<DiamondHang.Store>
Software Architecture Document

Version <1.4>

Revision History

Date	Version	Description	Author
<24/11/2023>	<1.0>	Describe Introduction, Architecture goals & constraint	Đoàn Gia Phú
<24/11/2023>	<1.1>	Update Use-case Model	Bùi Vũ Thế Minh
<27/11/2023>	<1.2>	Draw Package Diagram (Logical view)	Nguyễn Xuân Quang Minh
<30/11/2023>	<1.3>	Draw Class Diagram component: 4.1, 4.2, 4.3, 4.4	Bùi Vũ Thế Minh
<30/11/2023>	<1.3>	Draw Class Diagram component: 4.5, 4.6, 4.7, 4.8, 4.9	Đỗ Quốc Bảo, Mai Quý Đạt
<1/12/2023>	<1.4>	Relationship among components (4.x)	Nguyễn Xuân Quang Minh
<10/12/2023>	<1.5>	Provide detail script to components	Bùi Vũ Thế Minh
<13/12/2023>	<1.6>	Draw Deployment Diagram & implementation vỉew	Nguyễn Xuân Quang Minh, Đoàn Gia Phú

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides an overview of the entire Software Architecture Document. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the Software Architecture Document. It aims to document and communicate the important architectural choices made for the system.

1.2 Scope

The document covers the architectural overview, key components, interactions among system elements, and deployment strategies essential for understanding and implementing the DiamondHand web application. It also encapsulates the non-functional requirements, architectural goals, and constraints impacting the system's design and development.

1.3 Definitions, Acronyms and Abbreviations

- **API:** Application Programming Interface
- **HTTP:** HyperText Transfer Protocol
- **DB:** Database

1.4 References

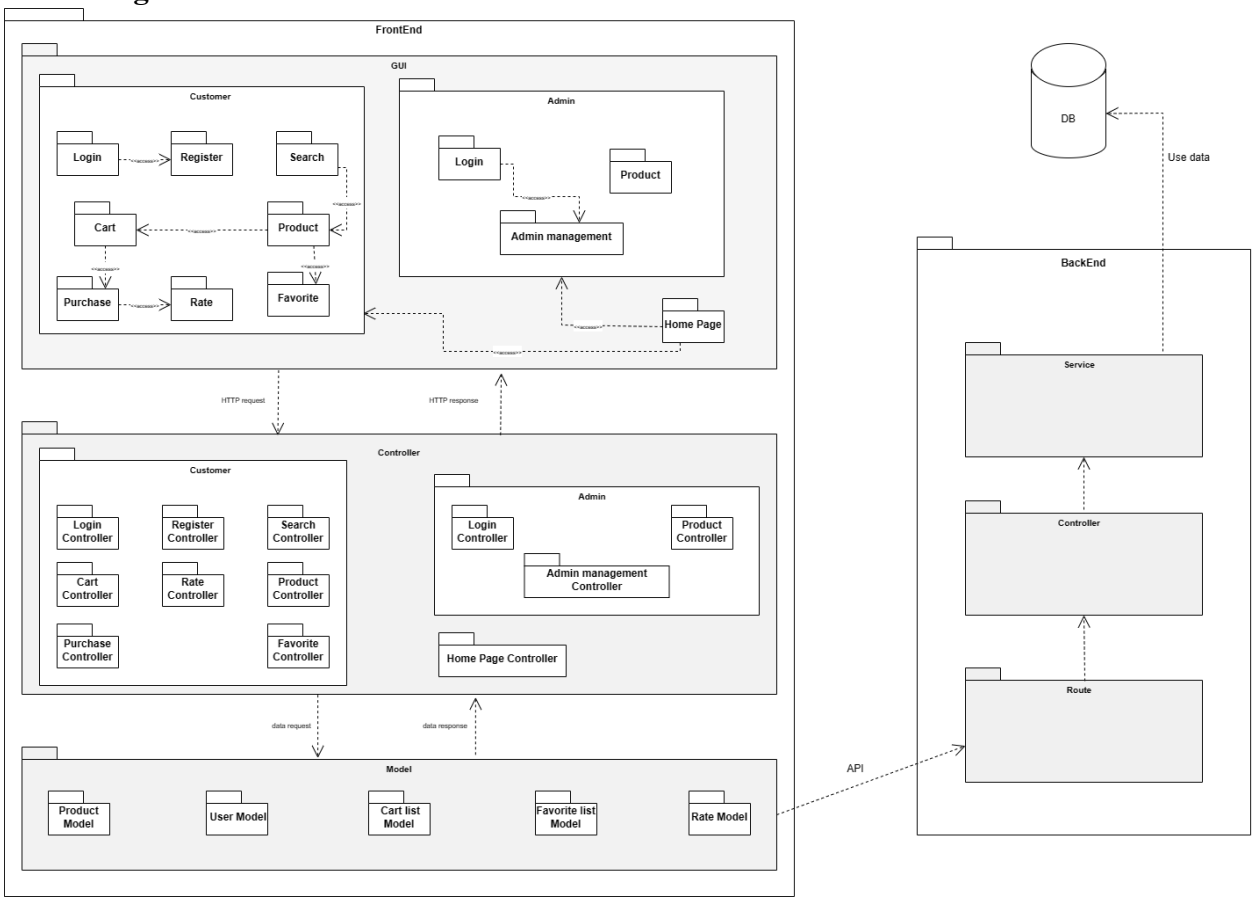
- Vision document
- Use-case model
- Use-case specification

2. Architectural Goals and Constraints

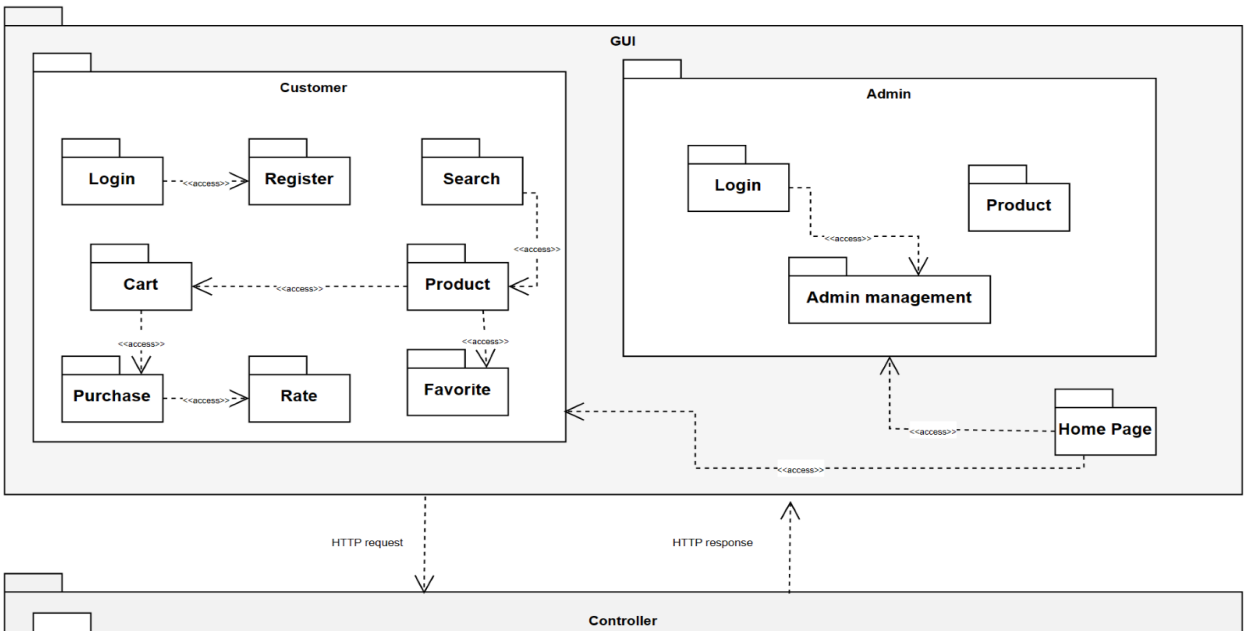
- **Technologies Requirements**: The website using MERN stack:
    - MongoDB — document database
    - Express(.js) — Node.js web framework
    - React(.js) — a client-side JavaScript framework
    - Node(.js) — the premier JavaScript web serve
- **Compatibility Requirements**: The website must be designed to work seamlessly across various browsers, operating systems, and devices, providing a smooth user experience regardless of the platform.
- **Intuitive User Interface**: The website's interface should be visually appealing and user-friendly, ensuring that all features are easily comprehensible and adaptable. Content must adhere to legal standards and be suitable for users of all age groups.
- **Robust Security Measures**: The website must implement strong security protocols to protect customer information, including profiles and purchase details.
- **Simplified Maintenance**: The website should be constructed for effortless maintenance and repairs, facilitating streamlined upkeep processes.
- **Scalable Database Capacity**: The website must feature a robust and expansive database capable of efficiently storing both website and user data.
- **Software Architecture Diagram**: All the diagrams that are use to describe the architecture of the website are store in this [link] in complete overview.

3. Use-Case Model

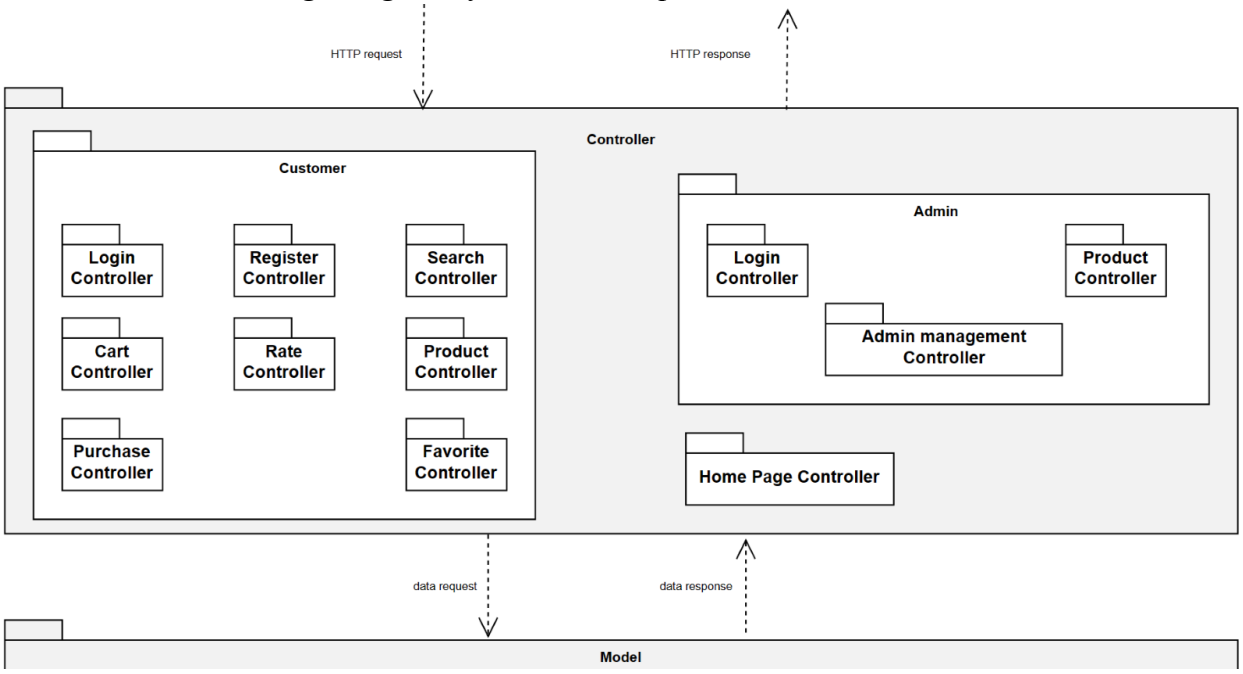
4. Logical View



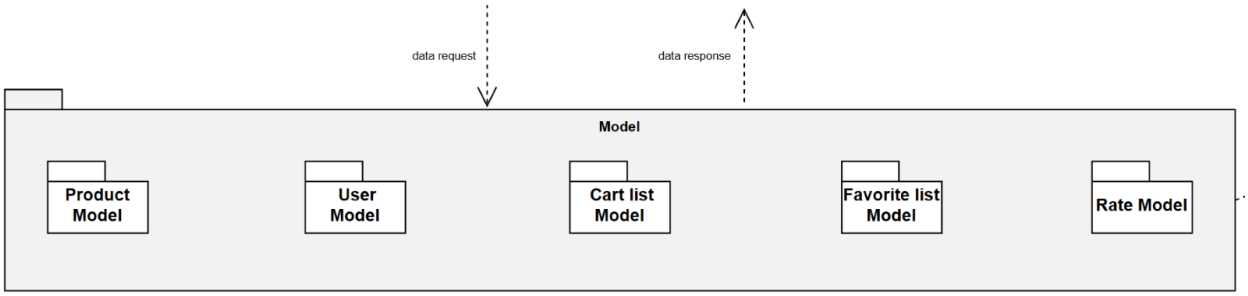
Pic1: Package diagram of the project (MVC model)



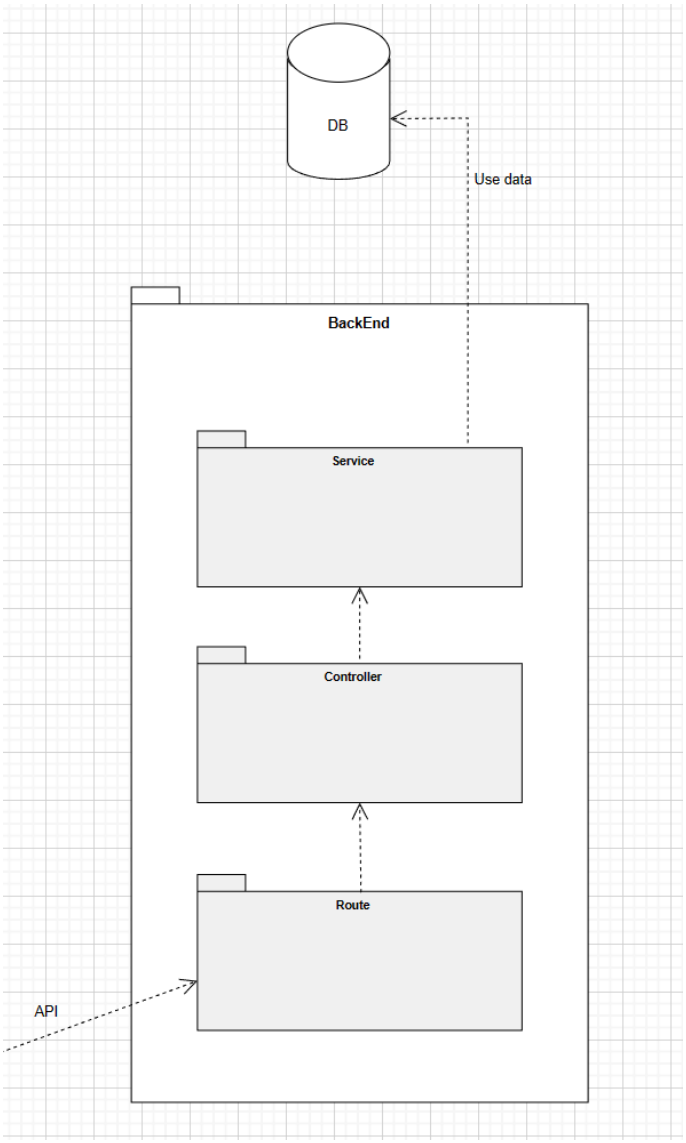
Pic2: Package diagram of GUI, have dependence relation with Controller



Pic3: Package diagram of Controller, have dependences relation with GUI and Model



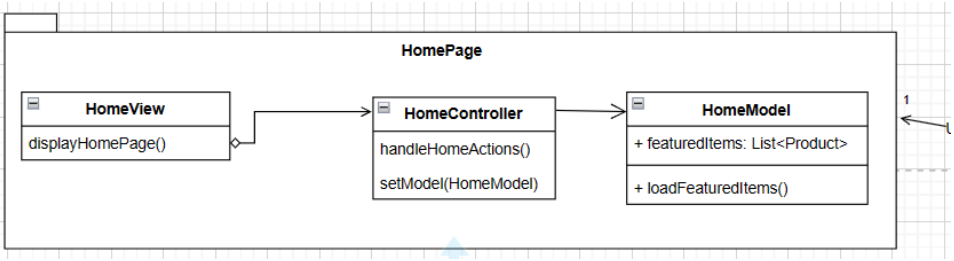
Pic4: Package diagram of Model, have dependence relation with Controller



Pic4: Package diagram of Backend, have dependences relation with Frontend & DataBase

Description for BackEnd component will be updated when we have well-armed knowledge about this.

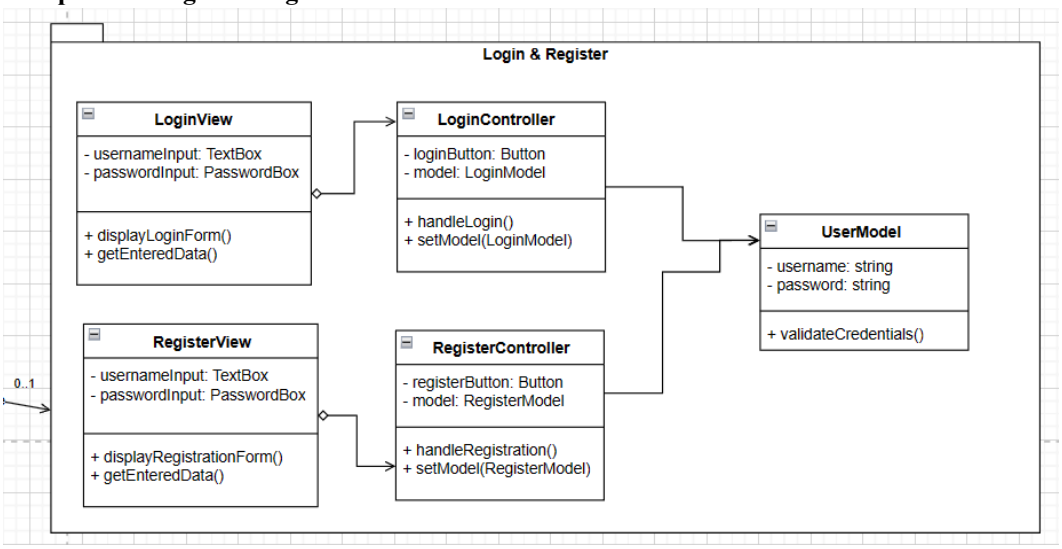
1. **Component: HomePage**



Relationship specification:

- HomeView to HomeController:
  - Type: Association
  - Explanation: Controlling and display controlling actions on user's screen need to be parallel. Therefore, HomeView and HomeController exchanges data with each other to complete this task
- HomeController to HomeView:
  - Type: Aggregation
  - Explanation: HomeView can display various controlling actions on screen at the same time. Therefore, it is associated with HomeCotroller and made up of one or more HomeController objects.
- HomeController to HomeModel:
  - Type: Association
  - Explanation: HomeController retrieves or manipulates data from HomeModel object.

2. **Component: Login & Register**



Relationship specification:

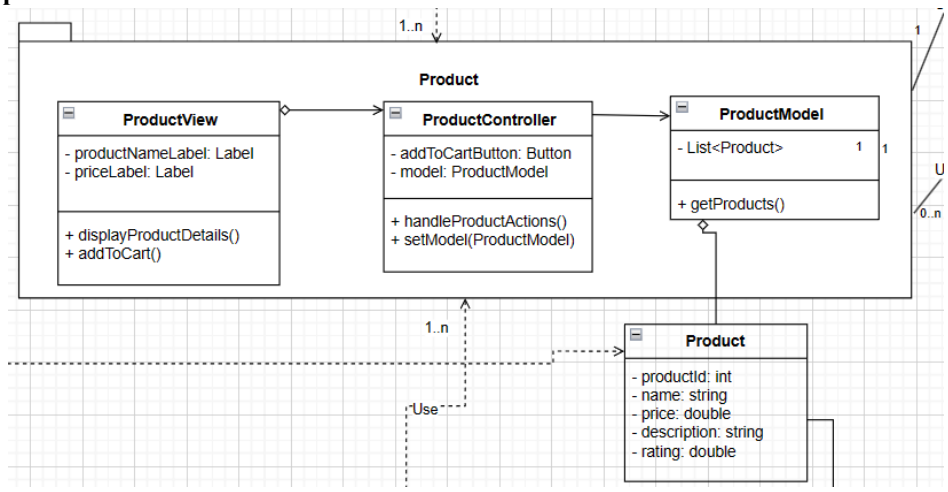
- LoginView to LoginController:
  - Type: Association
  - Explanation: Controlling and display controlling actions on user's screen need to be parallel. Therefore, LoginView and LoginController exchanges data with each other to complete this task
- LoginController to LoginView:
  - Type: Aggregation
  - Explanation: LoginView can display various controlling actions on screen at the same time. Therefore, it is associated with LoginController and made up of one or more HomeController objects.
- RegisterController to RegisterView:
  - Type: Aggregation
  - Explanation: RegisterView can display various controlling actions on screen

at the same time. Therefore, it is associated with RegisterCotroller and made up of one or more RegisterController objects.

- RegisterViewto RegisterController:
    - Type: Association
    - Explanation: Controlling and display controlling actions on user's screen need to be parallel. Therefore, RegisterView and RegisterController exchanges data with each other to complete this task

- RegisterController and LoginController to UserModel:
    - Type: Association
    - Explanation: RegisterController and LoginController retrieve or manipulate data from UserModel object.

3. **Component: Product**

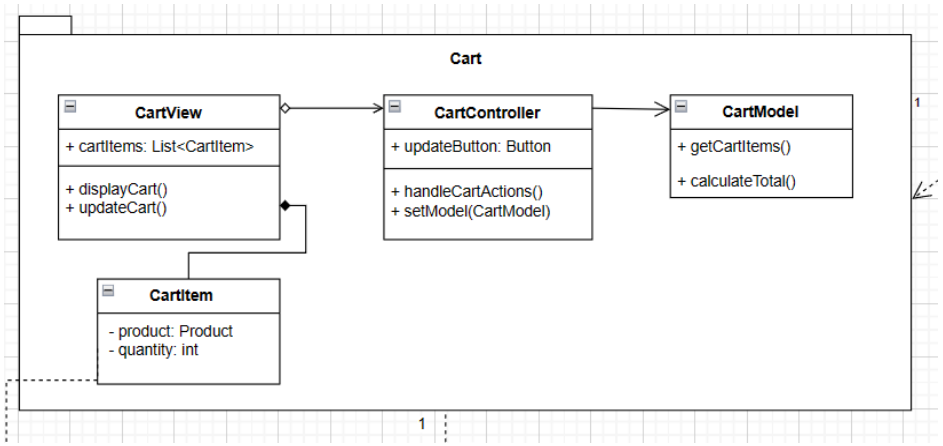


Relationship specification:

- ProductView to ProductController:
    - Type: Association
    - Explanation: Controlling and display controlling actions on user's screen need to be parallel. Therefore, ProductView and ProductController exchanges data with each other to complete this task
- ProductController to ProductView:
    - Type: Aggregation
    - Explanation: ProductView can display various controlling actions on screen at the same time. Therefore, it is associated with ProductController and made up of one or more ProductController objects.
- ProductController to ProductModel:
    - Type: Association
    - Explanation: ProductController retrieves or manipulates data from ProductModel object.
- Product to ProductModel:

- Type: Aggregation
- Explanation: a ProductModel object is made of many Product objects

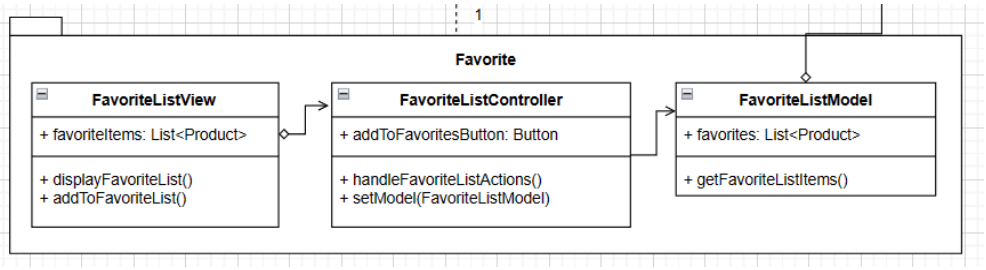
4. **Component: Cart**



Relationship specification:

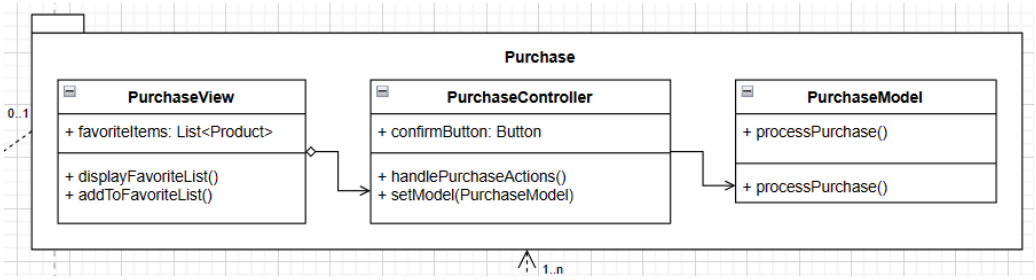
- CartView to CartController:
  - Type: Association
  - Explanation: Controlling and display controlling actions on user's screen need to be parallel. Therefore, CartView and CartController exchanges data with each other to complete this task
- CartController to CartView:
  - Type: Aggregation
  - Explanation: CartView can display various controlling actions on screen at the same time. Therefore, it is associated with CartController and made up of one or more CartController objects.
- CartController to CartModel:
  - Type: Association
  - Explanation: CartController retrieves or manipulates data from CartModel object.
- CartItem to CartView:
  - Type: Composition
  - Explanation: CartView object is made of various cartItem objects but whenever a CartView object is destroyed, CartItem objects cannot stand alone as they are only created to store information for items inside cart.

5. **Component: Favorite**



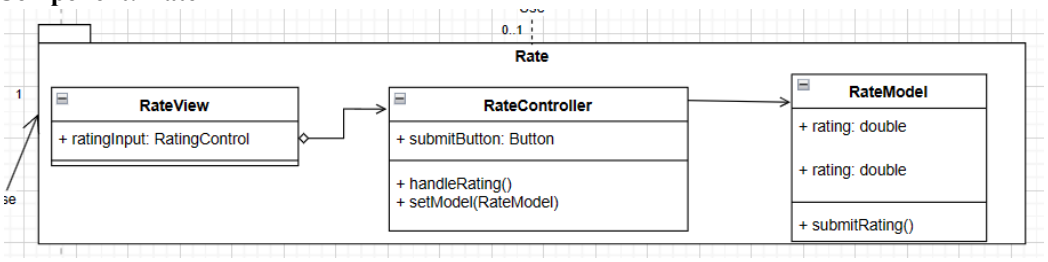
- FavoriteView to FavoriteController:
    - Type: Association
    - Explanation: Controlling and display controlling actions on user's screen need to be parallel. Therefore, FavoriteView and FavoriteController exchanges data with each other to complete this task
- FavoriteController to FavoriteView:
    - Type: Aggregation
    - Explanation: FavoriteView can display various controlling actions on screen at the same time. Therefore, it is associated with FavoriteController and made up of one or more FavoriteController objects.
- FavoriteController to FavoriteModel:
    - Type: Association
    - Explanation: FavoriteController retrieves or manipulates data from FavoriteModel object.

6. **Component: Purchase**



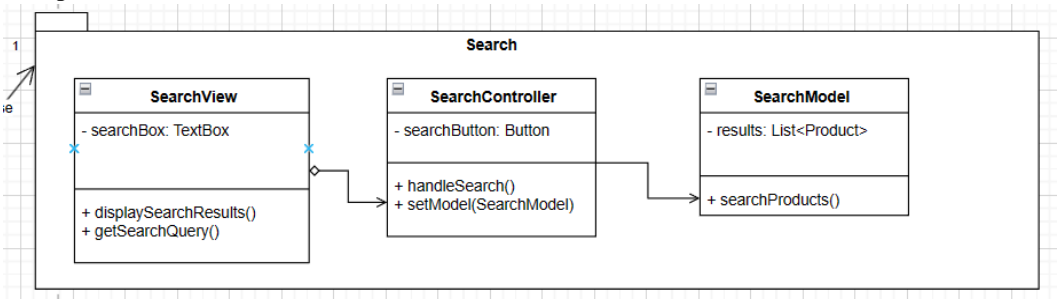
- PurchaseView to PurchaseController:
    - Type: Association
    - Explanation: Controlling and display controlling actions on user's screen need to be parallel. Therefore, PurchaseView and PurchaseController exchanges data with each other to complete this task
- PurchaseController to PurchaseView:
    - Type: Aggregation
    - Explanation: PurchaseView can display various controlling actions on screen at the same time. Therefore, it is associated with PurchaseController and made up of one or more PurchaseController objects.
- PurchaseController to PurchaseModel:
    - Type: Association
    - Explanation: PurchaseController retrieves or manipulates data from PurchaseModel object.

7. **Component: Rate**



- RateView to RateController:
  - Type: Association
  - Explanation: Controlling and display controlling actions on user's screen need to be parallel. Therefore, RateView and RateController exchanges data with each other to complete this task
- RateController to RateView:
  - Type: Aggregation
  - Explanation: RateView can display various controlling actions on screen at the same time. Therefore, it is associated with RateController and made up of one or more RateController objects.
- RateController to RateModel:
  - Type: Association
  - Explanation: RateController retrieves or manipulates data from RateModel object.

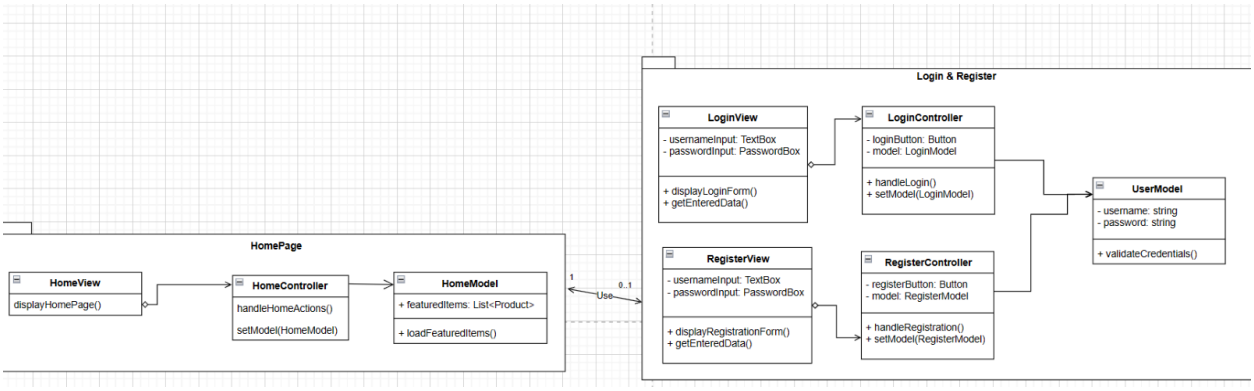
8. **Component: Search**



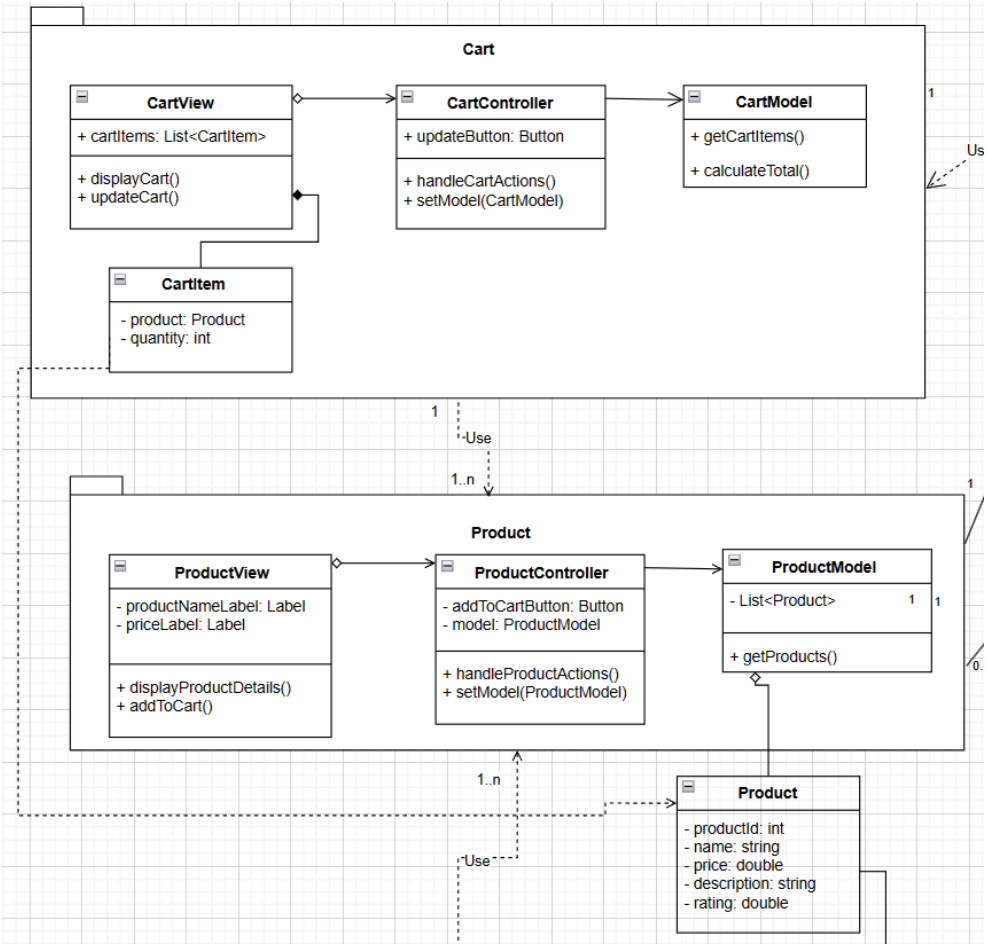
- SearchView to SearchController:
  - Type: Association
  - Explanation: Controlling and display controlling actions on user's screen need to be parallel. Therefore, SearchView and SearchController exchanges data with each other to complete this task
- SearchController to SearchView:
  - Type: Aggregation
  - Explanation: SearchView can display various controlling actions on screen at the same time. Therefore, it is associated with SearchController and made up of one or more SearchController objects.
- SearchController to SearchModel:
  - Type: Association
  - Explanation: SearchController retrieves or manipulates data from

SearchModel object.

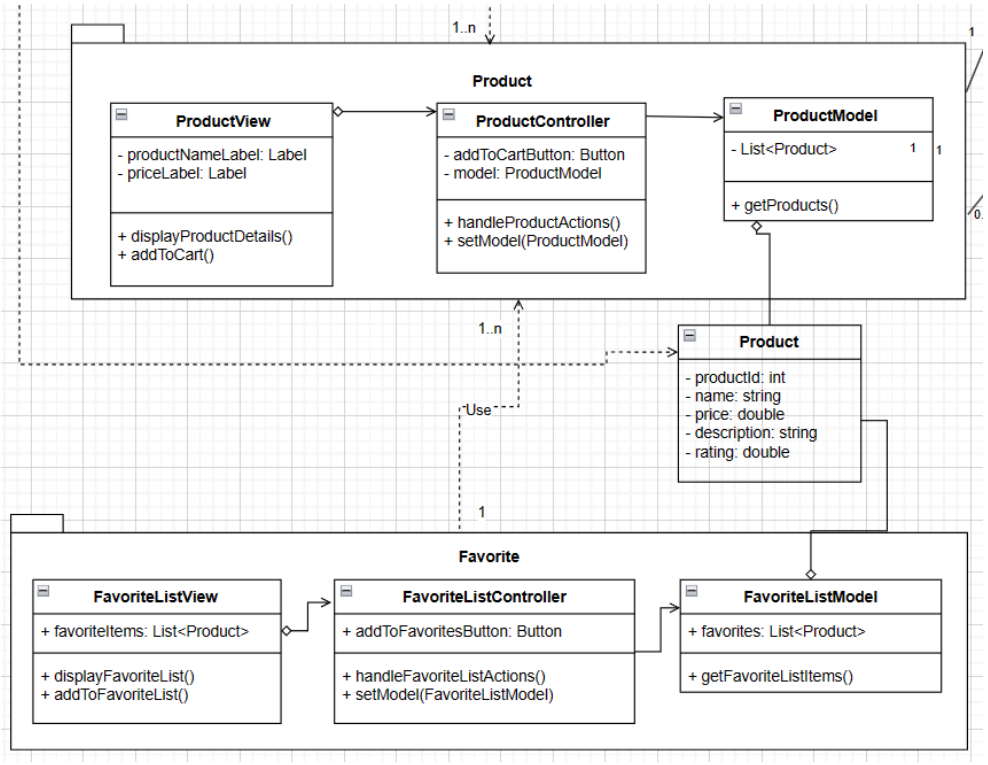
9. **Relationship between components**
   - For a complete overview of those relationships, please go to this [link](drive folder of diagrams for SA).
   - Between **Login & Register** & **HomePage**:
     The handle of login to navigation to HomePage for customer/ Admin management Center for admin are not indicated, it may be executed in handleLogin() or setModel()…



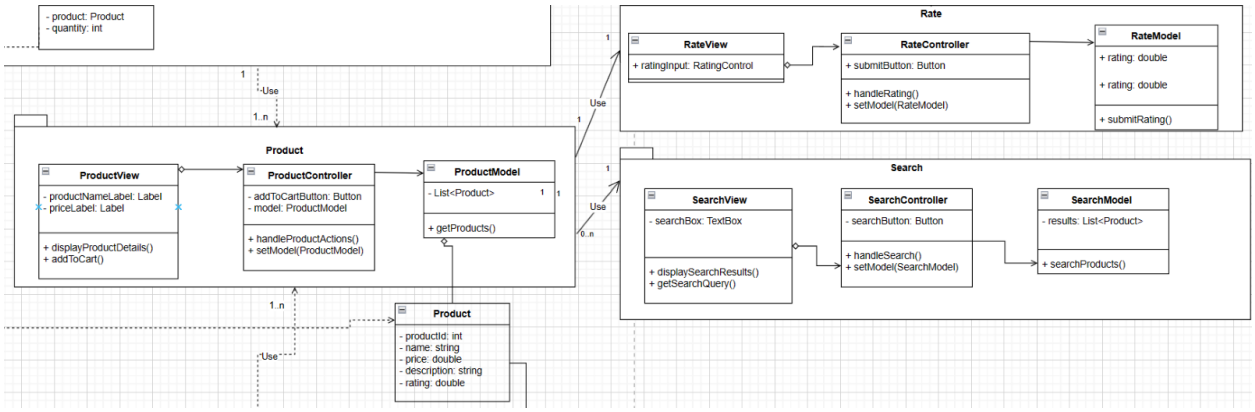
   - Between **Product** & **Cart**



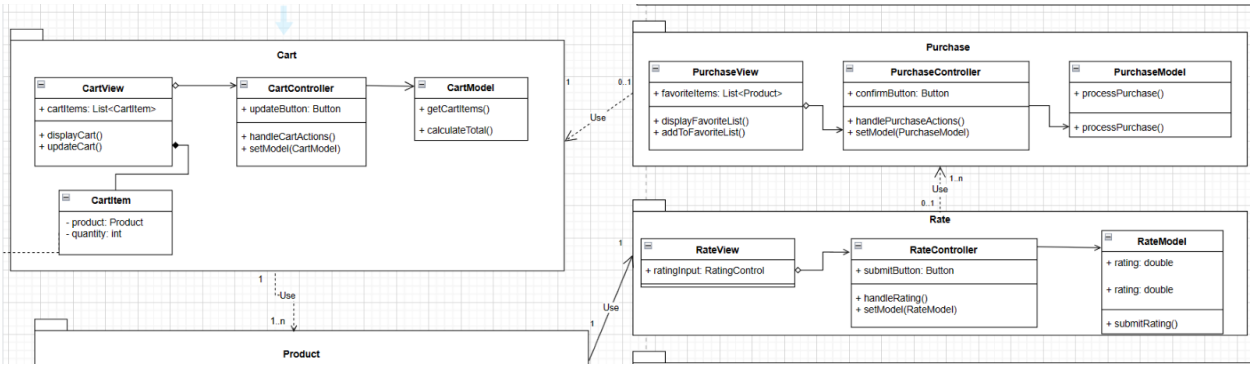
- Between **Product** & **Favorite**



- Between **Product** & **Rate/Search**

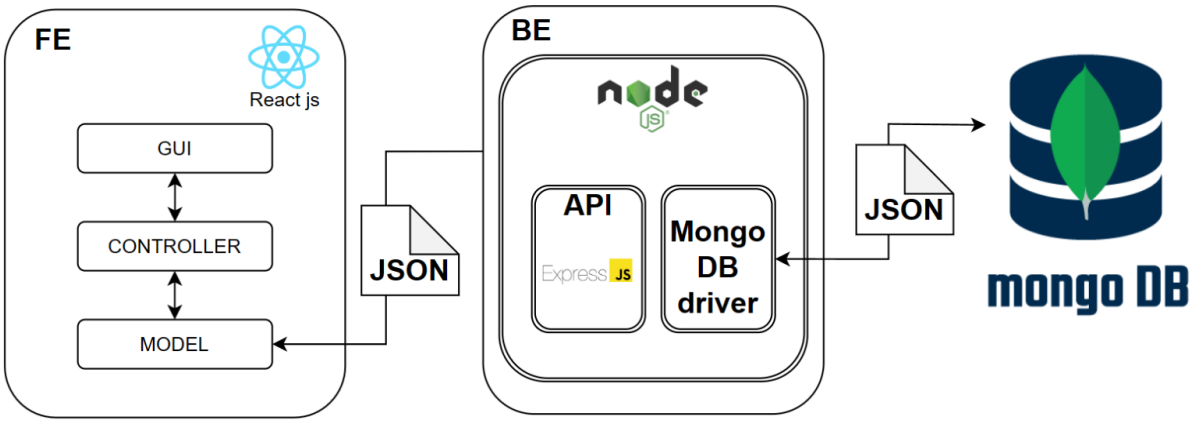


- Between **Cart** & **Purchase** & **Rate**

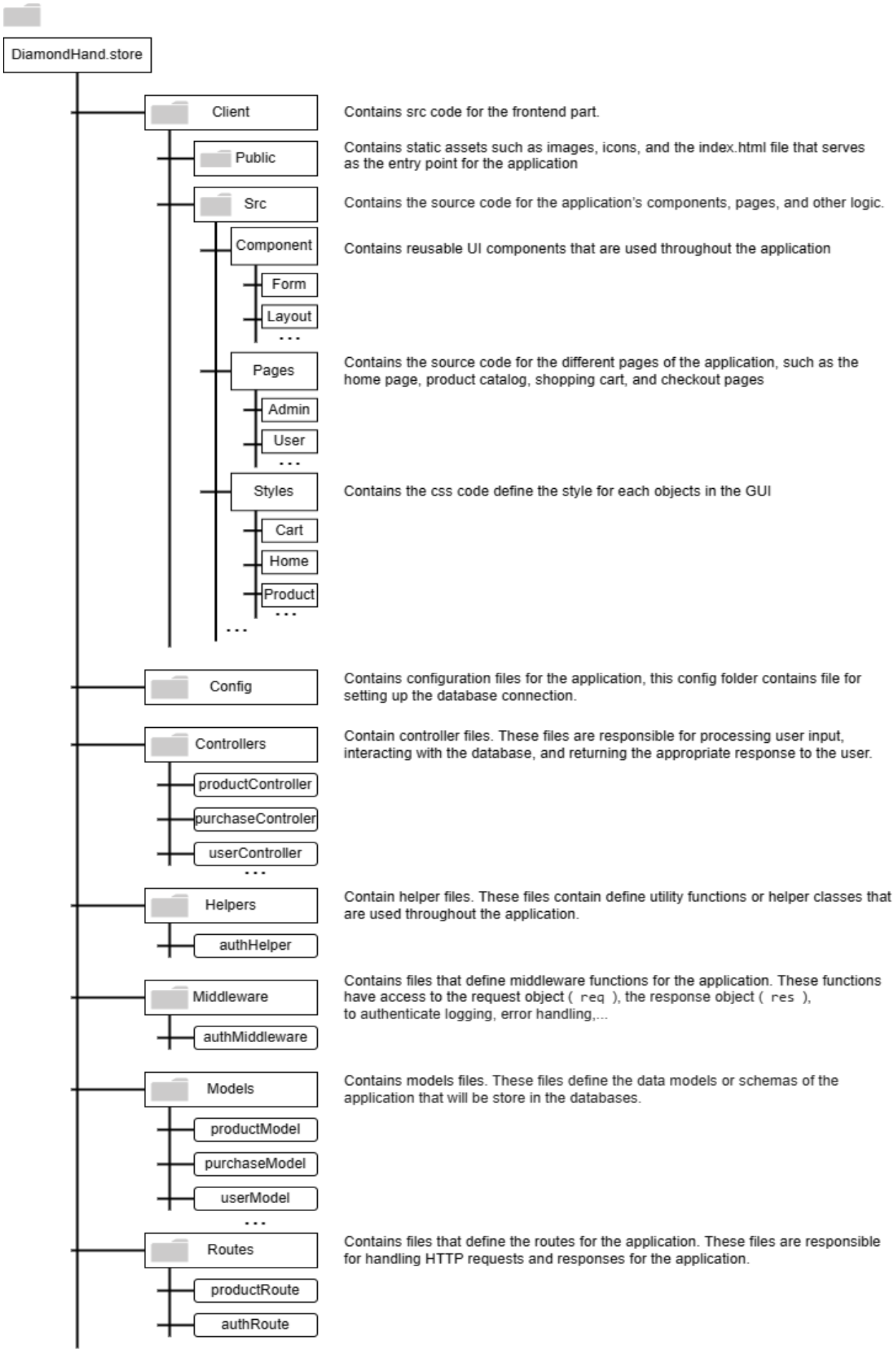


- Other relationships/attributes/methods have not indicated yet due to the lack of information or cannot specificated at this time (ex: the setting that make only login users use favorite list, or the search filter, …)

5. Deployment



6. Implementation View


DiamondHand.store
Client
Contains src code for the frontend part.
Public
Contains static assets such as images, icons, and the index.html file that serves as the entry point for the application
Src
Contains the source code for the application's components, pages, and other logic.
Component
Contains reusable UI components that are used throughout the application
Form
Layout
Pages
Contains the source code for the different pages of the application, such as the home page, product catalog, shopping cart, and checkout pages
Admin
User
Styles
Contains the css code define the style for each objects in the GUI
Cart
Home
Product
Config
Contains configuration files for the application, this config folder contains file for setting up the database connection.
Controllers
Contain controller files. These files are responsible for processing user input, interacting with the database, and returning the appropriate response to the user.
productController
purchaseControler
userController
Helpers
Contain helper files. These files contain define utility functions or helper classes that are used throughout the application.
authHelper
Middleware
Contains files that define middleware functions for the application. These functions have access to the request object (req), the response object (res), to authenticate logging, error handling,...
authMiddleware
Models
Contains models files. These files define the data models or schemas of the application that will be store in the databases.
productModel
purchaseModel
userModel
Routes
Contains files that define the routes for the application. These files are responsible for handling HTTP requests and responses for the application.
productRoute
authRoute