

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

---🙢🕮🙠---

A yellow and red sign with a red circle and a star

Description automatically generated

**BÁO CÁO HỌC PHẦN**

**PHÁT TRIỂN ỨNG DỤNG TRÊN THIẾT BỊ DI ĐỘNG**

**ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG ĐẶT ĐỒ ĂN**

**FUNMOUTH**

**GVHD: ThS. Vũ Thị Dương**

**Nhóm: 1**

**Sinh viên:**

1. **Bùi Đức Lương**
2. **Cao Hoàng Long**
3. **Lê Quí Long**
4. **Mai Công Sao**
5. **Nguyễn Tu Tâm**

**Lớp: 20231IT6029003 Khóa: 15**

Hà Nội – Năm học: 2023

# **TABLE OF CONTENTS**

[**TABLE OF CONTENTS** 2](#_Toc154439011)

[**PART 1: INTRODUCTION** 2](#_Toc154439012)

[**1.1.** **Reasons for choosing the topic** 2](#_Toc154439013)

[**1.2.** **Overview of Android** 2](#_Toc154439014)

[**1.2.1.** **What is the Android Operating System?** 2](#_Toc154439015)

[**1.2.2.** **Programming languages for Android** 3](#_Toc154439016)

[**1.3.** **Architecture of Android** 3](#_Toc154439017)

[**1.4.** **Java language** 4](#_Toc154439018)

[**1.5.** **SQLite** 4](#_Toc154439019)

[**PART 2: RESEARCH RESULTS** 6](#_Toc154439020)

[**2.1.** **Understanding the requirements** 6](#_Toc154439021)

[**2.2.** **Construction** 6](#_Toc154439022)

[**2.2.1.** **User requirements analysis** 6](#_Toc154439023)

[**2.2.2.** **Description of data class diagram** 18](#_Toc154439024)

[**2.3.** **System design** 18](#_Toc154439025)

[**2.3.1.** **Interface description** 19](#_Toc154439026)

[**2.4.** **Implementation of the problem** 27](#_Toc154439027)

[**2.4.1.** **Bui Duc Luong - Cart screen** 27](#_Toc154439028)

[**2.4.2.** **Mai Cong Sao – Favourite screen** 31](#_Toc154439029)

[**2.4.3.** **Le Qui Long – Home Screen** 35](#_Toc154439030)

[**2.4.4.** **Nguyen Tu Tam – User screen** 40](#_Toc154439031)

[**2.4.5.** **Cao Hoang Long – Category’s foods screen** 45](#_Toc154439032)

[**PART 3: ACQUIRED KNOWLEDGE AND LESSONS LEARNED** 49](#_Toc154439033)

[**3.1.** **Implemented content** 49](#_Toc154439034)

[**3.2.** **Achieved results** 49](#_Toc154439035)

[**3.3.** **Limitations of the topic** 49](#_Toc154439036)

[**3.4.** **Development directions** 49](#_Toc154439037)

[**REFERENCES** 51](#_Toc154439038)

**LIST OF SYMBOLS, ABBREVIATIONS**

|  |  |  |
| --- | --- | --- |
| ID | ANNOTATION | FULL CONTENT |
| 1 | UC | Use case |
| 2 | JVM | Java Virtual Machine |
| 3 | CSDL | Database |

# **PART 1: INTRODUCTION**

## **Reasons for choosing the topic**

This topic of mobile food ordering applications (order food mobile app) can be based on various reasons:

* Meeting Market Demands: It addresses the growing trend of online food ordering in the market.
* Business Opportunities: It provides insights into business opportunities in the mobile ordering sector.
* Enhancing User Experience: It can optimize the user experience, making the ordering and payment process convenient and fast.
* Addressing Specific Issues: If there are specific issues in the food ordering domain, this research could offer innovative solutions and practical experiments.

To carry out this project, our team applied relevant knowledge in interface processing, event handling, Fragment, RecyclerView, SQLite, etc., which we learned in our training program. Additionally, we explored additional concepts such as GridLayoutManager to ensure a comprehensive and well-organized report.

## **Overview of Android**

### **What is the Android Operating System?**

Android is an open-source operating system designed for touchscreen mobile devices and tablets, operating on the Linux platform. Initially developed by Android Inc. and later acquired by Google in 2005, it was officially introduced to users in 2007. Due to its open-source nature, Android allows programmers to easily modify and distribute it freely. This aspect has contributed significantly to Android becoming the most widely used platform for smartphone development worldwide.

As of the third quarter of 2012, Android held a 65% share of the global smartphone market. According to surveys, around 500 million devices were activated, with approximately 1.3 million daily activations. By October 2020, Android boasted over 700,000 applications, and the estimated number of downloads from Google Play reached around 25 billion. Despite the competition from Apple's iOS, Android has maintained its dominant position in the global market.

While the emergence of Apple's iOS has had some impact, Android continues to lead the worldwide market share, showcasing its robust presence and widespread adoption.

### **Programming languages for Android**

Currently, the Android operating system supports various programming languages, including Java, Kotlin, Flutter, C, C++, CSS, Python, Lua, and XML, among others. This diversity is one of the advantages that make it easier for developers to work with Android, particularly for newcomers who can easily access the Android operating system environment.

Among the mentioned languages, Java is considered the official programming language for Android, and it is the language that Android developers most frequently interact with. Java is one of the primary programming languages extensively used in the Android operating system. It is designed to be compatible with a wide range of development environments, making it more flexible compared to other C/C++ programming languages. Additionally, Java exhibits high performance and features a memory management system that releases unused objects. In the present day, Java has been further developed to enable the creation of multi-tasking programs through its multithreading capabilities. Moreover, the Java programming language supports robust security measures with encryption algorithms such as public key or one-way hashing.

## **Architecture of Android**

Android Studio is the official Integrated Development Environment (IDE) used for Android application development, built on the foundation of IntelliJ IDEA. The main function of Android Studio is to provide interfaces that enable users to create applications and handle complex file tools in the background. The programming language used in Android Studio is Java, and it comes pre-installed on your device. When using Android Studio, you only need to write, edit, and store your code in your projects and the files within those projects. Additionally, Android Studio provides access to the Android Software Development Kit (SDK).

Some notable features of Android Studio include

* Flexible Build System based on Gradle:
* Fast and Feature-rich Emulator:
* It offers a unified development environment where you can develop for all Android devices.
* Real-time updates of combined components in the emulator and real devices through the Live Edit feature.
* Android Studio includes code templates and integrates with GitHub to help you build common app features and import sample code.
* It supports various testing frameworks and tools for comprehensive app testing.
* Lint tools for source code to catch performance, usability, version compatibility, and other issues.
* Android Studio supports C++ and the Native Development Kit (NDK) for developers working with native code.
* Integrated support for Google Cloud Platform features, facilitating easy integration of Google Cloud Messaging and App Engine.

## **Java language**

Java is a modern, high-level, object-oriented, secure, and powerful programming language that also serves as a platform.

In the context of programming, a platform refers to any hardware or software environment in which a program runs. With its runtime environment (Java Runtime Environment or JRE) and application programming interface (API), Java is considered a platform.

The Java programming language was initially developed by Sun Microsystems, initiated by James Gosling, and released in 1995. The latest version of Java Standard Edition is Java SE 8. As Java progressed and gained widespread popularity, various configurations were built to suit different platforms. Examples include J2EE for enterprise applications and J2ME for mobile applications.

Newer versions have been renamed to Java SE, Java EE, and Java ME. The guiding principle of Java is "Write Once, Run Anywhere," meaning you can write code once, and it can run on multiple platforms. For instance, a program written on Windows can run on Linux, Android, J2ME devices, and more, adhering to the philosophy of platform independence.

## **SQLite**

SQLite is a relational database management system (DBMS) similar to MySQL and others. Its notable characteristics, which distinguish it from other DBMSs, include its compact size, lightweight nature, simplicity, and the absence of a server-client model. One of its key features is that it doesn't require installation, configuration, or a separate server-client setup. As a result, there is no concept of users, passwords, or permissions within an SQLite Database, and data is stored in a single file.

SQLite is typically not used in large-scale systems, but for small to medium-sized systems, it can match or even outperform other DBMS in terms of functionality and speed. Because it doesn't need installation or configuration, SQLite is often used in development and testing environments, helping to avoid complications associated with the installation process.

# 

# **PART 2: RESEARCH RESULTS**

## **Understanding the requirements**

**Problem Title:** Developing a Food Ordering Application

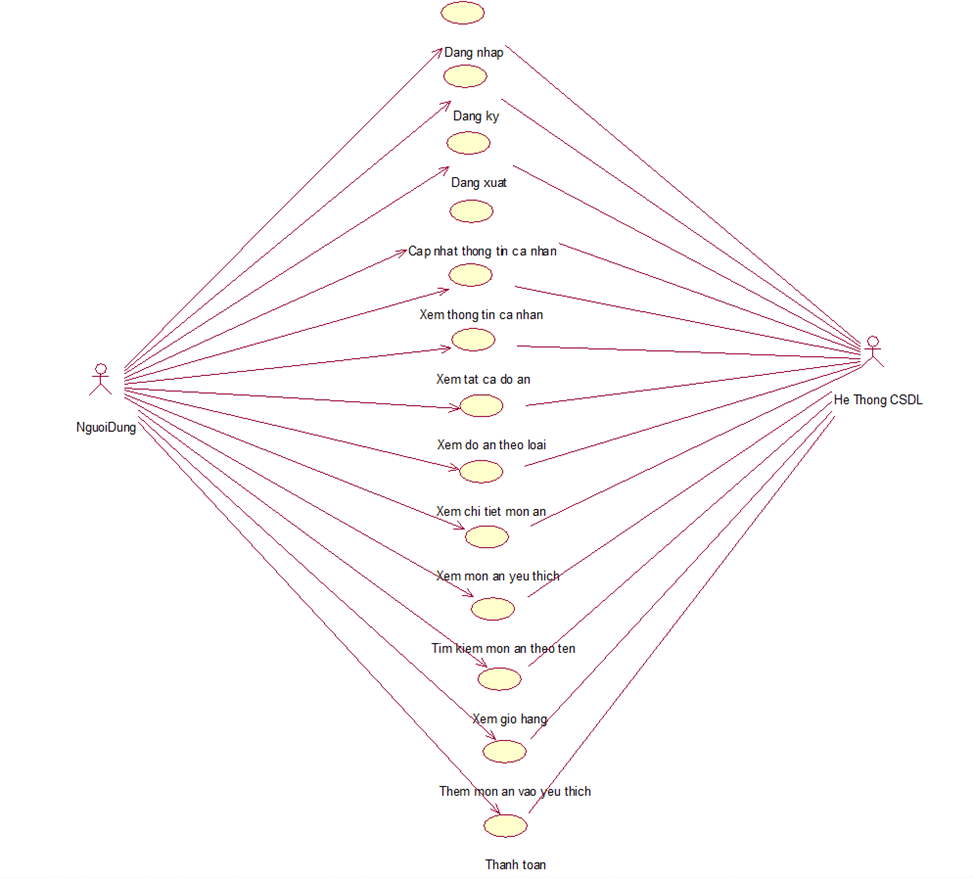
One of the main reasons is that modern life is increasingly busy, and many people do not have enough time or energy to cook at home. The food ordering application helps alleviate this burden by providing convenience, flexibility, and speed in the ordering process.

## **Construction**

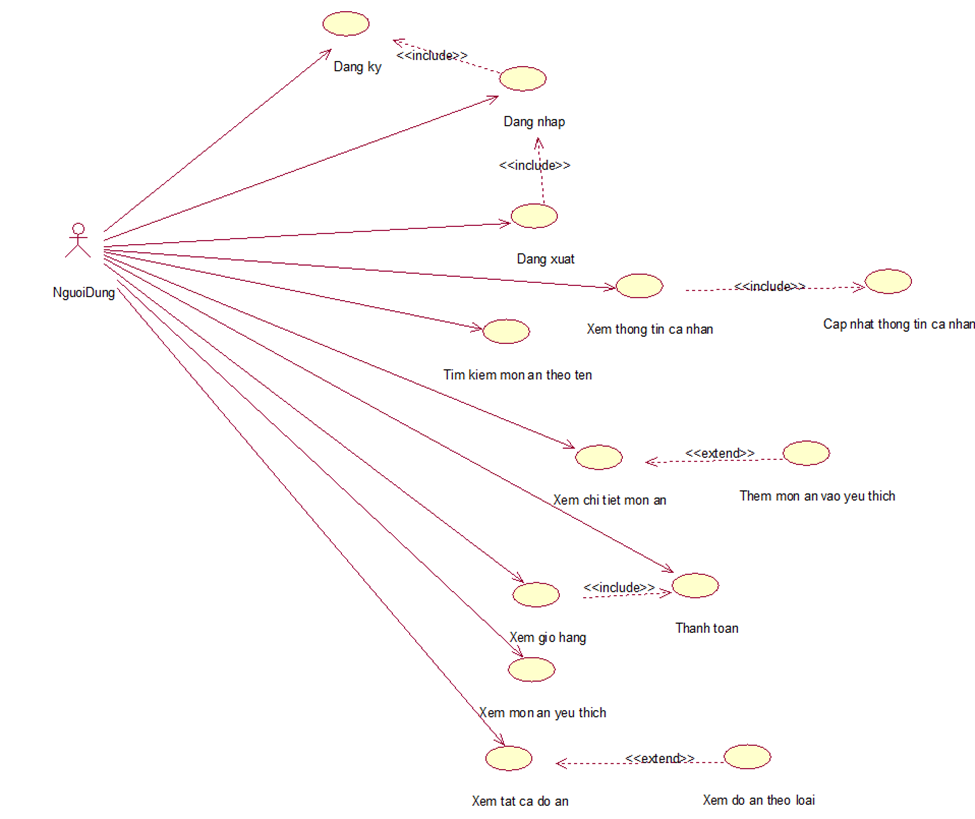
### **User requirements analysis**

Functional Requirements:

* Sign Up, Login
* Log Out
* View Food Items
* View Food Item Details
* Add Food Item to Cart
* Edit Shopping Cart
* Payment
* Edit, and Update Personal Information



*Image 1: Use Case Diagram*



*Image 2: Relationships Between Use Cases*

Summary of Tasks in Functional Use Cases (UC):

* Login: Allows users to log in to the OrderFood application.
* Log Out: Enables users to log out of their account within the application.
* Sign Up: Allows users to register a new account in the OrderFood application.
* Update Personal Information: Allows users to update their personal information, including Fullname, Address, and Phone number.
* View Personal Information: Allows users to view their personal information, including Image, Username, Fullname, Address, and Phone number.
* View All Food Items: Enables users to view all available food items in the database.
* View Food Items by Category: Allows users to view information about food items categorized in the database.
* View Food Item Details: Allows users to view detailed information about a selected food item, including image, name, price, and description.
* View Favorite Food Items: Allows users to view their favorite food items stored in the database.
* Add Favorite Food Item: Allows users to add selected food items to their list of favorites.
* Search for Food Item by Name: Allows users to search for a specific food item by its name.
* View Shopping Cart: Enables users to view information about the items in their shopping cart.
* Proceed to Payment: Allows users to proceed with the payment for the food items in their shopping cart.

#### **Use case Log In:**

* Use Case Name: Log In
* Brief Description: This use case allows users to log in to the application using a pre-registered account.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user opens the application, and the system displays the login screen with username and password fields.

2) The user enters the username and password, then selects "Log In." The system checks the login information in the User table and navigates to the next screen.

2.1. Successful Log in: If the username and password exist in the User table, the system redirects the user to the Home screen.

2.2. Failed Log In: If the username and password do not exist in the User table, the system displays a message: "Incorrect username or password.".

* + **Alternate Flow:**

3) At any point during the use case execution, if the system cannot connect to the database, it displays an error message, and the use case ends

#### **Use case Log Up:**

* Use Case Name: Sign Up
* Brief Description: This use case allows users to register an account to log in to the application.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user selects "Sign Up" on the login screen. The system displays the registration screen with fields for first name, last name, username, password, image, phone number, and address.

2) The user enters the information and selects "Sign Up." The system checks the registration information in the User table and navigates to the next screen.

2.1. Successful Registration: If the username does not exist in the User table, the system displays a message: "Registration successful!"

2.2. Failed Registration: If the username already exists in the User table, the system displays a message: "Username already exists!"

* + **Alternate Flow:**

3) At any point during the use case execution, if the system cannot connect to the database, it displays an error message, and the use case ends.

#### **Use case Log Out:**

* Use Case Name: Log Out
* Brief Description: This use case allows users to log out of the system.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user clicks on the "Log Out" button on the user information screen. The system displays a confirmation message for logging out.

2) The user clicks the "Agree" button. The system deletes the user's login information and returns to the login screen. The use case ends.

* + **Alternate Flows:**

3) At step 2 in the basic flow, if the user clicks the "Cancel" button, the system does not delete the user's login information.

4) At any point during the use case execution, if the system cannot connect to the database, it displays an error message, and the use case ends.

#### **Use case View All Food Items:**

* Use Case Name: View All Food Items
* Brief Description: This use case allows users to view all available food items.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user clicks on the "All" button on the main screen. The system retrieves the names, illustrations, and prices of all food items from the FOOD table and displays them on the screen. The use case ends.

* + **Alternate Flows:**

2) At any point during the use case execution, if the system cannot connect to the database, it displays an error message, and the use case ends.

#### **Use case View Food by Category:**

* Use Case Name: View Food by Category
* Brief Description: This use case allows users to view food items based on different categories.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user clicks on one of the buttons representing different food categories on the main screen. The system retrieves the names, illustrations, prices, and menu\_id of the food items with a menu\_id matching the category's menu\_id from the FOOD table and displays them on the screen. The use case ends.

* + **Alternate Flows:**

2) At any point during the use case execution, if the system cannot connect to the database, it displays an error message, and the use case ends.

#### **Use case Add to Cart:**

* Use Case Name: Add to Cart
* Brief Description: This use case allows users to add food items to the cart to continue the ordering process within the food ordering application.
* Event Flows:
  + **Basic Flow:**

1. The use case starts when the user selects a food item from the menu list. The system adds the selected food item to the user's cart. The cart is updated with the new food item. The use case ends.
   * **Alternate Flows:**

3) If unsuccessful in adding to the cart (e.g., connection error), the system displays an error message, and the use case ends.

4) At any point during the use case execution, if the system cannot connect to the database, it displays an error message, and the use case ends.

#### **Usecase Order Payment**

* Use Case Name: Order Payment
* Brief Description: This use case allows users to proceed with the payment of their order within the food ordering application.
* Event Flows:
  + **Basic Flow:**

1. The use case starts when the user views their cart and selects the payment option. The system requests the user to enter payment information and delivery address (if needed).
2. The user confirms the payment information and clicks the "Confirm Payment" button. The system processes the payment, notifies the user of the order status, and the use case ends.
   * **Alternate Flows:**

3) If unsuccessful in processing the payment (e.g., connection error), the system displays an error message, and the use case ends.

4) The user can retry or choose an alternative payment method. The use case continues.

#### **Usecase Update Personal Information**

* Use Case Name: Update Personal Information
* Brief Description: This use case allows users to update their personal information within the application.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user selects the "Update Personal Information" option in the account information.

2) The system displays the current personal information template of the user.

3) The user updates the necessary information (address, phone number, etc.).

4) The user clicks the "Save" button to save the changes.

* + **Alternate Flows:**

5) If unsuccessful in the updating process (e.g., connection error), the system displays an error message.

5.1) The user can retry or come back later.

#### **Use case View Cart:**

* Use Case Name: View Cart
* Brief Description: This use case allows users to view the food items and quantities in their shopping cart.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user selects the shopping cart icon at the bottom of the screen.

2) The system displays the shopping cart screen.

3) The user views the items in the cart and can increase or decrease the quantity of food items.

4) The user can press "Checkout" to proceed to payment.

* + **Alternate Flows:**

5) If there is a connection error with the database, the data cannot be displayed.

#### **Use case Search by Name:**

* Use Case Name: Search by Name
* Brief Description: This use case allows users to search for food items by name.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user selects the search bar and enters the name of the desired food item.

2) The system redirects to the search screen and displays the food items matching the user's input.

3) The user can add items to the cart or mark them as favorites.

* + **Alternate Flows:**

1) If there is a connection error with the database, the data cannot be displayed.

2) If no food items match the search, the screen will show no results.

#### **Use case Add to Favourites:**

* Use Case Name: Add to Favorites
* Brief Description: This use case allows users to add food items they like to their favorites list.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user is on screens such as "View All," "Chicken," "Korean," "Drink," or searching for food items, and the user taps the heart icon on each food item.

2) The system changes the color of the heart icon to red.

3) The selected food item is added to the favorites list.

4) The user can view favorite items in the "Favorites" section.

* + **Alternate Flows:**

1) If there is a connection error with the database, the food item will not be added to the favorites.

#### **Usecase View Food Details:**

* Use Case Name: View Food Details
* Brief Description: This use case allows users to view detailed information about a selected food item, including images, name, price, and description.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user clicks on the "View Details" button for a food item on the main screen.

2) The system retrieves detailed information about the food item from the database and displays it on the screen, including images, the name of the food, price, quantity, and buttons "Add to Cart" and "Favorite."

* + **Alternate Flows:**

3) If unsuccessful in adding to the cart (e.g., connection error), the system displays an error message, and the use case ends.

#### **Usecase View Favourite Foods:**

* Use Case Name: View Favorite Foods
* Brief Description: This use case allows users to view the food items they have added to their favorites list.
* Event Flows:
  + **Basic Flow:**

1) The use case starts when the user clicks on the "Favorites" button on the navigation bar.

2) The system retrieves the list of favorite food items associated with the user's account and displays them on the screen, including images, the name of the food, price, and the "Add to Cart" button.

* + **Alternate Flows:**

3) If there are no food items in the favorites list, the system displays a message indicating an empty list.

4) At any point during the use case execution, if there is no connection to the database, the system displays an error message, and the use case ends.

#### **Usecase View Personal Information:**

* Use Case Name: View Personal Information
* Brief Description: This use case allows users to view detailed personal information, including images, username, full name, address, and phone number.
* Event Flows:
  + **Basic Flow:**

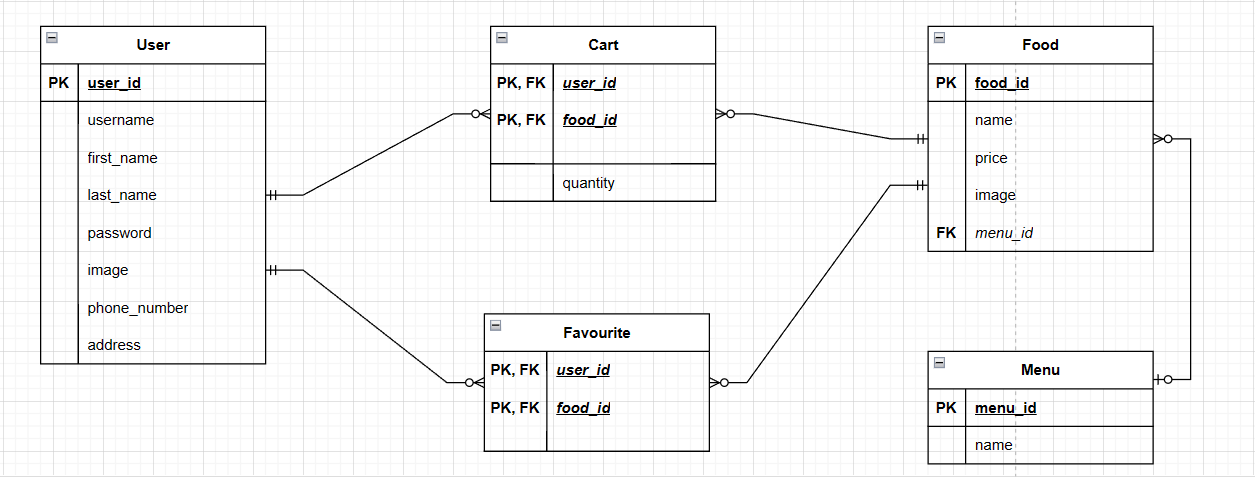
1) The use case starts when the user clicks on the "Personal Information" button on the navigation bar.

2) The system retrieves detailed user information from the database and displays it on the screen, including images, full name, address, and phone number.

* + **Alternate Flows:**

3) At any point during the use case execution, if there is no connection to the database, the system displays an error message, and the use case ends.

### **Description of data class diagram**



* User table: stores information about users.
* Food table: stores information about food items.
* Cart table: stores information about the user’s shopping cart.
* Favourite table: stores information about user favourites.
* Menu table: stores information about menus.

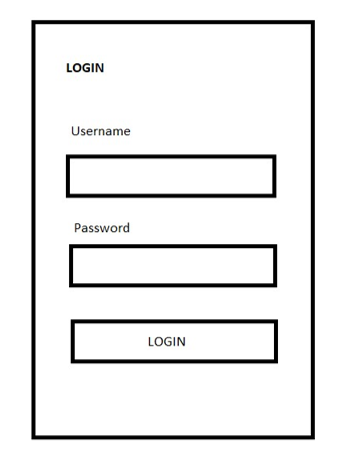
## **System design**

- Rules for Application Development:

* Color Scheme:
  + Use accent colors to highlight important elements or primary actions. For example, use red for "Delete" buttons or green for "Add" buttons.
* Size:
  + Define clear sizes and spacing between elements on the interface, ensuring balance and readability.
* Naming conventions: Use clear and understandable names for files, reflecting their content. Different types of files or folders may have distinct naming conventions.
  + Package: use lowercase letters without spaces.
  + Class: Follow PascalCase conventions. For example, if it's an adapter, name it NameAdapter. If it's an object, name it NameModel. For activities, use Name.
  + Follow snake\_case conventions. For example, if it's an activity, name it activity\_name. If it's a listView, name it lv\_name.

### **Interface description**

#### **Log In screen**

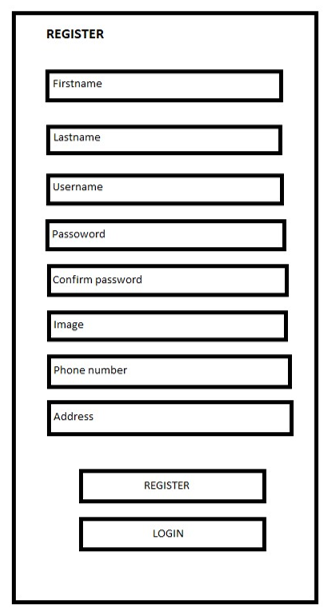


The login screen consists of 2 editText fields for users to enter account information and passwords.

There are labels to guide users in entering account information.

A button allows users to log in and access the main screen.

#### **Sign Up screen:**

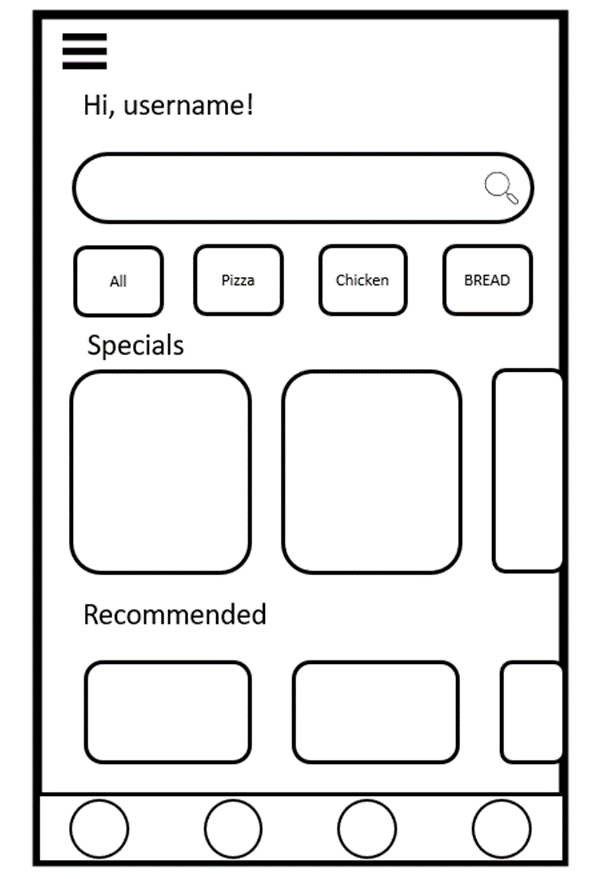


The registration screen includes 8 editText fields for users to enter account information, including: last name, first name, username, password, avatar image, phone number, and address.

There are labels to guide users in entering information.

There is a button to submit the registration and another button to return to the login screen.

#### **Home screen:**

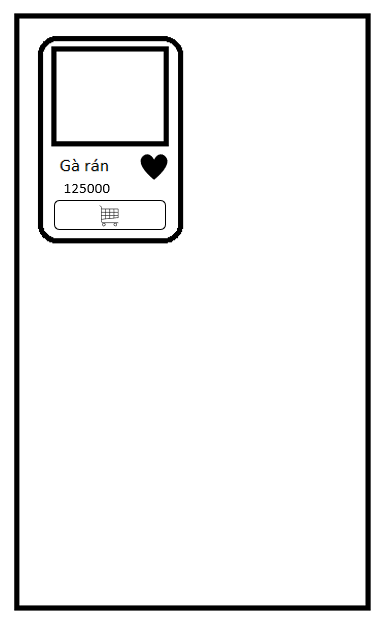


The Home screen includes a SearchView bar for users to search for food by name.

There are buttons to navigate to screens displaying food items by category or showing all food items.

There are two RecyclerViews to display special dishes and suggested items each time the user refreshes the application.

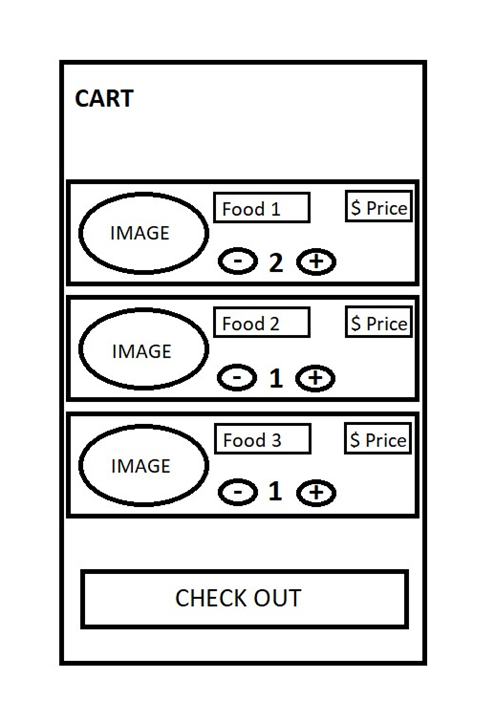
#### **Favorite screen:**



The Favorites screen consists of one RecyclerView to display the list of favorite food items for that user.

Within this RecyclerView, there are layout items for each favorite food item, including an image, the food's name, its price, an icon to remove it from the favorites list, and an icon to add it to the shopping cart.

#### **Cart screen:**

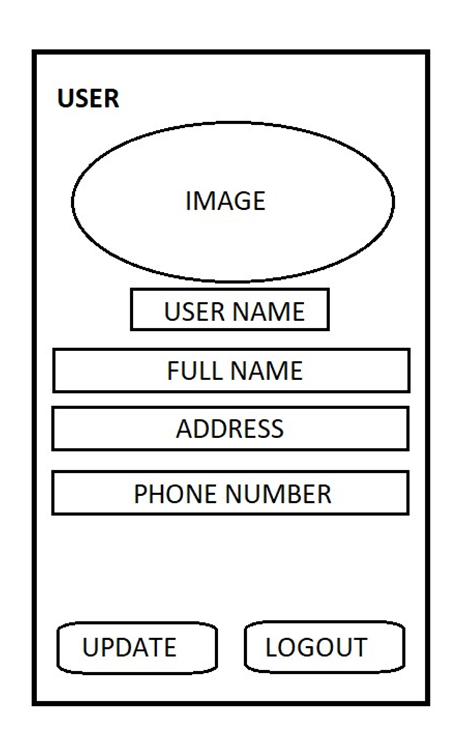


The registration screen includes a RecyclerView to display a list of food items. Each item in the RecyclerView contains an ImageView to show the image, three TextViews to display the name, price, and quantity, two Buttons to adjust the quantity, and one Button for payment.

Labels are included to guide users in entering information.

There are three Buttons to adjust the quantity and proceed to payment.

#### **User screen:**



The user screen includes four EditText fields to display user information, including username, full name, address, and phone number. It is also used to update the user's personal information.

Labels are provided to guide users in entering account information.

Two buttons are available for updating and logging out of the application.

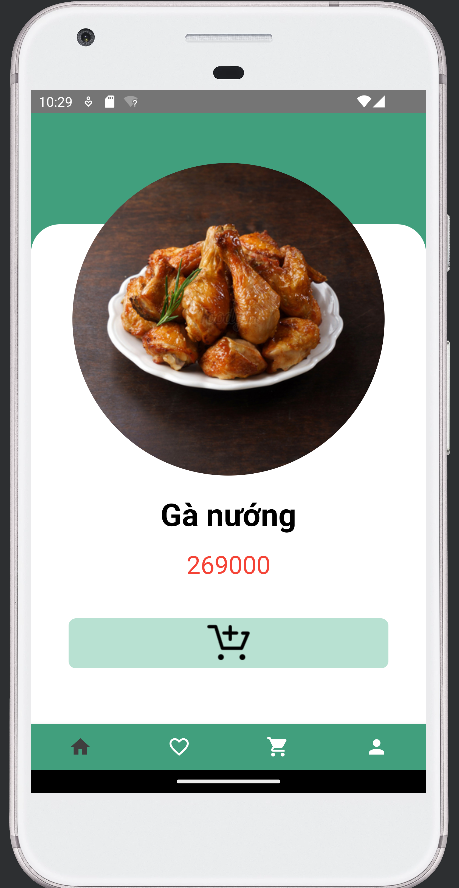
#### **View All screen:**

* The "View All" screen displays all the food items available in the store.
* The screen consists of food items, each displaying an image, the food name, price, a favorite button, and an "Add to Cart" button.
* Users can add or remove items from their favorites by selecting the favorite button.
* They can add items to the shopping cart by pressing the "Add to Cart" button.
* Users can view detailed information about a food item by clicking on its image.

#### **View by Category:**

* The "View by Category" screen is accessed by clicking on the chicken, Korean, or drink icons on the Home screen.
* This screen displays food items corresponding to the selected category in a two-column layout.
* Each item includes an image, food name, price, a favorite button, and an "Add to Cart" button.
* Users can add or remove items from their favorites by selecting the favorite button.
* They can add items to the shopping cart by pressing the "Add to Cart" button.
* Users can view detailed information about a food item by clicking on its image.

#### **Details screen:**



* When on the Home screen, if the user clicks the "SEE DETAILS" button for a specific category, the system will display detailed information about that food category.
* The information includes the food item's image, name, price, and an "Add to Cart" button.
* Users can add the item to the shopping cart by clicking the "Add to Cart" button.

#### **Màn hình thanh toán:**

#### 

* When on the Cart screen, if the user presses the "CHECKOUT" button, the system will navigate to the Payment screen.
* The Payment screen includes options to select the payment method, information about the total order value to be paid by the user, and the "CONFIRM PAYMENT" button.
* Once the user clicks the "CONFIRM PAYMENT" button, the system will display the order status on the screen.

## **Implementation of the problem**

### **Bui Duc Luong - Cart screen**

* **Screen:**
* **The purpose of Cart Activity:**

The purpose of the Cart Activity is to display the food items added to the cart along with their quantities. The screen consists of individual food item entries, each containing an image, food name, price, quantity, and two buttons, "+" and "-", to increase or decrease the quantity. If the user wishes to remove a food item from the cart, they can decrease the quantity to 0, and conversely, if they want to increase the quantity, they can press "+" until they reach the desired amount. The "Checkout" button is used when the user wants to proceed with the payment for the items in the cart.

* **The main algorithm on the screen CartFragment**

package com.example.orderfood.uis;

public class CartFragment extends Fragment {

DBHelper dbHelper;

CartRepository cartRepository;

List<Cart> list;

RecyclerView recyclerView;

CartAdapter cartAdapter;

Button btnCheckOut;

public CartFragment() {

// Required empty public constructor

}

public static CartFragment newInstance(String param1, String param2) {

CartFragment fragment = new CartFragment();

Bundle args = new Bundle();

args.putString(ARG\_PARAM1, param1);

args.putString(ARG\_PARAM2, param2);

fragment.setArguments(args);

return fragment;

}

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

if (getArguments() != null) {

mParam1 = getArguments().getString(ARG\_PARAM1);

mParam2 = getArguments().getString(ARG\_PARAM2);

}

}

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

// Inflate the layout for this fragment

return inflater.inflate(R.layout.fragment\_cart, container, false);

}

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

recyclerView=view.findViewById(R.id.rcvCart);

btnCheckOut=view.findViewById(R.id.btnCheckOut);

LinearLayoutManager linearLayoutManager=new LinearLayoutManager(this.getContext(),RecyclerView.VERTICAL,false);

recyclerView.setLayoutManager(linearLayoutManager);

dbHelper=new DBHelper(getContext());

cartRepository=new CartRepository(dbHelper,getContext());

list=cartRepository.getFoodsInCartByUserId(PrefManager.getUserId(getContext(),"username"));

cartAdapter=new CartAdapter(list,getContext());

recyclerView.setAdapter(cartAdapter);

btnCheckOut.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

NavController navController = NavHostFragment.findNavController(CartFragment.this);

Bundle bundle = new Bundle();

bundle.putInt("money", cartRepository.money());

navController.navigate(R.id.action\_cartFragment\_to\_payFragment, bundle);

}

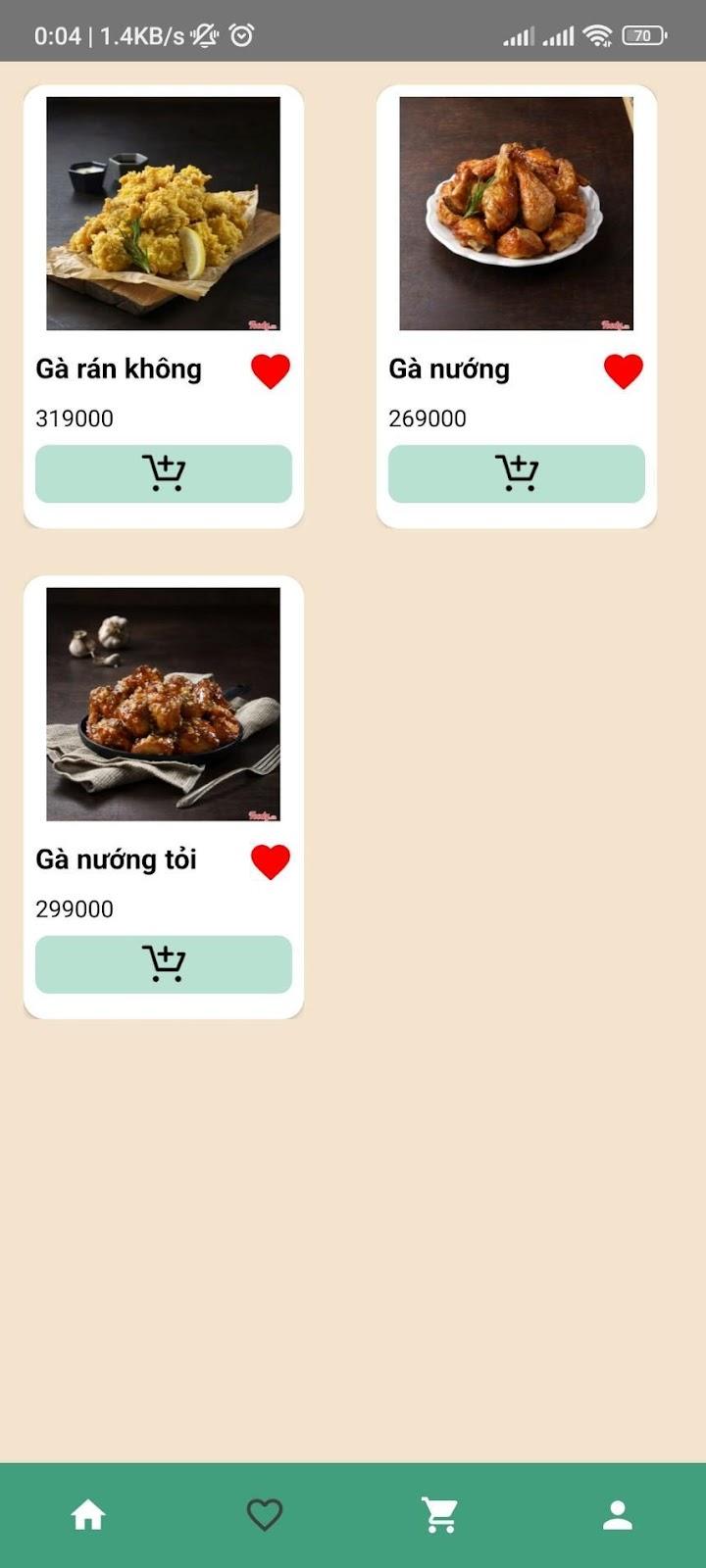
});

}

}

### **Mai Cong Sao – Favourite screen**

* **Screen:**

****

* **The purpose of Activity:**

The "Favorite" activity is designed to display the user's favorite dishes. Each displayed dish includes an illustration, dish name, dish price, an icon allowing users to remove the dish from their favorites when clicked, and an icon to add the dish to the cart. When users click on a dish, they are directed to the detailed view screen, where information about the dish is displayed. This detailed information provides users with more insights into the selected dish. The primary functionalities of this activity include viewing, managing, and navigating to detailed information for the user's favorite dishes.

* **The main algorithm on the screen:**

**FavoriteFragment:**

package com.example.orderfood.uis;

public class FavoriteFragment extends Fragment implements IClick {

private String mParam2;

List<Food> favoriteList;

FoodAdapter foodAdapter;

RecyclerView rcv;

DBHelper dbHelper;

FavouriteRepository favouriteRepository;

FoodRepository foodRepository;

UserRepository userRepository;

public static FavoriteFragment newInstance(String param1, String param2) {

FavoriteFragment fragment = new FavoriteFragment();

Bundle args = new Bundle();

args.putString(ARG\_PARAM1, param1);

args.putString(ARG\_PARAM2, param2);

fragment.setArguments(args);

return fragment;

}

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

if (getArguments() != null) {

mParam1 = getArguments().getString(ARG\_PARAM1);

mParam2 = getArguments().getString(ARG\_PARAM2);

}

}

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

// Inflate the layout for this fragment

return inflater.inflate(R.layout.fragment\_favorite, container, false);

}

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

rcv = view.findViewById(R.id.rcvFavoriteFood);

dbHelper = new DBHelper(getContext());

foodRepository=new FoodRepository(dbHelper);

favouriteRepository = new FavouriteRepository(dbHelper);

List<Integer> list=new ArrayList<>();

list=favouriteRepository.getFavorite(1);

for(int i=0;i<list.size();i++){

Log.d("TAG", "favorite: "+list.get(i));

}

userRepository=new UserRepository(dbHelper);

String username = PrefManager.getString(getContext(), "username");

User user=userRepository.getUserByUsername(username);

// favoriteList = favouriteRepository.getFoodsByUserId(user.getId());

favoriteList=new ArrayList<>();

for(int i=0;i<list.size();i++){

favoriteList.add(foodRepository.getFoodByFoodId(list.get(i)));

}

for(int i=0;i<favoriteList.size();i++){

Log.d("TAG", "fasvorite: "+ favoriteList.get(i).getName());

}

foodAdapter = new FoodAdapter(favoriteList,getContext(),FavoriteFragment.this);

LinearLayoutManager linearLayoutManager=new LinearLayoutManager(this.getContext(),RecyclerView.VERTICAL,false);

GridLayoutManager gridLayoutManager =new GridLayoutManager(this.getContext(),2);

rcv.setLayoutManager(gridLayoutManager);

rcv.setAdapter(foodAdapter);

foodAdapter.notifyDataSetChanged();

}

@Override

public void onClickDeleteFavorite(int foodId, int pos) {

favoriteList.remove(pos);

foodAdapter.notifyDataSetChanged();

}

@Override

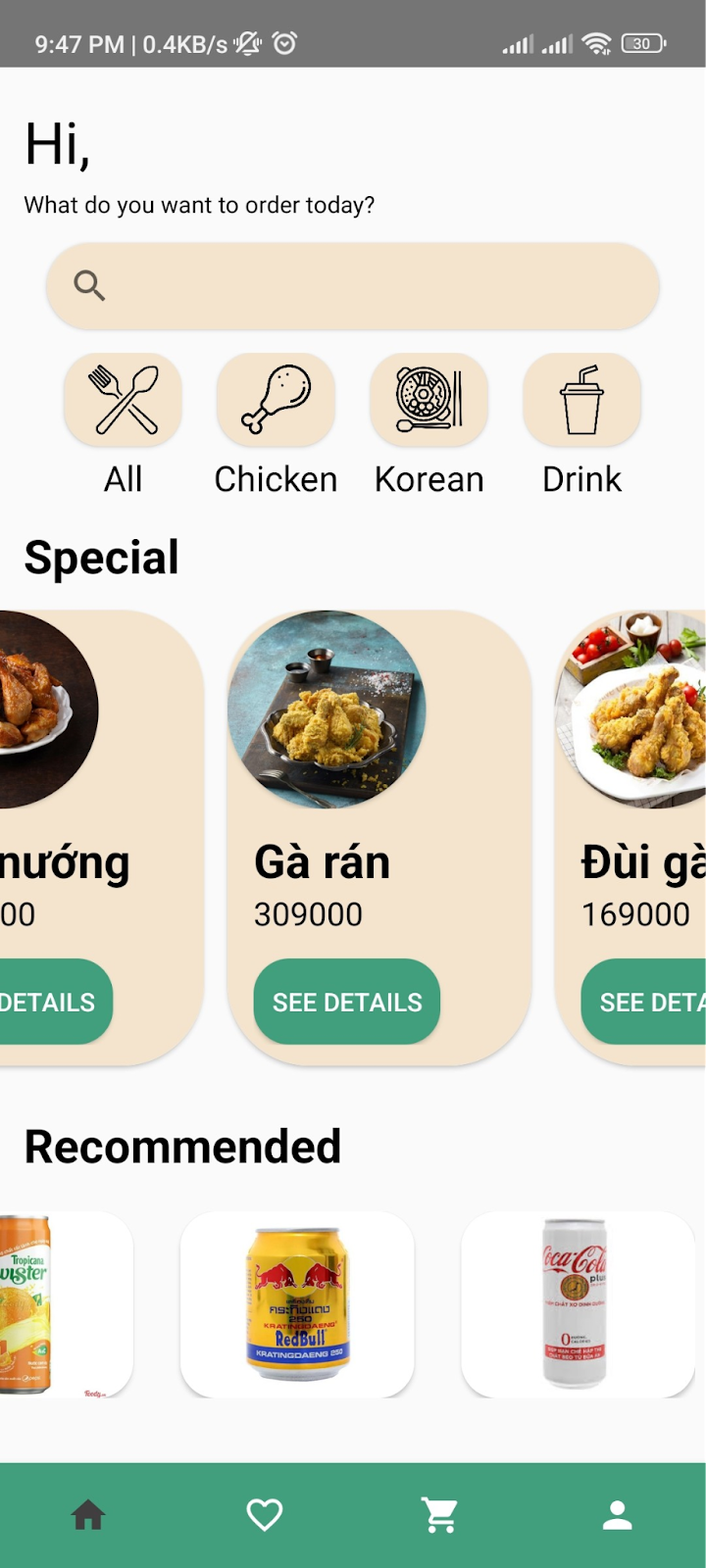
public void onClickFoodItem(Food food) {

}

}

### **Le Qui Long – Home Screen**

* **Screen:**

****

* **The purpose of Activity:**

The Activity Home is used to display general information about the application, including: 1 SearchView for users to search for dishes by name, 1 button 'All' for users to view the list of all dishes, and 3 buttons 'Chicken,' 'Korean,' 'Drink' to allow users to view the list of dishes by each category. The 'Special' section displays information about highlighted dishes, while 'Recommended' shows recommended dishes. When users click on a dish, they will be directed to the detail screen, where information about the dish will be displayed.

* **The main algorithm on the screen:**

**Home Fragment:**

public class HomeFragment extends Fragment {

DBHelper dbHelper;

List<Food> foodList;

List<SpecialContainer> specialList;

List<ImageView> recommendedList;

LinearLayout llChicken,llKorean,llDrink;

CardView cardView;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

// inflate layout

View view = inflater.inflate(R.layout.fragment\_home, container, false);

// create a db helper object and get all the food

dbHelper=new DBHelper(getContext());

foodList = new FoodRepository(dbHelper).getAll();

// create binding

getSpecialAndRecommend(view);

// display the food onto the screen

setSpecialAndRecommend();

return view;

}

public void setSpecialAndRecommend() {

final int quantity = 3;

for(int i = 0; i < quantity; i++) {

setSpecialDetail(specialList.get(i), getRandomFood());

setRecommenDetail(recommendedList.get(i), getRandomFood());

}

}

private Food getRandomFood() {

// get a random food from food list then remove it from the list to avoid duplicate

int index = (int)(Math.random()\*(foodList.size()-1));

Food food = foodList.get(index);

foodList.remove(index);

return food;

}

private void setSpecialDetail(SpecialContainer specialContainer, Food food) {

Glide.with(this).load(food.getImage()).into(specialContainer.getImageView());

specialContainer.getFoodName().setText(food.getName());

specialContainer.getFoodPrice().setText(food.getPrice() + " vnd");

specialContainer.getSeeDetail().setOnClickListener(view -> navigationToDetail(food.getId()));

}

private void setRecommenDetail(ImageView imageView, Food food) {

Glide.with(this).load(food.getImage()).into(imageView);

imageView.setOnClickListener(view -> navigationToDetail(food.getId()));

}

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

SearchView searchView = view.findViewById(R.id.home\_search\_view);

// navigate to all fragment when the user submit the search query, the search query is sent along

searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {

@Override

public boolean onQueryTextSubmit(String query) {

NavController navController = NavHostFragment.findNavController(HomeFragment.this);

Bundle bundle = new Bundle();

bundle.putBoolean("focusSearchView", true);

bundle.putString("query", query); navController.navigate(R.id.action\_homeFragment\_to\_allFragment, bundle);

return false;

}

});

cardView= view.findViewById(R.id.get\_all);

// navigate to all fragment when the user click on the all card view

cardView.setOnClickListener(v -> {

NavController navController = NavHostFragment.findNavController(HomeFragment.this);

Bundle bundle = new Bundle();

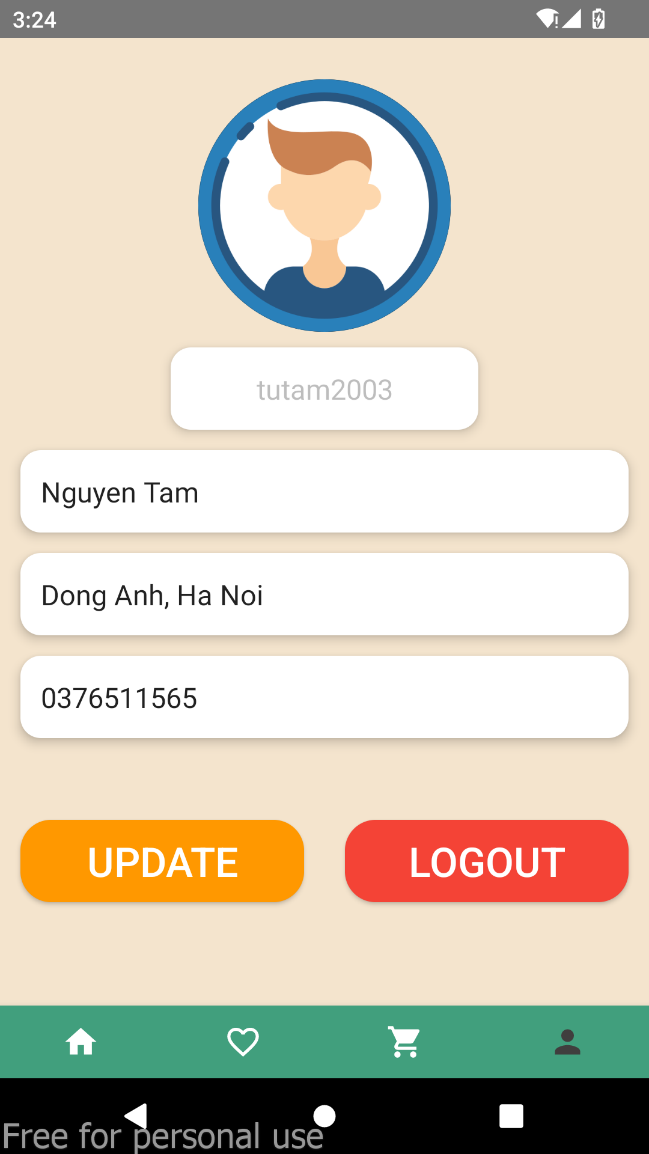
bundle.putBoolean("focusSearchView", false); navController.navigate(R.id.action\_homeFragment\_to\_allFragment, bundle);

});

}

### **Nguyen Tu Tam – User screen**

* **Screen:**



* **The purpose of Activity:**

The Activity User is used to display user information, including: 1 ImageView to show the profile picture. 4 EditText to display username, full name, address, and phone number. 2 Buttons to perform the tasks of updating and logging out.

* **The main algorithm on the screen:**

**User Fragment:**

public class UserFragment extends Fragment {

public UserFragment() {}

public static UserFragment newInstance(String param1, String param2) {

UserFragment fragment = new UserFragment();

Bundle args = new Bundle();

args.putString(ARG\_PARAM1, param1);

args.putString(ARG\_PARAM2, param2);

fragment.setArguments(args);

return fragment;

}

ImageView imageProfile;

EditText usernameProfile, fullnameProfile, addrProfile, phoneNumberProfile;

Button btnUpdate, btnLogout;

private static final int PICK\_IMAGE\_REQUEST = 1;

Uri imageUri = null;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

// Inflate the layout for this fragment

View view = inflater.inflate(R.layout.fragment\_user, container, false);

imageProfile = view.findViewById(R.id.image\_userProfile);

usernameProfile = view.findViewById(R.id.edt\_usernameProfile);

fullnameProfile = view.findViewById(R.id.edt\_nameProfile);

addrProfile = view.findViewById(R.id.edt\_addrProfile);

phoneNumberProfile = view.findViewById(R.id.edt\_phoneNumberProfile);

btnUpdate = view.findViewById(R.id.btnUpdate);

btnLogout = view.findViewById(R.id.btnLogout);

String username = PrefManager.getString(getContext(), "username");

DBHelper dbHelper = new DBHelper(getContext());

UserRepository userRepository = new UserRepository(dbHelper);

//Lấy thông tin User từ username

User user=userRepository.getUserByUsername(username);

getUserData(user);

imageProfile.setOnLongClickListener(new View.OnLongClickListener() {

@Override

public boolean onLongClick(View v) {

openGallery();

user.setImage(String.valueOf(imageUri));

return false;

}

});

btnLogout.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

logOut();

}

});

return view;

}

private void logOut() {

Intent intent = new Intent(requireContext(), RegisterActivity.class);

startActivity(intent);

requireActivity().finish();

}

private void getUserData(User userProfile) {

usernameProfile.setText(userProfile.getUsername());

fullnameProfile.setText(userProfile.getFirstname() +" "+ userProfile.getLastname());

addrProfile.setText(userProfile.getAddress());

phoneNumberProfile.setText(userProfile.getPhoneNumber());

}

}

**Class User in database:**

package com.example.orderfood.models;

public class User implements Serializable {

private int id;

private String username;

private String firstname;

private String lastname;

private String password;

private String image;

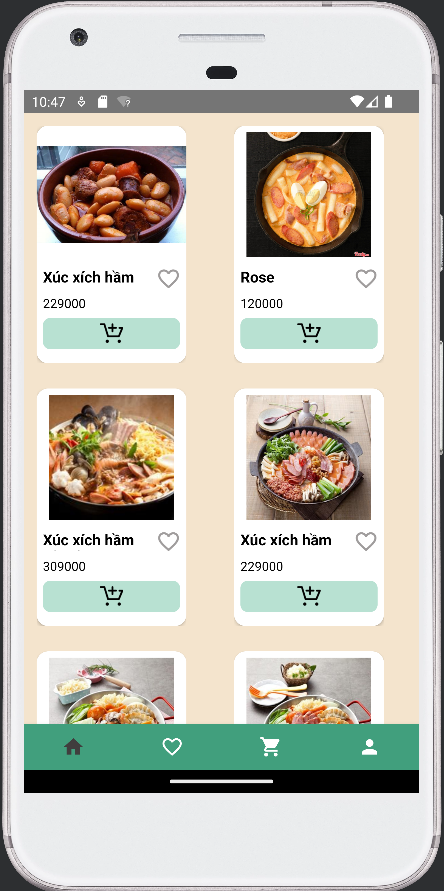
private String phoneNumber;

private String address;

}

### **Cao Hoang Long – Category’s foods screen**

* **Screen:**



* **The purpose of Activity:**

Activity Type được dùng để hiển thị thông tin các món ăn theo loại(ví dụ: Chicken, Korean, …). Các món ăn theo loại được hiển thị bao gồm ảnh minh họa, tên món ăn, giá món ăn, 1 icon cho phép người dùng xóa món ăn khỏi danh sách yêu thích khi kích vào, 1 icon thêm vào giỏ hàng. Khi người dùng click vào 1 món ăn, hệ thống sẽ chuyển đến màn hình detail, thông tin chi tiết về món ăn sẽ hiển thị

* **The main algorithm on the screen:**

From the home screen, each category will have a menuId so that when users click on a category, the system will recognize and switch to the screen of the selected category. Then the foods will be retrieved from the database with a menuId matching the menuId of the selected category. When users click on a food item to view details, a bundle will be used to send the key 'food\_id' to navigate to the details screen displaying detailed information about the food.

package com.example.orderfood.uis;

nager;

public class TypeFragment extends Fragment implements IClick {

List<Food> list;

FoodAdapter foodAdapter;

RecyclerView rcv;

DBHelper dbHelper;

CartRepository cartRepository;

FoodRepository foodRepository;

NavController navController;

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

*// Inflate the layout for this fragment*

return inflater.inflate(R.layout.*fragment\_type*, container, false);

}

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

navController= NavHostFragment.*findNavController*(TypeFragment.this);

rcv=view.findViewById(R.id.*rcvFood*);

dbHelper=new DBHelper(getContext());

foodRepository=new FoodRepository(dbHelper);

cartRepository=new CartRepository(dbHelper,getContext());

int menuId=getArguments().getInt("menu\_id");

list=foodRepository.getFoodByMenuId(menuId);

Log.*d*("TAG", "onViewCreated: "+list.get(4).getImage());

foodAdapter=new FoodAdapter(list,getContext(),TypeFragment.this);

LinearLayoutManager linearLayoutManager=new LinearLayoutManager(this.getContext(),RecyclerView.*VERTICAL*,false);

GridLayoutManager gridLayoutManager =new GridLayoutManager(this.getContext(),2);

rcv.setLayoutManager(gridLayoutManager);

rcv.setAdapter(foodAdapter);

}

@Override

public void onClickFoodItem(Food food) {

Bundle bundle=new Bundle();

bundle.putInt("food\_id", food.getId()); navController.navigate(R.id.*action\_typeFragment\_to\_detailsFragment*,bundle);

}

}

**Class Food in database:**

package com.example.orderfood.models;

public class Food implements Serializable {

private int id;

private String name;

private String price;

private String image;

private int menuId;

}

# **PART 3: ACQUIRED KNOWLEDGE AND LESSONS LEARNED**

## **Implemented content**

Through the team project, we have achieved several things:

- Acquired a solid understanding of handling events in an Android application, designing user interfaces, and working with SQLite to build a local database for the application.

- Developed skills in analyzing business requirements, constructing relationship diagrams, and creating class diagrams for an Android application.

- Enhanced teamwork skills.

- Acquired problem-solving skills.

- Expanded vocabulary in English related to information technology.

## **Achieved results**

Achievements in the course that team 1 has accomplished during the implementation of the Final Project:

* Proficient in implementing a simple problem using key components in mobile programming on the Android platform.
* Successfully implemented a specific problem on mobile devices, applying knowledge of mobile programming to solve real-world issues.
* Successfully deployed a real-world application on the Android platform, demonstrating the ability to apply acquired knowledge to develop highly practical applications.

## **Limitations of the topic**

Due to time constraints, our team has not been able to complete the project thoroughly, and we haven't had the opportunity to explore all the issues and challenges presented in the assignment.

## **Development directions**

* Integrate user interaction features with food ordering functionality.
* Implement category-based viewing feature, ...
* Continue to improve performance and optimize the application to ensure smooth operation on different mobile devices.

# **REFERENCES**

[1]. Android Developer Docs - <https://developer.android.com/docs>

[2]. Android Repository Pattern - <https://developer.android.com/codelabs/basic-android-kotlin-training-repository-pattern>

[3]. <https://xuanthulab.net/android-java/>