Index

0 Motivation & Storyline

Our story begins in a small room in Prince George's Park, 2023. Two tech-enthusiast students, Guoyi (1@sinopapers.com) and Jiajun(Ma@sinopapers.com), were studying at the National University of Singapore. One majoring in Computer Engineering with Data Science as 2nd Major and the other in Computer Engineering with interests in business. They both shared a passion for innovation and e-commerce. They noticed that despite the plethora of e-commerce platforms in the Singaporean market, none effectively catered to the needs of the Singaporean Chinese community wanting to purchase Chinese goods online.

Being part of the Chinese community themselves, Guoyi and Jiajun frequently faced difficulties in buying Chinese products. Traditional shopping platforms offered some Chinese goods, but the variety was limited, quality varied, and often faced language barriers, with unclear product descriptions. They believed that there were many others around them facing the same issues. So, the two young men decided to tackle this problem together. They wanted to develop an e-commerce platform specifically for the Chinese community, named "SinoShop". This platform would focus on offering a wide variety of Chinese goods, from daily necessities to electronic devices, from food to furniture. Moreover, they hoped to mitigate language barriers by using bilingual product introductions and customer support in both Chinese and English, making it more convenient for users to purchase what they needed.

Guoyi and Jiajun had clear division of labor. As a Computer Engineering student, Guoyi was responsible for building the technical framework of the platform, ensuring a user-friendly interface and stable system operation. Jiajun, a business management student, was in charge of operational strategy. He researched market demand, negotiated with suppliers to ensure that the products on the platform could meet the users' needs.

This was why they developed SinoShop. They hoped to solve the troubles the Chinese community faced when buying Chinese goods and to make it easier for them to purchase familiar Chinese products in Singapore through their efforts.

1 Abstract

SinoShop is a virtual network space for business activities on the Internet and a management environment to ensure the smooth operation of business. It is an important place to coordinate and merge the information flow, material flow and capital flow in an orderly, related, efficient, and popular way. Enterprises and businesses can make full use of the network infrastructure, payment platform, security platform, management platform and other shared resources provided by SinoShop to carry out their business activities effectively and at a low cost.

SinoShop is developed and implemented by using MySQL database, JSP and J2EE programming languages. SinoShop includes user registration, user login, commodity query, commodity addition, commodity deletion, user message, commodity evaluation and other major modules. It has the advantages of simple operation and high efficiency. With friendly interface, good flexibility, and stable operation, it is an ideal choice for neutron commerce.

SinoShop introduces its project development background and significance, describes the development and implementation process of the system, and makes a detailed analysis and description from the system requirements analysis, function module design, database design, detailed design to system testing.

2 Development Logfile

Our Develop Logfile and Timeline will be synchronized daily, please access it via:
https://blog.nus.edu.sg/chenguoyi/modules/cp2106/devlog/

3 Tech Stack

This chapter mainly introduces the relevant technologies involved in the SinoShop.

3.1 JavaScript

JavaScript is an object-based and event-driven client-side scripting language with relative security. It is also widely used in client-side web development and is commonly used to add dynamic features to HTML pages, such as responding to various user actions.

3.2 Ajax

The full name of Ajax is "Asynchronous JavaScript and XML", which refers to a web development technology for creating interactive web page applications. Ajax technology can provide highly interactive web applications, providing users with a richer page browsing experience. The implementation of Ajax functionality mainly relies on the XMLHttpRequest object and its properties and methods, sending requests and handling responses.

3.3 MySQL

MySQL is an open-source small-scale relational database management system.

3.4 SSM Framework

The SSM (Spring + SpringMVC + MyBatis) framework is integrated from the two open-source frameworks Spring and MyBatis (SpringMVC is part of Spring). It is often used as the framework for web projects with relatively simple data sources.

3.4.1 Spring

Spring is an open-source framework, a lightweight Java development framework that emerged in 2003. It was derived from some of the concepts and prototypes expounded by Rod Johnson in his book Expert One-On-One J2EE Development and Design. It was created to solve the complexity of enterprise application development. Spring uses basic JavaBeans to complete tasks that could only be done by EJBs in the past. However, the use of Spring is not limited to server-side development. From the perspectives of simplicity, testability, and loose coupling, any Java application can benefit from Spring. Simply put, Spring is a lightweight Inversion of Control (IoC) and Aspect-Oriented Programming (AOP) container framework.

3.4.2 SpringMVC

Spring MVC is a follow-up product to SpringFrameWork and has been integrated into Spring Web Flow. Spring MVC separates the roles of controllers, model objects, dispatchers, and handler objects, making them easier to customize. SpringMVC intercepts user requests in a project. Its core Servlet, the DispatcherServlet, plays the role of an intermediary or front-end, which directs user requests to match the Controller via HandlerMapping. The Controller corresponds to the specific actions performed in response to requests. SpringMVC is similar to struts in the SSH framework.

3.4.3 Mybatis

MyBatis is a persistence layer framework based on Java. Mybatis is an encapsulation of JDBC. It makes the operations of the database transparent. Mybatis's operations are centered around an SQLSessionFactory instance. Mybatis associates configuration files with Mapper files for each entity class through configuration files. The Mapper files configure the SQL statement mapping required by each class for the database. During each interaction with the database, you obtain an SQLSession from the SQLSessionFactory and then execute the SQL command. The page sends a request to the controller, the controller calls the business layer to process logic, the logic layer sends a request to the persistence layer, the persistence layer interacts with the database, then returns the result to the business layer, the business layer sends the processing logic to the controller, and the controller calls the view to display the data.

3.5 JSP

JSP (Java Server Pages) is a simplified Servlet design. It is a dynamic webpage technology standard established by Sun Microsystems and participated in by many companies. JSP technology is somewhat similar to ASP technology. It inserts Java program segments (Scriptlet) and JSP tags into traditional HTML (a subset of the standard generalized markup language) files (*.htm, .html), forming JSP files with the suffix (.jsp). Web applications developed with JSP are cross-platform and can run under Linux and other operating systems. JSP uses the Java programming language to write XML-like tags and Scriptlet to encapsulate the processing logic that produces dynamic web pages. Web pages can also access application logic existing on the server through tags and Scriptlet. JSP separates webpage logic from webpage design display, supports reusable component-based design, making the development of web-based applications fast and easy. JSP is a dynamic page technology, and its main purpose is to separate representation logic from Servlets.

3.6 B/S Mode

The B/S (Browser/Server) structure is a browser and server structure. It is a variation or improvement of the C/S structure that arose with the advent of Internet technology. In this structure, the user work interface is implemented through a WWW browser, a very small portion of transaction logic is implemented on the front end (Browser), but the main transaction logic is implemented on the server end (Server), forming a so-called three-tier structure. This greatly simplifies the load on the client computer, reduces the cost and workload of system maintenance and upgrades, and reduces the total cost of ownership (TCO). From the perspective of current technology, setting up a B/S structure network application on a LAN, and performing database applications under the Internet/Intranet mode, is relatively easy to grasp and cost-effective. It is a one-time development that allows different personnel to access and operate a common database in different ways; it can effectively protect the data platform and manage access rights, and the server database is also very safe.

The biggest advantage of the B/S model is that it can be operated anywhere without the need for the client to install any special software. As long as there is a computer with internet access, it can be used. The client requires zero maintenance, and system expansion is very easy. It has distribution characteristics and can process business at anytime and anywhere. Business expansion is simple and convenient, and server functionality can be increased by adding web pages. In terms of maintenance, changing web pages can synchronize updates for all users and has strong sharing characteristics. It achieves cross-platform system integrated services, provides open-ended foundation for heterogeneous machines, heterogeneous networks, and heterogeneous application services' online, networking, and unified services. However, C/S mode development is targeted, has certain requirements for the client, lacks universality, portability, business changes are not flexible, and maintenance and system upgrades are troublesome, and compatibility is poor. For different development tools, it is difficult for them to be compatible with each other and has significant limitations. New technologies are not easy to use. In addition, its development cost is high and requires technically competent personnel to complete. Figure 3-1 shows the network structure diagram of the B/S mode application system:

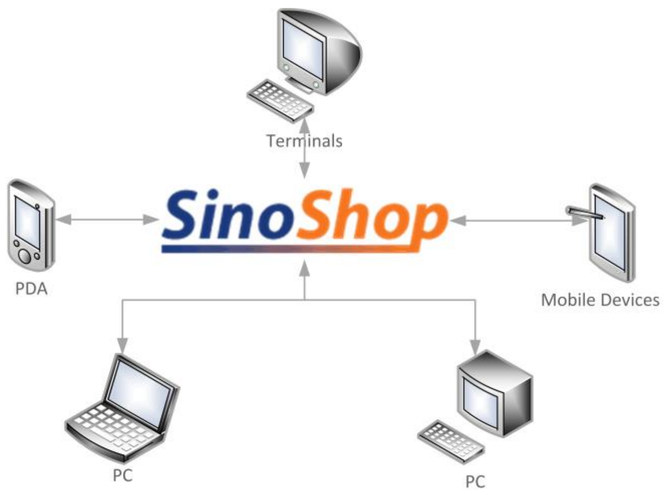


Figure 3-1 Network structure diagram of the B/S mode application system

3.7 EasyUI

jQuery EasyUI is a collection of UI plugins based on jQuery. The goal of jQuery EasyUI is to help web developers create feature-rich and beautiful UI interfaces more easily. Developers do not need to write complex JavaScript, nor do they need to have a deep understanding of CSS styles. All developers need to know are some simple HTML tags.

3.8 System Development Platform and Runtime Environment

3.8.1 System Development Platform

3.8.1.1 Eclipse

Eclipse is an open-source, extensible application development platform based on Java, providing programmers with a first-class Java integrated development environment (IDE). Initially used mainly for Java language development, Eclipse can support different computer languages such as C++ and Python through installing different plugins.

A Java EE version of Eclipse IDE is available on the official Eclipse website. Using Eclipse IDE for Java EE, you can create Java projects and also dynamic web projects.

Eclipse itself is just a framework platform, but the support of numerous plugins gives Eclipse flexibility that other IDE software with fixed functions finds hard to match. Many software developers use Eclipse as a framework to develop their own IDE.

Eclipse's plugin mechanism is a lightweight software componentized architecture. On the client platform, Eclipse uses plugins to provide all additional features, such as supporting languages other than Java. Existing separate plugins can already support C/C++ (CDT), Perl, Ruby, Python, telnet, and database development. The plugin architecture can support adding any extension to the existing environment, such as configuration management, and not just limited to supporting various programming languages.

The design philosophy of Eclipse is everything is a plugin. The core of Eclipse is very small, all other functions are added to the core of Eclipse in the form of plugins. The basic core of Eclipse includes graphic API (SWT/Jface), Java development environment plugin (JDT), plugin development environment (PDE), etc.

3.8.1.2 Tomcat Server

Tomcat is a small, lightweight application server, widely used in small and medium-sized systems and scenarios where concurrent user access is not very high and is the first choice for developing and debugging JSP programs. For a beginner, you can understand that once the Apache server is configured on a machine, it can be used to respond to access requests for HTML pages. In fact, Tomcat is an extension of the Apache server, but it runs independently. So, when you run Tomcat, it actually runs as a process independent of Apache.

Tomcat Server is a free open-source web application server, and the latest Servlet and JSP specifications are always reflected in Tomcat. Because of its advanced technology, stable performance, and free of charge, it is loved by Java enthusiasts and has been recognized by some software developers, becoming a popular web application server.

The technical advantages of the Tomcat environment mainly include the following aspects:

1. The application in Tomcat is a WAR (Web Archive) file. WAR is a web application format proposed by Sun, similar to JAR, which is also a compressed package of many files.
2. In Tomcat, the deployment of an application is very simple, you only need to put your WAR in the webapp directory of Tomcat, and Tomcat will automatically detect this file and decompress it.
3. Tomcat is not just a Servlet container, it also has the function of a traditional web server: handling HTML pages.
4. Tomcat can also be integrated with other software to achieve more functionality.

3.8.2 Runtime Environment

SinoShop is deployed on Alibaba ECS cloud server. The following are the key information about the running environment:

Table 3-1 Hardware Environment Table

Index Item	**Configuration Parameters**
Host Model	HP Z600
CPU Model	Xeon E5606 2.13GHz
CPU Quantity	2
Graphics Card	ATI FirePro V4800 1GB
Memory	DDR3 1333MHz ECC Unbuffered DIMM 24GB
Hard Drive	SATA 600GB*4

Table 3-2 Software Environment Table

Name	**Version**
Operating System	Windows Server 2012 R2
Development Tool	MyEclipse
Database	MySql 5.7
Web Server	Tomcat 7.0
Development Kit	JDK 1.8

4 Feature Design

The users of the SinoShop are mainly store owners and consumers. For these consumers, the functionalities to be implemented include browsing products (category browsing or condition-based searching), evaluating product features and information or the store owner, and leaving messages for the store owner. For store owners, functionalities include adding, modifying, and deleting product information, as well as querying product information.

4.1 Functional Requirements Analysis

The shopping mall system is developed according to the actual market situation and the requirements of networkization. Its goal is very clear, that is, to transform the offline trading mode of fixed-time and fixed-place stall goods into an online mode through the establishment of an online trading platform, making commodity trading more convenient, safe, standardized, and targeted.

The overall system functionality requirements are divided into user functionality and system management functionality. User functionality includes browsing products, logging in, registering, searching for products, payment, viewing personal information, etc. System management functionality includes administrator managing users, products, orders, and system settings. Based on the above functional requirements analysis, the main functions of the system are described through use case diagrams. The first step in constructing a use case model is to determine who the users are in the model, and the principles for determining users are: who maintains the system, who participates in the system, etc. The maintainer is internal to the system and has absolute control over the system; the participant is generally outside the system and beyond the control of the system. Now, it is determined that there are three types of use case models in this system, namely visitors, registered users, and system administrators. The functions of these three roles are described as follows:

4.1.1 Visitor

Visitors are unregistered users. They can browse and search for products. If they need to buy products, they must first register as website users. The main functions of visitors are as follows:

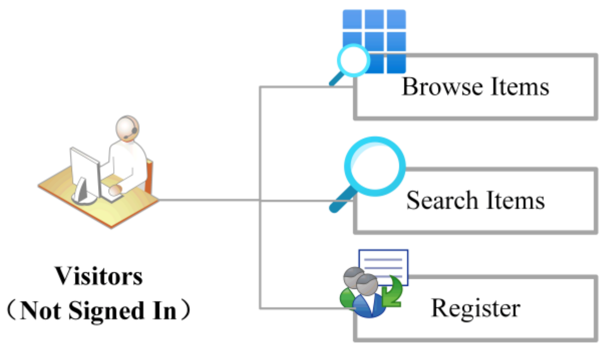


Figure 4-1 Visitor Use Case Diagram

4.1.2 Registered User

Registered customers are users who have been legally certified by the website. After logging in to the website, they can browse products, search for products, post products, collect products, purchase products, and view their personal center. The main functions of registered users are as follows (Figure 4-2 is a registered customer use case diagram):

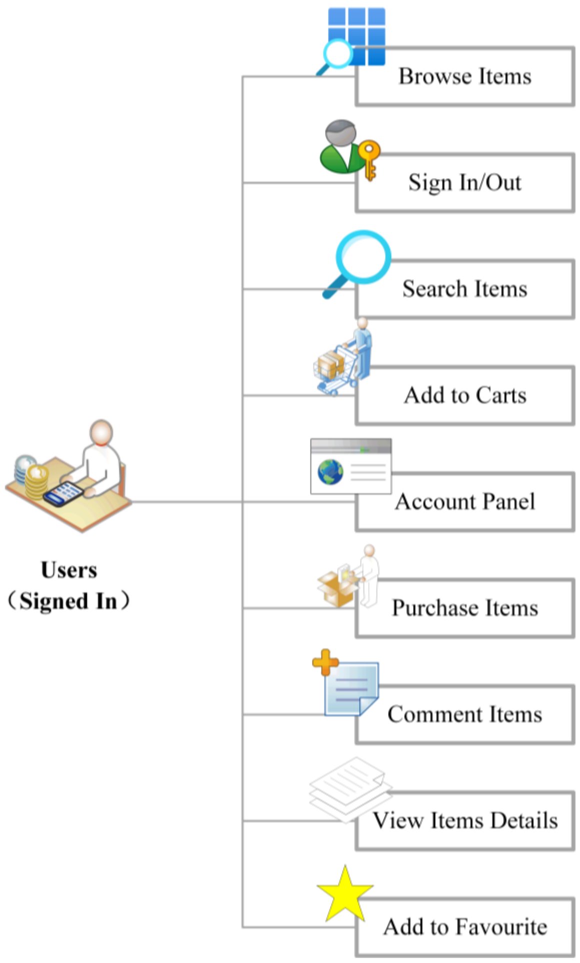


Figure 4-2 Customer Use Case Diagram

4.1.3 System Administrator

The system administrator is mainly responsible for the system's back-end management work, the main functions are as follows:

a. System Settings.
b. Item Management.
c. System Logfile.
d. Item Category.
e. User Management.
f. Order Management.
g. Comment Management.
h. Sales Statistics;

After determining the system users and user functions, a use case diagram of the mall system can be constructed, and the use case diagram of the entire system is shown in Figure 4-3 overall system use case diagram:

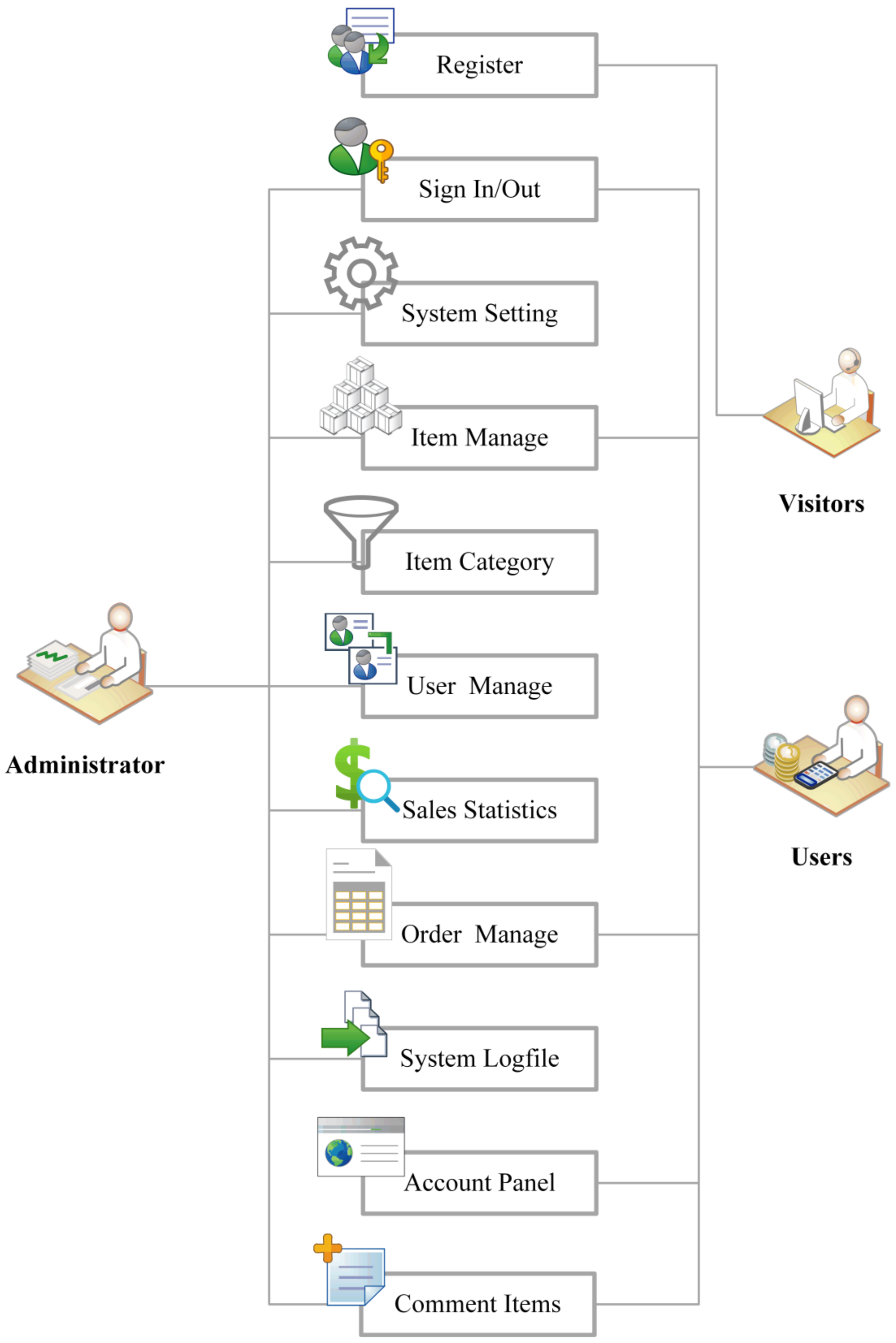


Figure 4-3 Overall System Use Case Diagram

4.2 Non-functional Requirement Analysis

4.2.1 System Practicality

System practicality refers to the design and development of system features to be as simple and practical as possible. Users can feel the convenience and speed of the system during use, without many cumbersome and redundant operations or functions.

4.2.2 System Security

The shopping mall system records important user information, including users' personal privacy information and account amount, etc. This information must be highly confidential and of economic value. Therefore, the system is required to have a certain degree of security to ensure that important data is not easily stolen and damaged.

4.2.3 System Stability

Poor system stability implies that users may encounter data operation errors, excessively long page response times, or even inability to respond when using the system. Therefore, system stability is one of the important indicators of user evaluation of the system. The system should use stable operating systems, databases, middleware, etc., to ensure the stability of the system.

4.2.4 System Openness

System openness refers to the system having good compatibility. It can run normally in most versions of the Windows operating system. In addition, it also supports different browser versions and can run normally on commonly used browsers such as Microsoft's IE browser and Google Chrome. The easy upgrade and management of the system are also manifestations of system openness.

4.3 Feasibility Analysis

Feasibility analysis mainly refers to whether the development of the website system has the necessary conditions and resources under the current specific conditions, including the economic feasibility, technical feasibility analysis, and operational feasibility analysis of the website.

4.3.1 Economic Feasibility

First of all, it does not cost much to compile the program, there is a server online, and the school provides it. There are many JAVA reference books and database books in the school library, and the school's servers and computer rooms provide convenient conditions for the "mall" experiment. Suqian Information Port and Yi Network have provided support in hardware and website space. It provides convenience for successfully completing the thesis. Therefore, it is economically feasible.

4.3.2 Technical Feasibility

This system is based on Jsp+SSM+Tomcat+MySQL and adopts the B/S mode. Because JSP and SSM are powerful, and Tomcat and MySQL are flexible and easy to maintain, they are convenient and quick in development, and have the characteristics of flexible use and widespread practical applications. Therefore, using JSP, SSM, Tomcat, and MySQL is the best combination for developing lightweight platforms, thereby indicating that this system is technically feasible.

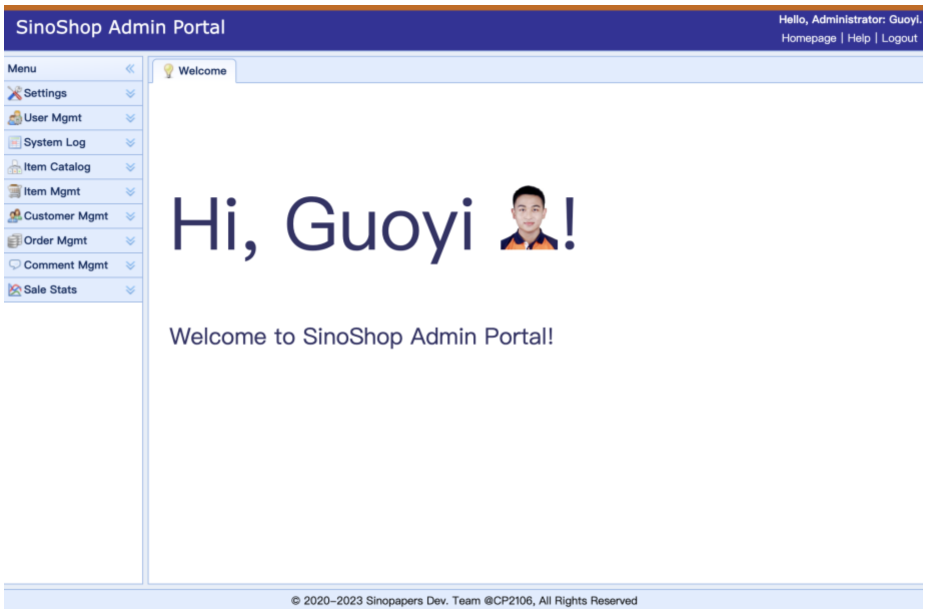
On the hardware side, with the rapid development of technology today, hardware updates are becoming faster and faster, capacities are getting larger, reliability is getting higher, and prices are getting lower, its hardware platform can fully meet the needs of this system.

4.3.3 Operational Feasibility

This system is based on the B/S architecture, which is very similar to the traditional BBS information release, users do not need to undergo special learning to achieve product purchases. As for system administrators, due to the availability of system management instructions, even system administrators who are not professional can operate the back-end management system quickly and conveniently.

5 Functionality

5.1 Background management function



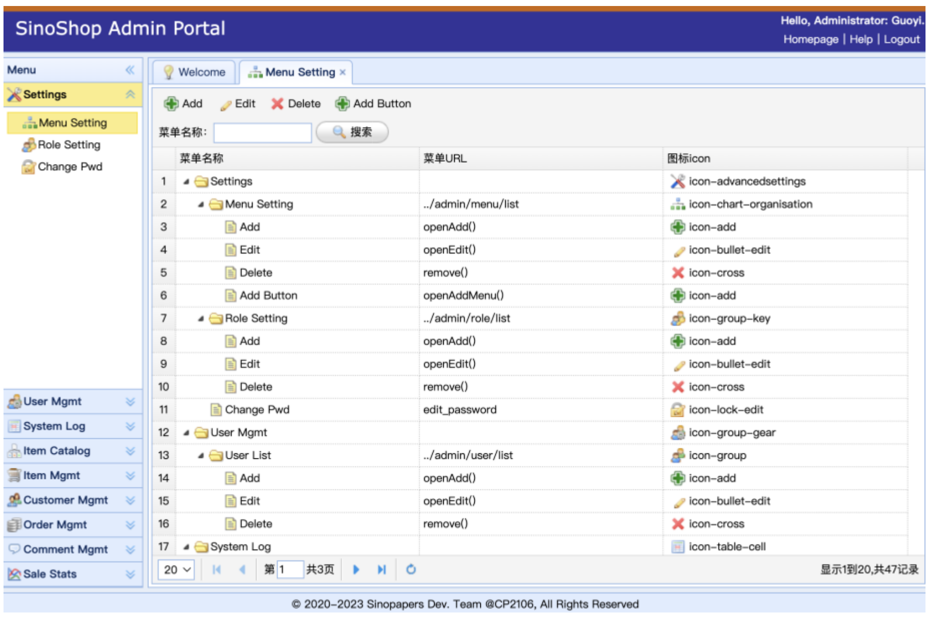
Hi, Guoyi !

Welcome to SinoShop Admin Portal!

© 2020–2023 Sinopapers Dev. Team @CP2106, All Rights Reserved

5.1.1 Commodity classification management

Commodity classification entity: id, classification name, parent classification, label (used to mark classification hierarchical relationship), remarks.



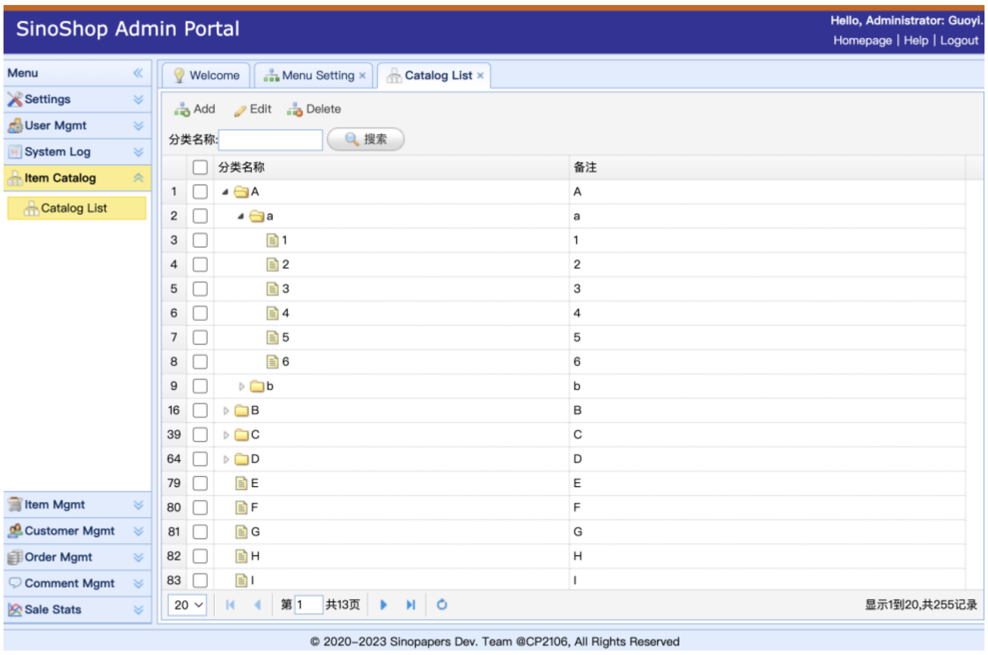
	菜单名称	菜单URL	图标icon
1	Settings		icon-advancedsettings
2	Menu Setting	../admin/menu/list	icon-chart-organisation
3	Add	openAdd()	icon-add
4	Edit	openEdit()	icon-bullet-edit
5	Delete	remove()	icon-cross
6	Add Button	openAddMenu()	icon-add
7	Role Setting	../admin/role/list	icon-group-key
8	Add	openAdd()	icon-add
9	Edit	openEdit()	icon-bullet-edit
10	Delete	remove()	icon-cross
11	Change Pwd	edit_password	icon-lock-edit
12	User Mgmt		icon-group-gear
13	User List	../admin/user/list	icon-group
14	Add	openAdd()	icon-add
15	Edit	openEdit()	icon-bullet-edit
16	Delete	remove()	icon-cross
17	System Log		icon-table-cell

20 第 1 共3页 显示1到20,共47记录

© 2020–2023 Sinopapers Dev. Team @CP2106, All Rights Reserved

5.1.2 Commodity management

Product entity: id, product category, product name, product main image, product price, inventory, sales volume, page views, comment volume, detailed description, and adding time.



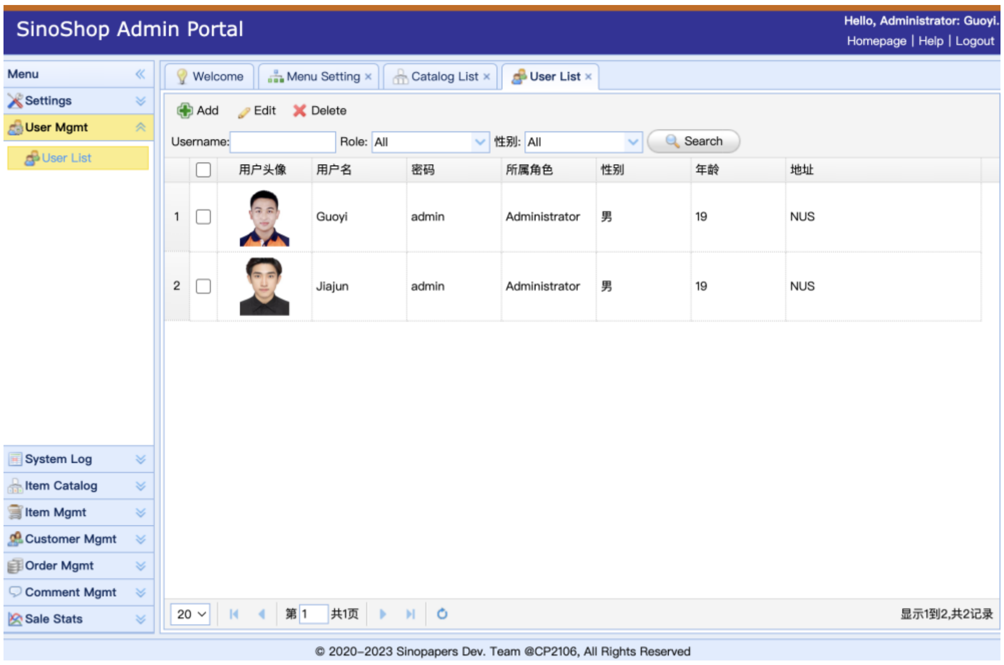
		分类名称	备注
1	☐	A	A
2	☐	a	a
3	☐	1	1
4	☐	2	2
5	☐	3	3
6	☐	4	4
7	☐	5	5
8	☐	6	6
9	☐	b	b
16	☐	B	B
39	☐	C	C
64	☐	D	D
79	☐	E	E
80	☐	F	F
81	☐	G	G
82	☐	H	H
83	☐	I	I

20 第1 共13页 显示1到20,共255记录

© 2020–2023 Sinopapers Dev. Team @CP2106, All Rights Reserved

5.1.3 User Management

User entities: id, username, password, email, real name, gender, registration time.



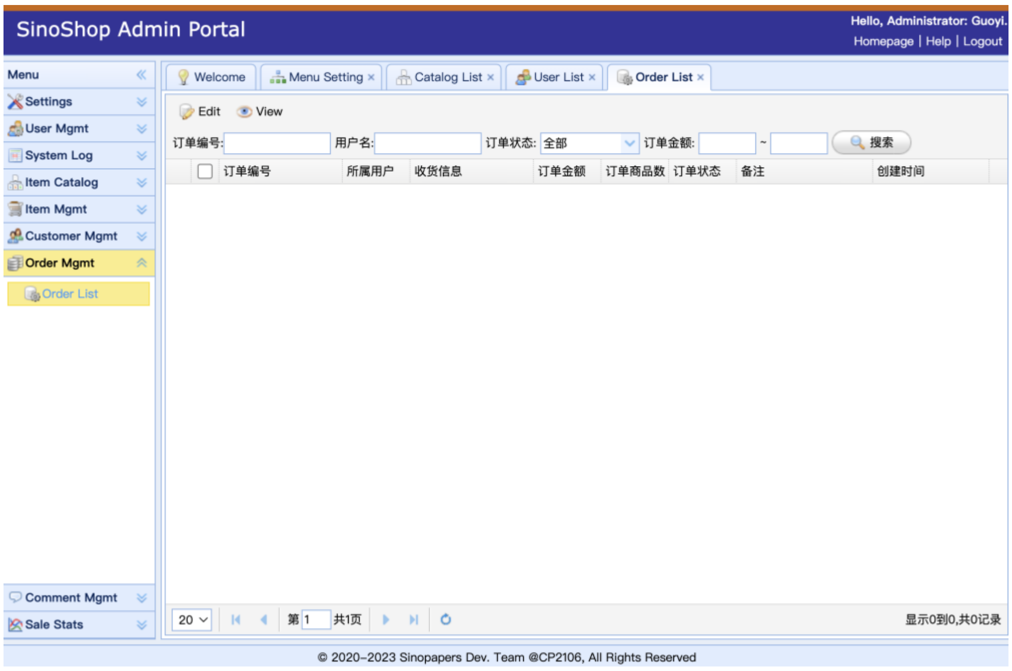
		用户头像	用户名	密码	所属角色	性别	年龄	地址
1	☐		Guoyi	admin	Administrator	男	19	NUS
2	☐		Jiajun	admin	Administrator	男	19	NUS

20 第1 共1页 显示1到2,共2记录

© 2020–2023 Sinopapers Dev. Team @CP2106, All Rights Reserved

5.1.4 Order Management

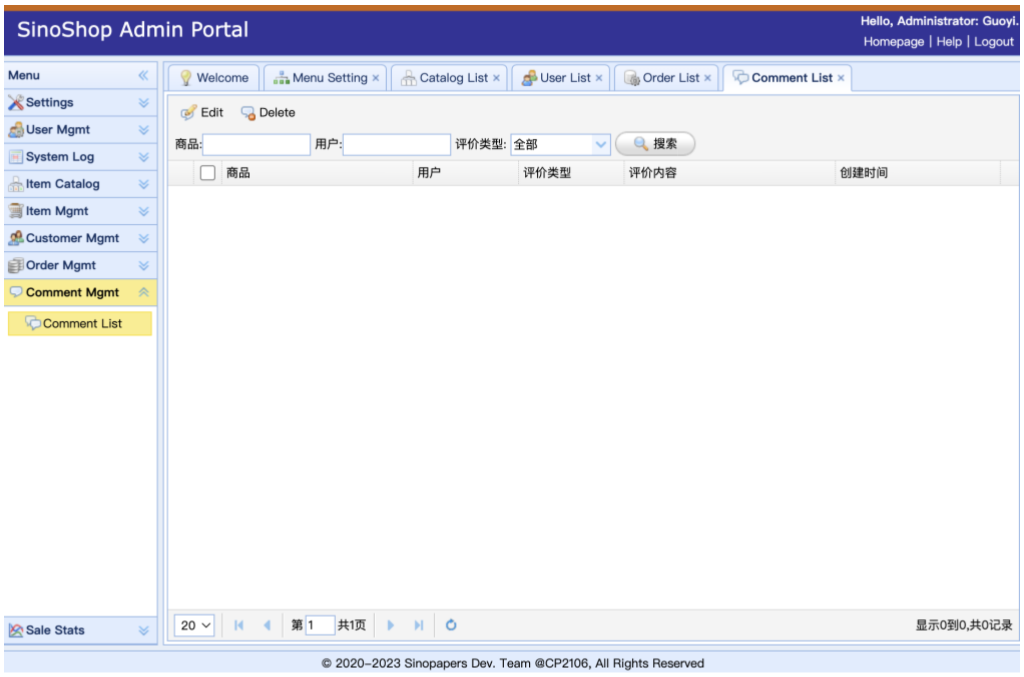
Order entity: id, order number, user, address, total order price, order item quantity, order notes, order creation time.
Order sub-items: id, order id, product id, product name, product price, quantity, total price.



© 2020–2023 Sinopapers Dev. Team @CP2106, All Rights Reserved

5.1.5 Comment Management

Comment entity: id, user, product, comment type, comment content, comment time.
Front desk functional requirements: Home product display, shopping cart
Entities: id, product id, user id, product name, product image, product price, product quantity, amount, and added time.



© 2020–2023 Sinopapers Dev. Team @CP2106, All Rights Reserved

5.2 Create a new project, import configuration, and run scaffolding.

5.2.1 Create a project

-Open Eclipse and select File > New > Project.
-In the pop-up dialog box, select the corresponding project type (eg Java project).
-Give the project a name and specify its storage location.
-Click the "Finish" button to complete the creation of the project.

5.2.2 Import configuration

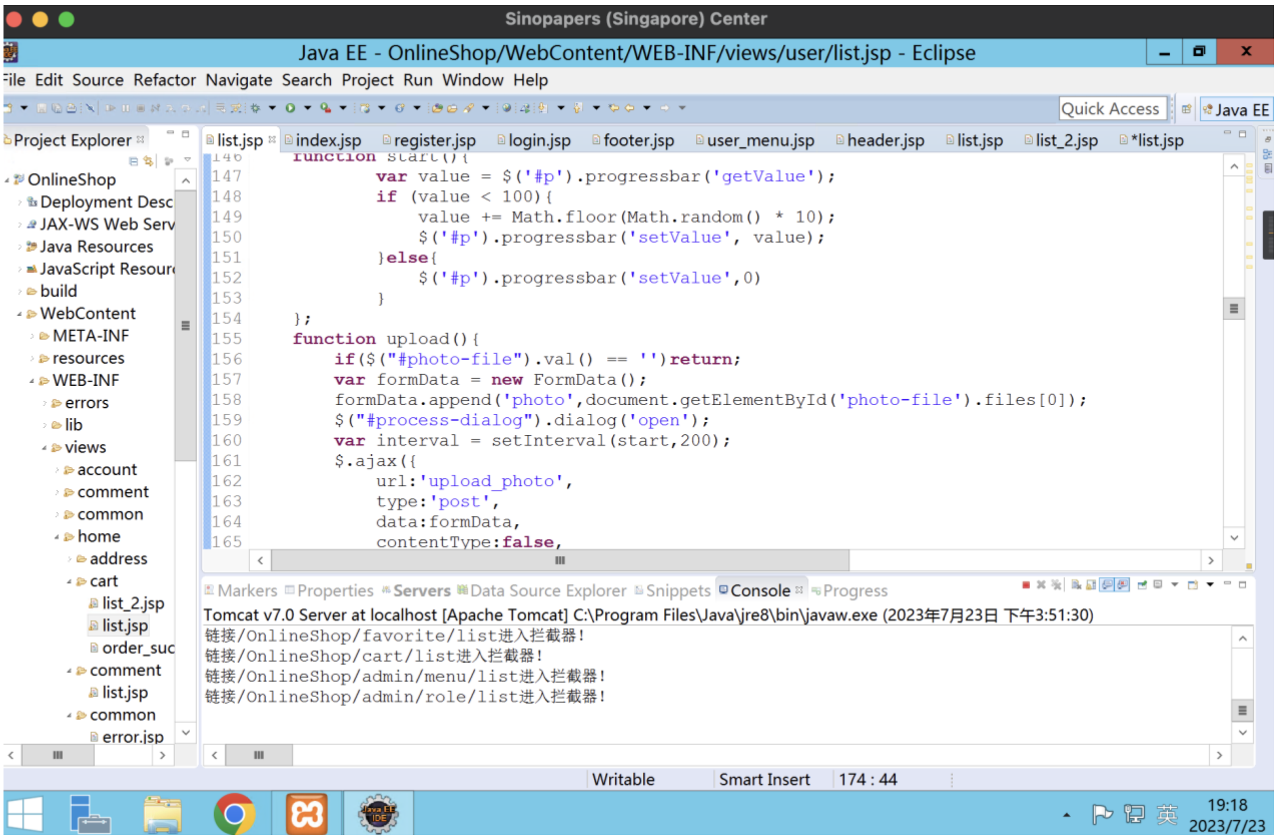
-In the created project, right click on the project name and select Properties.
-In the left panel, select related settings (such as Java Build Path, Server, Database, etc.).
-Make corresponding configurations as needed, such as setting JDK 1.8 as the Java runtime environment of the project.
-For the configuration of Tomcat and MySQL, additional operations are required, such as setting up the Tomcat server and connecting to the MySQL database. For specific operations, refer to the documentation or official guide of the corresponding tool.

5.2.3 Import scaffolding

-According to the scaffolding tool you choose, use the corresponding method to import scaffolding in the project.
-This may involve executing command line commands, using IDE plugins, or importing scaffolding related files directly.
-Refer to scaffolding's documentation and guides for how to properly import and configure scaffolding.

5.2.4 Run the scaffolding

-Once the scaffolding is successfully imported and configured, you can run the corresponding command or use the IDE plugin to start the scaffolding.
-Running scaffolding will generate basic code, framework or start the development server, depending on the specific scaffolding tools and project requirements.
-Refer to Scaffolding's documentation and guides on how to run and use Scaffolding properly.



```
    function start(){
            var value = $('#p').progressbar('getValue');
            if (value < 100){
                value += Math.floor(Math.random() * 10);
                $('#p').progressbar('setValue', value);
            }else{
                $('#p').progressbar('setValue',0)
            }
    };
    function upload(){
        if($("#photo-file").val() == '')return;
        var formData = new FormData();
        formData.append('photo',document.getElementById('photo-file').files[0]);
        $("#process-dialog").dialog('open');
        var interval = setInterval(start,200);
        $.ajax({
            url:'upload_photo',
            type:'post',
            data:formData,
            contentType:false,
```

```
Tomcat v7.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre8\bin\javaw.exe (2023年7月23日 下午3:51:30)
链接/OnlineShop/favorite/list进入拦截器!
链接/OnlineShop/cart/list进入拦截器!
链接/OnlineShop/admin/menu/list进入拦截器!
链接/OnlineShop/admin/role/list进入拦截器!
```

5.3 Product classification management function design and implementation

5.3.1 Design the database table structure

First, design an appropriate table structure in the MySQL database to store information related to commodity classification. You can create a table named "categories", which may contain fields as follows:

-id:	category ID (primary key)
-name:	category name
-parent_id:	Parent category ID (used to represent the hierarchical relationship of categories)

5.3.2 Create a model

In the MVC framework, the model is responsible for interacting with the database. Create a model class called "Category" that corresponds to the "categories" table in the database. The model class should contain the following methods:

-getCategories():	Get all product categories from the database.
-getCategoryById(id):	Get the specified category from the database according to the

	category ID.
-createCategory(data):	Create a new product category.
-updateCategory(id, data):	Update the information of the specified category.
-deleteCategory(id):	Delete the specified category.

5.3.3 Create a controller

The controller is responsible for processing user requests and calling the corresponding model methods. Create a controller class called "CategoryController", which should contain the following action methods:

-index():	Display a list of all product categories.
-show(id):	Display detailed information of a single product category.
-create():	Display the form page for creating product categories.
-store():	Process the form data submitted by the user to create product categories.
-edit(id):	Display the form page for editing product categories.
-update(id):	Process the form data submitted by the user to edit the product category.
-destroy(id):	Delete the specified product category.

5.3.4 Create a view

The view is responsible for displaying data and receiving user input. Create appropriate view files for displaying category listings, form pages, and details. The following view files should be included:

-index.html:	The page that displays the list of product categories.
-show.html:	A page showing detailed information of a single product category.
-create.html:	Form page for creating product categories.
-edit.html:	The form page for editing product categories.

5.3.5 Configure routing

In the MVC framework, routing is used to map requests to corresponding controller actions. In the routing configuration file, define appropriate routing rules so that user requests can correctly access the corresponding controller actions.

5.4 Implement business logic

According to specific requirements, implement the corresponding business logic in the action method of the controller. For example, when creating a product category, get data from the form, and call the createCategory method in the model to store the data in the database. When updating and deleting product categories, similarly call the corresponding model method to realize the corresponding function.

5.5 Commodity management function design and implementation

5.5.1 Design product table structure

First, design an appropriate table structure in the database to store product-related information. You can create a table named "products" which may contain the following fields:

-id:	product ID (primary key)
-name:	product name
-description:	product description
-price:	commodity price
-category_id:	category ID (used to indicate the category to which the product belongs)
-created_at:	creation time
-updated_at:	updated time

5.5.2 Create a product model

Create a model class named "Product" that corresponds to the "products" table in the database. The model class should contain the following methods:

-getAllProducts():	Get all products from the database.
-getProductById(id):	Get the specified product from the database according to the product ID.
-createProduct(data):	Create a new product.
-updateProduct(id, data):	Update the information of the specified product.
-deleteProduct(id):	Delete the specified product.

5.5.3 Create a Product Controller

Create a controller class called "ProductController" that is responsible for handling product-related requests. It should contain the following action methods:

-index():	Display a list of all products.
-show(id):	Show detailed information of a single item.
-create():	Display the form page for creating a product.
-store():	Process the form data submitted by the user to create a product.
-edit(id):	Display the form page for editing items.
-update(id):	Process the form data submitted by the user to edit the product.
-destroy(id):	Delete the specified product.

5.5.4 Create Product Views

Create appropriate view files for displaying product listings, form pages, and details. It should contain the following view files:

-index.html:	The page that displays the product list.
-show.html:	A page showing detailed information of a single product.
-create.html:	Form page for creating products.
-edit.html:	The form page for editing products.

5.6 Configure product routing

In the routing configuration file, define appropriate routing rules to map product-related requests to corresponding controller actions.

5.7 Product information multi-condition search function

5.7.1 Define product search form

Create a form for entering search criteria, which can contain multiple input fields, such as product name, price range, category, etc. These fields can be defined using HTML form elements, such as text boxes, drop-down lists, etc.

5.7.2 Create product search view

Create a view file called "search.html" that renders the product search form. The view file should contain a form element where the user can enter search criteria and trigger the search action via a submit button.

5.7.3 Create a product search route

In the routing configuration file, define a routing rule for processing product search requests. Map that route to an appropriate controller action, such as "ProductController@search".

5.7.4 Create a search method in the product controller

Create an action method called "search" in the "ProductController" in the product controller. This method is responsible for receiving the search conditions submitted by the user and querying according to the conditions. The specific implementation can define query statements according to the language and framework used, such as using SQL statements to construct queries or using ORM methods to query

5.7.5 rocess search conditions

In the search method, construct a query statement based on the search conditions submitted by the user.

According to the presence or absence of different conditions, construct the appropriate query statement, and pass the query result to the result view for presentation.

5.7.6 Create a search result view

Create a view file named "searchResults.html" to display the product search results obtained according to the search criteria. The view file should display relevant information of the product as required, such as product name, price, picture, etc. A looping structure can be used to traverse the result set in the view and present the data to the user.

5.7.7 Return search results

In the search method, pass the query results to the search result view, and call the result view in the controller for display.

5.8 Customer information management function design and implementation

5.8.1 Database Design

-First, design the data table of customer information, and determine the fields that need to be saved, such as customer name, contact information, address, etc. Create corresponding tables and define field types and constraints.
-As needed, foreign keys can be added to the customer information table to associate with other tables, such as the order table or product table.

5.8.2 Create a model

-In the model layer of the MVC framework, create a customer information model (Customer Model) to define the structure and behavior of customer information.
- In this model, create attributes (such as customer name, contact information, etc.) and methods (such as adding customers, deleting customers, updating customer information, etc.) to operate on customer information.

5.8.3 Create a controller

-In the controller layer of the MVC framework, create a customer information controller (Customer Controller) for receiving and processing requests from users, and calling corresponding models and views.
- In this controller, implement the following functions:
-Display customer list: Receive user request, get customer list data from the model, and pass it to the relevant view for display.

-Add customer: Receive the form data submitted by the user, call the method of the model to add the customer, and then return the result to the corresponding view and prompt the user.
-Edit customer: Receive the customer ID selected by the user to edit, obtain the corresponding customer data, and pass it to the edit view for display. Receive the form data submitted after editing, call the method of the model to update the customer information, and return the result to the corresponding view and prompt the user.
-Delete customer: Receive the customer ID to be deleted selected by the user, call the method of the model to perform the delete operation, and return the result to the corresponding view and prompt the user.

5.8.4 Create a view

- In the view layer of the MVC framework, create related view files for customer information management, such as views for displaying customer lists, views for adding customers, views for editing customers, etc.
-Each view file is responsible for displaying the corresponding page, interacting with the controller, receiving user input and passing it to the controller for processing.

5.8.5 Set up routing

-In the routing configuration file of the MVC framework, add corresponding routing rules for the customer information management function, and route the request to the corresponding controller and action method.

5.9 Order management function design and implementation

5.9.1 Database Design

-First, design the data table of order information, and determine the fields that need to be saved, such as order number, customer information, product information, order status, payment information, etc. Create corresponding tables and define field types and constraints.
-As needed, you can add foreign keys to the order information table to associate with other tables, for example, to associate with the customer information table or product information table.

5.9.2 Create a model

-In the model layer of the MVC framework, create an order model (Order Model) to define the structure and behavior of order information.
-In this model, create attributes (such as order number, customer information, product information, order status, payment information, etc.) and methods (such as create order, cancel order, update order status, etc.) to operate on order information.

5.9.3 Create a controller

- In the controller layer of the MVC framework, create an order controller (Order Controller) for receiving and processing requests from users, and calling corresponding models and views.
- In this controller, implement the following functions:

-Display order list: Receive user request, get order list data from the model, and pass it to related views for display.

- Create order:	Receive the form data submitted by the user, call the method of the model to create the order, and then return the result to the corresponding view and prompt the user.
- Cancel order:	Receive the order ID selected by the user to cancel, call the method of the model to cancel the order, and return the result to the corresponding view and prompt the user.
-Update order status:	Receive the order ID and new status value selected by the user to update the status, call the method of the model to update the order status, and return the result to the corresponding view and prompt the user.

5.9.4 Create a view

-In the view layer of the MVC framework, create related view files for order management, such as views for displaying order lists, views for creating orders, views for canceling orders, etc.
-Each view file is responsible for displaying the corresponding page, interacting with the controller, receiving user input and passing it to the controller for processing.

5.9.5 Set up routing

-In the routing configuration file of the MVC framework, add corresponding routing rules for the order management function, and route the request to the corresponding controller and action method.

In addition to the above basic functions, the order needs to contain many factors, and other functions can be added according to specific needs, such as:

-Search order:	You can search for orders by keywords such as order number, customer name, and product name.
-Export orders:	Orders can be exported in Excel, PDF and other formats for easy saving and printing.
-Order details:	Click an order in the order list to view the order details, including product details, payment status, etc.
-Timing task:	If you need to process orders or notify at a specific time, you can use timing tasks to automate operations.

5.10 Product review management function design and implementation

5.10.1 Model design

-Create a comment (Comment) model, which contains necessary fields, such as comment ID, product ID, user ID, comment content, comment time, etc.
- Define the necessary methods in the review model, such as saving reviews, getting a list of reviews, getting reviews of a product, etc.

5.10.2 View design

- Create a view page for displaying a list of comments, which can be implemented using HTML, CSS and JavaScript.
-Design corresponding interface elements according to requirements, such as comment list, comment form, etc.

5.10.3 Controller design

-Create a controller (CommentController) with methods to handle comment related requests.
-Define action methods in the controller that handle requests to get a list of comments, save comments, etc.

5.10.4 Routing settings

-In the routing configuration file, add corresponding routing rules for the comment management function, and map the request to the corresponding controller and method.

5.10.5 Database Design:

-Create a database table to store the comment data, create corresponding columns based on the fields defined in the comment model.
-Use the database connection tool to connect to the database, and write code in the controller for database operations, such as saving comments, querying comments, etc.

5.10.6 Front-end and back-end interaction:

- In the comment list view, use Ajax or other front-end and back-end interaction methods to send a request to the server to obtain the comment list and save comments.
-In the controller, receive and process the request sent by the front end, perform data manipulation as needed, and return the corresponding result.

5.10.7 Security considerations

-Reasonable input validation and filtering of comments submitted by users to prevent security issues such as XSS (cross-site scripting attacks).
- Perform permission control on comment operations to ensure that only legitimate users can perform comment operations.

5.11 user registration and login function design and implementation

5.11.1 Model

-Create a database table to store user information. This table should contain at least the necessary fields like username and password.
-Create a model class that interacts with the database. This model class should have methods to perform query, insert and update user information operations.

5.11.2 View

- Create an interface for user login (login form), with input fields for username and password, and a login button.
- Create an interface for user registration (registration form), with input fields for username, password and confirm password, and a register button.
-Write corresponding client-side validation code for each form to ensure the validity of the input data.

5.11.3 Controller

-Create a controller class responsible for handling user login and registration requests.
-In the login request, the controller should receive the user name and password submitted by the user, verify whether it exists in the database, and perform corresponding operations according to the verification results (such as redirecting to the home page if the login is successful, and returning an error message if the login fails).
- In the registration request, the controller should receive the username and password submitted by the user, perform necessary validation (for example, check if the username already exists), and then save the user information to the database.

5.12 Home product display function design and implementation

5.12.1 Model

-Create a database table to store product information. This table should contain attributes such as product name, description, price, stock, etc.
- Create a model class that interacts with the database. This model class should have methods to perform the operation of querying product information.

5.12.2 View

-Create a view for displaying the home page. This view should contain an area for product display, which can be in the form of a list or a card.
-Use a template engine or front-end framework in the view to dynamically render product information.

5.12.3 Controller

-Create a controller class responsible for handling the request for the home page.
-Call the method of the model in the controller to get product information.
- Pass the obtained commodity information to the view.

In this process, the controller acts as a middleman, handling the request of the home page and interacting with the model and view. The model is responsible for interacting with the database to obtain product information. The controller passes the acquired product information to the view for display.

In the view, you can use HTML, CSS, and JavaScript to design the appearance and interaction of product displays. Rendering product information dynamically is easier with a templating engine or front-end framework.

In addition, you can also add some logic in the controller, such as filtering products according to specific conditions, paging, etc.

To sum up, the design and implementation of the product display function on the home page requires the collaboration of the three components of the model, view and controller under the MVC framework. The model is responsible for data storage and query, the view is responsible for displaying product information, and the controller is responsible for processing requests and coordinating data transfer between the model and view. In this way, a flexible, maintainable and easy-to-extend home page product display system can be realized.

5.13 product list search page design and implementation

5.13.1 Model layer

-Define the data structure and attributes of the item, such as item name, price, description, etc.

-Realize operation methods such as addition, deletion, modification and query of product data, such as obtaining product lists, searching for products based on keywords, etc.

5.13.2 View layer

-Create an HTML page to display product list and search function.

- Use HTML and CSS to design the layout and style of the page, including the search box, the display of the product list, etc.

5.13.3 Controller layer

-Create a JavaScript file to handle user interaction events and data acquisition and display.

-In the controller, by listening to the user's search button click event, get the keyword in the input box, and call the method in the model layer to search for the product.

- Dynamically generate HTML from the search results and insert them into the product list.

5.13.4 CSS styling

-Create a CSS file that will be used to style the product listing search page.

- According to your own needs, design the layout, color, font and other styles of each element.

5.14 Product Details Search Page design and implementation

5.14.1 Model design

-Define a product model, including product attributes, such as product name, price, description, picture, etc.

-Consider the mapping of commodity models and databases, you can use object-relational mapping (ORM) tools to simplify operations, such as Django's ORM or Entity Framework.

5.14.2 Controller implementation

-Create a controller for the product details page, which is responsible for handling user requests and returning responses.

-Define a route (URL) in the controller to display the product detail page.

-When the user requests the product details page, the controller will obtain the product data according

to the request parameters, and pass the data to the view for rendering.
-The controller can call the method or query statement of the model layer to obtain the product data from the database.

5.14.3 Business logic

-According to requirements, some additional business logic may need to be implemented, such as:
-Verify user permissions to ensure that only authorized users can view product details.
- Handle user interactions, such as saving items, adding to cart, etc.

5.14.5 Data Display

-Use the product data passed by the model in the view to display various information of the product, such as name, price, description and picture.
- HTML and CSS can be used to create suitable layouts and styles to enhance user experience.
-For pictures, you can use the picture URL as an attribute to load and display pictures through the <img> tag of HTML.

5.14.6 View design

-Create an HTML template for a product detail page.
-Define appropriate tags and styles in the template, including the display of product name, price, description, pictures and other information.
-Consider using a template engine, such as Django's template engine (Django Template) or Razor view engine.

5.15 Shopping Cart Management Function

5.15.1 Model design

-Define a product model, including product attributes, such as product name, price, description, picture, etc.
-Consider the mapping of commodity models and databases, you can use object-relational mapping (ORM) tools to simplify operations, such as Django's ORM or Entity Framework.

5.15.2 Controller implementation

-Create a controller for the product details page, which is responsible for handling user requests and returning responses.
-Define a route (URL) in the controller to display the product detail page.

-When the user requests the product details page, the controller will obtain the product data according to the request parameters, and pass the data to the view for rendering.
-The controller can call the method or query statement of the model layer to obtain the product data from the database.

5.15.3 Business logic

-According to requirements, some additional business logic may need to be implemented, such as:
-Verify user permissions to ensure that only authorized users can view product details.
- Handle user interactions, such as saving items, adding to cart, etc.

5.15.4 Data Display

-Use the product data passed by the model in the view to display various information of the product, such as name, price, description and picture.
- HTML and CSS can be used to create suitable layouts and styles to enhance user experience.
-For pictures, you can use the picture URL as an attribute to load and display pictures through the <img> tag of HTML.

5.15.5 View design

-Create an HTML template for a product detail page.
- Define appropriate tags and styles in the template, including the display of product name, price, description, pictures and other information.
- Consider using a template engine, such as Django's template engine (Django Template) or Razor view engine.

5.16 pre-order process of shopping cart list design and implementation

5.16.1 Model design

- Define a shopping cart model, which can use a list or collection to store the items purchased by the user.
- The shopping cart model can contain the information of the product, such as product ID, name, price, quantity, etc.
- You can consider using a database to store the shopping cart data, or you can save the shopping cart data in the session (session).

5.16.2 View design

-Create an HTML template for the shopping cart page, which is used to display the items in the shopping cart and calculate the total price and other information.
- Define appropriate markup and styles in the template to display product information in the shopping cart.
- Provide necessary interaction methods, such as modifying quantity, deleting products and other operations.

5.16.3 Controller implementation

-Create a shopping cart controller that is responsible for handling shopping cart related user requests and returning responses.
- Define the routing (URL) related to the shopping cart function in the controller, such as adding items to the shopping cart, updating the number of items in the shopping cart, deleting items in the shopping cart, etc.
-When the user performs shopping cart-related operations, the controller will call the method or logic of the model layer to process the data and state changes accordingly

5.16.4 Business logic

-Implement relevant business logic of shopping cart management, for example:
- Add product to shopping cart operation: Get product data from the database and add product items to the shopping cart model.
-Update the shopping cart product quantity operation: receive the quantity value submitted by the user, and update the quantity of the corresponding product item in the shopping cart model.
-Delete item in the shopping cart operation: remove the corresponding item from the shopping cart model according to the ID of the item submitted by the user.

5.17 Data display and calculation:

-Use the data passed by the shopping cart model in the view to display the information of the items in the shopping cart, such as name, price, quantity, etc.
- For operations such as modification and deletion of the quantity of goods, the corresponding event processing is triggered through the routing related to the controller.
-In the shopping cart page, when calculating the total price, you can traverse the item items in the shopping cart model, and accumulate the price and quantity of each item item.

5.18 Collection Function design and implementation

5.18.1 Model Design

-Define a model to represent the collection data and its attributes.
-Consider the mapping between the collection model and the database using an ORM tool.
-Determine the relationships between the collection model and other relevant models, such as users or items.

5.18.2 View Design

-Create an HTML template to display the collection data.
-Define appropriate HTML tags and styles in the template to present the collection information.
-Consider implementing pagination or filtering options if the collection is expected to have a large number of items.

5.18.3 Controller Implementation

-Create a controller responsible for handling the collection-related actions.
-Define appropriate routes (URLs) in the controller for the collection functionality.
-Implement methods in the controller to retrieve and manipulate collection data.
-The controller should interact with the model layer to perform necessary CRUD (Create, Read, Update, Delete) operations on the collection data.

5.18.4 Collection Functionality

-Determine the specific functionality of the collection, such as adding items, removing items, or favoriting items.
-Implement methods in the controller to support these collection actions.
-For example, you might have methods like “addItemToCollection,” “removeItemFromCollection,” or “toggleFavoriteStatus” in the controller.
-These methods should interact with the corresponding model layer methods to update the collection data accordingly.

5.19 Displaying the Collection

-In the view template, render the collection data fetched from the model layer.
-Use appropriate HTML tags and styles to present the collection information, such as a list or grid view.
-If required, provide user interactions like buttons or links to perform collection actions, such as adding or removing items.
-Utilize the controller methods to handle these user interactions and update the collection data.
User Authentication and Authorization:
-Consider implementing user authentication and authorization to ensure that only authorized users can access and modify collections.
-Define appropriate authentication and authorization mechanisms in the controller, such as checking user roles or permissions before allowing collection actions.

5.20 Delivery address management design and implementation

5.20.1 Model Design

-Define a model to represent the harvesting address data.
-Decide on the attributes needed for the harvesting address, such as address line, city, state, postal code, etc.
-Consider any relationships with other models, such as a user model if the harvesting address is associated with a user.

5.20.2 View Design

-Create a form in the view to allow users to input their harvesting address details.
-Design the form elements (e.g., input fields, dropdowns) to capture the address information.
-Include appropriate validations, such as required fields or format restrictions, to ensure data integrity.
Controller Implementation:
-Create a controller responsible for handling harvesting address-related actions.
-Define routes (URLs) in the controller for the harvesting address functionality.
-Implement methods in the controller to handle creating, updating, retrieving, and deleting harvesting addresses.
-These methods should interact with the harvesting address model to perform the necessary CRUD operations.

5.21 Submit order management function design and implementation

5.21.1 Model Design

-Define a model to represent the order data, including any additional attributes specific to the "hand-in" functionality.
-Consider attributes such as order ID, customer ID, order date, products, quantities, total price, delivery details, and any other relevant information.
-Determine if you need to establish relationships with other models, such as a customer model or a product model.

5.21.2 View Design

-Create views that will display and interact with the "hand-in" order-related information.
-Design views to include order listing, order details, order creation, order editing, and order status updates specifically for the "hand-in" process.

-Determine the specific information and forms required for the “hand-in” functionality, such as delivery scheduling, delivery address, or any other special requirements.
Controller Implementation:
-Create a controller responsible for handling “hand-in” order-related actions.
-Define routes (URLs) in the controller for the “hand-in” order management functionality.
-Implement methods in the controller to handle creating, updating, retrieving, and deleting “hand-in” orders.
-These methods should interact with the order model to perform the necessary CRUD operations specific to the “hand-in” process.

5.22 member center function design and implementation

5.22.1 Identify the Requirements

-Define the goals and objectives of the member center.
-Determine the features and functionalities required, such as user registration, login, profile management, password reset, etc.

5.22.2 Design the Data Model

-Identify the data entities or objects involved, such as User, Profile, etc.
-Define the relationships between these entities, such as one-to-one, one-to-many, etc.
-Design the database schema or data model to represent these entities and relationships.
Implement the Model:
-Create the necessary data models or classes to represent the entities in your chosen programming language.
-Define the attributes and methods required for each model.
-Implement validation and business logic within the models, such as password hashing, email verification, etc.
-Configure the database connection and handle CRUD (Create, Read, Update, Delete) operations.

5.22.3 Design the Views

-Identify the screens or pages required for the member center, such as registration form, login page, profile page, etc.
-Design the UI/UX for these views, including forms, buttons, input fields, etc.
-Use HTML, CSS, and any frontend frameworks like Bootstrap to create the views.
-Ensure a consistent design and layout across the views.

5.22.4 Implement the Controller

-Create controller classes or functions to handle user actions and interaction with the model and views.

-Implement routing or URL mappings in the framework to map requests to the appropriate controller methods.
-Write code to handle user registration, login, profile updates, password reset, etc.
-Validate user input, handle errors, and provide appropriate responses.

5.22.4 Connect the Components

-Configure the MVC framework to connect the models, views, and controllers.
-Ensure the models are accessible within the controllers and views.
-Retrieve and manipulate data using the model within the controllers.
-Display the retrieved data from models in the appropriate views.

5.22.5 Test and Refine

-Perform unit testing to identify and fix any issues with your code.
-Test the member center function end-to-end to ensure all features work as expected.
-Gather feedback from users and make any necessary refinements or improvements.

5.22.6 Security Considerations

-Implement secure practices, such as password hashing, encryption, and secure session management.
-Implement mechanisms to prevent common security vulnerabilities like SQL injections, cross-site scripting (XSS), cross-site request forgery (CSRF), etc.

5.23 review delete function design and implementation

5.23.1 Identify the Requirements

-Determine the specific requirements for the evaluation deletion function.
-Consider factors such as user roles, authorization, and access control.

5.23.2 Design the Data Model

-Identify the data entities or objects involved in evaluations, such as Evaluation, User, etc.
-Define the relationships between these entities, such as one-to-one or one-to-many with User entity.
-Review your existing data model and make any necessary modifications to accommodate the evaluation deletion functionality.

5.23.3 Implement the Model

-Create the necessary data models or classes to represent the Evaluation entity.
-Define the attributes and methods required for the Evaluation model.

-Implement any relevant validation and business logic for evaluation deletion, such as checking if the current user has the necessary permissions to delete an evaluation.

5.23.4 Design the Views

-Identify the views or interfaces associated with evaluation management, such as a list of evaluations or a detailed evaluation view.
-Determine how the deletion function will be represented in the user interface, such as an icon/button, a confirmation dialog, or a dedicated delete page.
-Design the UI elements and interactions to allow users to perform the evaluation deletion.

5.23.5 Implement the Controller

-Create controller classes or functions to handle user actions related to evaluation deletion.
-Implement routing or URL mappings to map the appropriate requests to the corresponding controller methods.
-Write code to handle evaluation deletion, including fetching the necessary data, verifying user permissions, and executing the deletion operation.
-Handle errors and provide appropriate responses to the user.

5.23.6 Connect the Components

-Configure the MVC framework to connect the models, views, and controllers.
-Ensure that the necessary models are accessible within the controllers and views.
-Connect the evaluation deletion function to the relevant views and controller methods.

5.23.7 Test and Refine

-Perform unit testing to ensure the evaluation deletion function works as expected.
-Test the function in conjunction with other related features to identify any potential issues.
-Gather feedback from users to improve the usability and effectiveness of the evaluation deletion function.

5.23.8 Security Considerations

-Implement proper authorization checks to ensure that only authorized users can delete evaluations.
-Validate user input to prevent any security vulnerabilities such as SQL injections or cross-site scripting (XSS).

5.24 statistical function design and implementation

5.24.1 Identify Statistical Requirements

-Determine the specific statistical functions and calculations needed, such as mean, median, standard deviation, etc.
-Consider the input data requirements, such as data types, formats, and sources.
Implement the Model:
-Create the necessary data models or classes to represent the statistical calculations.
-Implement the statistical functions within the model layer.
-Define methods or functions for each statistical calculation required.

5.24.2 Design the Views

-Identify the views or interfaces where the statistical results will be presented.
-Determine the presentation format for the statistical data, such as tables, charts, or textual representations.
-Design the UI elements, visualizations, and interactions to display the statistical results effectively.
Implement the Controller:
-Create controller classes or functions to handle user requests related to statistical calculations.
-Implement routing or URL mappings to map the appropriate requests to the corresponding controller methods.
-Retrieve the necessary data from the model for the statistical calculations.
-Call the appropriate statistical functions from the model, passing the required data as arguments.
-Prepare the data for presentation in the views.

5.24.3 Connect the Components

-Configure the MVC framework to connect the models, views, and controllers.
-Ensure the models are accessible within the controllers and views.
-Connect the statistical functions to the relevant views and controller methods to pass the calculated data.

5.24.4 Test and Refine

-Perform unit testing on the statistical functions to ensure they produce accurate results.
-Test the integration of the statistical functions with the views and controllers.
-Validate the statistical calculations against sample data or existing statistical benchmarks.
-Refine and optimize the calculations as needed based on test results.

5.24.5 Security Considerations

-Validate and sanitize user inputs to prevent security vulnerabilities like SQL injections or cross-site scripting (XSS).
-Implement appropriate access controls and permissions to limit access to the statistical functions and data.

5.25 Harvesting Address Functionality

-Determine the specific functionality required for the harvesting address, such as adding/updating the address, retrieving the address, or deleting the address.
-Implement corresponding methods in the controller.
-For example, you might have methods like “createHarvestingAddress,” “updateHarvestingAddress,” “getHarvestingAddress,” or “deleteHarvestingAddress.”

5.25.1 Displaying and Editing the Harvesting Address

-In the view, render the harvesting address form for users to enter/update their address details.
-Pre-fill the form with the user’s existing harvesting address if available.
-Utilize the controller methods to handle form submissions and update the harvesting address data accordingly.

5.25.2 User Authentication and Authorization

-Consider implementing user authentication and authorization to ensure only authorized users can access and modify their harvesting address.
-Define appropriate authentication and authorization mechanisms in the controller, such as checking user roles or permissions before allowing address-related actions.

5.26 Hand-in Order Management Functionality

-Determine the specific functionality required for managing “hand-in” orders.
-Implement corresponding methods in the controller to handle these functionalities, such as creating new “hand-in” orders, updating delivery details, retrieving “hand-in” orders, or canceling “hand-in” orders.
-Consider additional features for the “hand-in” functionality, such as tracking order delivery status or providing delivery notifications.

5.26.1 Displaying and Managing “Hand-in” Orders

-In the views, render order listing pages specifically for “hand-in” orders, with relevant details such as delivery status, delivery address, and scheduled delivery date.
-Implement features to update delivery details, including address changes or rescheduling delivery.
-Provide options for canceling “hand-in” orders, if applicable.

5.26.2 User Authentication and Authorization

-Consider implementing user authentication and authorization to ensure only authorized users can access and manage “hand-in” orders.
-Define appropriate authentication and authorization mechanisms in the controller, such as checking user roles or permissions before allowing “hand-in” order-related actions.

6 overall test function design and implementation

6.1 Unit Testing

-Write unit tests for each individual component of the MVC architecture, including the models, views, and controllers.
-Ensure that the models are tested for correctness of data handling, calculations, and any other functions they provide.
-Test the views by verifying that they correctly display the information retrieved from the models.
-Test the controllers by mocking or simulating user interactions and asserting that the expected actions are performed.

6.2 Integration Testing

-Perform integration testing to validate the interaction and communication between the models, views, and controllers.
-Test the flow of data between the components and ensure that it is accurate and consistent.
-Verify that the controllers correctly handle user inputs and invoke the appropriate actions in the models and views.
-Test scenarios where multiple components are involved, such as submitting a form or navigating through different views.

6.3 End-to-End Testing

-Conduct end-to-end testing to ensure the overall function of the application.
-Test the complete user workflow, from initiating an action to receiving the expected output.
-Use automated testing frameworks or tools to simulate user interactions and verify the correct behavior of the application.
-Validate that the input data is processed correctly by the models, displayed accurately in the views, and respond appropriately to user interactions.

6.4 Test Data and Edge Cases

-Include tests with various types of input data, covering both normal and boundary cases.
-Test scenarios with empty or invalid data to ensure proper error handling and validation.
-Verify that the statistical functions handle special cases, such as empty datasets or rare edge cases, gracefully and produce correct results.

6.5 Performance Testing

-Assess the performance of the overall function under different load conditions.

-Measure the response time of the application and evaluate its scalability.
-Stress test the statistical functions by providing large datasets and ensuring they perform within acceptable limits.

6.6 Usability Testing

-Conduct usability testing to evaluate how users interact with the statistical features.
-Observe users performing tasks related to the statistical functions and gather feedback on their experience.
-Validate that the user interface is intuitive, the data presentation is clear, and the interactions are easy to understand.