<E-Commerce Web Application For Selling Digital Products>

Software Architecture Document

Version <1.9>

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| <19/Jul/24> | <1.0> | Complete part 1 | Nguyễn Kim Anh |
| <23/Jul/24> | <1.1> | Complete part 2 | Nguyễn Kim Anh |
| <24/Jul/24> | <1.2> | Add sample format for part 3 | Nguyễn Kim Anh |
| <26/Jul/24> | <1.3> | Modify part 4 | Nguyễn Kim Anh |
| <26/Jul/24> | <1.4> | Complete part 4 | Đinh Vũ Huân |
| <29/Jul/24> | <1.5> | Add diagram for components and update the description for 4.4 | Nguyễn Kim Anh |
| <29/Jul/24> | <1.6> | Class diagram | Quách Trần Quán Vinh |
| <05/Aug/24> | <1.7> | Section 6 | Lê Bảo Giang |
| <05/Aug/24> | <1.8> | Section 5 (Diagram) | Nguyễn Kim Anh |
| <07/Aug/24> | <1.9> | Modify section 6 | Nguyễn Kim Anh |

Table of Contents

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

[1.1. Purpose 5](#_heading=h.saudvn7mfmkn)

[1.2. Scope 5](#_heading=h.dhouz5itie9o)

[1.3. References 5](#_heading=h.dntt5jziav3)

[1.4. Acronyms 5](#_heading=h.joqceqxinkav)

[1.5. Overview 5](#_heading=h.ayg22nbpeu84)

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

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

[**4. Logical View 6**](#_heading=h.2et92p0)

[4.1 Component: View (Front - end) 7](#_heading=h.tyjcwt)

[4.1.1. Homepage 7](#_heading=h.sp4a9cvu0tlz)

[4.1.2. Cart Page 8](#_heading=h.sr2v99bz5nvc)

[4.1.3 Trending Page 8](#_heading=h.2kfjhl9x874o)

[4.1.4 Wishlist Page 8](#_heading=h.5ov12udm7ato)

[4.1.5 Buy Page (Checkout Page) 8](#_heading=h.uy9pl99ijwcl)

[4.1.6 Login Page 9](#_heading=h.r2u8jltbtv2k)

[4.1.7 Contact Page 9](#_heading=h.zdbpxd5x7hq0)

[4.1.8 Admin Page 9](#_heading=h.x8z0ria1t8h4)

[4.2 Component: Controllers (Back - end) 10](#_heading=h.lycbrpsj081o)

[4.2.1 Authentication controller 11](#_heading=h.2kdx1xdq09qw)

[4.2.2 Register controller 11](#_heading=h.e2tctoiiak6h)

[4.2.3 Login controller 11](#_heading=h.36z4yfgryc0p)

[4.2.4 Routes controller 11](#_heading=h.d7dvrk4jikdy)

[4.2.5 Products controller 11](#_heading=h.tsxua1cz8h0b)

[4.2.6 Users controller 11](#_heading=h.19as9uos9s2f)

[4.2.7 Cart controller 12](#_heading=h.o9eoo1nj4xht)

[4.3 Component: Model 12](#_heading=h.o3d7lsf7unb4)

[4.3.1 Users 13](#_heading=h.b23h9sffh169)

[4.3.2 Products 14](#_heading=h.z61r7hvn1qak)

[4.3.3 Order 14](#_heading=h.nezfh6xvaswx)

[4.3.4 Payment 15](#_heading=h.wn89z3xhtabx)

[4.3.5 Carts 15](#_heading=h.dmbmgvgafjdm)

[4.3.6 SingleCartItem 15](#_heading=h.fmjxro18ur3a)

[4.3.7 CheckOut 15](#_heading=h.dqncwtozfq7)

[4.3.8 Wishlist 16](#_heading=h.hjzcwutg4iw0)

[4.3.9 Admin 16](#_heading=h.f11dq81mpe9w)

[4.3.10 Customer 17](#_heading=h.1lykq2c0k9dm)

[4.3.11 Authentication 17](#_heading=h.wtb1ic37avbb)

[4.4 Connections between components: 18](#_heading=h.ol4zvnt1kp3c)

[4.5 Class diagram about the relationship of the components: 19](#_heading=h.wut776hwrn3r)

[**5. Deployment 20**](#_heading=h.3dy6vkm)

[**6. Implementation View 21**](#_heading=h.1t3h5sf)

Software Architecture Document

# Introduction

## 1.1. Purpose

This document provides the goal of software, descriptions of system structure and features, use cases, conventions and necessary requirements for the system as the different viewpoints and notations of different stakeholders.

## 1.2. Scope

The main scope of the document is about the architecture decisions made during the development process and architectural views of system architecture and use case model.

## 1.3. References

- Use case model, use case specification

- Vision document

- Diagram image from Geeksforgeeks

## 1.4. Acronyms

None

## 1.5. Overview

The document includes:

- Architectural goals and constraints

- Use-case model

- Logical view

- Components

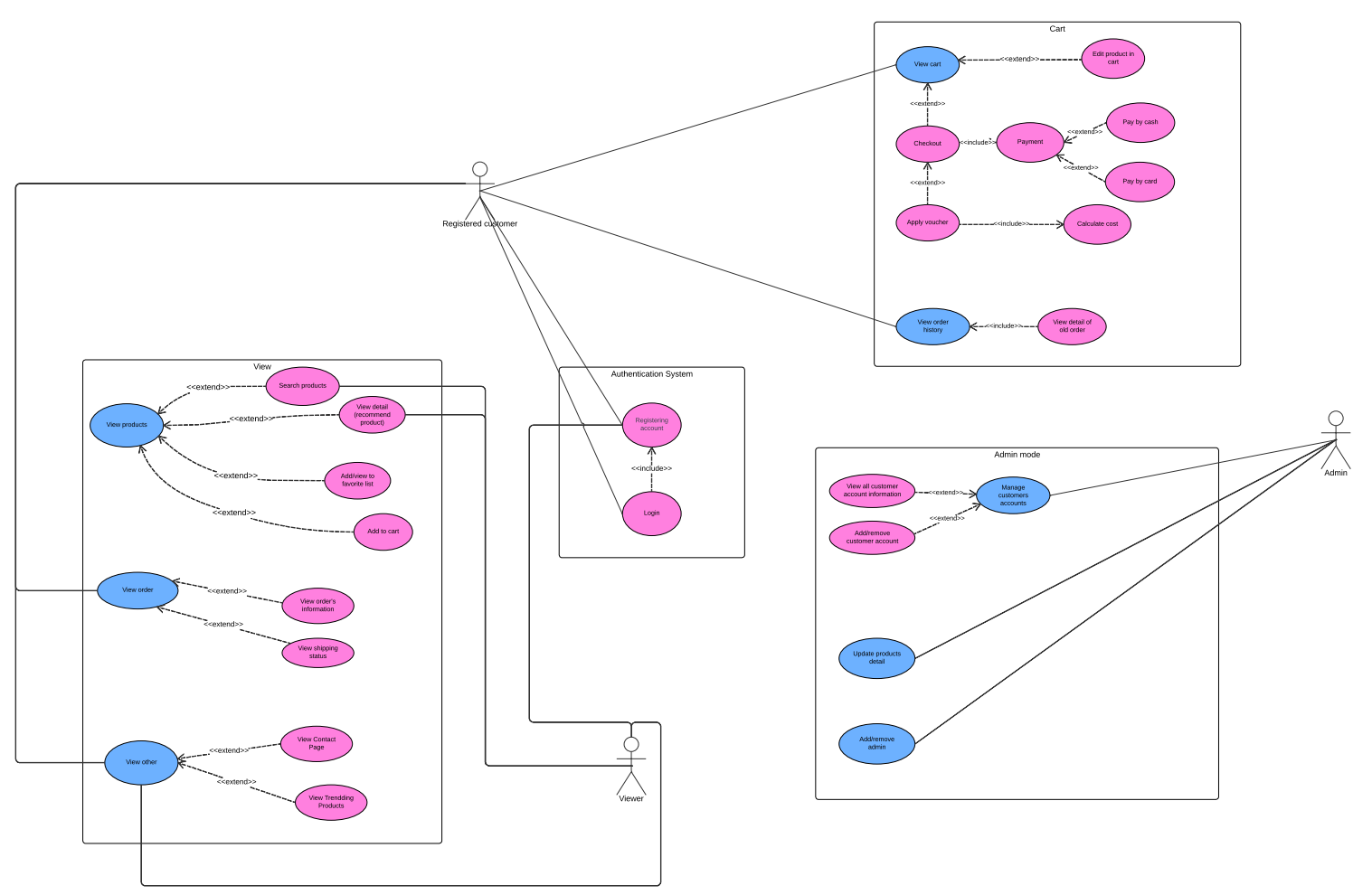
- Deployments

- Implementation views

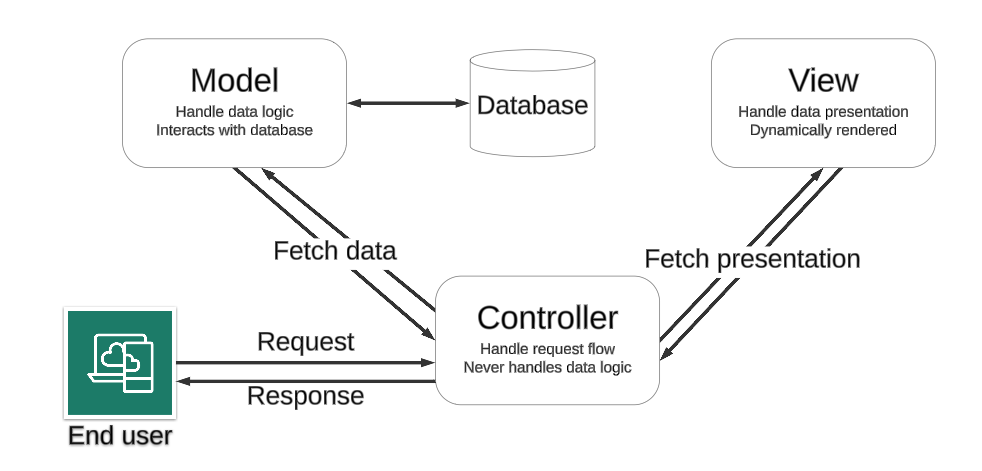
# Architectural Goals and Constraints

* Programming language:
* Front end: HTML/CSS
* Back end: NodeJS
* Development tools: Visual Studio Code, MongoDB Compass, Trello, Slack
* Team structure: consisting front-end developers, back-end developers, tester, BA to ensure collaboration and integration across all aspects of the application development.
* Schedule: A defined timeline for project milestones, including 3 phase: Inception, Elaboration, Construction. Dividing into 6 sprints (e.g., Agile methodology) to ensure timely delivery and adaptability to changes.
* Security: Data encryption, secure authentication to protect user data and privacy
* Design and implementation strategy:
* Adoption of a modular architecture to facilitate easier updates and maintenance: MVC (Model - View - Controller) design pattern
* Regular code reviews and testing to ensure functionality and reliability.
* Quality constraints:
* User friendliness: The user interface must be designed to be intuitive and user-friendly to provide the best user experience.
* Performance: The system must ensure fast response times and quick website loading to keep users from waiting (less than 20 seconds).
* Scalability: It should be able to scale to handle the growth in the number of users and data volume without affecting performance and availability.
* Portability: The application should be able to run on various hardware and operating systems with minimal changes.
* Distribution: The application should be easily deployable across different environments and compatible with different browsers, devices, and operating systems to ensure broad accessibility.
* Reuse: Encourage the reuse of existing libraries and frameworks to accelerate development and reduce redundancy.
* System constraints: Users already have browser and connect to the internet

# Use-Case Model

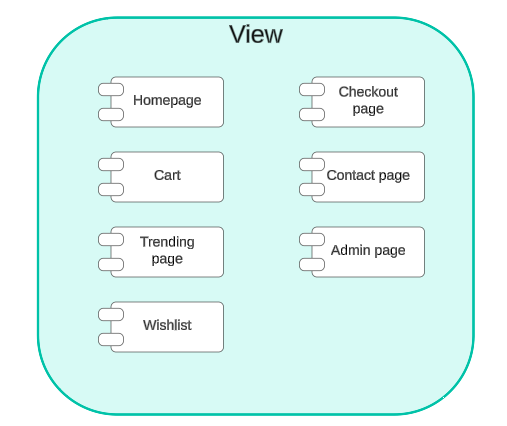


# Logical View

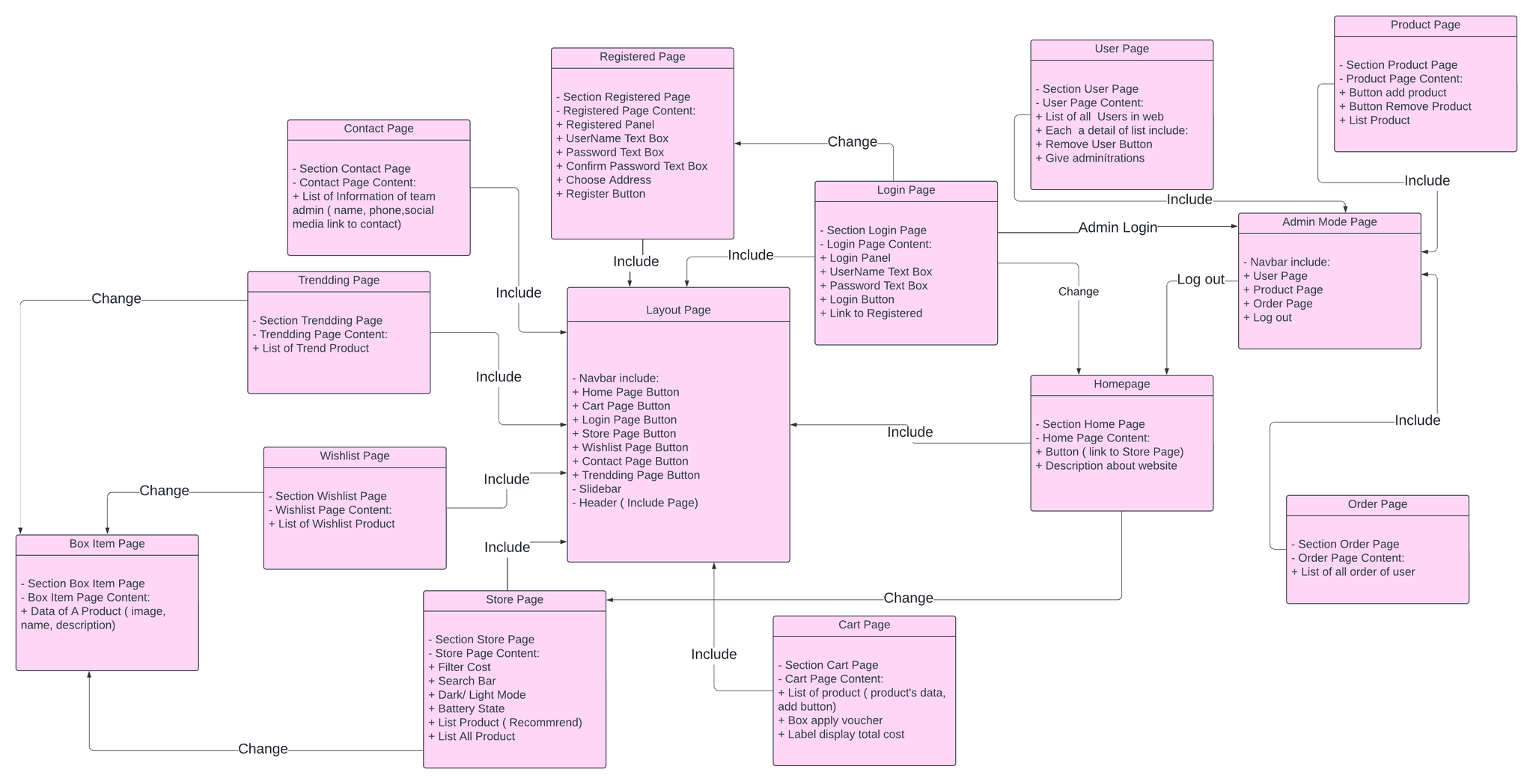


* **Description:** The architecture of the e-commerce website is based on a client-server model combined with the MVC (Model - View - Controller) design pattern. This structure ensures clear separation of concerns, scalability, and maintainability.
* Client: View manages the the user interface for browser interactions.
* Server: Controllers handles requests by retrieving data from models and returning it. Models takes responsibility for specify data structures and connect to the database. Controllers use models to fetch and process data then oversees UI updates and API communication.

## 4.1 Component: View (Front - end)



* **Description:** Frontend components handle the presentation and user interaction aspects of the application. They render the user interface, manage client-side logic, and communicate with the backend to fetch or update data as needed. The frontend is responsible for displaying data, processing user input, and maintaining a responsive and interactive user experience.
* **Technologies:**
* HTML: Provides the basic structure and content of web pages.
* CSS: Manages the layout, styling, and visual presentation of HTML elements.
* JavaScript: Handles client-side logic, user interactions, and dynamic content updates. It also manages communication with the backend through HTTP requests.



### 4.1.1. Homepage

* **Responsibilities:**
  + Display featured products and promotions.
  + Provide an overview of product categories and highlight trending items.
* **Components:**
  + **Product Carousel**: Showcases featured products with images and brief descriptions.
  + **Category Overview**: Displays a list or grid of product categories.
  + **Trending Products Section**: Highlights popular or trending products.
* **Communication:**
  + Utilizes HTTPS requests to the Backend to retrieve data on featured products, categories, and trending items.

### 4.1.2. Cart Page

* **Responsibilities:**
  + Display items currently in the shopping cart.
  + Manage item quantities, remove items, and update cart details.
  + Show the total price of items in the cart.
  + Proceed to checkout.
* **Components:**
  + **Cart Items List**: Displays the items added to the cart with options to modify quantities or remove items.
  + **Price Summary**: Shows the total price including any applicable taxes or discounts.
  + **Checkout Button**: Initiates the checkout process.
* **Communication:**
  + Utilizes HTTPS requests to the Backend for updating cart details and retrieving item information.

### 4.1.3 Trending Page

* **Responsibilities:**
  + Display a list of products currently trending or popular among users.
* **Components:**
  + **Trending Products Grid**: Showcases a list of trending products with images and brief details.
* **Communication:**
  + Utilizes HTTPS requests to the Backend to fetch data on trending products..

### 4.1.4 Wishlist Page

* **Responsibilities:**
  + Display and manage the user’s wishlist of saved products.
  + Allow users to add or remove products from the wishlist.
* **Components:**
  + **Wishlist Items List**: Displays products saved in the user’s wishlist.
  + **Add/Remove Buttons**: Provides functionality to modify the wishlist.
* **Communication:**
  + Utilizes HTTPS requests to the Backend to fetch and update wishlist data.

### 4.1.5 Buy Page (Checkout Page)

* **Responsibilities:**
  + Facilitate the checkout process by collecting user information and payment details.
  + Validate user input and process payment.
* **Components:**
  + **Order Summary**: Displays a summary of items to be purchased and total cost.
  + **Shipping Information Form**: Collects user’s shipping address and contact information.
  + **Payment Details Form**: Collects payment information and processes payment.
* **Communication:**
* Utilizes HTTPS requests to the Backend for order processing and payment integration.

### 4.1.6 Login Page

* **Responsibilities:**
  + Allow users to log in to their account or register for a new account.
* **Components:**
  + **Login Form**: Collects user credentials (email and password).
  + **Registration Form**: Collects information for new user accounts.
* **Communication:**
  + Utilizes HTTPS requests to the Backend for user authentication and registration processes.

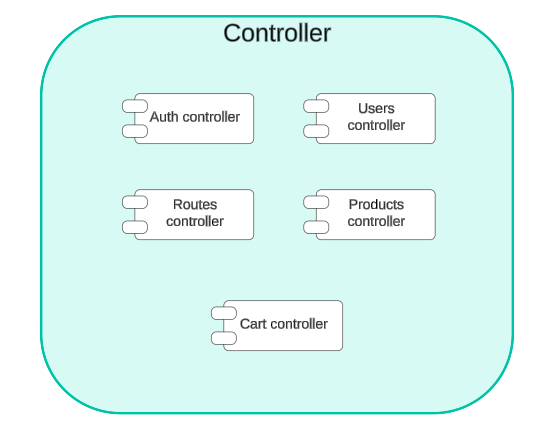
### 4.1.7 Contact Page

* **Responsibilities:**
  + Provide contact information for customer support.
  + Include a form for users to submit inquiries or feedback.
* **Components:**
  + **Contact Information**: Displays contact details such as phone number and email address.
  + **Contact Form**: Allows users to submit messages or requests.
* **Communication:**
  + Utilizes HTTPS requests to the Backend to submit the contact form data.

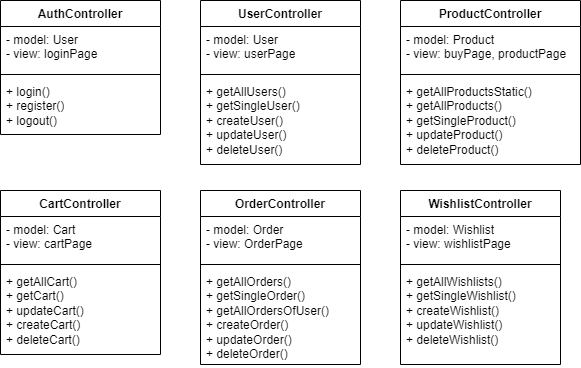
### 4.1.8 Admin Page

* **Responsibilities:**
  + Manage site content including products, orders, and user accounts.
  + Perform administrative tasks such as adding or editing products, viewing orders, and managing users.
* **Components:**
  + **Product Management Interface**: Allows for adding, editing, or deleting product listings.
  + **Order Management Interface**: Displays and manages orders placed by users.
  + **User Management Interface**: Manages user accounts and permissions.
* **Communication:**
  + Utilizes HTTPS requests to the Backend for performing administrative tasks.

## 4.2 Component: Controllers (Back - end)



* **Description:** Controllers act as an intermediary between the Model and the View. They process incoming requests, manipulate data using the model, and return the appropriate responses to the view.
* **Technologies:**
* Node.js: Server-side runtime environment.
* Express.js: Web framework for routing and middleware management.



### 4.2.1 Authentication controller

* **Responsibilities:** Manages user authentication, including login, registration, and logout processes.
* **Functions:**
  + Handles user registration by creating a new user in the database.
  + Manages user login by verifying credentials and generating authentication tokens.
  + Processes user logout by clearing session cookies.
  + Provides access to user-specific information on the dashboard.
* **Communication:** Uses HTTP/HTTPS for communication.

### 4.2.2 Register controller

* **Responsibilities**: Handles user registration logic and interacts with the database to create new users.
* **Functions**:
  + Checks if a username already exists.
  + Creates a new user record in the database.
  + Returns a response indicating the success or failure of the registration process.
* **Communication**: Uses HTTP/HTTPS for communication.

### 4.2.3 Login controller

* **Responsibilities**: Manages user login, verifying credentials, and issuing authentication tokens.
* **Functions**:
  + Validates user credentials against the database.
  + Generates and returns authentication tokens upon successful login.
  + Returns error messages for invalid credentials.
* **Communication**: Uses HTTP/HTTPS for communication.

### 4.2.4 Routes controller

* **Responsibilities**: Handles incoming requests HTTP, the controller determines which action should handle that request and mapping URLs to controller actions
* **Functions**:
  + Guiding requests to the appropriate place.
  + Mapping URLs to specific controller actions.
  + Apply constraints to route values.
* **Communication**: Uses HTTP/HTTPS for communication.

### 4.2.5 Products controller

* **Responsibilities**: Manages product-related operations such as fetching product details, creating new products, and updating product information.
* **Functions**:
  + Retrieves all products with optional filters and pagination.
  + Adds new products to the database.
  + Fetches detailed information about specific products.
* **Communication**: Uses HTTP/HTTPS for communication.

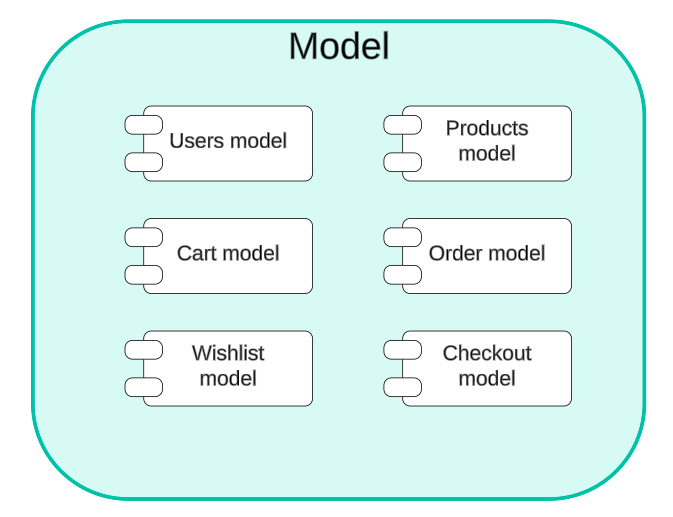
### 4.2.6 Users controller

* **Responsibilities**: Manages user-related operations such as fetching user details and updating user information.
* **Functions**:
  + Retrieves detailed information about specific users.
  + Updates user information such as passwords and profile details.
* **Communication**: Uses HTTP/HTTPS for communication.

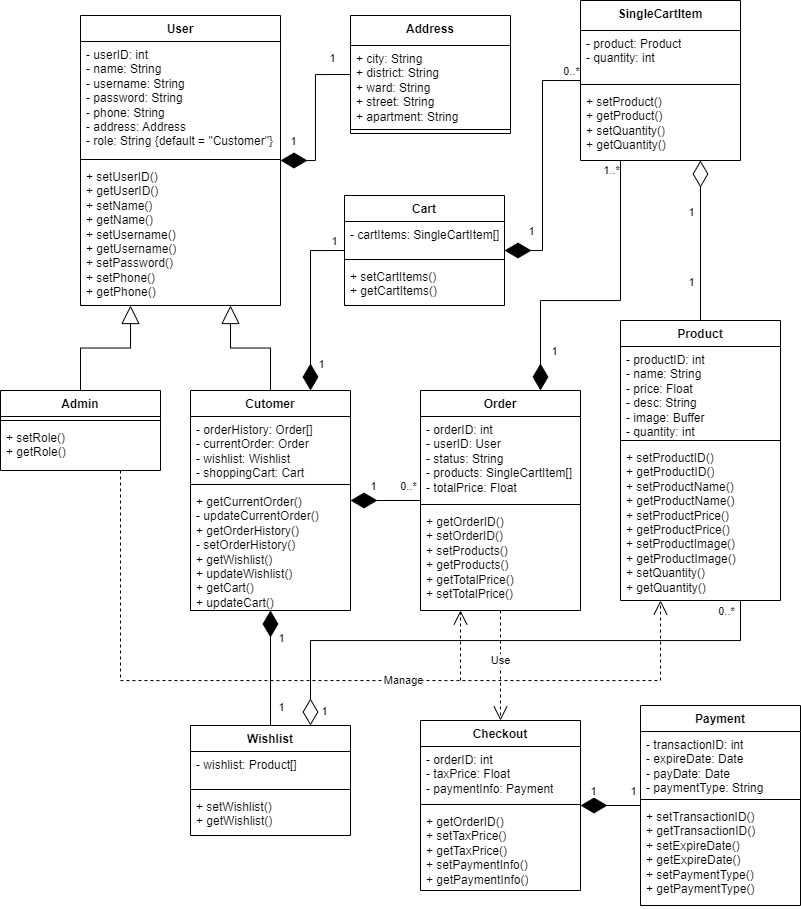
### 4.2.7 Cart controller

* **Responsibilities**: Manages the shopping cart functionality, including adding, removing, and updating products in the cart.
* **Functions**:
  + Adds products to the user's cart.
  + Removes products from the cart.
  + Updates the quantity of products in the cart.
* **Communication**: Uses HTTP/HTTPS for communication.

## 4.3 Component: Model



* **Description:** The Model component in this application is responsible for defining the data structures and handling interactions with the MongoDB database. It uses Mongoose to define schemas and models for different entities such as products and users.
* **Technologies:**
* **Mongoose:** An ODM (Object Data Modeling) library for MongoDB and Node.js. It provides a straightforward, schema-based solution to model application data.
* **MongoDB:** A NoSQL database used to store application data.



### 4.3.1 Users

* **Responsibilities:**
  + Define the structure and constraints for user data.
  + Handle user authentication processes, including password hashing and token generation.
  + Manage interactions with the MongoDB database for user-related operations.
* **Attributes:**
  + **UserID:** Integer, auto-generated. A unique identifier for each user.
  + **name:** String. The name of the user.
  + **username:** String. The unique username chosen by the user.
  + **password:** String. The user's password for authentication.
  + **phone:** String. The user's phone number, which should be 10-11 digits.
  + **address:** Object**.** Contains the user's address details
    - **City**: String. The city of the user's address.
    - **District**: String. The district of the user's address.
    - **Ward**: String. The ward of the user's address.
    - **Street**: String. The street of the user's address.
    - **Apartment**: String. The apartment or unit number of the user's address.
  + **role:** String, enum with values 'admin' and 'user', default is 'user'.
  + **methods:**
    - **getAllUsers():** Retrieves all users.
    - **getSingleUser():** Retrieves detailed information about a specific user by their ID.

### 4.3.2 Products

* **Responsibilities:**
  + Define the structure and constraints for product data.
  + Handle interactions with the MongoDB database for product-related operations.
* **Attributes:**
  + **ProductID:** int, auto-generated. A unique identifier for each product.
  + **name:** String. The name of the product.
  + **price:** Number. The price of the product.
  + **desc:** String. A description of the product.
  + **image:** Buffer for image data and String for content type. Contains the image of the product and its type.
  + **quantity:** int. The available stock quantity of the product.
  + **method:** 
    - **getAllProductsStatic():** Retrieves all products without any filters or pagination.
    - **getAllProducts():** Retrieves all products with optional filters and pagination.
    - **getSingleProduct():** Retrieves detailed information about a specific product by its ID.

### 4.3.3 Order

* **Responsibilities:**
  + Define the structure for order data.
  + Store and manage order information including customer ID, payment ID, shipping details, total cost, and items.
* **Attributes:**
  + **OrderID:** integer, auto-generated. Unique identifier for each order.
  + **UserID:** User. Reference to the user placing the order.
  + **status:** String. Current status of the order (e.g., pending, shipped, delivered).
  + **products:** SingleCartItem[]. Array of **SingleCartItem** objects.
  + **totalPrice:** Float. Total price for the entire order after all calculations.
  + **Methods:**
    - **getAllOrders():** Retrieves all orders with optional filters and pagination.
    - **createOrder():** Creates a new order with the provided data.
    - **deleteOrder():** Deletes an order by its ID.
    - **checkOutOrder():** Processes the checkout for a specific order, finalizing the order and updating its status.

### 4.3.4 Payment

* **Responsibilities:**
  + Define the structure for payment data.
  + Store and manage payment information including transaction ID, expiration date, payment date, and payment type.
* **Attributes:**
  + **transactionId:** Integer. Unique identifier for each payment transaction.
  + **expireDate:** Date. The expiration date of the payment method or transaction.
  + **payDate:** Date. The date when the payment was made.
  + **paymentType:** String. The type of payment (e.g., credit card, PayPal, bank transfer).

### 4.3.5 Carts

* **Responsibilities:**
  + Manage the storage and organization of products within the cart.
  + Allow users to modify product quantities and remove items as needed.
  + Provide detailed information about the contents of the cart.
* **Attributes:**
  + **userID:** int - An integer representing the ID of the user who owns this cart. This attribute links the cart to the user's account.
  + **cartItems:** SingleCartItem[] - An array of SingleCartItem objects. Each SingleCartItem represents a specific product in the cart.
  + **Methods:**
    - getCart() - Retrieves detailed information about the cart, including the list of items within it.
    - createCart() - Creates a new cart for a user.
    - updateQuantity() - Updates the quantity of a product within the cart.
    - deleteProductInCart() - Removes a product from the cart.

### 4.3.6 SingleCartItem

* **Responsibilities:**
  + Hold information about a specific product within the cart.
  + Track the quantity of the product in the cart.
* **Attributes:**
  + **product:** Product - An instance of the Product class that represents the product in the cart. This attribute includes all the details about the product.
  + **quantity:** int - An integer representing the quantity of the product in the cart.

### 4.3.7 CheckOut

* **Responsibilities:**
  + Manage the checkout process for an order.
  + Calculate the final price including taxes and discounts.
  + Validate payment information and handle receipts.
* **Attributes:**
  + **orderId: int** - A unique integer that identifies the order.
  + **taxPrice: float** - The amount of tax applied to the order.
  + **paymentInfo: Payment** - An instance of the Payment class that stores detailed payment information.
  + Methods:
    - **applyCoupon()** - Applies a discount code to the order.
    - **calcFinalPrice()** - Calculates the total final price of the order after applying any discounts and taxes.
    - **validatePayment()** - Validates the payment information provided.
    - **sendReceipt()** - Sends a receipt to the customer.

### 4.3.8 Wishlist

* **Responsibilities:**
  + Manage the user’s list of favorite products.
  + Allow for adding, updating, or removing items from the wishlist.
  + Provide functionality to view and delete the entire wishlist.
* **Attributes:**
  + **userID: int** - An integer representing the ID of the user who owns this wishlist. This links the wishlist to the user's account.
  + **wishlist: Product[]** - An array containing instances of the Product class. Each Product object represents a specific item in the wishlist, including details such as product ID, name, price, image, etc.
  + **Methods:**
    - **getWishlist()** - Retrieves detailed information about a specific user's wishlist, including the list of products contained in it.
    - **createWishlist()** - Creates a new wishlist for a user.
    - **updateWishlist()** - Updates the wishlist, such as adding a new product, removing a product, or changing the quantity of an item.
    - **deleteWishlist()** - Completely removes a wishlist.

### 4.3.9 Admin

* **Responsibilities:**
  + Manage user accounts and permissions.
  + Oversee and handle orders.
  + Add, update, and remove products from the system.
* **Methods:**
  + **manageUsers()** - Allows the administrator to perform various user management tasks such as adding new users, editing user details, deleting users, and changing user roles.
  + **manageOrders()** - Enables the administrator to manage orders, including viewing the list of orders, searching for specific orders, canceling orders, and updating order statuses.
  + **addNewProduct()** - Allows the administrator to add new products to the system, including entering details such as product name, price, description, and images.
  + **updateProduct()** - Enables the administrator to update information for an existing product in the system.
  + **deleteProduct()** - Allows the administrator to remove a product from the system.

### 4.3.10 Customer

* **Responsibilities:**
  + Manage personal profile and information.
  + Track the status of orders.
  + Review past orders and purchase history.
* **Methods:**
  + **updateUserProfile()** - Allows customers to update their personal information such as name, address, phone number, and email.
  + **checkOrderStatus()** - Enables customers to check the current status of their placed orders.
  + **checkOrderHistory()** - Allows customers to view their order history, including details of past purchases.

### 4.3.11 Authentication

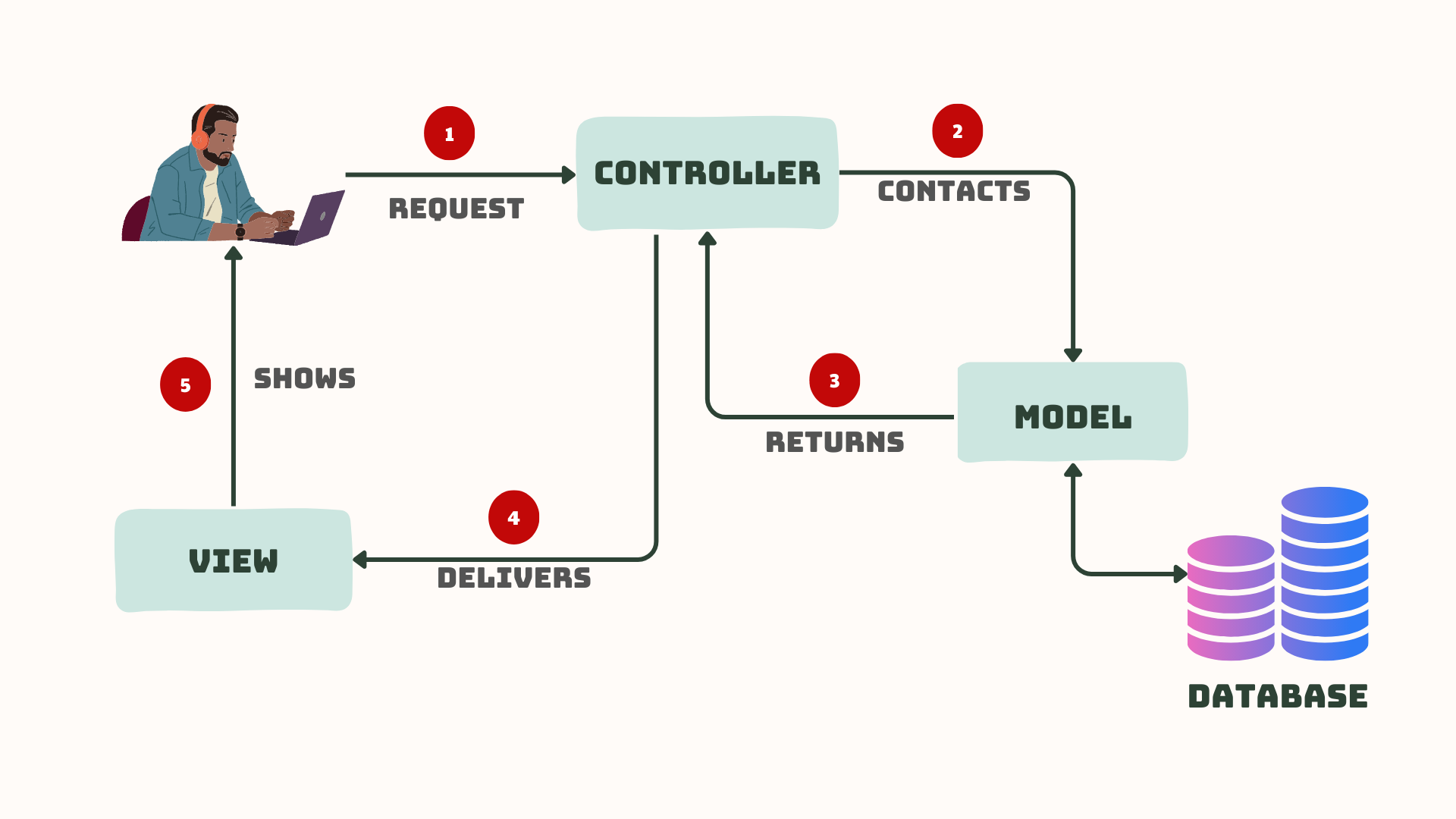
**Responsibilities:**

* + Validate user login credentials.
  + Handle user registration.
  + Ensure password strength.
  + Verify user permissions based on roles.

**Attributes:**

* + **user:** User - A private attribute that holds an instance of the User class, storing details about the currently logged-in or registering user. (Note: This is not shown in the diagram.)
  + **Methods:**
    - **loginUser()** - Called when a user attempts to log in. It verifies whether the provided username and password match the stored credentials in the database.
    - **registerUser()** - Called when a user wants to create a new account. It creates a new User object and stores the user’s registration details in the database.
    - **validatePassword()** - Checks the strength of a password. It ensures that the password meets criteria such as length, and includes uppercase letters, lowercase letters, numbers, and special characters.
    - **checkRolePermission()** - Verifies if the current user has permission to access a specific functionality based on their role.

## 4.4 Connections between components:



* **Description:** When a user clicks a button or submits a form, **View** notices and the **Controller** steps in. **Controller** responsible for handling user requests, logic, interacts with both the **Model** and **View** components. It talks to the **Model**, asking for data or telling it to update its state. **Model** component responds, providing data or modifying its internal state. **Model** talks to **View** and **View** component renders the user interface (UI) based on data received.

**1. User Interaction with View**

The user interacts with the View, such as clicking on a button or entering text into a form.

**2. View Receive User Input**

The View receives the user input and forwards it to the Controller.

**3. Controller Processes User Input**

* The Controller receives the user input from the View.
* It interrupts the input, performs any necessary operations (such as updating the Model), and decides how to respond.

**4. Controller Updates Model**

The Controller updates the Model based on the user input or application logic.

**5. Model Notifies View of Changes**

If the Model changes, it notifies the View.

**6. View Requests Data from Model**

The View requests data from the Model to update its display.

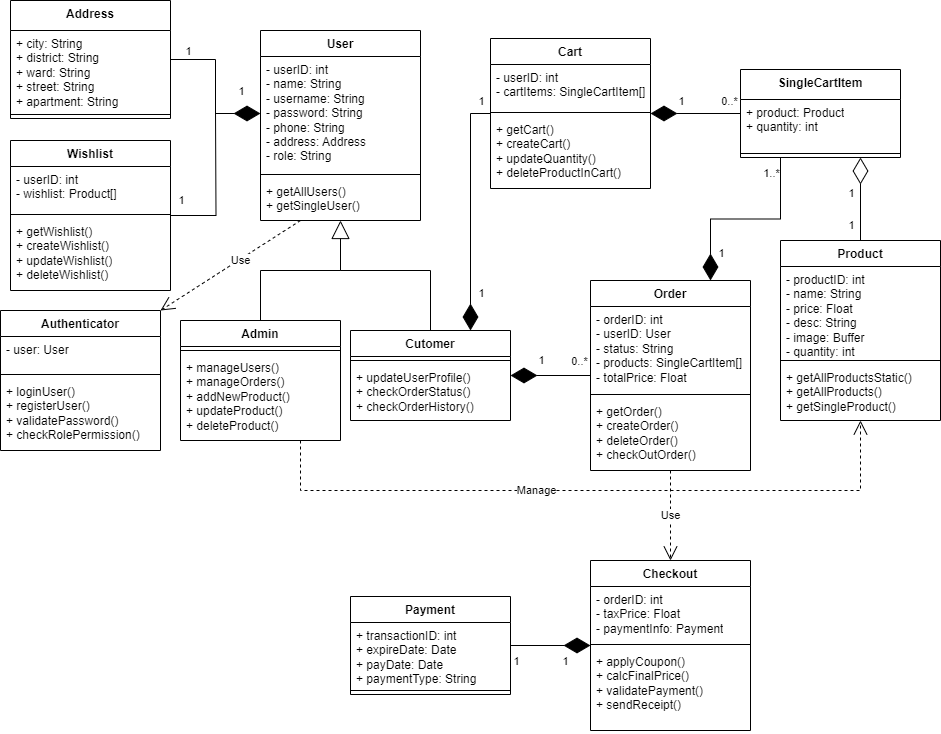
**7. Controller Updates the View**

The Controller updates the View based on the changes in the Model or in response to user input.

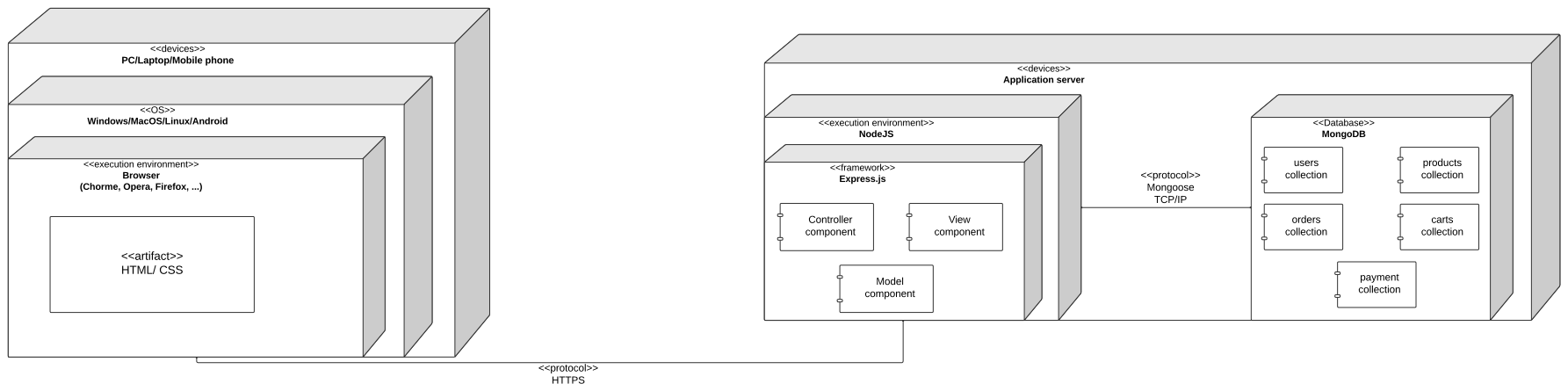
**8. View Renders Updated UI**

The View renders the updated UI based on the changes made by the Controller.

## 4.5 Class diagram about the relationship of the components:



# 5. Deployment



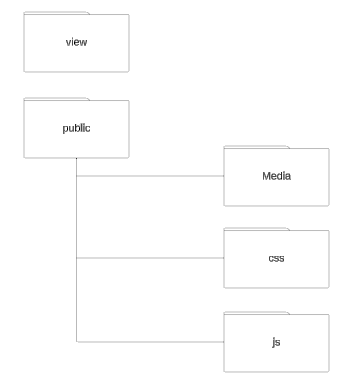
*Deployment diagram*

* **Description:**
* Server:
* The server is executed in NodeJS environment using ExpressJS framework.
* The server has **Controller** component to manage the server logic, interacts with both the **Model** and **View** components.
* **Model** component interact with database (responds, providing data)
* **View** component in server control the process of rendering the user interface (UI)
* For the database, we use MongoDB (NoSQL database) that works with NodeJS, to store data from the application. There are 5 collections: users, products, orders, carts and payment
* Client:
* The application uses HTML/CSS to create the interface, runs on web browsers: Chrome, Opera, Firefox… and also run on different operating systems: Windows, MacOS, Android
* Client communicates with server by HTTPS as protocol

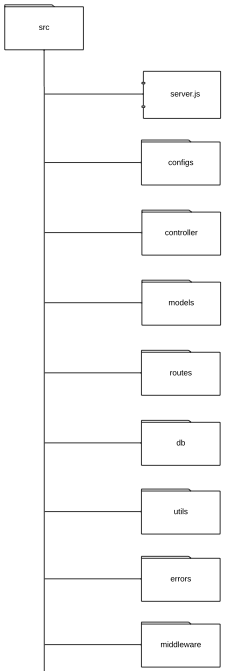
# 6. Implementation View

***Source Code implementation***

* For front-end:



* Folder **view**: is used to store the view templates for the application. The view engine is set to ejs, this folder will contain .ejs files which are used to render HTML pages dynamically.
* Folder **public:** is used to serve static files such as JavaScript (js) that are used on the client side responsible for handling some front-end interactions, CSS (css), images (Media). These files are directly accessible by the client.
* For back-end and database:



* Folder **controllers:** contains controller files that defined the logic for handling requests, interact with models to fetch and update data and then send responses back to the client
* Folder **models:** contains model definitions that represent the data structure of this application. **Models** interact with the database to perform operations.
* Folder **services:** contains service files that encapsulate the business logic and interact with models. **Services** are used by **controllers** to perform operations (authService.js: business logic related to authentication…)
* Folder **middleware:** Contains middleware functions that process requests before they reach the route handlers (check authentication, handling errors).
* Folder **routes:** contains route definitions for different parts of the application. These routes handle HTTP requests and responses
* Folder **config:** contains configuration files for the application, such as database configuration, environment variables, and other settings.
* Folder **ultils:** contain utility functions (create token users…)
* Folder **db:** contain code file to take responsibility for connecting to the database.
* **server.js**:
* Location: ./src/server.js
* The main entry point of the application. This file sets up the Express server, configures middleware, and defines the routes.