191788)

Dr. Kok Ven Jyn

Faculty of Information Technology & Science

National University of Malaysia