澳門理工大學
Universidade Politécnica de Macau
Macao Polytechnic University

Faculty of Applied Sciences
Bachelor of Science in Computing

COMP321 Information System Implementation Final Report

Academic Year 2022/23
2nd semester

Mobile Online Shopping Mall

Project number:	Group 6
Team members:	Grant Jing (P2010411)
	Veronica Chen (P2010562)
	Polo Liu (P2010556)
Supervisor:	Mr. Philip
Assessor:	Ms. Calana
Submission Date:	Apr 24, 2023

Table of Contents

1 Introduction 5

1.1 Overview 5

1.2. Objectives 5

2 Background and Related Work 6

2.1 Background 6

2.2 Related Work 7

3 System Design 10

3.1 Data Modelling 10

3.2 Dynamic Modelling 13

3.2.1 State Diagrams 13

3.2.2 Activity Diagrams 14

3.2.3 Sequence Diagram 17

4 System Implementation 19

4.1 Architecture 19

4.1.1 MVC Structure 20

4.1.2 Front-end 21

4.1.3 Back-end 25

4.2 REM Adaptation Solution for Mobile Screen 26

4.3 <keep-alive> Cache Components Data 27

4.4 UI and UX Design 30

4.4.1 Vant Component Library 30

4.4.2 Visual Hierarchy in UX 31

4.4.3 Gulf of Execution and Evaluation 32

4.5 Why We Choose Spring Boot for the Backend 34

4.6 Pagination 34

4.7 CORS 36

4.8 Search 37

4.8.1 Automatic Submission 37

4.8.2 Keyword Processing 38

4.8.3 Fuzzy Search 40

4.9 Personalized Recommendation 42

4.10 Android 44

4.10.1 Android WebView 44

4.10.2 File Upload 44

5. Results and Discussion 47

5.1 Project Outcome 47

5.1.1 Customer: Product list page with recommendation, filtering, searching, and pagination 47

5.1.2 Customer: Search page 49

5.1.3 Customer: Product detail page 50

5.1.4 Customer: Shopping cart page 52

5.1.5 Customer: Order generation page 53
5.1.6 Customer: Order list page with filtering 53
5.1.7 Customer: Order detail page 54
5.1.8 Customer: Account page 55
5.1.9 Customer: Login page and signup page 56
5.1.10 Vendor: Product list page with searching and adding a new product 57
5.1.11 Vendor: Order list page 58
5.1.12 Vendor: Order detail page 59
5.2 Testing and System Evaluations 60
5.2.1 Test on Recommendation Mechanism 60
5.2.2 Debug On a Real Android Phone 62
6 Conclusion and Further Work 63
References 64
Appendix A. Project Management 67
Project Scope Management 67
Project Schedule Management 67
Project Human Resource Management 68
Responsibility Assignment Matrix (RAM) 68
Resource Histogram 69
Changes and Adjustment 70

1 Introduction

Does the fast-paced lifestyle bother you for a long time? Have you ever been frustrated to have no time to go shopping after exhausting work? You may want to try the mobile App that we designed to provide comfort and convenience for situations like this, allowing you to wind down without having to go out shopping after a long day of work.

1.1 Overview

The Web has become an essential tool for people in their daily lives. In addition, the mobile phone is quite portable means to reach the Internet, thus increasing the number of mobile phone users. Therefore, mobile App has become the mainstream of Web App development. Due to the fast-paced lifestyle, many nearly have insufficient time to go shopping after tired work, so brick-and-mortar stores have fewer customers. In order to solve this type of problem, a variety of shopping mobile App like Amazon [1] and Taobao [2] appear. With the App, customers can shop during their leisure time without visiting a brick-and-mortar store. Vendors could add their inventory and present the details of products directly by posting images or textual descriptions online, which means they could reduce money for rent.
This project, our mobile App (Niubility) aims to establish a mobile platform for customers to buy mobile phones from various brands. Vendors showcase their merchandise in a user-friendly manner and facilitate purchases by potential customers. Furthermore, this App provides brilliant services which could show details of every purchase order.

1.2. Objectives

The main objective of this project is to create an intuitive mobile App for online shopping that allows vendors to sell their products to customers. Similarly, customers can also make purchases more conveniently. For those logged in as a vendor, this App allows them to maintain product catalogues. For example, they can browse the product catalogue, edit some product attributes, and add new products. They can also list purchase orders by different status and ship, hold or cancel them on the purchase order processing page.

For the customer side, customers can conveniently browse and filter the products and add them to their shopping cart. They can also click on a specific product and view detailed information. After placing an order, the customer can check the order processing status on the order page. Moreover, this App allows customers and vendors to manage their accounts securely, such as registering, logging in, and logging out.

So far, the mobile shopping App has been briefly introduced. The structure of this report is as follows: Chapter 2 introduces the background and related work of our work. Chapter 3 presents the system design of our design approach. Chapter 4 shows the implementation of our system architecture and module design. Chapter 5 displays the result of our project outcome and discussion. Chapter 6 summarizes our entire project and future work.

2 Background and Related Work

E-commerce (electronic commerce) is the buying and selling of goods and services, or the transmitting of funds or data, over an electronic network, primarily the internet [3]. This chapter will be divided into 2 sections. The first section is background. In this section, the general features of the e-commerce platforms and the reasons why we choose to construct mobile shopping App will be interpreted. The second section concerns the related work of the project. The comparison made with several mainstream platforms will be illustrated in this section.

2.1 Background

Generally speaking, an e-commerce platform establishes an online market consisting of various stores. So, the primary features of these platforms are buying and selling. Different stores showcase their distinctive products and provide one-to-one online chat if customers have some questions with the specific product. Customers can search for the desired products or stores by simply typing their names into the search engine. After searching, customers are capable of using the filter to select more satisfying products. In the end, all the desired products can be checked out at one time. Here the shopping cart plays an important role. The platform provides a shopping cart where you can put satisfying products from different stores. In addition, payment is also a crucial feature of e-commerce platforms. Customers need to pay for the selected items and they can go through this process in several types. After the payment, the platform will store the information about the shipping process for a while until the order is completed. And the previous orders are all stored on the platform. Moreover, after the order is completed, the user can comment on the product and upload pictures, which can help other users to choose the product. In the following, we will illustrate the reasons why we choose mobile e-commerce App.

According to statistics, the penetration rate of smartphones in China reached almost 72 percent and the total number of smartphone users has exceeded one billion in 2022 [4]. In reality, it is essential for us to hold and use smartphones everywhere. We use it to search for information, watch online videos, or even pay through our mobile phones. Not exaggerating to say, we cannot live without smartphones.

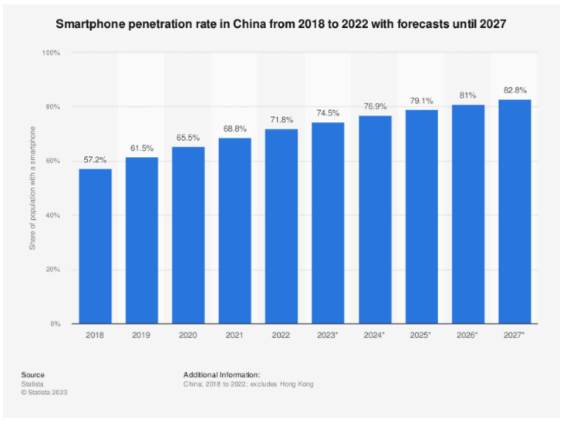


Figure 2-1-1: Bar chart about smartphone penetration rate in China

Where is your mobile phone now? It's probably close to hand and that's the case for a majority of smartphone users. Even if you are looking for something quickly on your phone, it is still highly possible for you to purchase some stuff resulting from random searching. So, this is why mobile commerce is growing at such a dramatical rate: It is incredibly convenient [5].

Besides a fast-responsive and intuitive mobile App could shorten the time to purchase an item and give users immediate gratification. Because they only ought to tap on the screen with several actions to get the desired merchandising. The product will be shipped to their families in around two days. These are two reasons why we are determined to develop a mobile App.

2.2 Related Work

There are a lot of mobile online shopping malls in today's world. The mainstream platforms are Taobao [6], JD.com [7] and Amazon [8]. They sell different kinds of products. But our mobile App mainly focuses on mobile phones. In the following, we would like to compare Niubility (our mobile App) with these platforms.

First of all, it is quite common for us to return to the same place when we are tapping into the detail page of a product in the shopping cart and exit in these renowned platforms. Actually, it is an efficient and convenient design for users. The same situation also happens on the product list page, order list page, and so on. Niubility also has this function. Imagine the scenario, when a user is attracted by a product, he just goes to the detail page to obtain more information. When he exits, he is forced to return to the beginning point of the product list page if the App does not have this function. It is a waste of time for the user and may destroy the delightful experience of the user.

Secondly, these three online shopping malls acknowledge some data from other Apps on mobile phones and then analyze what products the user may like. It will recommend a list of items on the home page which is also an effective way to promote sales volume. Well, Niubility is just a simple mobile App that could not get any data from other Apps on mobile phones. But we thought this function was quite fantastic and intriguing. We actually write an algorithm to do a similar thing but require users to stamp some products as "like" or "dislike". After getting this information, Niubility will work the same as the mainstream online shopping malls to recommend several distinctive products to specific users.

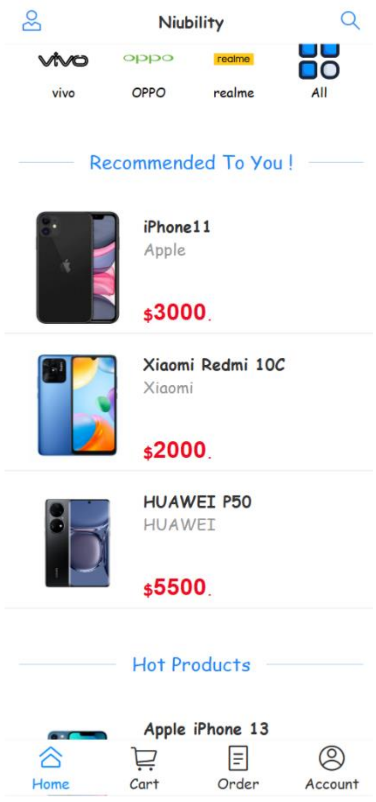


Figure 2-2-1: Result of recommend algorithm

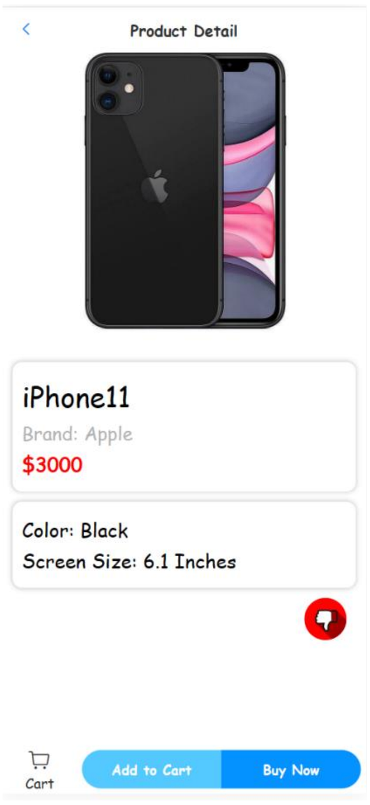


Figure 2-2-2: A product stamped as "like"

Additionally, we consider the feature that the cart item on the bottom bar of the home page can show the total number of products in the shopping cart to be intuitive and efficient. Not only will this promote the shopping experience of customers because it could give customers an apparent view of the number of added items so that they can manage the content of their shopping cart, but this will also stimulate the desire of customers to check out the products in shopping cart which will increase the sales volume further. During the comparison, we found JD.com and Amazon both have this feature, but Taobao does not have this feature. Even though not all the mainstream platforms have this feature, we still consider it is crucial and necessary to contain the feature which will promote the user-friendliness of our mobile App.

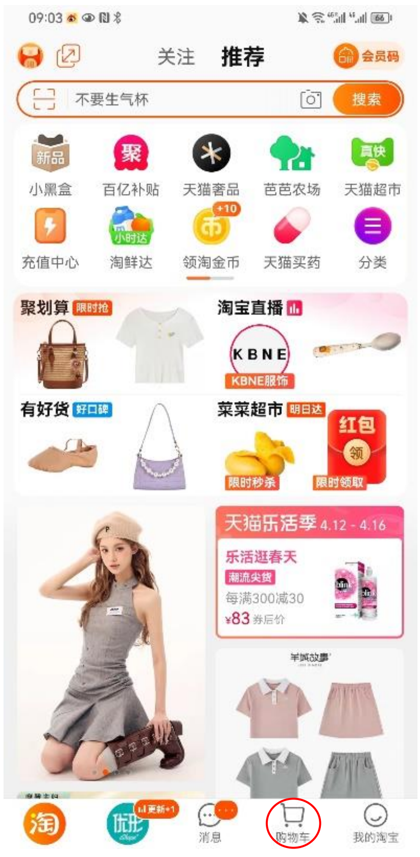


Figure 2-2-3: Home page of Taobao

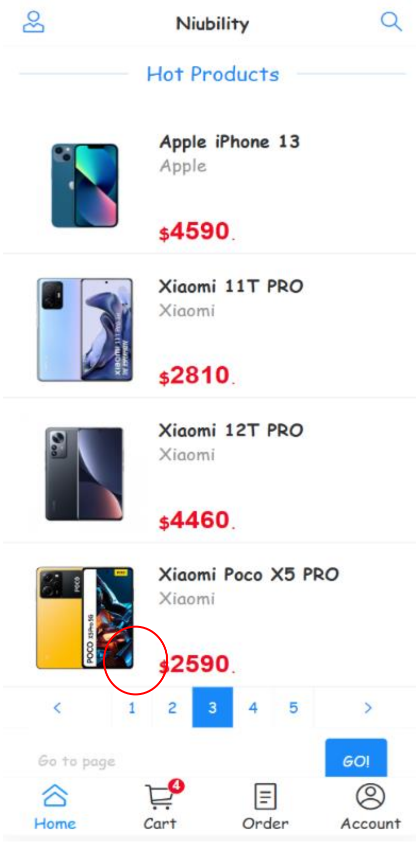


Figure 2-2-4: Home page of Niubility

3 System Design

In this chapter, we will describe the data modeling and dynamic modeling for our mobile shopping App. In the data modeling section, we will use Entity Relationship Diagram to describe the structure of our database and our design decisions. In the dynamic modeling section, we will use state diagrams, activity diagrams, and sequence diagrams to illustrate the interaction and dynamic aspects of our system.

3.1 Data Modelling

In this section, we will use the Entity Relationship Diagram to describe the main entities of our system and the relationships between them.

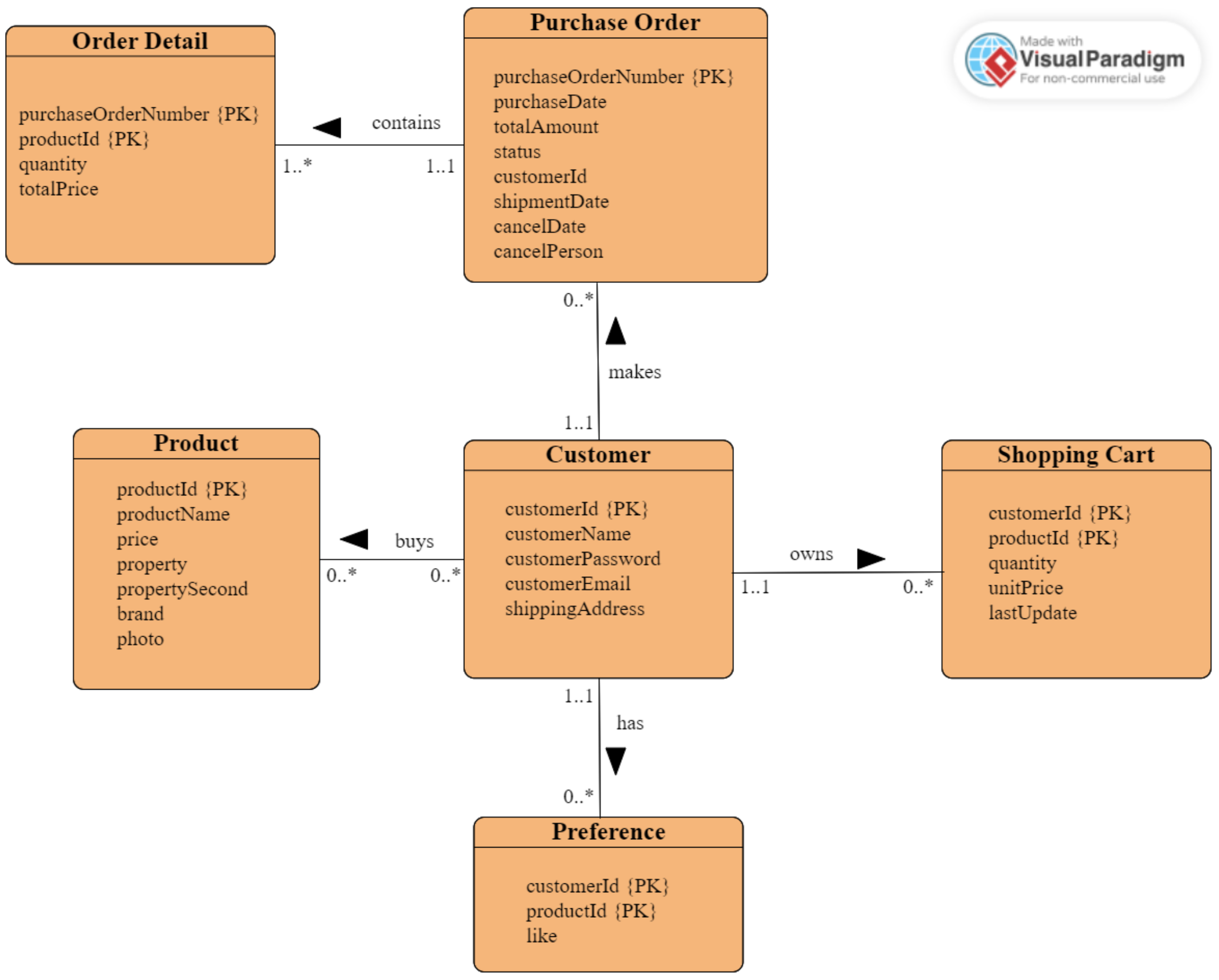


Figure 3-1-1: ER diagram

Figure 3-1-1 illustrates our ER diagram for Niubility (our mobile App). The main entities are:

- Customer: This entity represents the user who has registered in Niubility (Note that we assume there is only one vendor in this App). It stores information about customers' unique ID numbers, names, email addresses, passwords, and shipping addresses.
- Product: This entity stores information about each product that is available for sale on Niubility, such as its distinctive ID number, name, price, two properties, brand, and photo.
- Shopping Cart: This entity stores all the products the customer chooses which also includes its quantity, unit price, and date when it is added.
- Purchase Order: This entity stores information about all orders the customer makes, which includes its unique number, purchase date, total amount, order status, and customer Id. If the products of this order are shipped, it will record the shipment date. If the order is canceled, it will record the cancel person and the cancel date.
- Order Detail: This entity stores the detailed information of each order, such as the specific product(productId), the number of the specific product, and the total price of this product.
- Preference: This entity stores a list of products that are stamped as "like", including the unique customer Id number, product Id number, and the identifier whether the customer like this product.

The relationships between these entities are:

- A customer can buy zero or many products. A product can be bought by zero or many customers.
- A customer can own zero or many shopping carts (Note: the entity Shopping Cart records one product and one customer at one time. You could read the figure below to get a better understanding). A shopping cart can be owned by one and only one customer.

customerId	productId	quantity	unitPrice	lastUpdate
1	6	2	5000	2023-3-26 15:00:06

Figure 3-1-2: One shopping cart in the database

- A customer can make zero or more purchase orders. A purchase order can be made by one and only one customer.
- A purchase order can contain one or many order details. But an order detail can be contained by one and only one purchase order.
- A customer can have one or more preferences. But one preference could only belong to one customer.

The logical data model is listed below, giving further details on the foreign key allocation.

Customer (customerId, customerName, customerEmail, customerPassword, shippingAddress)

Primary Key: customerId

Product (productId, productName, price, property, propertySecond, brand, photo)

Primary Key: productId

Shopping Cart (customerId, productId, quantity, unitPrice, lastUpdate)

Primary Key: customerId, productId

Foreign Key: customerId references Customer(customerId)

productId refereces Product(productId)

Purchase Order (purchaseOrderNumber, purchaseDate, totalAmount, status, customerId, shipmentDate, cancelDate, cancelPerson)

Primary Key: purchaseOrderNumber

Foreign Key: customerId references Customer(customerId)

Order Detail (purchaseOrderNumber, productId, quantity, totalPrice)

Primary Key: purchaseOrderNumber, productId

Foreign Key:

purchaseOrderNumber references Purchase Order(purchaseOrderNumber)

productId refereces Product(productId)

Preference (customerId, productId, like)

Primary Key: customerId, productId

Foreign Key: customerId references Customer(customerId)

productId refereces Product(productId)

Figure 3-1-3 shows the data dictionary of all the entities and the related attributes of Niubility.

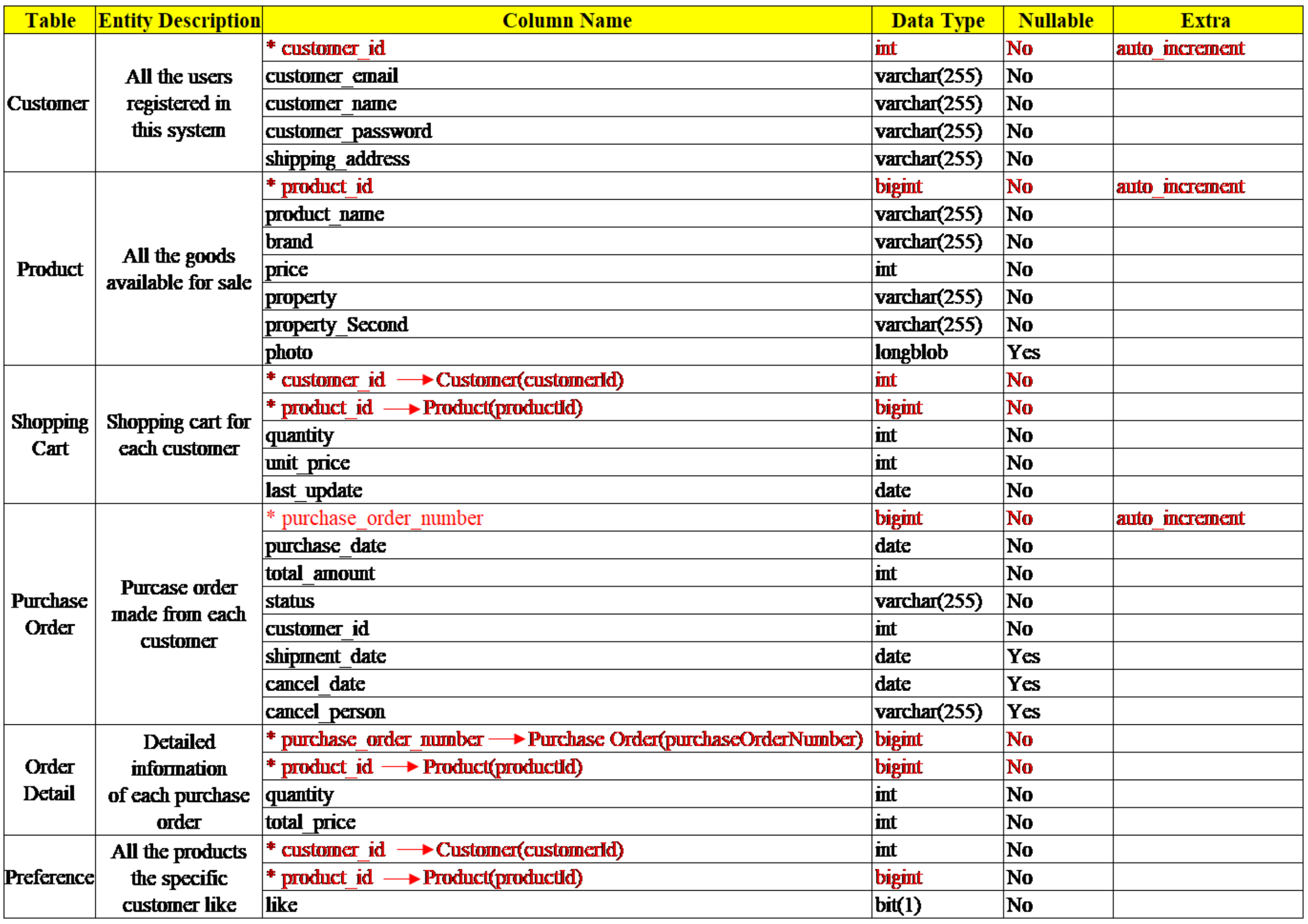


Table	Entity Description	Column Name	Data Type	Nullable	Extra
Customer	All the users registered in this system	* customer_id	int	No	auto_increment
		customer_email	varchar(255)	No	
		customer_name	varchar(255)	No	
		customer_password	varchar(255)	No	
		shipping_address	varchar(255)	No	
Product	All the goods available for sale	* product_id	bigint	No	auto_increment
		product_name	varchar(255)	No	
		brand	varchar(255)	No	
		price	int	No	
		property	varchar(255)	No	
		property_Second	varchar(255)	No	
		photo	longblob	Yes	
Shopping Cart	Shopping cart for each customer	* customer_id → Customer(customerId)	int	No	
		* product_id → Product(productId)	bigint	No	
		quantity	int	No	
		unit_price	int	No	
		last_update	date	No	
Purchase Order	Purcase order made from each customer	* purchase_order_number	bigint	No	auto_increment
		purchase_date	date	No	
		total_amount	int	No	
		status	varchar(255)	No	
		customer_id	int	No	
		shipment_date	date	Yes	
		cancel_date	date	Yes	
		cancel_person	varchar(255)	Yes	
Order Detail	Detailed information of each purchase order	* purchase_order_number → Purchase Order(purchaseOrderNumber)	bigint	No	
		* product_id → Product(productId)	bigint	No	
		quantity	int	No	
		total_price	int	No	
Preference	All the products the specific customer like	* customer_id → Customer(customerId)	int	No	
		* product_id → Product(productId)	bigint	No	
		like	bit(1)	No	

Figure 3-1-3: Data dictionary

3.2 Dynamic Modelling

The dynamic modeling section of an online shopping mall project report should describe the interaction and dynamics of our system. In this section, we will use the state diagram, activity diagram, and sequence diagram to model the dynamic behavior of our mobile shopping App.

3.2.1 State Diagrams

The state diagram is to describe the behavior of a single object in response to a series of events in a system. Figure 3-2-1-1 shows the order state diagram and the order state changes with the interaction between the customer and vendor. This state diagram shows the different states that an order can be in: “pending”, “hold”, “shipped”, and “cancelled”. First, the state of the order is “pending” after the customer checks out. After that, the vendor can change the state to “hold” or “shipped” according to the stock.

For out-of-stock, if there is stock later, the vendor can change the state from “hold” to “shipped”. In addition, both customer and vendor can only change the state to “cancelled” if the order is in the pending” or “hold” state.

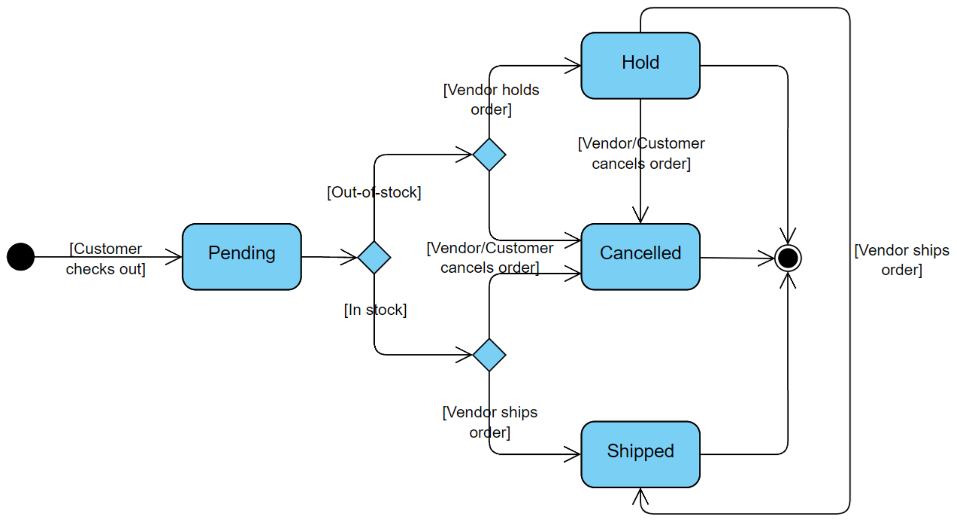


Figure 3-2-1-1: Order state diagram

3.2.2 Activity Diagrams

The activity diagram is to show the flow from one activity to another in a system or process. In this section, we will use three activity diagrams to model the behavior of customer who is not logged in, customer logged in, and vendor.

Figure 3-2-2-1 shows the activity diagram for customers who are not logged in or the new user. In this diagram, customers who are not logged in (including the new users) can only operate on a small number of pages, such as the home page, search page, and product detail page. First, customers can go to the home page to browse the product list and they can filter the products by brands. Moreover, if they want to find the products that they are interested in, they can search the products by keywords on the search page, and they can also filter the products by brands on the search page.

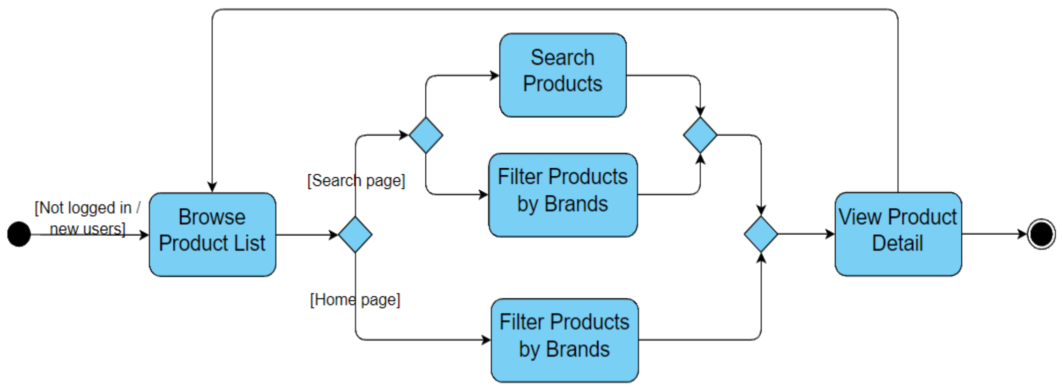

[Not logged in / new users]
Browse Product List
[Search page]
[Home page]
Search Products
Filter Products by Brands
Filter Products by Brands
View Product Detail

Figure 3-2-2-1: Customer not logged in or new customer

Figure 3-2-2-2 illustrates the customer activity diagram that describes the activities of the customers who can log in. First, customers can log in to the system by entering their login credentials. For the new users, they can create their accounts on the register page, and after registration, they will log in automatically. Next, they can browse the products and add the products they want to buy to their shopping cart. In the shopping cart, customers can choose to change the quantity of the items they added or remove the items from their cart. Then, customers can check out and go to the confirm page. In addition, the shopping cart will become empty after they check out. On the confirm page, customers are allowed to change their addresses. After confirming the order, the system will show the order detail page of the newly created order. Additionally, customers can change their usernames and passwords on the account management page. At last, after the customers log in, they can directly navigate to the home page, shopping cart page, order list page, and account page according to their needs.

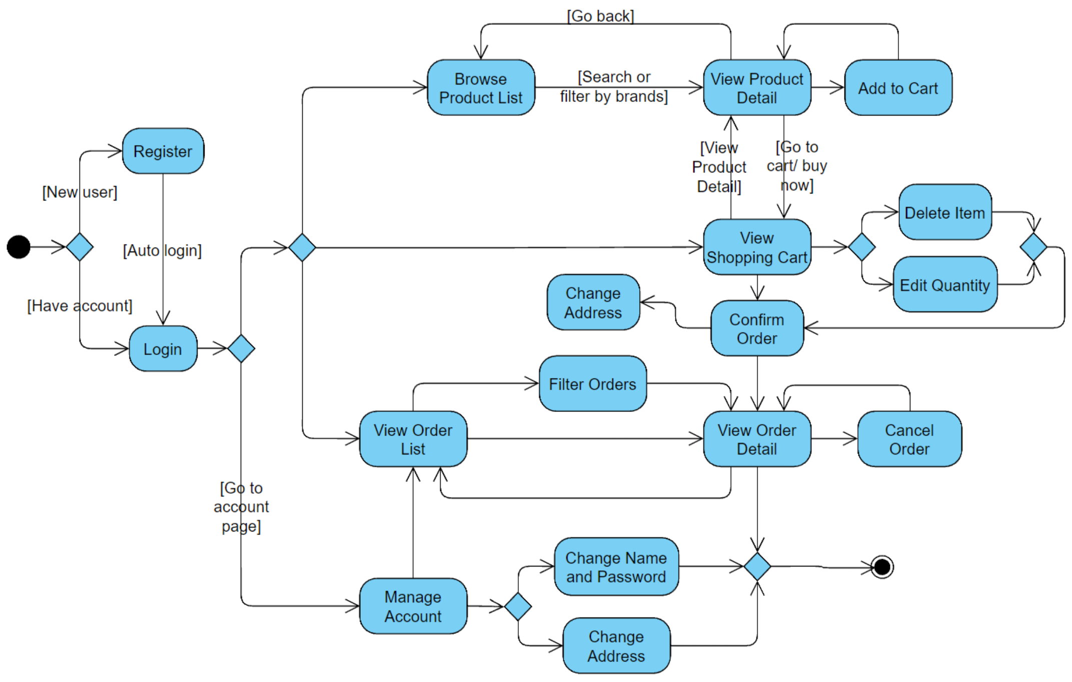


Figure 3-2-2-2: Customer activity diagram

Figure 3-2-2-3 exhibits the vendor activity diagram that shows the behavior of the vendor in the system. In this diagram, the vendor can enter detailed information about the new products and add them to the catalog. When browsing the product list, the vendor can search and filter the products in the same way as customers. In addition, the vendor can also find a specific product by entering the product ID. Moreover, the vendor can filter the orders by the order states or search for the specific order by entering the order ID. On the order detail page, the vendor can change the order states. The vendor can ship the orders if their states are “pending” or “hold”, and hold the orders if the orders are in the “pending” state. The vendor can also cancel the order if necessary.

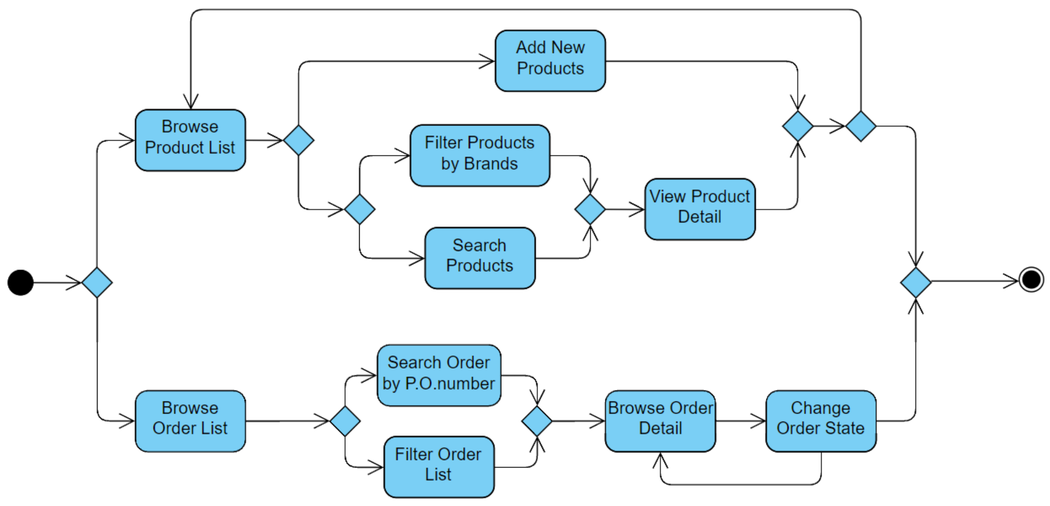


Figure 3-2-2-3: Vendor activity diagram

3.2.3 Sequence Diagram

A sequence diagram consists of a group of objects that are represented by lifelines, and the messages that they exchange overtime during the interaction. In this section, we will use two sequence diagrams to illustrate the interaction between the customer, mobile shopping App, and vendor.

Figure 3-2-3-1 shows the interaction between the customer and the App from the login to the checkout process. In the beginning, the existing users are required to enter their login credentials, and the new users are required to enter their registration information. After verification, the system will show the account page to users. Then, users can request to filter and search the products and the system will show the result to users. After the user taps, the system will return the product detail page and the user can choose to add to their shopping cart. Next, users can request the shopping cart page to change the items' quantity or remove the item from their cart. And then they can click to check out. After checking out, the system will show the confirm page to users and users need to confirm the order. Finally, the system will show the order detail page to users.

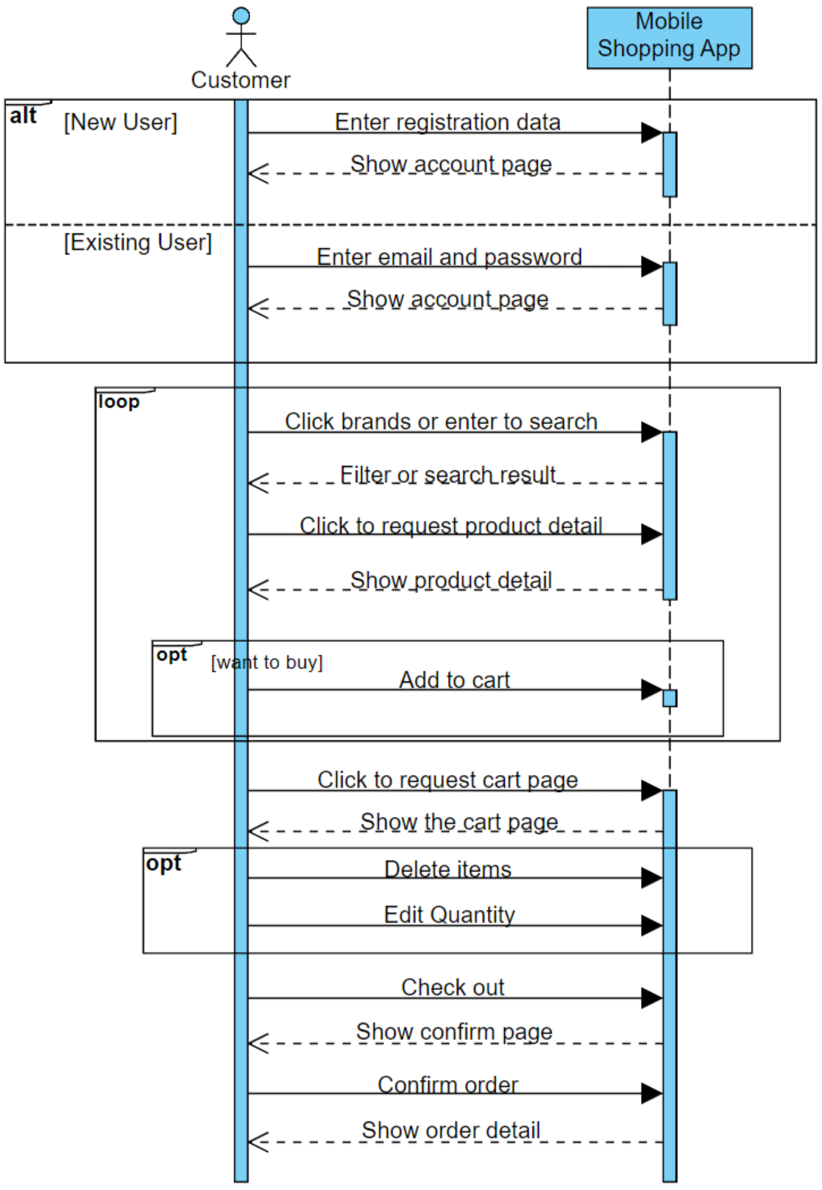


Figure 3-2-3-1: Customer shopping sequence diagram

4 System Implementation

Our project adopts a very popular technology for nowadays Web App development: the front-end and back-end separation development model. “Separation of front and back end” has become the industry benchmark for Internet project development, which is effectively decoupled by “nginx + tomcat” and lays a solid foundation for future large-scale distributed architecture, elastic computing architecture, microservice architecture, and multi-terminated services. The front-end technology stack of our project is Vue [9], and the back-end technology stack is Spring Boot [10]. Both technology stacks are very popular in their respective fields. The front-end and back-end have their own responsibilities and do not depend on each other, which greatly improves our development efficiency. And the front-end and back-end are in JSON format for data interaction, making our data transfer more concise and efficient.

4.1 Architecture

Our system is implemented in traditional client-server architecture. The front-end is implemented in Java (for Android App development) and Vue.js (front-end UI development). The backend is implemented in the Spring Framework with the connection to a MySQL database server. The front-end retrieves data by sending requests and receiving responses at the API that the backend exposes. *Figure 4-1-1* is the deployment diagram which provides an overview of our system architecture. Details of backend servers are elaborated in **section 4.1.3.**

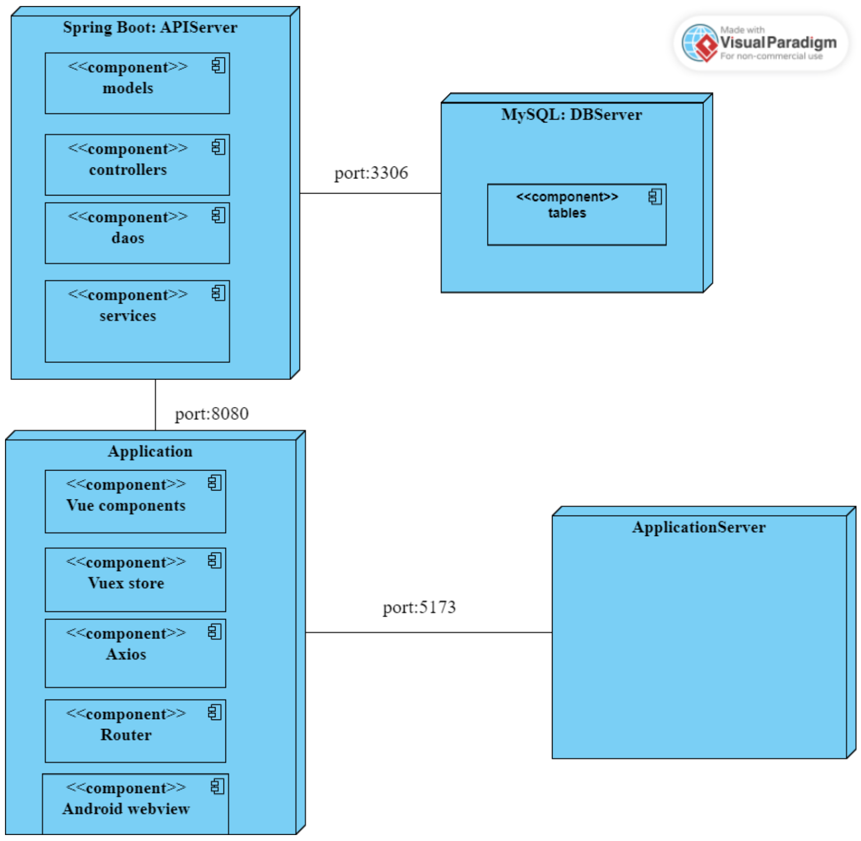


Figure 4-1-1: Development Diagram for Back-end

4.1.1 MVC Structure

Our application adopts an MVC structure. MVC consists of a Model, View, and Controller. *Figure 4-1-1-1* demonstrates the overall MVC architecture of our system.

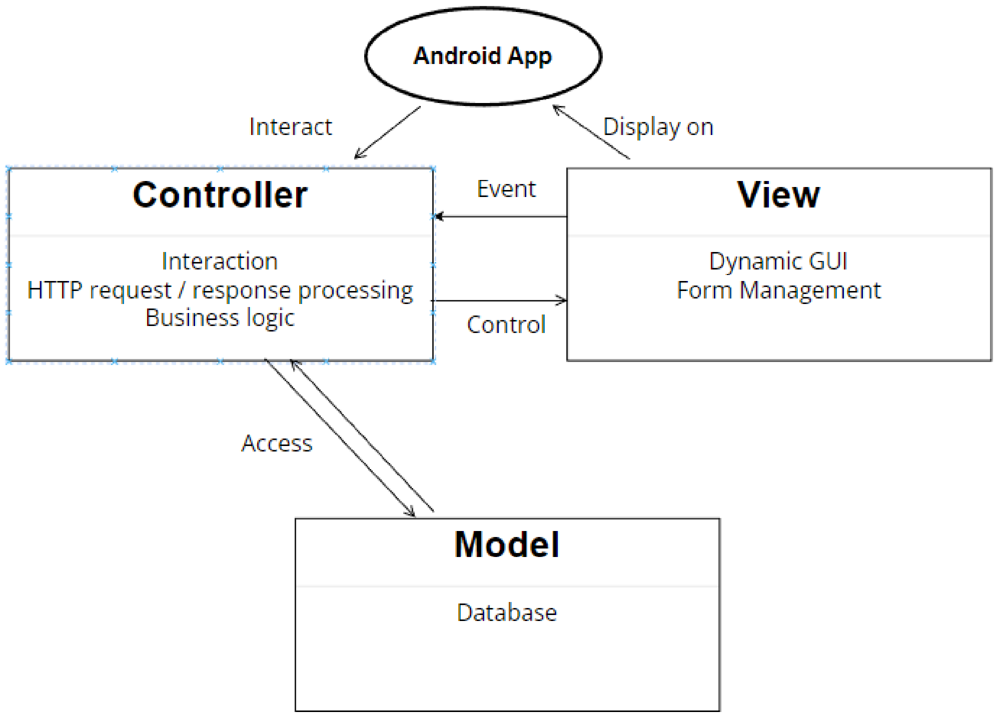
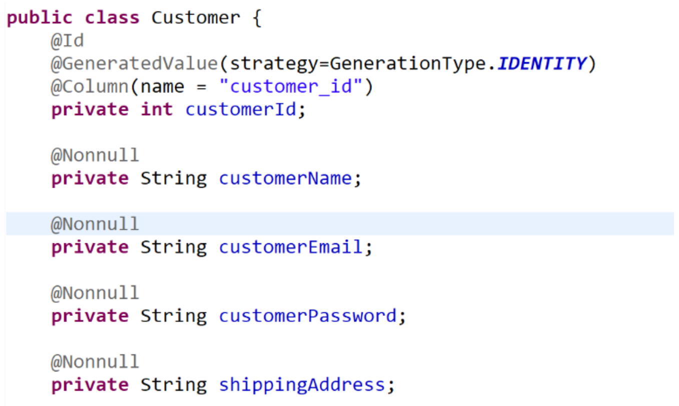


Figure 4-1-1-1: Development Diagram for Back-end

Model component represents data model and defines the storage of all the application's data objects [11]. Besides, it all contains the logic of data operation. In Niubility, Spring Boot JPA actually helps us deal with this component. we will use Customer entity to illustrate. *Figure 4-1-1-2* shows we simply create a class to define the specific entity and its related attributes. Spring Boot JPA will automatically generate corresponding tables in MySQL server. So how can we query the database server to get the desired data? Here Dao (Data Access Object) plays an important role, which could allow developers simply write the method name without requiring them to input the detailed query. And Dao actually provides the basic CRUD functions with some specific situations. *Figure 4-1-1-3* present the method in ShoppingcartDao which get the shopping cart records belonging to the specific customer.

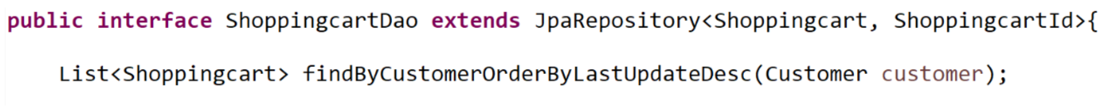
View is the component which is associated with User Interface. It provides the visual representation of MVC model [11]. In other words, it displays the output to the user. In the application, Vue component actually in charge of this. They are dynamically rendered under Vue.js framework so that the user can get direct visualization of data in View component.

Controller component takes care of request handler [11]. It actually like a bridge between Model and View. There are two types of controllers in our system. As for the model controllers running at the back-end, they control API as entry points and mapping among them. Accordingly, the view controller is able to access model with the help of back-end controllers.



```
public class Customer {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name = "customer_id")
    private int customerId;

    @Nonnull
    private String customerName;

    @Nonnull
    private String customerEmail;

    @Nonnull
    private String customerPassword;

    @Nonnull
    private String shippingAddress;
```

Figure 4-1-1-2: Customer Entity



```
public interface ShoppingcartDao extends JpaRepository<Shoppingcart, ShoppingcartId>{

    List<Shoppingcart> findByCustomerOrderByLastUpdateDesc(Customer customer);
```

Figure 4-1-1-3: ShoppingcartDao

4.1.2 Front-end

The front-end architecture of our project mainly includes four parts, they are Vue Components, Router [12], Vuex Store [13], and Axios [14]. Vue Components is mainly used for UI rendering. Router is used to manage page navigation and jump. Vuex Store is a centralized storage to manage the state of all components. Axios is a promise-based network request library. Each of them will be introduced in detail in the following sections. *Figure 4-1-2-1* is the architecture diagram for the front end.

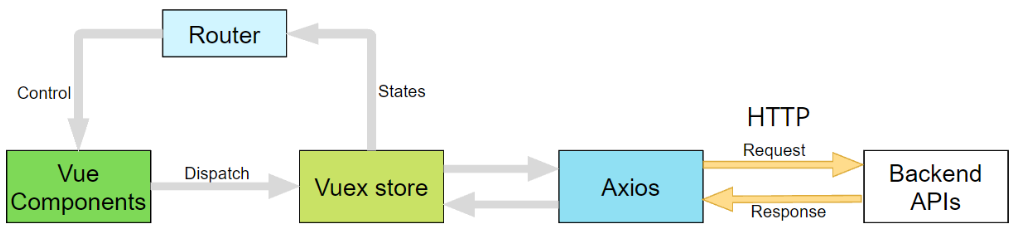
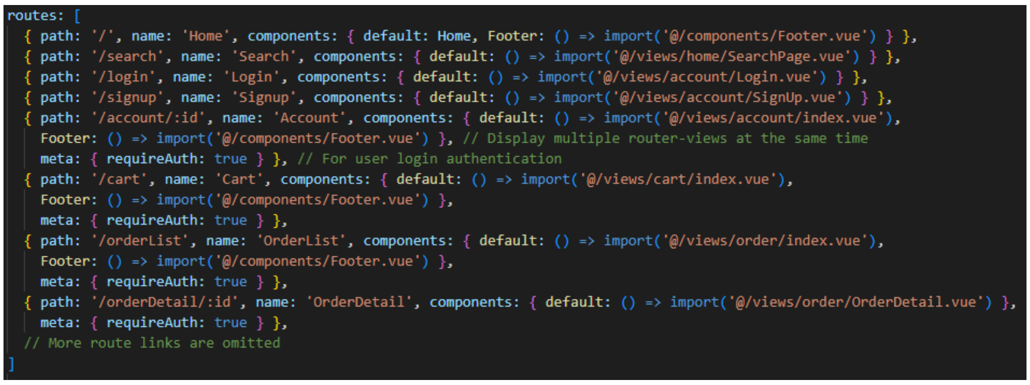


Figure 4-1-2-1: Front-end Architecture Diagram

For the front-end design, we use Vue.js 3.0 or Vue 3 which is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative and component-based programming model that makes it more efficient to develop the UI. And we also use the Single Page Application [15] which is also the JavaScript framework for our front-end design. Compare with Vue 3, Vue 2 is also a very popular framework in recent years. But we choose to use Vue 3 not only because it is the current and latest major version of Vue, but also for many reasons. First, Vue 3 contains new features that are not present in Vue 2, such as SFC Composition API Syntax Sugar [16], Suspense, and multiple root elements per template. For instance, we use <script setup> to implement the Composition API inside the Single-File Components (SFC) [17]. Another reason is Vue 3 provides smaller bundle sizes, better performance, better scalability, and better TypeScript / IDE support. In general, Vue 3 makes the UI development of our project easier and more efficient, enabling us to better fulfill the functional requirements.

Because we use Vue.js to develop our mobile online shopping App, so it is not a native mobile App. If we just use links to navigate to the new pages. Every time when users jump to a new page, it will be reloaded, which will greatly affect the user experience. In order to make users feel that it is more like a mobile App. We used Single-Page Application (SPA) which is a Web App or website supported by Vue.js. We want it to rewrite the page with new content fetched from a Web server as the user interacts with it instead of loading a new page for every interaction. Thus, to achieve this function, we used Vue Router. Vue Router is the official routing library for Vue.js, which is great for handling the routing in Single-Page Applications. This means our App is only loaded once from the server to the browser and the result is that the browser does not need to reload when routing between pages and gives the user a smooth navigation experience between different pages. Here is an example of the router link in our project:



```
routes: [
  { path: '/', name: 'Home', components: { default: Home, Footer: () => import('@/components/Footer.vue') } },
  { path: '/search', name: 'Search', components: { default: () => import('@/views/home/SearchPage.vue') } },
  { path: '/login', name: 'Login', components: { default: () => import('@/views/account/Login.vue') } },
  { path: '/signup', name: 'Signup', components: { default: () => import('@/views/account/SignUp.vue') } },
  { path: '/account/:id', name: 'Account', components: { default: () => import('@/views/account/index.vue'),
    Footer: () => import('@/components/Footer.vue') }, // Display multiple router-views at the same time
    meta: { requireAuth: true } }, // For user login authentication
  { path: '/cart', name: 'Cart', components: { default: () => import('@/views/cart/index.vue'),
    Footer: () => import('@/components/Footer.vue') },
    meta: { requireAuth: true } },
  { path: '/orderList', name: 'OrderList', components: { default: () => import('@/views/order/index.vue'),
    Footer: () => import('@/components/Footer.vue') },
    meta: { requireAuth: true } },
  { path: '/orderDetail/:id', name: 'OrderDetail', components: { default: () => import('@/views/order/OrderDetail.vue') },
    meta: { requireAuth: true } },
  // More route links are omitted
]
```

Figure 4-1-2-2: Router Links

To better manage component state, store and manage data obtained from the back-end, and provide users with a better experience, we adopted Vuex which is a state management pattern + library for Vue.js application. It serves as a centralized store for all the components and it has the ability to store and share reactive data across the app without trading off performance, testability, or maintainability. If our system is not too complicated, we don't need to use Vuex, but our Web app involves multiple components that need to share data or need to transfer and synchronize data between different components. For example, from the order list page to the order detail page not only need to share order data (such as product and user data), but also need to synchronize the data between the two pages (such as order state). At this time Vuex is very useful. Because Vuex is a centralized store, any component in the application can access the data in Vuex, and the data in Vuex is reactive data, so if one component changes the value of a variable in Vuex, the value of the variable in other components will also change accordingly.

Figure 4-1-2-3 shows the structure of Vuex. In this structure diagram, if the components want to change the data, they can use the Dispatch method to call the Actions to process complex business logic or interact with the back-end. Then, the Actions will call the methods in the Mutations through the Commit method to change the State. Finally, the reactive data in the State will be rendered by components.

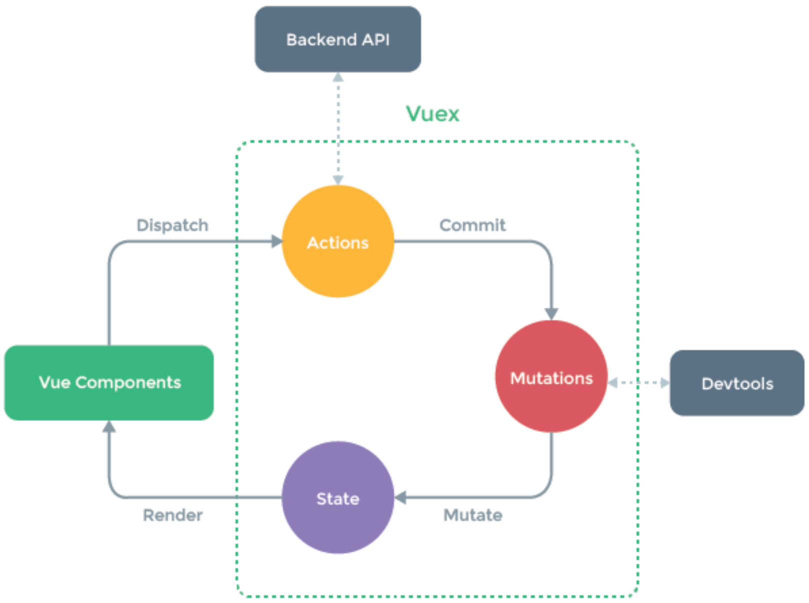
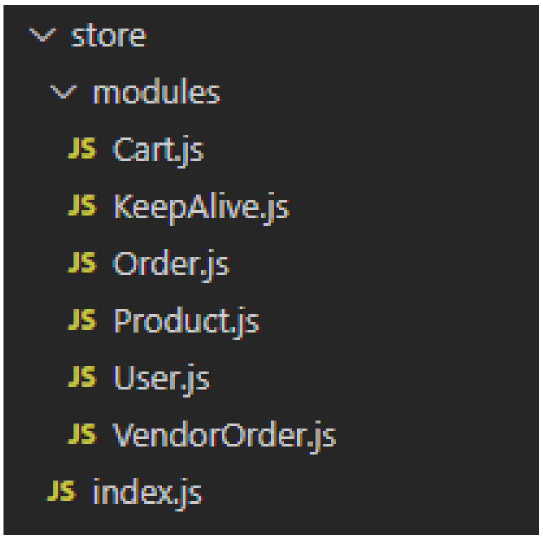


Figure 4-1-2-3: Vuex Structure

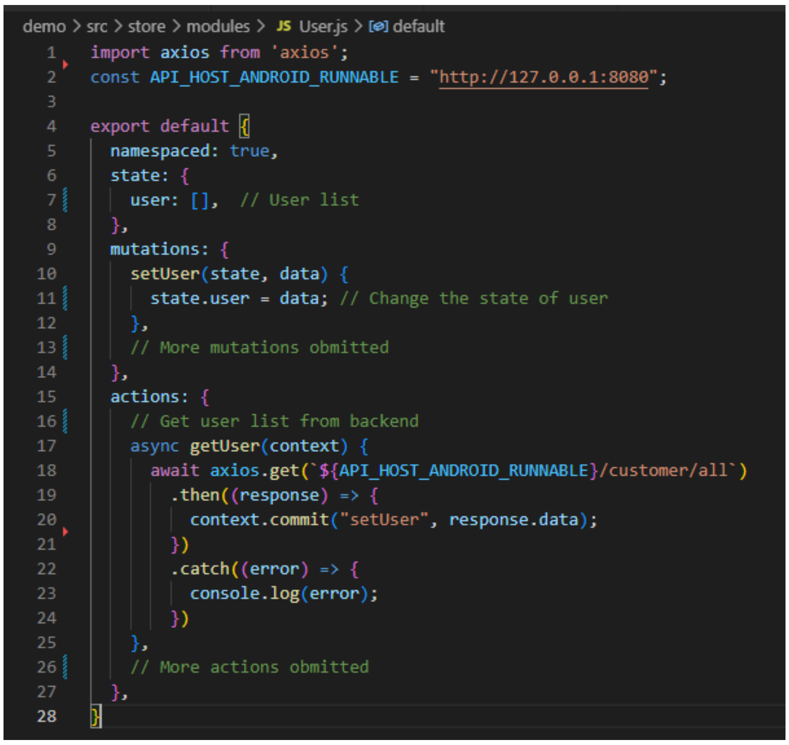
Figure 4-1-1-4 displays all the modules of the store in our system, such as User.js and Product.js. Each module can contain its own state, mutations, and actions.



```
store
  modules
    Cart.js
    KeepAlive.js
    Order.js
    Product.js
    User.js
    VendorOrder.js
  index.js
```

Figure 4-1-2-4: All the states in the store

Figure 4-1-1-5 exhibits a part of the User.js module. As you can see, the actions can get the user list from the back end. After getting the response, it will call the mutations to change the “user” in the state.



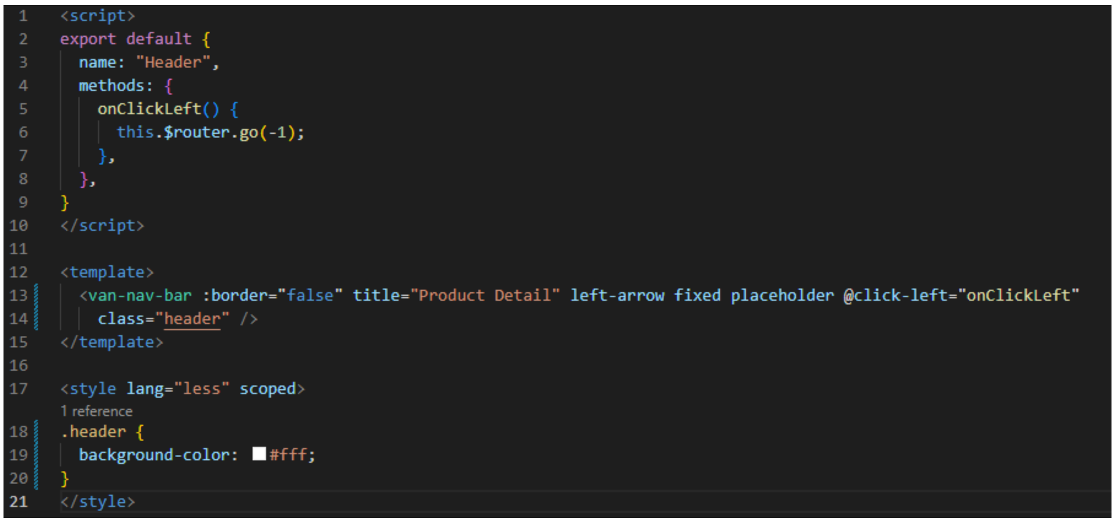
```
import axios from 'axios';
const API_HOST_ANDROID_RUNNABLE = "http://127.0.0.1:8080";

export default {
  namespaced: true,
  state: {
    user: [],  // User list
  },
  mutations: {
    setUser(state, data) {
      state.user = data; // Change the state of user
    },
    // More mutations obmitted
  },
  actions: {
    // Get user list from backend
    async getUser(context) {
      await axios.get(`${API_HOST_ANDROID_RUNNABLE}/customer/all`)
        .then((response) => {
          context.commit("setUser", response.data);
        })
        .catch((error) => {
          console.log(error);
        })
    },
    // More actions obmitted
  },
}
```

Figure 4-1-2-5: User module of the store

To connect to the back end, we employed Axios in the front end. Axios is a promise-based HTTP Client for node.js and the browser. Compare with Fetch, Axios is easier to use because of its built-in APIs. Axios also supports more functions such as built-in CSRF protection [18], canceling requests, and request timeout.

For the UI design, we use Vant 4 [19] which is a lightweight, customizable Vue UI library for mobile Web apps. This component library is dedicated to providing mobile components for Web apps. It provides more than 70 high-quality components, covering mainstream mobile scenarios, and has excellent performance. The average component size is less than 1KB. The most important thing is that we don't need to spend a lot of time writing very complicated CSS and it is very good for UI and UX design. Compare with Vant 4, there is also a popular Web app component library called mint-ui. At first, we wanted to use it to help design the UI of our Web app. but we found that it is actually a component library designed based on Vue 2, is not well compatible with Vue 3, and the maintenance and update speed of this component library is very slow, so we decided to use Vant 4.

For the file structure, we author Vue components using an HTML-like file format called Single-File Component (also known as *.vue files, abbreviated as SFC). A Vue SFC, as the name suggests, encapsulates the component's logic (JavaScript), template (HTML), and styles (CSS) in a single file. *Figure 4-1-2-6* is an example of the SFC file format in our project.



```
<script>
export default {
  name: "Header",
  methods: {
    onClickLeft() {
      this.$router.go(-1);
    },
  },
}
</script>

<template>
  <van-nav-bar :border="false" title="Product Detail" left-arrow fixed placeholder @click-left="onClickLeft"
    class="header" />
</template>

<style lang="less" scoped>
.header {
  background-color: #fff;
}
</style>
```

Figure 4-1-2-6: Single-File Components (SFC)

4.1.3 Back-end

Refer to *Figure 4-1-1*, the implementation includes API server, Application server and MySQL server. After consideration, we choose Spring Boot as our API server, which is a popular, open source, enterprise-level framework based on Java [10]. For Application server, it is responsible for transmitting Vue. While MySQL server is mainly used for storing the related data of our mobile App.

Spring Boot is an API server which is implemented with Java. It has high performance and is super popular among the developers, which means we could get more help and reference through resources on the Internet. Besides, it provides powerful tool named Spring Data JPA. It actually takes the advantage of the JPA specification, including the entity and association mappings, the entity lifecycle management, and JPA's query capabilities. Besides, it has extra functions which do not require codes to implement with the repository pattern on a higher abstraction level. And it also provides the function which help generate MySQL queries automatically based on the method names [20].

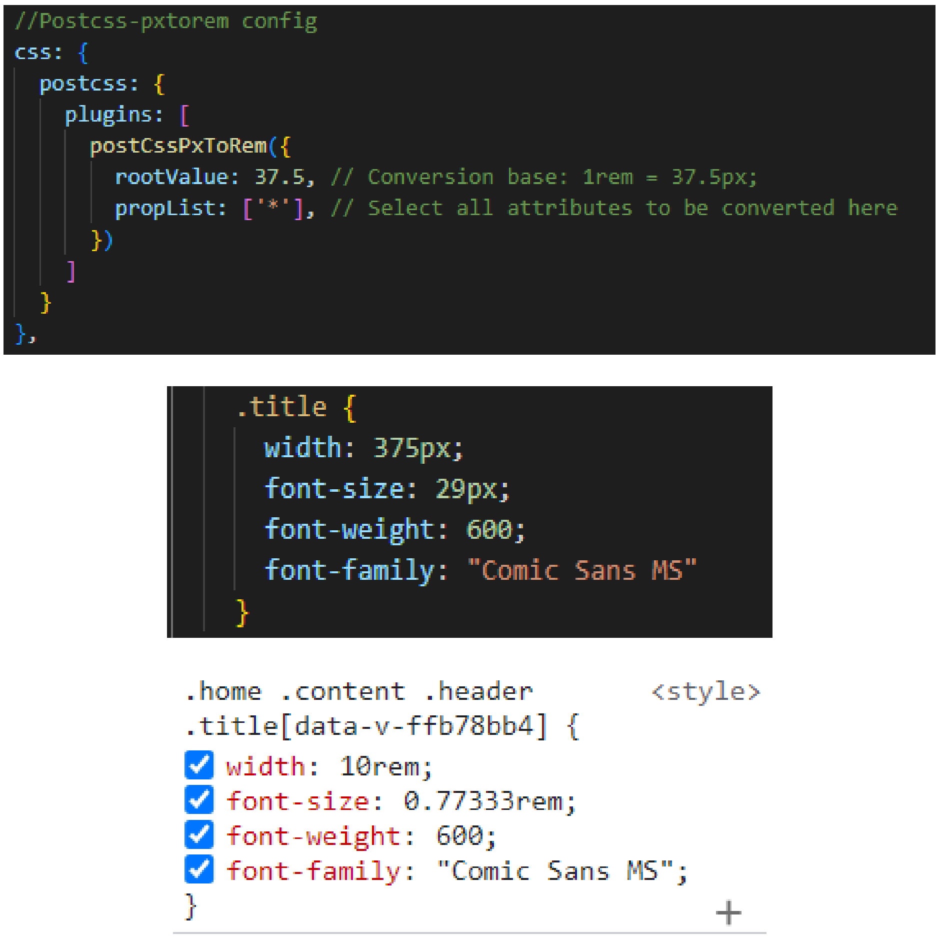
MySQL server is a perfect choice to store all the related data to accompany with Spring Boot as API server. MySQL is an open-source relational database management system (RDBMS) and it has also been tested to be a "fast, stable and true multi-user, multi-threaded SQL database server" [21], which means it has excellent performance. Since its first internal release on 23 May 1995, it has been developing for years. This means it has wide range of members applying it and we could get a variety of community support. Meanwhile, Spring Boot has already had many MySQL related tools and libraries. Developers could use MySQL more easily and quickly.

For Application server, it has the responsibility for sending static files to the client side. Here the static files are HTML files (Vue files).

4.2 REM Adaptation Solution for Mobile Screen

Nowadays, the brands of mobile phones are becoming more and more abundant, and the sizes of mobile phone screens are even more varied. Since our project is to develop a mobile Web App, we must first consider the problem of displaying Web content adaptively to the different screen sizes.

To solve this problem, we should first consider the unit that controls the page size. All lengths in the browser are in units of CSS pixels, which is what we often call "px", but px is absolute unit in CSS. Rem, on the other hand, is relative unit that is based on the document's root font size. So, it can change with the root font size, thus realizing the adaptive function. To convert the CSS unit px to rem. We used postcss-pxtorem [22] which is a plugin for PostCSS that generates rem units from pixel units. *Figure 4-2-1* shows how to configure postcss-pxtorem. In this file, we set the conversion base to 37.5 which is the “rootValue”. This means that 1rem is equal to 37.5px. For example, in the second picture, I set the width of the title to 375px, and the width rendered in the browser is 10rem, just as shown in the third picture.



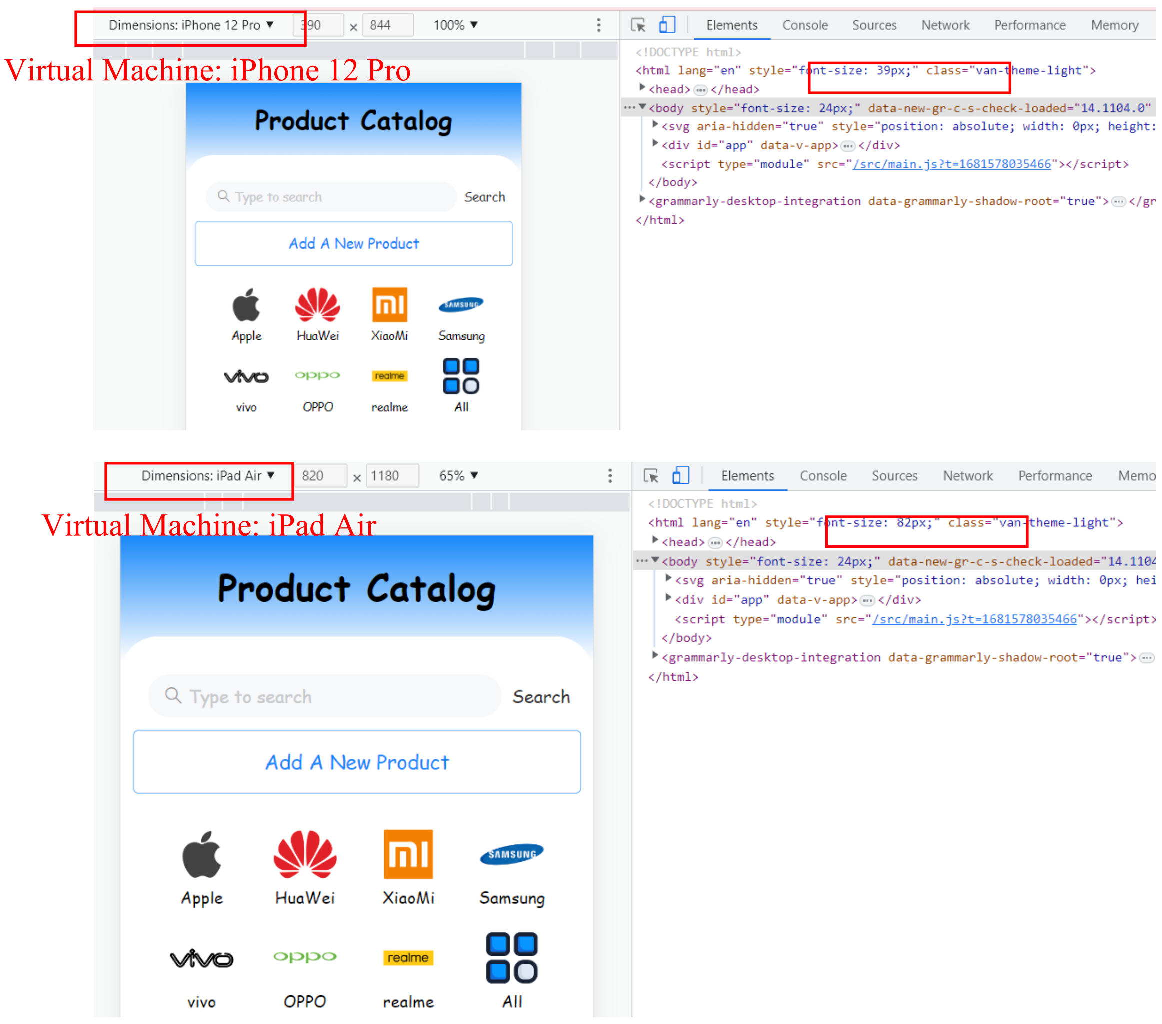
```
//Postcss-pxtorem config
css: {
  postcss: {
    plugins: [
      postCssPxToRem({
        rootValue: 37.5, // Conversion base: 1rem = 37.5px;
        propList: ['*'], // Select all attributes to be converted here
      })
    ]
  }
},
```

```
.title {
  width: 375px;
  font-size: 29px;
  font-weight: 600;
  font-family: "Comic Sans MS"
}
```

```
.home .content .header        <style>
.title[data-v-ffb78bb4] {
  width: 10rem;
  font-size: 0.77333rem;
  font-weight: 600;
  font-family: "Comic Sans MS";
}
```

Figure 4-2-1: Postcss-pxtorem Configuration

If we just converted the CSS units, it would be the same as before changing the units. If we change the size of the screen, the size of the content of the Web page will not change, which will cause the problem that the font is small even though the screen of the mobile phone is large. To solve this problem, we use postcss-pxtorem and amfe-flexible [23] together to dynamically read the screen size of the user's mobile phone. As we mentioned before, postcss-pxtorem will convert px to rem, and amfe-flexible will set 1rem to viewWidth/10, which means that dynamically convert the font size to one-tenth of the screen width. In this way, the font size of our Web app will change with the size of different mobile phone screens. The following two pictures show that in the case of different screens of different virtual machines. The font size will change with different screen sizes. For example, the font size in the iPhone 12 Pro is 39px, while the font size in the iPad Air The size is 82px.



```
<!DOCTYPE html>
<html lang="en" style="font-size: 39px;" class="van-theme-light">
 ▸<head>⋯</head>
⋯▾<body style="font-size: 24px;" data-new-gr-c-s-check-loaded="14.1104.0"
   ▸<svg aria-hidden="true" style="position: absolute; width: 0px; height:
   ▸<div id="app" data-v-app>⋯</div>
    <script type="module" src="/src/main.js?t=1681578035466"></script>
  </body>
 ▸<grammarly-desktop-integration data-grammarly-shadow-root="true">⋯</gr
</html>
```

```
<!DOCTYPE html>
<html lang="en" style="font-size: 82px;" class="van-theme-light">
 ▸<head>⋯</head>
⋯▾<body style="font-size: 24px;" data-new-gr-c-s-check-loaded="14.1104
   ▸<svg aria-hidden="true" style="position: absolute; width: 0px; hei
   ▸<div id="app" data-v-app>⋯</div>
    <script type="module" src="/src/main.js?t=1681578035466"></script>
  </body>
 ▸<grammarly-desktop-integration data-grammarly-shadow-root="true">⋯
</html>
```

Figure 4-2-2: iPhone 12 Pro vs iPad Air

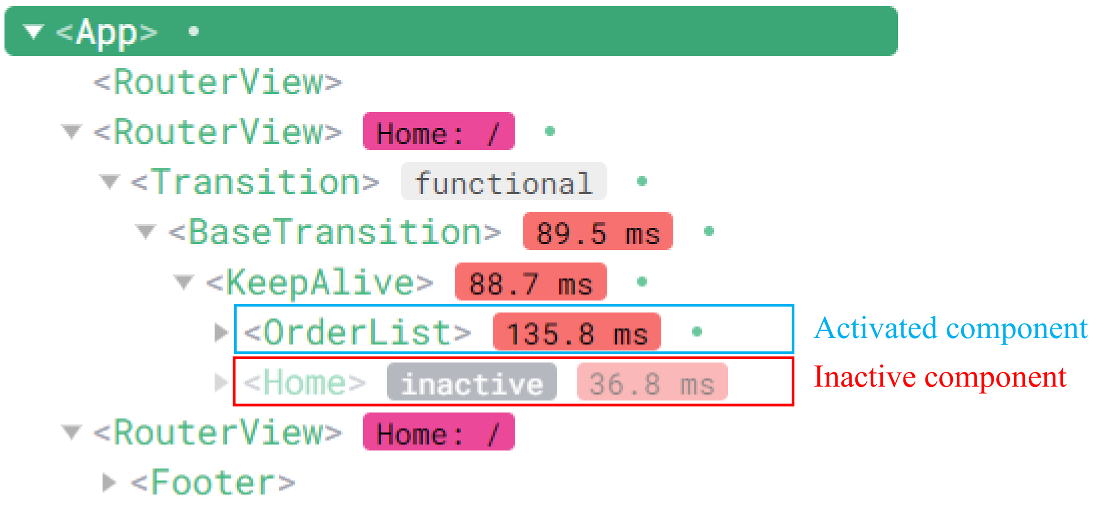
4.3 <keep-alive> Cache Components Data

By default, an active component instance in Vue will be unmounted when switching away from it. This will cause any changed state it holds to be lost. When this component is displayed again, a new instance will be created with only the initial state. This means

After the user jumps to other pages. The previous page will be destroyed, and when the user returns to the previous page again. All the operations performed by the user on the previous page will be refreshed.

Although this problem does not have a great impact on some pages, such as the product detail page, order detail page, and account page. Because on these pages, there is not much data that users can manipulate. But for some pages, if the app refreshes the pages after the jumping, it will greatly affect the user experience. For instance, a user browses the product list on the home page. If the user opens the detailed page of a product on one of the product list pages. When the user returns to the home page again, the home page will reload and display the content of the first page.

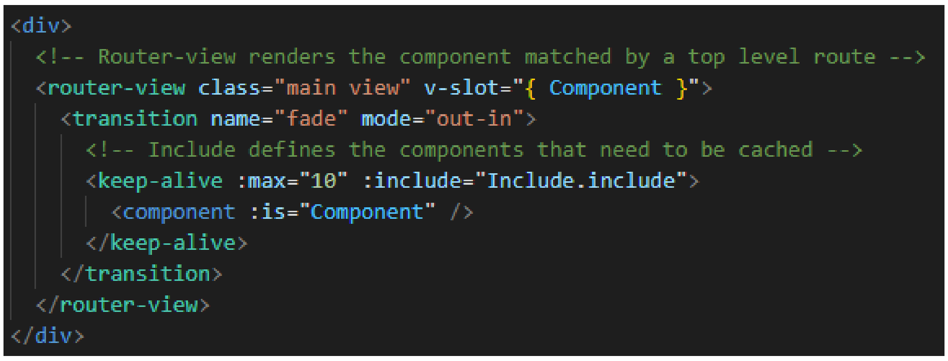
To solve this problem, we adopted <keep-alive> in the front end, which is a built-in component that allows us to conditionally cache component instances when dynamically switching between multiple components. When a component instance is removed from the DOM but is part of a component tree cached by <keep-alive>, it goes into an inactive state instead of being unmounted. When a component instance is inserted into the DOM as part of a cached tree, it will be activated. *Figure 4-3-1* illustrates the two components that are inside the <keep-alive> component. This picture represents that the user is currently on the order list page, and the home page caches the data when the user left the page for the last time.



```
▼ <App>
    <RouterView>
  ▼ <RouterView> Home: /
    ▼ <Transition> functional
      ▼ <BaseTransition> 89.5 ms
        ▼ <KeepAlive> 88.7 ms
          ▶ <OrderList> 135.8 ms
          ▶ <Home> inactive 36.8 ms
  ▼ <RouterView> Home: /
    ▶ <Footer>
```

Figure 4-3-1: Activated and inactive components

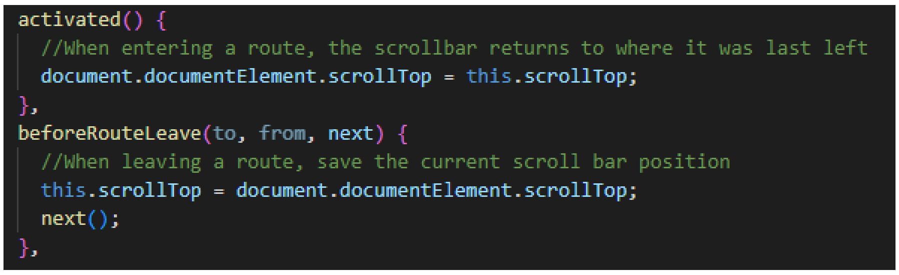
Figure 4-3-2 shows how to use <keep-alive> in our code. As you can see, <router-view> is used to render the component matched by a top-level route, which means that all main page components of our project will be rendered in it. In <keep-alive>, the "max" attribute limits the number of cached components. The purpose is to prevent the page from being stuck due to too much data cached by the browser. And the "include" attribute defines the components that need to be cached, such as the home page, order list page, and search page.



```
<div>
  <!-- Router-view renders the component matched by a top level route -->
  <router-view class="main view" v-slot="{ Component }">
    <transition name="fade" mode="out-in">
      <!-- Include defines the components that need to be cached -->
      <keep-alive :max="10" :include="Include.include">
        <component :is="Component" />
      </keep-alive>
    </transition>
  </router-view>
</div>
```

Figure 4-3-2: The use of <keep-alive>

Now, although the data of the page can be recorded and cached, <keep-alive> cannot cache the scrolling position of the page when it was left last time, so two <keep-alive> lifecycle hooks are used at this time. A <keep-alive> component can register lifecycle hooks for these two states using “activated” and “beforeRouteLeave” hooks. The “activated” hook function will be triggered every time the component is called on initial mount or it is re-inserted from the cache. On the other hand, the “beforeRouteLeave” hook function will be triggered whenever the component for the current location is about to be left. So, we can record the scrolling position of the page every time before each component jumps, and record it in the “scrollTop” which will be cached in the data of the component. When returning to this page next time, it will directly read and position the page to this position. The code is shown below in *Figure 4-3-3*.

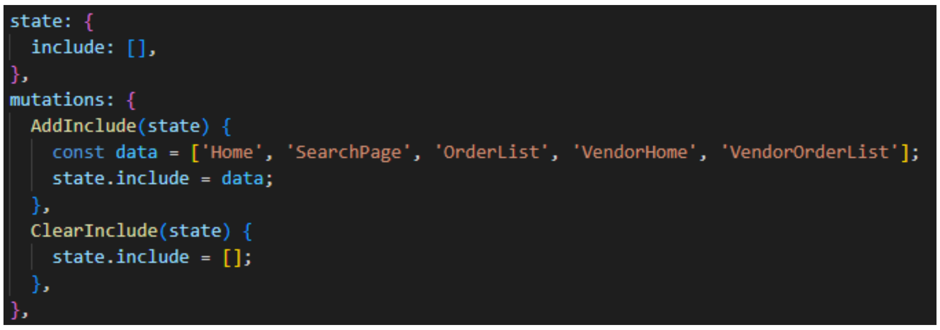


```
activated() {
  //When entering a route, the scrollbar returns to where it was last left
  document.documentElement.scrollTop = this.scrollTop;
},
beforeRouteLeave(to, from, next) {
  //When leaving a route, save the current scroll bar position
  this.scrollTop = document.documentElement.scrollTop;
  next();
},
```

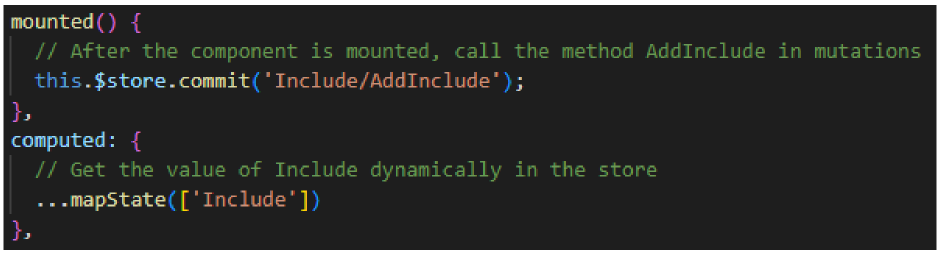
Figure 4-3-3: Recording scrollbar position

Now, although we have implemented the function of caching page data, but we will encounter another problem. Since <keep-alive> is used, the component will not be destroyed directly when leaving the page. So, when users log out or switch accounts, the page still retains the data from the last time they left, which will confuse the users. So, this time we need to clear the cache. But Vue.js only provides the method how to cache the data, it does not have a method like "clearKeepAliveCache" to clear the cache. So how to clear the cache becomes the new problem.

To solve this problem, we searched for information and found that the "include" attribute in <keep-alive> has a hidden feature. If we remove the component from the "include", the Web app will not record and delete the cached data of the component, which achieves the purpose of clearing the cache. To take advantage of this feature, we used <keep-alive> in conjunction with the Vuex we mentioned earlier. We store all component names that need to be cached in the Vuex store, and whenever the user clicks the logout button, the "include" will be cleared. And whenever any component in "include" is mounted for the first time, the names of all components will be stored in the "include". The code is shown below in *Figure 4-3-4.*



```
state: {
  include: [],
},
mutations: {
  AddInclude(state) {
    const data = ['Home', 'SearchPage', 'OrderList', 'VendorHome', 'VendorOrderList'];
    state.include = data;
  },
  ClearInclude(state) {
    state.include = [];
  },
},
```



```
mounted() {
  // After the component is mounted, call the method AddInclude in mutations
  this.$store.commit('Include/AddInclude');
},
computed: {
  // Get the value of Include dynamically in the store
  ...mapState(['Include'])
},
```

Figure 4-3-4: Implementation of "include"

4.4 UI and UX Design

Nowadays, mobile phone Apps are used in every aspect of our daily life, so user interface (UI) and user experience (UX) design is becoming more and more important. If an App is difficult to navigate or the layout of it is confusing, users will be gone in seconds. In this section, we will explain the UI or UX design used by our system.

4.4.1 Vant Component Library

Vant 4 is a lightweight, customizable Vue UI library and it contains more than 70 Vue UI components for building mobile Apps. The library is very lightweight with an average size of 1kb per component (min+gzip). This will greatly reduce the rendering time of the component and bring a good user experience to the customer. Vant also has the ability of theme customization, online theme preview tool, internationalization, and so on. It can organize many different styles through rich CSS variables to suit many different types of customers. The styles of the various components of Vant also conform to the UI design of the mobile Web app.

One example of the UI design of the Vant component library catering to the mobile user experience is "SwipeCell". As shown in *Figure 4-4-1-1* below, if the user wants to delete the item from the shopping cart, they can slide left and click the "Delete" button

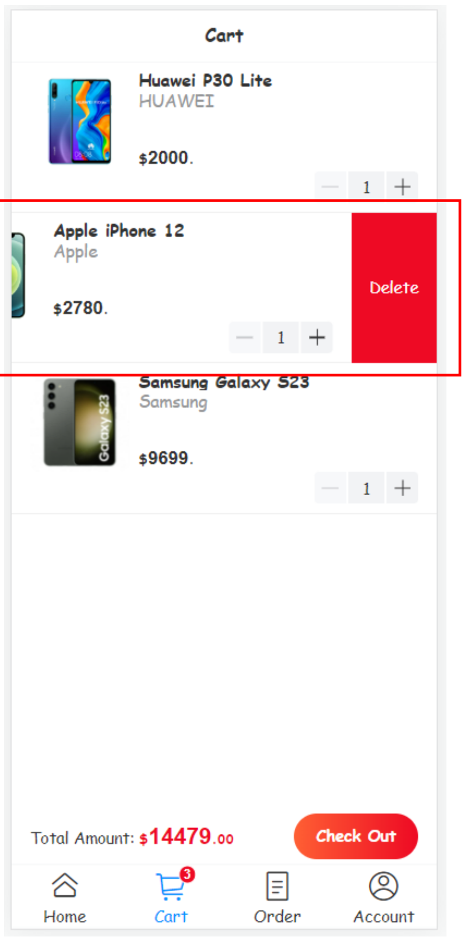


Figure 4-4-1-1: SwipeCell

4.4.2 Visual Hierarchy in UX

Visual hierarchy is a clear visual hierarchy guides the eye to the most important elements on the page. It can be created through variations in color and contrast, scale, and grouping. The iOS design specification states that the current label should be highlighted. This is because when users shop on their mobile phones, basically all the information is obtained through their eyes. In the unconscious perception of visual depth, the user can judge the closest object to himself and know where he is. Therefore, in the design of the tab bar, the current tab usually needs to be highlighted, while other tabs should be appropriately weakened. In this way, the current page label and other labels can be visually separated from each other. This enables users to know their location more conveniently and intuitively [24].

Figure 4-4-2-1 displays the tab bar in our project, we not only use highlighting to show the current location of the user, but also add a badge to display the quantity in the upper right corner of the shopping cart icon.

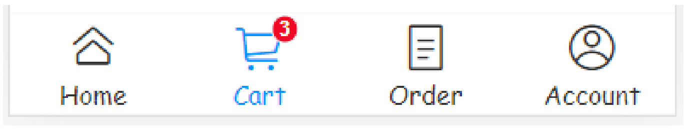


Figure 4-4-2-1: Tabbar

4.4.3 Gulf of Execution and Evaluation

Gulf of execution [24] is the degree of ease with which a user can understand the current state of a system. It is the difference between the intentions of the users and what the system allows them to do. For example, a person can easily determine the current state of the system (whether the light is on or off) and how to operate the switch by looking at a light switch.

Gulf of evaluation [24] is the degree of ease with which a user can perceive and interpret whether or not the action they performed was successful. This gulf is small when the system provides information about its state in a form that is easy to receive, interpret, and matches the way the person thinks of the system.

In our project, we use blue icons with thumbs-up to represent "Like", and conversely, red icons with thumbs-down to represent "Dislike". At first, the user is in an unselected state. If he likes the item, he can click the blue thumbs-up icon. After clicking, a green message prompt with "Like It!" will appear at the top of the screen, and the icon will change to a red icon with a thumbs-down. At this time, it means that the user likes the product. If the user taps the red icon again, a red message with "Don't Like It!" will appear at the top, and it will change back to a blue icon. At this time, it is a state of dislike. As shown in the following two figures.

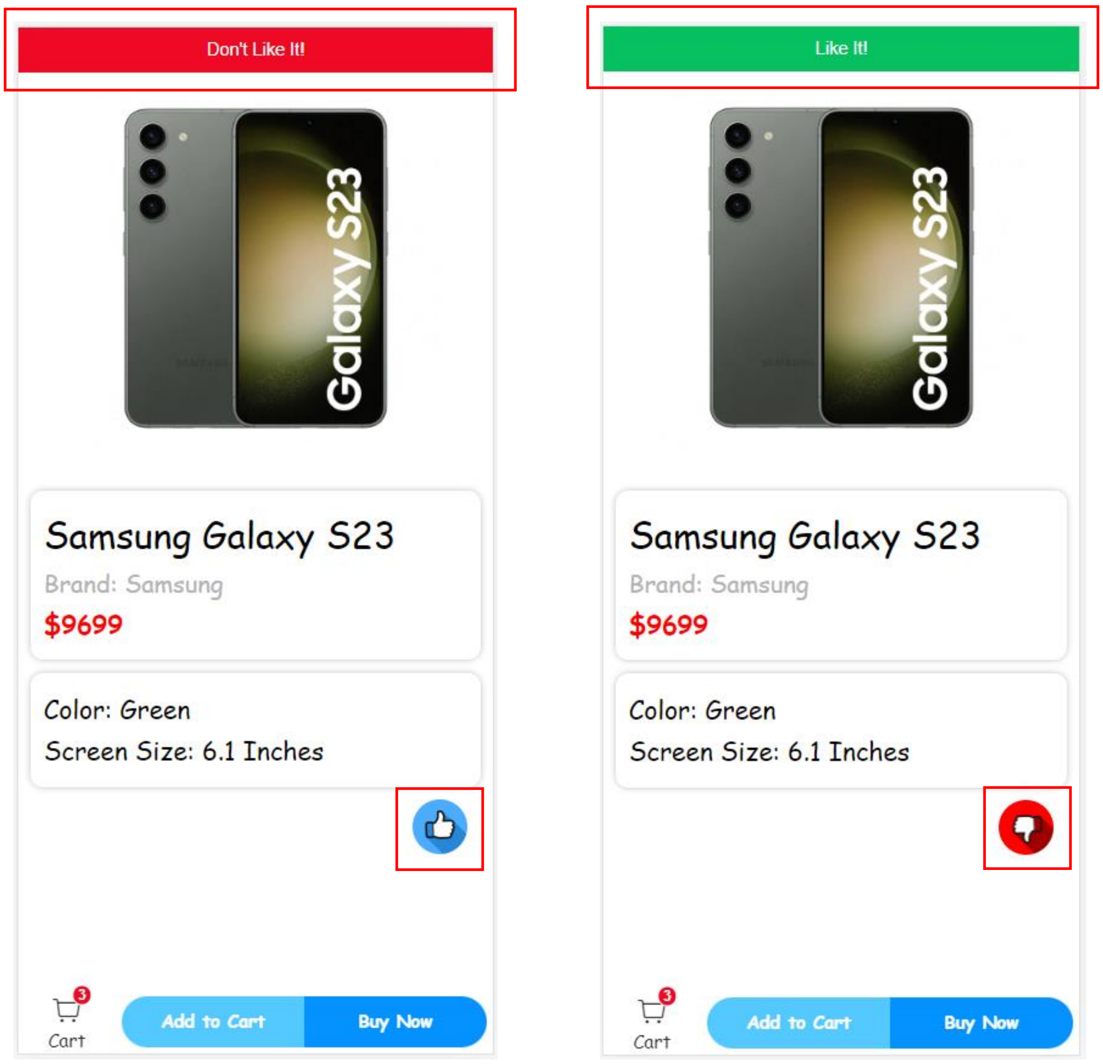


Figure 4-3-3-1: Like and Dislike

Therefore, our design can not only represent the user's status at this time through vivid icons, but also let users know what their operations mean and whether they are successful or not through message feedback in different colors.

4.5 Why We Choose Spring Boot for the Backend

In school, we have learned two different programming languages: Python and Java. And we even study Django as backend framework which is based on Python. After looking through the Internet, we find out Spring Boot is also a good choice for us since it is a server-side Java framework.

Before implementation, we make comparison between these two frameworks in order to choose the most suitable one. Frist of all, in terms of performance, we found Spring Boot is faster in running the code as it is written in java comparing to Django. Since Google recommends fast websites and it also affects a website's ranking, this is one of the most crucial factors we choose Spring Boot [25].

Secondly, we found that Django could only handle one request at once. However, Spring Boot is able to handle multiple requests at once. Using Spring Boot as backend will shorten the waiting time of users. In other words, it will promote the brilliant experience of users.

What's more, as we know, Django and Spring Boot are both open-source technologies. According to statistics, Spring Boot has 25.8K GitHub forks and 39.8K GitHub stars, while Django has 42,000 ratings and 18,000 forks [25]. From above, Spring Boot seems more popular than Django, which means we could get more reference and help benefiting from this. (Moreover, the responsible person is more familiar with Java.)

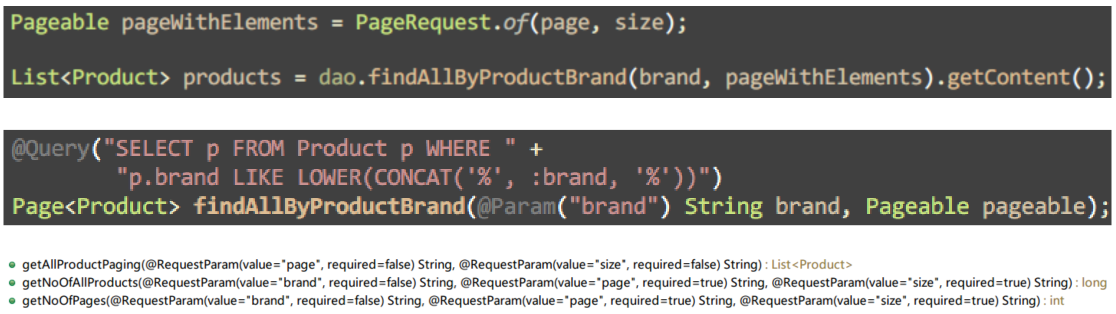
So, these are the reasons why we choose Spring Boot as the framework of our backend.

4.6 Pagination

Pagination, also known as paging, is the process of dividing a Web content into discrete pages. According to the requirement specification, products are displayed in multiple pages.

There are mainly two approaches of implementing pagination: client-side pagination and server-side pagination. They are simply slicing the whole product list either on the server or on the client. Our solution is the hybrid of both server-side and client-side pagination in the homepage and in the search page.

In the homepage, we use server-side pagination. Because the mass products may consume much bandwidth. Only few items are displayed at once in server-side pagination mode, and as the change of pages, the data of products are retrieved on demand. *Figure 4-6-1* illustrates how paging functionality is utilized provided by JPA Repository and Spring framework.

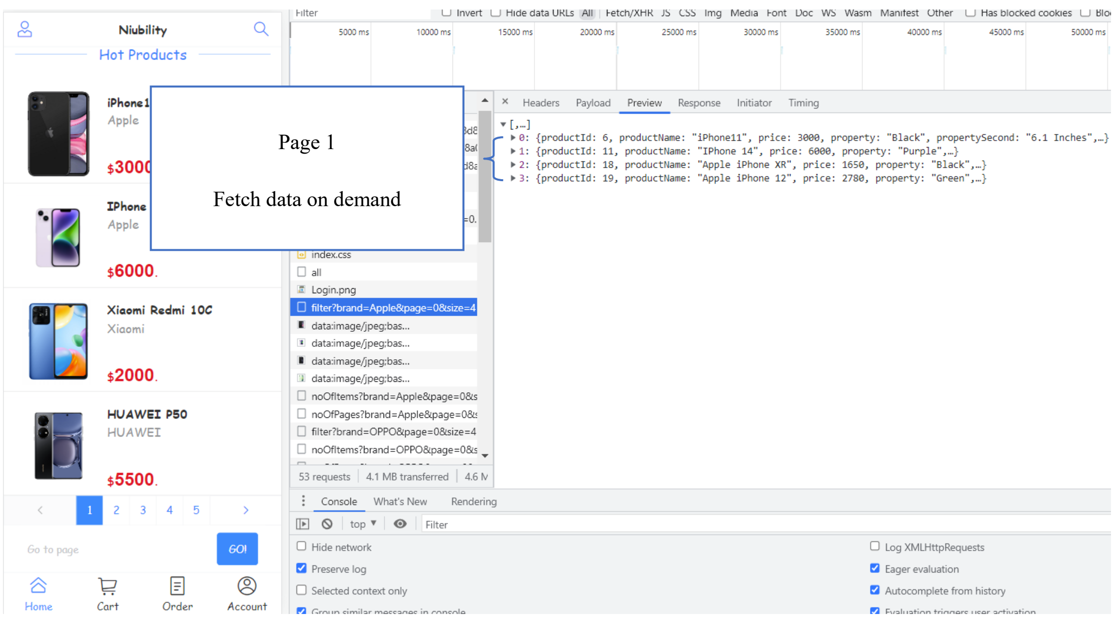


```
Pageable pageWithElements = PageRequest.of(page, size);

List<Product> products = dao.findAllByProductBrand(brand, pageWithElements).getContent();
```

```
@Query("SELECT p FROM Product p WHERE " +
        "p.brand LIKE LOWER(CONCAT('%', :brand, '%'))")
Page<Product> findAllByProductBrand(@Param("brand") String brand, Pageable pageable);
```

- getAllProductPaging(@RequestParam(value="page", required=false) String, @RequestParam(value="size", required=false) String) : List<Product>
- getNoOfAllProducts(@RequestParam(value="brand", required=false) String, @RequestParam(value="page", required=true) String, @RequestParam(value="size", required=true) String) : long
- getNoOfPages(@RequestParam(value="brand", required=false) String, @RequestParam(value="page", required=true) String, @RequestParam(value="size", required=true) String) : int

Figure 4-6-1 Server-side pagination implementation



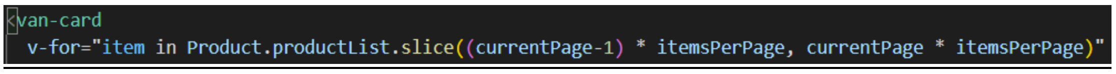
Page 1

Fetch data on demand

```
▼ [,…]
  ▶ 0: {productId: 6, productName: "iPhone11", price: 3000, property: "Black", propertySecond: "6.1 Inches",…}
  ▶ 1: {productId: 11, productName: "IPhone 14", price: 6000, property: "Purple",…}
  ▶ 2: {productId: 18, productName: "Apple iPhone XR", price: 1650, property: "Black",…}
  ▶ 3: {productId: 19, productName: "Apple iPhone 12", price: 2780, property: "Green",…}
```

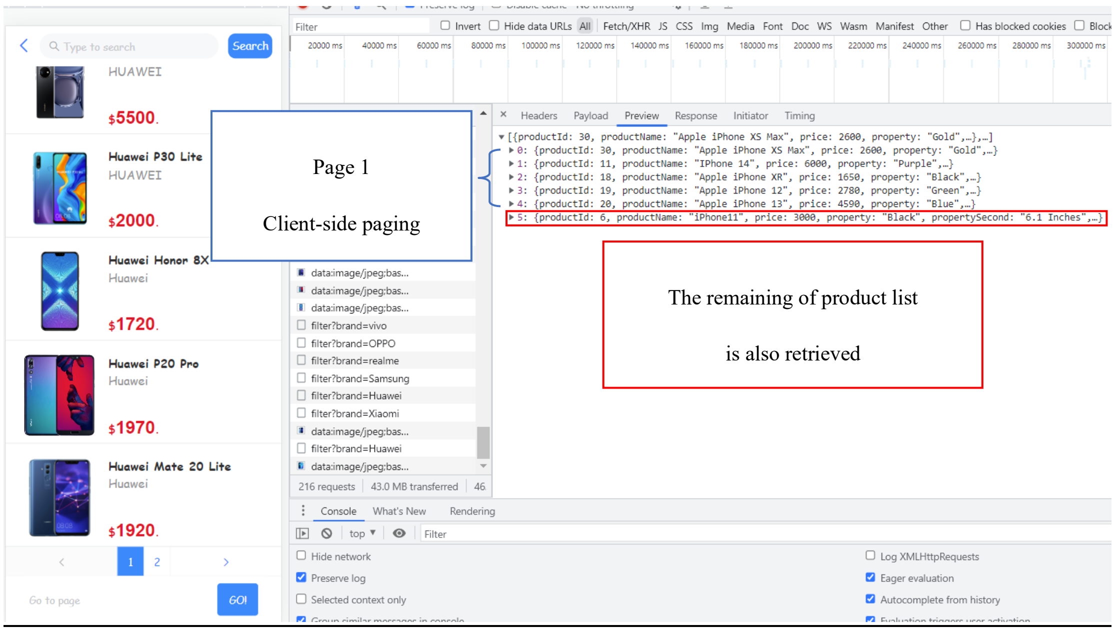
Figure 4-6-2 Server-side pagination

In the search page, we assume much smaller cardinality of data set is returned. Another assumption is switching speed among is highly required when the user is searching and filtering by brand simultaneously. Because the user tends to go through all products fast to look for the product that the user wants. Therefore, we decide to apply client-side pagination in the search page. The entire list of searching results is returned first, the paging and filtering are done by client-side program subsequently. It is believed that using client-side pagination here increases the responsiveness of the search page when the user is quickly switches among pages. In *Figure4-6-3*, products are sliced just before rendering.



```
<van-card
  v-for="item in Product.productList.slice((currentPage-1) * itemsPerPage, currentPage * itemsPerPage)"
```

Figure 4-6-3 Client-side pagination implementation



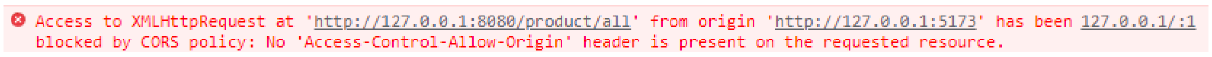
```
▼[{productId: 30, productName: "Apple iPhone XS Max", price: 2600, property: "Gold",…},…]
  ▶0: {productId: 30, productName: "Apple iPhone XS Max", price: 2600, property: "Gold",…}
  ▶1: {productId: 11, productName: "IPhone 14", price: 6000, property: "Purple",…}
  ▶2: {productId: 18, productName: "Apple iPhone XR", price: 1650, property: "Black",…}
  ▶3: {productId: 19, productName: "Apple iPhone 12", price: 2780, property: "Green",…}
  ▶4: {productId: 20, productName: "Apple iPhone 13", price: 4590, property: "Blue",…}
  ▶5: {productId: 6, productName: "iPhone11", price: 3000, property: "Black", propertySecond: "6.1 Inches",…}
```

Figure 4-6-4 Client-side pagination

4.7 CORS

Cross-origin resource sharing (CORS) is a mechanism which restricts access of resources sent by a different domain. According to the *same-origin security* policy forbids cross-origin access, for instance, Ajax call, to another origin to resources [26] [27].

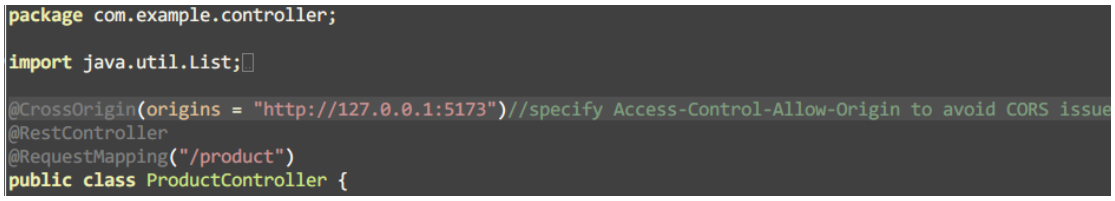
It is where the problem emerged. In our architecture design (refer to *Figure 4-1-1*: deployment diagram), our Web server (running on port 5173) and API server (running on port 8080) are separate as loosely coupled modules. It allows us to develop View-controller and Model-controller concurrently and conveniently. In our development environment, they are running in different ports. However, the cross-origin Ajax are forbidden so that our front-end JavaScript cannot access the API.



```
Access to XMLHttpRequest at 'http://127.0.0.1:8080/product/all' from origin 'http://127.0.0.1:5173' has been 127.0.0.1/:1
blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
```

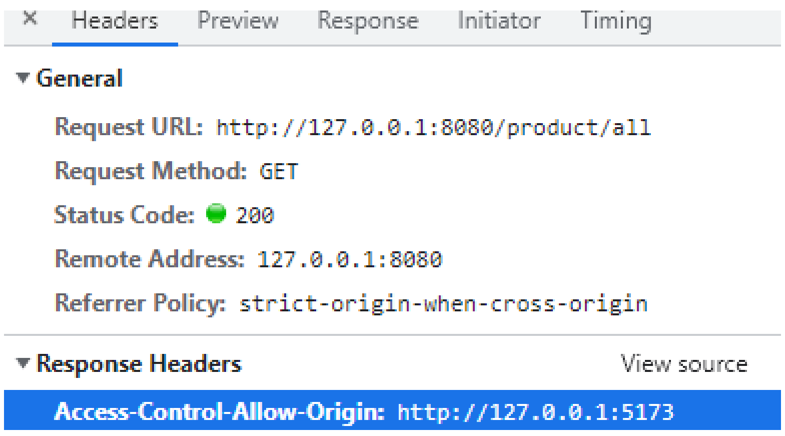
Figure 4-7-1. Without adding the control header

To solve the problem, we utilized an annotation provided by Spring framework. It adds Access-Control-Allow-Origin HTTP header to HTTP responses requested by our Web server.



```
package com.example.controller;

import java.util.List;

@CrossOrigin(origins = "http://127.0.0.1:5173")//specify Access-Control-Allow-Origin to avoid CORS issue
@RestController
@RequestMapping("/product")
public class ProductController {
```

Figure 4-7-2. @CrossOrigin annotation



```
▼ General
    Request URL: http://127.0.0.1:8080/product/all
    Request Method: GET
    Status Code: ● 200
    Remote Address: 127.0.0.1:8080
    Referrer Policy: strict-origin-when-cross-origin
▼ Response Headers
    Access-Control-Allow-Origin: http://127.0.0.1:5173
```

Figure 4-7-3. Access-Control-Allow-Origin header

After configuring the API server, we successfully overcome this technical problem.

4.8 Search

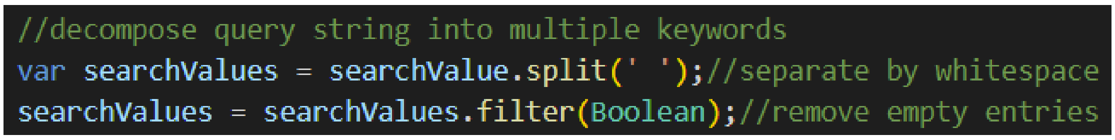
In consideration of user experience, the search function of the App should be efficient, accurate, and highly available.

4.8.1 Automatic Submission

Although typing one more word will send another request, which consumes a lot more data traffic than sending the request till the user tabs "Search" button, we still want to keep the function due to high performance and availability. We reckon when the user is searching instead of browsing, there must be some intention or aim. Therefore, the sooner the product is found or result is displayed, the sooner requirements or intention is satisfied or resolved. Such functionality greatly reduces the probability of causing anxiety when a customer cannot find a certain item in a short period of time.

4.8.2 Keyword Processing

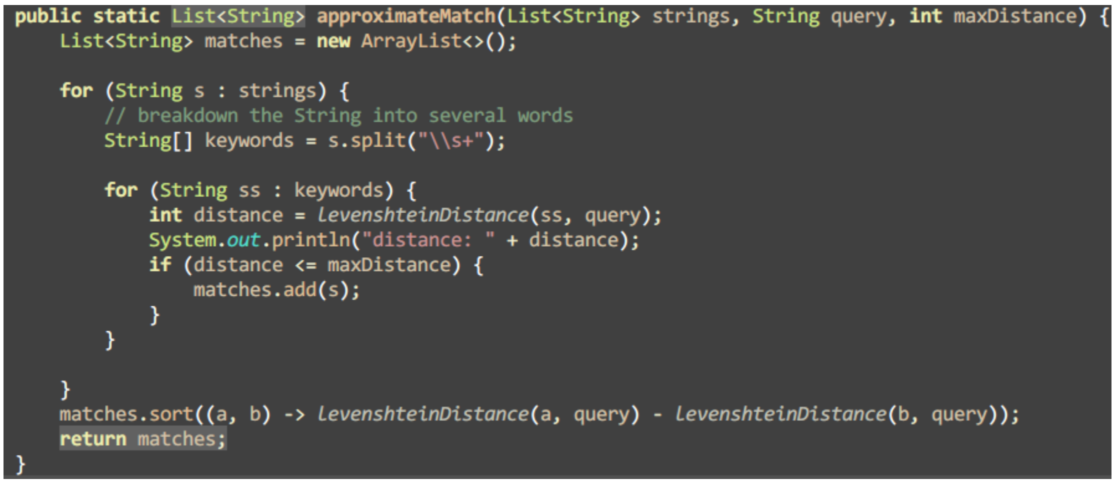
At the frontend, the input text is decomposed for searching in finer granularity.



```
//decompose query string into multiple keywords
var searchValues = searchValue.split(' ');//separate by whitespace
searchValues = searchValues.filter(Boolean);//remove empty entries
```

Figure 4-8-2-1. Split the User Input Text

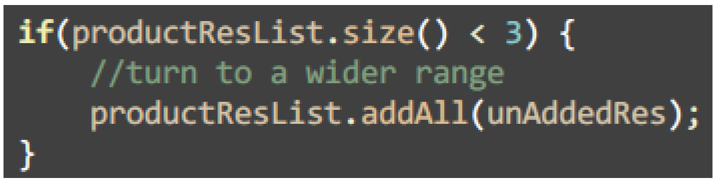
At the backend, product names are decomposed for more exact matching through searching keywords.



```
public static List<String> approximateMatch(List<String> strings, String query, int maxDistance) {
    List<String> matches = new ArrayList<>();

    for (String s : strings) {
        // breakdown the String into several words
        String[] keywords = s.split("\\s+");

        for (String ss : keywords) {
            int distance = LevenshteinDistance(ss, query);
            System.out.println("distance: " + distance);
            if (distance <= maxDistance) {
                matches.add(s);
            }
        }

    }
    matches.sort((a, b) -> LevenshteinDistance(a, query) - LevenshteinDistance(b, query));
    return matches;
}
```

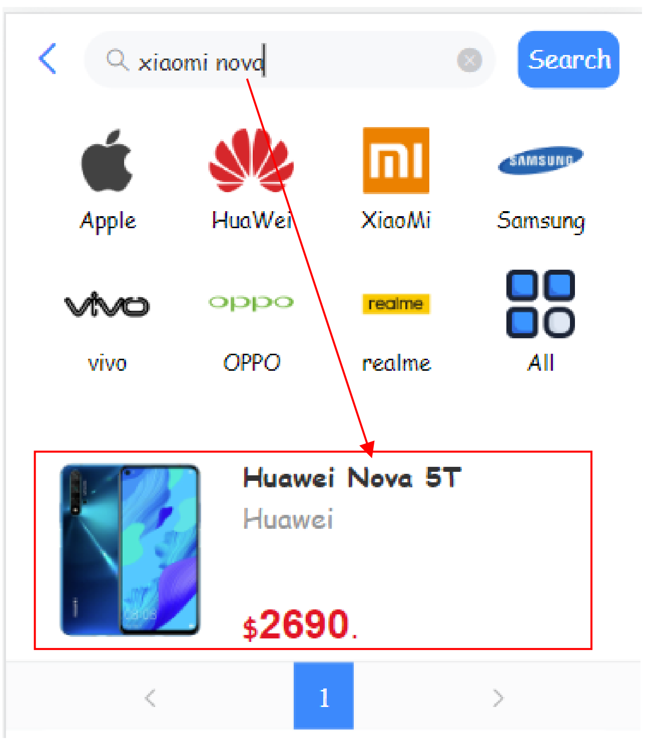
Figure 4-8-2-2. Approximate match function

In addition, we have eased the condition from satisfying all keywords to satisfying either one of them to ensure availability to a certain extent.



```
if(productResList.size() < 3) {
    //turn to a wider range
    productResList.addAll(unAddedRes);
}
```

Figure 4-8-2-3. Extend result list when there are few results found



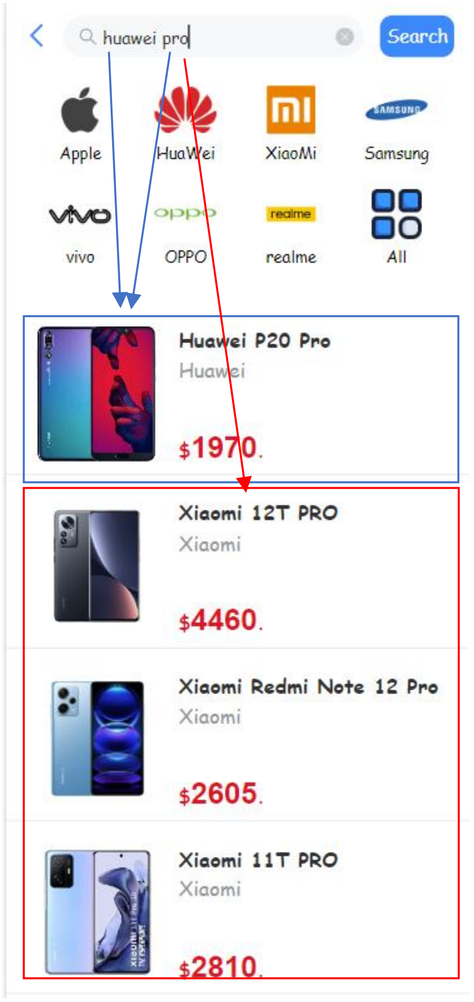
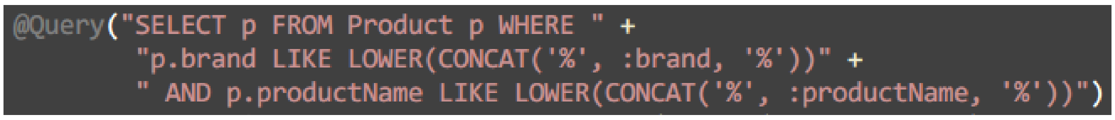


Figure 4-8-2-4. Complementary Results

For instance, in *Figure 4-8-2-4 (left)*. If the user types "huawei pro". In our test dataset, there is only 1 product that simultaneously match "huawei" and "pro". Besides, "Pro" model of "Xiaomi" also appears as the complementary of the result. Another example demonstrated in *Figure 4-8-2-4 (right)*. Because there are only 1 result that match both 2 keywords ("huawei", and "pro"), extra results are also returned even if they only match the keyword "pro". Users are offered with flexibility so that they are encouraged to try to input everything they tend to obtain – even if there is no "Nova" model of brand "Xiaomi".

4.8.3 Fuzzy Search

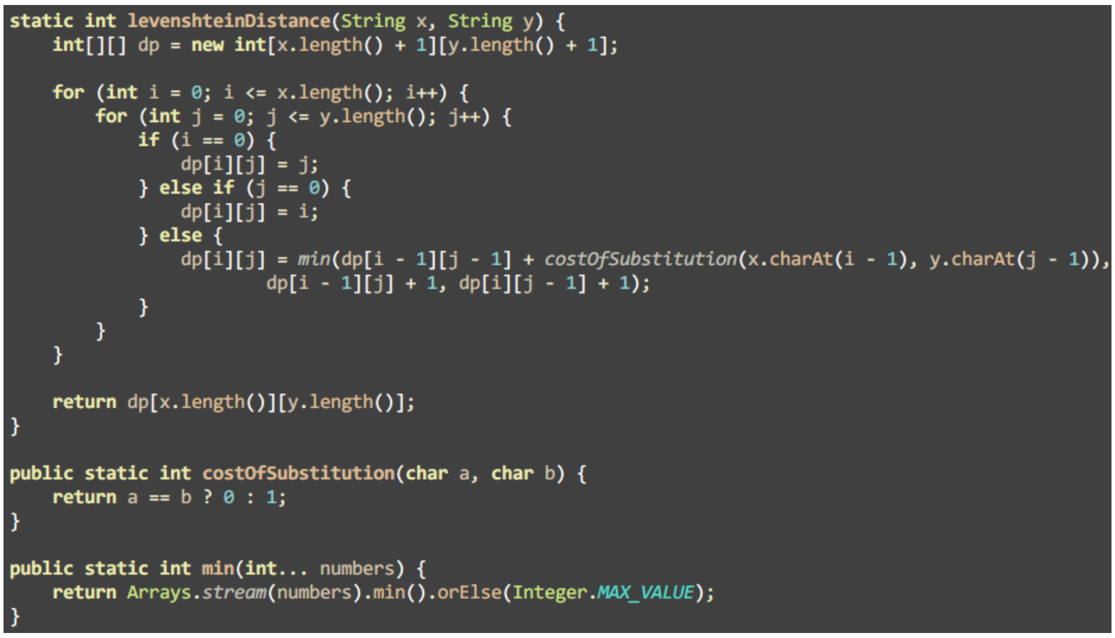
Initially, our search function is simply implemented with a SQL statement to fetch matching records from MySQL database with the help of Query function provided by JPA Repository. The LIKE keyword and LOWER keyword enable searching by product name through a substring of it case-insensitively. For example, after the user typing “hua”, “Huawei P50” will be one record of the searching results provided to the user.



```java
@Query("SELECT p FROM Product p WHERE " +
        "p.brand LIKE LOWER(CONCAT('%', :brand, '%'))" +
        " AND p.productName LIKE LOWER(CONCAT('%', :productName, '%'))")
```

Figure 4-8-3-1. @Query annotation

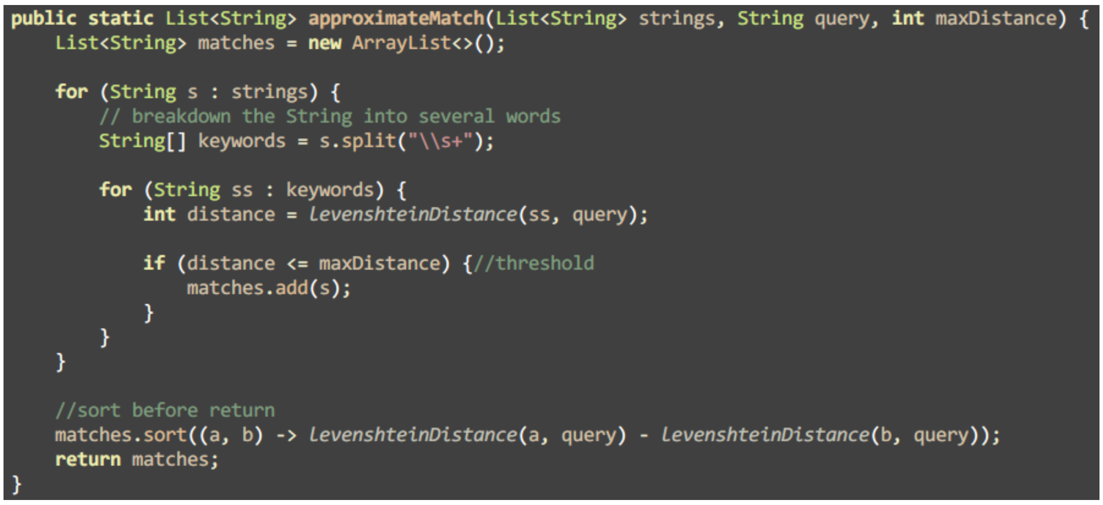
However, it’s a common situation that users sometimes type a misspelt word unconsciously. In order to resolve the requirements, we have developed a simple fuzzy match algorithm. Similar to Hamming distance, Levenshtein Distance is exploited for evaluating the similarity between two strings. In other word, it is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other [28]. The code of calculating Levenshtein Distance between two strings are showed in *Figure 4-8-3-2*.



```java
static int levenshteinDistance(String x, String y) {
    int[][] dp = new int[x.length() + 1][y.length() + 1];

    for (int i = 0; i <= x.length(); i++) {
        for (int j = 0; j <= y.length(); j++) {
            if (i == 0) {
                dp[i][j] = j;
            } else if (j == 0) {
                dp[i][j] = i;
            } else {
                dp[i][j] = min(dp[i - 1][j - 1] + costOfSubstitution(x.charAt(i - 1), y.charAt(j - 1)),
                        dp[i - 1][j] + 1, dp[i][j - 1] + 1);
            }
        }
    }

    return dp[x.length()][y.length()];
}

public static int costOfSubstitution(char a, char b) {
    return a == b ? 0 : 1;
}

public static int min(int... numbers) {
    return Arrays.stream(numbers).min().orElse(Integer.MAX_VALUE);
}
```

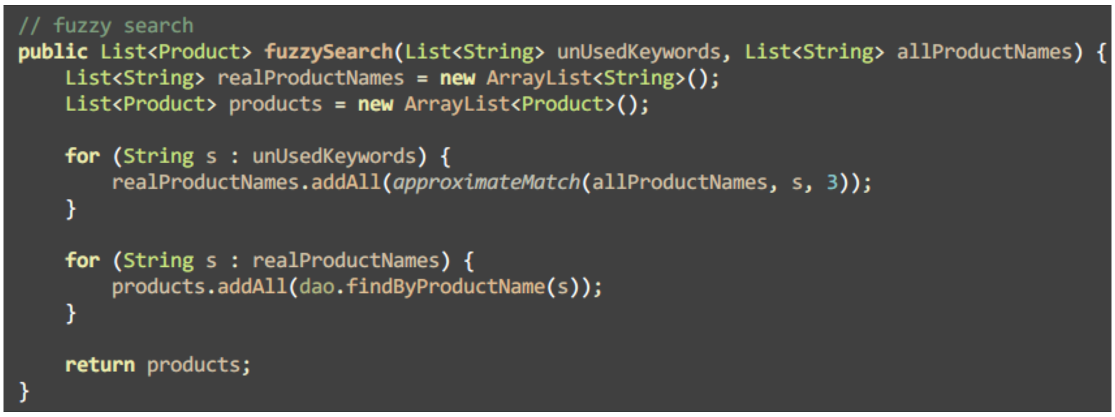
Figure 4-8-3-2. Levenshtein distance algorithm

The fuzzy match function is implemented by restricting the maximum Levenshtein Distance (we set it to 3) as the threshold, and return the results in ascending order of Levenshtein Distance between keywords that the user has input and keywords from product names. Finally, we successfully implemented a memory-based fuzzy search function.



```java
public static List<String> approximateMatch(List<String> strings, String query, int maxDistance) {
    List<String> matches = new ArrayList<>();

    for (String s : strings) {
        // breakdown the String into several words
        String[] keywords = s.split("\\s+");

        for (String ss : keywords) {
            int distance = LevenshteinDistance(ss, query);

            if (distance <= maxDistance) {//threshold
                matches.add(s);
            }
        }
    }

    //sort before return
    matches.sort((a, b) -> LevenshteinDistance(a, query) - LevenshteinDistance(b, query));
    return matches;
}
```

Figure 4-8-3-3. Approximate match function



```java
// fuzzy search
public List<Product> fuzzySearch(List<String> unUsedKeywords, List<String> allProductNames) {
    List<String> realProductNames = new ArrayList<String>();
    List<Product> products = new ArrayList<Product>();

    for (String s : unUsedKeywords) {
        realProductNames.addAll(approximateMatch(allProductNames, s, 3));
    }

    for (String s : realProductNames) {
        products.addAll(dao.findByProductName(s));
    }

    return products;
}
```

Figure 4-8-3-4. Fuzzy search

Because the fuzzy search algorithm is at least *O(n^2)*, we try to reduce the number of accesses. The exact match function is applied at first for a keyword decomposed from user input. If there is no results found, the keyword will be added to an auxilliary array. After adding all the results retrieved from exact match function, fuzzy match will be applied accordingly for the unused keywords. By this mean, the correct or partially correct keywords (substrings of correct product name) are not fuzzily searched, thus the performance of our search function has increased.

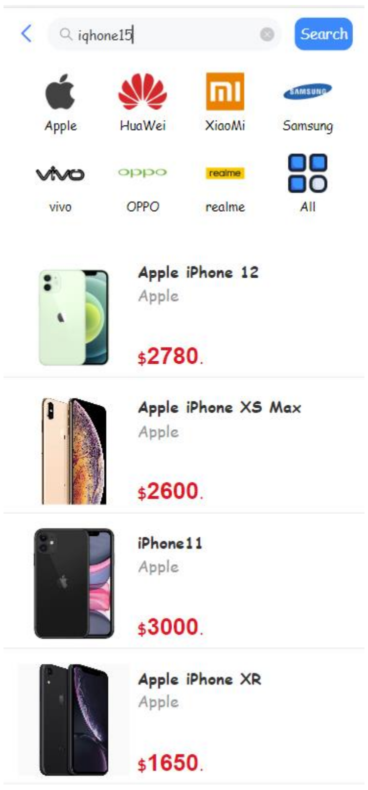


Figure 4-8-3-5. Demonstration of fuzzy search

It is illustrated in *Figure 4-8-3-5* that "iqhone15" is a typical example of misspelt keyword and an advent of non-existing model (by far, the latest version of iPhone is iPhone 14). The outcome of returning all products related to iPhone satisfies the fuzzy match requirement.

4.9 Personalized Recommendation

Recommendation algorithm is a crucial functionality in nowadays e-commerce Apps. It is stated that 63% of smartphone users are more likely to purchase from mobile apps offer them relevant recommendations [28]. We have applied a simple recommendation algorithm to generate 3 products that a customer may favor.

A like or dislike is a direct index which determines whether a customer likes a product or not. However, we still want to estimate how likely a customer would prefer a product without existing feedback from that customer. We are inclined to provide several recommendations to the customer … So that increase the likelihood for the customer to purchase a product.

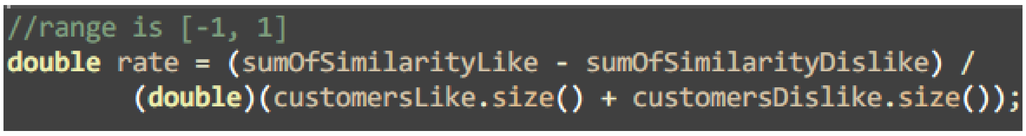
The algorithm is based on Jaccard index. Formula 1 calculates similarity between 2 users given the products they like (L_1, L_2) and dislike (D_1, D_2). Formula 2 derives the probability of a customer preferring a product from the similarity of the user and the users who have rated that product. For in-depth explanation for the algorithm, please refer to [29].In brief, it is assumed that a customer may like the product which is favored by similar customers.

$$S(U_1,U_2) = (|L_1 \cap L_2| + |D_1 \cap D_2| - |L_1 \cap D_2| - |L_2 \cap D_1|) \div |L_1 \cup L_2 \cup D_1 \cup D_2|$$

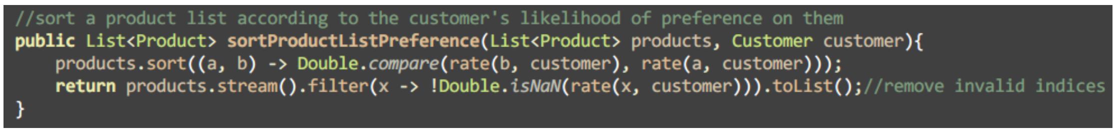
$$P(U,M) = (Z_L - Z_D) \div (|M_L| + |M_D|)$$

Figure 4-9-1. Jaccard-index-based recommendation algorithm

A part of the implementation in Java is demonstrated in *Figure 4-9-1*. In development, we noticed that if a product has never been liked or disliked by any user, the denominator will be zero, which contributes a *NaN* in Java. It is filtered out (*Figure 4-9-2*) in our consideration for valid and reasonable recommendation.



```java
//range is [-1, 1]
double rate = (sumOfSimilarityLike - sumOfSimilarityDislike) /
        (double)(customersLike.size() + customersDislike.size());
```



```java
//sort a product list according to the customer's likelihood of preference on them
public List<Product> sortProductListPreference(List<Product> products, Customer customer){
    products.sort((a, b) -> Double.compare(rate(b, customer), rate(a, customer)));
    return products.stream().filter(x -> !Double.isNaN(rate(x, customer))).toList();//remove invalid indices
}
```

Figure 4-9-2. Recommend base on preferences

Based on that, we push the top 3 products that are most likely to be favored to a customer and display them at the top of product list in the homepage.

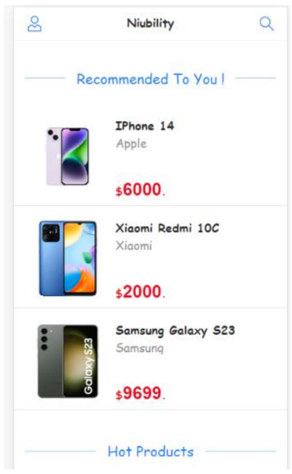


Figure 4-9-3. Top 3 recommendations to display

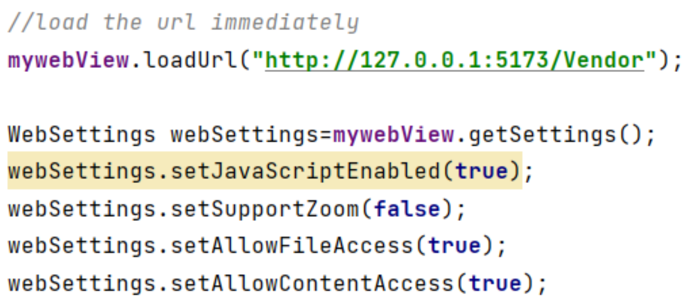
To evaluate the effectiveness of the recommendation algorithm, we have conducted a test case. For the details, please refer to **section 5.2**

4.10 Android

4.10.1 Android WebView

WebView (also called embedded browser control) is like an embedded Web browser in a native app to display Web content.

Implementing the mobile App by means of mobile Web App + Android *WebView*, the development process is more agile compared with traditional native App development. Also, we are able to extend our Web App so that it can be more powerful, for example, access to Android API and full-control over the program (JavaScript, zooming, file access, etc.).



```
//load the url immediately
mywebView.loadUrl("http://127.0.0.1:5173/Vendor");

WebSettings webSettings=mywebView.getSettings();
webSettings.setJavaScriptEnabled(true);
webSettings.setSupportZoom(false);
webSettings.setAllowFileAccess(true);
webSettings.setAllowContentAccess(true);
```

Figure 4-10-1 Android WebView

4.10.2 File Upload

Because Android *WebView* doesn't support some features that a fully-developed browser has. File upload function is unavailable directly using <*input*> tag in HTML. As showed in *Figure 4-10-2*, the user is able to upload file by using a web browser, but it didn't work in our App using WebView.

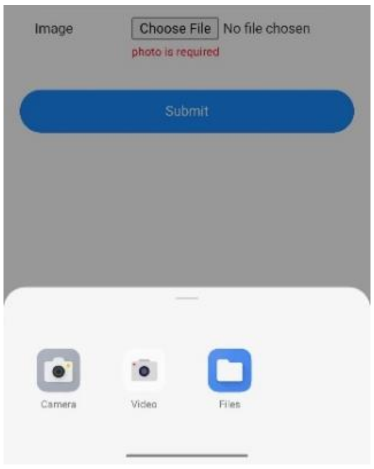
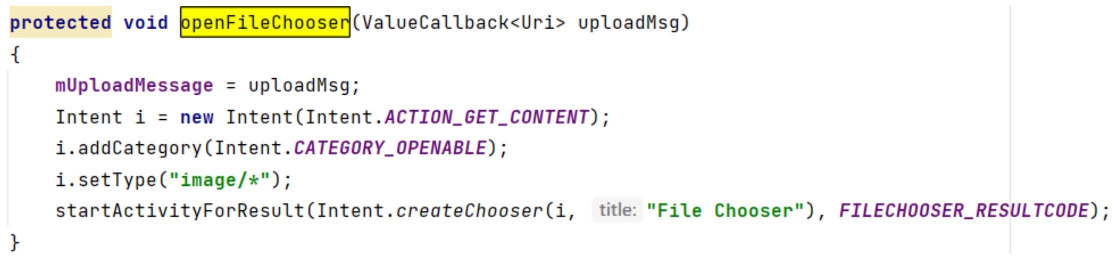


Figure 4-10-2. File access in Google Chrome

To fix the issue, Android File Chooser function is involved. When the user tabs on the button, the system will automatically open the file chooser on the user's Android phone so that they can choose an image from the system file explorer.



```
protected void openFileChooser(ValueCallback<Uri> uploadMsg)
{
    mUploadMessage = uploadMsg;
    Intent i = new Intent(Intent.ACTION_GET_CONTENT);
    i.addCategory(Intent.CATEGORY_OPENABLE);
    i.setType("image/*");
    startActivityForResult(Intent.createChooser(i, title: "File Chooser"), FILECHOOSER_RESULTCODE);
}
```

Figure 4-10-3. File chooser API

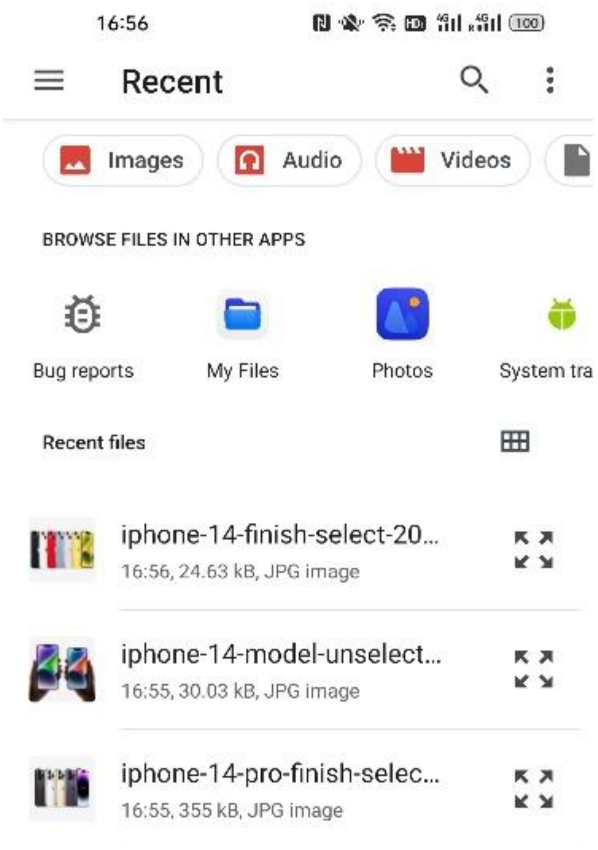
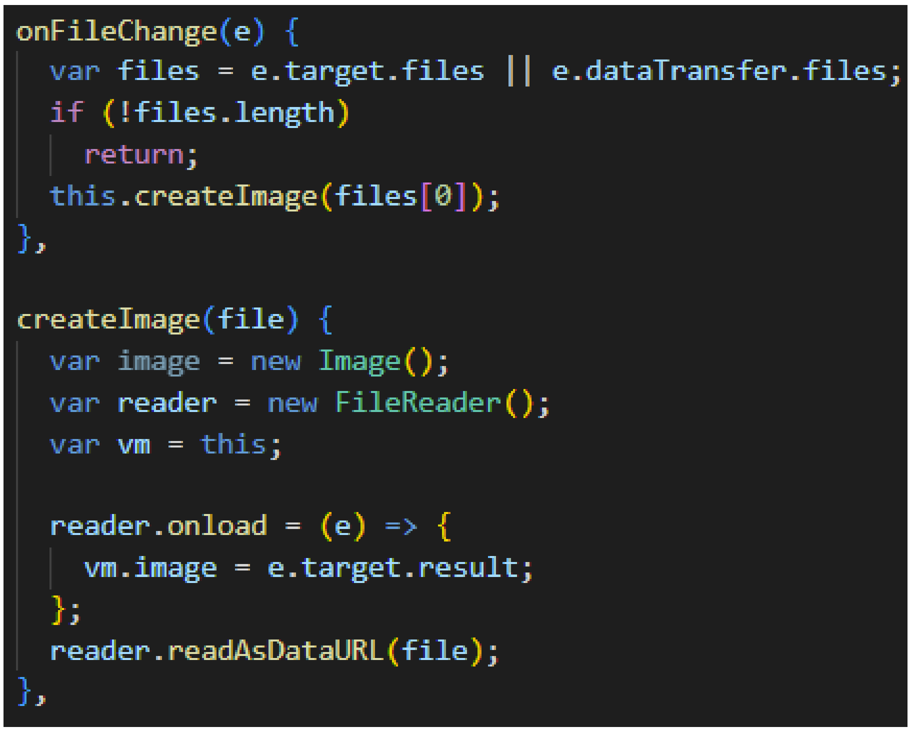

16:56
Recent
Images
Audio
Videos
BROWSE FILES IN OTHER APPS
Bug reports
My Files
Photos
System tra
Recent files
iphone-14-finish-select-20...
16:56, 24.63 kB, JPG image
iphone-14-model-unselect...
16:55, 30.03 kB, JPG image
iphone-14-pro-finish-selec...
16:55, 355 kB, JPG image

Figure 4-10-4. System file explorer of Android

After the user uploading an image, the image will be converted to Base64 format and send to the server.



```
onFileChange(e) {
  var files = e.target.files || e.dataTransfer.files;
  if (!files.length)
    return;
  this.createImage(files[0]);
},

createImage(file) {
  var image = new Image();
  var reader = new FileReader();
  var vm = this;

  reader.onload = (e) => {
    vm.image = e.target.result;
  };
  reader.readAsDataURL(file);
},
```

Figure 4-10-2. Convert image to Base64

5. Results and Discussion

In this chapter, we have two sections. The first section is Project Outcome, where we would like to talk about major functions of Niubility (our mobile shopping App). In the second section, we mainly use functional testing to check the effectiveness of the recommendation mechanism and debug on a real Android phone.

5.1 Project Outcome

In our project, we have completed a total of 21 functional requirements and some additional functions related to customers and vendors. For the customer, in the requirements of showing the product list, we have completed A1, A2, A3, A4, and A6. In the requirements of account management, we have completed B1 and B2. In the functional requirements of the shopping cart, we have completed C1, C2, C3, and C5. Finally, in the requirements of purchase tracking, we have completed D1, D2, and D3. For the vendor, we have completed E1, E2, and E3 in the requirements of product catalog maintenance. In the requirements of purchase order processing, we have completed F1, F2, F3, and F4. In addition to these 21 functions, we also completed other additional functions, such as fuzzy search, recommended function, and page cache function. In the following, we will talk about these functions and their outcome in detail.

5.1.1 Customer: Product list page with recommendation, filtering, searching, and pagination

When the customer opens Niubility, he will directly enter the product list page (home page as well). Before some customers stamped a variety of products as “like” or “dislike”, so the algorithm would use the previous data to calculate the result of which product this customer might like even if the customer is new to this application.

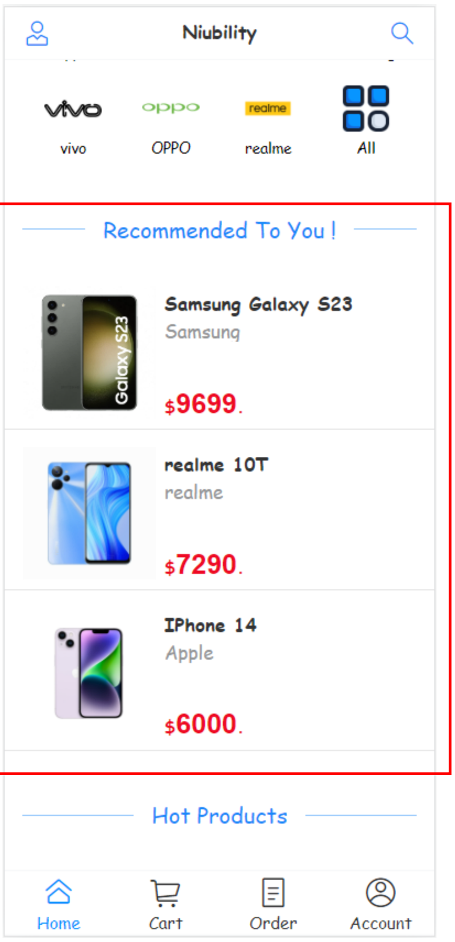


Figure 5-1-1-1: Recommended list for users

As we know all the products will be shown on the product list page. But the space is limited, which means we could not display all the products on one page. Therefore, we design pagination to tackle this problem. We could turn to the next page by simply tapping “>” or return to the previous page by tapping “<”. If we want to turn to a specific page, we just tap the identifier of that page or simply input the number in the box and tab “Go!”. If the customer inputs a number that is greater than the total number of pages, the system will just guide him to the last page.

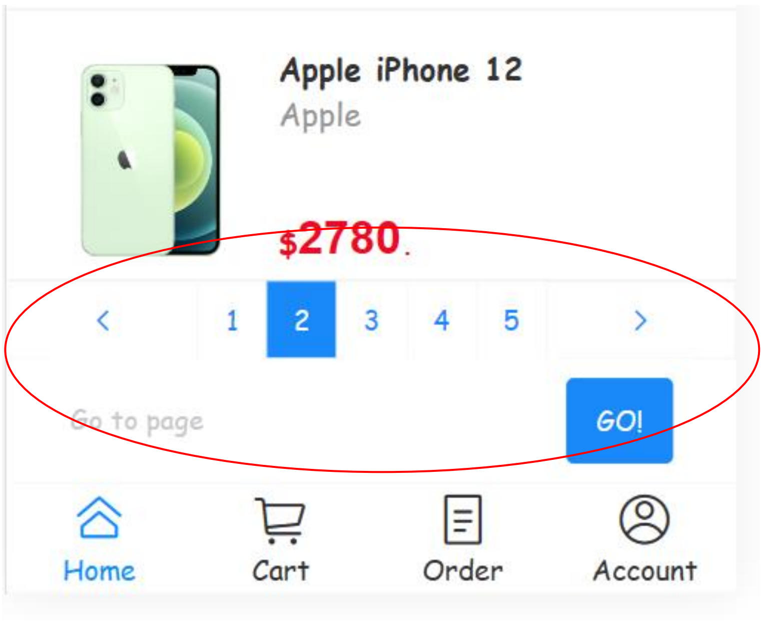


Figure 5-1-1-2: Product list page pagination

5.1.2 Customer: Search page

If the customer wants to search for a specific product, he could just tap the search icon on the top of the product list page and then input the corresponding name in the search engine. If he wants to search for iPhone 12, he could firstly filter the brand “Apple” (the font color of the filtered brand will be blue) and simply type “12”. The result will show iPhone 12 exactly. But if he accidentally inputs “16” instead, the application will consider it as a fuzzy search because the system lacks the product whose name contains “16”. It will generate a list of products that it thinks might conform to the desire of the customer.

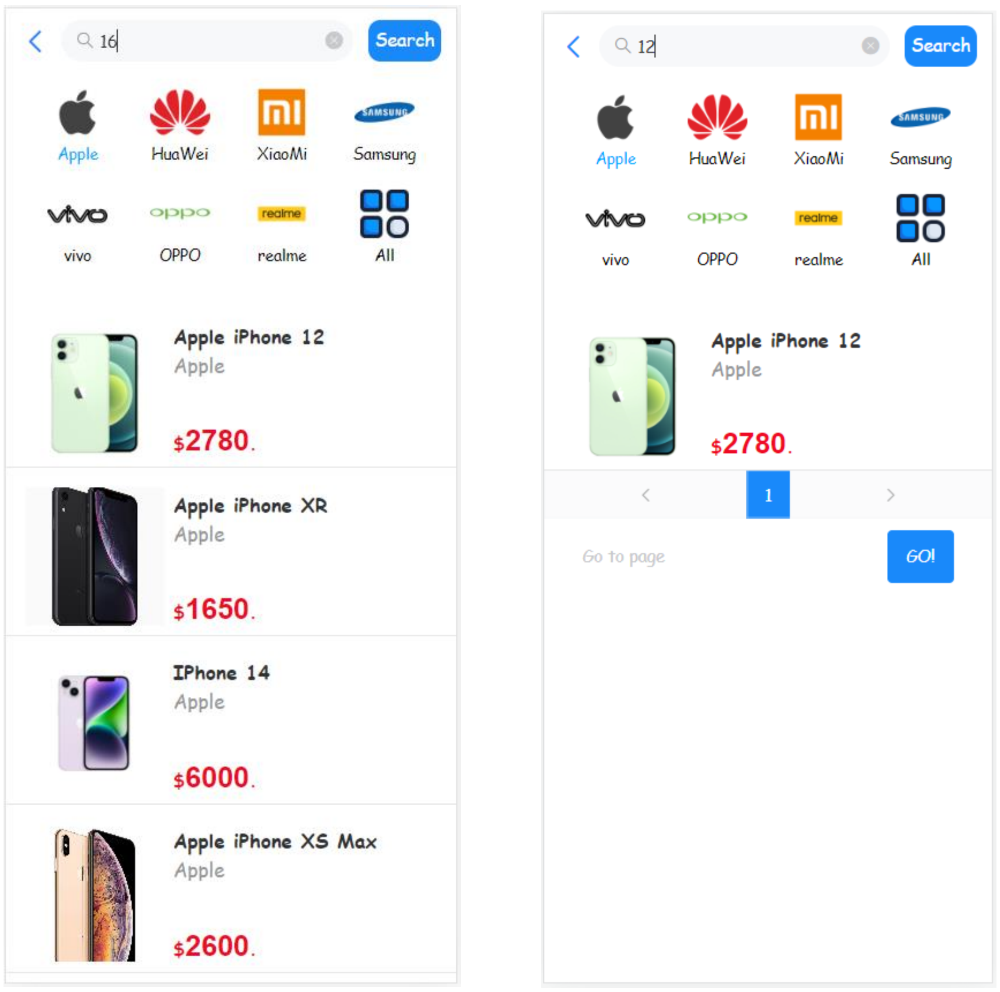
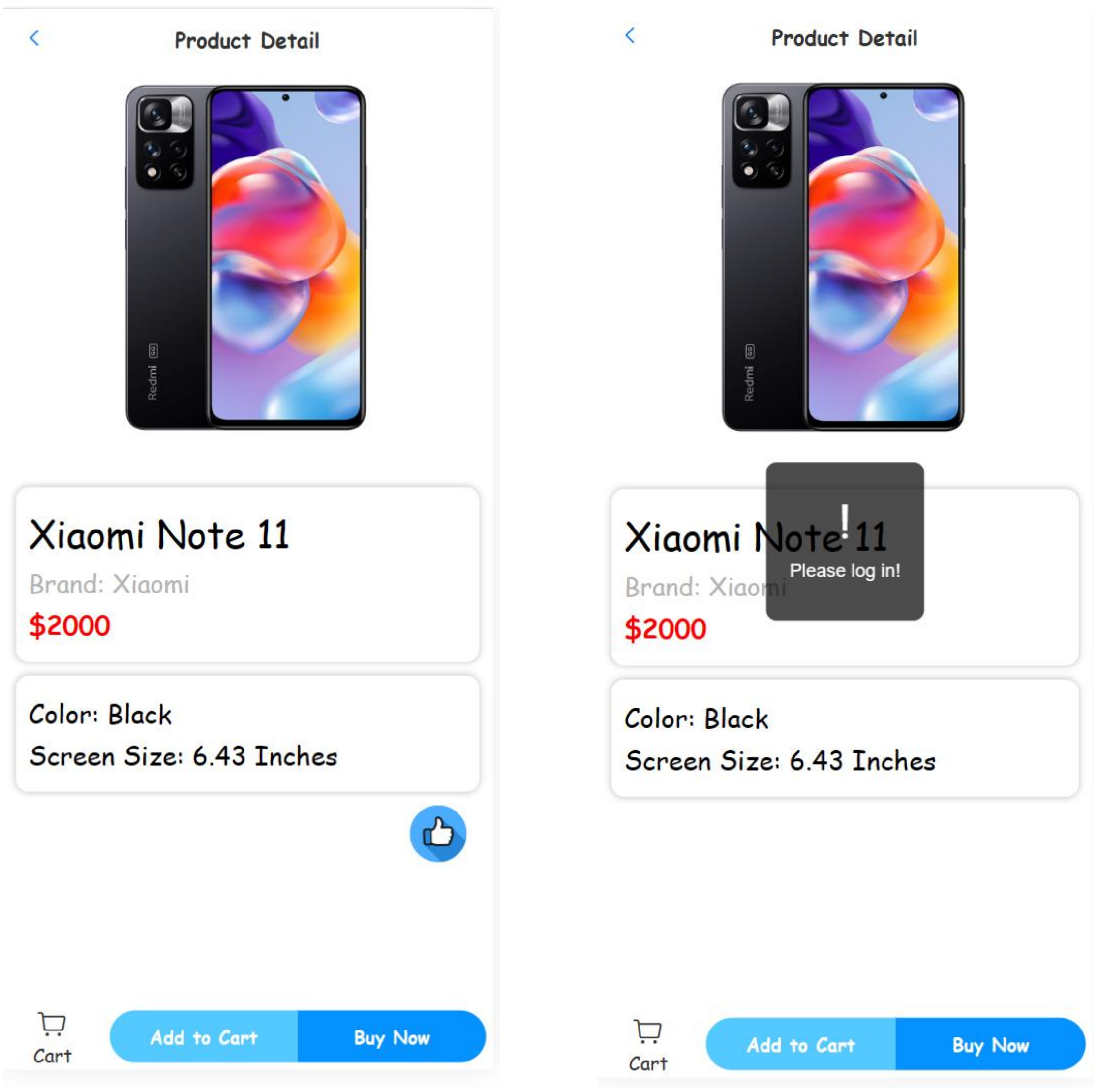


Figure 5-1-2: Search for iPhone 12 (left) and fuzzy search (right)

5.1.3 Customer: Product detail page

If the customer does not log in to the App, he will receive a reminding message which asks him to log in when he enters the product detail page. And if he just wants to directly buy this product or just simply add it to the shopping cart, the App will automatically lead him to the login page. Note that the biggest difference between the product detail page when the customer has logged in and the product detail page when the customer does not log in is the existence of an icon to determine whether the customer like this product. If the customer taps the "Add to Cart" button, the item will be added to his shopping cart, but the page will not jump to the shopping cart page. If the customer only wants to buy one item, he can click on the "Buy Now" button. At this time, when the product is added to the shopping cart, the page will directly jump to the shopping cart page.


Product Detail

Xiaomi Note 11

Brand: Xiaomi

$2000

Color: Black

Screen Size: 6.43 Inches

Product Detail

Xiaomi Note 11

Brand: Xiaomi

$2000

!

Please log in!

Color: Black

Screen Size: 6.43 Inches

Figure 5-1-3: Product detail page where the user is logged in (left) or not logged in (right)

5.1.4 Customer: Shopping cart page

After the customer adds products to the shopping cart, this page will show the quantity and unit price of each product and the total amount of all the products in that shopping cart. For all products added to the shopping cart, their quantity defaults to 1. The customer can change the quantity of each product or remove them from the shopping cart. By the way, if there are products in the shopping cart, the upper right of the shopping cart icon in the footer will display how many products there are instead of the sum of the total number of each product. And the button "Check Out" allows the customer to turn into the final stage of shopping which will construct order in the end.

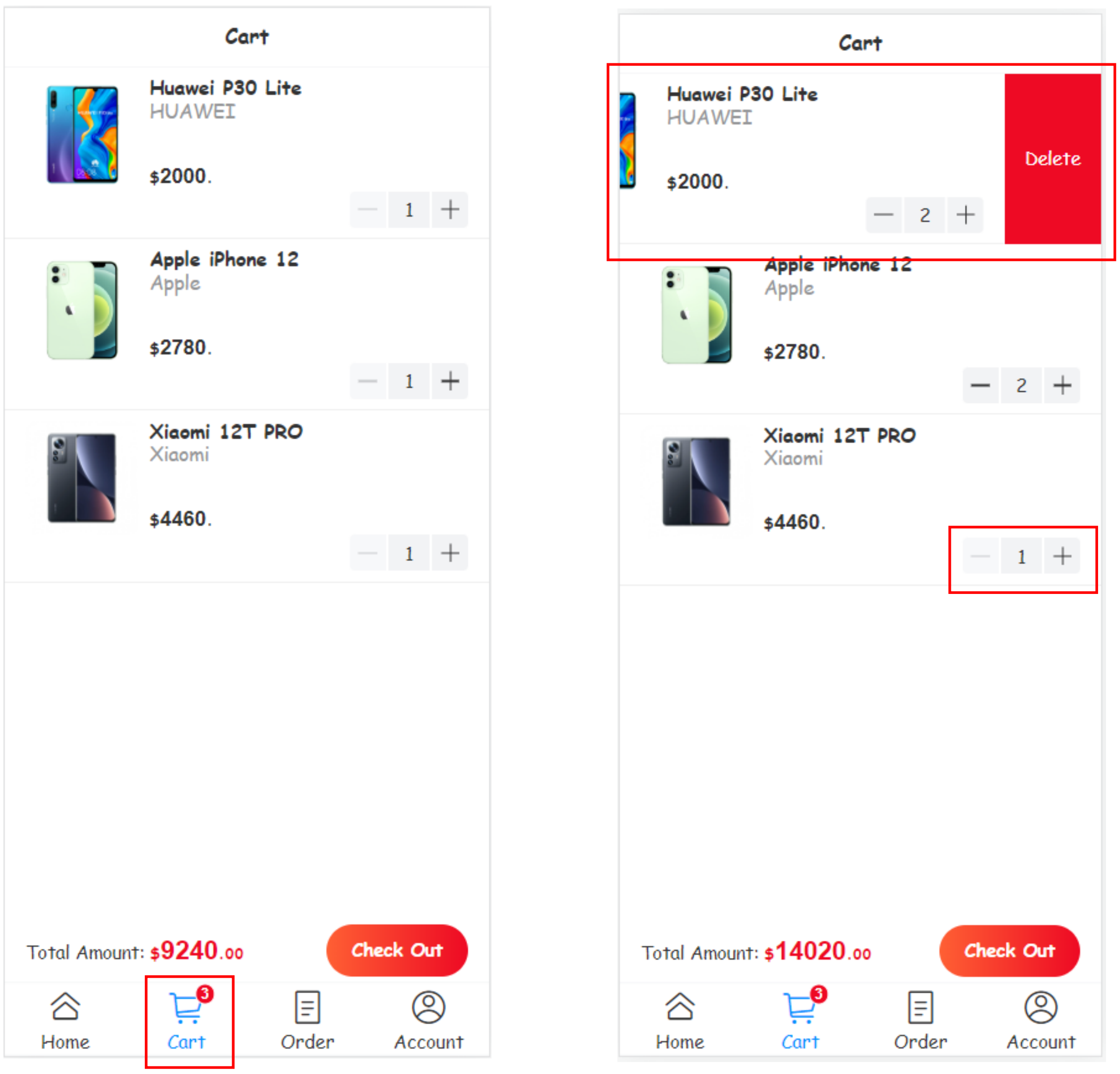
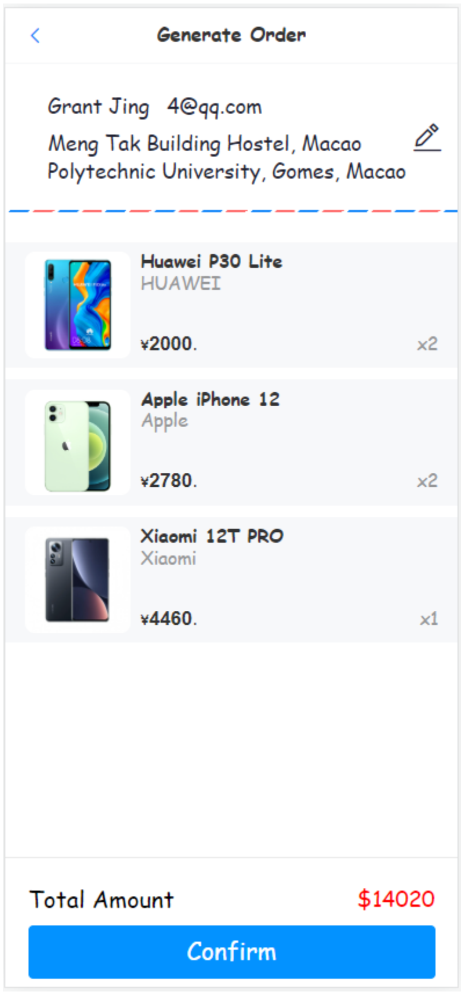


Figure 5-1-4: Shopping cart page

5.1.5 Customer: Order generation page

After tapping the “Check Out” button, the customer will be navigated to the order generation page. On this page, customers can click the “edit” icon to change their shipping address. If nothing needs to be changed, they can just click the “Confirm” button to confirm the order. After confirming the order, it will automatically jump to the order detail page.



Grant Jing 4@qq.com
Meng Tak Building Hostel, Macao Polytechnic University, Gomes, Macao

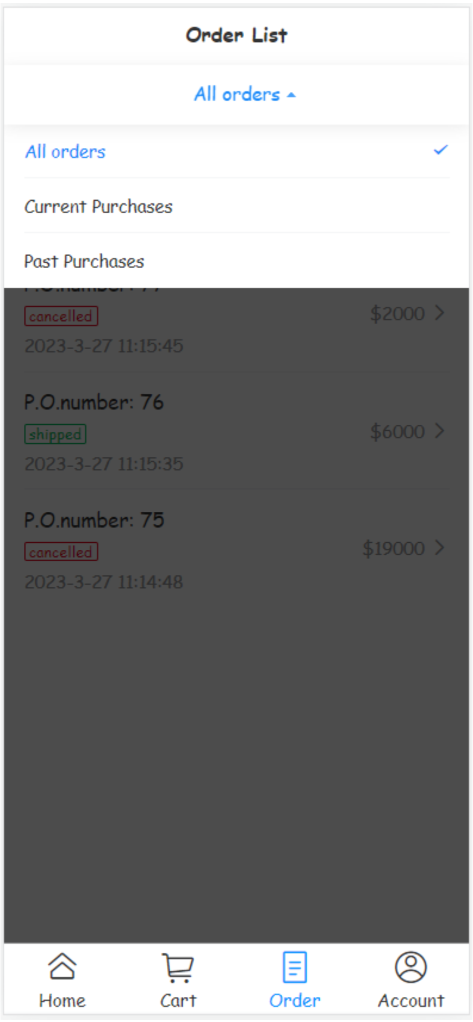
Huawei P30 Lite HUAWEI	¥2000.	x2
Apple iPhone 12 Apple	¥2780.	x2
Xiaomi 12T PRO Xiaomi	¥4460.	x1

Total Amount $14020

Figure 5-1-5: Confirm page

5.1.6 Customer: Order list page with filtering

On this page, it will present the list in chronological sequence. Customers could filter the orders between “All Orders”, “Current Purchases” and “Past Purchases”.


Order List

All orders ▴

All orders ✓

Current Purchases

Past Purchases

cancelled $2000 >
2023-3-27 11:15:45

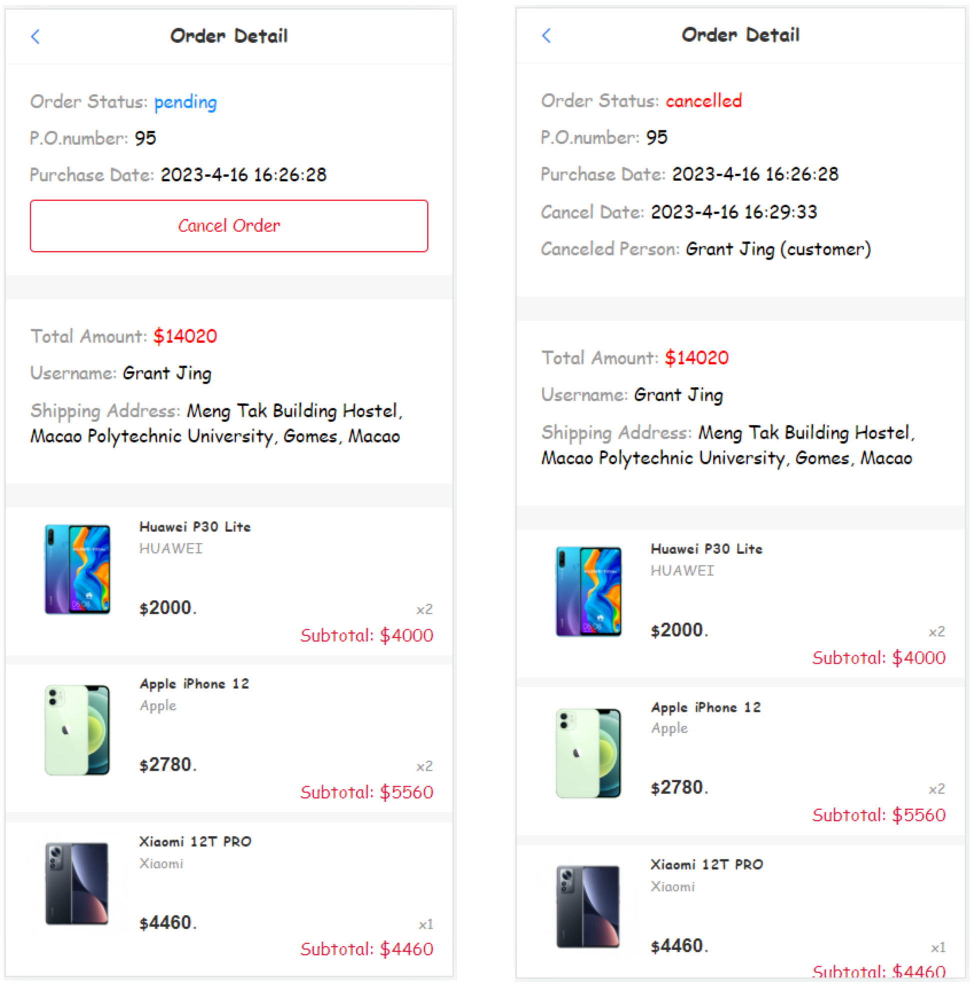
P.O.number: 76
shipped $6000 >
2023-3-27 11:15:35

P.O.number: 75
cancelled $19000 >
2023-3-27 11:14:48

Figure 5-1-6: Order list page and filtering

5.1.7 Customer: Order detail page

On this page, three aspects are displayed. They are order information, customer information, and product information respectively. The order status and purchase date will be displayed in the order information. In addition, after canceling the order, the order status will be changed and it will display the cancel date and the person who canceled it. Moreover, in the product information, the subtotal of each product will also be displayed.


Order Detail

Order Status: pending

P.O.number: 95

Purchase Date: 2023-4-16 16:26:28

Cancel Order

Total Amount: $14020

Username: Grant Jing

Shipping Address: Meng Tak Building Hostel, Macao Polytechnic University, Gomes, Macao

Huawei P30 Lite
HUAWEI
$2000. x2
Subtotal: $4000

Apple iPhone 12
Apple
$2780. x2
Subtotal: $5560

Xiaomi 12T PRO
Xiaomi
$4460. x1
Subtotal: $4460

Order Detail

Order Status: cancelled

P.O.number: 95

Purchase Date: 2023-4-16 16:26:28

Cancel Date: 2023-4-16 16:29:33

Canceled Person: Grant Jing (customer)

Total Amount: $14020

Username: Grant Jing

Shipping Address: Meng Tak Building Hostel, Macao Polytechnic University, Gomes, Macao

Huawei P30 Lite
HUAWEI
$2000. x2
Subtotal: $4000

Apple iPhone 12
Apple
$2780. x2
Subtotal: $5560

Xiaomi 12T PRO
Xiaomi
$4460. x1
Subtotal: $4460

Figure 5-1-7: Order detail page

5.1.8 Customer: Account page

The username and e-mail of the current user will be displayed on the user account page. Users can perform corresponding operations by tapping the three options below the user card. For example, users can tap "Account Management" to modify their usernames and passwords. And they can also tap the eye icon to the right of the password input box to change the password's visibility.

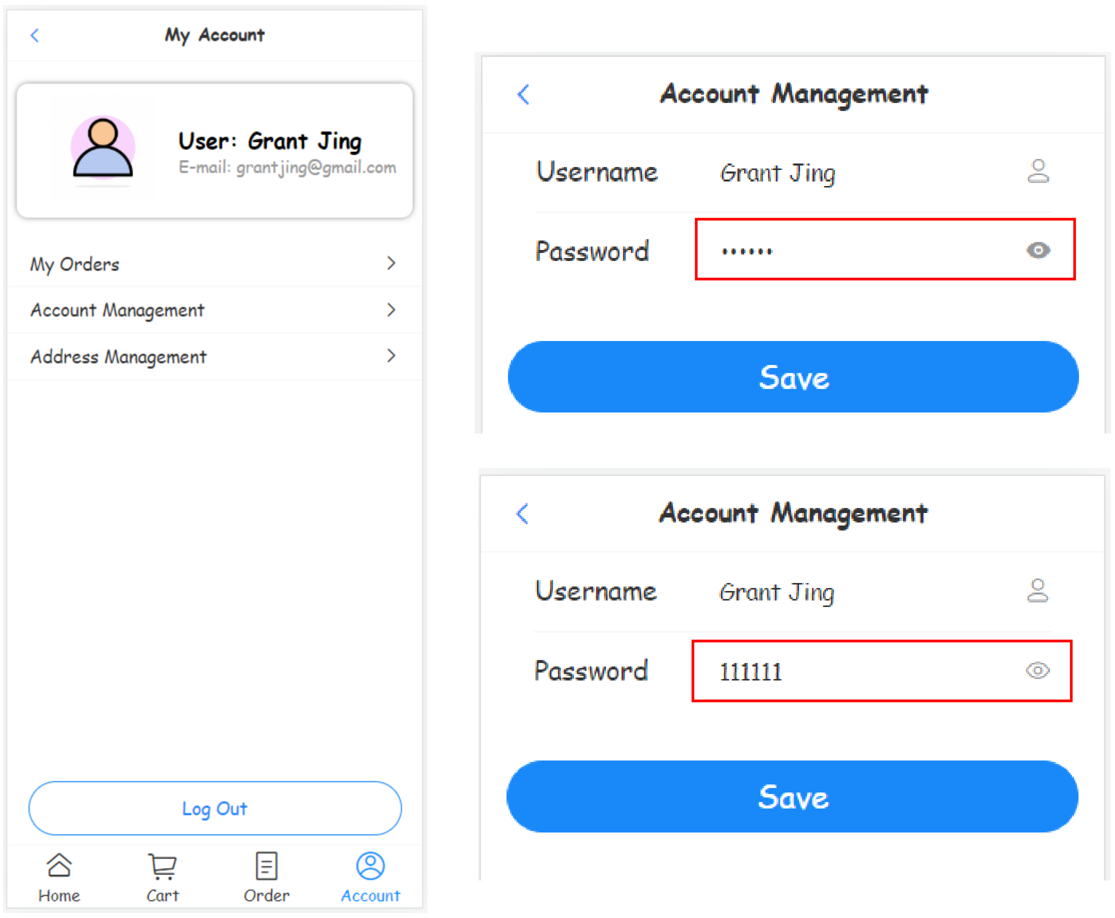


Figure 5-1-8: Account page and Account management page

5.1.9 Customer: Login page and signup page

On the Login page and signup page, if the user enters the wrong email format, an error message with the words "Please enter valid email" will appear at the bottom of the input box. If the user enters a password with less than 6 characters, an error message with the words “Password must be at least 6 characters” will appear. If the user does not enter anything, an error message "Please enter ***" will appear below the input box.

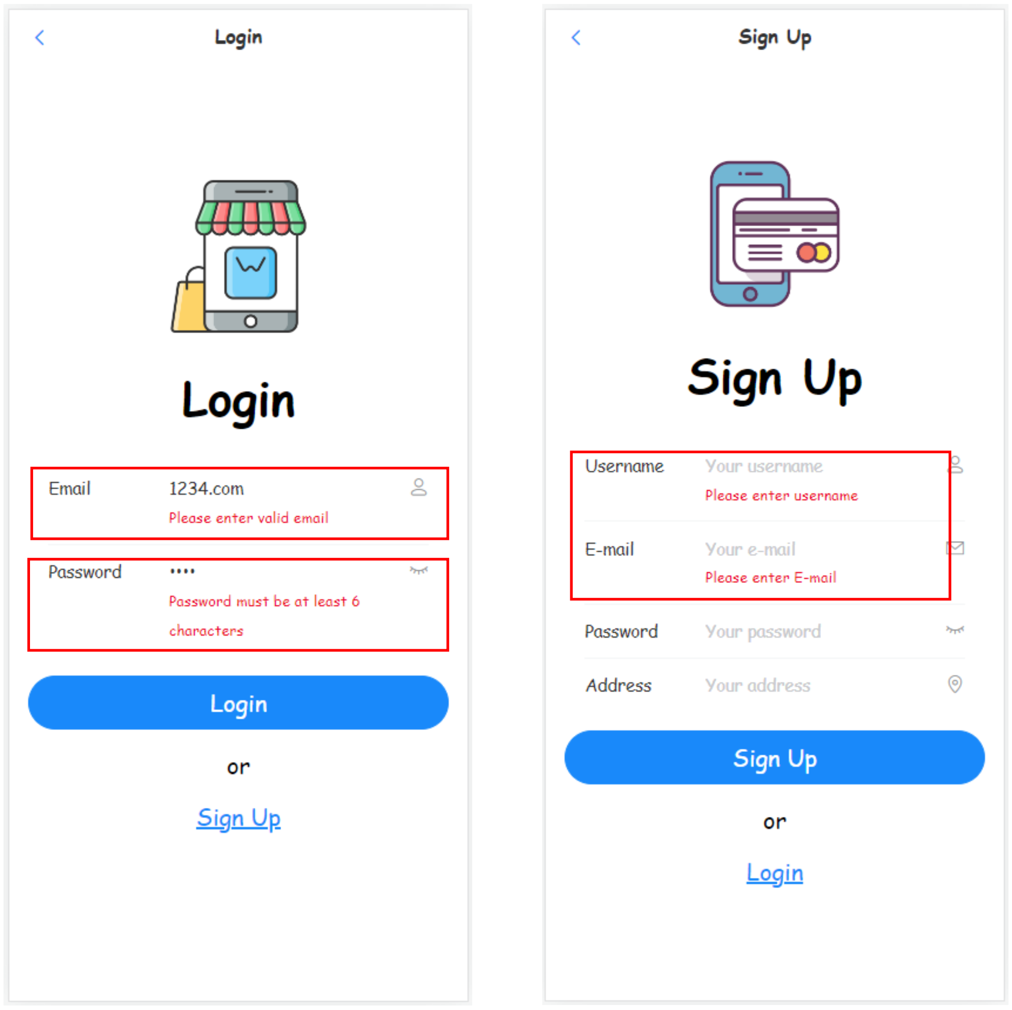


Figure 5-1-9: Login page (left) and Sign up page (right)

5.1.10 Vendor: Product list page with searching and adding a new product

Compared to the product list page for the customer, the biggest differences are a new way to search for a product and the port to add a new product into the system. For searching, the vendor could directly input the productId to search for the corresponding product (Just simply input "id:xx" in the search engine). And if the vendor wants to add a new product to the system, he could directly tap the button "Add a new product" and input all the related attributes of that product. After submitting, the system will automatically generate a productId for the product.

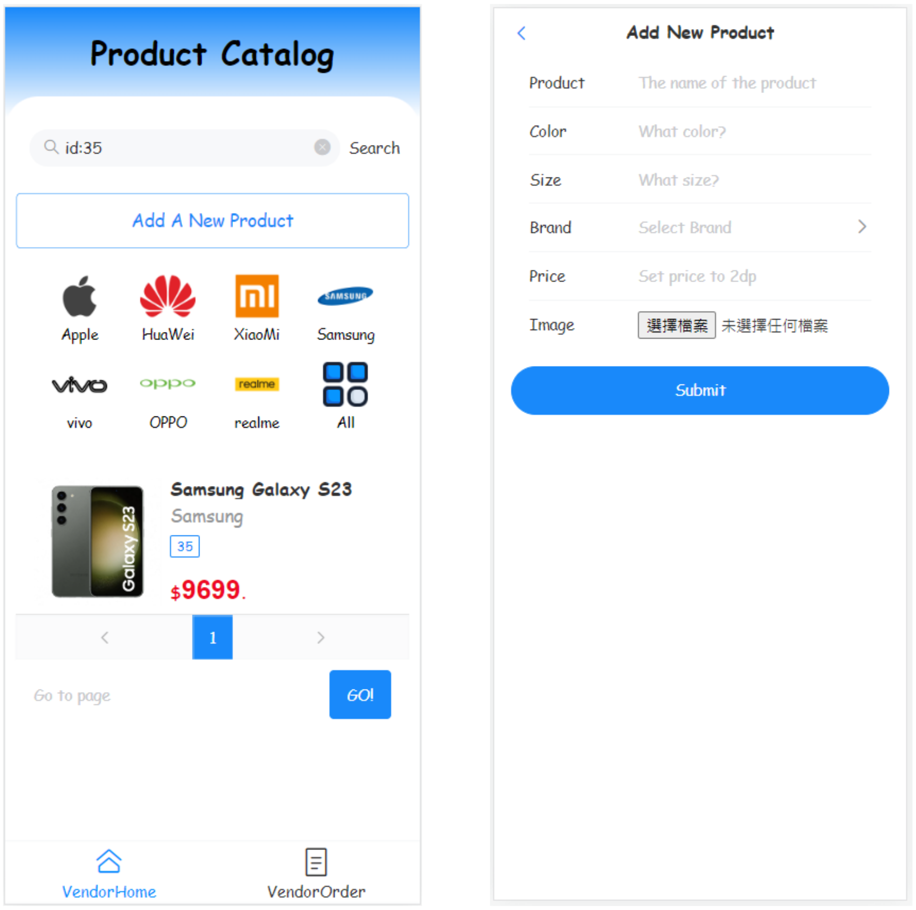
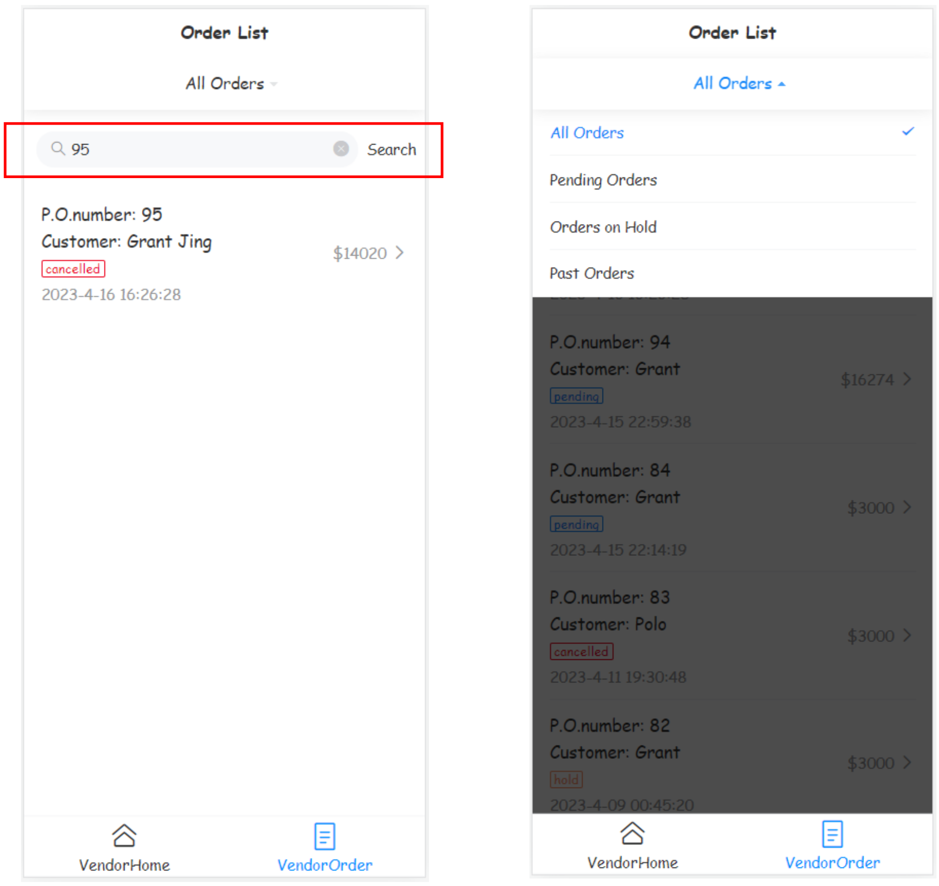


Figure 5-1-10: Vendor product list page (left) and add new product page (right)

5.1.11 Vendor: Order list page

On the order list page, the vendor could directly see which order belongs to the specific customer or input the specific purchase order number to get the corresponding order. Meanwhile, he can filter the system’s orders based on “All Orders”, “Pending Orders”, “Orders on Hold” and “Past Orders”.


Order List

All Orders

95 Search

P.O.number: 95
Customer: Grant Jing
cancelled
2023-4-16 16:26:28
$14020

VendorHome VendorOrder

Order List

All Orders

All Orders ✓
Pending Orders
Orders on Hold
Past Orders

P.O.number: 94
Customer: Grant
pending
2023-4-15 22:59:38
$16274

P.O.number: 84
Customer: Grant
pending
2023-4-15 22:14:19
$3000

P.O.number: 83
Customer: Polo
cancelled
2023-4-11 19:30:48
$3000

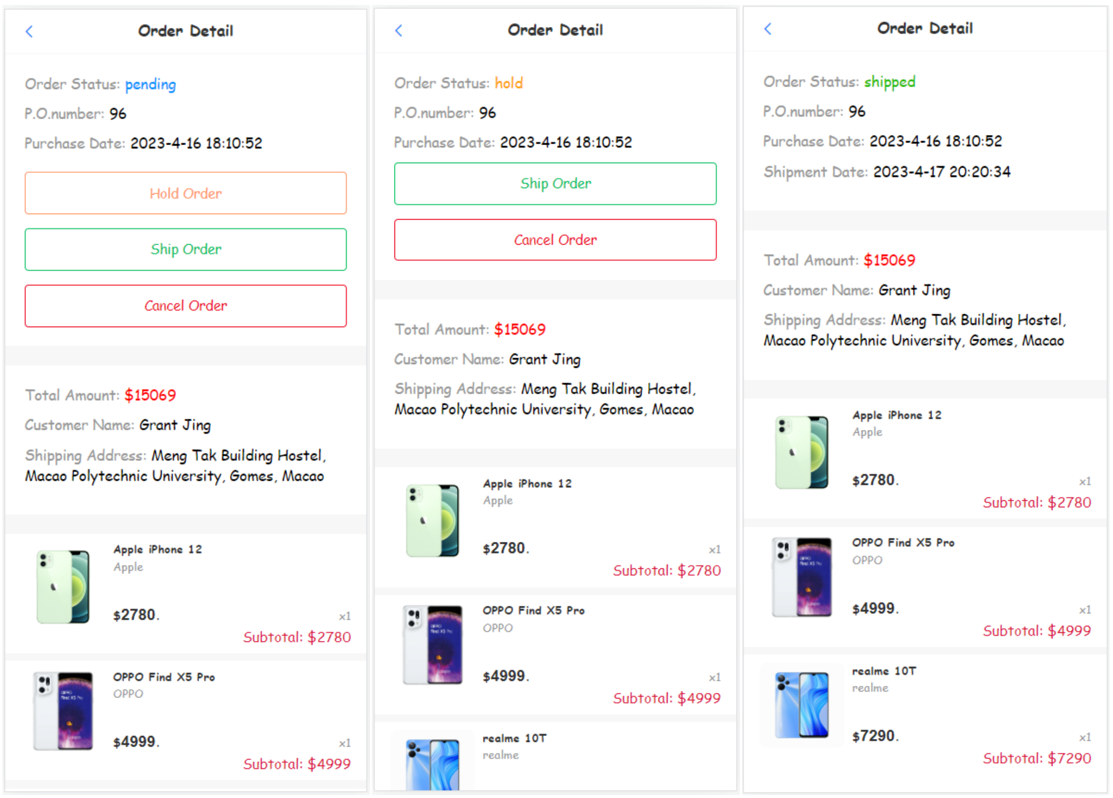
P.O.number: 82
Customer: Grant
hold
2023-4-09 00:45:20
$3000

VendorHome VendorOrder

Figure 5-1-11: Vendor order list page

5.1.12 Vendor: Order detail page

Compared to the order detail page for customers, the vendor has the right to hold, ship, and cancel the order. After choosing the specific status, the order detail will record the change time correspondingly.


Order Detail

Order Status: pending
P.O.number: 96
Purchase Date: 2023-4-16 18:10:52

Hold Order
Ship Order
Cancel Order

Total Amount: $15069
Customer Name: Grant Jing
Shipping Address: Meng Tak Building Hostel, Macao Polytechnic University, Gomes, Macao

Apple iPhone 12
Apple
$2780. x1
Subtotal: $2780

OPPO Find X5 Pro
OPPO
$4999. x1
Subtotal: $4999

Order Detail

Order Status: hold
P.O.number: 96
Purchase Date: 2023-4-16 18:10:52

Ship Order
Cancel Order

Total Amount: $15069
Customer Name: Grant Jing
Shipping Address: Meng Tak Building Hostel, Macao Polytechnic University, Gomes, Macao

Apple iPhone 12
Apple
$2780. x1
Subtotal: $2780

OPPO Find X5 Pro
OPPO
$4999. x1
Subtotal: $4999

realme 10T
realme

Order Detail

Order Status: shipped
P.O.number: 96
Purchase Date: 2023-4-16 18:10:52
Shipment Date: 2023-4-17 20:20:34

Total Amount: $15069
Customer Name: Grant Jing
Shipping Address: Meng Tak Building Hostel, Macao Polytechnic University, Gomes, Macao

Apple iPhone 12
Apple
$2780. x1
Subtotal: $2780

OPPO Find X5 Pro
OPPO
$4999. x1
Subtotal: $4999

realme 10T
realme
$7290. x1
Subtotal: $7290

Figure 5-1-12: Vendor order detail page

5.2 Testing and System Evaluations

5.2.1 Test on Recommendation Mechanism

To evaluate the effectiveness of the recommendation algorithm, we have designed and conducted a simple test case based on preferences on brands.

Firstly, we create 2 new accounts as User A and User B. We assume that User A likes Huawei. Oppositely, User B dislikes Huawei and like Apple. They both like Xiaomi.

Figure 5-2-1-1 shows the top 3 products recommended to Tester 1, 2. Tester 1 disliked some Huawei, and in the recommendation generated next time has become all iPhones. Meanwhile, Tester 2 disliked some iPhones, and the brand of 3 recommendation products are consistent with the desired brand: Huawei. According to the brands of the products, we summarize the outcomes in Table 1.

In conclusion, after conducting the test case, we noticed that our recommendation algorithm is sufficiently effective to find out the similar users of a new user and recommend the preferences of them to the new user.

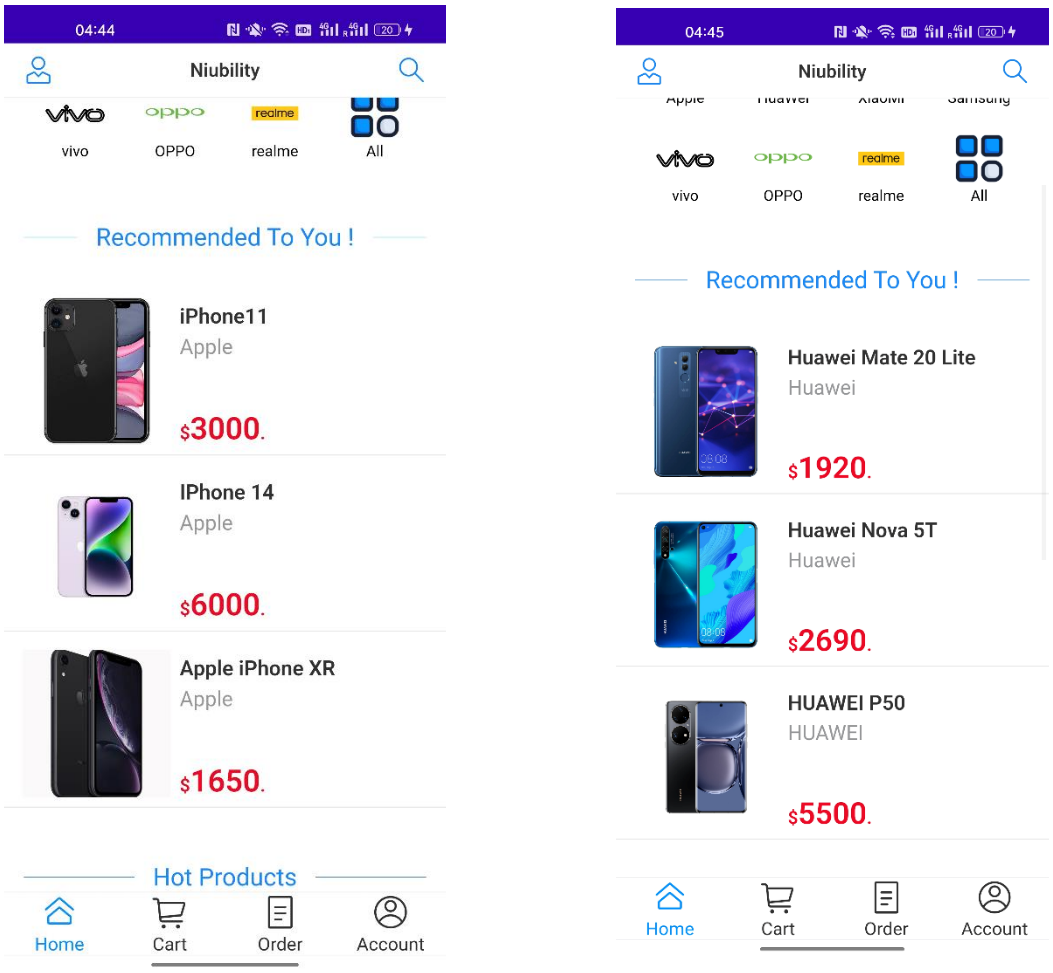


Figure 5-2-1-1: Recommendation of Tester1 and Tester2

Brand / User	Huawei	Apple	Xiaomi	Desired Outcome	Outcome	Similar User
User A	√	×	√	\	\	\
User B	×	√	√	\	\	\
Tester 1	×	N/A	N/A	Apple	Apple	User B
Tester 2	N/A	×	N/A	Huawei	Huawei	User A

Table 1: Outcomes

5.2.2 Debug On a Real Android Phone

Android Debug Bridge (adb) is a versatile command-line tool that enables communications with devices.

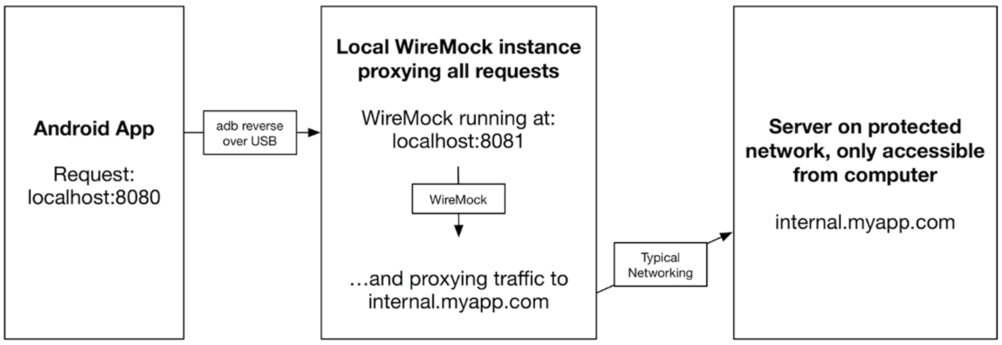
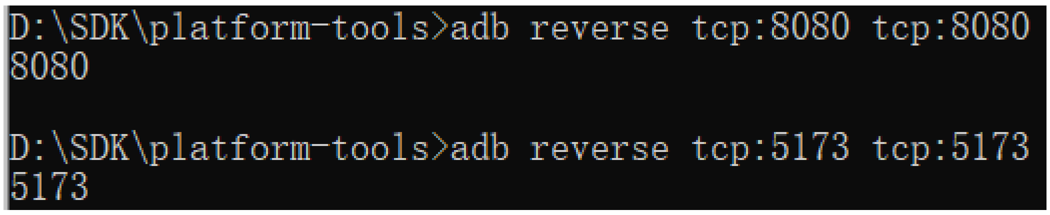

Android App
Request: localhost:8080
adb reverse over USB
Local WireMock instance proxying all requests
WireMock running at: localhost:8081
WireMock
...and proxying traffic to internal.myapp.com
Typical Networking
Server on protected network, only accessible from computer
internal.myapp.com

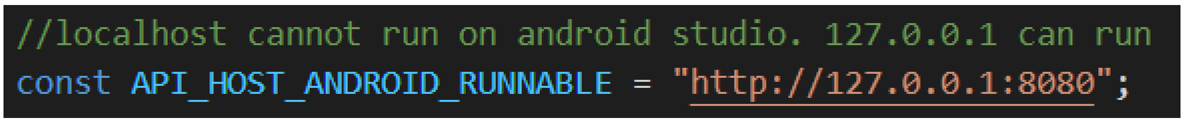
Figure 5-2-2-1: AndroidAndroid Debug Bridge (ADB)



```
D:\SDK\platform-tools>adb reverse tcp:8080 tcp:8080
8080

D:\SDK\platform-tools>adb reverse tcp:5173 tcp:5173
5173
```

Figure 5-2-2-2: ADB Commands

By using the adb command, it allows the mobile phone to access host 127.0.0.1 (loopback address), in the same way that we debug in the PC browser. The API server running on port 8080 should also be bridged to allow data communication between the mobile app and the API server. Moreover, we are able to test our mobile App by simply accessing 127.0.0.1/5173 without any deployment



```
//localhost cannot run on android studio. 127.0.0.1 can run
const API_HOST_ANDROID_RUNNABLE = "http://127.0.0.1:8080";
```

Figure 5-2-2-3: The vue.js accesses API server running on port 8080.

6 Conclusion and Further Work

In conclusion, we have developed a user-friendly mobile online shopping mall application based on Java (Spring Boot as back-end) and JavaScript (Vue.js as front-end).

We have developed a mobile-specific graphical user interface by exploiting UI component libraries for the mobile Web App so that customers can browse the products and make purchases in this App easily through their mobile phones. We have implemented the vendors' mode for the same App, which is a different view. Thus, the vendors can also post and edit products on stock. Purchase orders are maintained in the MySQL database. Customers and vendors can view and manipulate purchase orders conveniently. In addition, authentication modules are added for more secure account management.

Based on the above design and implementation, our mobile application offers a convenient, efficient, and user-friendly solution for recreational and practical consumption.

Traditional Web Apps and Native Apps can be merged to some extent. They are called Progressive Web Apps (PWAs). Since our project uses Android Web View, most functionalities are based on the Web. It is not available offline, and it requires users' installation in order to be used. However, we can extend our App to a PWA: make it an installable Web App so that the users can use our Web App usually without a network connection [26]. Moreover, the user can access our App with installation from the App Store or a downloaded browser [27]. Thus, the performance and functionality will increase while keeping the accessibility and reliability unaffected.

References

[1] "Amazon (company)," [Online]. Available: https://en.wikipedia.org/wiki/Amazon_(company).

[2] "Taobao (company)," [Online]. Available: https://en.wikipedia.org/wiki/Taobao.

[3] "The definition of e-commerce," [Online]. Available: https://www.techtarget.com/searchcio/definition/e-commerce.

[4] [Online]. Available: https://www.statista.com/statistics/321482/smartphone-user-penetration-in-china/ .

[5] "Advantages of a Mobile Shopping Experience," [Online]. Available: https://imaginovation.net/blog/advantages-of-a-mobile-shopping-experience/.

[6] "Taobao," [Online]. Available: https://www.taobao.com/.

[7] "JD.com," [Online]. Available: https://www.jd.com/.

[8] "Amazon," [Online]. Available: https://www.amazon.com/.

[9] Vue.js, "Introduction of Vue," [Online]. Available: https://vuejs.org/guide/introduction.html.

[10] "Spring Boot definition," [Online]. Available: https://www.ibm.com/topics/java-spring-boot.

[11] "MVC Structure," [Online]. Available: https://www.geeksforgeeks.org/benefit-of-using-mvc/.

[12] Vue.js, "What is Vue Router," [Online]. Available: https://router.vuejs.org/introduction.html.

[13] Vue.js, "What is Vuex," [Online]. Available: https://vuex.vuejs.org/.

[14] "What is Axios," [Online]. Available: https://axios-http.com/docs/intro.

[15] C. BasuMallick, "What Is a Single-Page Application? Architecture, Benefits, and Challenges," 18 October 2022. [Online]. Available: https://www.spiceworks.com/tech/devops/articles/what-is-single-page-application/.

[16] "SFC Composition API Syntax Sugar," Vue, [Online]. Available: https://vuejs.org/api/sfc-script-setup.html.

[17] "What, how and why SFC," Vue, [Online]. Available: https://vuejs.org/guide/scaling-up/sfc.html.

[18] "CSRF protection," [Online]. Available: https://inertiajs.com/csrf-protection.

[19] "Vant 4," [Online]. Available: https://vant-ui.github.io/vant/#/en-US.

[20] "Spring Boot JPA," [Online]. Available: https://www.linkedin.com/pulse/introduction-spring-data-jpa-omar-ismail.

[21] "MySQL," [Online]. Available: https://en.wikipedia.org/wiki/MySQL .

[22] "Postcss-pxtorem," [Online]. Available: https://github.com/cuth/postcss-pxtorem.

[23] "Amfe-flexible," [Online]. Available: https://github.com/amfe/lib-flexible.

[24] "Gulf of Evaluation and Gulf of Execution," [Online]. Available: https://www.interaction-design.org/literature/book/the-glossary-of-human-computer-interaction/gulf-of-evaluation-and-gulf-of-execution.

[25] "Django VS Spring Boot," [Online]. Available: https://optymize.io/blog/django-vs-springboot-which-is-better-for-your-website/.

[26] "What are Progressive Web Apps?," [Online]. Available: https://web.dev/i18n/en/what-are-pwas/.

[27] "Overview of Progressive Web Apps (PWAs)," [Online]. Available: https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/.

[28] [Online]. Available: https://www.techtarget.com/searchcio/definition/e-commerce.

Appendix A. Project Management

Project management is the application of processes, methods, skills, knowledge and experience to achieve specific project objectives according to the project acceptance criteria within agreed parameters. Project management has final deliverables that are constrained to a finite timescale and budget. There are four parts in our project management, they are Project Scope Management, Project Schedule Management, Project Human Resource Management, and Changes and Adjustment.

Project Scope Management

A work breakdown structure (WBS) is a visual, hierarchical and deliverable-oriented deconstruction of a project. For our project, we divided it into 6 parts: Initiating, Analysis, Design, Implementation, Testing, and Deployment. Figure A-1 is the WBS Diagram of our project.

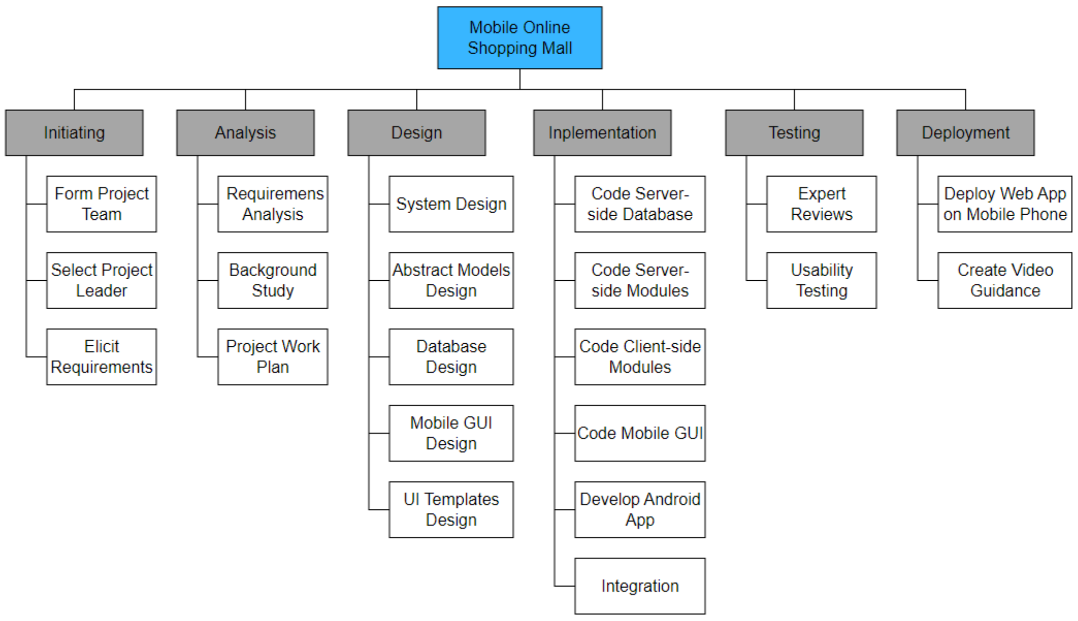
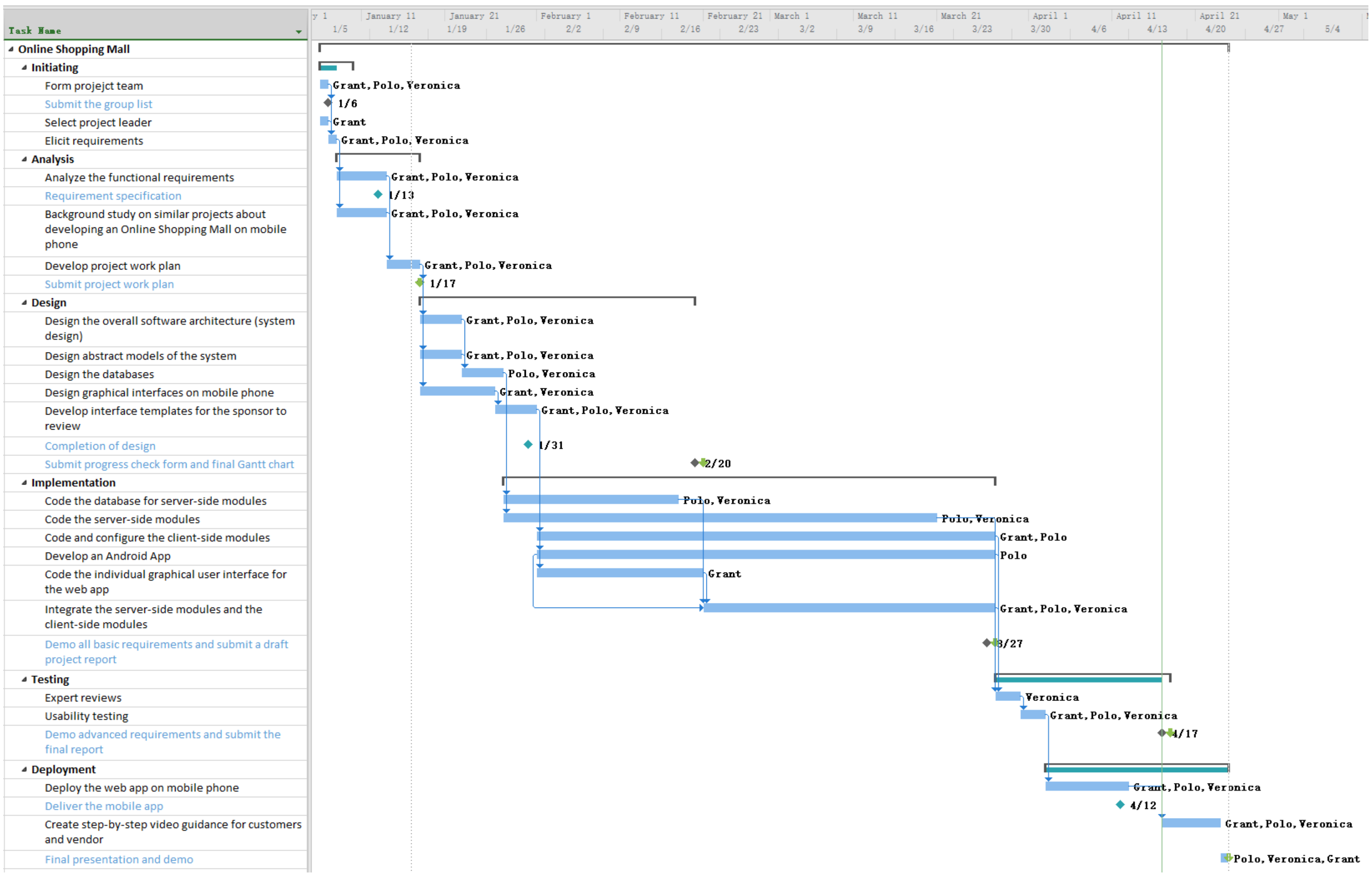


Figure A-1: WBS diagram

Project Schedule Management

Project Schedule Management is the process of defining project tasks and their durations, dependencies, and assigned resources in order to complete the project within a designated time frame. In our project, we use the Gantt chart to display our project schedule. Figure shows all the tasks of our project, the duration of each task, the human resource of each task, and the milestones.



Task Name	
Online Shopping Mall	
Initiating	
Form projejct team	Grant, Polo, Veronica
Submit the group list	1/6
Select project leader	Grant
Elicit requirements	Grant, Polo, Veronica
Analysis	
Analyze the functional requirements	Grant, Polo, Veronica
Requirement specification	1/13
Background study on similar projects about developing an Online Shopping Mall on mobile phone	Grant, Polo, Veronica
Develop project work plan	Grant, Polo, Veronica
Submit project work plan	1/17
Design	
Design the overall software architecture (system design)	Grant, Polo, Veronica
Design abstract models of the system	Grant, Polo, Veronica
Design the databases	Polo, Veronica
Design graphical interfaces on mobile phone	Grant, Veronica
Develop interface templates for the sponsor to review	Grant, Polo, Veronica
Completion of design	1/31
Submit progress check form and final Gantt chart	2/20
Implementation	
Code the database for server-side modules	Polo, Veronica
Code the server-side modules	Polo, Veronica
Code and configure the client-side modules	Grant, Polo
Develop an Android App	Polo
Code the individual graphical user interface for the web app	Grant
Integrate the server-side modules and the client-side modules	Grant, Polo, Veronica
Demo all basic requirements and submit a draft project report	3/27
Testing	
Expert reviews	Veronica
Usability testing	Grant, Polo, Veronica
Demo advanced requirements and submit the final report	4/17
Deployment	
Deploy the web app on mobile phone	Grant, Polo, Veronica
Deliver the mobile app	4/12
Create step-by-step video guidance for customers and vendor	Grant, Polo, Veronica
Final presentation and demo	Polo, Veronica, Grant

Figure A-2: Gantt chart

Project Human Resource Management

Project Human Resource Management involves developing and managing the project team. And in the following, Responsibility Assignment Matrix and Resource Histogram will be presented.

Responsibility Assignment Matrix (RAM)

A Responsibility Assignment Matrix (RAM) is a matrix used to define project responsibilities among the project team.

	Grant	Polo	Veronica
Background Study	A	R	C
Database Design	C	A	R
UI Design	R	C	A
Frontend Development	R	C	A
Backend Development	C	A	R
Integration	A	R	C
Android App Development	R	R	A
Testing	A	C	R

Figure A-3: Responsibility Assignment Matrix (RAM)

Resource Histogram

A Resource Histogram is a column chart that shows the number of resources assigned to a project over time. (Note that the unit of y-axis is "hour").

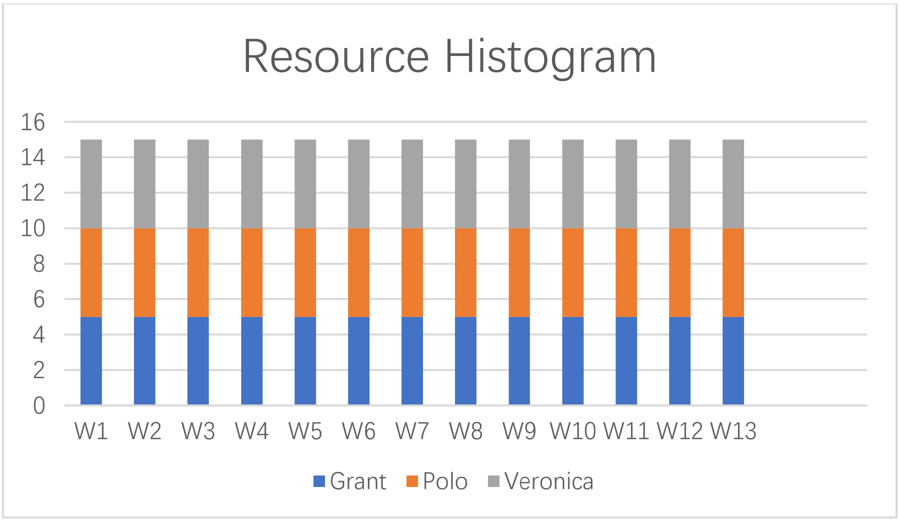


Figure A-4: Resource Histogram

Changes and Adjustment

By week 4, we were struggling with dealing with the totally new technological stack (Android Studio) to develop the Android mobile App directly. This stack was not that popular which means we could not get much community help and reference. So we were totally lost and made slight progress during these weeks, which indicated we were left behind. After searching the Internet, we finally decided to develop a web-mobile App and choose the appropriate backend framework at the beginning. After this decision, we made an effort to achieve the goal and catch up with the progress schedule.