Group 05

Justatea E-commerce website
Software Architecture Document

Version 2.0

Revision History

Date	Version	Description	Author
27/7/2024	1.0	First version of Software Architecture Document	TDHPhuoc, TDBinh, HTPhuc
11/8/2024	2.0	Second version of Software Architecture Document	TDHPhuoc, TDBinh, HTPhuc

Table of Contents

Software Architecture Document

1. Introduction

The Software Architecture Document (SAD) for the Justatea e-commerce website provides a comprehensive overview of the system's architecture. This document is intended to outline the high-level design and architecture of the system, capturing the key components, their responsibilities, and their interactions. It serves as a blueprint for the development team and stakeholders, ensuring a common understanding of the system's structure and design principles.

○ 1.1. Purpose

The purpose of this document is to detail the architecture of the Justatea e-commerce website, a platform designed to facilitate online transactions for the food and beverage (F&B) industry. The architecture is intended to support a robust, scalable, and secure online shopping experience for users while enabling vendors to manage their product offerings efficiently. This document aims to:

- Provide a clear description of the system's architecture and its components.
- Define the system's design goals and constraints.
- Outline the use-case model and logical view of the system.
- Describe the deployment strategy and implementation details.

○ 1.2 Scope

The scope of this document includes the architecture of the Justatea e-commerce website, focusing on the following areas:

- Architectural Goals and Constraints: Defines the requirements and constraints impacting the architecture, including performance, security, and scalability considerations.
- Use-Case Model: Provides an overview of the use cases and interactions between users and the system.
- Logical View: Describes the system's components, their responsibilities, and relationships.
- Deployment: Outlines the deployment strategy and environment for the system.
- Implementation View: Details the folder structure and code organization for implementing the system.

○ 1.3 Definitions, Acronyms, and Abbreviations

- SAD: Software Architecture Document
- F&B: Food and Beverage
- UI: User Interface
- API: Application Programming Interface
- HTTP: Hypertext Transfer Protocol
- HTTPS: Hypertext Transfer Protocol Secure
- URL: Uniform Resource Locator
- JSON: JavaScript Object Notation

○ 1.4 References

- Software Engineering, 9th Edition, by Ian Sommerville, Addison Wesley, 2011
- MVC Framework Introduction (geeksforgeeks.org)

○ 1.5 Overview

The Justatea e-commerce website is designed to offer a seamless online shopping experience for customers seeking F&B products. It features user authentication, product browsing, customization options, and a secure checkout process. The architecture is designed to be modular and scalable, allowing for future enhancements and integrations. This document provides a detailed overview of the architectural design to guide development, ensure alignment with project goals, and facilitate effective communication

among stakeholders.

2. Architectural Goals and Constraints

This section outlines the key goals and constraints that guide the architecture of the Justatea e-commerce website. These elements are derived from the non-functional requirements in the Vision document and are crucial for shaping the design and implementation strategy of the system.

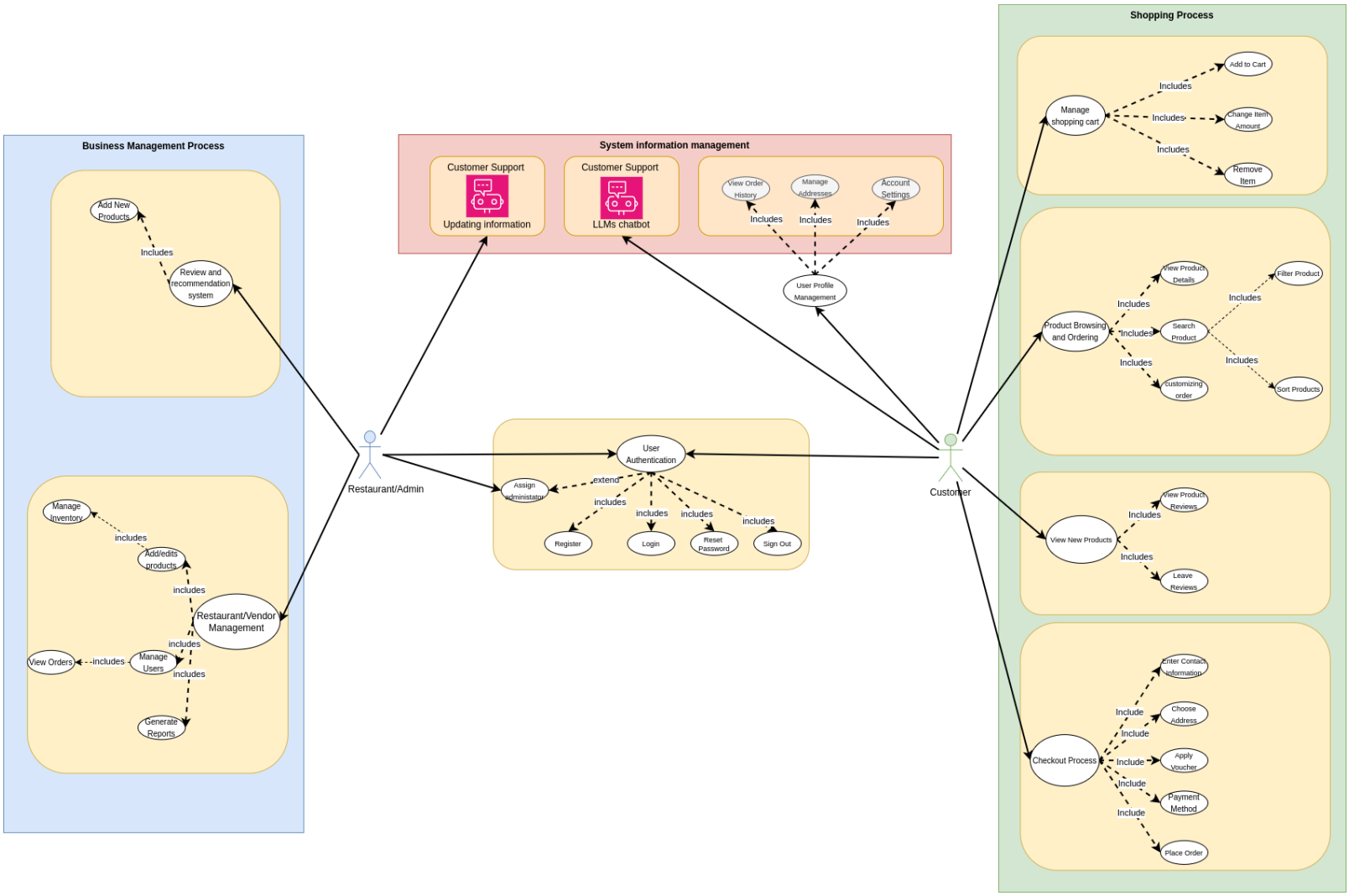
2.1 Architectural Goals

- **Scalability:** Ensure the system can handle a growing number of users and transactions without performance degradation, up to 200 concurrent user.
- **Security:** Implement robust security measures to protect user data and prevent unauthorized access.
- **Performance:** Optimize system performance to ensure fast load times and responsive interactions.
- **Reliability:** Build a reliable system with minimal downtime and robust error-handling mechanisms.
- **Usability:** Design an intuitive and user-friendly interface to enhance the user experience.
- **Modularity:** Develop the system in a modular fashion to facilitate maintenance and future enhancements.
- **Reusability:** Ensure that components can be reused across different parts of the system and in future projects.

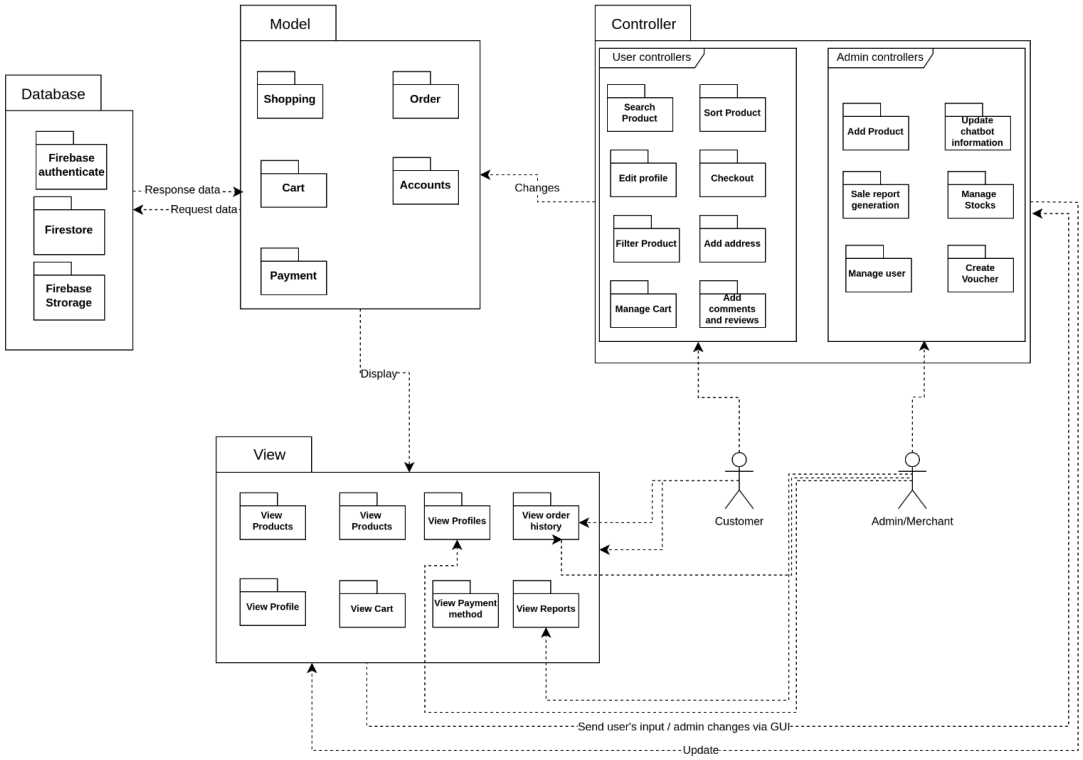
2.2 Architectural Constraints

- **Programming Language:** The system will be developed using JavaScript for the front-end and utilize Firebase service for the backend.
- **Application Environment:** The system will be deployed as a web application accessible via desktop and mobile browsers.
- **Security Requirements:** Implement HTTPS for all communications and encrypt sensitive data.
- **Performance Constraints:** Ensure the system can handle up to 200 concurrent users..
- **Development Tools:** Use Git for version control.
- **Team Structure**: Development will be carried out by a distributed team with defined roles such as front-end developers, back-end developers and product manager.
- **Schedule:** The project must be completed within six months, with key milestones defined for each development phase.
- **Legacy Code:** Integrate with existing systems using APIs without disrupting current operations.

3. Use-Case Model


Business Management Process
Add New Products
Includes
Review and recommendation system
Manage Inventory
includes
Add/edits products
includes
Restaurant/Vendor Management
includes
View Orders
includes
Manage Users
includes
Generate Reports
System information management
Customer Support
Updating information
Customer Support
LLMs chatbot
View Order History
Manage Addresses
Account Settings
Includes
Includes
Includes
User Profile Management
Restaurant/Admin
User Authentication
Assign administator
extend
includes
includes
includes
includes
Register
Login
Reset Password
Sign Out
Customer
Shopping Process
Manage shopping cart
Includes
Add to Cart
Includes
Change Item Amount
Includes
Remove Item
Product Browsing and Ordering
Includes
View Product Details
Includes
Search Product
Includes
Customizing order
Includes
Filter Product
Includes
Sort Products
View New Products
Includes
View Product Reviews
Includes
Leave Reviews
Checkout Process
Include
Enter Contact Information
Include
Choose Address
Include
Apply Voucher
Include
Payment Method
Include
Place Order

4. Logical View



Package Diagram

The system is designed using the Model-View-Controller (MVC) architecture pattern, which separates the application into three interconnected components: Model, View, and Controller. Each component has a specific role that contributes to the overall separation of concerns.

Model: Manage Data

The Model is essential for managing the application's data. It handles data manipulation, storage, and access, ensuring data integrity through interactions with external APIs or databases. The Model is responsible for retrieving and updating data as needed.

View: Present Data to Users

The View is responsible for presenting data to users in a human-readable format. It generates the user interface based on the data provided by the Model. In our system, the View includes the customer's UI, merchant’s UI, and admin’s UI, each tailored to their specific needs.

Controller: Handle User Interactions and Updates

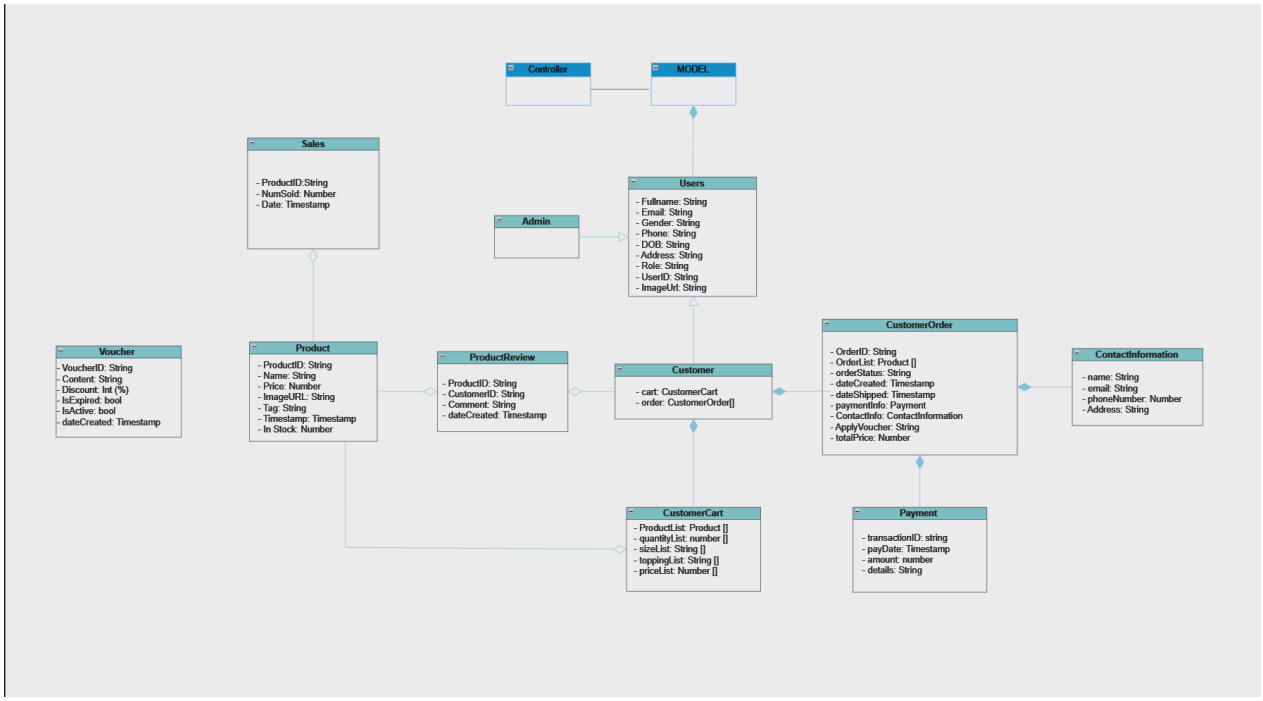
The Controller acts as a bridge between the View and the Model. It processes user input, updates the Model or View as needed,controls the application's flow, selects which View to display, and responds to user interactions.

Interaction Between Components

The interaction between the Model, View, and Controller follows a clear flow:

- User Interaction: When a user interacts with the View (e.g., clicking a button or submitting a form), the associated Controller action is triggered.
- Controller Updates Model: The Controller processes the user input and updates the Model with the relevant data.
- Model Notifies View: The updated data is sent to the View for rendering.
- View Updates UI: The user can see the changes reflected on the user interface.

4.1 Component: Model



Users:

- Responsibilities:
  - + Responsible for setting up and managing user objects within the system's structure.
  - + Defines common attributes and behaviors for user entities to facilitate system interaction and management.
- Attributes: Full name, Email, Gender, Phone, DOB, Address, Role, UserID, ImageURL.
- The two inherited classes of the **Users** class are **Admin** and **Customer**

Customer:

- Responsibilities:
  - + Represents individual users who are customers within the system.
- Attributes: Inherits from the **Users** and includes two properties:

    + **Cart**: Represents the user's shopping cart, allowing for the management of items intended for purchase.
    + **Order**: An array of CustomerOrder objects that keeps track of the customer's past orders.
- Relationship with other classes:
    + The **Customer** class has an aggregation relationship with the **ProductReview** class.
    + The aggregation indicates that while a customer can write multiple reviews, the existence of instance **Customer** is independent of the **ProductReview**.

Admin:

- Responsibilities:
    + Used to define the extended functions that a merchant can request.
- Attributes:
    + Inherits from the **Users**

CustomerCart:

- Responsibilities:
    + Manages the customer's shopping cart, including the items they wish to purchase and their quantities.
- Attributes:
    + **CartList**: An array of **Product** objects representing the items currently in the customer's cart.
    + **SizeList**: An array of values representing sizes corresponding to the index of products in the cart.
    + **ToppingList**: An array of strings representing multiple topping corresponding to the index of products in the cart.
    + **PriceList:** An array of values representing prices corresponding to the index of products in the cart. Initialize by 0
- Relationship with other classes:
    + The **CustomerCart** class has a composition relationship with the **Customer** class.

CustomerOrder:

- Responsibilities:
    + Represents a customer's order within the system.
    + Manages the details of the order, including the items purchased, their quantities, payment information, and order status.
- Attributes:
    + **OrderID**: String - A unique identifier for the order.
    + **OrderList**: Product[] - An array of **Product** objects included in the order.
    + **OrderStatus**: String - The current status of the order ("Pending", "Shipped", "Delivered").
    + **DateCreated**: Timestamp - The date and time when the order was created.
    + **DateShipped**: Timestamp - The date and time when the order was shipped.
    + **PaymentInfo**: **Payment** - Contains payment-related information.
    + **ContactInfo**: **ContactInformation** - Contains contact details for the order, such as shipping address and phone number.
    + **AppliedVoucher**: string - To store which vouchers applied
    + **TotalPrice:** number - To store price including shipping
- Relationship with other classes:
    + The **CustomerOrder** class has a composition relationship with the **Customer** class. This indicates that an order is strongly associated with a specific customer, and the order’s lifecycle is dependent on the customer. This object appears when customers order products.

ContactInformation:

- Responsibilities:
  + Provides a comprehensive record of contact details necessary for communication and delivery.
- Attributes:
  + **Name**: String - The name of the contact person.
  + **Email**: String - The email address for communication.
  + **PhoneNumber**: Number - The phone number for contact.
  + **Address**: String - The shipping or billing address.
- Relationship with other classes:
  + The **ContactInformation** class has a composition relationship with the **CustomerOrder** class. This implies that this object appears when a customer order appears too.

Payment:

- Responsibilities:
  + Manages and records the payment details for an order.
- Attributes:
  + **TransactionID**: String - A unique identifier for the payment transaction.
  + **OrderID:** String - Indicates the order will be purchased
  + **Amount**: Number - The total amount of the payment.
  + **Message**: String - Additional details or description related to the payment.
- Relationship with other classes:
  + The **Payment** class has a composition relationship with the **CustomerOrder** class. This means that each **CustomerOrder** includes a **Payment** object, and the payment is tightly coupled with the order.

Product:

- Responsibilities:
  + Represents individual products available in the system.
  + Manages product details such as identification, pricing, availability, and categorization.
- Attributes:
  + **ProductID**: String - A unique identifier for the product.
  + **Name**: String - The name of the product.
  + **Price**: Number - The price of the product.
  + **ImageURL**: String - A URL pointing to an image of the product.
  + **Tag**: String - A tag or category associated with the product
  + **Timestamp**: Timestamp - The date and time when the product was added or last updated.
  + **InStock**: Number - The quantity of the product currently available in stock.
- Relationship with other classes:
  + Aggregation with **ProductReview**:
    - This means that **ProductReview** instances reference **Product** instances via ProductID. Each review is associated with a specific product.
    - The aggregation indicates that while reviews are linked to products, they do not impact the existence of the **Product** instances. If a review is deleted, the **Product** itself remains unaffected. Products can exist independently of their reviews.
  + Aggregation with **CustomerCart:**
    - This means that **CustomerCart** uses instances of **Product** to manage the products that a customer has added to their cart.
    - Products can exist independently of the cart. If a product is removed from a cart, the product itself remains unaffected.
  + Aggregation with **Sales:**

- The aggregation shows that while **Sales** records track product sales, the existence of **Sales** records does not affect the product itself. Products remain in the system regardless of sales records.

ProductReview:

- Responsibilities:
  + Represents feedback provided by customers for products.
  + Manages details such as ratings, comments, and the time when the review was created.
- Attributes:
  + **ProductID**: String - A unique identifier for the product being reviewed. This links the review to a specific product.
  + **CustomerID**: String - A unique identifier for the customer who wrote the review. This links the review to a specific customer.
  + **Comment**: String - The textual feedback provided by the customer about the product.
  + **DateCreated**: Timestamp - The date and time when the review was submitted.

Sales:

- Responsibilities:
  + Represents the sales record for products in the system.
  + Tracks the quantity of products sold and the date of the sale.
- Attributes:
  + **ProductID**: String - A unique identifier for the product being sold. This links the sales record to a specific product.
  + **NumSold**: Number - The quantity of the product sold in the transaction.
  + **Date**: Timestamp - The date and time when the sale occurred.

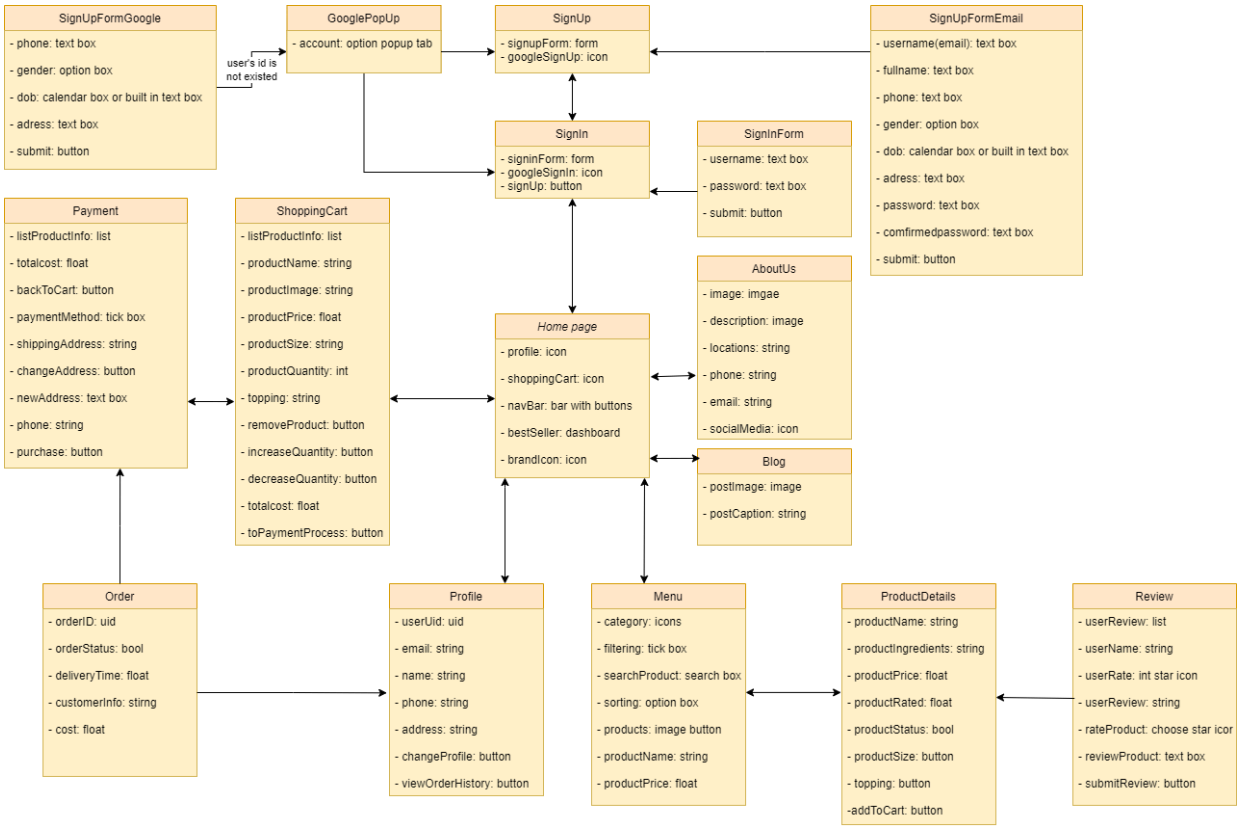
Voucher:

- Responsibilities:
  + Represents discount vouchers or promotional codes available within the system.
  + Manages voucher details, including the content of the voucher, the discount amount, and its status.
- Attributes:
  + **VoucherID**: String - A unique identifier for the voucher.
  + **Content**: String - The text or code that represents the voucher, often used by customers to apply discounts.
  + **Discount**: Int (%) - The percentage discount provided by the voucher.
  + **IsExpired**: Bool - Indicates whether the voucher is expired.
  + **IsActive**: Bool - Indicates whether the voucher is currently active and can be used.
  + **DateCreated**: Timestamp - The date and time when the voucher was created.

4.2 Component: View

- This component is used for UI's application logic. View is for showing customer/admin information and interaction with the application.

4.2.1 User view



Sign Up

This component offers two distinct forms for users to create an account: SignUpFormGoogle and SignUpFormEmail.

- **SignUpFormGoogle** includes fields such as phone number, gender, date of birth, address, and a submit button. It also interacts with GooglePopUp for users to sign up using their Google account.
- **SignUpFormEmail** requires the username (email), full name, phone number, gender, date of birth, address, password, confirmed password, and a submit button.
- Users can choose to sign up via Google or email. After signing up successfully, the user will be redirected to the Home Page..

Sign In

This component allows users to log in using their credentials.

- **SignInForm** consists of a username and password field along with a submit button.
- **googleSignIn** is an icon that interacts with GooglePopUp for users to sign in. If they don’t have an account on our website, they will transfer to SignUpFormGoogle.
- Users can enter their corrected email and password, use Google account to sign in or move to the SignUp component if they don't have an account yet.
- After successfully signing in, the user will be redirected to the Home Page.

Home Page

The central component of the application, providing users with access to various features.

- Users can navigate to different sections like their profile, shopping cart, menu, about us, blog and view new product, sales events on the dashboard.
- Some of the best-sellers and new products will be shown on the small menu below the dashboard for users to notice.
- Users can go to their profile section to look at it or change their user's information.
- Shopping cart can be accessed from the Home page and users can manage it.

Shopping Cart

This component lets users manage their selected products.

- Displays a list of products with details like name, image, price, size, customization and quantity.
- Provides buttons to remove, increase, or decrease product quantity.
- Offer customization in selected products like toppings, size.
- Shows total cost and a button to proceed to the payment process.
- Users can review and modify their cart before proceeding to checkout.

Payment

This component handles the payment process for the user's order.

- Displays a list of products, total cost, payment method, phone number and shipping address.
- Includes options to change the address or use the one that you registered with.
- Provide options for payment methods that users can freely choose: momo and cash.
- Users can finalize their purchase by finishing payment if using momo and double checking shipping details, contact information.
- If the payment process is successful they will be shown an order information. Otherwise, they will be navigated back to their cart.

Order

This component manages order details and status.

- Includes fields for order ID, status, delivery time, customer info, and cost.
- Users can view and track their orders.

Profile

This component allows users to manage their personal information.

- Fields include:
    - User uid
    - Email
    - Name
    - Phone number
    - Address
- There will be an automatically generated profile picture by default and users can change it by uploading their preferred picture.
- Options to change profile details and view order history.

Menu

This component provides a list of available products.

- A list of available products will be displayed with images, names, and prices.
- Features:
    - categories: tea, coffee, juice, smoothie, cake.
    - filtering options for some ingredients that users may be allergic to.
    - search box: search the name of the product.
    - sorting by price, name.
- Users can click on a product to view its full detail and customize before adding to their cart.

Product Details

This component shows detailed information about a specific product.

- Includes information:
    - product name
    - ingredients
    - price
    - rating
    - status: in stock or out of stock
    - size, and topping options.
- Provides an add-to-cart button after the size of the product has been chosen.

Review

This component allows users to leave reviews for products.

- Displays a list of other user reviews with username, rating, review text.
- Users can rate and write reviews for the product and publish their reviews via a submit review button.

About Us

This component provides information about the company.

- Includes images and descriptions, locations, phone numbers, and email addresses.
- Users can learn more about the company by clicking the icon to visit their social media webpages.

Blog

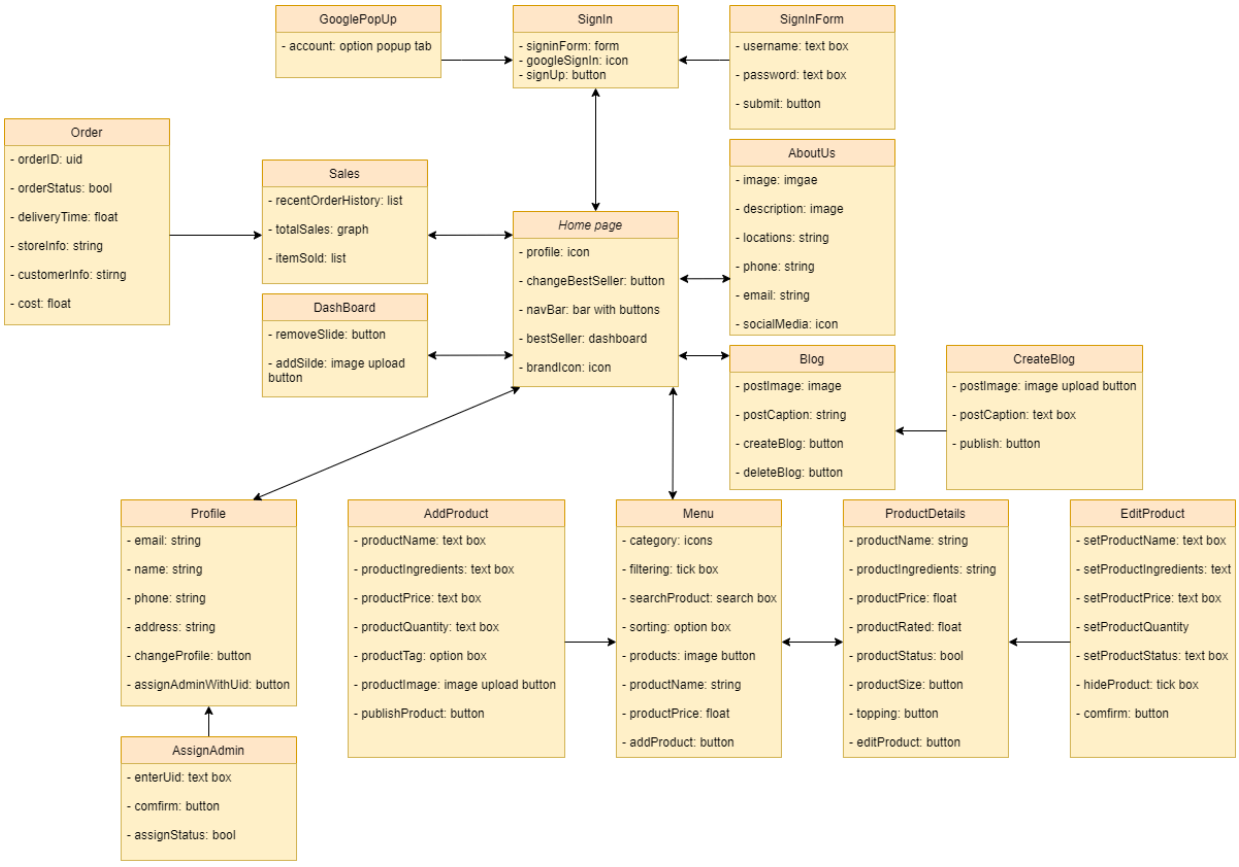
This component displays blog posts.

- Includes post images and captions.
- Users can read and view the latest blog posts.

4.2.2 Amin/Merchant view

Our admin interface is divided into two distinct components: the web admin UI and the Firebase admin UI. Firebase provides a user-friendly and comprehensive interface for managing user accounts, which we will utilize. Other administrative features will be available on our website.

- **Web Admin UI**



SignIn

- This component shows a login form with blank textbox fields such as username and password.
- Providing a submit “Sign in” button for admin to log in to their account if they enter correct information.
- Or they can log in through Google authentication with a button.
- After logging in successfully, the user's account that has role = “Admin” user will be redirected to the Admin Home page otherwise they will be treated as customer.

SignInForm

- This component contains input fields for the username and password.
- It also has a submit button for form submission.

GooglePopUp

- This component allows the user to choose their Google account for authentication.
- It appears when the user opts to sign in with Google.

HomePage

- This component represents the main dashboard view after logging in.
- It includes various icons such as profile, shopping cart, brand icon, and navigation bar with buttons for different actions like viewing the best sellers and performing dashboard operations.

Profile

- This component displays and allows editing of the admin's profile information.
- It includes fields such as email, name, phone, and address.
- Additionally, it has buttons to change the profile and assign admin rights to another user by entering their UID.

AssignAdmin

- This component provides functionality to assign admin rights to a user by entering their UID and confirming the assignment.
- It includes fields for entering the UID and buttons for confirmation and assigning status.

Sales

- This component shows recent order history and sales data with graphs.
- It includes a list of recent orders, a graph of total sales, and a list of items sold.

DashBoard

- This component allows managing the home page dashboard.
- It includes buttons to remove and add slides, with the option to upload images.

AboutUs

- This component provides details about the company include:
    - an image
    - description
    - locations
    - phone number
    - email address
    - icon link to social media

Blog

- This component allows admins to create and manage blog posts.
- It includes fields for changing images, editing captions, creating and deleting blogs.

CreateBlog

- This component is used for creating a new blog post.
- It includes fields for uploading images, entering text, and a publish button.

Menu

- A list of available products will be displayed with images, names, and prices.
- Features:
    - categories: tea, coffee, juice, smoothie, cake.
    - filtering options for some ingredients that users may be allergic to.

  - search box: search the name of the product.
  - sorting by price, name.
- It also has a button to add new products.

AddProduct

- This component is used for adding new products to the catalog.
- It includes fields for:
  - product name
  - ingredients, price
  - quantity
  - tags
  - button to upload images.
- A button is provided to publish the product.

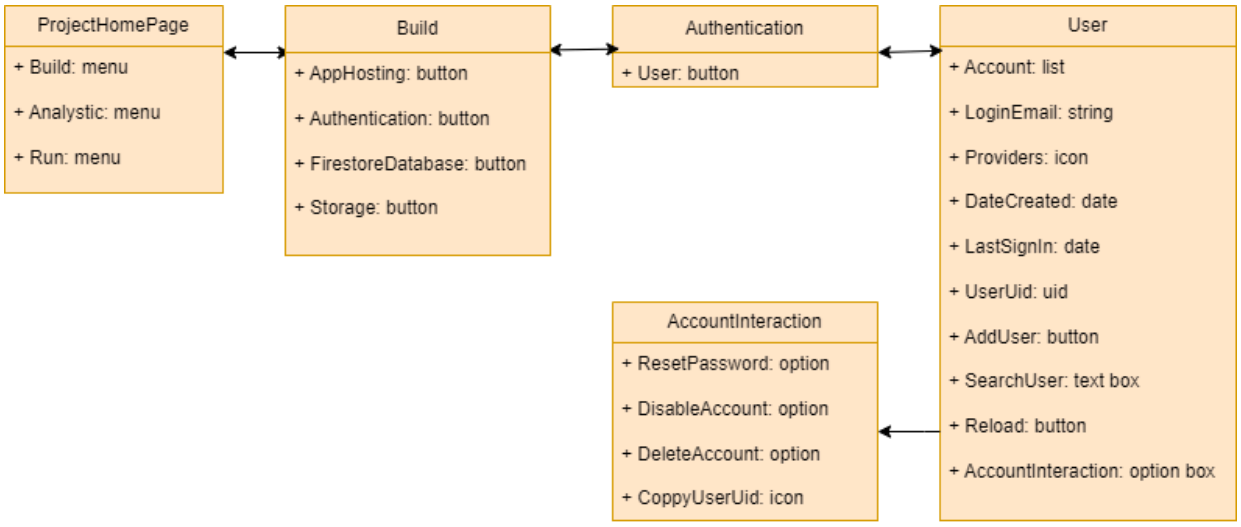
ProductDetails

- This component shows detailed information about a specific product like in User View.
- It includes fields for the product name, ingredients, price, rating, status, size options, and toppings.
- It also has a button to edit the product.

EditProduct

- This component allows editing the details of an existing product.
- It includes fields for setting the product name, ingredients, price, quantity, status, and options to hide the product.
- It also has a confirm button to save changes.

- **Firebase Admin Ui**



ProjectHomePage

- This component shows the main project home page with menu options.
- It includes build, analytics, and run menu options.

Build

- This component provides various build options.
- It includes buttons for app hosting, authentication, Firestore database, and storage.

Authentication

- This component handles user authentication.
- It includes a user button for accessing user-related functionalities.

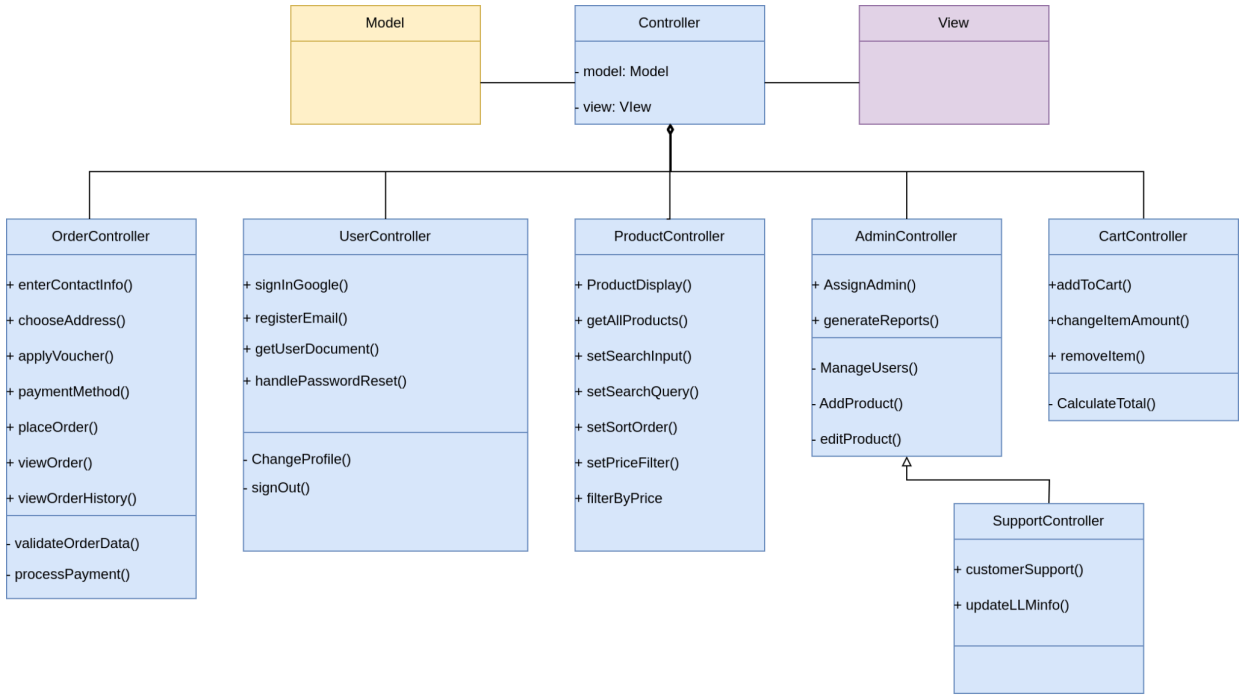
User

- This component manages user accounts.
- It includes a list of accounts that include
    - login email
    - provider icons
    - creation date
    - last login date
    - user UID
    - buttons for adding users and searching users
    - option box for user account interaction
- It also includes a reload button and an account interaction option box.

AccountInteraction

- This component provides four options to interact with a user account:
    - reset password
    - disable account
    - delete account
    - copy user’s id.

4.3 Component: Controller



OrderController

This controller manages the order processing flow.

Public Methods:

- enterContactInfo():Allows the user to input their contact information for the order, collects name, email, and phone number.
- chooseAddress():Enables the user to select or enter a shipping address. offer options like selecting a saved address or entering a new one.
- applyVoucher():Lets the user apply a discount voucher to their order.Should validate the voucher code and apply the discount if valid.
- paymentMethod():Presents available payment options to the user. Could include credit card, MOMO, bank transfer, etc.
- placeOrder():Finalizes the order process.
- viewOrder():Displays details of a specific order.
- viewOrderHistory():Shows the user's past orders include options to filter or search through order history.

Private Methods:

- validateOrderData(): Checks if all necessary order information is complete and valid.Ensures data integrity before processing the order.
- processPayment(): Handles the actual payment transaction. integrates with a payment gateway or service.

UserController

This controller manages user-related operations.

Public Methods:

- signInGoogle(): Implements Google Sign-In functionality. Authenticates the user using their Google account.
- registerEmail():Handles new user registration using email, includes email verification process.
- getUserDocument():Retrieves user data from a database. Used to populate user profile or for authentication.
- handlePasswordReset():Manages the password reset process. Typically involves sending a reset

link via email.

Private Methods:

- ChangeProfile():Updates user profile information include changing name, email, or other details.
- signOut():Logs out the current user. Clears session data and redirects to appropriate page.

AdminController

This controller handles administrative functions.

Public Methods:

- AssignAdmin(): Grants admin privileges to a user, restricted to super-admin access.
- generateReports(): Creates various administrative reports, include sales, user activity, or inventory reports.

Private Methods:

- ManageUsers(): Allows admins to view, edit, or delete user accounts.
- AddProduct(): Adds new products to the inventory.
- editProduct(): Modifies existing product information.

CartController

This controller manages shopping cart operations.

Public Methods:

- addToCart(): Adds a product to the user's shopping cart.
- changeItemAmount(): Modifies the quantity of an item in the cart.
- removeItem(): Removes a specific item from the cart.

Private Methods:

- CalculateTotal():Computes the total cost of items in the cart, include tax and shipping calculations.

ProductController

This controller handles product-related operations.

Public Methods:

- ProductDisplay(): Renders the product details page.
- getAllProducts(): Retrieves all products, possibly for catalog display.
- setSearchInput(): Sets the user's search input for product search.
- setSearchQuery():Processes the search input into a query.
- setSortOrder(): Sets the sorting order for product listings.
- setPriceFilter(): Applies price range filters to product listings.
- filterByPrice(): Filters products based on the set price range.

SupportController

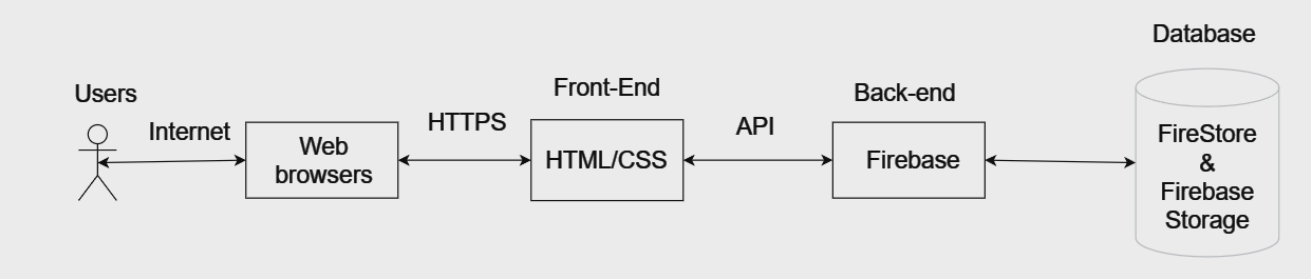
This controller manages customer support functions.

Public Methods:

- customerSupport() :Handles customer support requests, include ticket creation or chat support.
- updateLLMinfo(): Updates information for a language model, used in chatbots.

5. Deployment

This section shows the deployment view of Justatea application with the nodes as below:



Users use web application services using their personal devices accessing a web browser through an 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, HTML and CSS.
- 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