T-2HAND

Software Architecture Document

**(Small Project)**

Version <1.2>

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 21/11/2024 | 1.0 | 1. [Introduction](#_heading=h.gjdgxs) 2. [Architectural Goals and Constraint](#_heading=h.30j0zll)s 3. [Use-Case Model](#_heading=h.1fob9te) 4. [Logical View](#_heading=h.3znysh7) | Trần Đan Huy  Ôn Gia Bảo  Lâm Sỹ Tân |
| 01/12/2024 | 1.1 | 1. Deployment 2. [Implementation](#_heading=h.3dy6vkm) View | Trần Đan Huy |
| 31/12/2024 | 1.2 | Modify Logical view & Deployment view | Ôn Gia Bảo |
|  |  |  |  |

Table of Contents

[**1. Introduction 5**](#_heading=h.gjdgxs)

[1.1 Purpose 5](#_heading=h.gte1rznliwfh)

[1.2 Scope 5](#_heading=h.gek9t8suf3nb)

[1.3 Definitions, Acronyms, and Abbreviations 5](#_heading=h.lgnc4j3rpt43)

[1.4 References 5](#_heading=h.yy7491t4skv2)

[1.5 Overview 5](#_heading=h.tu3xrr9kj1ez)

[**2. Architectural Goals and Constraints 5**](#_heading=h.30j0zll)

[2.1 Technology Stack 5](#_heading=h.3ll2fcx3b2by)

[2.2 General Requirements 5](#_heading=h.5rzg8we72eq0)

[2.3 Dependencies, Constraints, and Security 6](#_heading=h.5gkhvytqsl0j)

[**3. Use-Case Model 6**](#_heading=h.1fob9te)

[**4. Logical View 7**](#_heading=h.3znysh7)

[4.1 Component: View(GUI) 9](#_heading=h.2et92p0)

[4.1.1 Authentication Page 9](#_heading=h.8vzvusuzmi2c)

[4.1.2 Guest Page 10](#_heading=h.bm1vtqyjo10x)

[4.1.3 ChooseRolePage 10](#_heading=h.h4e2kc5cqx36)

[4.1.4 AdminPage 11](#_heading=h.2k1bn31zt641)

[4.1.5 Chat Page 12](#_heading=h.ipk17ycpvd6c)

[4.1.6 SellerPage 13](#_heading=h.vkqkboxirvdo)

[4.1.7 BuyerPage 14](#_heading=h.wacjvjeaim4f)

[4.1.8 ChatWithAdminPage 14](#_heading=h.37ulginldrg)

[4.2 Component: Model 15](#_heading=)

[4.2.1 AccountModel 15](#_heading=h.2zsqzg15tnfd)

[4.2.2 BuyerModel 16](#_heading=h.nbyi49afo67g)

[4.2.3 SellerModel 16](#_heading=h.nln7zuv0zml9)

[+ 4.2.4 ProductModel 17](#_heading=h.r9cra78vs047)

[4.2.5 CartModel 18](#_heading=h.zc9zl4jth8ag)

[4.2.6 OrderModel 18](#_heading=h.l7tnngsgnyey)

[4.2.7 FeedbackModel 19](#_heading=h.7acrxr98fe3s)

[4.3 Component: Controller 19](#_heading=h.7wlzgqynekvi)

[4..3.1 AccountController 20](#_heading=h.26qj862qgmcv)

[4.3.2 BuyerController 20](#_heading=h.1uam2u75xs19)

[4.3.3 SellerController 20](#_heading=h.q0fejscb8p1f)

[4.3.4 ProductController 21](#_heading=h.3dyul9hywd4h)

[4.3.5 CartController 21](#_heading=h.5ody9uaamcm7)

[4.3.6 OrderController 21](#_heading=h.igs3htzhqc07)

[4.3.7 FeedbackController 22](#_heading=h.lvy6h3asl1fo)

[4.4 Component: Routes 22](#_heading=h.7nvwlyy6sn9e)

[4.5 Class Diagram 23](#_heading=h.spqkqwozb6v2)

[**5. Deployment 24**](#_heading=h.tyjcwt)

[5.1. Client (Frontend): 24](#_heading=h.5cew9c3t31xt)

[5.2. Server (Backend): 25](#_heading=h.biwo6cmrgsmt)

[5.3. Database: 26](#_heading=h.ot27ordkw7g5)

[5.3.1. Technology Used: 26](#_heading=h.7omvfa59fi53)

[5.3.2. Schemas: 26](#_heading=h.l0vkh3ufo4f1)

[**6. Implementation View 27**](#_heading=h.3dy6vkm)

[6.1 FRONT-END 28](#_heading=h.ppjr35iu1va3)

[6.2 BACK-END 28](#_heading=h.wt7d2r9fola6)

Software Architecture Document

# Introduction

## 1.1 Purpose

The purpose of this Software Architecture Document is to deliver a detailed architectural overview of the T-2Hands website application. It outlines various architectural perspectives to illustrate different system aspects, including the Architectural Goals and Constraints, the Use-Case Model, the Logical View, the Deployment, and the Implementation View. This document aims to capture and communicate the key architectural decisions made for the system.

## 1.2 Scope

This Software Architecture Document applies to the T-2Hands website application, which will be developed by Group 08 - BabyX. It describes the high-level architecture of the entire system, including major components, their relationships, and the principles and guidelines that shape their design and evolution.

## 1.3 Definitions, Acronyms, and Abbreviations

| Item | Object | Description |
| --- | --- | --- |
| 1 | T-2Hands | T-2Hands is a website application where users can purchase or sell used items, which is created as our project |

## 1.4 References

Software Architecture Document (Version 2001.02). California State University, Northridge, from [Example: Software Architecture Document](https://www.ecs.csun.edu/~rlingard/COMP684/Example2SoftArch.htm)

Slides provided by Mr. Phạm Hoàng Hải, Mr. Trần Duy Hoàng, and Mr. Ngô Ngọc Đăng Khoa in the Course of Introduction to Software Engineering.

## 1.5 Overview

The Software Architecture Document contains the following information:

* Architectural Goals and Constraints: Lists the key requirements and constraints that have a significant impact on the architecture.
* Use-Case Model: Describes the architecturally significant use cases and scenarios.
* Logical View: Describes the architecturally significant parts of the design model.
* Deployment: Describes the physical deployment of the system.
* Implementation View: Describes the overall structure of the implementation model.

# Architectural Goals and Constraints

## 2.1 Technology Stack

* Programming language:
  + Front-end: HTML, CSS, JavaScript, ReactJS
  + Back-end: ExpressJS
* Programming environment: Visual Studio Code
* Application environment: Website

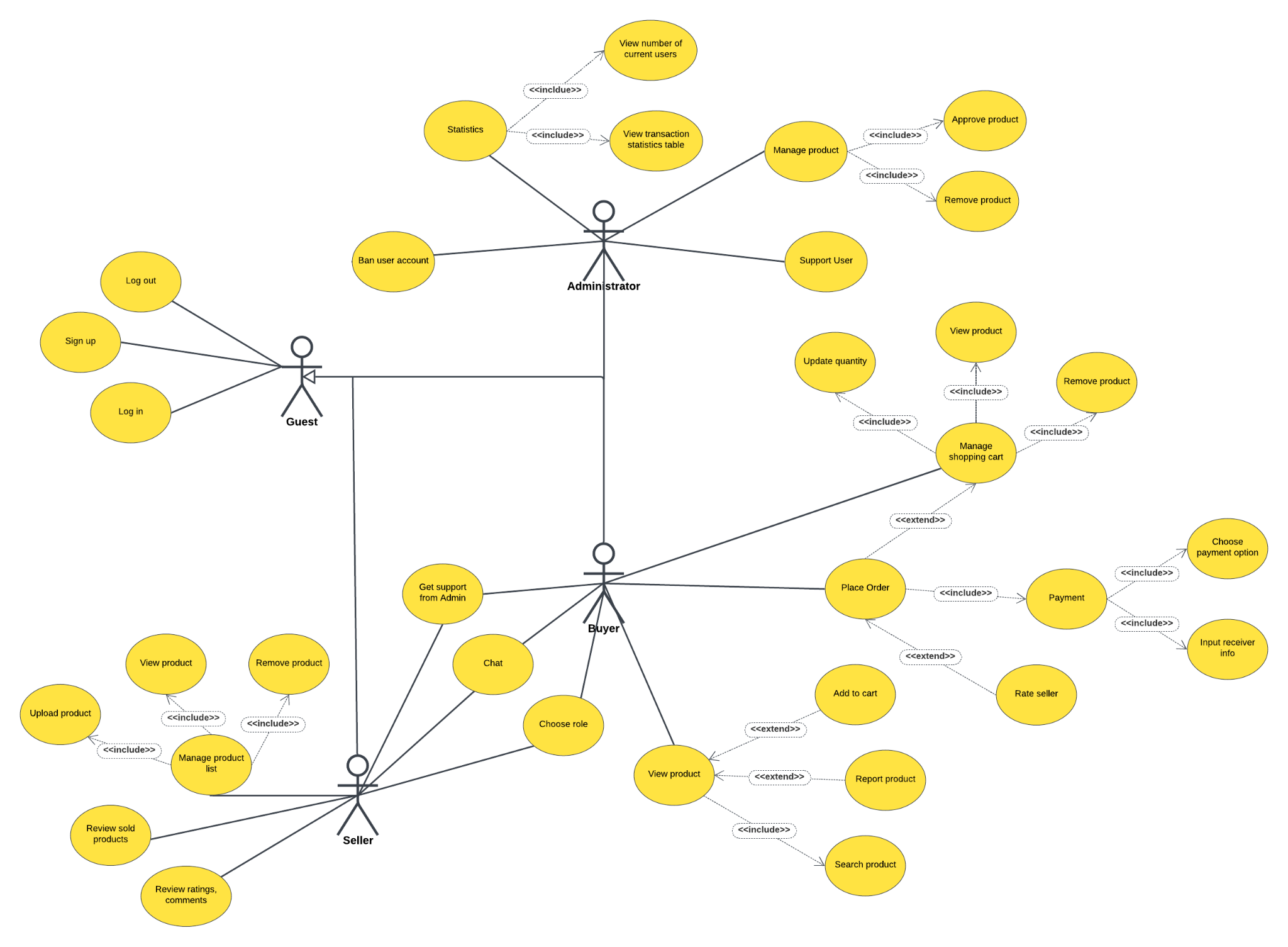
## 2.2 General Requirements

* Portability Requirements
  + Browser: The website must function correctly across major browsers such as Chrome, Firefox, Safari and Edge.
  + Device: The website should work seamlessly across devices, including desktop, tablet, and mobile, with responsive design to adapt to different screen sizes.
* Performance requirements:
  + Record tasks performed by the client: User actions such as accessing, searching, interacting, etc. will be recorded and stored to serve intelligent search or suggest useful information for users.
  + Page Load Speed: The website should load within 2 seconds to prevent user drop-off.
  + Response Time: The response time for actions like clicking "Add to Cart" or "Purchase" should be under 0.5 seconds, clicking “Search” should be under 1 second.
  + Throughput: The system should handle up to 200 concurrent users without performance degradation, the following visits will go into a queue to wait for their turn.
* Availability Requirements
  + The web should have 99.9% uptime annually, with planned maintenance downtime restricted to specific off-peak hours to minimize disruptions.
* Maintainability Requirements:
  + Each quarterly system upgrade should not take more than 60 minutes.

## 2.3 Dependencies, Constraints, and Security

* Constraints:
  + Username: Must be at least 8 characters long.
  + Password: Must be at least 8 characters long and include at least 1 letter, 1 number, and 1 special character.
  + Account Identification: Accounts will be identified by email (Gmail).
  + Guest Access: If you don't register for an account, you can still view the homepage as a guest.
* Dependencies:
  + The default language of our platform will be English, whether the event organized or joined is in English or not.
* Security:
  + Account verification: After 30 minutes without acting on the website, users will be sent a request to sign in again to ensure the security for each account.

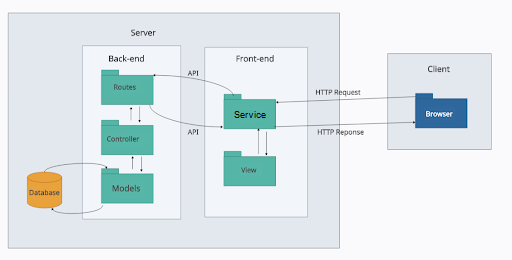
# Use-Case Model



The Use-Case Model is presented in the Use-Case Model and Use-Case Specification provided by Group 08 - BabyX. The image above is the complete Use-Case Model which is required in this Software Architecture Document.

# Logical View

**Description:** The architecture of our website consists of multiple components, categorized into the client, server, and database. The logical view describes how these components interact to create a dynamic and interactive website application.



a. Client

The browser sends HTTP requests to retrieve and modify data with the server. It sends requests to the server for resources like web pages and images. Upon receiving responses, it interprets and renders the content, creating the visual experience the user sees. This dynamic interaction between browser and server allows for real-time updates and user input, enhancing the overall web experience.

b. Server

* Front-end:

+ Vite, a modern frontend build tool, powers the T-2Hands web project as a Multi-Page Application (MPA). It streamlines development by handling API requests and page transitions efficiently. When users interact with the system, Vite sends requests to backend routes for data processing or authentication. As users navigate between pages like Home and Search, Vite dynamically renders the necessary content, providing a smooth user experience without full page reloads..

+ View: Implemented using React, the server-side view can be used for server-side rendering or to provide initial HTML content before the client-side React takes over.

* Back-end:

+ Routes: ExpressJS organizes the routing process, mapping specific URL paths to designated controller functions. This enables the application to handle diverse client requests effectively.

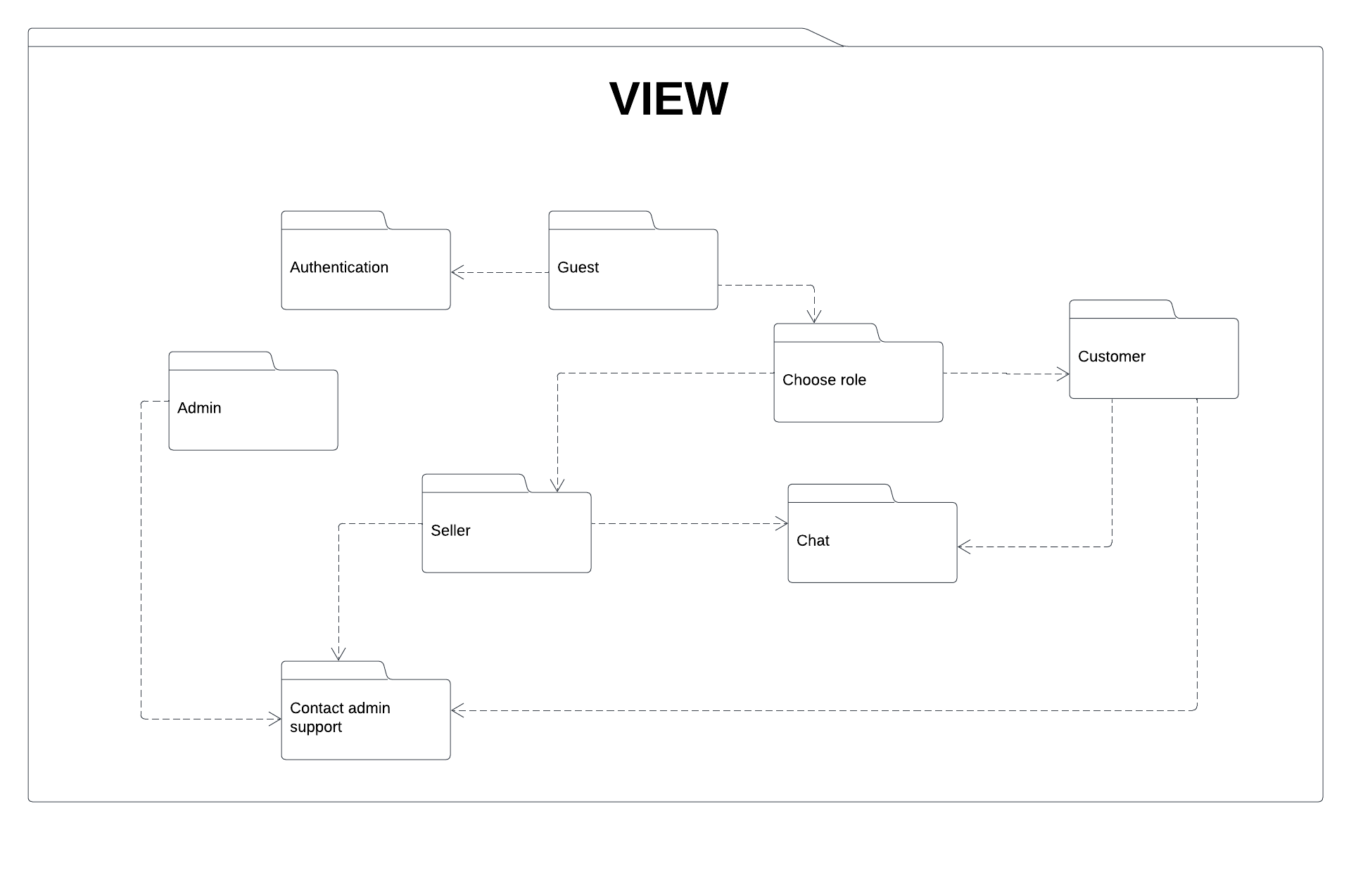
+ Controller: Controllers act as intermediaries, receiving requests from routes, processing data, and interacting with the database model to fetch or update information. Once the data is processed, the controller sends appropriate responses back to the client.

+ Models: Database models define the structure and schema for data storage. Using **Mongoose**, they facilitate interactions with **MongoDB Atlas**, enabling CRUD (Create, Read, Update, Delete) operations. These models ensure efficient and organized data management while enforcing schema validation and relationships.

* Database MongoDB Atlas: MongoDB, a NoSQL database, uses a flexible JSON-like format to store application data. Models define schemas for various data types, while Mongoose ensures smooth communication between the server and the database.

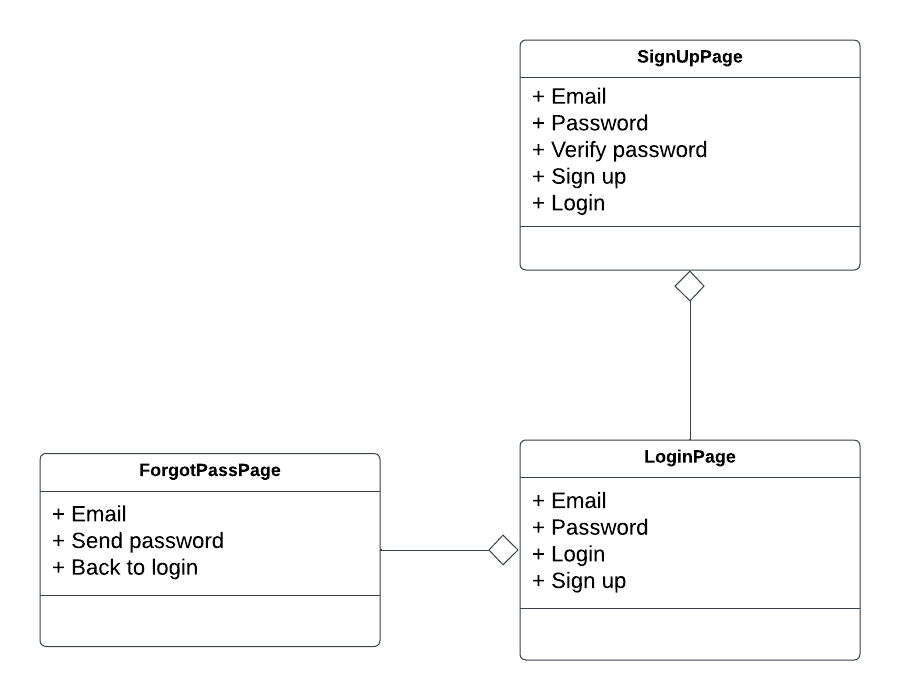
## Component: View(GUI)

**Package Diagram**



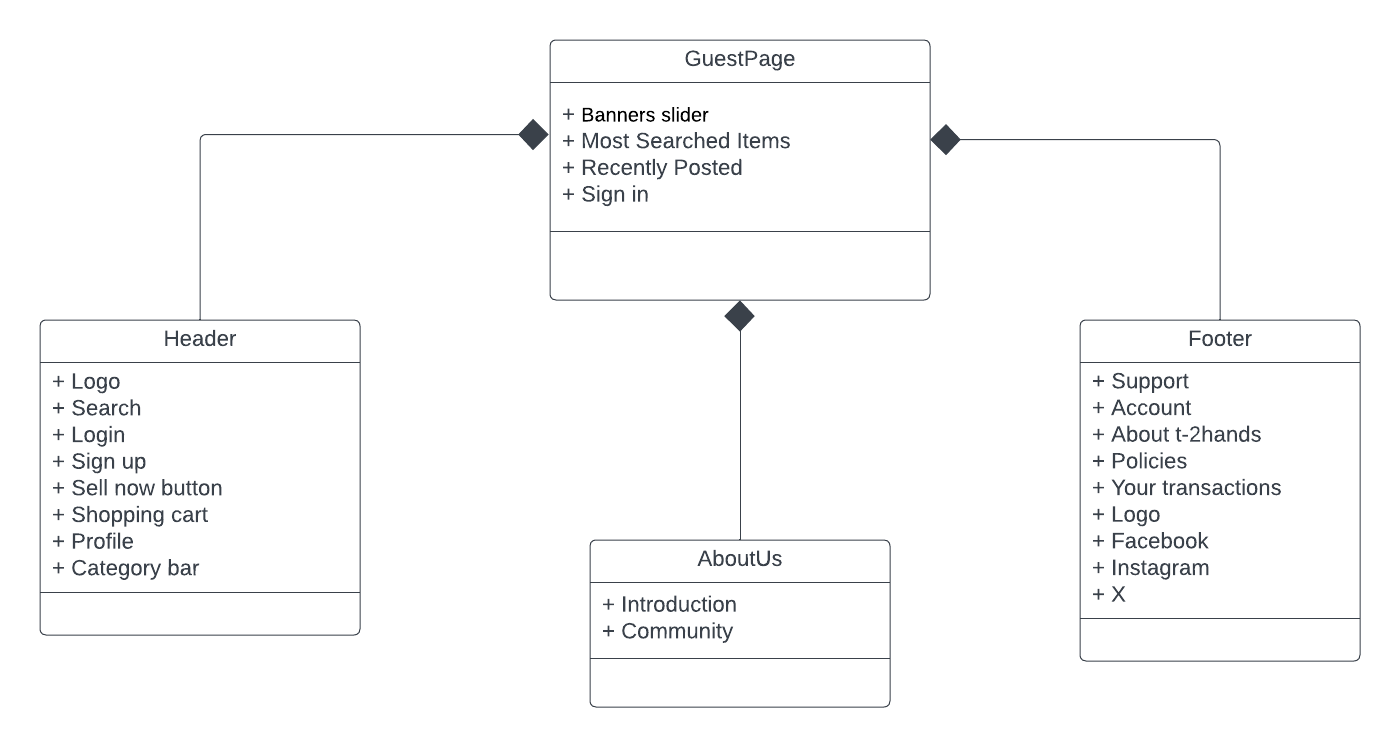
**Description**: The view package contains all the components responsible for rendering the user interface and presenting data to the user. It is built using React and includes various pages to support different functionalities of the T2Hands Web application.

### 4.1.1 Authentication Page



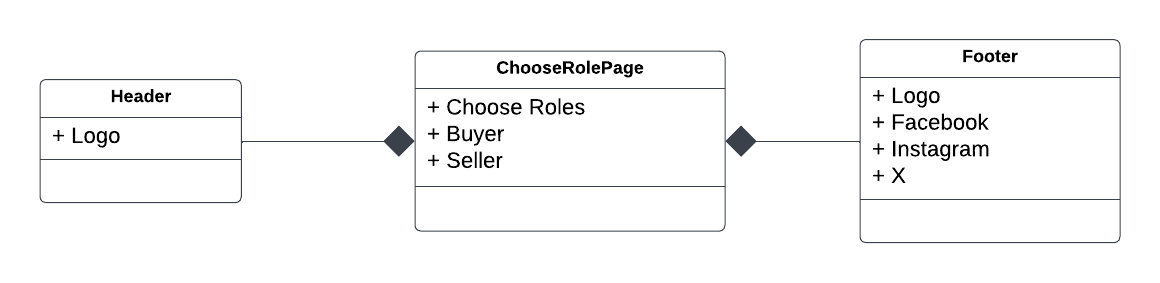
**Description:** This package handles user authentication, including login, sign-up, and password recovery. It ensures that users are properly authenticated and authorized to access parts of the platform.

### 4.1.2 Guest Page



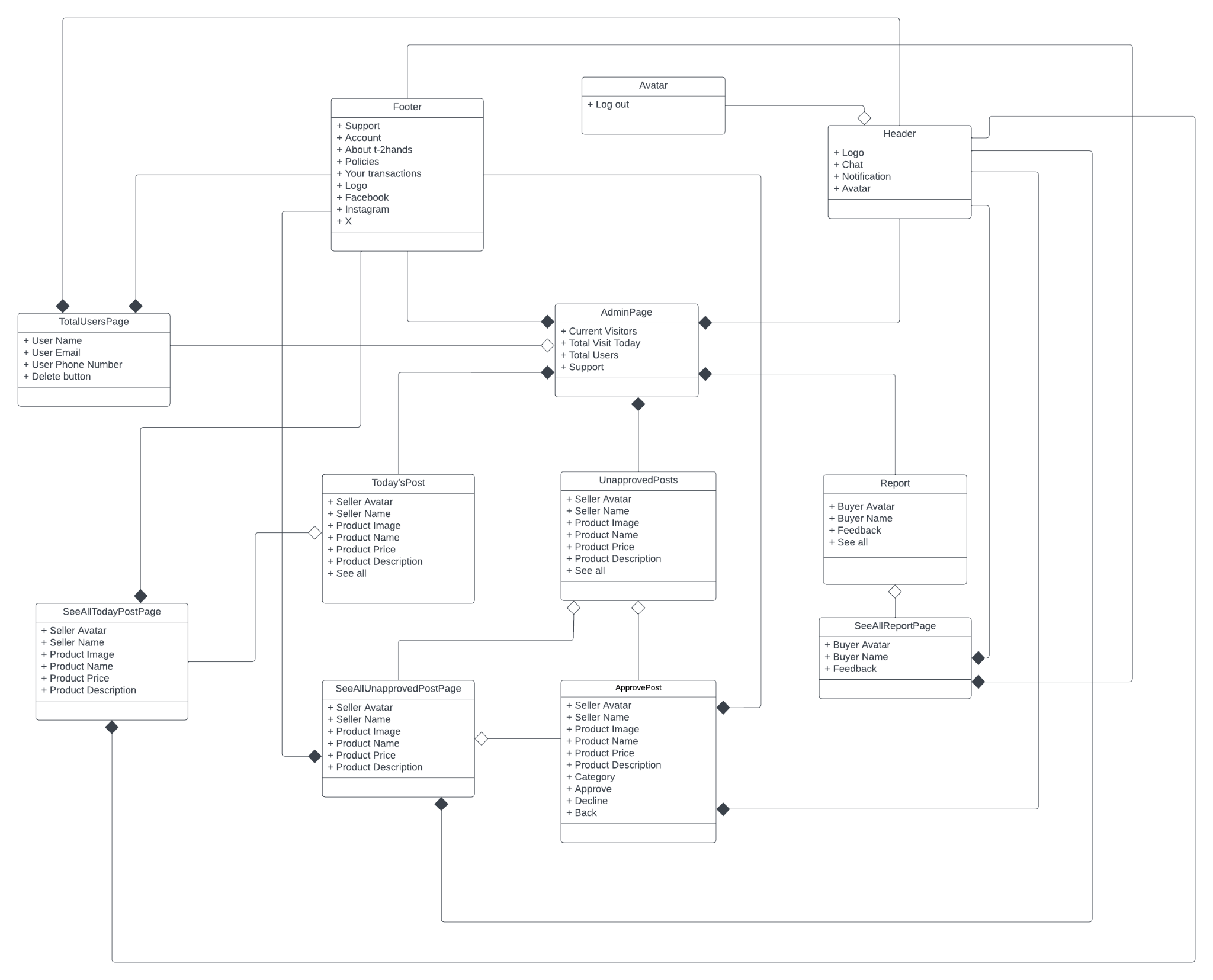
**Description**: This page is designed for users who have not signed into the system. It includes an overview of the platform, and options to login or sign up.

### 4.1.3 ChooseRolePage



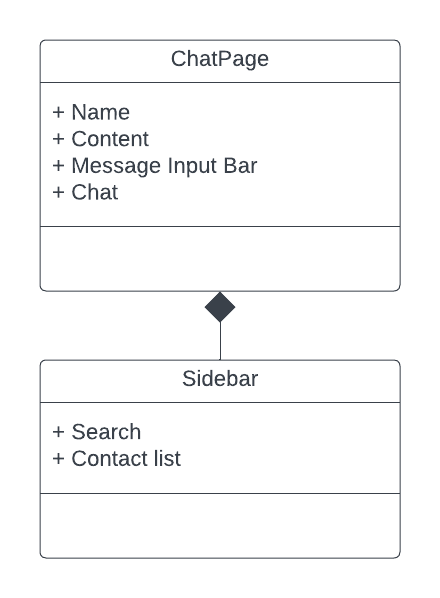
**Description:** This page is designed for users to choose whether to be a buyer or a seller in T-2Hands.

### 4.1.4 AdminPage



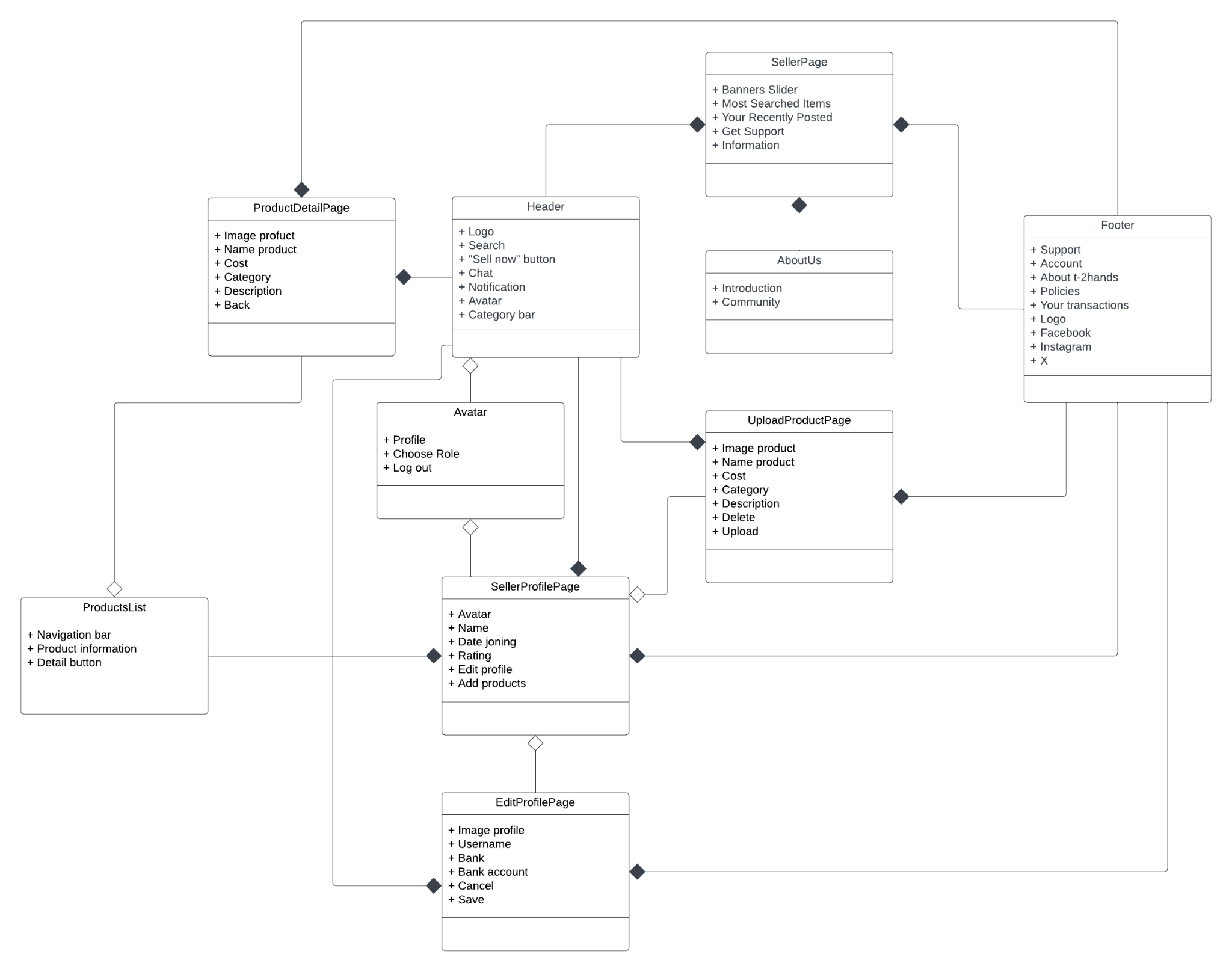
**Description:** This page is built for administrators of the platform. It provides options to manage users’ accounts, posts and products.

### 4.1.5 Chat Page



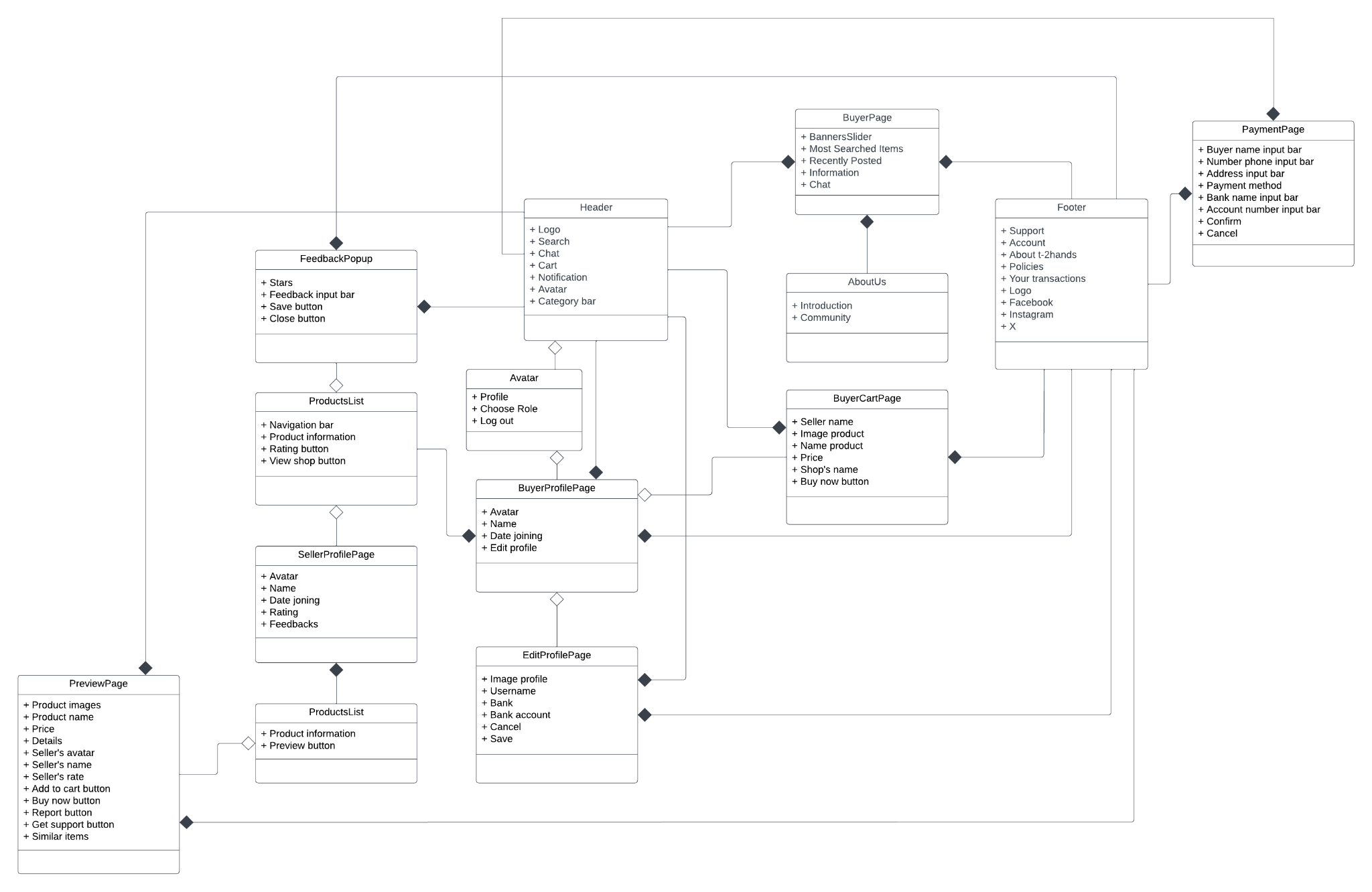
**Description:** This page is designed to exchange necessary information between buyers and sellers to make purchasing more convenient

### 4.1.6 SellerPage



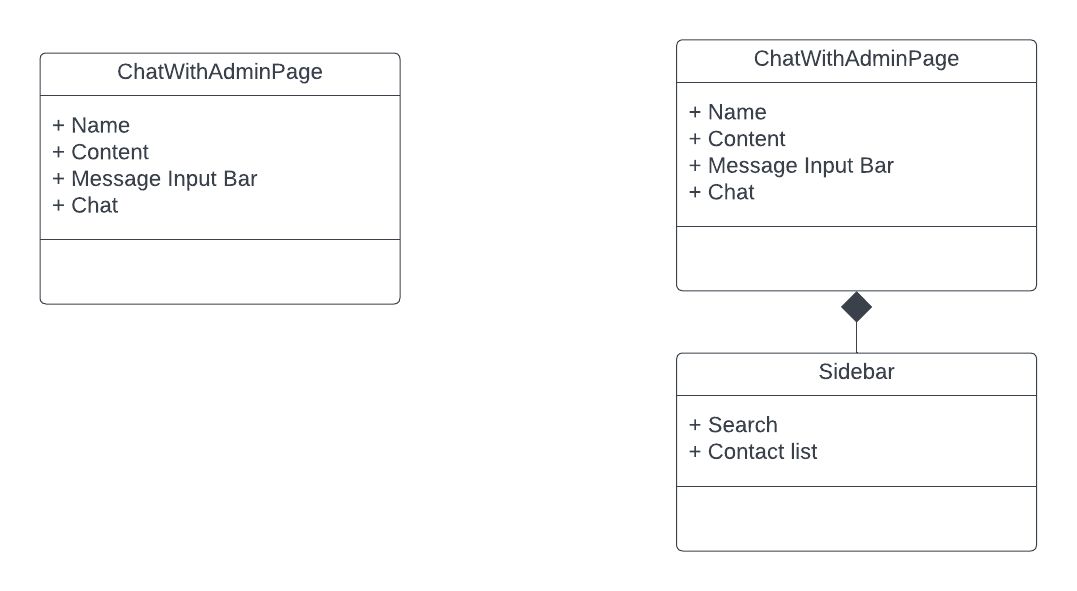
**Description:** The page is designed to give us an overview of the seller’s interface as well as a better understanding of how to shop and operate on T-2Hands. It provides sellers with many different functionalities such as: modifying profile, uploading products and reviewing feedback from their customers. The home page also displays the products of the others so that the sellers could survey the market to adjust their products’ information.

### 4.1.7 BuyerPage



**Description:** The page is designed to give us an overview of the buyer's interface as well as a better understanding of how to shop and operate on T-2Hands. This page will display product information so buyers can follow it, and also record reviews and feedback from users. And finally, there will be a payment page so users can choose a method to pay for the product they want to buy

### 4.1.8 ChatWithAdminPage

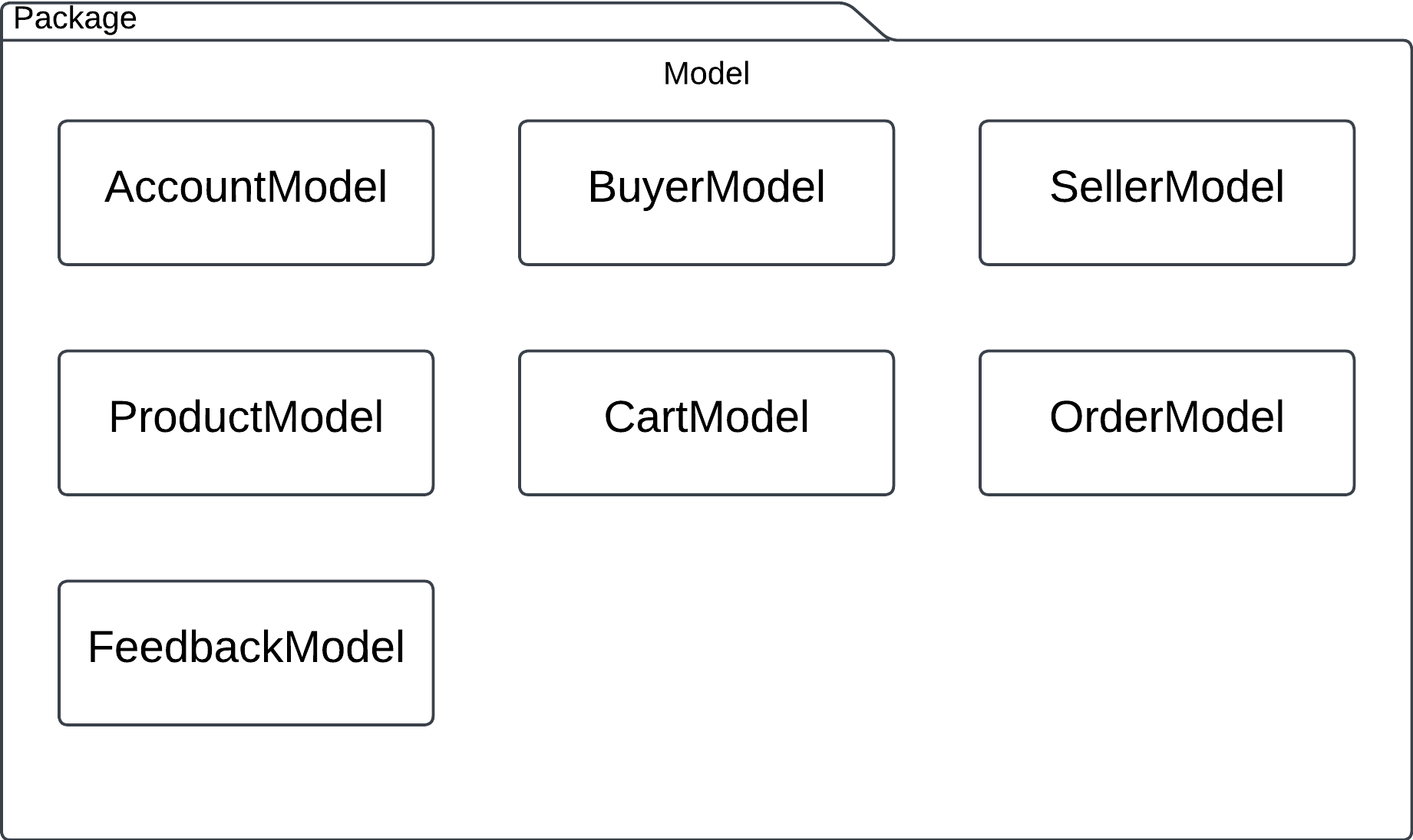


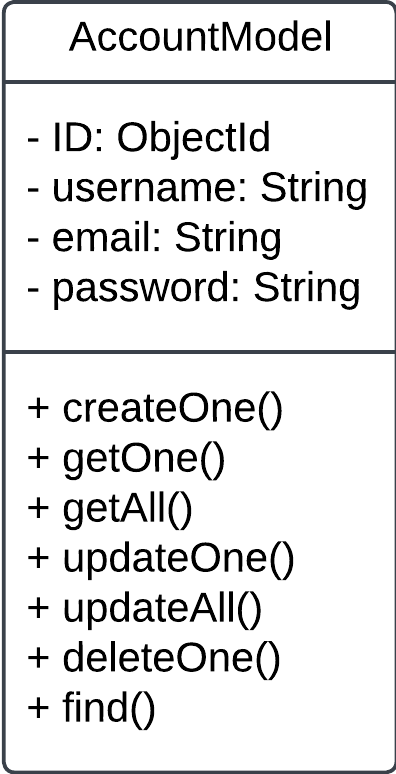
**Description**: This page is designed to create an interface so that the T-2Hands web admin can support, help, and resolve problems or requests from buyers or sellers. There are two viewpoints:

* User’s viewpoint: Users just chat with only 1 admin. Therefore, the sidebar in this viewpoint is not necessary to exist.
* Admin’s viewpoint: Admin has to support more than 1 user so he/she needs a sidebar to select a user.

## Component: Model

**Package Diagram**



**Description**: A model represents the data structures used in the application in the context of the user's system architecture. Models are responsible for defining the schema of the data and providing methods to interact with the database. In this case, the models are defined using Mongoose for MongoDB Atlas. There are seven main models, each with its purpose and features.

### 4.2.1 AccountModel

Purpose: The AccountModel is used to store and manage user account information

within the system.

Fields:

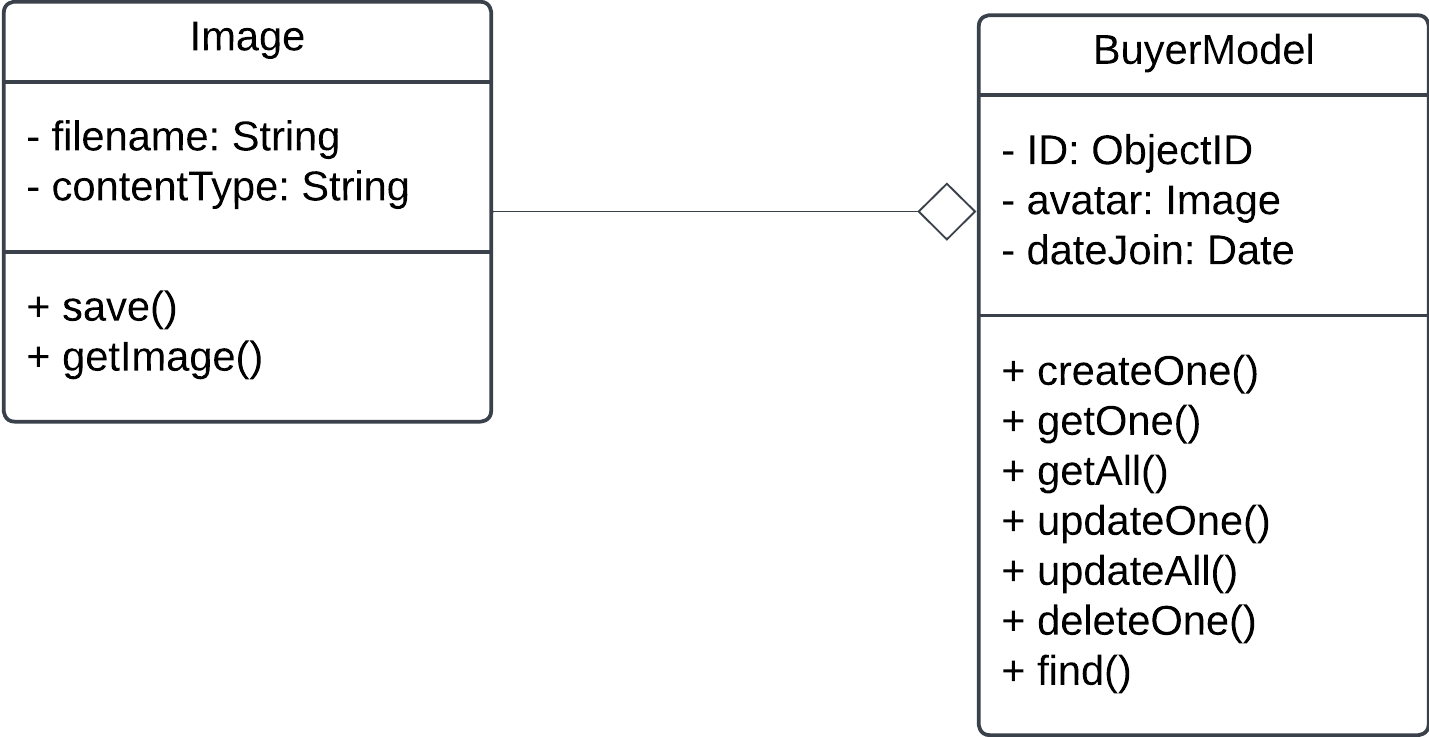
* ID: A unique identifier for each account.
* username: The user’s chosen name for logging into the system
* email: The user’s email address for password recovery
* Password: The user’s password for logging into the system

### 4.2.2 BuyerModel

Purpose: The BuyerModel is used to store and manage detailed information of the buyer, which is linked to their account.

Fields:

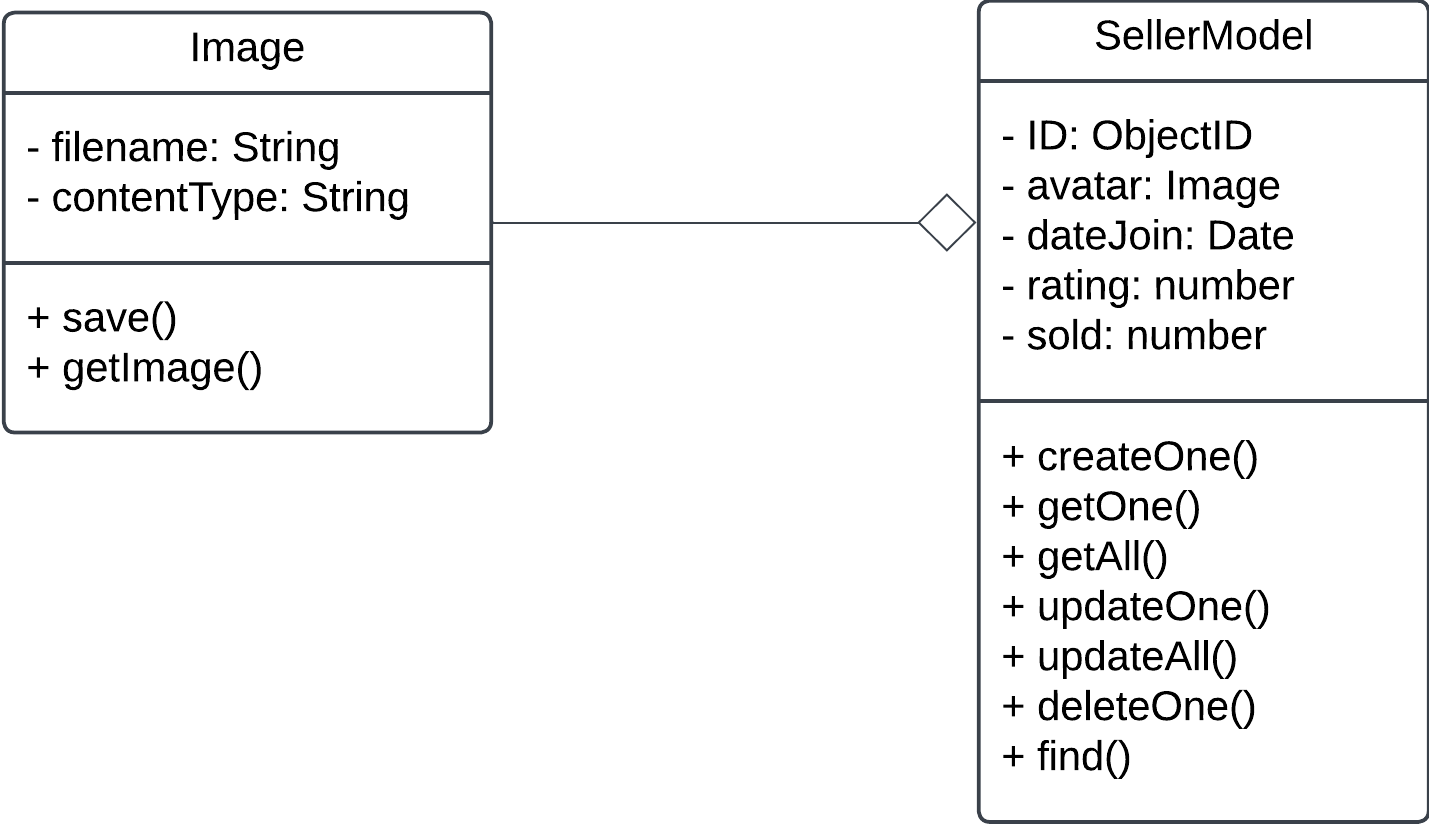
* ID: A unique identifier linked to the account
* avatar: A user’s profile image
  + filename: The name of the image file.
  + contentType: The MIME type of the image (.jpeg, .png, …)
* dateJoin: The date the user registered



### 4.2.3 SellerModel

Purpose: The SellerModel is used to store and manage detailed information of the seller, which is linked to their account.

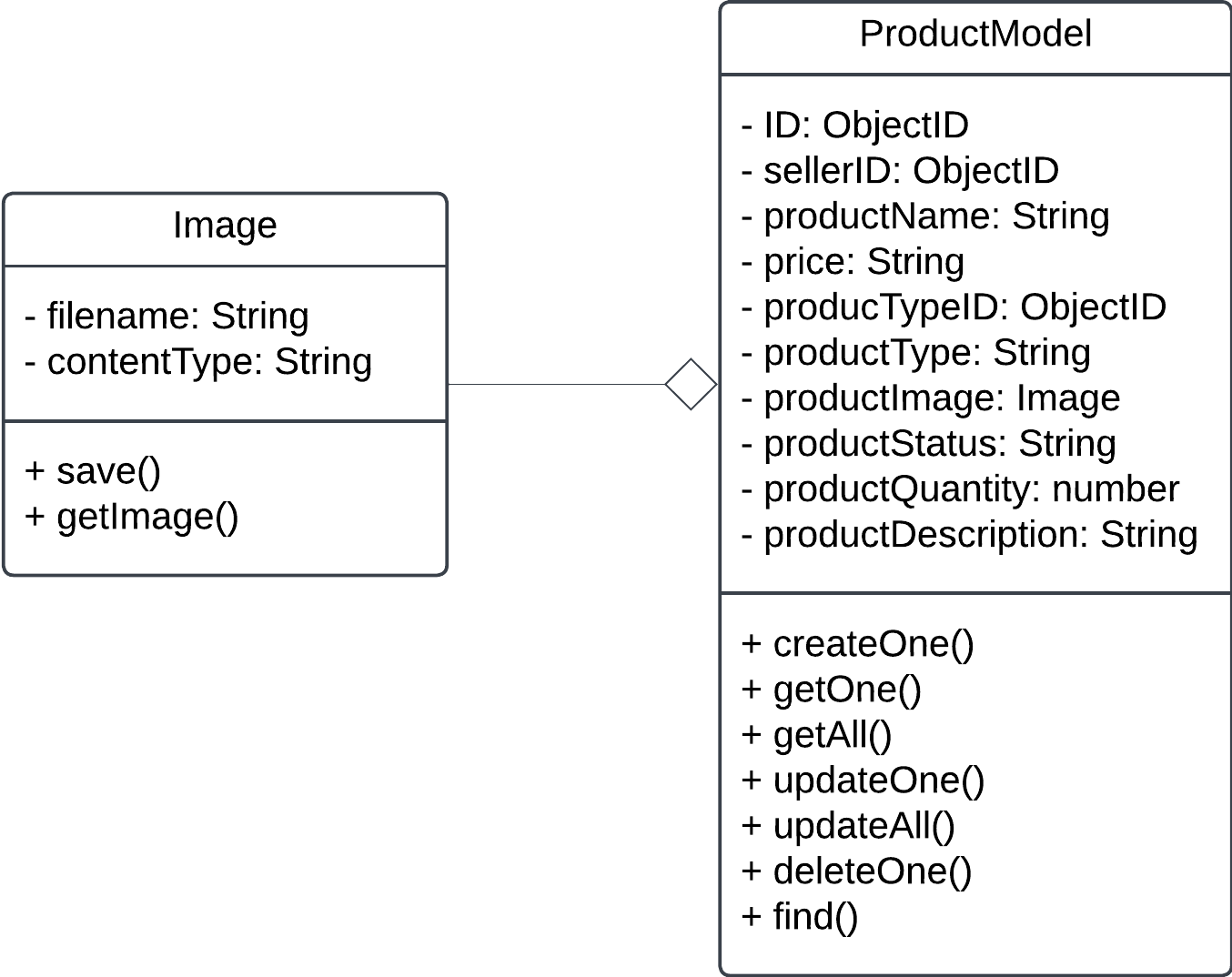
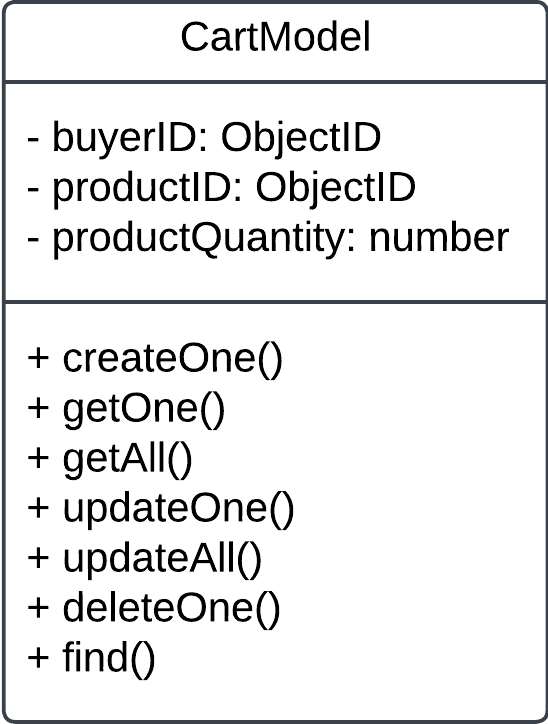
Fields:

* ID: A unique identifier linked to the account
* avatar: A user’s profile image
  + filename: The name of the image file.
  + contentType: The MIME type of the image (.jpeg, .png, …)
* dateJoin: The date the user registered
* rating: the seller rating, based on feedback about them
* sold: the number of products that the seller has sold

### 4.2.4 ProductModel

Purpose: The ProductModel is used to store and manage detailed information about each product that the seller uploads to their product list.

Fields:

* ID: A unique identifier for each product
* sellerID: ID of the seller of the product
* productName: A name of the product
* price: The price of the product
* productType: A type of the product
* productTypeID: A unique identifier for the type of product
* productImage: A product’s image
  + filename: The name of the image file.
  + contentType: The MIME type of the image (.jpeg, .png, …)
* productStatus: The current status of the product (available, sold, flag)
* productQuantity: The number of product the seller want to sell
* productDescription: The description of the product

### 4.2.5 CartModel

Purpose: The CartModel is used to manage detailed information about the buyer’s cart. This model stores the product that the buyer has already added to their cart.

Fields:

* buyerID: ID of the buyer of the cart
* productID: ID of the product in the cart
* productQuantity: the number of the above products that the buyer

has added to their cart

### 4.2.6 OrderModel

Purpose: The OrderModel is used to store and manage detailed information of the buyer’s order. This model captures all information about the order that the buyer has confirmed to purchase.

Fields:

* ID: A unique identifier for each order
* buyerID: ID of the buyer of the order
* productID: ID of the product that the buyer purchased
* orderDate: The date that the buyer confirmed to purchase
* orderStatus: The current status of the order (pending, delivered,canceled,...)
* paymentOption: The payment method that the buyer chooses
* receiverName: The name of the receiver
* address: The shipping address of the receiver
* phoneNumber: The phone number of the receiver

### 4.2.7 FeedbackModel

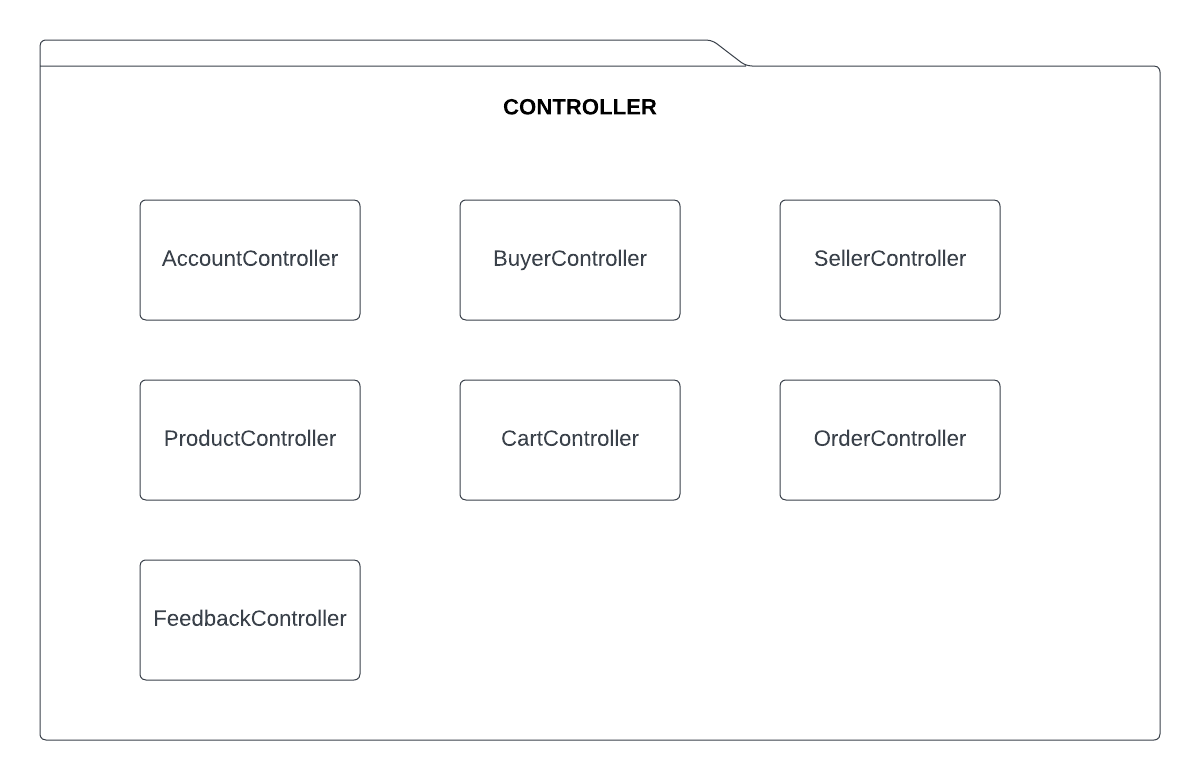
Purpose: The FeedbackModel is used to store and manage detailed information of the feedback that the seller received from the buyer.

Fields:

* ID: A unique identifier for each feedback
* orderID: ID of the order of the feedback
* comment: The comment of the buyer for the seller
* rating: The rating of the buyer for the seller

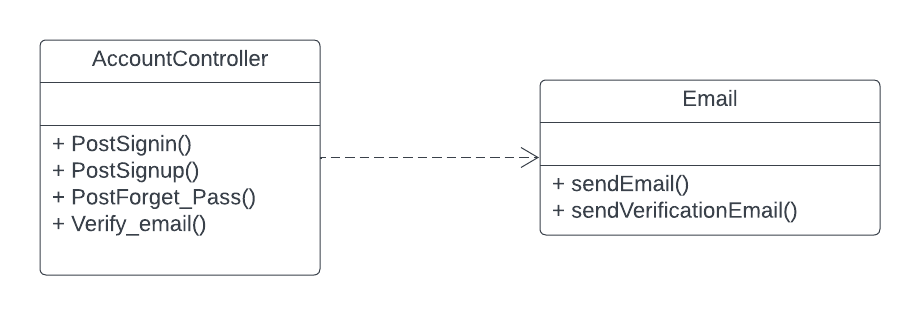
## 4.3 Component: Controller

**Package Diagram**



**Description:** Function as the core business logic of the application. Controllers handle incoming requests, communicate with models to fetch or modify data, and deliver responses back to the client. Each controller is dedicated to managing a specific part of the application.

### 4..3.1 AccountController



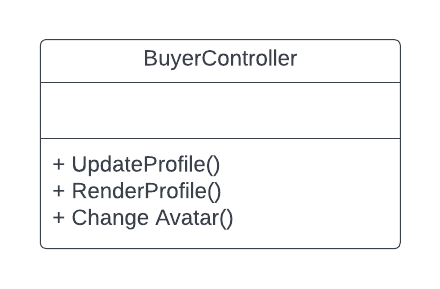
**Purpose:** Manage user account operations.

**Functions:**

* PostSignin: Verifies user credentials and logs them in if they are valid.
* PostSignUp: Registers a new account if the email and username are not already taken.
* PostForget\_Pass: Sends an email to reset the password if the provided email matches an existing account.
* Verify\_email: Confirms the user's email address by validating a token.

### 4.3.2 BuyerController

**Purpose:** Manage buyer profile operations.

**Functions:**

* UpDateProfile: Updates an existing profile or creates a new one if it doesn’t exist. Handles updating basic profile information and saves the updated or new profile to the database.
* RenderProfile: Retrieves and returns profile information based on the buyer ID.
* ChangeAvatar: Updates a buyer’s avatar image by processing the uploaded file and saving it to the buyer’s profile.

### 4.3.3 SellerController

**Purpose:** Manage seller profile operations.

**Functions:**

* UpdateProfile: Updates an existing profile or creates a new one if it doesn’t exist. Handles updating basic profile information and saves the updated or new profile to the database.
* RenderProfile: Retrieves and returns profile information based on the seller ID.
* ChangeAvatar: Updates a seller’s avatar image by processing the uploaded file and saving it to the seller’s profile.

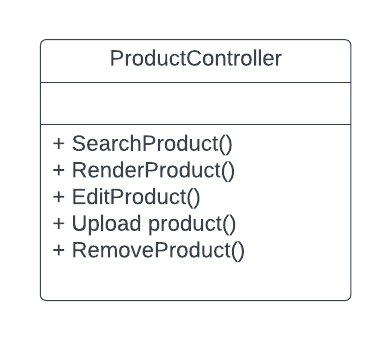
### 

### 

### 

### 4.3.4 ProductController

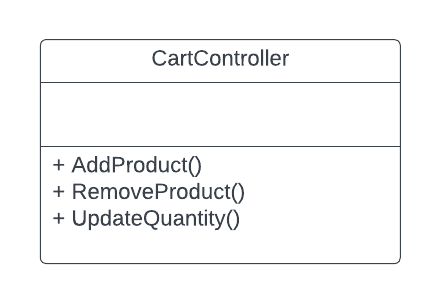
**Purpose**: Manage product-related activities in the system

**Functions:**

* SearchProduct: Searches for products based on criteria such as name, category, price, or product code. Accepts user input, queries the database, and returns a list of matching products for display.
* RenderProduct: Displays detailed information about a specific product. Retrieves product details using a product code or ID, fetches data from the database, and presents information such as name, price, description, and images.
* EditProduct: Edits information of an existing product in the system. Provides an input form for the admin to update product details, confirms changes, and saves the updated data to the database.
* UploadProduct: Adds a new product to the system. Allows the admin to input product details (name, price, description, category, images) and saves the new product to the database.
* RemoveProduct: Deletes a product from the system when it is no longer needed or valid. Takes a product ID, validates it, and deletes the product's data from the database.

### 4.3.5 CartController

**Purpose:** Manages the shopping cart, supporting adding, removing, and updating product quantities to ensure a smooth shopping experience

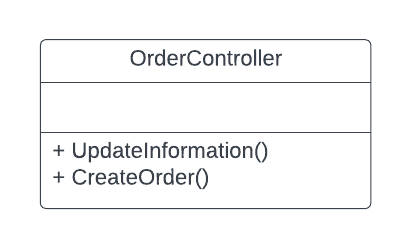


**Functions:**

* AddProduct: Adds a product to the cart. Receives product ID and quantity, checks the cart, adds a new item or increases the quantity, then saves the cart.
* RemoveProduct: Removes a product from the cart. Receives the product ID, finds and removes the product from the cart, and updates the cart state.
* UpdateQuantity: Updates the quantity of a product in the cart. Receives the product ID and new quantity, updates the quantity or removes the product if the quantity is zero, then saves the changes.

### 

### 4.3.6 OrderController

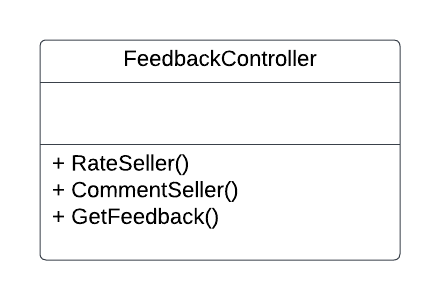


**Purpose:** Handle orders when a buyer makes a purchase.

**Functions:**

* UpdateInformation: updates orders details (eg: address, status,..). Fetches the orders, applies change, validates and saves updates.
* CreateOrder: Creates a new order, updates the product’s count and removes it out of the products list if the count is zero, and renders a success page.

### 4.3.7 FeedbackController

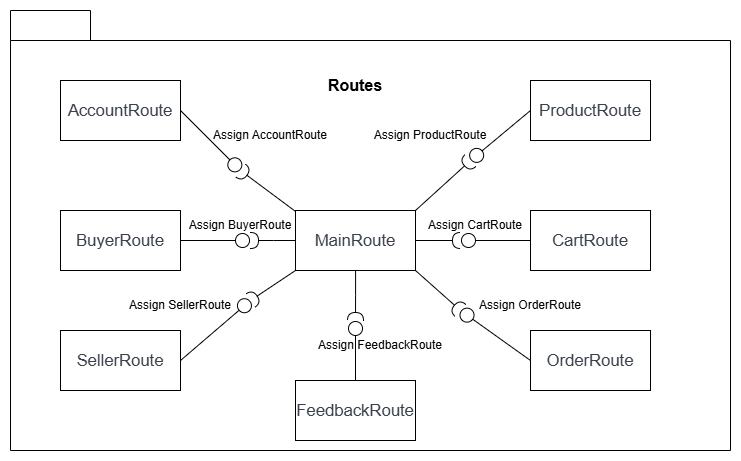
**Purpose**: Deal with feedback on sellers.

**Functions:**

* RateSeller: Updates a seller’s feedback by rating stars, verifies profile existence, and creates or updates feedback records.
* CommentSeller: Updates a seller’s feedback by adding comments, verifies profile existence, and creates or updates feedback records.
* GetFeedback: Retrieves and returns feedback, including stars and comments, and profile details for feedback.

## 4.4 Component: Routes

**Package diagram**



**Description:** In the system, each route matches up with a controller. Routes handle the URLs from

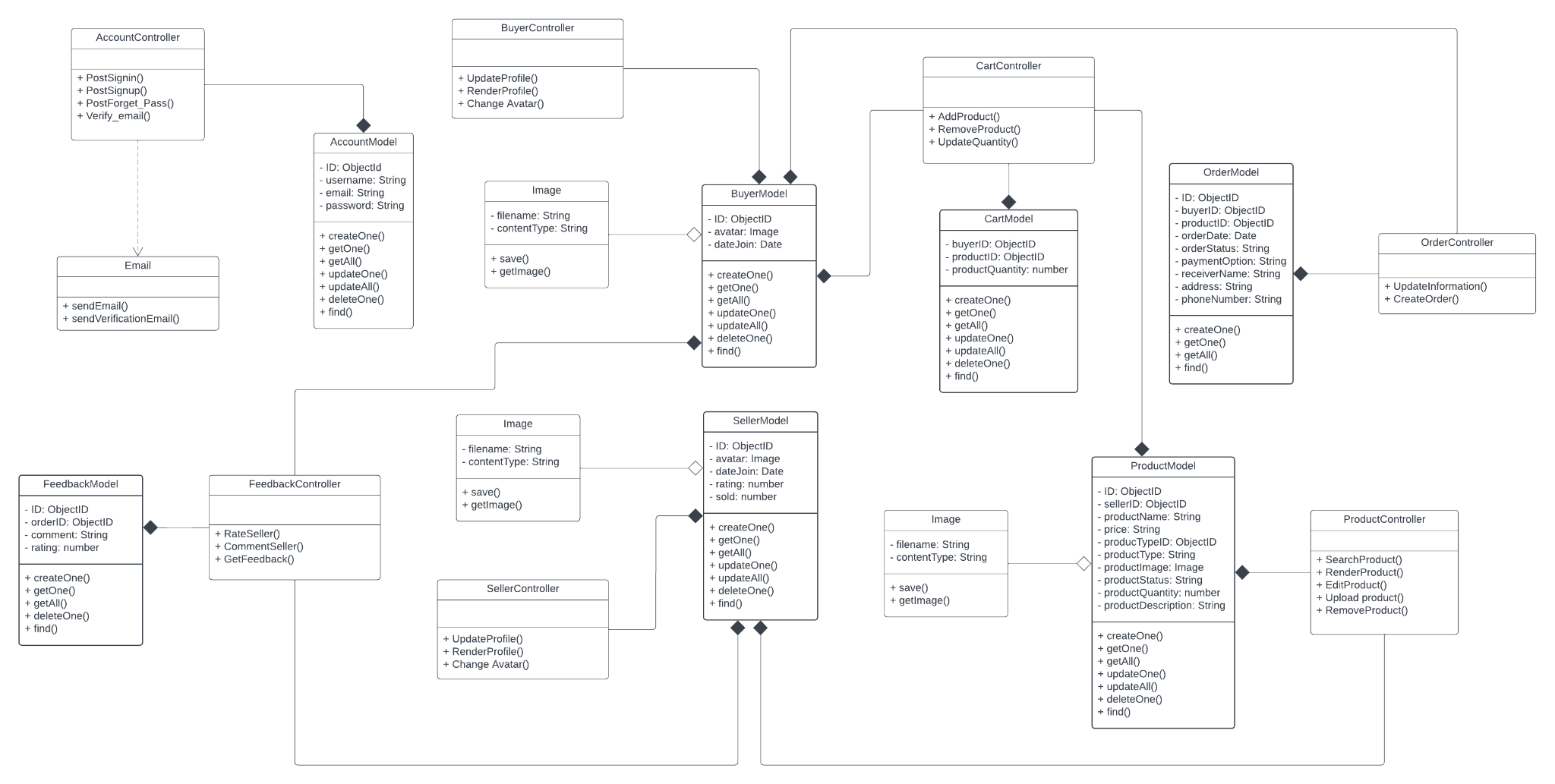
client request and send them to the controller for processing. Since there are 7 controllers ( Account, Buyer, Seller, Product, Cart, Order, Feedback), there will be 7 routes. Additionally, there will be a Main

route that sorts of requests and directs them to the correct route.

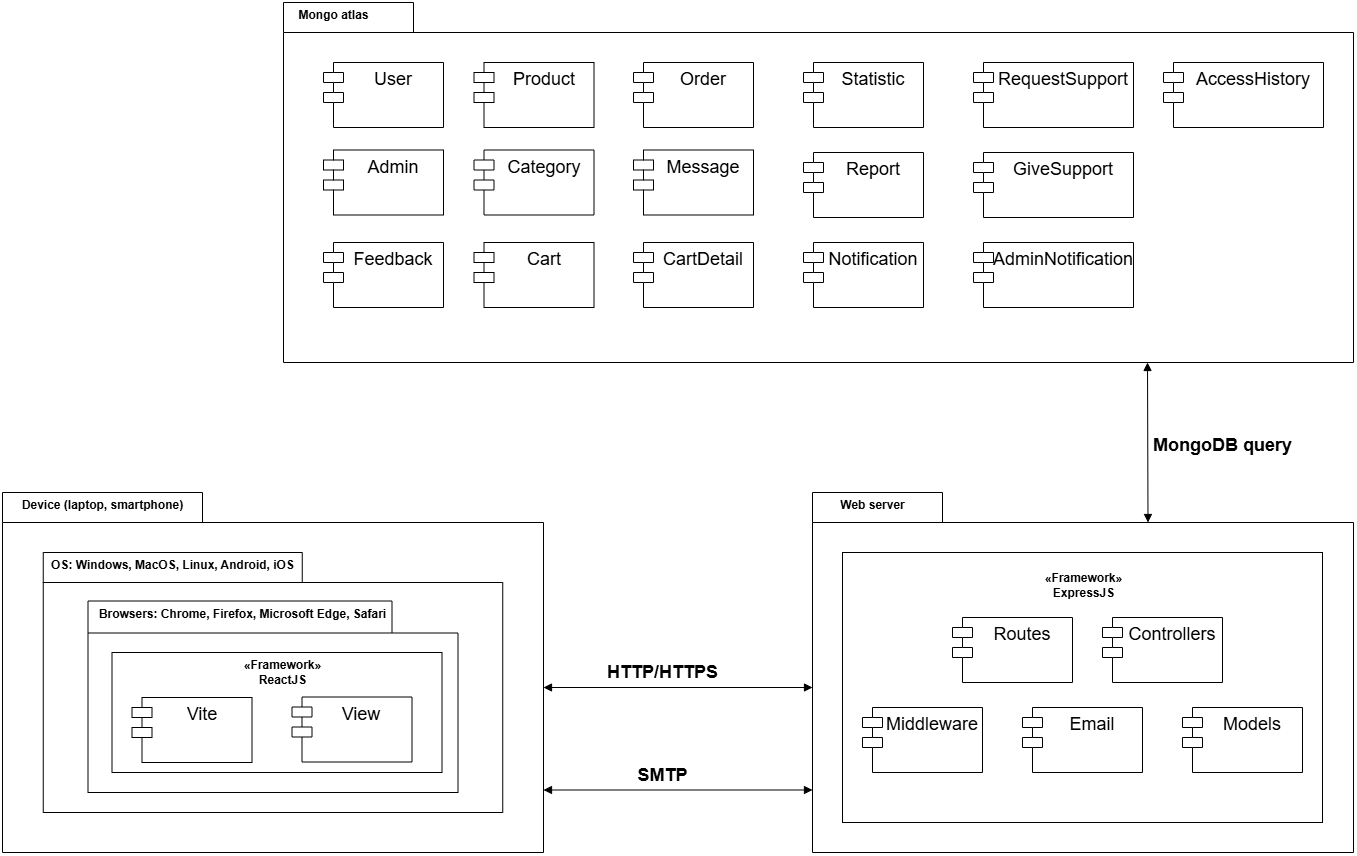
**For example**:

In the T-2Hands system, the Main Route serves the role of receiving user requests and redirect them to the appropriate routes for each specific function. Each route is linked to a controller that handles tasks such as account registration, viewing products, adding products to the cart, processing orders, and submitting feedback. For example, when a user wants to view a product, the Main Route will redirect the request to the route of the Product Controller to display the product. When the user wants to make a purchase, the request will be directed to the Cart Controller to add the product to the cart, and then to the Order Controller to complete the payment. This approach helps the system to be easily managed, organized, and extended with new functionalities.

## 4.5 Class Diagram



# Deployment

****

## 5.1. Client (Frontend):

* **Access:**
  + Users can access the website through browsers on personal devices such as computers and mobile phones.
* **Compatibility:**
  + Multi-platform support: Windows, macOS, Linux, Android, iOS.
  + Compatible with popular browsers like Chrome, Firefox, Microsoft Edge, Safari.
* **Connection:**
  + Browsers send HTTP requests (including GET, POST) to the backend server.
  + Connects to the server via HTTPS for security.
* **Technology Used:** React + Vite to build the user interface.
  + **Vite:** Used in the frontend for optimized builds and accelerated development.
  + **View:** Displays the user interface (includes pages that call components such as header, footer, etc.).
* **Features:**
  + Users can:
    - Log in/sign up for an account.
    - Receive verification emails when signing up or requesting a password reset.
    - Select roles (buyer/seller) and manage or edit personal information.
    - View product lists, add or delete products (for sellers).
    - View products, add or remove products from the cart, make purchases, and process payments (for buyers).

## 5.2. Server (Backend):

* **Technology Used:**
  + **Backend:** Express.js for handling business logic and API management.
  + **Email:** Nodemailer for sending verification or password recovery emails.
* **Hosting:**
  + **Frontend:** Vercel or Netlify for deploying the user interface.
  + **Backend:** Railway or Heroku for deploying the server.
  + **Database:** MongoDB Atlas for storing user and product data.
* **Main Components:**
  + **Email Handling:**
    - Create an endpoint to send account verification emails.
    - Create an endpoint to send password recovery emails.
    - Use Nodemailer to communicate with SMTP.
  + **Routes:**
    - A collection of endpoints defining URL paths and HTTP methods (GET, POST, PUT, DELETE).
    - Routes are organized by functionality, including:
      * Routes for authentication (login, signup).
      * Routes for user management.
      * Routes for products.
      * Routes for orders.
    - Each route assigns a controller or middleware to handle the request logic.
    - A main route file (index.js in the routes directory) connects all routes and is called in the main file.
  + **Controller:** Manages specific API logic for routes.
    - Organized by functionality:
      * Authentication and role management.
      * User management: Retrieve, edit, and delete accounts.
      * Product management: Retrieve product lists, get product details, add, and delete products.
      * Order management: Create orders, cancel orders, and handle transactions.
  + **Middleware:** Intermediate functions to handle shared logic before requests reach the controller, divided by functionality:
    - Verify if the user is a guest or logged in.
    - Role-based authorization: admin, buyer, seller.
    - Error handling.
    - Data validation and formatting.
  + **Models:** Define MongoDB schemas for users, products, and verification tokens.
  + **Email:** Implement email services for verification and password recovery support.

## 5.3. Database:

### 5.3.1. Technology Used:

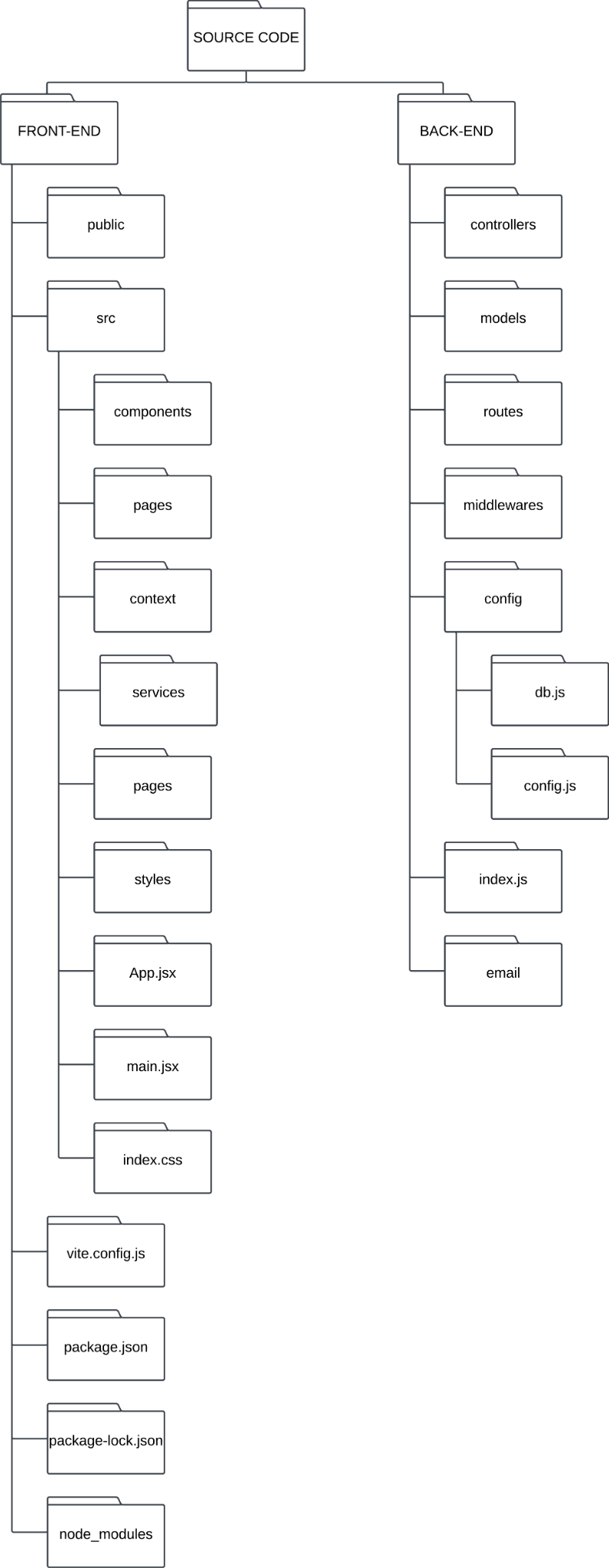
* **Database Management System:** MongoDB
  + Flexible, unstructured data, easily scalable.
  + Supports relationships between documents via **ObjectId**.
  + Integrates seamlessly with Node.js through the Mongoose library.

### 5.3.2. Schemas:

* **User:** Stores user information.
* **Admin:** Stores admin information.
* **Category**: Stores product categories.
* **Product:** Manages product details uploaded by sellers.
* **Order:** Manages orders.
* **Feedback:** Stores reviews and ratings from buyers.
* **Cart:** Manages buyers’ shopping carts.
* **CartDetails:** Stores detailed information about products in shopping carts.
* **AccessHistory**: used to manage Users' access history
* **Statistics:** Stores statistical data.
* **Messages:** Stores messages exchanged between users.
* **Notification:** Stores system-generated notifications for users.
* **AdminNotification**: Stores system-generated notifications for admins.
* **RequestSupport**: Manages support messages from users to admins.
* **GiveSupport**: Manages support messages from admins to users.
* **Report:** Store buyers’ reports.

# Implementation View

Our folder structure is organized to support both front-end and back-end development, with each part of the project clearly defined. It includes two main sections:



## 6.1 FRONT-END

This section contains all the files related to the user interface and client-side logic.

1. public: used for static assets like images, and icons.
2. src: the main directory contains the source code for the front-end application.
   * components: contains reusable React components or UI elements that can be used across different parts of the application.
   * pages: holds the main page components representing the routes ( Home, About, etc.) of the application.
   * context: implements React Context to manage the global state shared across components.
   * services: stores files related to API calls, context API for global state
   * styles: contains CSS or styling files for designing the front-end.
3. App.jsx: the main React component serves as the root of the application and contains routes and application-level logic.
4. main.jsx: the entry point for rendering the React application into the DOM.
5. index.css: global CSS styles for the application.
6. vite.config.js: configuration file for Vite, the front-end build tool used for bundling and serving the application.
7. package.json: defines project metadata, dependencies, scripts, and configurations for Node.js.
8. package-lock.json: auto-generated file that locks down the versions of dependencies installed, ensuring consistency across environments.
9. node\_modules: directory containing all the installed npm packages and dependencies.

## 6.2 BACK-END

This section contains all files related to server-side logic and data management.

1. controllers: contains functions or classes handling business logic, invoked by routes.
2. models: represents the structure of data entities, defining database schemas.
3. routes: contains endpoint definitions that connect HTTP requests to specific controller functions.
4. middlewares: includes middleware functions to process requests, such as authentication, and logging.
5. config: contains configuration files and settings used across the back-end, such as environment variables or database connections.
   * db.js: manages database connections or initialization logic.
   * config.js: stores project-wide configuration variables like API keys or mode settings.
6. index.js: entry point for starting the back-end server. It sets up the Express app, connects to the database, and listens for requests.
7. email: encapsulates all logic necessary for handling various email operations in the application such as sending password recovery via email.