Giadinhtiendung e-commerce website

Software architect document

Version 0.3

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 30/11/2023 | 0.1 | Initial writings | Lê Quang Trường |
| 30/11/2003 | 0.1 | design use-case model | Nguyễn Gia Khánh |
| 02/11/2023 | 0.2 | Complete writing | Nguyễn Văn Tuấn Kiệt |
| 02/11/2023 | 0.2 | Design Logical View | Bùi Nguyên Hanh |
| 02/11/2023 | 0.2 | Organize and formalize the document | Trần Ngọc Trường Thịnh |
| 02/11/2023 | 0.2 | Support Implementation view | Nguyễn Gia Khánh |
| 02/11/2023 | 0.2 | Complete writing | Lê Quang Trường |
| 12/12/2023 | 0.3 | Initiated folder structure diagram. | Lê Quang Trường |
| 15/12/2023 | 0.3 | Completed deployment diagram | All members |
| 15/12/2023 | 0.3 | Designed web server folder structure | Trần Ngọc Trường Thịnh |
| 15/12/2023 | 0.3 | Designed/Wrote description for some packages in Logical view | Bùi Nguyên Hanh |
| 15/12/2023 | 0.3 | Designed packages of User Client Logical View | Nguyễn Gia Khánh |
| 15/12/2023 | 0.3 | Designed packages of Admin Logical View | Nguyễn Văn Tuấn Kiệt |

Table of Contents

1. Introduction 4

2. Architectural Goals and Constraints 4

3. Use-Case Model 4

4. Logical View 4

5. Deployment 4

6. Implementation View 4

Software Architecture Document

# Introduction

Giadinhtiendung web based application, providing an online trading platform for vendors and users to sell and buy household products in a fast and convenient way.

Purpose:

* This document serves as a comprehensive guide to the architecture of the Giadinhtiendung system. Delving into various aspects of the architecture.

Scope:

* The scope of this SAD is to explain the architecture of the system. This document describes many aspects of the system architecture,in which each element takes an important role in constructing a stable system..

Definitions, Acronyms and Abbreviations:

* OOP: Object oriented programming

Overview:

* This document provides a deep understanding of the system architecture. Future sections cover the details of the 3 main views: (logical, deployment and implementation).

# Architectural Goals and Constraints

* **Server side:** The website database is hosted using Plannetscale. All communication with clients has to comply with public HTTPS,TCP/IP communication protocol standards. The website is a web based application so that it can be accessed using any regular browser such as Chrome, Microsoft Edge,...
* **Client side:** Users will be able to access ‘Giadinhtiendung e-commerce website’ only online. Clients/users are expected to use a modern web browser such as Edge, Google Chrome to get full user experience
* **Security:** On both client and server side, security is handled by NextJs built-in protection features, database connection is protected by prismadb.
* **Persistence:** Data will be stored in a central database, hosted on Plannetscale. The database is designed to meet the BCNF standard, assure minimal data duplication. All transactions will be handled with high consistency automatically by using the Prisma interface.
* **Reliability/ Availability:** System will have many undergone tests every development sprint to make sure that this will show strong reliability in practice. Furthermore, the server is designed to handle all requests separately and retrieve data from the server with well handled concurrent transactions.
* **Performance:** The system responds to any request, mainly fetching data from the database. System performance depends on the hardware specifications, network connectivity on the client side. The server’s performance is ensured by Vercel and Plannetscale scaling abilities. However, real performance can only be determined after testing and deploying the system.
* **Portability and Reuse:** In order to maintain code reusability, all functional components are well structured and organized, OOP standards are also strictly adhered, combining with NextJs well-known compatibility with various deployment services.
* **Development tools:**

Programming: Visual Code IDE, Github

Api and Front-end: Nextjs, with several utility packages.

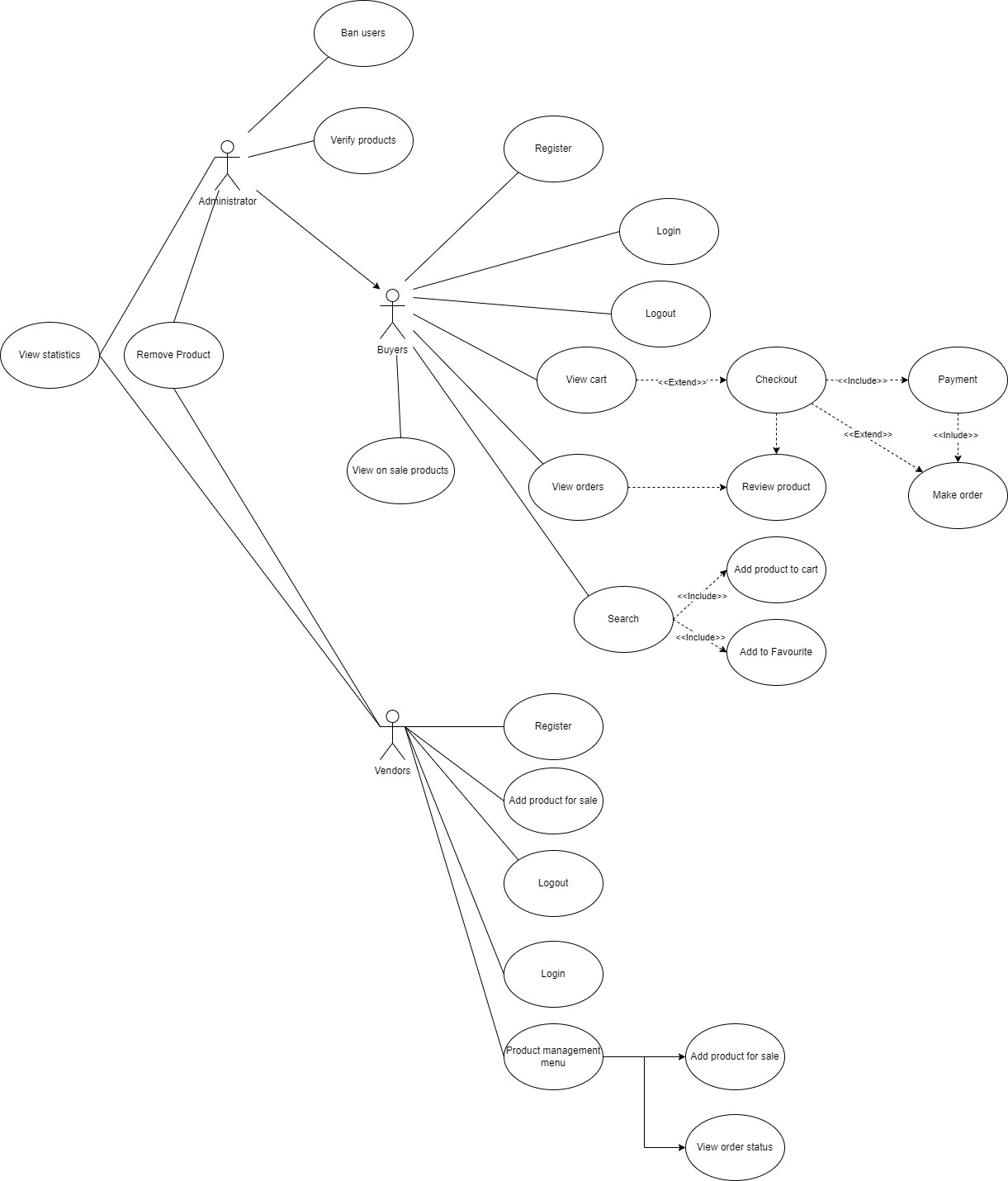
Database: Plannetscale

Diagram: DRAW.IO

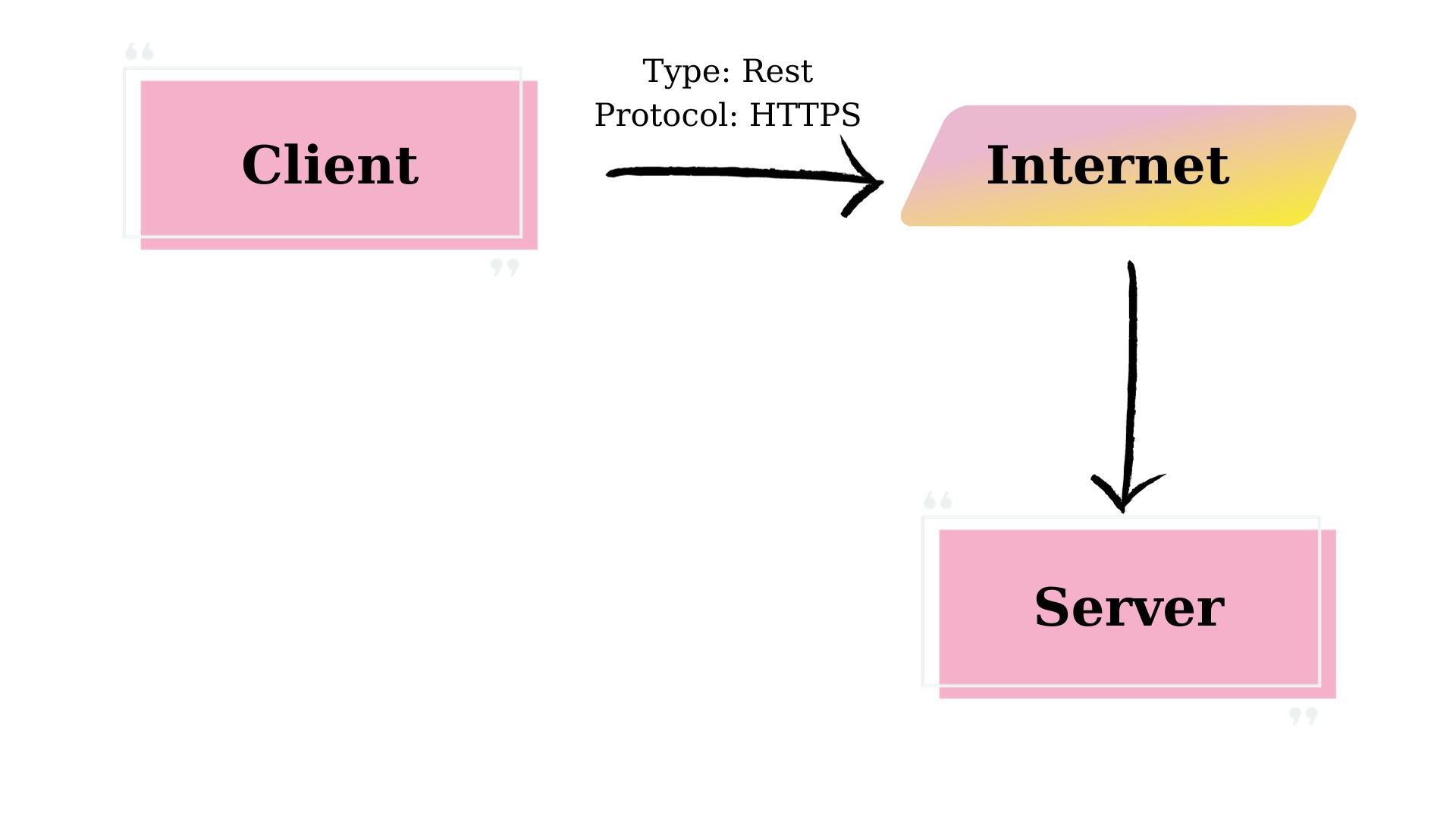
Database connection: Prisma

Schedule: Jira

# Use-Case Model

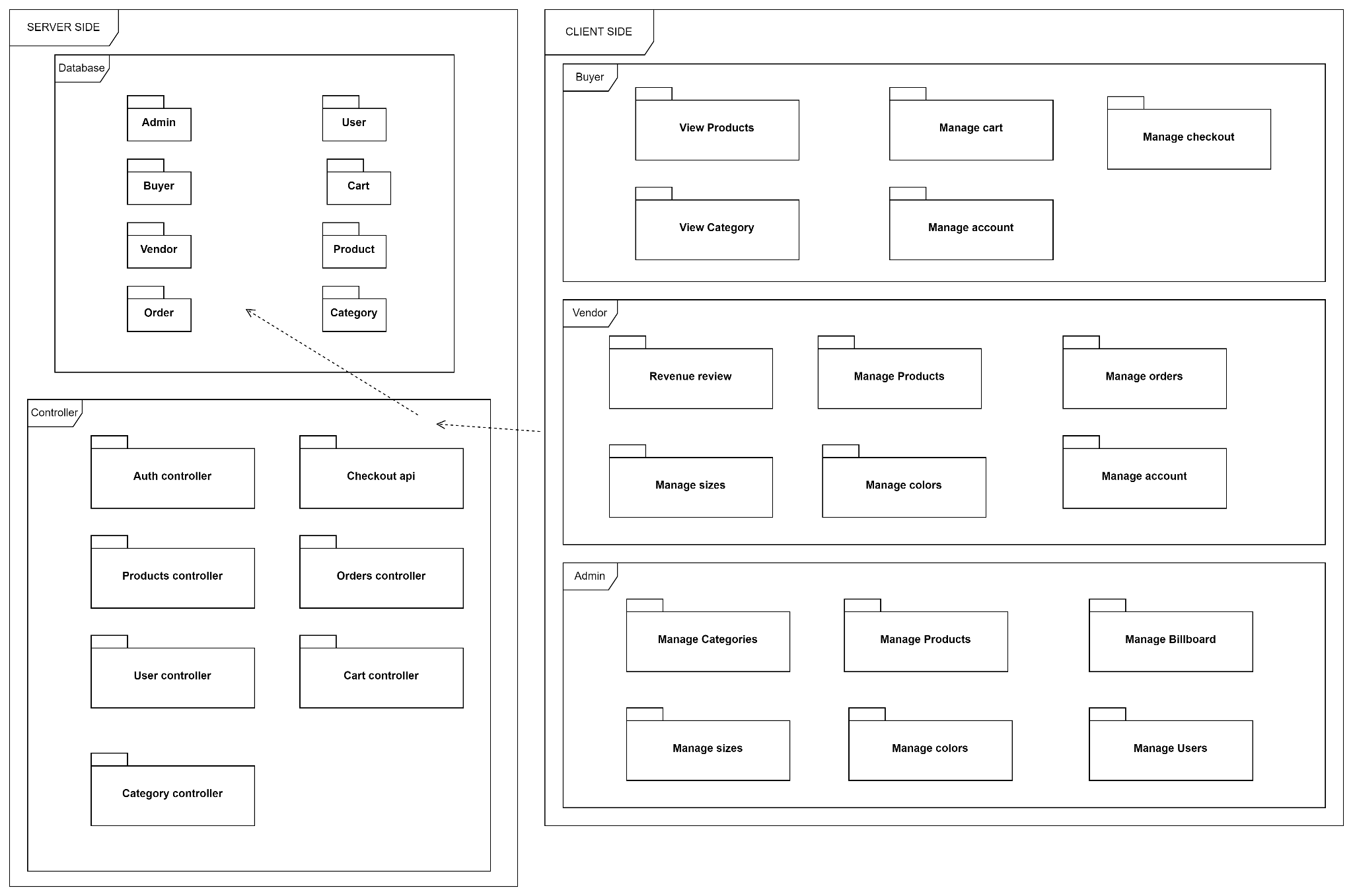
**

# Logical View

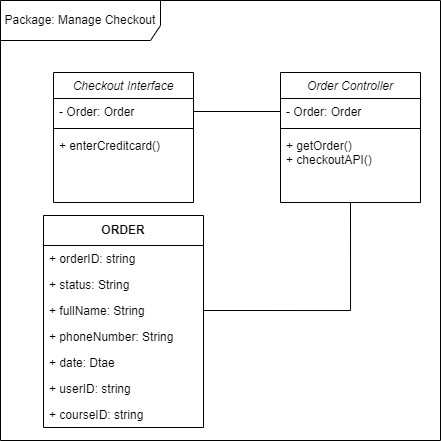


| | **Components** | | --- | | | **Responsibility** | | --- | |
| --- | --- | --- | --- |
| | Client | | --- | | | ● Present users with an HTML-based  user interface accessible through a  web browser  ● Interact with Server component to  submit, retrieve and show data from  database, also create users and  identify users. | | --- | |
| | Server | | --- | | | ● Handle all requests from the client  ● Response clients with appropriate  data  ● Communicate with the database by  executing some query commands  according to the client’s request | | --- | |

Language: Javascript, Typescript

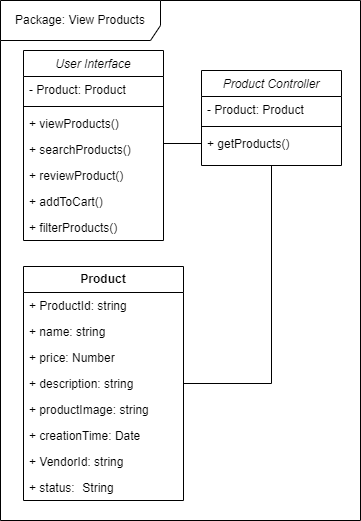


* 1. **Buyers:**
     1. Manage checkout:

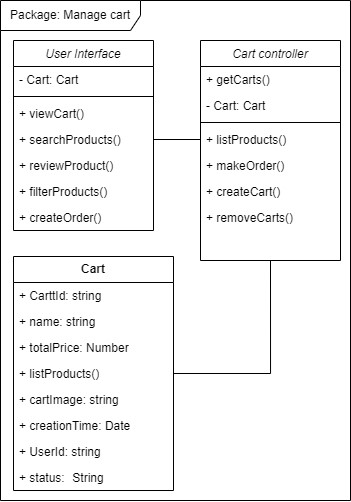


* **Class Checkout Interface:** manage all user’s functions about the checkout from Interface
  + **enterCreditcard()**: user enter creditcard’s ID
* **User:** to store user’s information.
* **Checkout Controller :** manage requests which are sent from the Client side on the checkout**:**
  + **getOrder():** get orderID
  + **checkoutAPI():** get credit card information

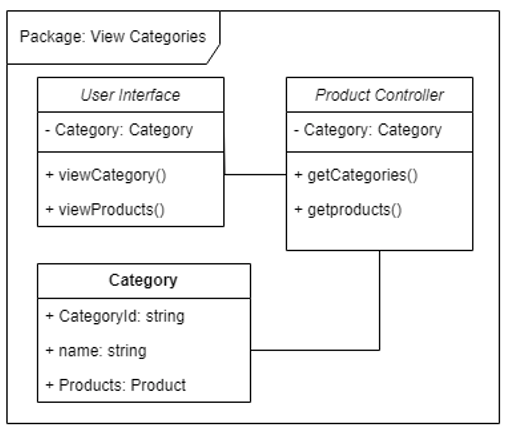
* + 1. Package: View Product



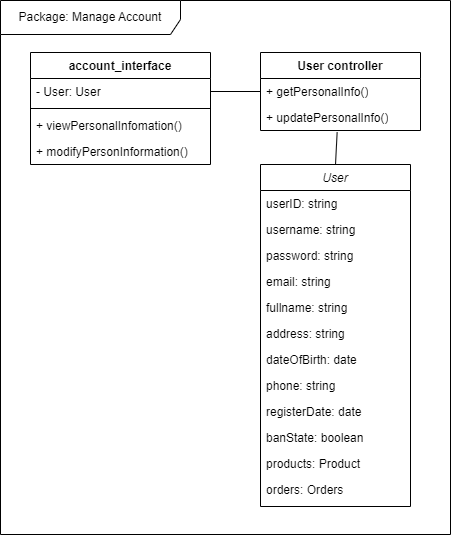
* **User Interface:**
  + **Product:** holds product list for display
  + **viewProducts():** Display product
  + **searchProduct():** search for products using search box
  + **reviewProduct():** give feedback to products
  + **addToCart():** add product to cart for later purchase
  + **filterProducts():** find products using a filter mechanism
* **Product Controller:**
  + **Product:** hold retrieved products from the database
  + **getProducts():** retrieve products from the database
* **Product:** Relation schema of Product in database
  + 1. Package: Manage cart



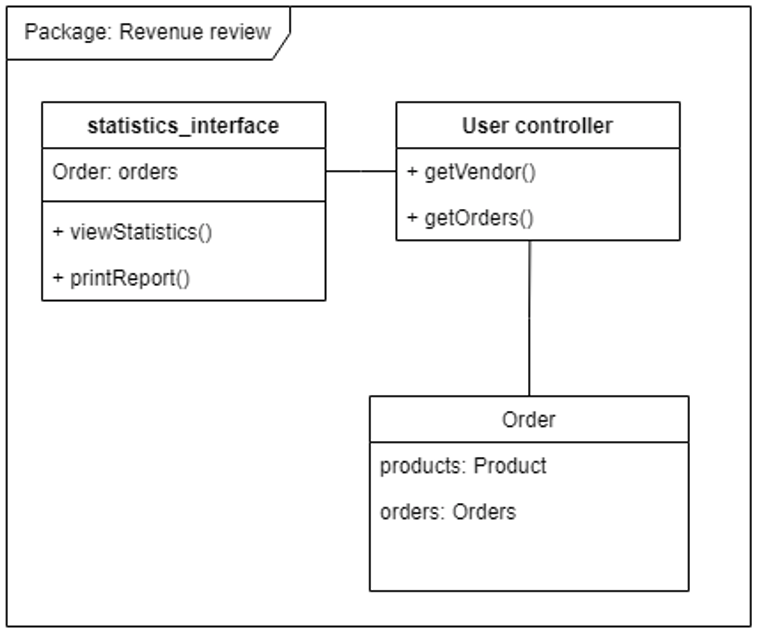
* **User Interface:** manage all cart’s functions from Interface:
  + **viewCart():** view all products in cart
  + **searchProduct():** search product in cart
  + **fillterProduct():** filter product by an attribute ( such as : size, color…)
  + **createOrder():** create an order
* **Cart:** to store cart’s information.
* **Cart Controller :** manage requests which are sent from the Client side:
  + **listProduct():** list all the products in cart
  + **makeOrder():** make an order
  + **createCart():** create a new cart
  + **removeCart():** remove a cart
    1. Package: View category



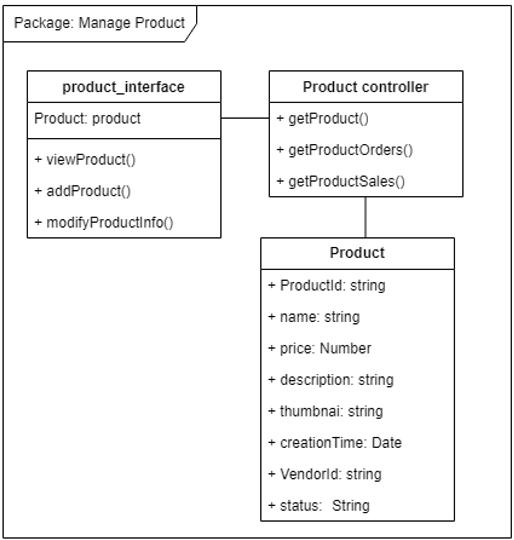
* **User Interface:** manage all category’s functions from Interface
  + **viewCategory():** view all categories
  + **viewProduct():** view all products with categories
* **Category:** to store category’s information.
* **Product Controller :** manage requests which are sent from the Client side:
  + **getCategories():** get all categories
  + **getProduct():** get all product
    1. Package: Manage Account



* **account\_interface:**
  + **User:** holds current user informations
  + **viewPersonalInformation():** call a controller to get personal information from database
  + **modifyPersonInformation():** call a controller to update personal information from a database in the database.
* **User controller**
  + **getPersonalInfo():** retrieve personal information from database
  + **updatePersonalInfo():** modify personal data on database
* **class User:** relation schema for users
  1. Vendor
     1. Package: Revenue review

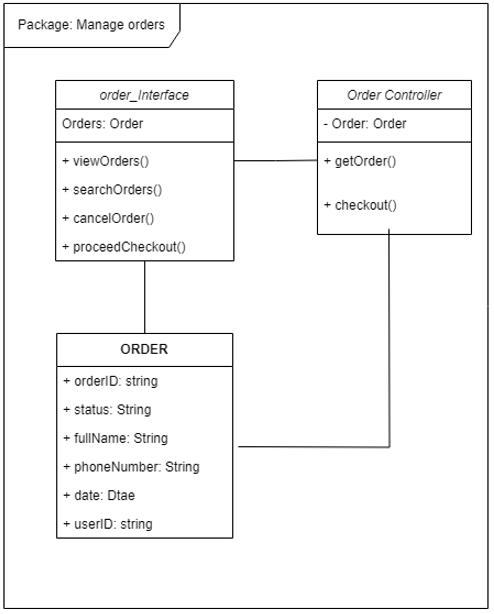


* **statistic\_interface**
  + **orders:** holds a list of orders relate to the vendor
  + **viewStatics():** Calculate and visualize statistics of sales, popularity,...
  + **printReport():** print a well organized report for statistics that is recorded in a period of time.
* **User controller**
  + **getVendor():** view curtain Vendors informations for statistical calculation
  + **getOrders():** Get orders that is related to the vendor from the database
* **Order:** Relation schema for Orders in the database
  + 1. Package: Manage Products

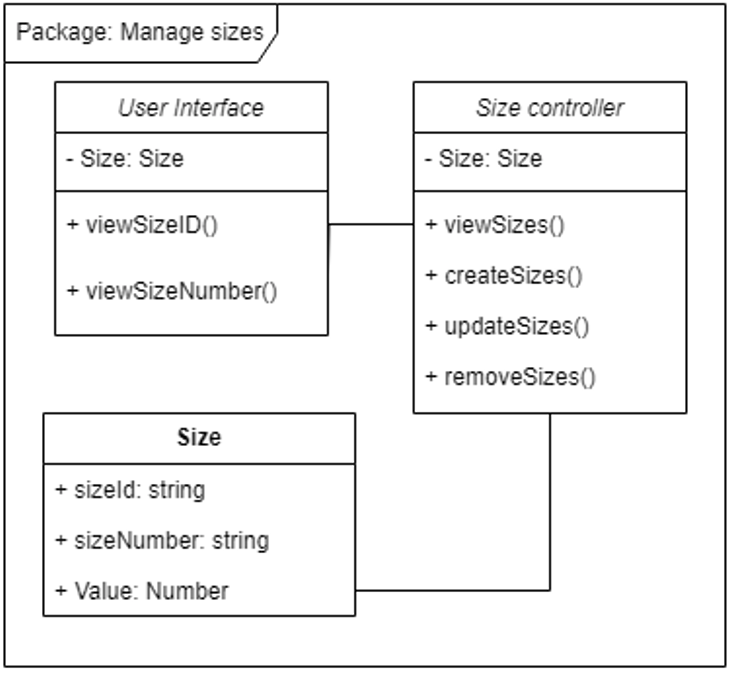


* **product\_interface:** manage all product’s functions from Interface:
  + **viewProduct()**: view all product’s information
  + **addProduct()**: add new product.
  + **modifyProduct()**: filter product by an attribute ( such as : size, color…)
* **Product:** to store cart’s information.
* **Product Controller :** manage requests which are sent from the Client side:
  + **getProduct():** get a product
  + **getProductOrders():** get an order that contains that product
  + **getProductSales():** get sales of that product

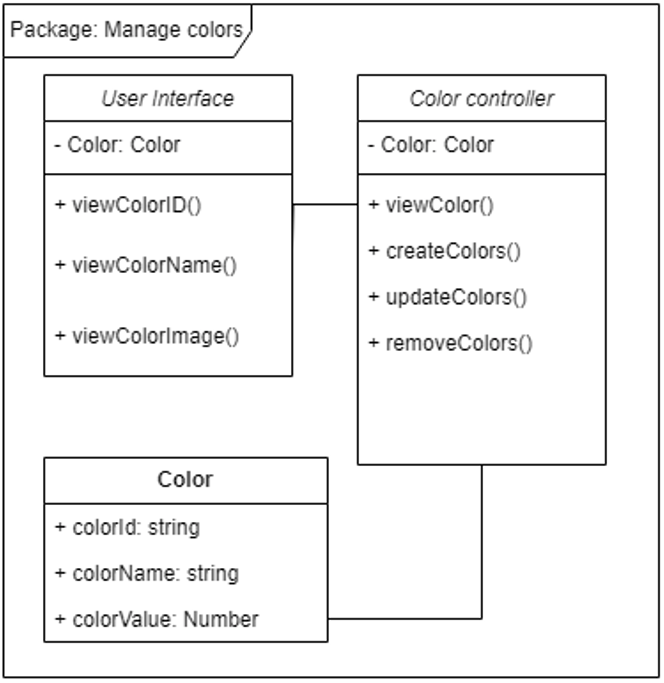
* + 1. Package: Manage orders



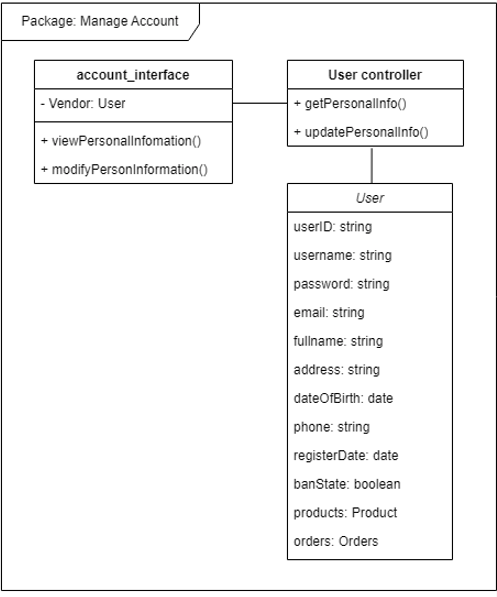
* **order\_interface:** manage all order’s functions from Interface:
  + **viewOrders()**: view all orders of users
  + **searchOrders()**: search orders of users
  + **cancelOrder()**: cancel order of user
  + **proceedCheckout()**: checkout status
* **Order:** to store order’s information.
* **Order Controller :** manage requests which are sent from the Client side:
  + **getOrder():** list all the products in order
  + **checkout():** store checkout’s information
    1. Package: Manage size



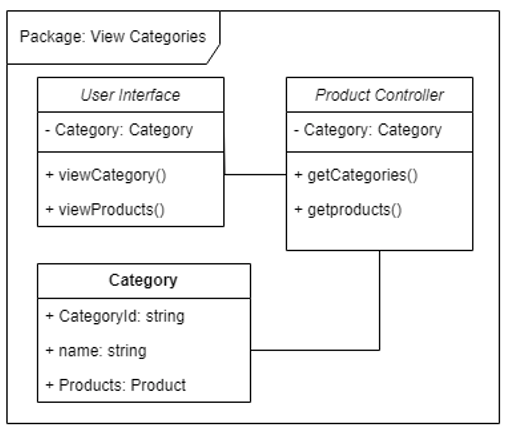
* **User Interface:**
  + **Size**: available sizes
  + **viewSizeID()**: show size IDs
  + **viewSizeNumber()**: show size numbers
* **Size controller:**
  + **Size**: available sizes
  + **viewSizes()**: retrieve sizes from database
  + **createSizes()**: create new size and save it to the database
  + **updateSizes()**: modify a size in the database
  + **removeSizes()**: remove a size from database
* **Size:** relation schema for sizes in the database
  + 1. Package: Manage colors



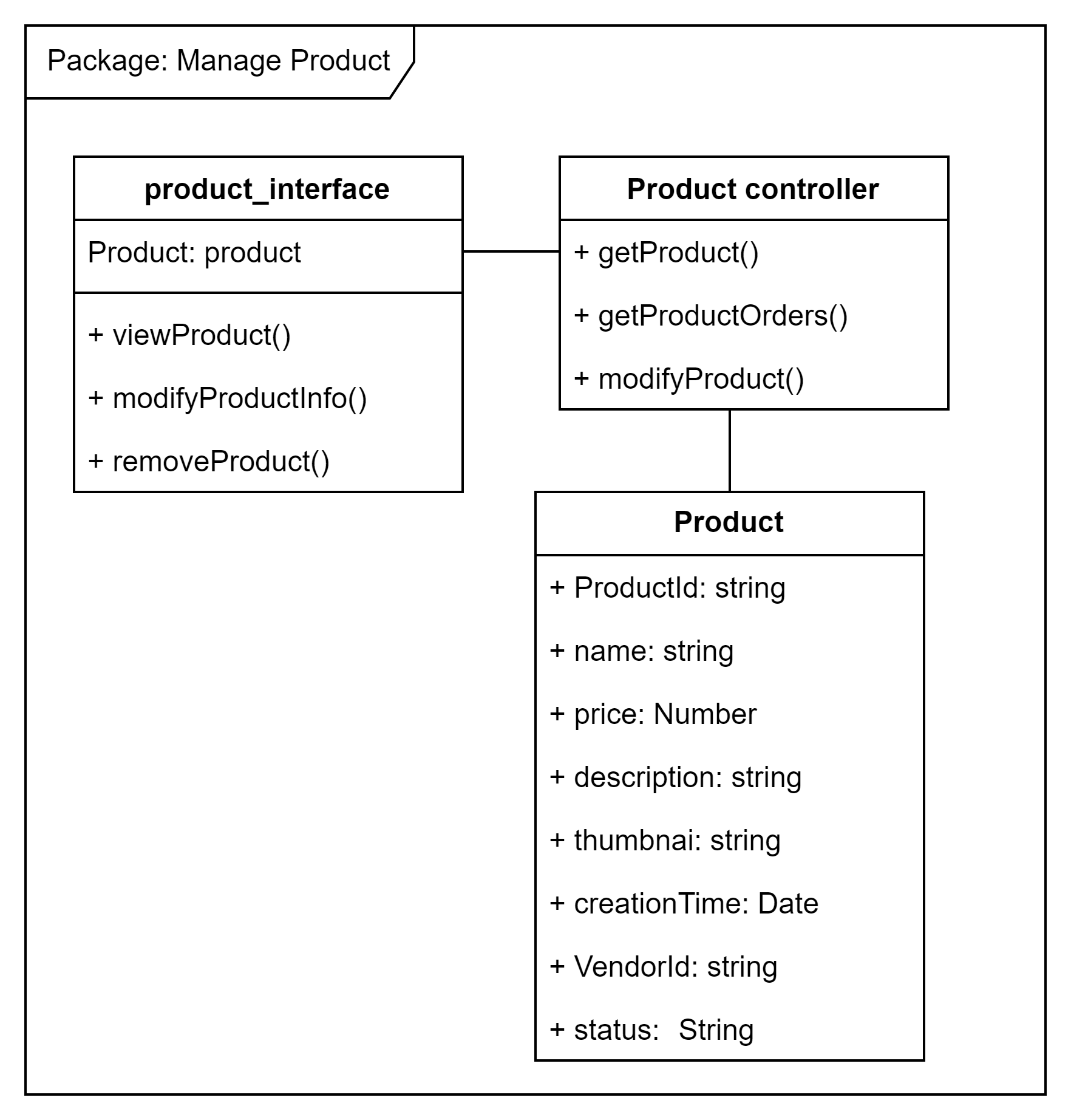
* **User Interface:** manage all category’s functions from Interface
  + **viewColorID():** view ID of a color
  + **viewColorName():** view name of a color
  + **viewColorImage():** view an image of a color
* **Color:** stor a color’s ID, name and value
* **Color Controller**
  + **viewColor():** view a color
  + **createColors():** create a color
  + **updateColors():** update a color
  + **removeColors():** remove a color
    1. Package: Manage account



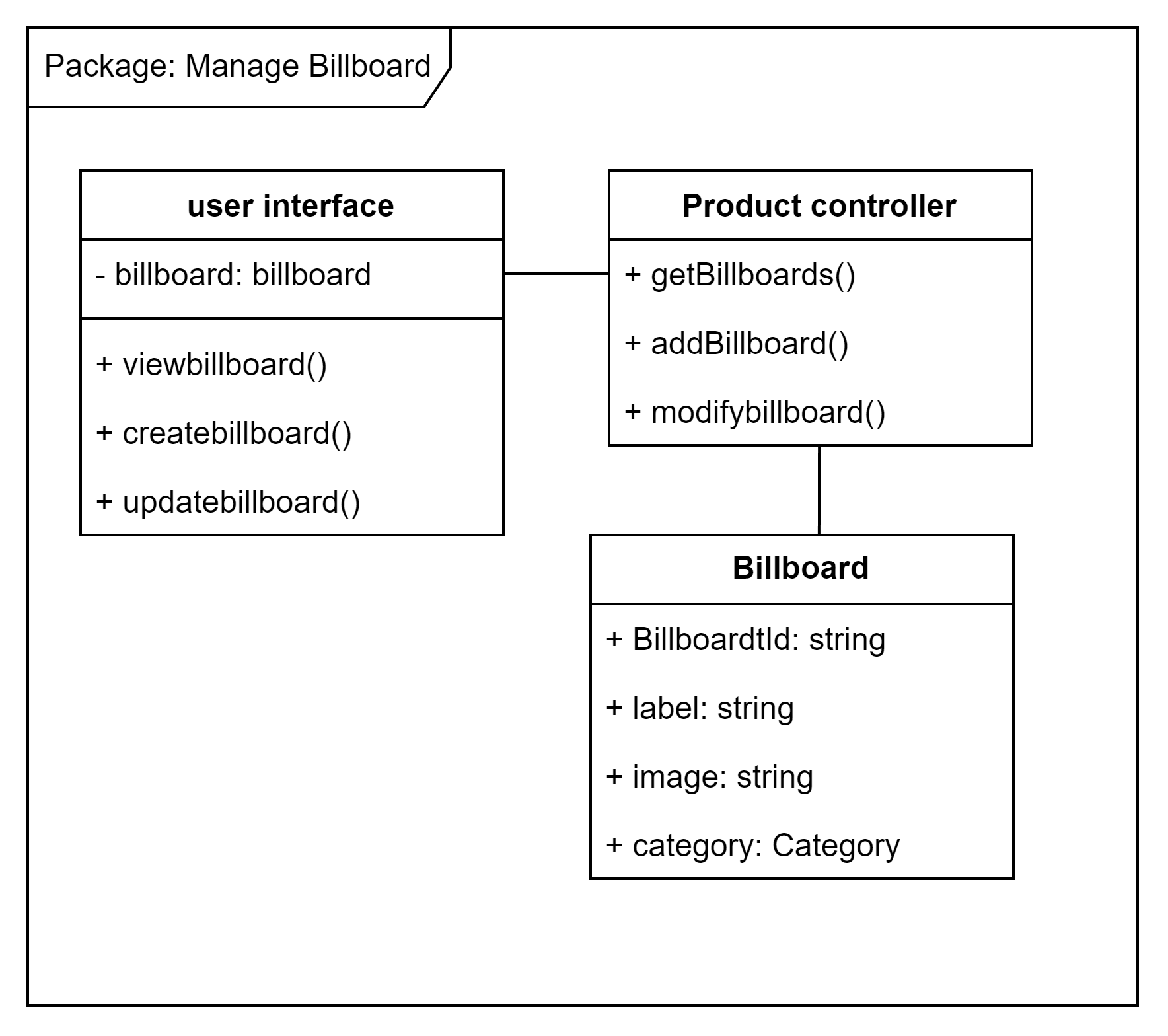
* **account\_interface:**
  + **User:** holds current user informations
  + **viewPersonalInformation():** call a controller to get personal information from database
  + **modifyPersonInformation():** call a controller to update personal information from a database in the database.
* **User controller**
  + **getPersonalInfo():** retrieve personal information from database
  + **updatePersonalInfo():** modify personal data on database
* **class User:** relation schema for users
  1. Admin
     1. Package: Manage Categories



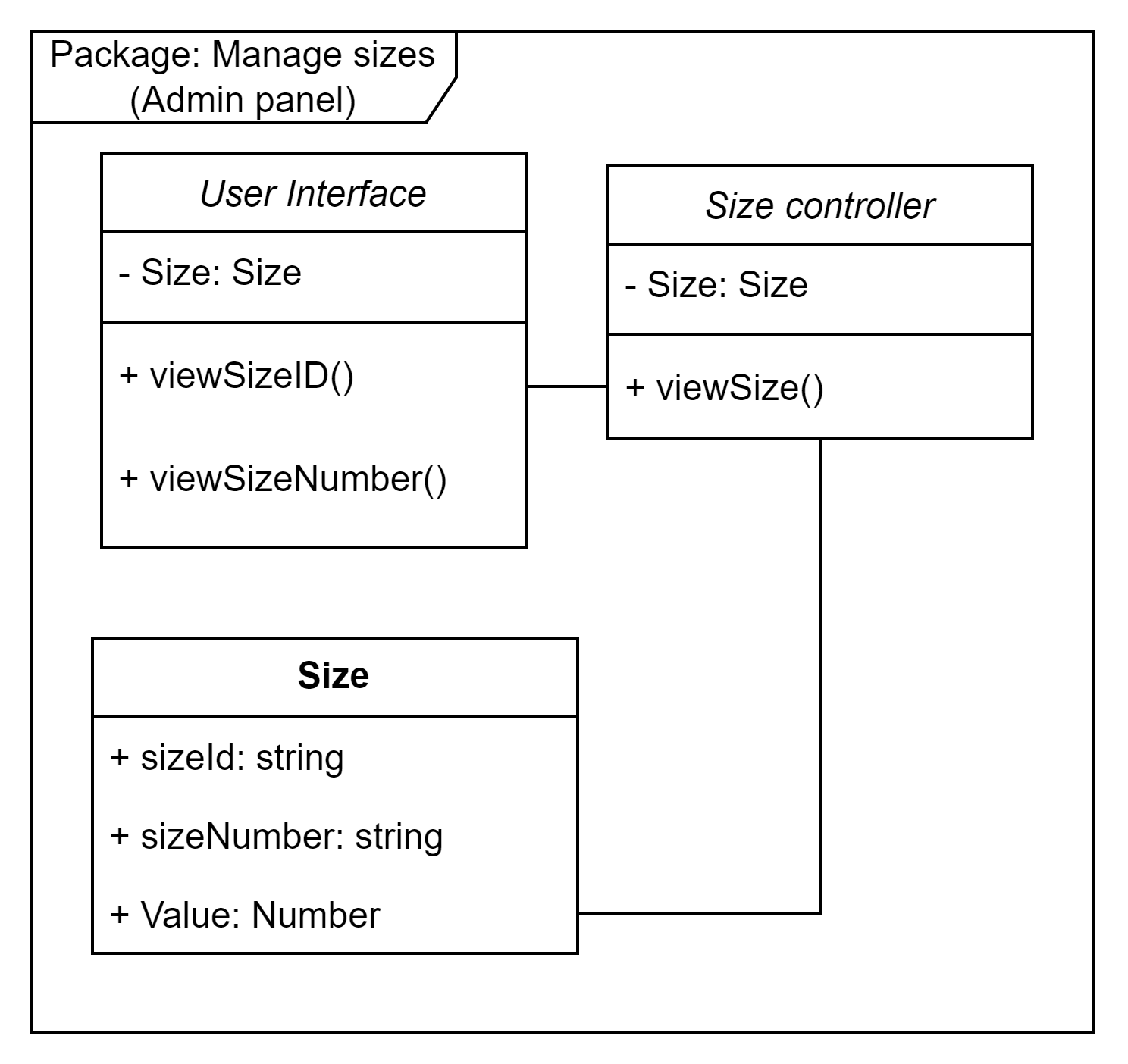
* **User Interface:** manage all category’s functions from Interface
  + **manageCategory():** manageCategories
  + **viewProduct():** view all products with categories
* **Category:** to store category’s information.
* **Product Controller :** manage requests which are sent from the Client side:
  + **getCategories():** get all categories
  + **getProduct():** get all product
    1. Package: Manage Products



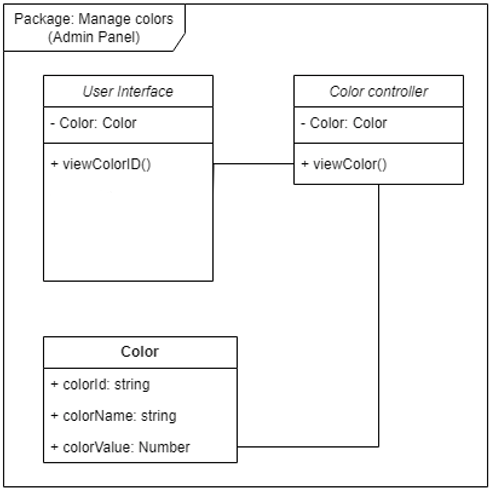
* **product\_interface:**
  + **product:** hold a list of product
  + **viewProduct():** display product
  + **modifyProductInfo():** Change some information of a product
  + **removeProduct():** delete a product from database
* **Product controller**
  + **getProduct():** retrieve products from database
  + **getProductOrders():** get orders of related a product in database
  + **modifyProduct():** update/delete products in database
* **Product:** Relation schema of products in database
  + 1. Package: Mange Billboard



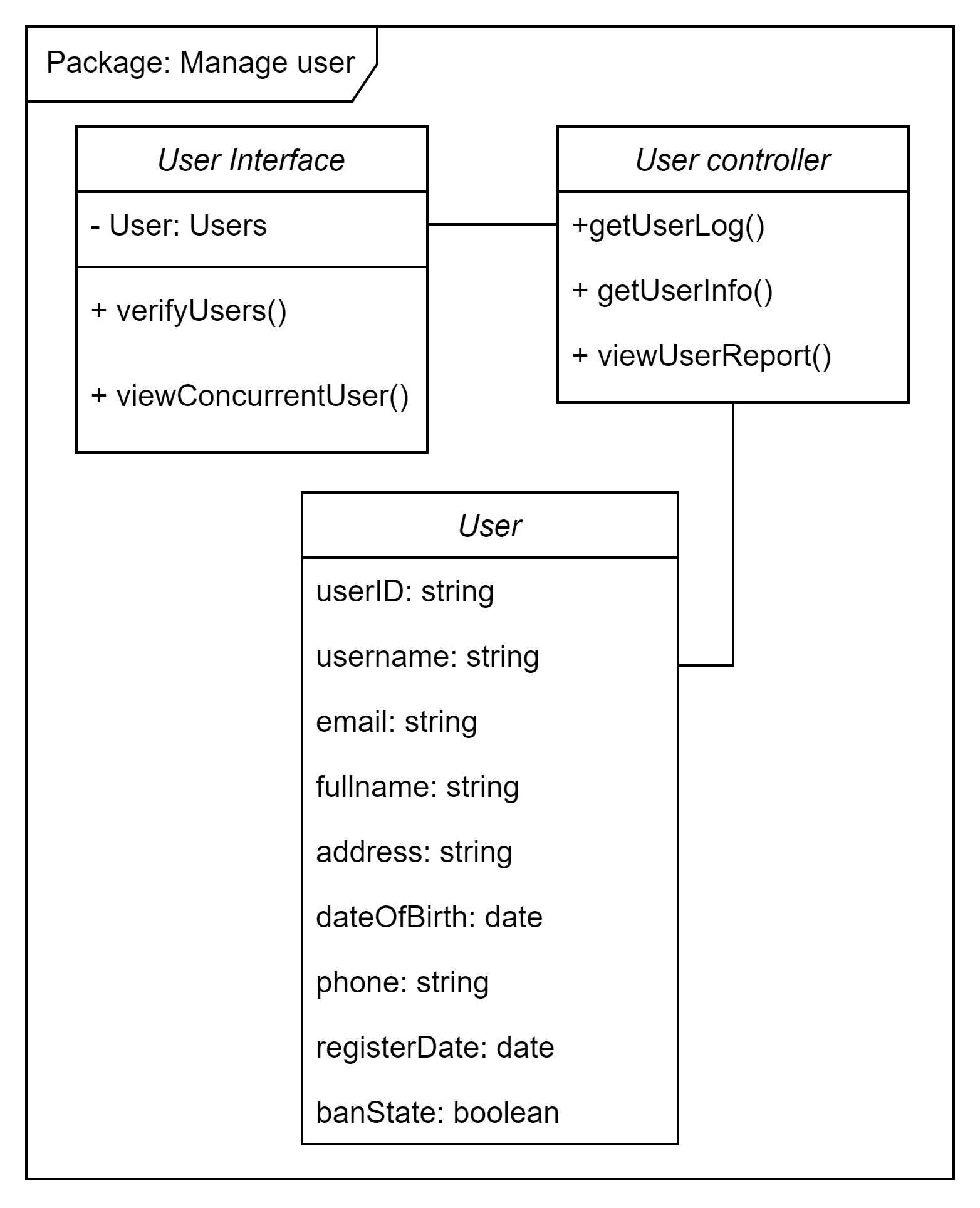
* **user interface**
  + **billboard:** hold a list of billboards
  + **viewbillboard():** get and display the list of billboard
  + **createbillboard():** create a new billboard
  + **updatebillboard():** update an existing billboard
* **Product controller**
  + **getBillboards():** get billboard from database
  + **addBillboard():** insert a billboard to database
  + **modifybillboard():** update a billboard and save it to database
* **Billboard:** Relation schema for billboards in database
  + 1. Package: Manage sizes



* **User Interface:**
  + **Size**: available sizes
  + **viewSizeID()**: show size IDs
  + **viewSizeNumber()**: show size numbers
* **Size controller:**
  + **Size**: available sizes
  + **viewSizes()**: retrieve sizes from database
* **Size:** relation schema for sizes in the database
  + 1. Package: Mange colors



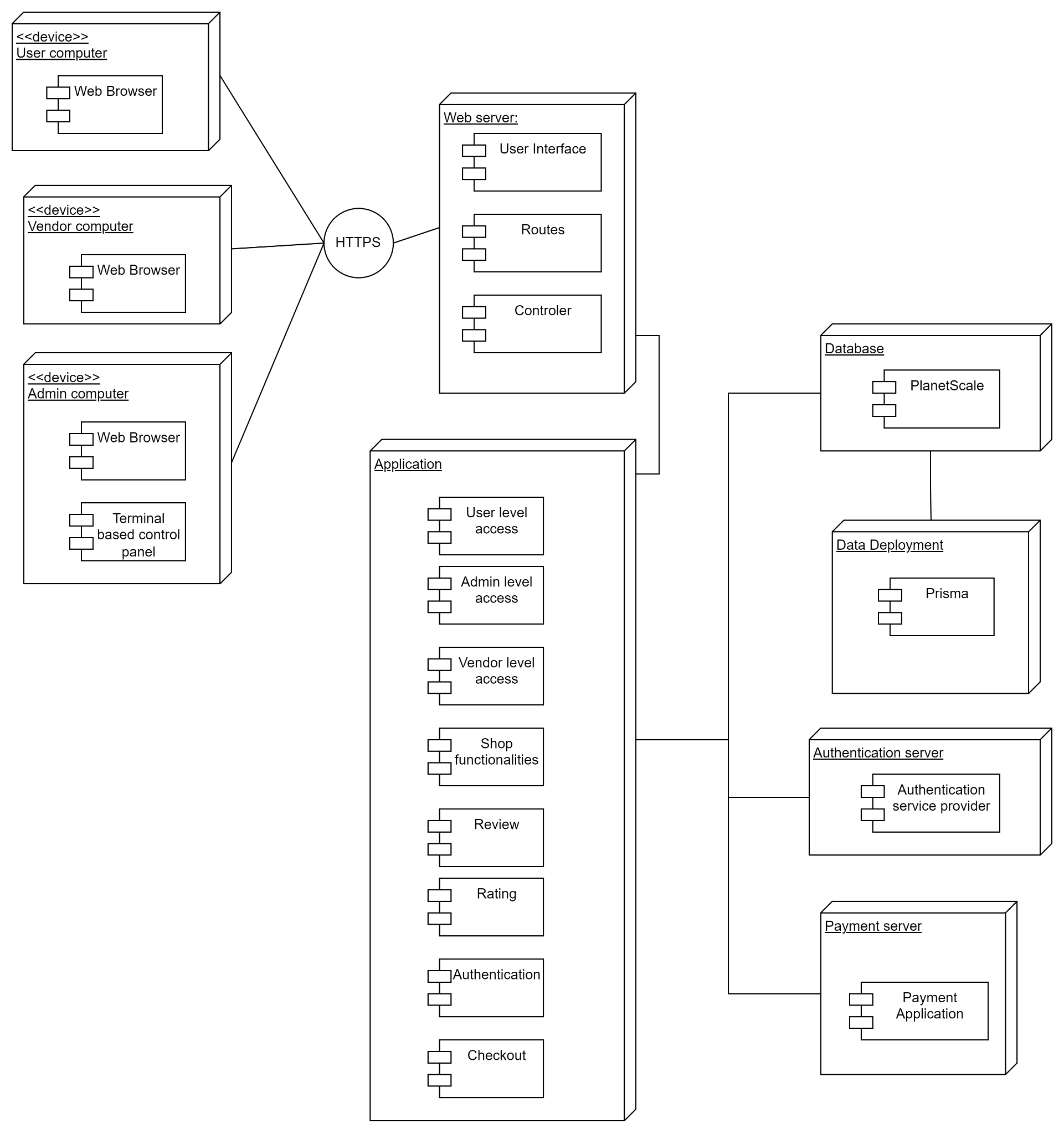
* **User Interface:** manage all category’s functions from Interface
  + **viewColorID():** view ID of a color
* **Color:** store a color’s ID, name and value
* **Color Controller**
  + **viewColor():** view a color
    1. Package: Manage Users



* **User Interface**
  + **User:** holds users data
  + **verifyUsers():** verify a user
  + **viewConcurrentUser():** view logged in users
* **User Controller**
  + **getUserLog():** get a user’s activities on the website
  + **getUserInfo():** get information of a user
  + **viewUserReport():** View a user’s reports
* **User:** Relation schema for users data in database

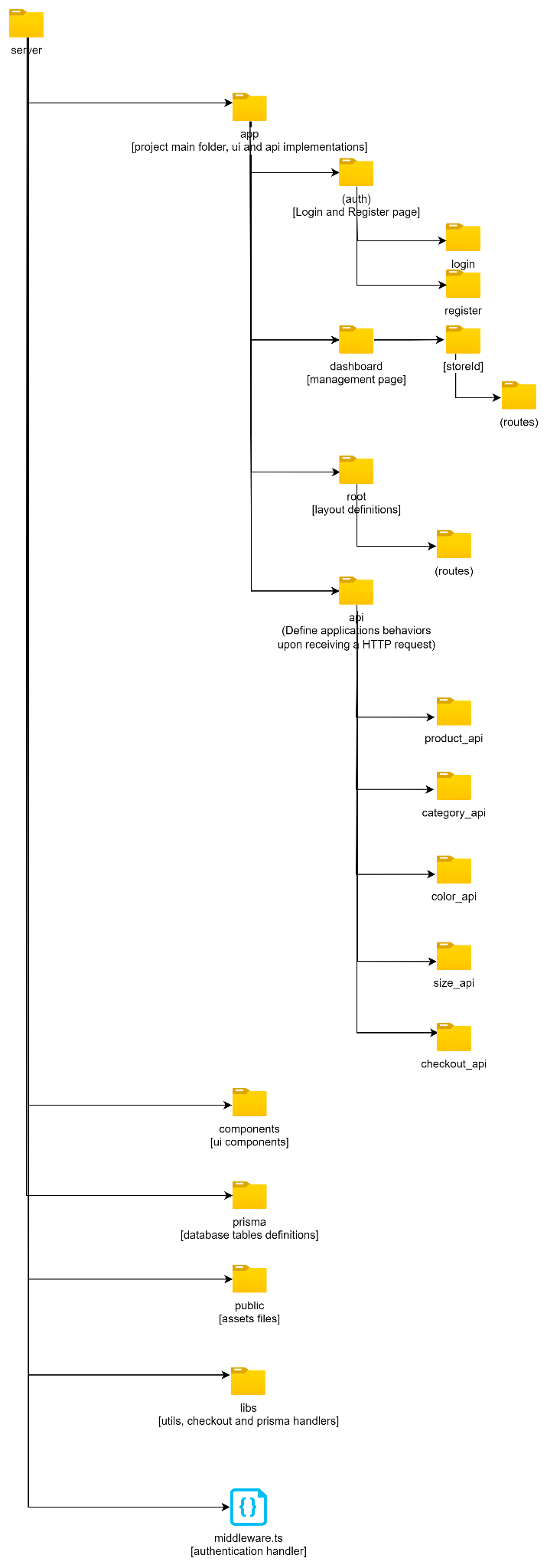
# Deployment

This system is hosted on a distant server since it is a web application (Vercel). Another hosting space will house the database(Plannetscale).The backend does all of the work, saving the client machine from using a lot of CPU power. When creating statistical graphs, all data processing is done at the front end, requiring a certain degree of speed from the client computer.



# Implementation View

**6.1. Web server folder structure:**

**

**6.2. Web client folder structure:**

