PotPal

Software Architecture Document

Version 1.2

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 20/11/24 | 0.1 | Introduction + Architectural Goals and Constraints | Lê Thanh Minh Trí |
| 21/11/24 | 0.2 | Added UML diagram and description of Controller | Phạm Anh Văn |
| 22/11/24 | 0.3 | Added UML diagram and description of View | Nguyễn Khánh Toàn |
| 22/11/24 | 0.4 | Added UML diagram and description of Model  (4.1.1 to 4.1.5) | Lê Thanh Minh Trí |
| 22/11/24 | 0.5 | Added UML diagram and description of Model  (4.1.6 to 4.1.9) | Cao Hoàng Lộc |
| 22/11/24 | 0.6 | Added Logical View diagram | Phạm Minh Mẫn |
| 23/11/24 | 0.7 | Added Database diagram | Cao Hoàng Lộc |
| 24/11/24 | 1.0 | Updated Admin View and First version | Lê Thanh Minh Trí |
| 06/11/24 | 1.1 | Added Deployment Overview and Diagram | Lê Thanh Minh Trí |
| 07/11/24 | 1.2 | Added Implementation View | Lê Thanh Minh Trí |

Table of Contents

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

[1.1 Purpose 5](#_heading=h.30j0zll)

[1.2 Scope 5](#_heading=h.xy2r2a3d6fsu)

[**2. Architectural Goals and Constraints 5**](#_heading=h.tynb18vkm9zk)

[■ Goals 5](#_heading=h.b9iok2ic2jwh)

[■ Constraints 6](#_heading=h.76heeksh2u70)

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

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

[4.1 Model 8](#_heading=h.2et92p0)

[4.1.1 Component: User 8](#_heading=h.3d2crn1hhc7l)

[4.1.2 Component: Product 9](#_heading=h.1fb344rgykj9)

[4.1.3 Component: Category 10](#_heading=h.130uhhuvzxjh)

[4.1.4 Component: Cart 10](#_heading=h.4wmhyczi89sg)

[4.1.5 Component: Discount 11](#_heading=h.r3m64hjecox1)

[4.1.6 Component: Order 11](#_heading=h.gz0gbwu1mfne)

[4.1.7 Component: Statistic 12](#_heading=h.tcs54faq81jp)

[4.1.8 Component: Checkout 13](#_heading=h.sax970vdm2nk)

[4.1.9 Component: Review 13](#_heading=h.y3zvi1scydus)

[4.2 Controller 14](#_heading=)

[4.2.1 Component: Authentication Controller 14](#_heading=h.9jyv50d9e4fv)

[4.2.2 Component: Product Controller 15](#_heading=h.no7pvv5w5cly)

[4.2.3 Component: Account Controller 16](#_heading=h.wo2cnjol8mao)

[4.2.4 Component: Discount Controller 17](#_heading=h.akxn0759l4nq)

[4.2.5 Component: Category Controller 18](#_heading=h.uja2xw5vwqqi)

[4.2.6 Component: Order Controller 18](#_heading=h.qwv4lp6rt45w)

[4.2.7 Component: Checkout Controller 19](#_heading=h.mc2we5ipr53q)

[4.3 View 20](#_heading=h.f159lad788sc)

[4.3.1 General View 20](#_heading=h.glmy7j9m9a7f)

[4.3.1.1 Component: App 20](#_heading=h.tfudjie7cyzp)

[4.3.1.2 Component: NavBar 20](#_heading=h.89crordby9sd)

[4.3.1.3 Component: Footer 20](#_heading=h.yg6wdpr1vdsd)

[4.3.1.4 Component: Main 20](#_heading=h.aflqulkztwc7)

[4.3.1.5 Component: Home 21](#_heading=h.qe515fvv9034)

[4.3.1.6 Component: Products 21](#_heading=h.clgjr4s0vp8c)

[4.3.1.7 Component: DetailedProduct 21](#_heading=h.5qn4w8vgfmkh)

[4.3.1.8 Component: Cart 21](#_heading=h.oj63o5c5wpkh)

[4.3.1.9 Component: Payment 21](#_heading=h.q3vuhe12gfy)

[4.3.1.10 Component: UserDashboard 21](#_heading=h.y9epgy2wqh0y)

[4.3.1.11 Component: Login 21](#_heading=h.ywnyblvi3blk)

[4.3.1.12 Component: Signup 22](#_heading=h.hcomq37ja69s)

[4.3.1.13 Component: About 22](#_heading=h.j16bxadziwxz)

[4.3.2 Admin View 22](#_heading=h.kozd1nbzow2d)

[4.3.2.1 Component: Navigator 22](#_heading=h.5h8m3f18me4w)

[4.3.2.2 Component: ProductManage 22](#_heading=h.yoea2x79hq68)

[4.3.2.3 Component: CustomerManage 23](#_heading=h.q0e8cqq1lix5)

[4.3.2.4 Component: OrderManage 23](#_heading=h.rja6pmyi2mx8)

[4.3.2.5 Component: DiscountManage 23](#_heading=h.64bu804i0xjd)

[4.3.2.6 Component: StatisticView 23](#_heading=h.eg38wocwcb3e)

[4.4 Component: Database 24](#_heading=h.qycnjvts0whk)

[**5. Deployment 25**](#_heading=h.tyjcwt)

[5.1 Deployment Overview 25](#_heading=h.aq18wz3yjcdq)

[5.2 Deployment Diagram 25](#_heading=h.elhdng32w946)

[**6. Implementation View 26**](#_heading=h.3dy6vkm)

[6.1 Implementation View Overview 26](#_heading=h.od041ltjt5v7)

[6.2 Implementation View Structure 27](#_heading=h.4u7k0wtyw73m)

Software Architecture Document

# Introduction

This document is intended to guide stakeholders in understanding the system's architecture, key components, and their interactions. It outlines the purpose and scope of the architecture, the constraints and requirements shaping its design, and the logical and deployment views of the system.

## Purpose

The purpose of this document is to define the architectural framework of *PotPal*, ensuring the system's scalability, maintainability, security, and usability. It serves as a reference for the development and deployment phases, aiding in the implementation of a reliable and efficient system.

## Scope

This document provides a structured overview of the software architecture for *PotPal*. It serves as a comprehensive guide for understanding the system's design, including its components, interactions, and the rationale behind key architectural decisions.

The scope of this document includes:

* A detailed overview of the system's architecture and its organization.
* Descriptions of individual components and their roles within the system.
* Explanation of how components interact and exchange data.
* Documentation of the technology stack, frameworks, and tools used to implement the system.
* Diagrams and models describing the system architecture.

# Architectural Goals and Constraints

This section outlines the primary goals and constraints that guide the architecture of *PotPal*, ensuring alignment with the project's requirements and technical feasibility.

### **Goals**

The architectural goals for *PotPal* include:

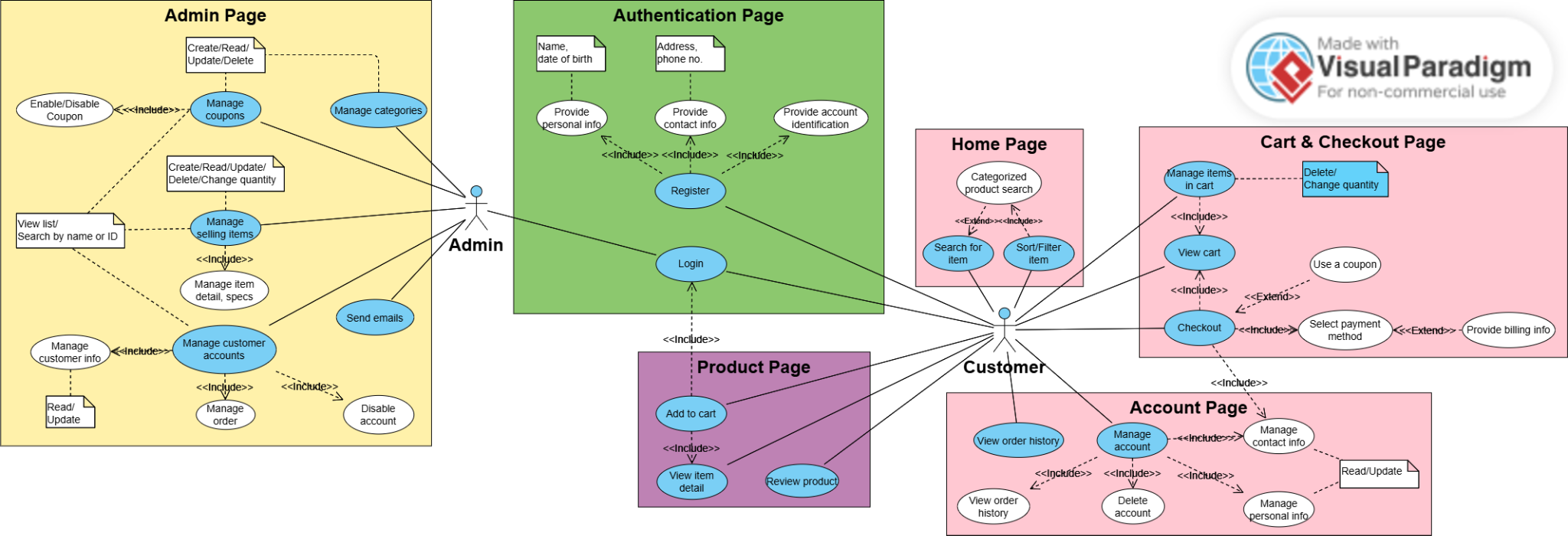
* **Scalability**: Ensure the system can handle increasing numbers of users, transactions, and product data without performance degradation.
* **Performance**: Provide fast page loading times and real-time updates to create a seamless shopping experience for users.
* **Security**: Implement robust security measures to protect sensitive data, such as user credentials, payment information, and order details.
* **Maintainability**: Ensure the system’s codebase is modular and well-structured to simplify future updates, bug fixes, and feature additions.
* **Compatibility**: Build a responsive platform accessible to various popular web browsers (Chrome, Firefox, Edge) with easy to use and navigate layouts, high responsiveness offering a smooth and quality experience.
* **Integration Capability**: Allow for seamless integration with third-party services, such as payment gateways (e.g., MoMo, E-Banking), shipping providers, and analytics tools.
* **User-Centric Design**: Ensure the system is intuitive and user-friendly, enhancing customer satisfaction and retention.

### **Constraints**

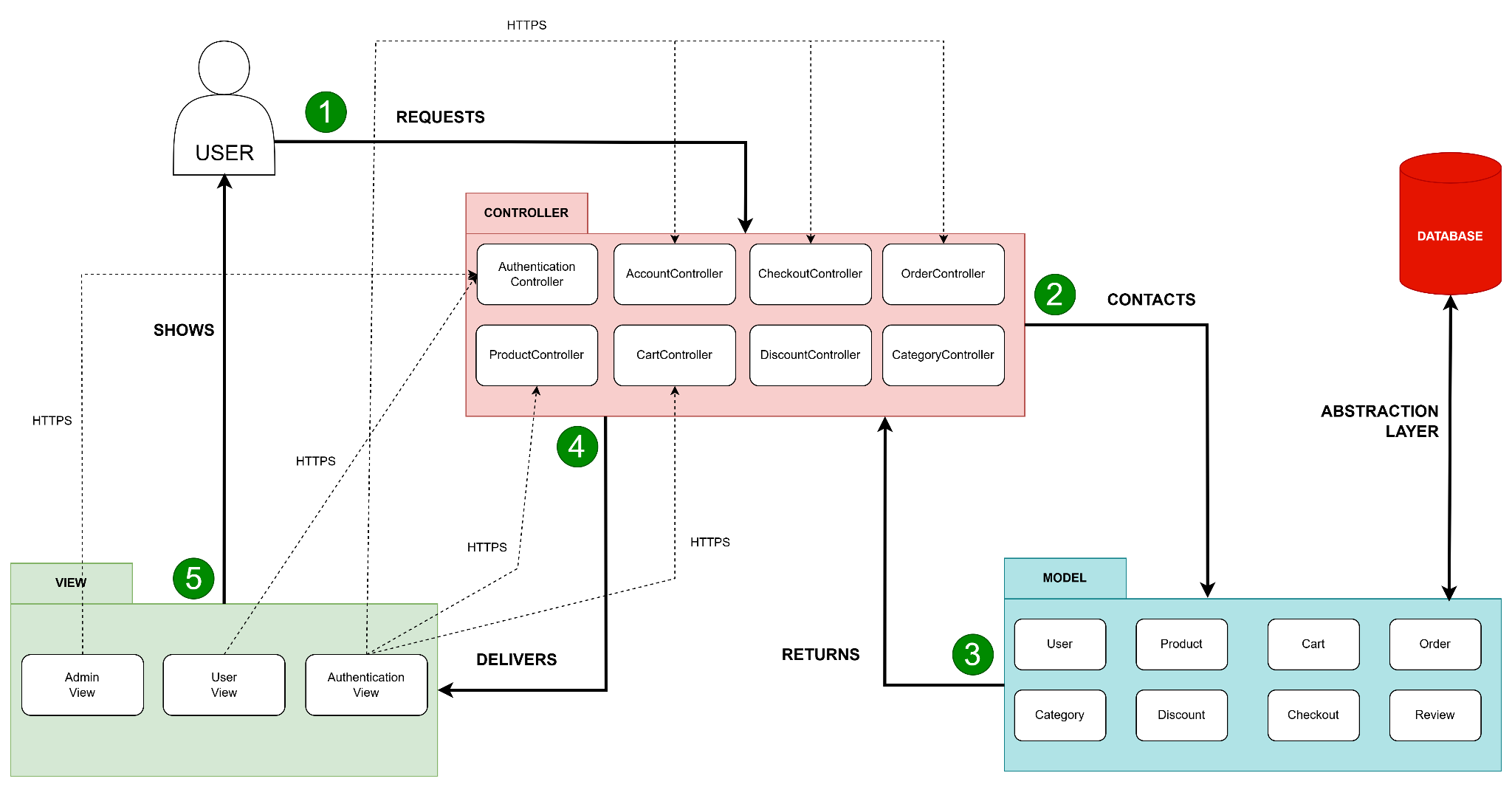
The architectural design is shaped by the following constraints:

* **Technology Stack**: The system is built using Node.js for the backend, Vue.js for the frontend, and MySQL for the database. These tools define the core environment and influence design choices.
* **API Dependencies**: The system may leverage additional APIs and frameworks, such as Express.js for building the backend and Axios for handling HTTP requests. These dependencies should be compatible with the existing stack.
* **Data Consistency**: MySQL is used for data storage, requiring careful schema design and efficient queries to maintain consistency and performance under high loads.
* **Programming Languages:** HTML, CSS, JavaScript, TypeScript.
* **Security Compliance**: The system must adhere to industry standards such as HTTPS, GDPR (if applicable), and PCI DSS for payment handling.
* **Responsiveness:** The system must be able to process users’ requests and load in less than 10 seconds.
* **Optimization:** The system total RAM cost must not exceed 1GB.

# Use-Case Model



# Logical View



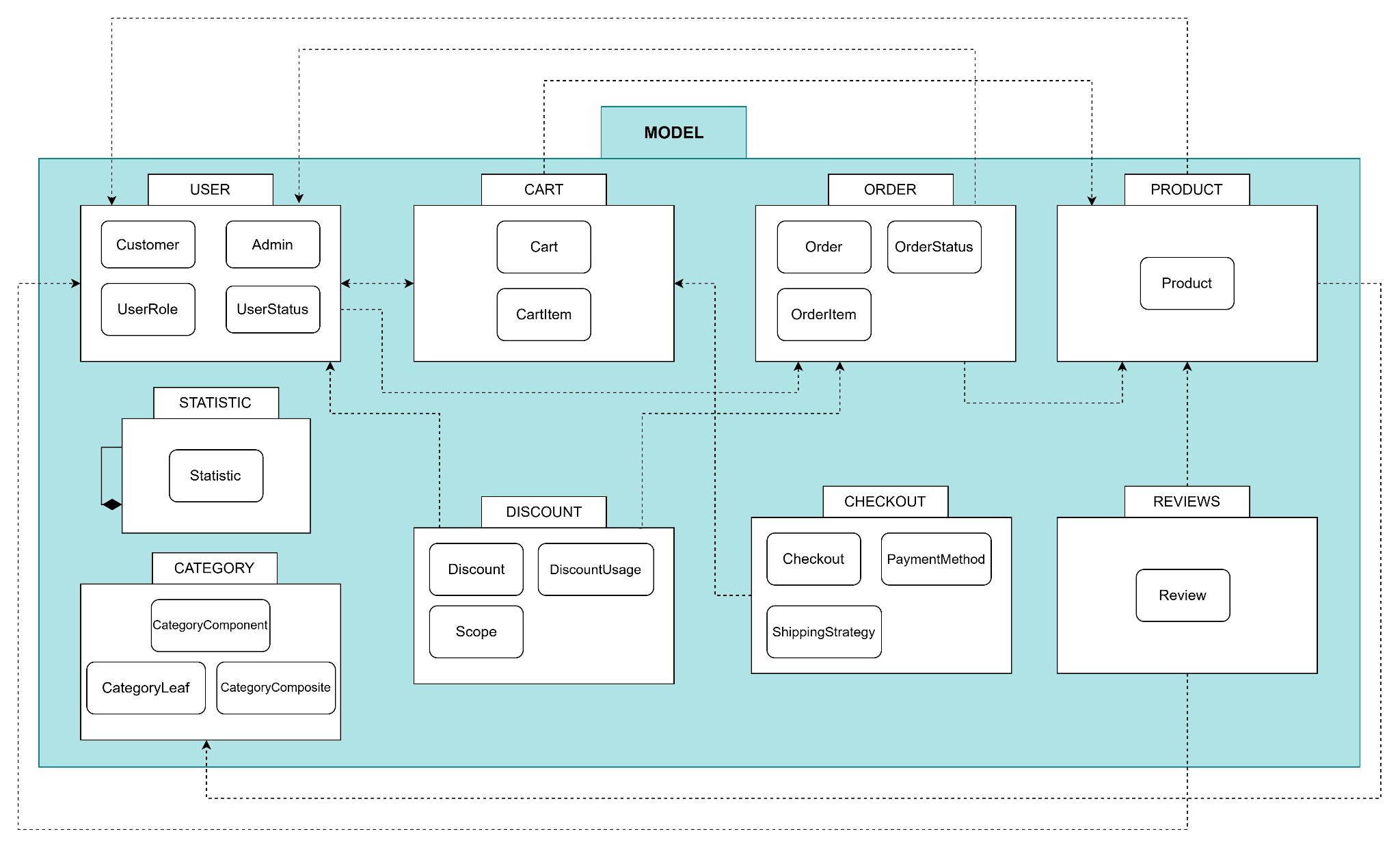
**Insight about this Model - View - Component diagram:**

* **Model :** The Model component in the diagram demonstrates the data and business logic of the website. It is responsible for managing the website data, processing, and responding to requests for information from other components, such as the View and the Controller.
* **View** : Displays the data from the Model to the user and sends user inputs to the Controller.
* **Controller** : Controller acts as an intermediary between the Model and the View. It handles user input and updates the Model accordingly and updates the View to reflect changes in the Model.

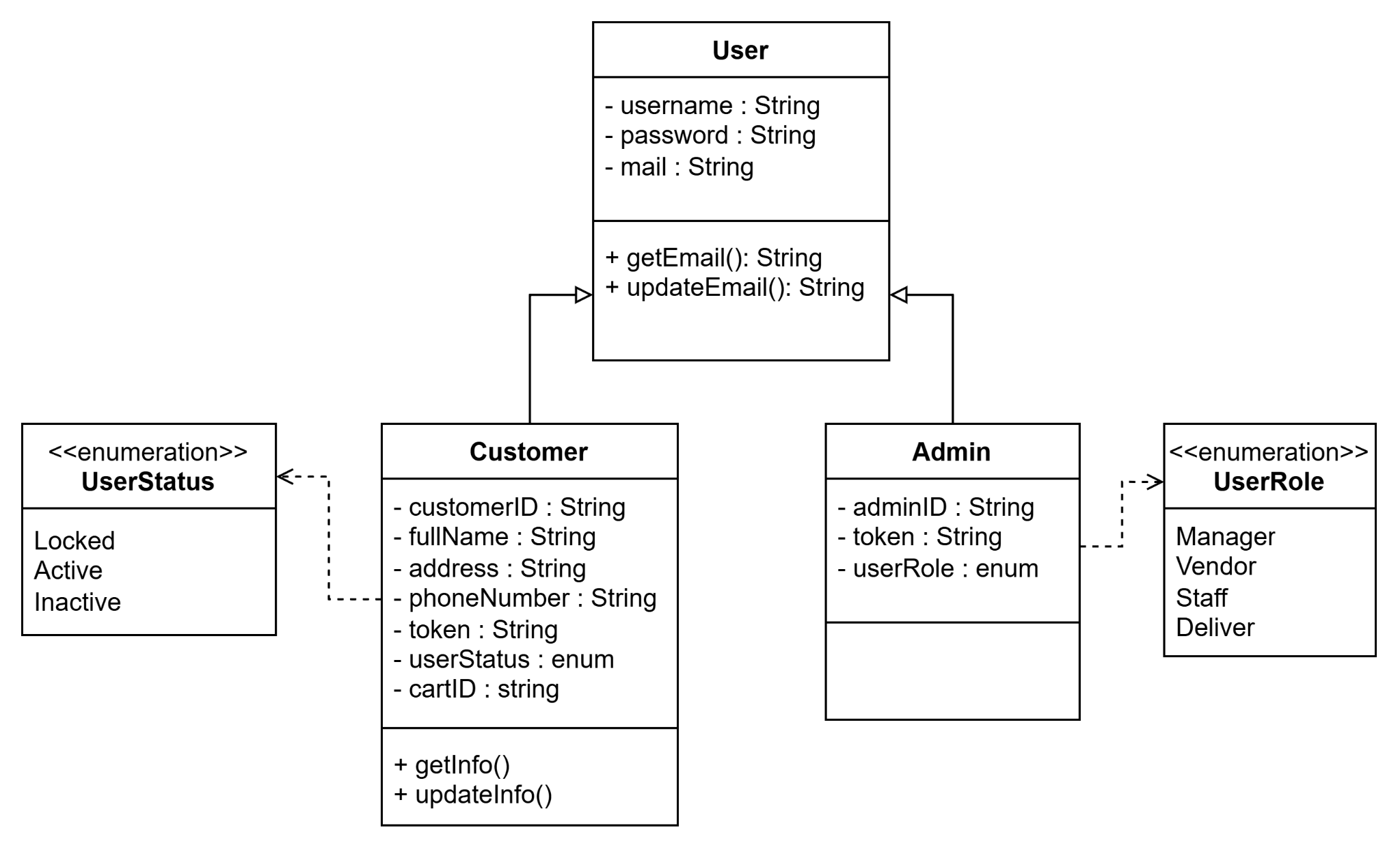
**Communication between the components:**

* **User Interaction with View:** The user interacts with the View, such as clicking buttons on the website.
* **View Receives User Input:** The View receives the user input and forwards it to the Controller.
* **Controller Processes User Input:** The Controller receives the user input from the View. It interprets the input, performs any necessary operations and decides how to respond.
* **Controller Updates Model:** The Controller updates the Model based on the user input or application logic.
* **Model Notifies View of Changes:** If the Model changes, it notifies the View.
* **View Requests Data from Model:** The View requests data from the Model to update its display.
* **Controller Updates View:** The Controller updates the View based on the changes in the Model or in response to user input.
* **View Renders Updated UI:** The View renders the updated UI based on the changes made by the Controller.

## Model

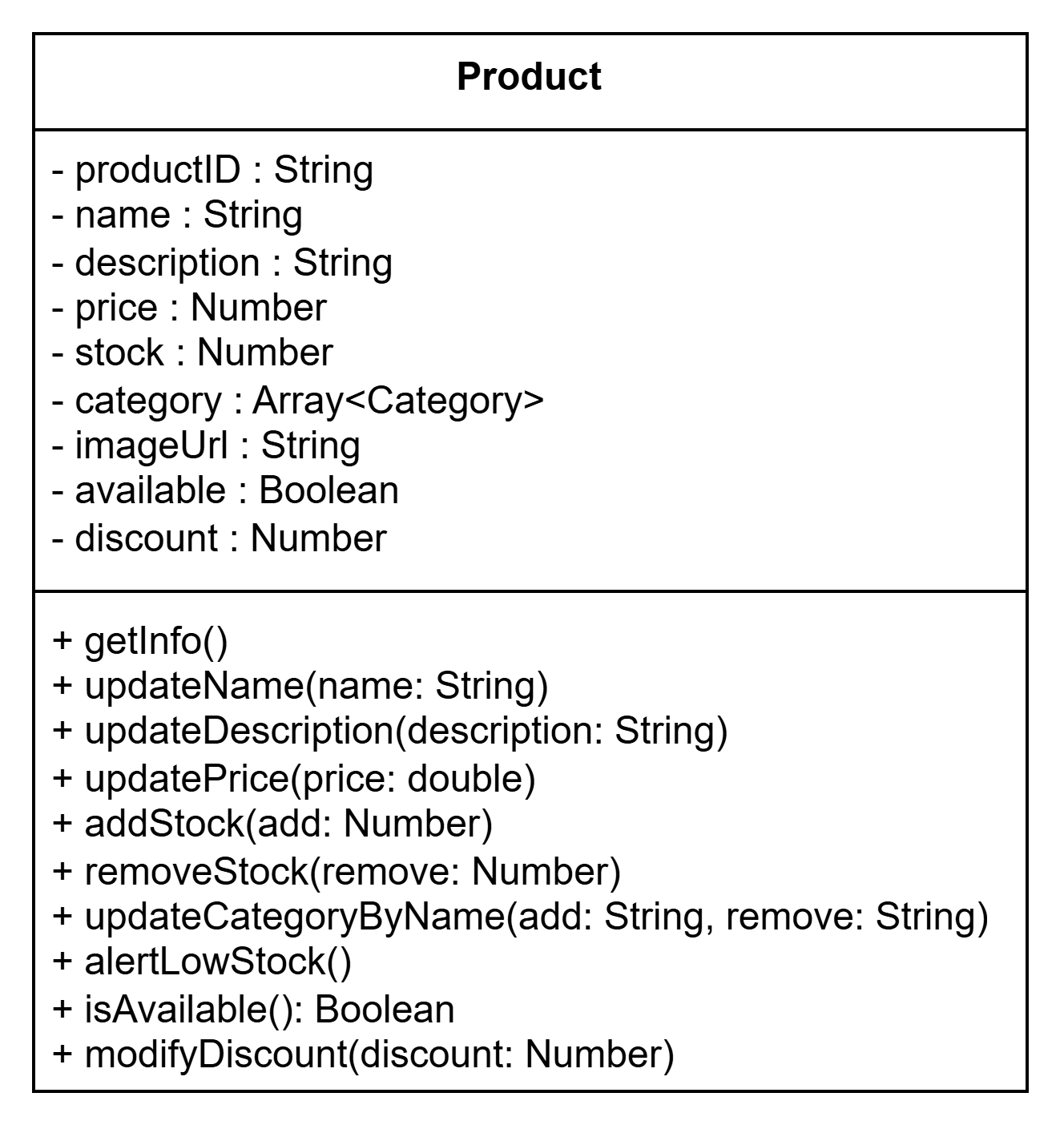


### **Component: User**

* **Description:** 
  + The **User** component represents a generic user in the system. It only contains the bare minimum attributes that a yet-to-identified user needs. Any user can see and change their email address.
  + After logging in, the user will be identified as a customer or an administrator, gaining a token verifying their access to the system.
  + A customer will have typical attributes of an online customer such as name, address, phone number, etc.
  + The customer will have one status attribute describing their account status, whilst the admin will have a role attribute specifying their account role and associated privileges.
* **Diagram:**  
  

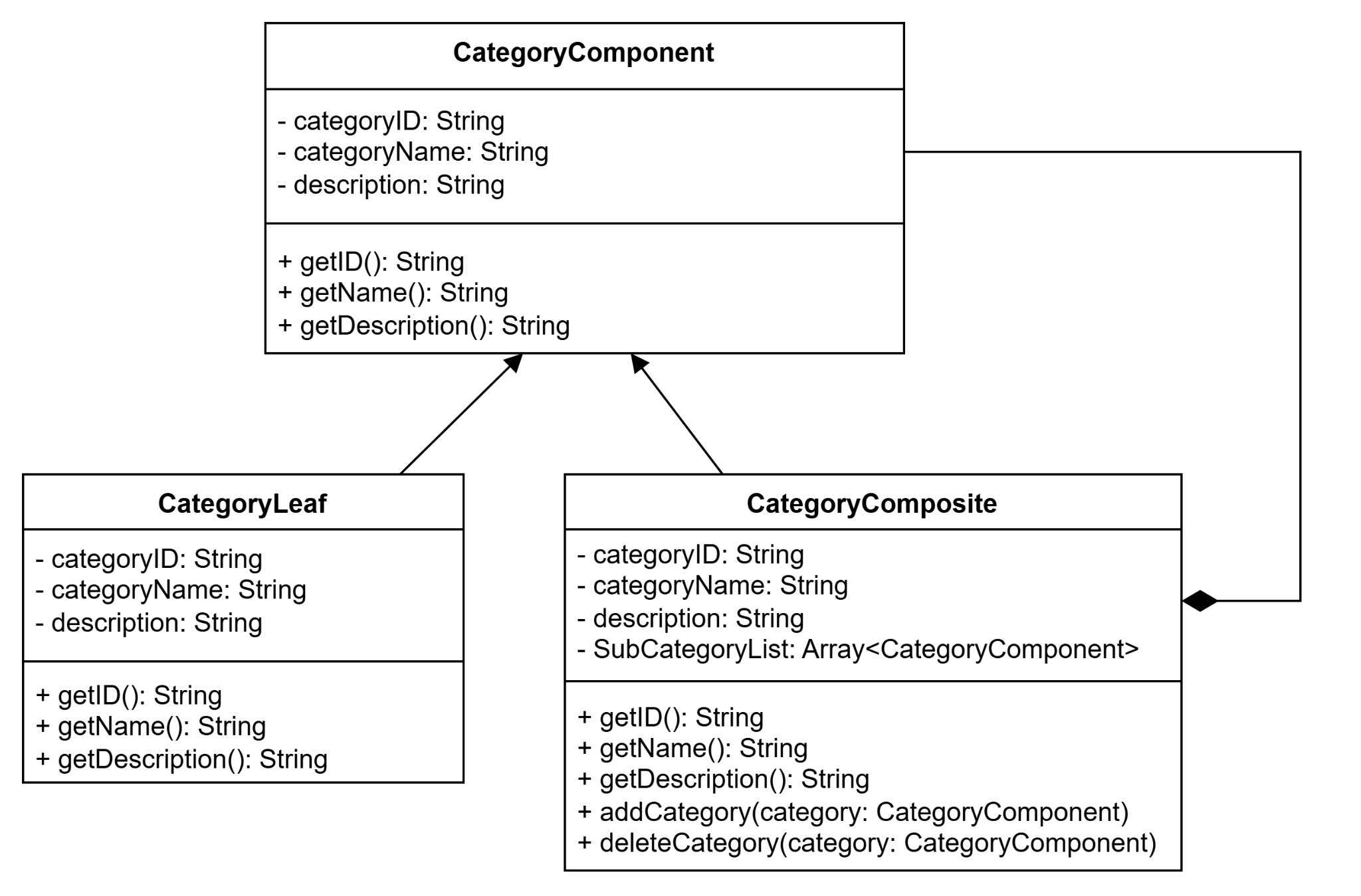
### **Component: Product**

* **Description:** The **Product** component represents an individual item available in the system, providing necessary information for customers and other system functionalities. This component serves as the foundation for managing the catalog of items in an e-commerce platform or inventory system.  
  (Note: The discount attributes is used for discounts that apply directly onto the Product price, different from the Discount class)
* **Diagram:**

****

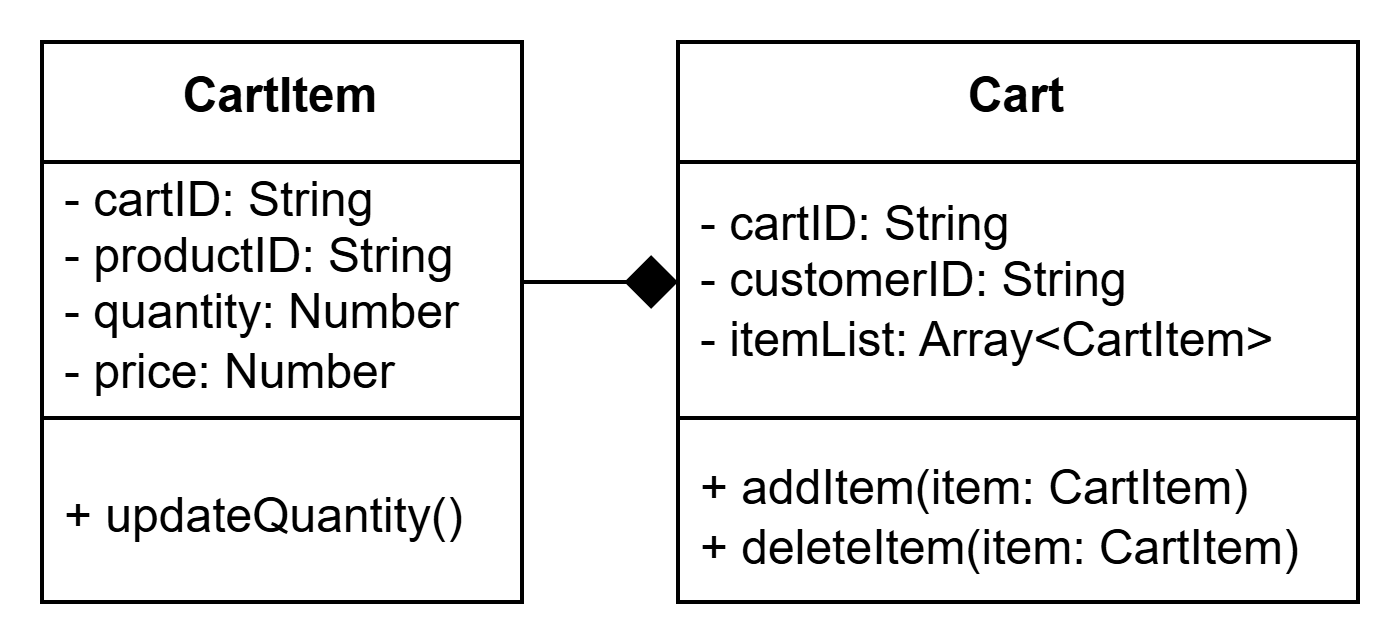
### **Component: Category**

* **Description:** The **Category** component is designed following the Composite Design Pattern. It allows organizing products into a flexible hierarchy, enabling:
  + Efficient navigation through categories and sub-categories.
  + Dynamic operations such as adding/removing categories or products.
  + Searching/Filtering products.
* **Diagram:**

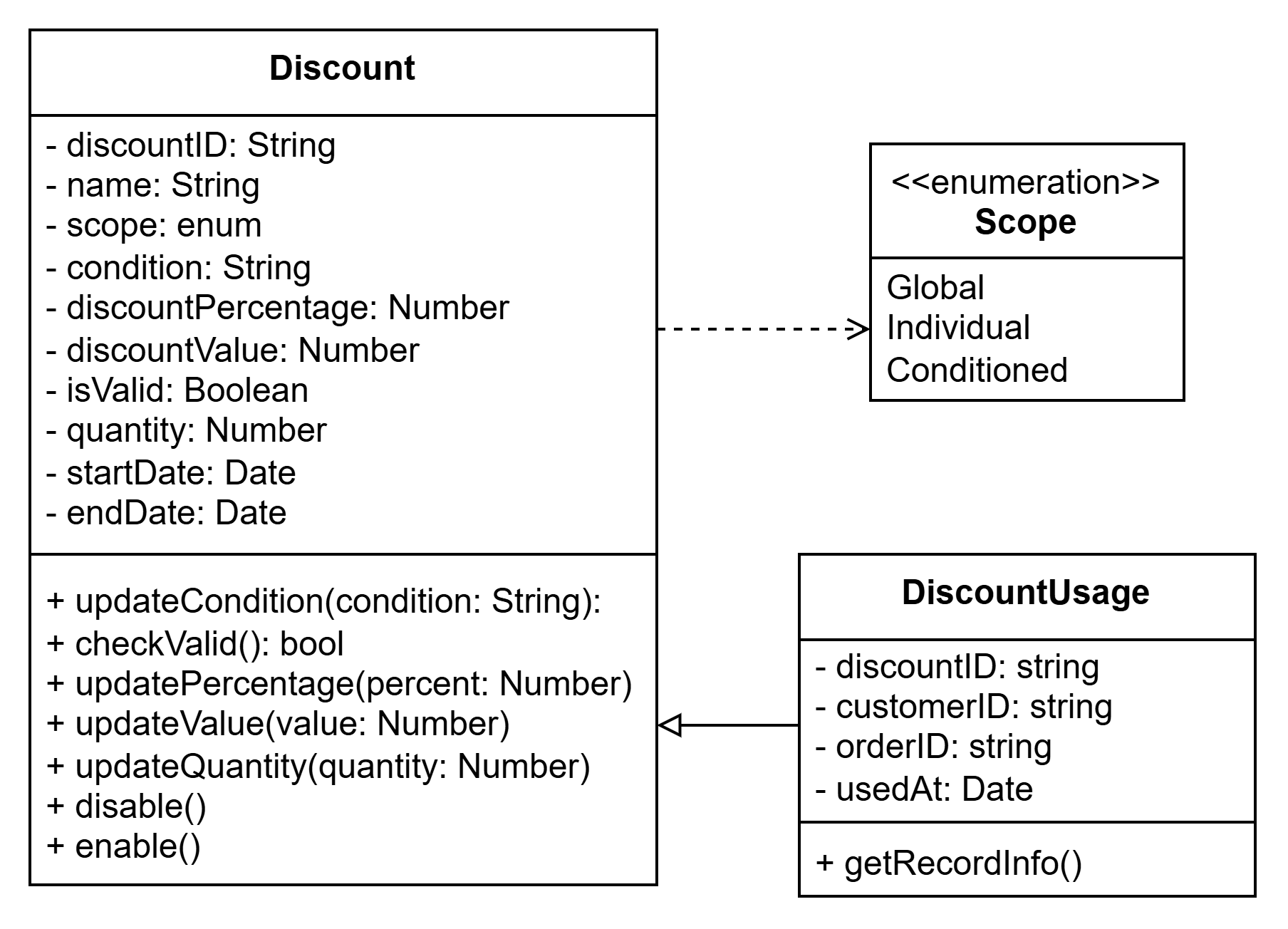
****

### **Component: Cart**

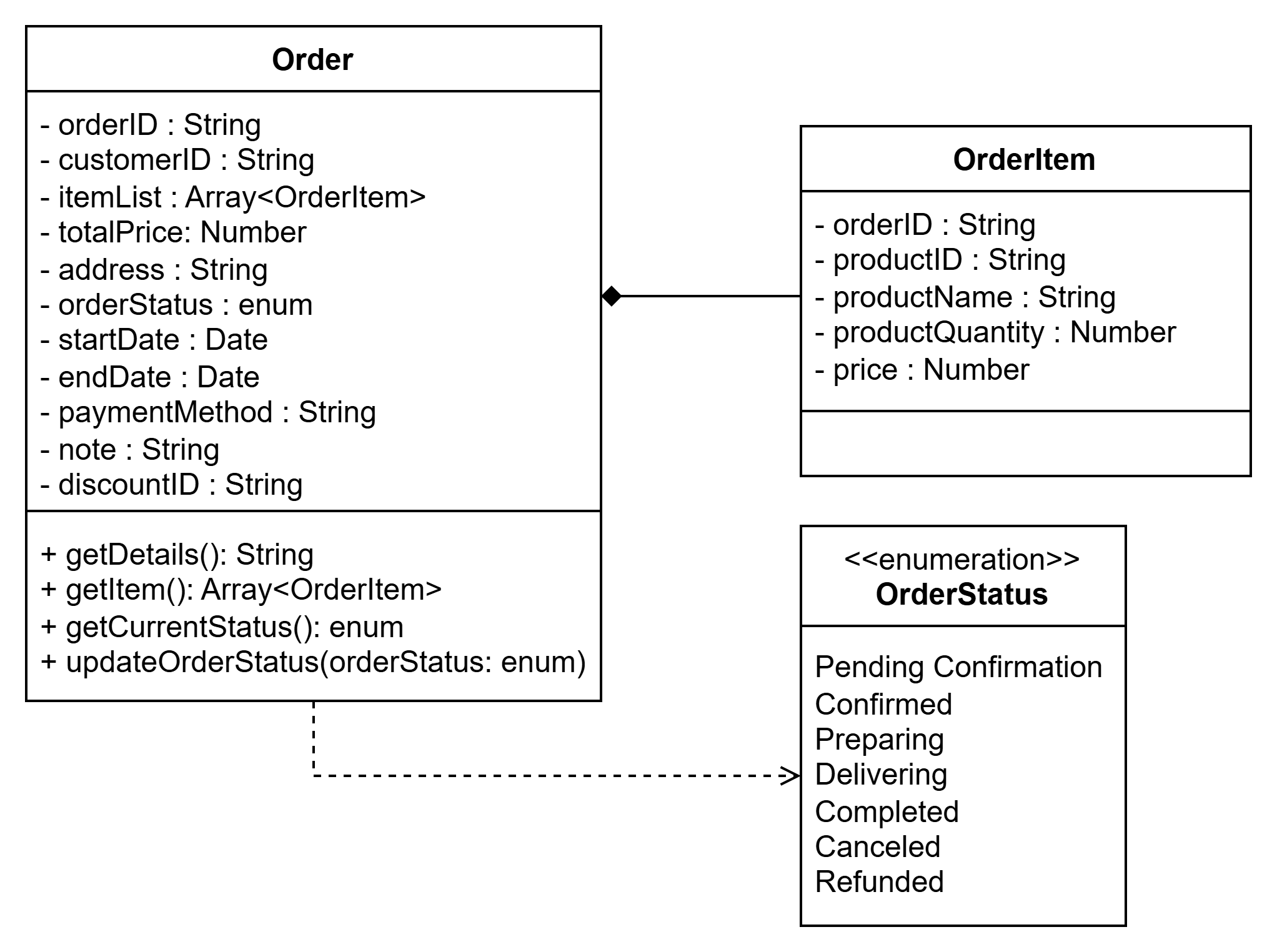
* **Description:** The **Cart** component represents a virtual shopping cart that manages the items selected by the user during the shopping process. It consists of two key parts:
  + **CartItem:** Each CartItem represents an individual item in the cart. It only contains details identifying the product, such as its ID, name, quantity, and price. This subcomponent enables the addition, removal, or modification of individual items within the cart.
  + **Cart:** The Cart component aggregates multiple CartItems to provide a complete view of the user's shopping session. It manages the total cost, tracks the number of items, and interacts with other systems. The Cart ensures consistency by validating stock availability and calculating the total dynamically as items are added or removed.
* **Diagram:**

****

### **Component: Discount**

* **Description:** 
  + The **discount** component manages discount codes in the shopping system.
  + Aside from necessary information, discounts have a **Scope** attribute to determine whether the discount codes are limited globally or given to each existing customer account or only permitted to be used under specific conditions.
  + There will be multiple **DiscountUsage** stored in the database each time a customer uses a discount code corresponding to an existing Discount.
* **Diagram:**  
  

### **Component: Order**

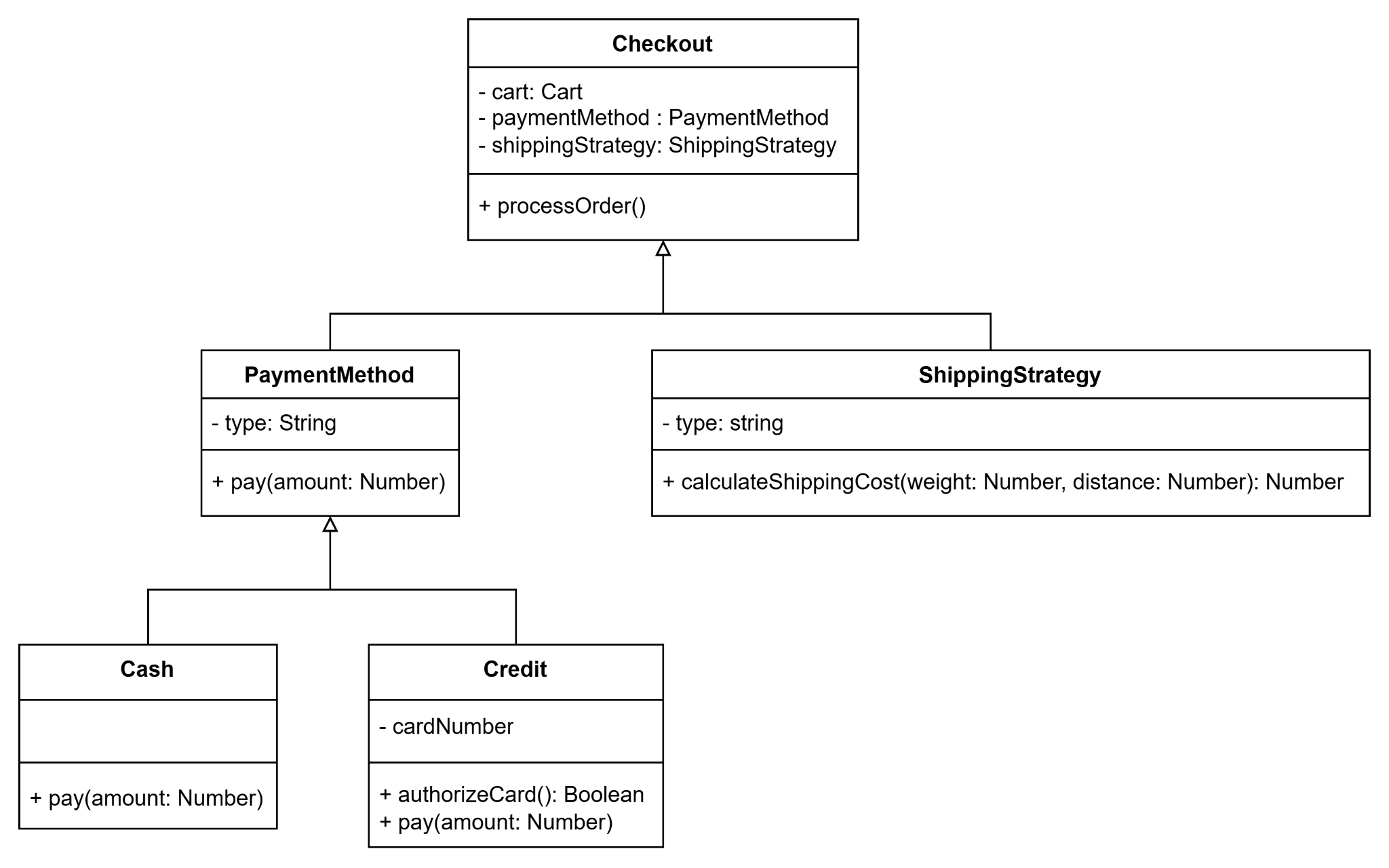
* **Description:** 
  + The **Order** component describing a customer’s order.
  + Consists of basic necessary attributes such as the order’s ID, delivery address, list of items in that order, the order start and end date, etc.
  + This class has an attribute specifying the order status. Once an order has either been marked as “Completed”, “Canceled” or “Refunded”, that order is archived and no further changes permitted.
  + Purchasing history (or order history) will list all orders that that customer has, but only orders marked as “Completed” will count towards total money spent.
* **Diagram:**  
  

### **Component: Statistic**

* Description: A singleton class upon creation will copy all existing orders currently on the system into itself, letting the admin see the statistics of completed purchases for analyzing purposes.
* Diagram:  
  

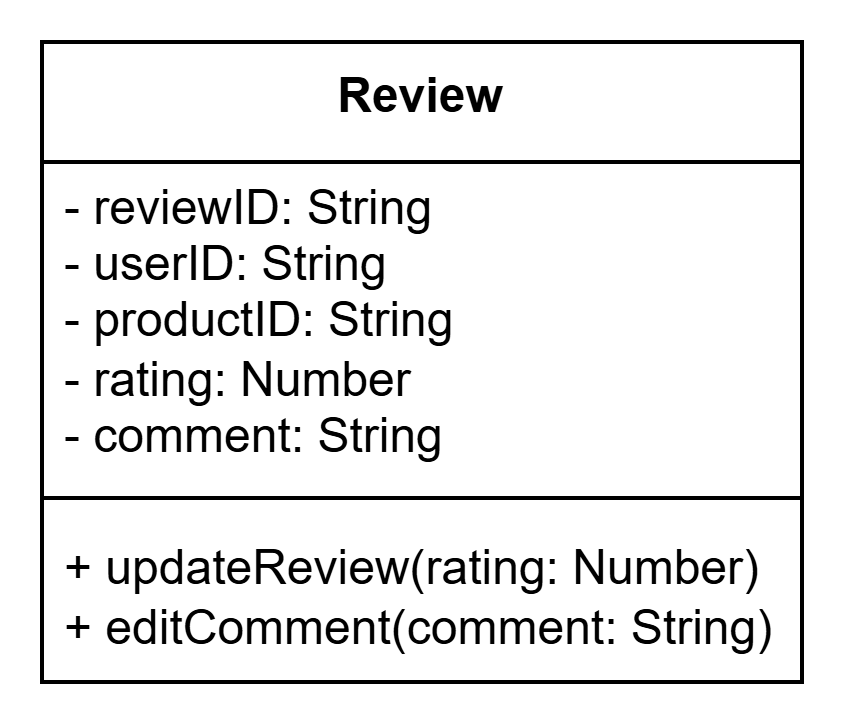
### **Component: Checkout**

* **Description:** The **Checkout** component consists of two main subcomponents: PaymentMethod and ShippingStrategy.
  + **PaymentMethod**: This subcomponent handles the payment options available to the customer during the checkout process. It provides various payment methods such as credit cards or cash.
  + **ShippingStrategy**: This subcomponent manages the shipping options for the order. It offers different shipping strategies. **ShippingStrategy** enables the user to choose an appropriate shipping method, such as express delivery or standard shipping, according to their preferences.
* **Diagram:**



### Component: Review

* **Description:** The review component collects and handles customer’s feedback which includes reviews and rating for products. This component helps to improve the customer's shopping experience.
* **Diagram:**

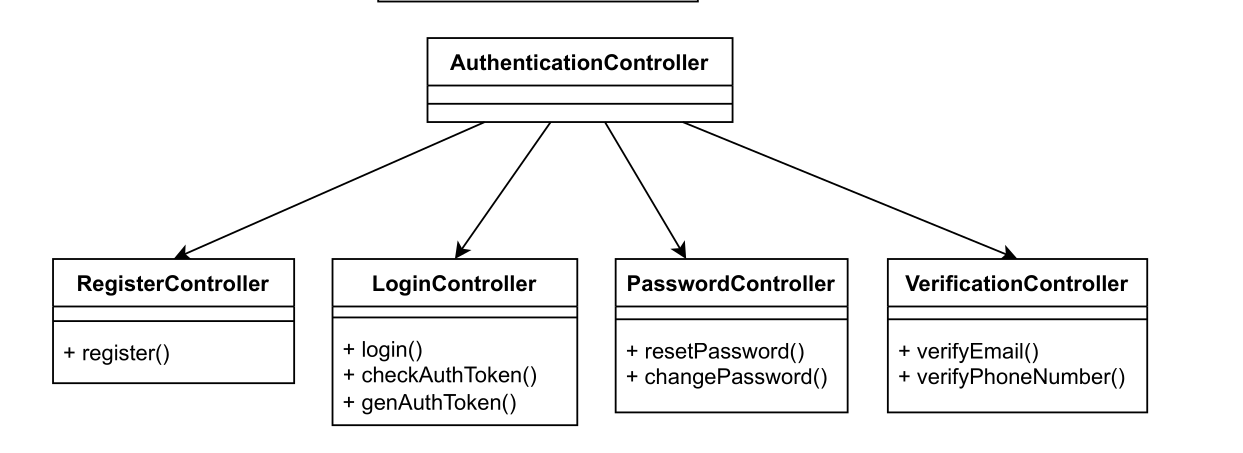


## Controller

### Component: Authentication Controller

Handles the authentication processes of the web application, ensuring secure user login and registration.

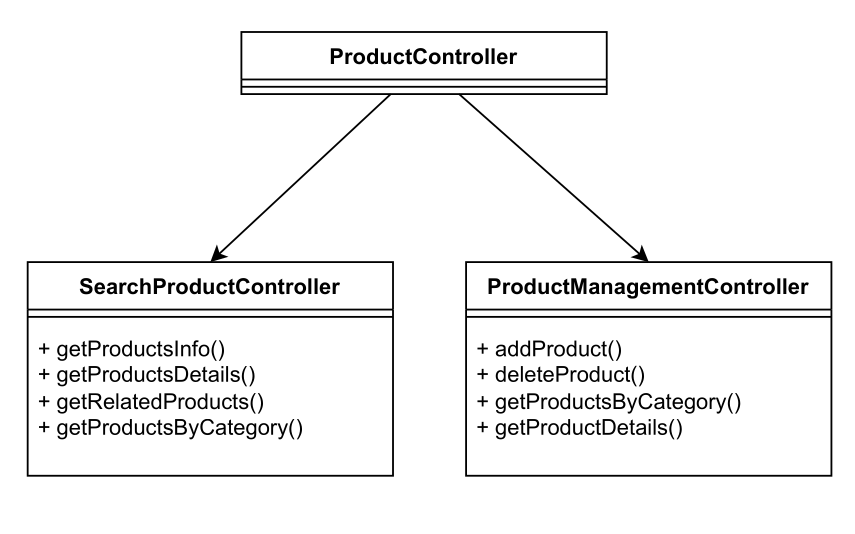
* **RegisterController**: Manages user sign-ups, including validation and saving user information.
* **LoginController**: Facilitates user login and session creation.
* **PasswordController**: Handles password reset and update functionalities.
* **VerificationController**: Deals with email/phone verification or two-factor authentication processes.



### Component: Product Controller

Manages the products available on the platform, including search and administrative features.

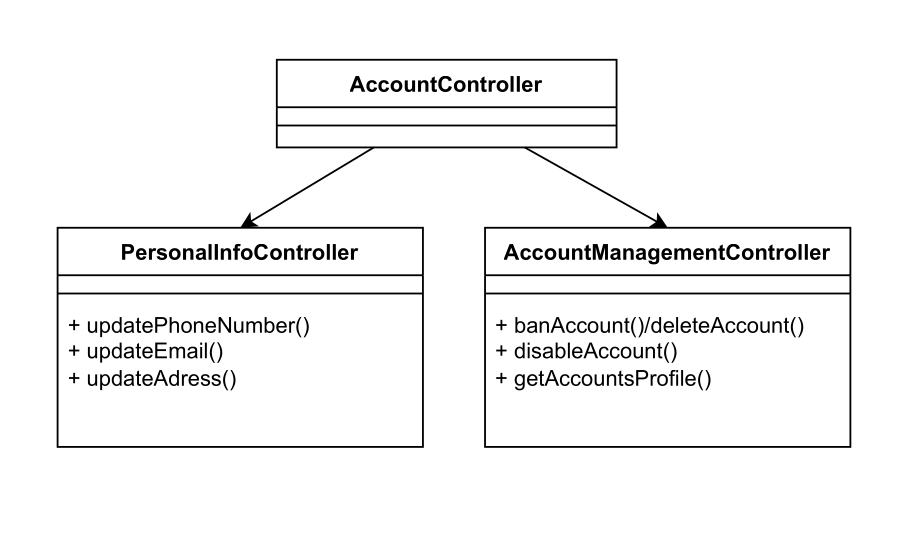
* **SearchProductController**: Provides functionality for searching and filtering products by attributes like name, price, or availability.
* **ProductManagementController**: Handles CRUD operations (Create, Read, Update, Delete) for products in the inventory.



### Component: Account Controller

Manages user accounts, focusing on personal information and administrative tasks.

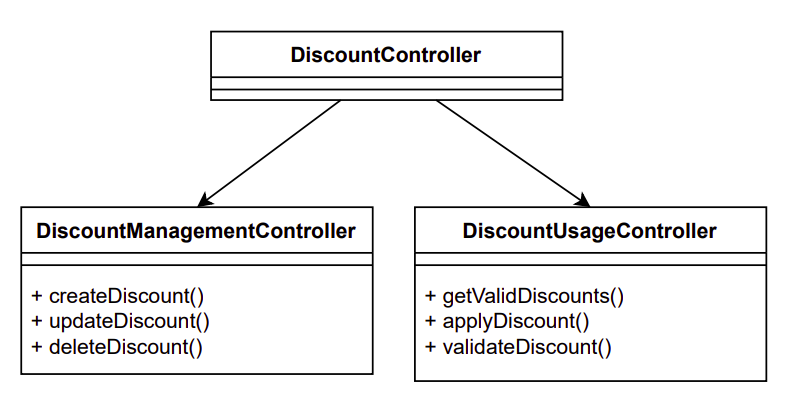
* **PersonalInfoController**: Updates and retrieves user profile details such as name, contact, or address.
* **AccountManagementController**: Handles account-related actions like deactivation, reactivation, or privilege changes.



### Component: Discount Controller

Manages discounts and their applications across products or orders.

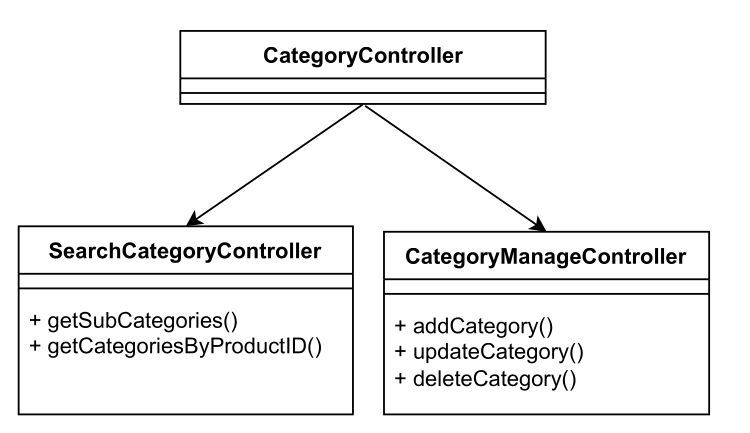
* **DiscountManagementController**: Administers the creation and management of discount codes or offers.
* **DiscountUsageController**: Tracks and validates the use of discounts during checkout.



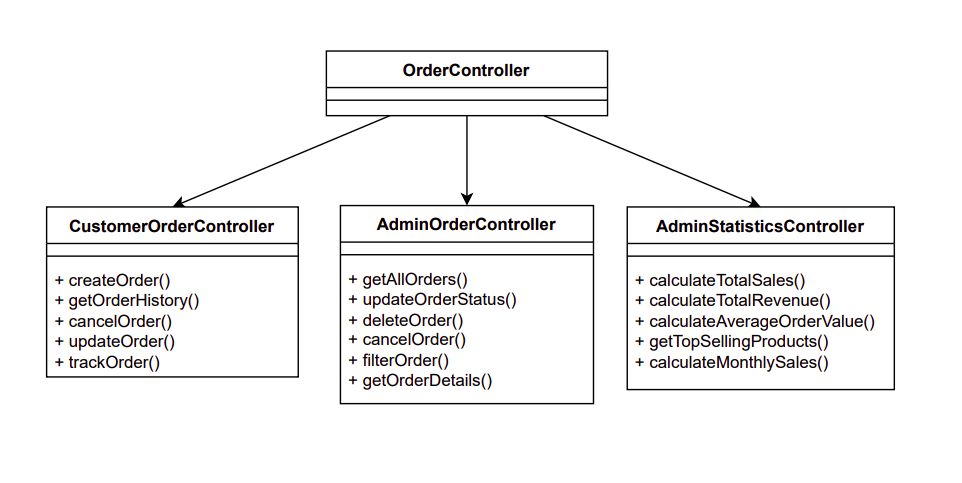
### Component: Category Controller

Organizes and manages product categories, allowing for efficient browsing and retrieval.

* **SearchCategoryController**: Enables users to search or filter categories based on keywords or criteria.
* **CategoryManageController**: Administer category structures, including adding, updating, or deleting categories.



### Component: Order Controller



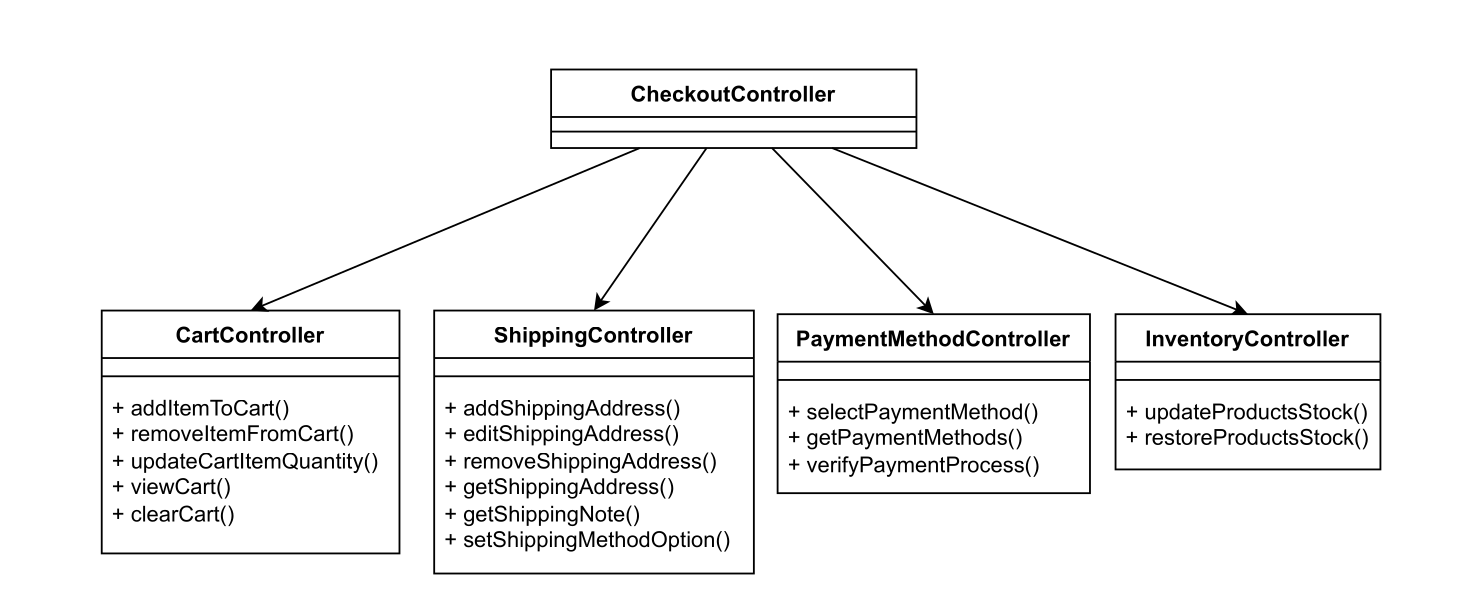
Manages all operations related to orders, including order creation, tracking, and administration.

* **CustomerOrderController**: Focuses on customer-facing order functionalities, such as placing orders, viewing order details, and tracking shipment status. It also enables customers to cancel orders when applicable. This controller will search sequentially and then collect all previous orders of the customer and create a history of previous orders/purchases.
* **AdminOrderController**: Provides administrators with tools to monitor and manage orders across the system. This includes viewing all orders, updating their status, handling returns
* **AdminStatisticsController:** Generates reports for analytics and operational oversight of selling statistics.

### Component: Checkout Controller

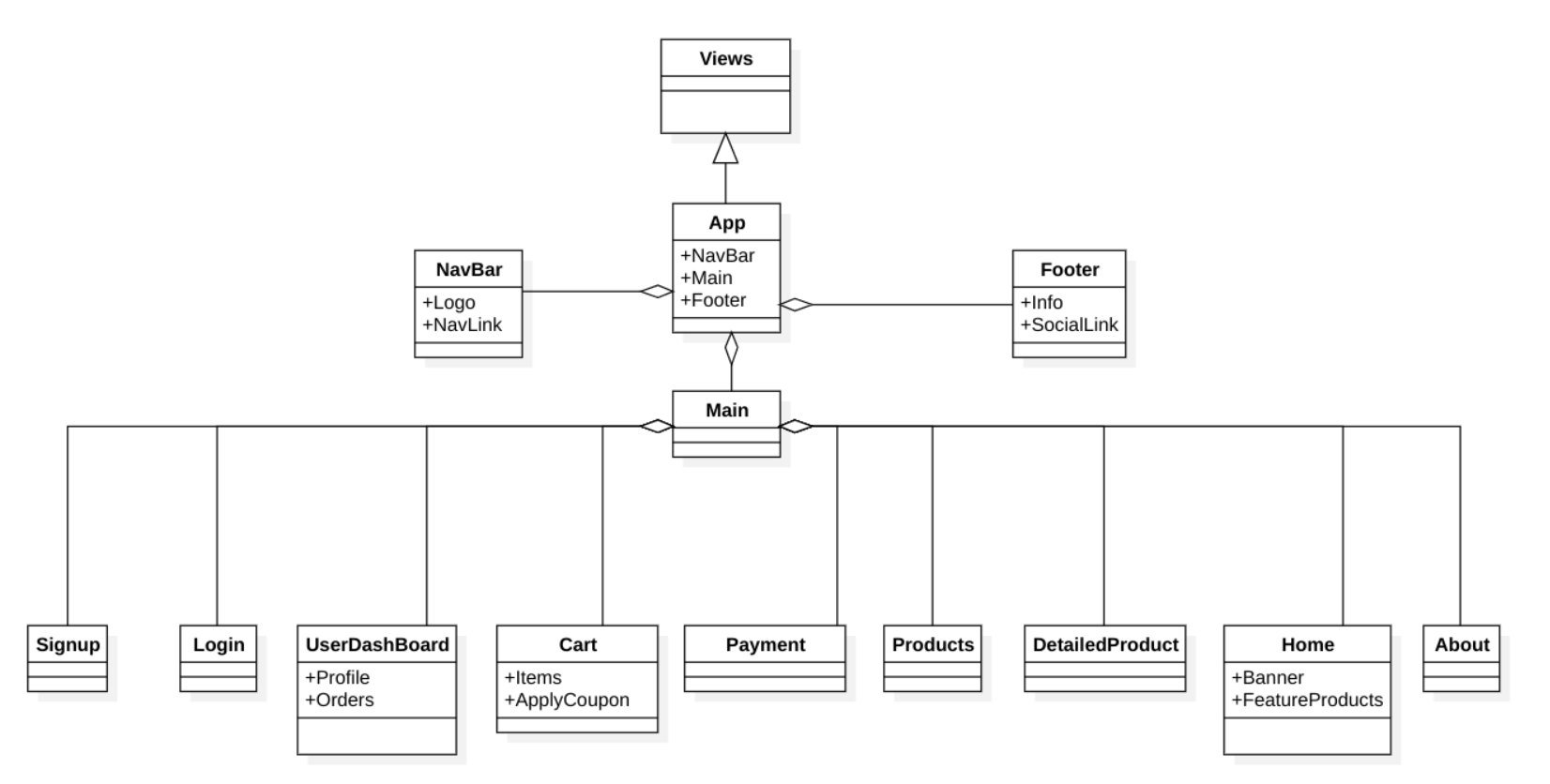
Oversees the purchasing process, ensuring a seamless transition from cart to order placement.

* **CartController**: Manages the shopping cart, allowing users to add, update, or remove items.
* **ShippingController**: Handles shipping details, including address management and delivery options.
* **PaymentMethodController**: Processes payments and manages payment methods like credit cards or wallets.
* **InventoryController**: Ensures product availability and updates inventory post-checkout.



## View

### General View



### Component: App

The root component that serves as the main container for the application.

* **NavBar**: Contains the top navigation component
* **Main**: Houses the main content area
* **Footer**: Contains the bottom footer component
* Manages routing

### Component: NavBar

Top navigation component providing main navigation controls.

* **Logo**: Displays website branding
* **NavLink**: Navigation menu items
* Handles responsive menu behavior
* Manages user navigation state

### Component: Footer

Bottom footer component containing site information and links.

* **Info**: Company and website information
* **SocialLink**: Social media links and icons
* Displays copyright, terms, and additional navigation

### Component: Main

Core content container managing the main view area.

* Handles routing between different views
* Manages content transitions
* Controls layout for primary content area

### Component: Home

Landing page component showcasing featured content.

* **Banner**: Hero section/promotional content
* **FeatureProducts**: Highlighted product showcase
* Manages homepage layout and content organization

### Component: Products

Product listing component displaying available items.

* Manages product grid/list views
* Handles product filtering and sorting
* Implements pagination/infinite scroll

### Component: DetailedProduct

Individual product view component.

* Displays comprehensive product information
* Manages product images and variants
* Handles "Add to Cart" functionality
* Shows related products and recommendations

### Component: Cart

Shopping cart management component.

* **Items**: List of cart items
* **ApplyCoupon**: Discount code functionality
* Manages quantity updates
* Calculates totals and subtotals
* Handles cart modifications

### Component: Payment

Payment processing component.

* Manages payment method selection
* Handles payment form validation
* Processes transactions
* Displays order summary

### Component: UserDashboard

User account management component.

* **Profile**: User information and settings
* **Orders**: Order history and tracking
* Manages user preferences
* Handles account updates

### Component: Login

User authentication component.

* Manages login form and validation
* Handles authentication errors
* Provides password recovery options
* Maintains session state

### Component: Signup

New user registration component.

* Manages registration form and validation
* Handles user data submission
* Implements verification process
* Provides onboarding flow

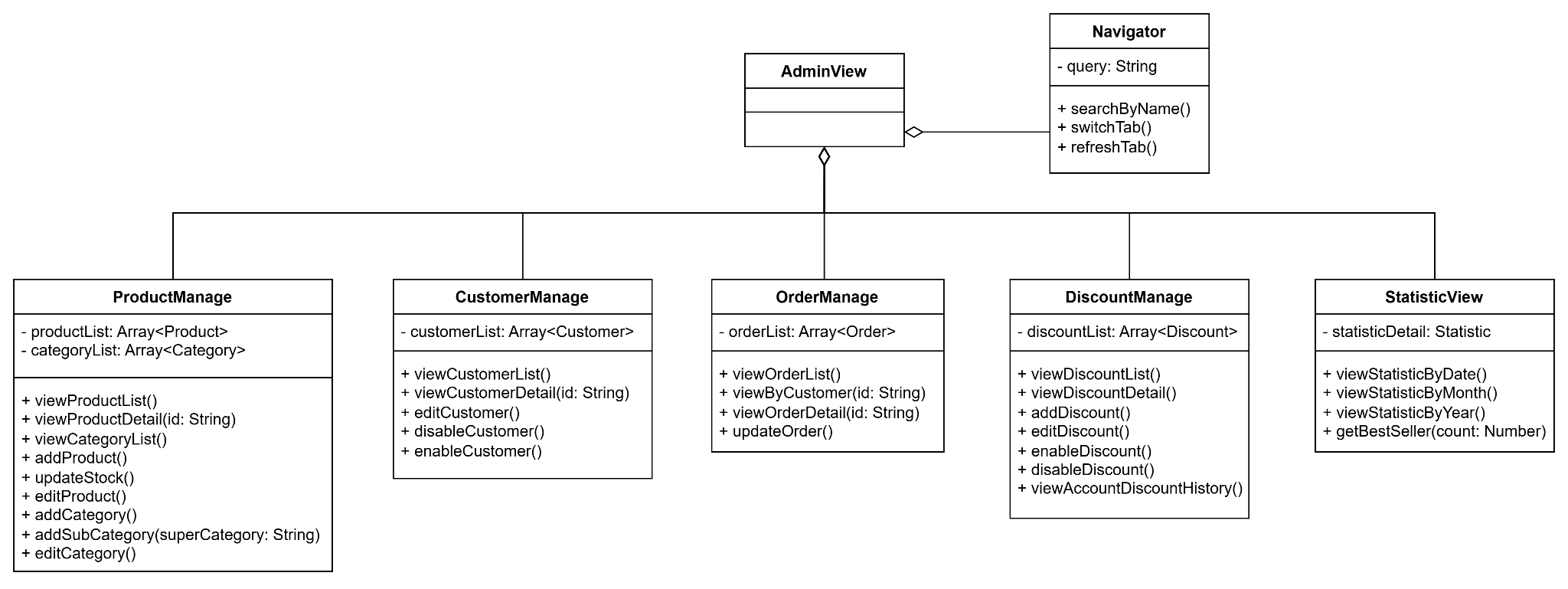
### Component: About

Group information component.

* Displays company history and mission
* Shows team information
* Presents company policies and values

## Admin View

Suppose the user login with the credential of an administrator, they will be led to Admin View.



### Component: Navigator

A universal component, as it is critical for central routing and navigating within the application.

* The admin can switch between tabs to manage different types of objects or entities.
* Can filter out unnecessary entities by searching.

### Component: ProductManage

Handles data of all **Product** *and* **Category** objects.

* View Products and Categories by list. Can also view detailed information of a Product object.
* Add and edit Products and Categories.

### Component: CustomerManage

Manages and views all customer accounts.

* View by list. Can also view an account’s details.
* Edit any account info.
* Enable/Disable any customer account.

### Component: OrderManage

Manages and views all orders, including completed orders that are archived.

* View by list. Can also view any order’s details.
* Edit/Update an ongoing order.
* Admin may view orders owned by a specific customer account.

### Component: DiscountManage

Manages and views all discounts (e.g. coupon).

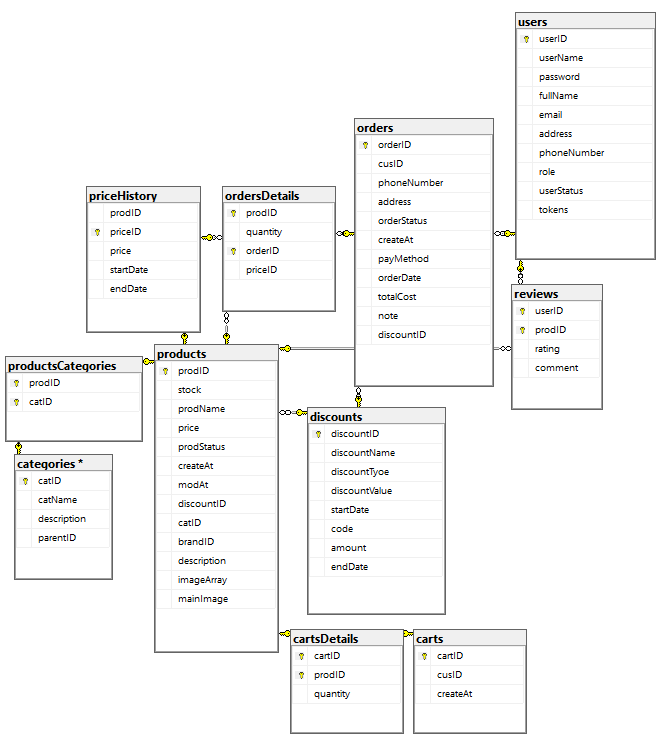
* View by list. Can also view any discount’s details.
* Add and edit any discount.
* Enable/Disable any discount.
* View a customer account discount usage history.

### Component: StatisticView

Displays analytics and statistics for the business.

* View profit or return by day/month/year.
* View a list of best selling products.

## Component: Database

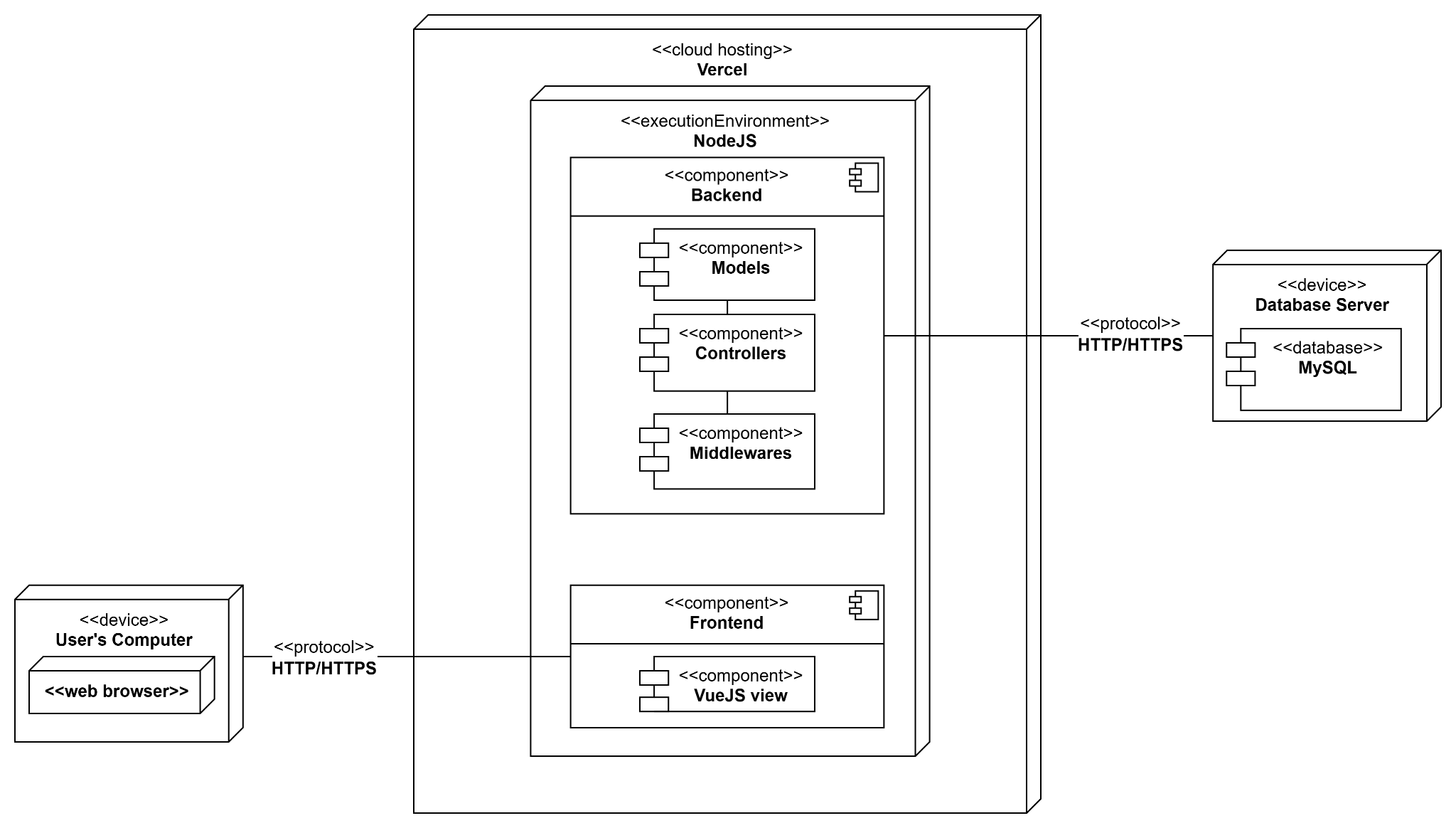
**

# Deployment

## Deployment Overview

* **Cloud Hosting (Vercel)**: The application is hosted on Vercel platform, ensuring scalability and ease of deployment for the web application.
* **Execution Environment (Node.js)**: The backend and frontend components run on a Node.js runtime environment, providing server-side JavaScript (and/or TypeScript) execution.
* **Backend**:
  + **Models**: Handles data representation and interaction with the database.
  + **Controllers**: Manages the business logic, processing incoming requests, and returning responses.
  + **Middlewares**: Includes intermediate processing tasks such as authentication, error handling, and request logging.
* **Frontend**:
  + **Vue.js View**: Represents the user interface (UI) built with Vue.js, a progressive JavaScript framework for building responsive frontends.
* **User’s Computer**:
  + Acts as the client device accessing the e-commerce application through a web browser.
  + Communication between the user’s browser and the application uses **HTTP/HTTPS**.
* **Database Server**:
  + A separate device hosts the **MySQL** database, which stores product, user, order, and other data.
  + Communication between backend and database server uses **HTTP/HTTPS**.

## Deployment Diagram



# Implementation View

## Implementation View Overview

**Backend (BE/)**

* **config/**: Stores configuration files, such as database credentials or environment variables.
* **controllers/**: Implements the core logic for handling HTTP requests and orchestrating services.
* **helpers/**: Contains utility functions used across the backend, promoting code reuse.
* **middlewares/**: Hosts middleware logic, such as authentication, validation, or error handling, to process requests.
* **models/**: Represents the database schema and handles interactions with the database.
* **routes/**: Defines the API endpoints exposed by the backend.

**Frontend (FE/)**

* **public/**: Contains static files (e.g., images, icons) that are directly served without processing.
* **src/**: Core application code, with component-based architecture:
  + **assets/**: Static resources such as images or stylesheets.
  + **components/**: Modular Vue.js components that make up the UI.
  + **router/**: Manages application routes, navigation between different pages.
  + **services/**: API calls and communication with the backend.
  + **views/**: Represents individual page templates or screens.
  + **App.vue**: The root Vue.js component.
  + **helper.js**: Provides utility functions..
  + **main.js**: Main entry point for the Vue.js application.
* **Configuration Files**:
  + **vite.config.js**.
  + **eslint.config.js**

**Other Files**:

* **knexfile.js**: Configuration file for the Knex.js query builder, used for database interaction.
* **package.json**: Defines project dependencies and scripts for both backend and frontend.
* **server.js**: Entry point for starting the backend server.
* **package-lock.json**: Ensures consistent dependency versions across environments.

## Implementation View Structure

**PotPal/**

**├── BE/**

**│ ├── config/**

**│ ├── controllers/**

**│ ├── helpers/**

**│ ├── middlewares/**

**│ ├── models/**

**│ ├── routes/**

**├── FE/**

**│ ├── public/**

**│ ├── src/**

**│ │ ├── assets/**

**│ │ ├── components/**

**│ │ ├── router/**

**│ │ ├── services/**

**│ │ ├── store/**

**│ │ ├── views/**

**│ │ ├── App.vue**

**│ │ ├── helper.js**

**│ │ └── main.js**

**│ ├── index.html  
│ ├── eslint.config.js  
│ ├── vite.config.js**

**│ └── README.md  
├── knexfile.js**

**├── package.json**

**├── package-lock.json**

**└── server.js**