# GROUP05

# Awn Anw
# Software Architecture Document

**Version 1.1**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 05/08/23 | 1.0 | Initial version | Integrated context |
| 19/08/23 | 1.1 | Fill section 5, 6 | Tran Huy Ban, Ngo Phuoc Tai |
| | | | |
| | | | |

# Table of Contents


**1. Introduction** **4**

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions 4

1.4 References 4

1.5 Overview 4

1.6 Acronyms 4

**2. Architectural Goals and Constraints** **4**

**3. Use-Case Model** **5**

**4. Logical View** **7**

4.1 Component: Buyer’s UI 9

4.2 Component: Merchant’s UI 11

4.3 Component: Admin’s UI 12

4.4 Component: Application logic 13

4.5 Component: Database 14

**5. Deployment** **14**

**6. Implementation View** **14**

# Software Architecture Document

## 1. Introduction

### 1.1 Purpose

This document presents a comprehensive architectural overview of the system by using various architectural views to illustrate different aspects of the system. It aims to capture and communicate the significant architectural decisions that have been made on the system.

### 1.2 Scope

This SAD provides a high-level overview of the architecture of Awn Anw. It includes a summary of the main components, their interactions, and the key design principles of the system.

### 1.3 Definitions

**Awn Awn:** A food delivery web app which allows HCMUS-students to easily order food and get it delivered to them in minutes. This system also cooperates with merchants to serve a wide range of great food.

**Merchant:** is a food shop (partners of the web application) to provide food via website application (Awn Anw). These user's shops have to be located in district 5.

**Buyer:** is students (learning at University of Science) who use Awn Anw to order food.

**Admin:** is the team that manages the Awn Anw system. This user manages both information of buyer and merchant and also has responsibility to operate the system.

### 1.4 References


Sommerville, Software Engineering, 9th Edition, Addison Wesley, 2011

Pressman, Software Engineering, A Practitioners Approach, 5th Edition, McGraw Hill, 2001

Theoretical slides


### 1.5 Overview

Use-case model

Logical View

Deployment

Implementation View

### 1.6 Acronyms

**SAD**: Software Architecture Document

**UI:** User Interface

**CSS:** Cascading-Style Sheets, document that describes the appearance of web pages

**API**: Application Programming Interface, a protocol used as an interface to allow communication between different components

**CRUD:** Create, Read, Update, and Delete (CRUD) are the four basic functions that models should be able to do, at most.

## 2. Architectural Goals and Constraints

There are some important requirements and constraints that have a significant bearing on Awn Anw system:

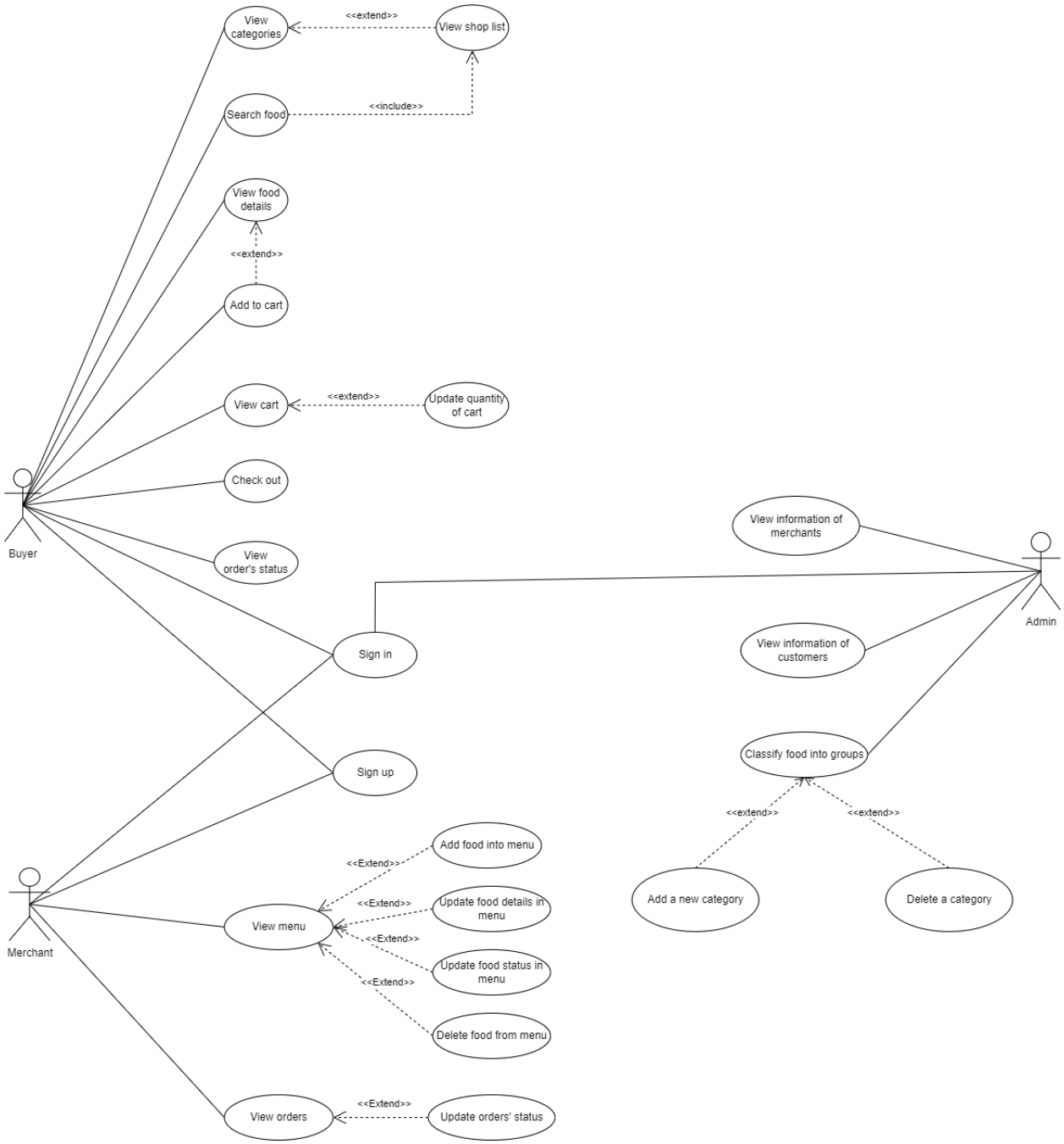
- The system shall support up to 100 users against the central database at any given time.
- The server shall be implemented by using Firebase services.
- The users' devices must connect to the Internet.

This section defines the quality ranges for performance, robustness, fault tolerance, usability and similar characteristics for the Awn Anw system:

- Availability: The System shall be available 24 hours a day, 7 days a week.
- Usability: The System should be easy to use and shall be appropriate for the target market of internet-literate students. Users should not require the use of a hardcopy manual to use the system.
- Maintainability: The system shall be designed for ease of maintenance. All users' data should be modifiable without recompilation of the System.
- Performance: The system shall response all the user's requirements in no more 0.0001 second
- User friendly: the final users easily move around the site to find what they are looking for. A clear and easy-to-follow structure is the most important.

## 3. Use-Case Model

Use-case diagram:


View categories
<<extend>>
View shop list
Search food
<<include>>
View food details
<<extend>>
Add to cart
View cart
<<extend>>
Update quantity of cart
Check out
View order's status
Buyer
Sign in
Sign up
View information of merchants
Admin
View information of customers
Classify food into groups
<<extend>>
<<extend>>
Add a new category
Delete a category
Add food into menu
<<Extend>>
Update food details in menu
<<Extend>>
View menu
<<Extend>>
Update food status in menu
<<Extend>>
Delete food from menu
Merchant
View orders
<<Extend>>
Update orders' status

# 4. Logical View

The architecture pattern used for this system is a layered architecture pattern. It includes 3 layers: User Interface (UI), Application Logic and Database (we use Firebase services for this layer). We consider them as 5 main packages of the system diagram, in which the layer UI splits into 3 packages which refers to 3 actors of the application: merchants, buyers and the administrators.

**a) UI**

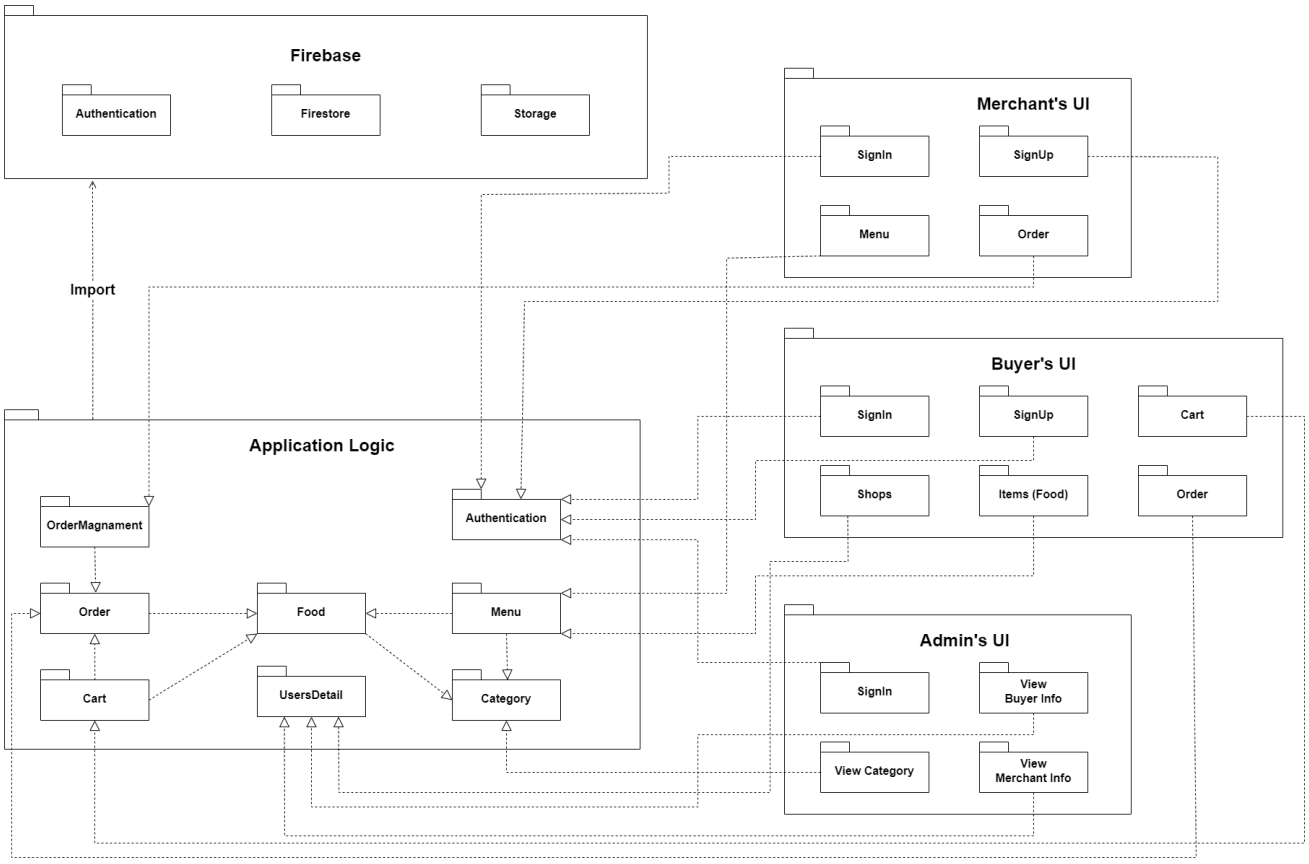
For the UI layer, the system has 3 packages including: buyer's UI, merchant's UI and admin's UI. However, the functionality provided by the UIs is relatively similar. UI will provide the user with a graphical interface for interacting with the system and also a place to receive the actions that the user performs. Then with user requests, the UI will send these requests down to the Application Logic layer for processing. In addition, we use ReactJS library and TailwindCSS framework to build UIs.

**b) Application Logic**

The Application Logic contains all the business-related logic and handles incoming requests. Typically, it responds by invoking a method within the model. We use the API provided by Firebase services to call down to the database to get data and process it according to the request of the corresponding function, and then return it to the application to display on the UI. In addition, we use Javascript to set up this layer.

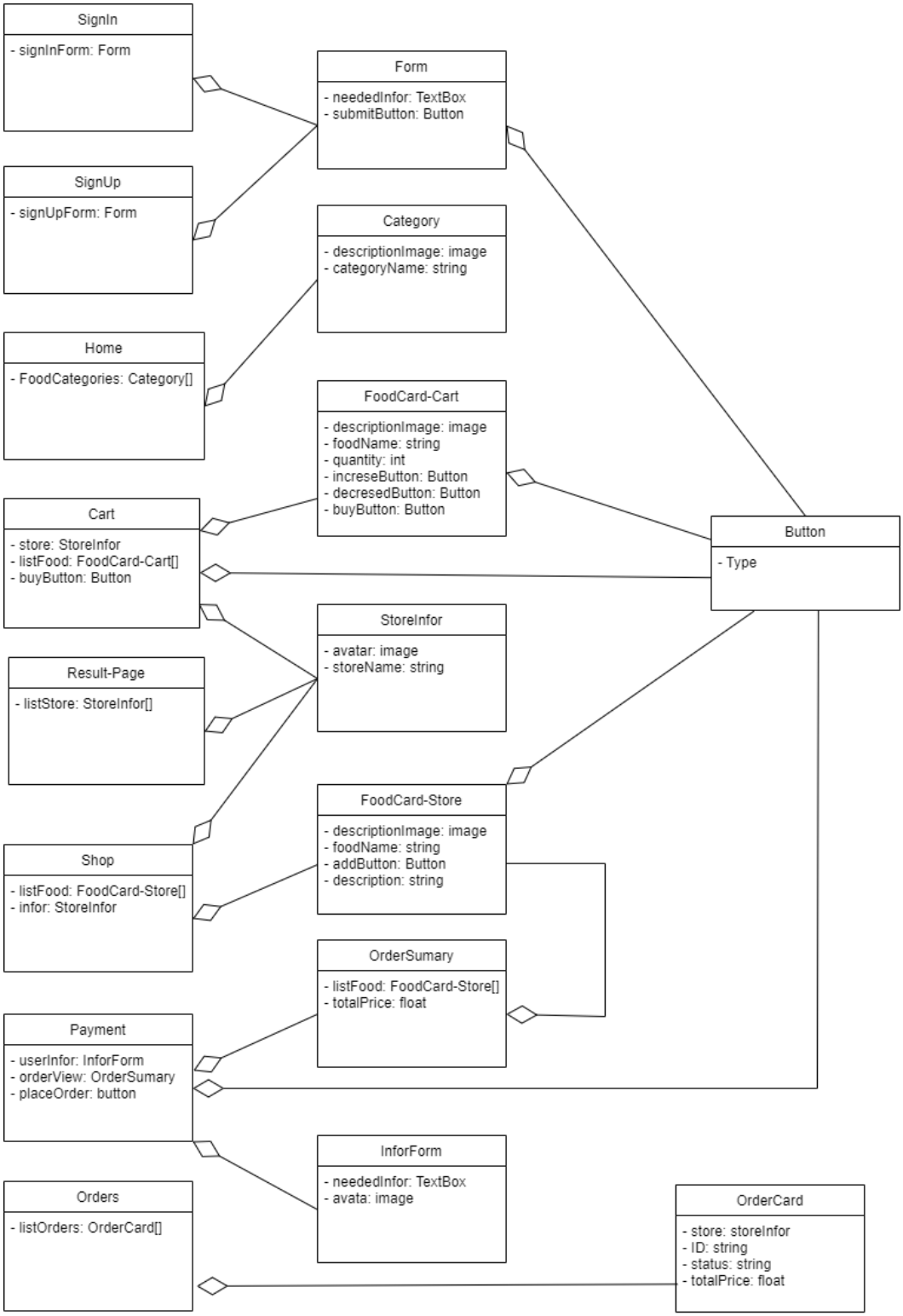
**c) Database**

The database used for this system is Firebase. Firebase is a fairly new tool for storing data for current software systems with a lot of support for logging, archiving, etc. It will be the place to store the data and receive requests from the application logic to perform CRUD operations of data and send it back to display it on the UI. It is an easy-to-use database that can work well with ReactJS as well as a good level of CRUD handling for small projects.




*Package Diagram of system*

### 4.1 Component: Buyer's UI




*Class diagram Buyer's UI*

Buyer's UI has key components below:

**Sign in**

- This component has a form with text-boxes and a submit button ("Sign in" button) for users to enter their account's information.

**Sign up**

- This component has a form with text-boxes and a submit button ("Sign up" button) for users to enter their personal's information to create an account.

**Home**

- This component has a list of categories. Each category includes a represented image and name of this category.

**Cart**

When buyer add a food in the cart, this component will contains 3 attributes as 3 components:

- The information of the store: name, image of this store.
- A list of food cards. Each card displays food information: images, name, description, quantity, corresponding price.
- A button for users to buy these foods.

**Result-Page**

- This component includes a list of stores which contain the buyers' needed food. Each store (attribute) includes its name and a represented image. Thus, this component displays stores' information after searching.

**Shop**

This component has 2 attributes as 2 components:

- The information of store: name, image of this store
- A list of food cards. Each card displays food information: images, name, description and a button for users to add this food into their cart.

**Payment**

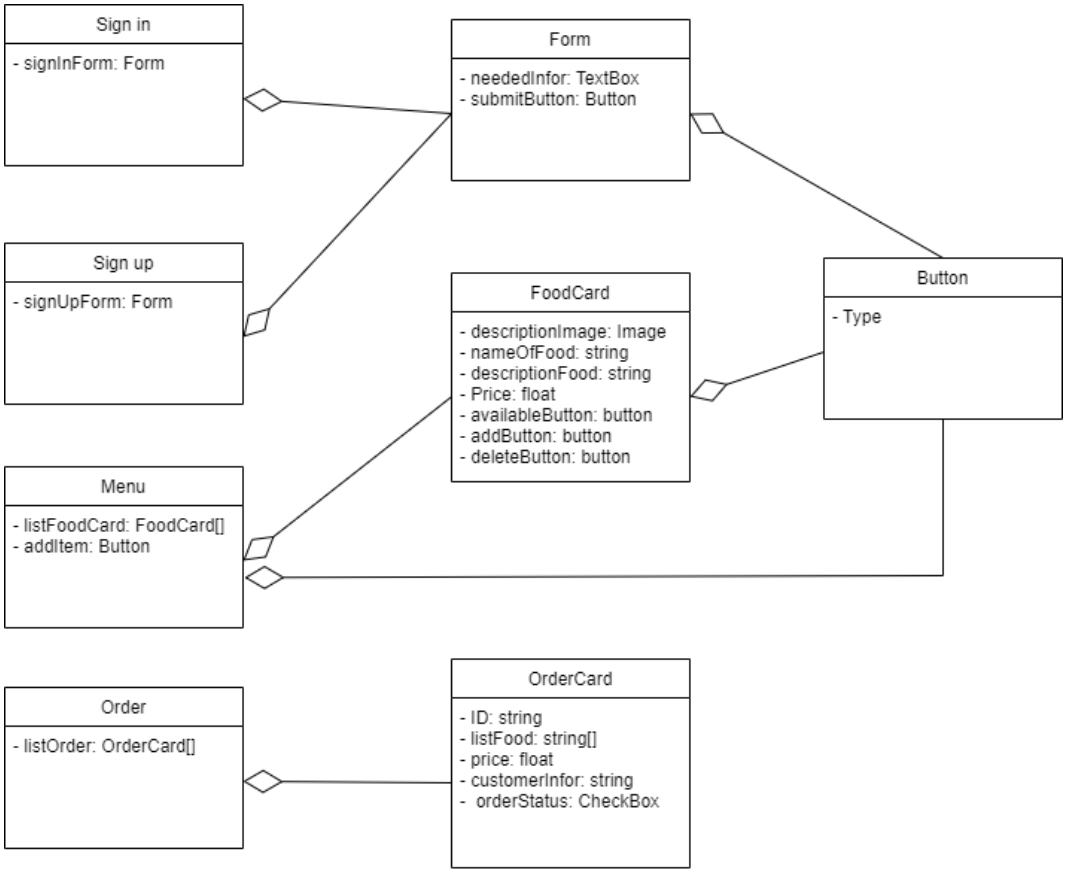
The payment component include 3 key attributes:

- A form for users to enter their information (such as: name, phone number,...)
- A list of foods in the order and corresponding price and the total price for this order.
- A button for users to place their order.

**Order**

- This component includes a list of order-cards. An order-card contains the ID of each order, the store information, status of this order, and the total price that the buyer has to pay.

### 4.2 Component: Merchant's UI




*Class diagram Merchant's UI*

Merchant's UI includes main components:

**Sign in**

- This component has a form with text-boxes and a submit button ("Sign in" button) for users to enter their account's information.

**Sign up**

- This component has a form with text-boxes and a submit button ("Sign up" button) for users to enter their personal's information to create an account.

**Menu**

This component includes a list of food cards and an "Add item" button.

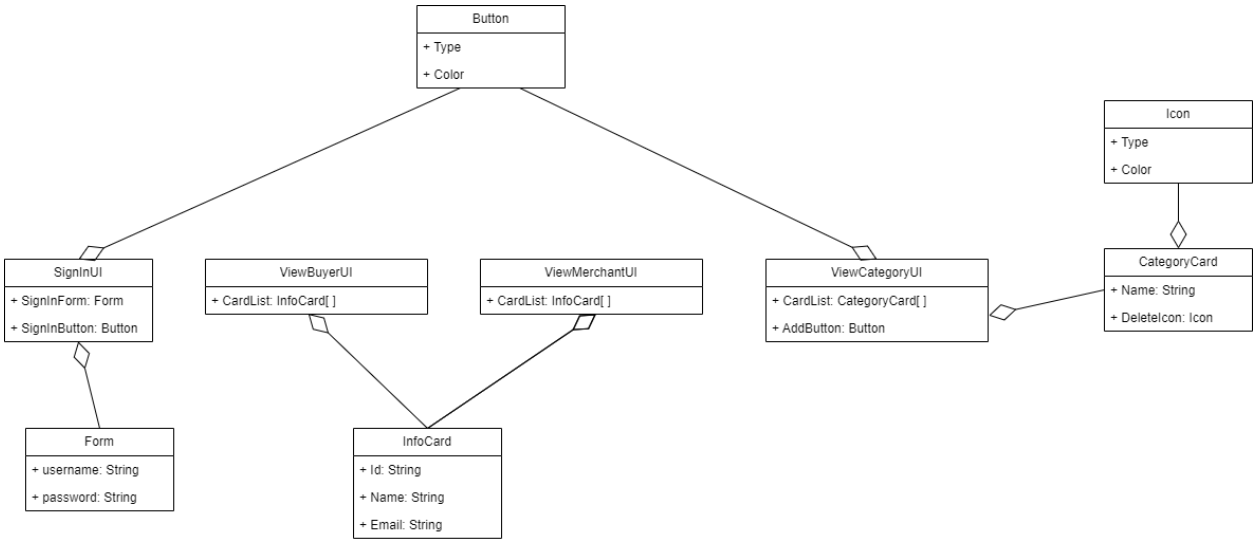
- The "Add item" button is used for adding a new food into the menu.
- The food card, it is a card has:
  + Information about food such as: images, name of food, description, price.
  + Available/Unavailable status.
  + A button to update food information.
  + A button to delete food from the menu.

**Order**

This component provides a list of order-cards for the shop to manage.

- The order-card includes the information of the order such as: ID of each order, list of food in order, price, customer information, status of the order.

### 4.3 Component: Admin's UI




*Class diagram Admin's UI*

Admin's UI includes main components:

**Sign In UI**

- This component has a SignInForm and a SignInButton**.**
- The SignInForm is a **Form** component that has 2 attributes for admins to enter their account's information:
  + username
  + password
- The SignInButton is a **Button** component for users to press after entering email and password.

**View Buyer UI**

- This component has a list of buyer information**.**
- Each buyer information is an **Info Card** component that has 3 attributes:
  + Id
  + Name
  + Email

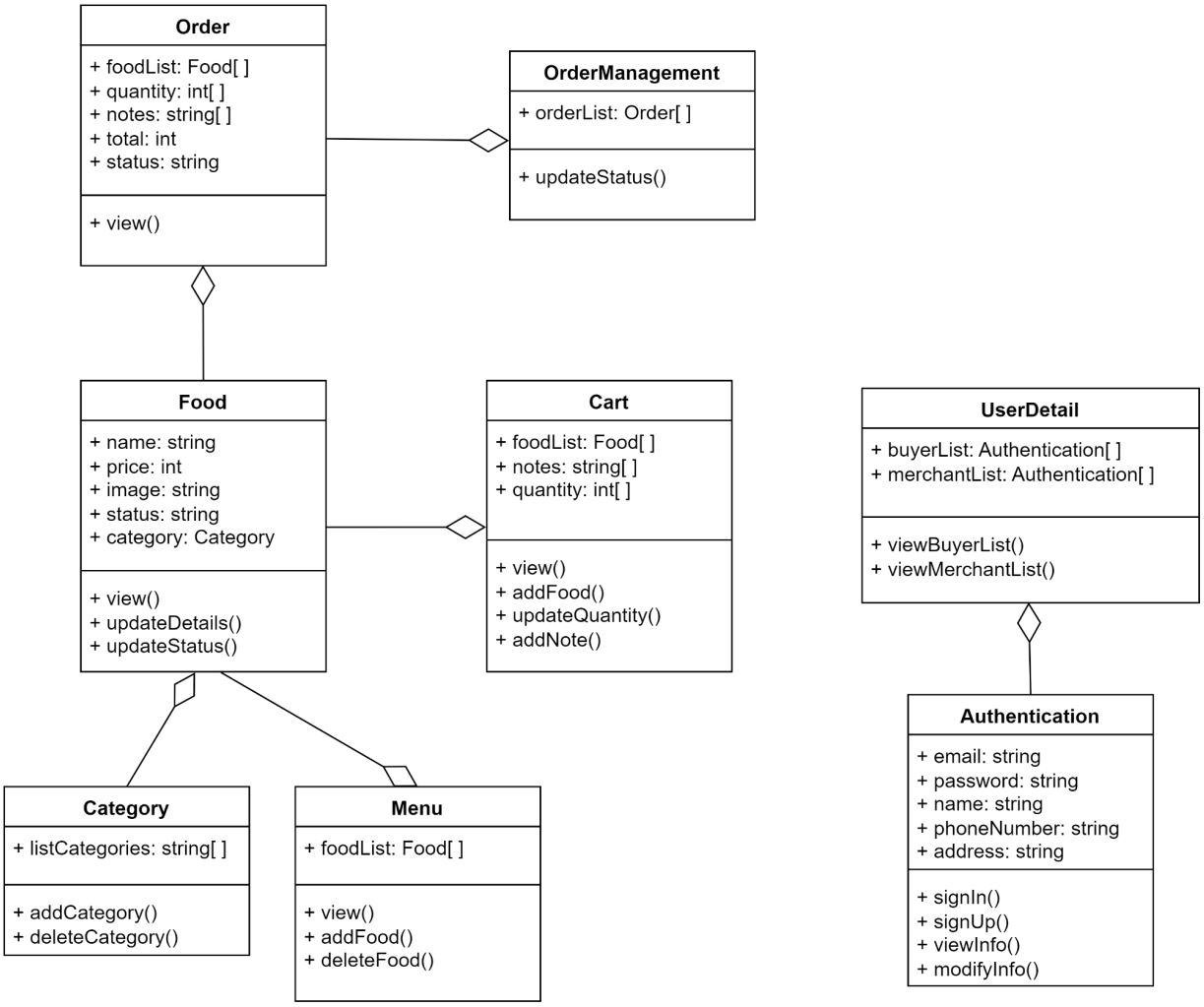
**View Merchant UI**

- This component has a list of merchant information.
- Each merchant information is an **Info Card** component that has 3 attributes:
  + Id
  + Name
  + Email

**View Category UI**

- This component has a list of categories and an "add" button.
- An "add" button is a **Button** component for users to press if they want to add more categories to the list.
- A list of categories is a **CategoryCard** component that has 2 attributes:
  + Category's Name.
  + A DeleteIcon is an **Icon** component for users to press if they want to delete a category.

### 4.4 Component: Application logic




*Class diagram Application logic*

Application logic package has main classes which are UserDetail, Authentication, Order, OrderManagement, Food, Cart, Category, Menu as shown in the above diagram.

**Authentication**

- This component provides authentication and authorization services for each user.

**UserDetail**

- This component includes a list of personal information of buyer and merchant for admin to manage and view. Each user information refers to the Authentication component.

**Food**

- The component provides logic functions for handling food in menus. It allows buyers to view food details and allows merchants to update food details or food status. Food class has an attribute which is a category relating to the category component.

**Category**

- The component is a list of categories managed by the administrators to divide food into their appropriate kinds.

**Menu**

- The component has a list of food components, which allows the merchants to add or delete food in

their menu. This component is shared for buyers to view and choose which food they want to add to their carts.

**Cart**

- This component provides buyers a cart to store their food chosen from a shop with notes. They can also view the cart or modify it.

**Order**

- The order component has information about the buyer's order including a list of their desired food relating to the food component. It allows the buyers and merchants to view.

**OrderManagement**

- This component controls the management of each order component. The merchants can update the status of the order.

### 4.5 Component: Database

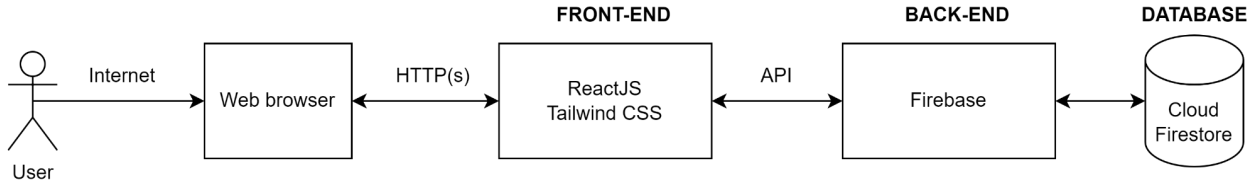
Utilizing the 'Firebase' web service plays a pivotal role in our application's architecture. It encompasses:

- **Authentication**: Enabling users to sign in/sign up using email credentials.
- **Firestore**: Storing data in organized collections and documents for quick retrieval.
- **Storage**: Safely storing images for app utilization.

Thanks to Firebase, we've crafted an app with a seamless user experience, efficient data management, and convenient image storage.

## 5. Deployment

This section shows the deployment view of Awn Anw application with various nodes as below:

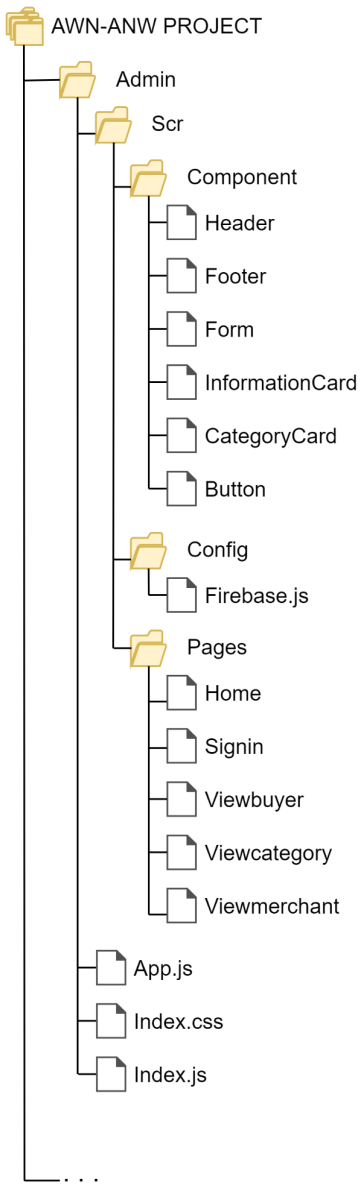


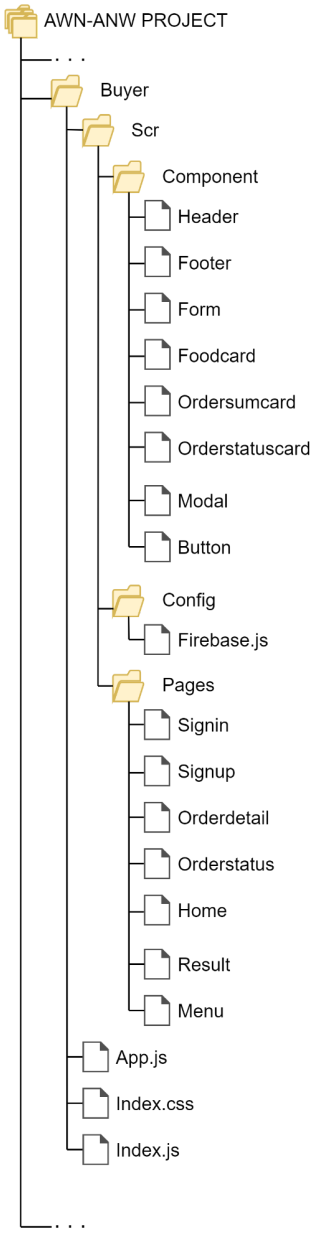

*Deployment diagram*

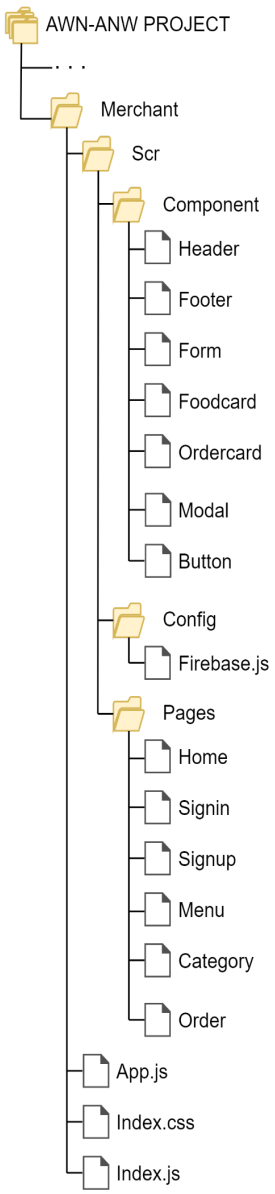
Users use the services of the web-application using their personal devices which access to web browsers via Internet connection. The browser sends HTTP requests to the web server to get the HTML file and displays it to the users.

The web-application has 2 main parts which are front-end and back-end. The front-end (user interface) of the application is implemented with ReactJS library and Tailwind CSS framework. When a user performs any action on the website, it uses the API provided by Firebase to handle that request at the back-end. Our application uses Cloud Firestore provided by Firebase services as our database.

## 6. Implementation View


AWN-ANW PROJECT
Admin
Scr
Component
Header
Footer
Form
InformationCard
CategoryCard
Button
Config
Firebase.js
Pages
Home
Signin
Viewbuyer
Viewcategory
Viewmerchant
App.js
Index.css
Index.js
. . .


AWN-ANW PROJECT
. . .
Buyer
Scr
Component
Header
Footer
Form
Foodcard
Ordersumcard
Orderstatuscard
Modal
Button
Config
Firebase.js
Pages
Signin
Signup
Orderdetail
Orderstatus
Home
Result
Menu
App.js
Index.css
Index.js
. . .


AWN-ANW PROJECT
. . .
Merchant
Scr
Component
Header
Footer
Form
Foodcard
Ordercard
Modal
Button
Config
Firebase.js
Pages
Home
Signin
Signup
Menu
Category
Order
App.js
Index.css
Index.js