GROUP 08: SUPER IDOL

SIBOOKS WEB
Software Architecture Document

Version 2.2

Revision History

Date	Version	Description	Author
23/07/2024	1.0	Formatting the report.	Lê Phước Phát
31/07/2024	1.1	Write the introduction, goals and constraints.	Bùi Lê Khôi
01/08/2024	1.2	Do the specification of the Model layer.	Lê Phước Phát
01/08/2024	1.3	Do the specification of the Service layer.	Tô Quốc Thanh
01/08/2024	1.4	Do the specification of the View layer.	Thái Huyễn Tùng
02/08/2024	1.5	Update Use-case model Do the specification of the Controller layer.	Ngô Văn Khải
02/08/2024	1.6	Do the remaining specifications of the Model layer.	Bùi Lê Khôi
03/08/2024	1.7	Uploading and updating all pictures about the logical view (figure1, 2, 3, 4, 5, 6, 7, 8, 9)	Lê Phước Phát
03/08/2024	1.8	Formatting the final report and checking all errors.	Lê Phước Phát
15/08/2024	2.0	Checking all issues from sections 1 to 4.	Lê Phước Phát
15/08/2024	2.1	Write a description and draw a deployment diagram (section 5).	Lê Phước Phát
16/08/2024	2.2	Write a description and draw a folder structure diagram (section 6).	Lê Phước Phát

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This **Software Architecture Document** (hereafter referred to as SAD) provides the reader with a comprehensive overview of the architecture of the entire **SIBOOKS WEB** system, using various different architectural perspectives to describe different aspects of the system. This **SAD** aims to capture the key architectural decisions that were made on the system.

1.2 Scope

This **SAD** is used in the development of the team's project - **SIBOOKS WEB**.

1.3 Overview

Below is the structure and contents provided in this SAD.

- **Introduction**: provide a brief description of the usage of this document.
- **Architecture Goals and Constraints**: provide the overall goals of this project, along with constraints and requirements for the architecture.
- **Use-case Model**: The use-case model of the system.
- **Logical View**: This section includes the list of components in the system and the relationships between them, including package and class diagrams.
- **Deployment**: Provide the method used in the project to deploy application components.
- **Implementation View**: The source code structure and guide to implement the components.

2. Architectural Goals and Constraints

Below are the goals and constraints that the project needs to achieve, as well as the project requirements.

- **Programming Language**
  - The project uses HTML, CSS, JavaScript and ReactJS framework for front-end development, Python for back-end development, and PostgreSQL for database management.
- **Programming Environment**
  - Visual Studio Code for front-end and back-end development, pgAdmin4 for handling database design.

- **Performance requirements**
  - The system can work on several web browsers.
  - The system can operate continuously 24/7.
  - The system can handle at least 500 users logged in at the same time.
  - The system response time for basic features should be less than 5 seconds.
  - The system must ensure encryption and security of personal information, as well as legal principles and policies.
  - Friendly and suitable UI, easy to learn and use for everyone.
  - The system can easily be maintained and upgraded when needed.
- **Constraints**
  - When creating an account, the username is unique and the system must prevent the user from being able to create one, while also ensuring user account security and encryption.
  - Users have limited access to the website in the guest mode. To access all features and have a better experience, signing in by an account is required.
- **Dependencies**
  - A stable internet connection is required to ensure a good user experience.
- **User Manual Document**
  - Detailed documentation on accessing and using the website's features and device configuration will be released when the project is complete.

3. Use-Case Model

3.1 Admin Use-Case Model

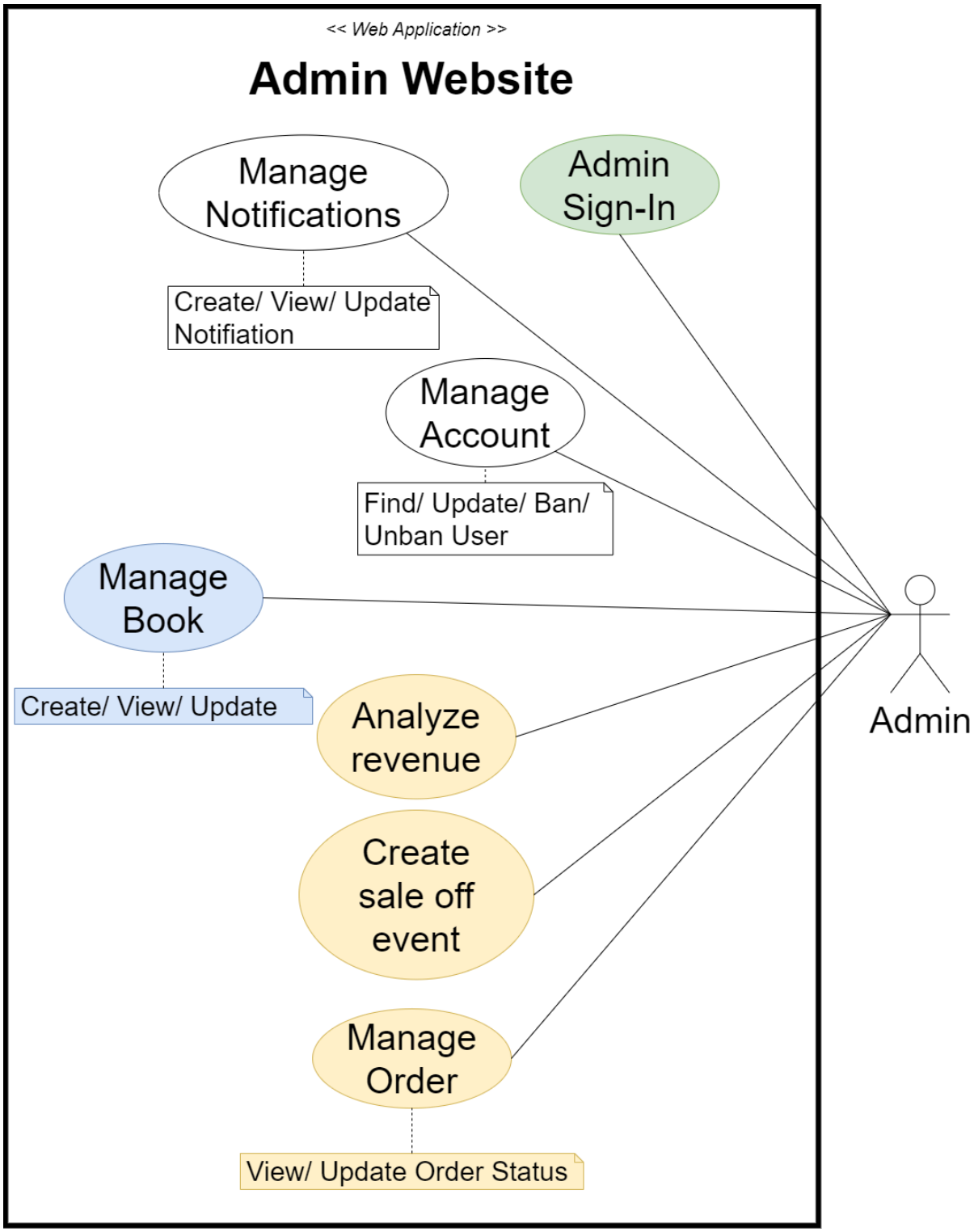

Figure 1. *The use case model of Admin*

3.2 Customer Use-Case Model

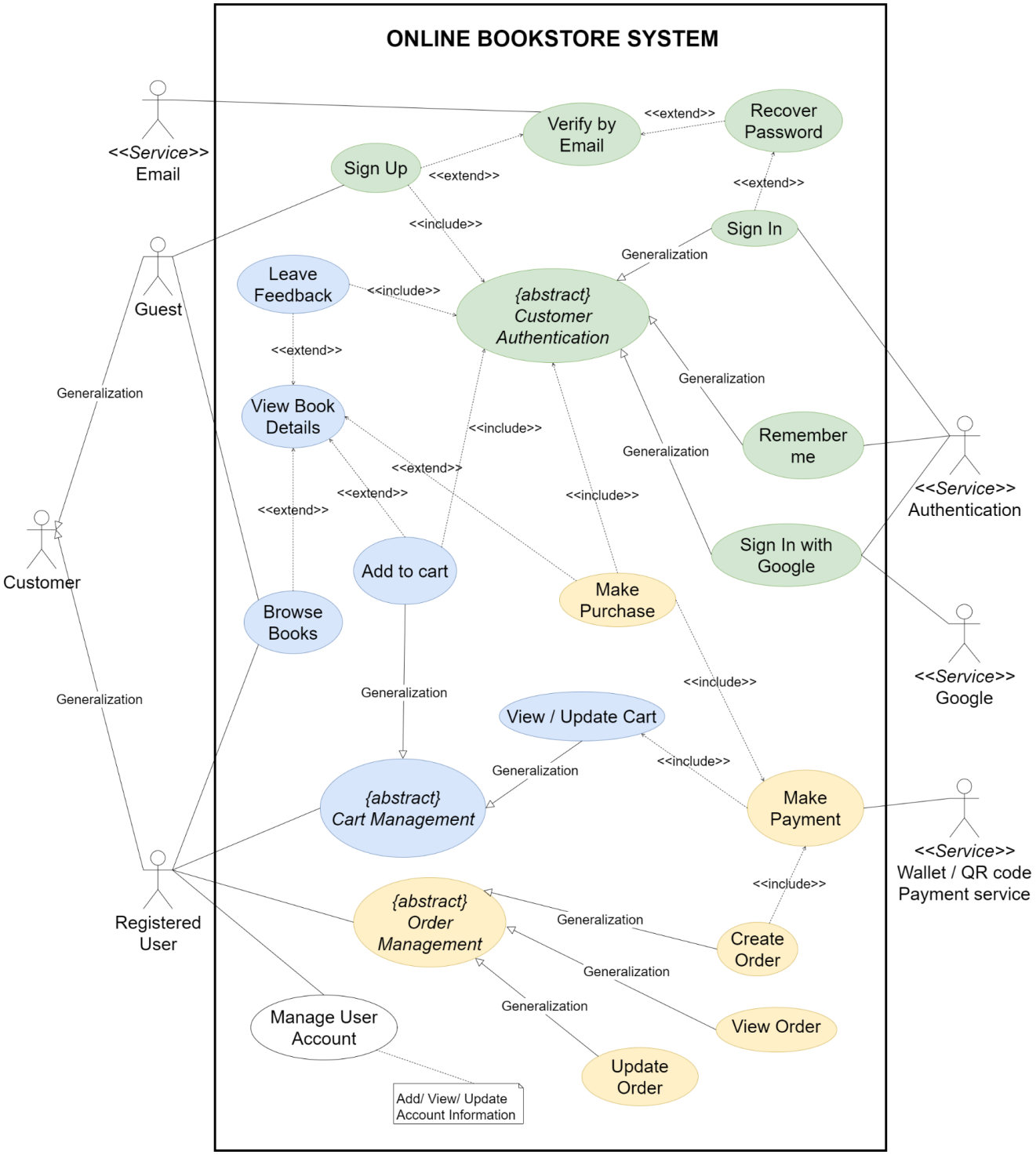

Figure 2. *The use case model of Customer*

4. Logical View

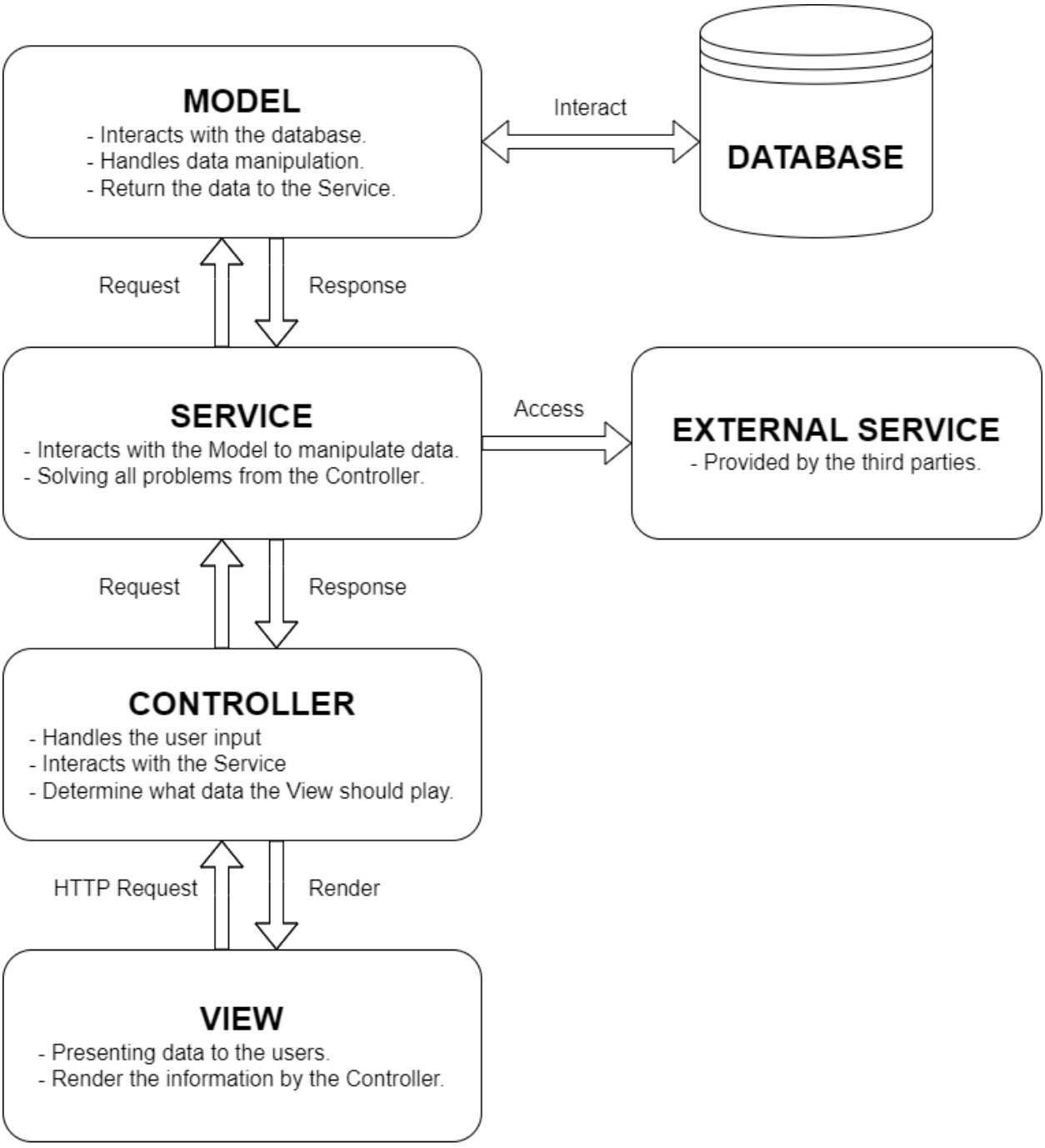

Figure 3. *The MVC model for SIBOOK WEB*

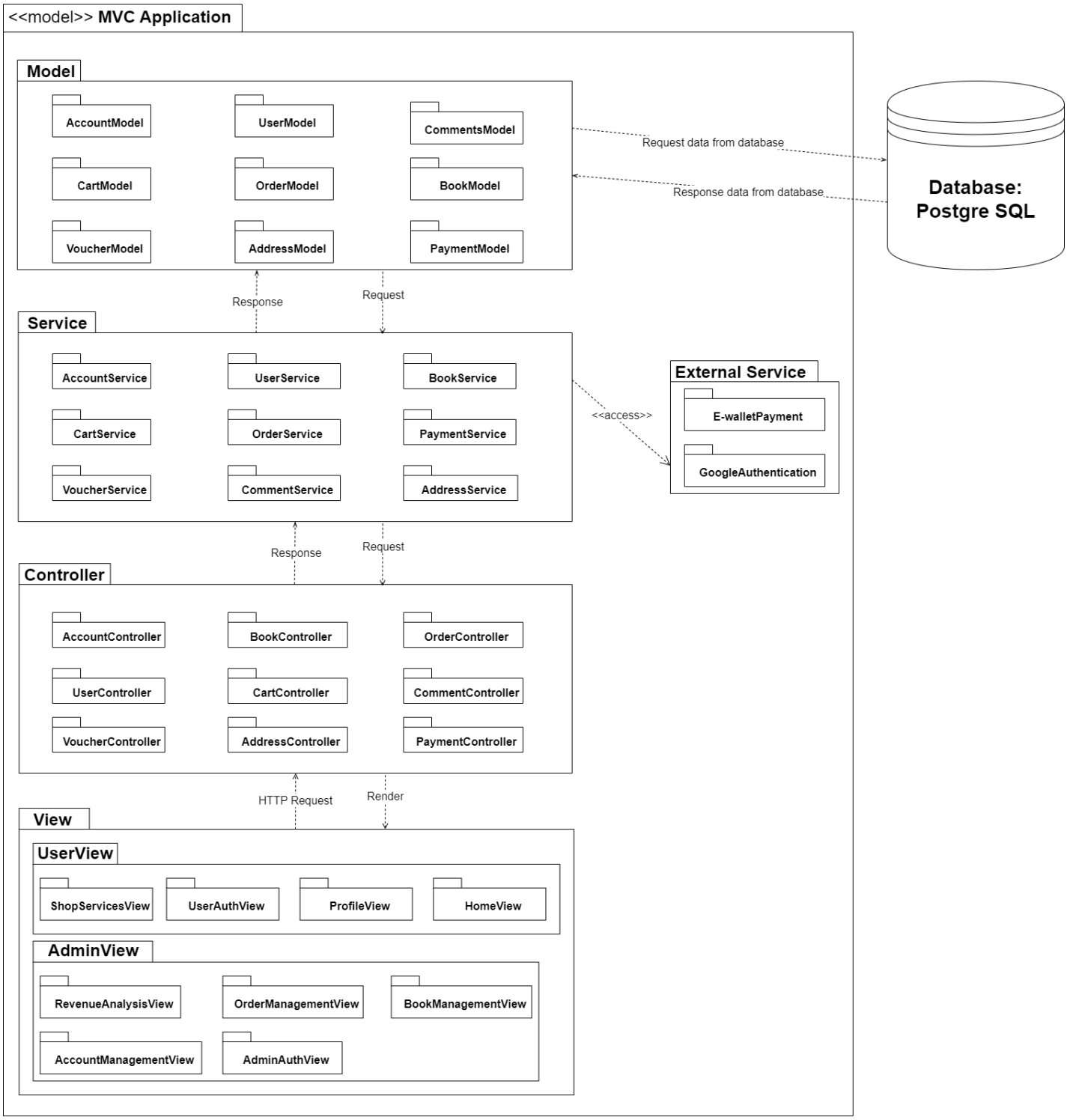

***Figure 4**. The general logical view of SIBOOK WEB*

- **Model**: represents the application's data structure. They interact with the database, handle data manipulation, and return the data to the Services.

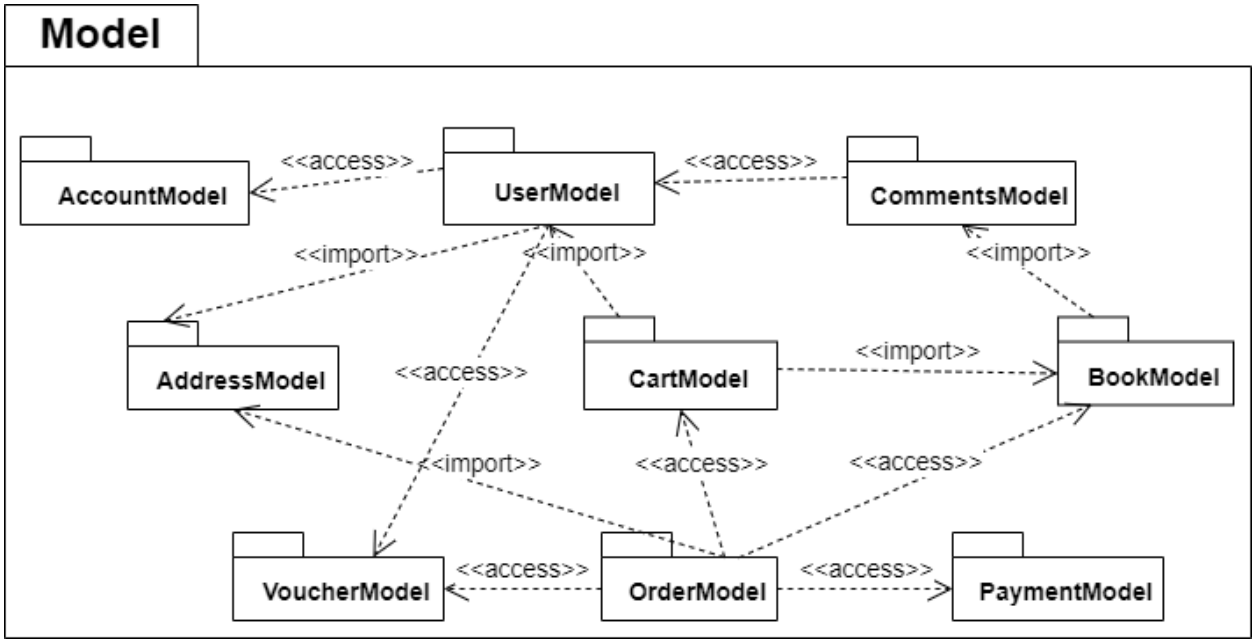

***Figure 5**. The package diagram of the Model*

- **Service**: acts as an intermediary between the **Controller** and the **Model.** It encapsulates business logic and interacts with the **Model** to fetch or manipulate data. This layer helps to keep the **Controller** thin by offloading business logic, making it easier to maintain and scale.

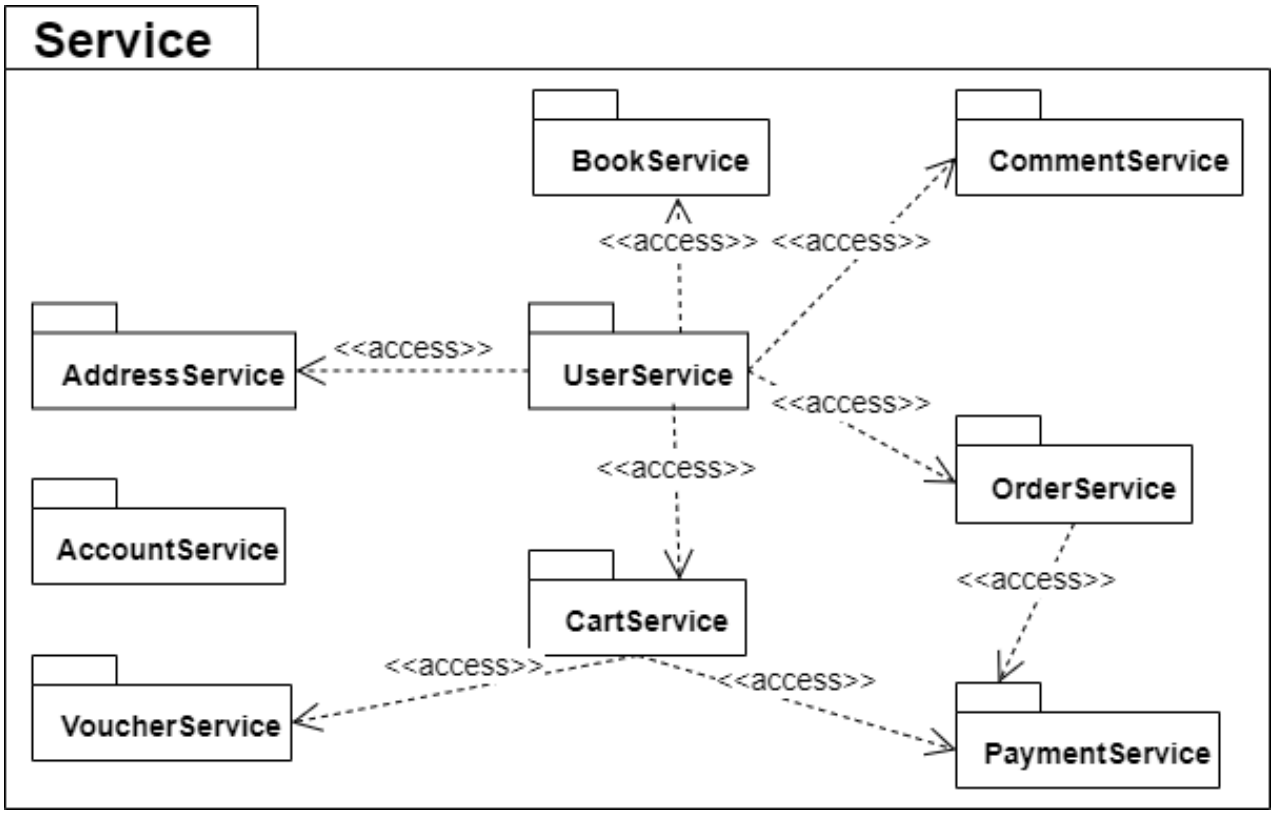

Figure 6. *The package diagram of the Service*

- **External services**: The services provided by third parties are called via APIs to support certain system features.

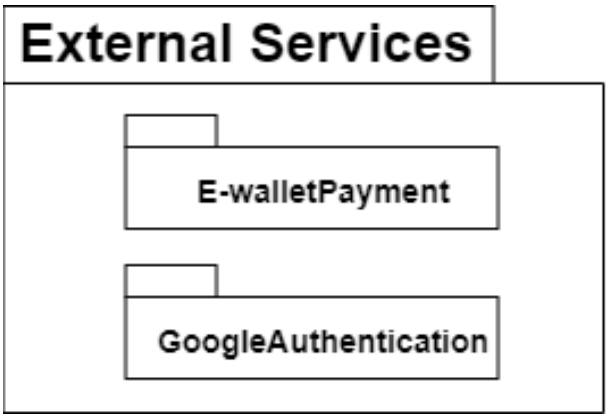

Figure 7. *The package diagram of the External Services*

- **Controller**: handles user input, interacts with the **Service**, and determines what data the **View** should display. It processes incoming requests, works with the **Service** to fetch or modify data, and then updates the **View** accordingly.

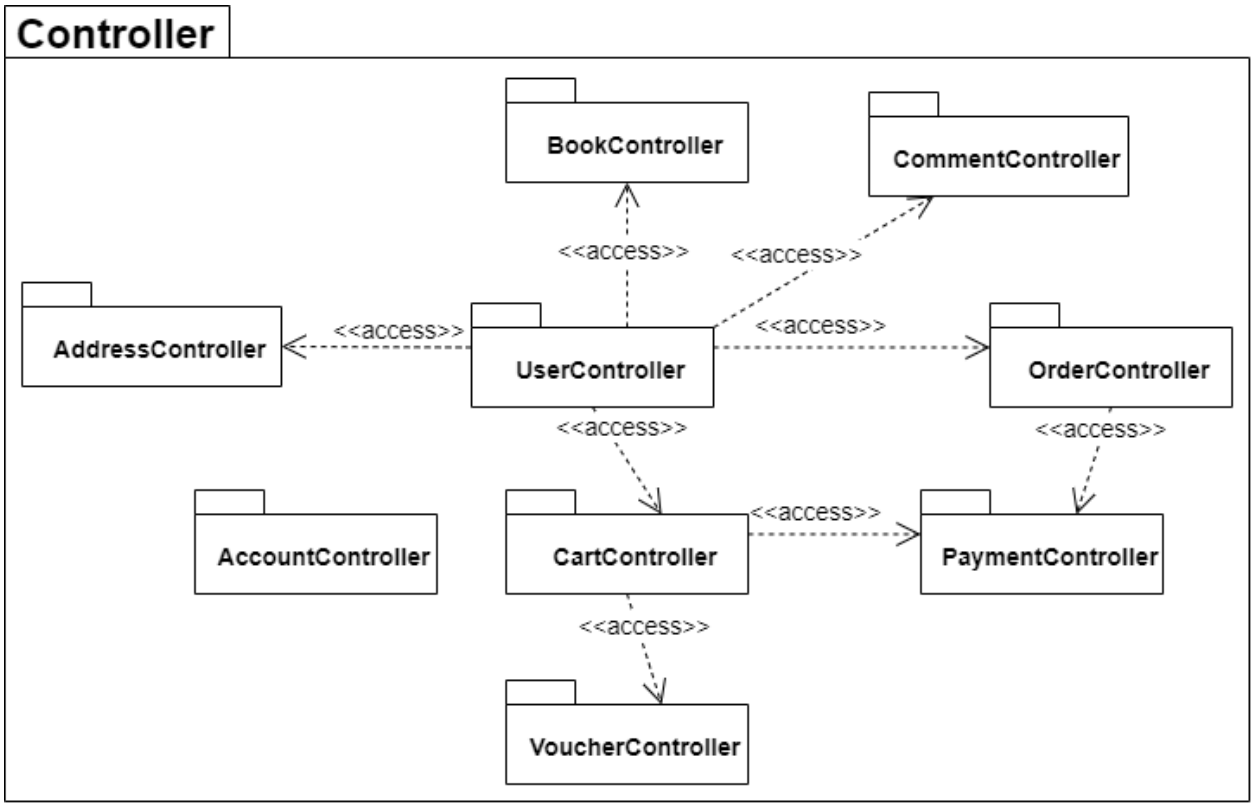

***Figure 8**. The package diagram of the Controller*

- **View**: are responsible for presenting data to the user. They render the information provided by the **Controller** in a format that is understandable and usable.

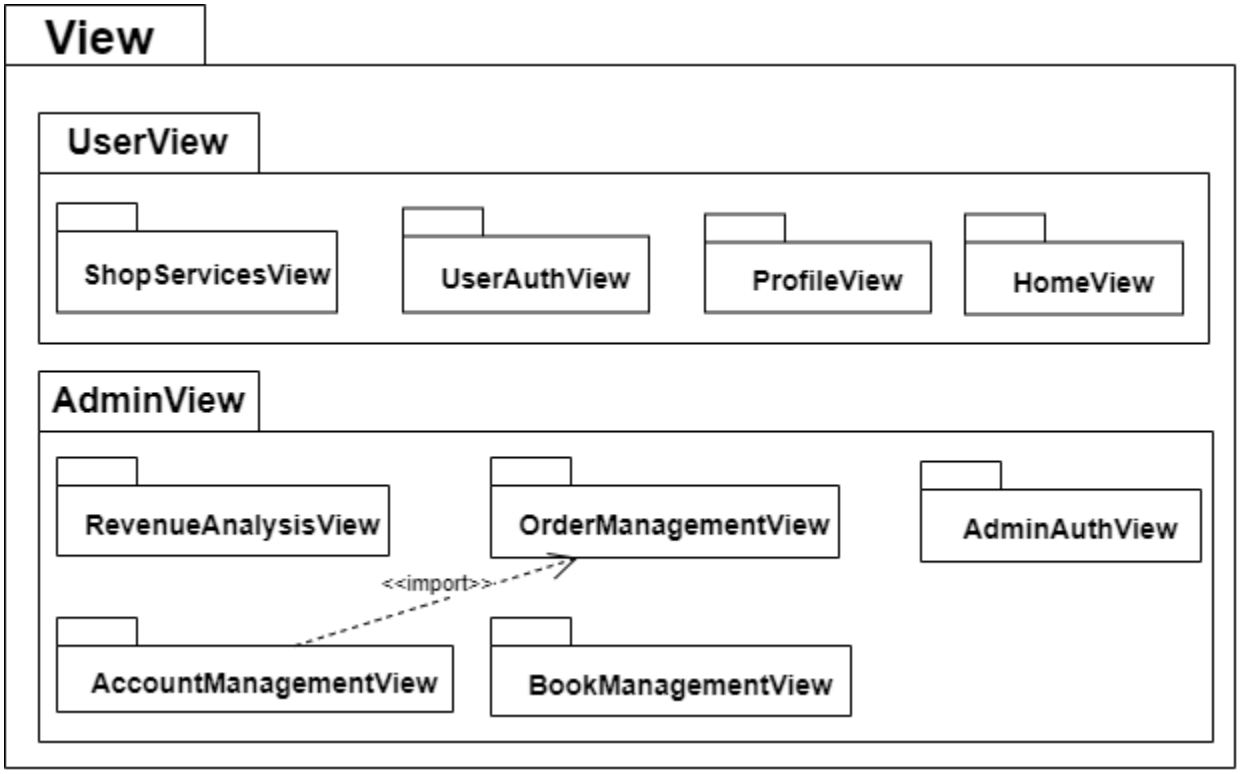

Figure 9. *The package diagram of the View*

4.1 Component: UserAuthView (View)

- Description
  - This component is responsible for displaying users' authentication view of the website.

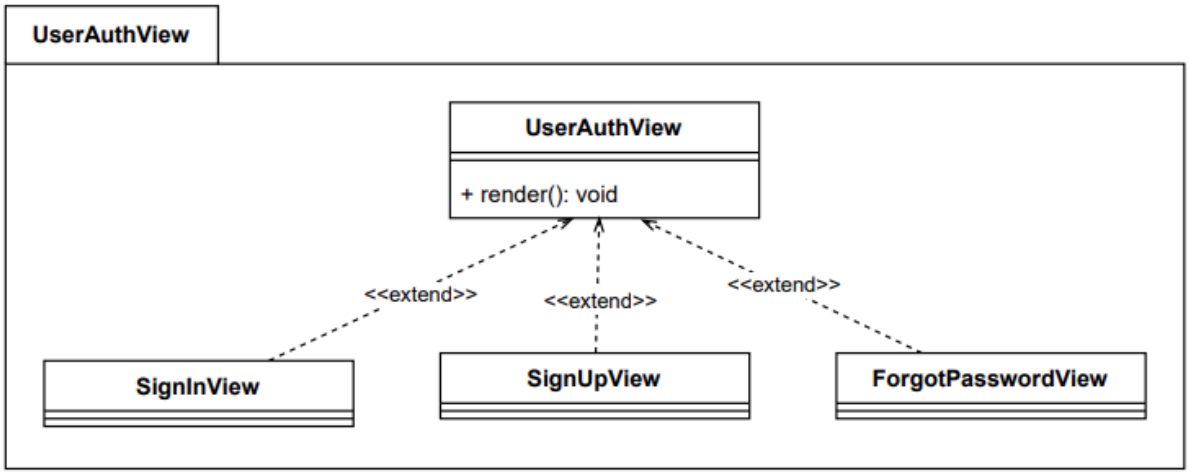

Figure 10. *The UserAuthView Component*

- Explanation
    - Function **render()** of the **UserAuthView** class helps display the view in the authentication section to users.
    - Derived classes such as **SignInView**, **SignUpView**, and **ForgotPasswordView** will use the **render()** function from the base class **UserAuthView** to render a specific view for users.

4.2 Component: ProfileView (View)

- Description
    - This component is responsible for displaying the profile of users..

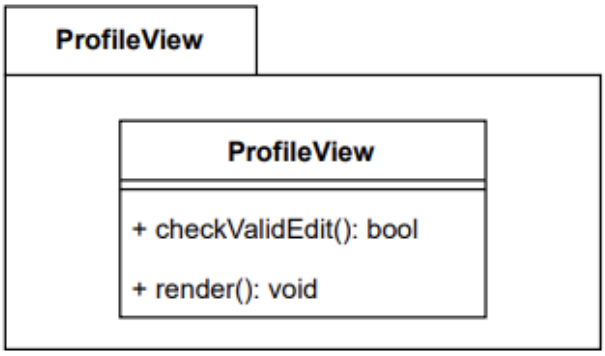

Figure 11. *The ProfileView Component*

- Explanation
    - Function **render()** of the **ProfileView** class helps display the view in the profile section to users.
    - Function **checkValidEdit()** of the **ProfileView** class can validate the input fields when users change their personal information.

4.3 Component: HomeView (View)

- Description
    - This component is responsible for showing the home page of the website to users.

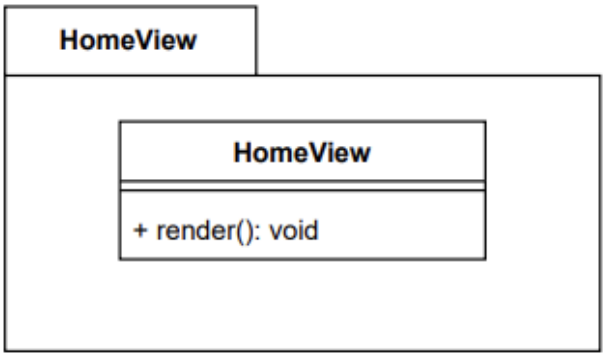

Figure 12. *The HomeView Component*

- Explanation
    - Function **render()** of the **HomeView** class helps display the view in the home page section to users.

4.4 Component: ShopServicesView (View)

- Description
    - This component is a collection of different types of views and is responsible for rendering a specific view of a page when users want to use that service. For instance, users from the book list on the shopping page can access the detailed information of a specific book they want to view.

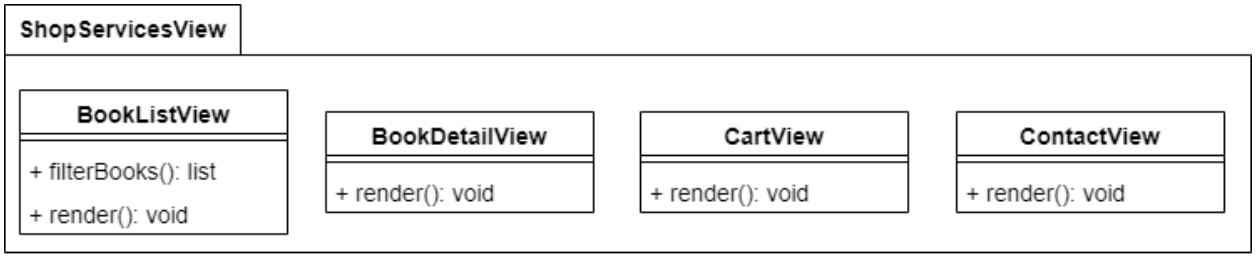

Figure 13. *The ShopServicesView*

- Explanation
    - Function **render()** in the **BookListView** class is used for displaying the lists of books when users access the shopping page.
    - Function **filterBooks()** in the **BookListView** class is used for filtering the books by different conditions such as prices or popularity.
    - Function **render()** in the **BookDetailView** class is used for displaying the detailed information of a book.
    - Function **render()** in the **CartView** class is used for displaying products which are added to the cart.
    - Function **render()** in the **ContactView** class is used for sending questions about services or concerns of the website.

4.5 Component: AdminAuthView (View)

- Description
    - This component is responsible for rendering the view of the authentication section if the admin tries to sign in.

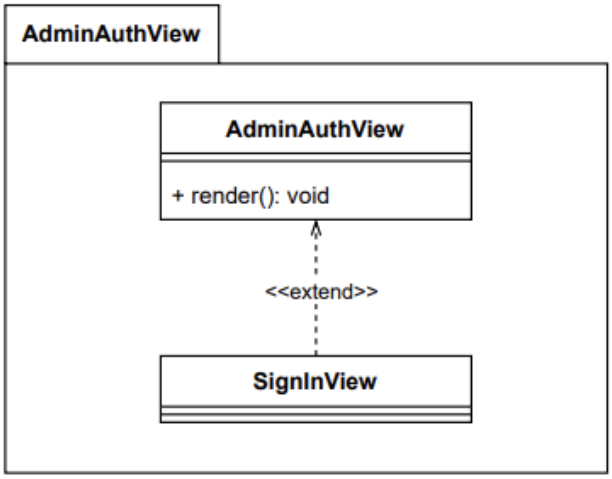

Figure 14. *The AdminAuthView Component*

- Explanation
  - Function **render()** of the **AdminAuthView** class helps display the view in the authentication section to the admin.
  - Derived the **SignInView** class will use the render() function from base class AdminAuthView to render a specific view for users.

4.6 Component: RevenueAnalysisView (View)

- Description
  - This component is responsible for rendering the view of the revenue statistics section in the admin dashboard. It is also a collection of render types depending on each specific purpose.

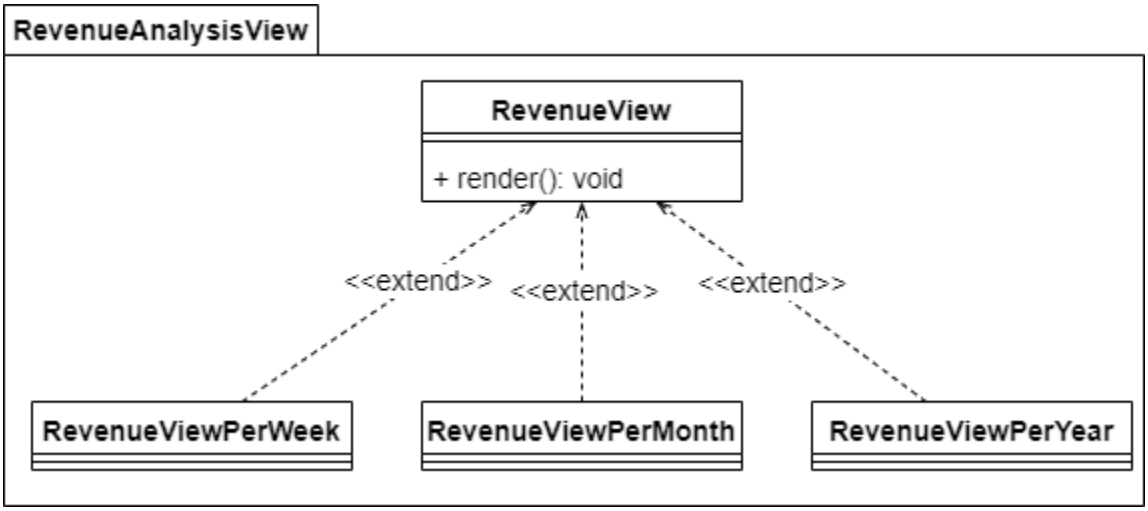

Figure 15. *The RevenueAnalysisView Component*

- Explanation
    - Function **render()** of the **RevenueAnalysisView** class helps display the view in the revenue statistics section to the admin.
    - Derived classes such as **RevenueViewPerWeek**, **RevenueViewPerMonth** and **RevenueViewPerYear** will use the **render()** function from the base class **RevenueAnalysisView** to render a specific view depending on the viewing purposes of the admin.

4.7 Component: BookManagementView (View)

- Description
    - This component is responsible for rendering the view of the management a book in the admin. If the admin wants to create or update a book or view the books in the store, the UI of those actions will be displayed.

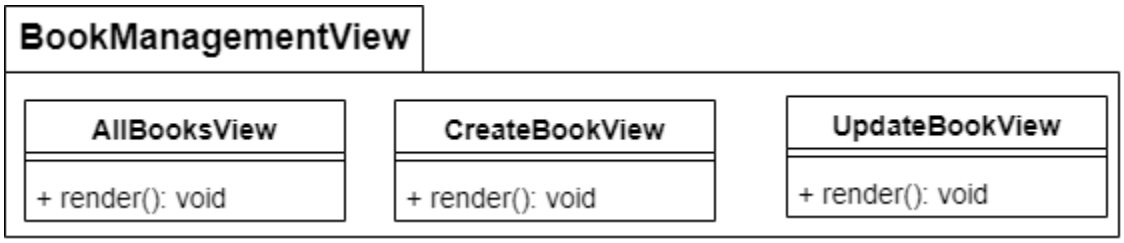

Figure 16. *The BookManagementView Component*

- Explanation
    - Function **render()** of the **AllBookView** class helps display the view of all books which exist in the shop for admin.
    - Function **render()** of the **CreateBookView** class helps display the view of a book when the admin creates a book (all input fields are blanks in popup form when creating a book).
    - Function **render()** of the **UpdateBookView** class helps display the view of a book when the admin updates a book (all input fields are displayed with the book's information - the book which is updated and the admin can change those input values).

4.8 Component: OrderManagementView (View)

- Description
    - This component is responsible for rendering the view of the management order or all orders in the admin. If the admin wants to update a status for an order view all orders or view a specific order from the users, the UI of those actions will be displayed.

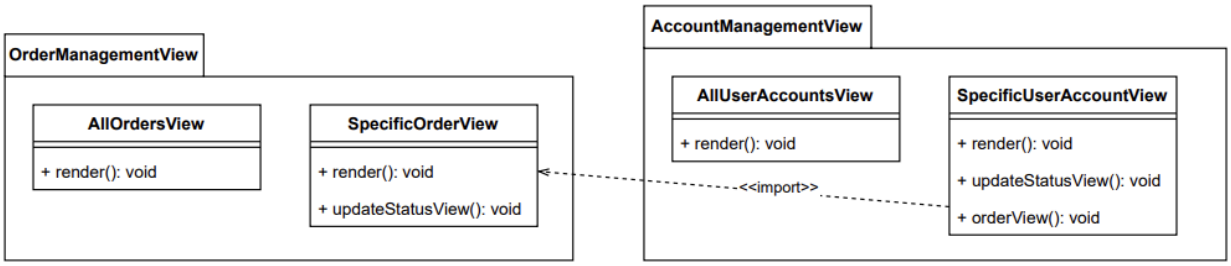

Figure 17. *The OrderManagementView Component*

- Explanation
  - Function **render()** in the **AllOrdersView** class is responsible for rendering the view of all orders coming from users of the website. In each order, the admin can view the details of it.
  - Function **render()** in the **SpecificOrderView** class is responsible for rendering the view of an order so that the admin can check or update the status of that order.
  - Function **updateStatusView()** in the **SpecificOrderView** class is responsible for updating the status of an order. For example, the admin can update the status of an order from unprocessed to confirmed.

4.9 Component: AccountManagementView (View)

- Description
  - This component is responsible for rendering the view of the management an account of the user or all accounts in the admin. If the admin wants to update a status for an account (Ban/ Unban) or view all accounts or view a specific account from the users, the UI of those actions will be displayed.

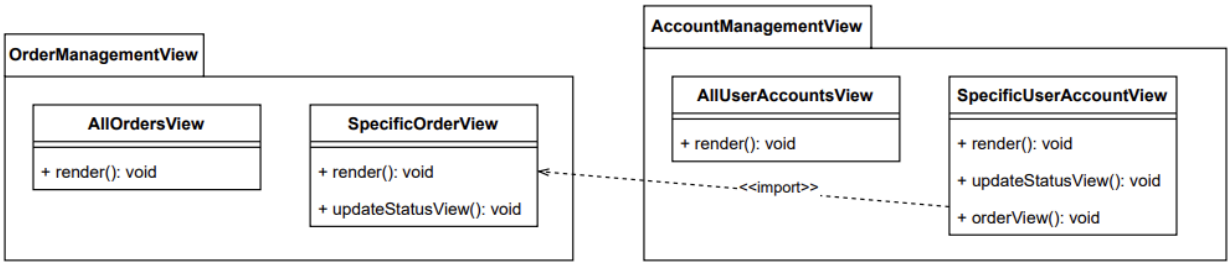

Figure 18. *The AccountManagementView Component*

- Explanation:
  - Function **render()** in the **AllUserAccountsView** class is responsible for rendering the view of all users' accounts. In each account, the admin can view the details of it or set a status for it.
  - Function **render()** in the **SpecificUserAccountView** class is responsible for rendering the view of an account so that the admin can view all orders of that account or update the status for that account too.
  - Function **updateStatusView()** in the **SpecificUserAccountView** class is responsible for updating the status of an account. For example, the admin can

update the status of an account from unban to ban and vice versa.

- Function **orderView()** in the **SpecificUserAccountView** class will display all orders which had already been paid previously in a specific account. In addition, if the admin wants to view the details or update the status of those orders he/ she can view it as when they view the details of a specific order in the **SpecificOrderView** class. Function **orderView()** imports the **SpecificOrderView** class.

4.10 Component: AccountController (Controller)

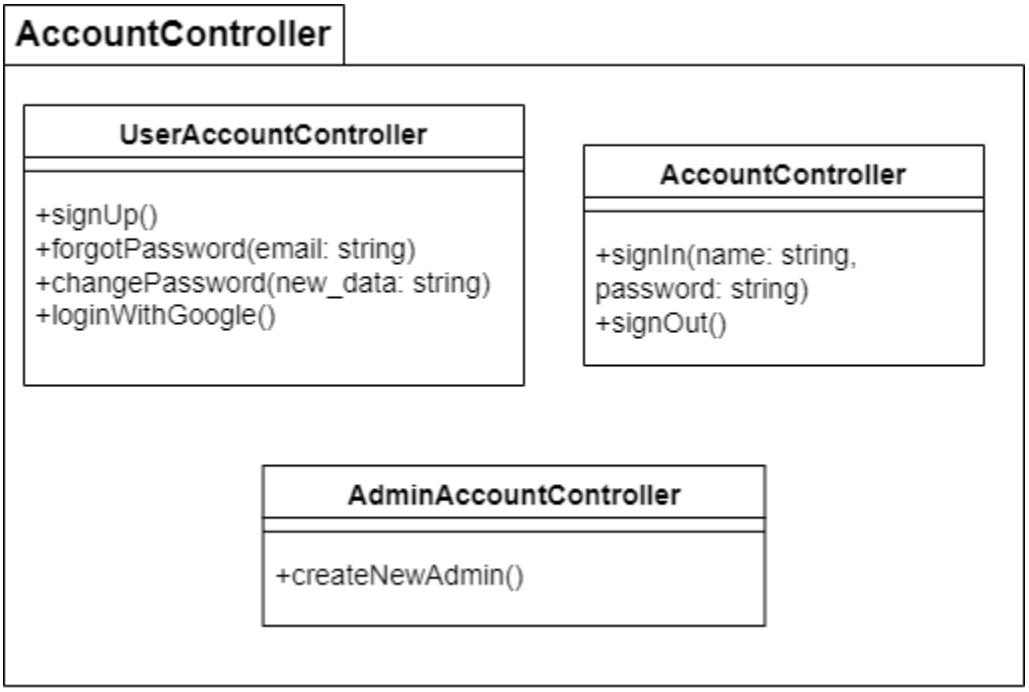

Figure 19. *The AccountController Component*

- Description
  - This component is responsible for reading and validating data from user accounts and admin account-related components.
- Explanation
  - The **"UserAccountController"** class
    - Description: The class handles navigation for users.
    - Attribute: None.
    - Method:
      - **signUp()**: The user signs up for a new account.
      - **forgotPassword(email: string)**: If the user forgets his/her account password, he/she can enter the email of the account to reset the password. A reset link will be sent to the registered email.
      - **changePassword(new_data: string)**: The user can change his/her

password after logging in to the system. This method is also called when the user gets a link from forgotPassword(email: string).
- **loginWithGoogle()**: Log into the system using a Google account.

- The **"AccountController"** class
  - Description: The class handles navigation for users and admin.
  - Attribute: None.
  - Method:
    - **signIn(name: string, password: string)**: This method checks whether the imputed name and string were correct. If the information is correct, the users can access the website by their account.
    - **signOut()**: Signs out.
- The **"AdminAccountController"** class
  - Description: The class handles navigation for admin.
  - Attribute: None.
  - Method:
    - **createNewAdmin()**: create a new account for Admin

4.11 Component: UserController (Controller)

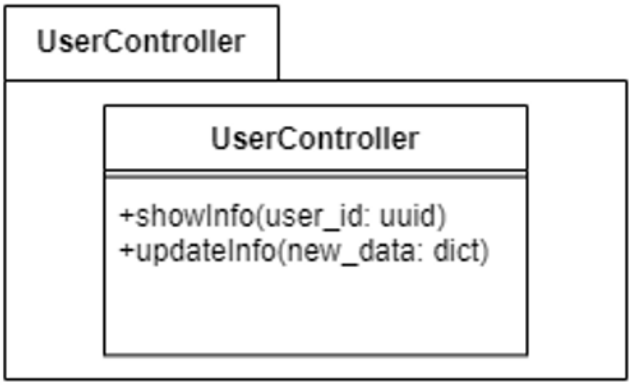

Figure 20. *The CommentController Component*

- Description
  - This component contains a user controller-related class.

- Explanation:
  - The **"UserController"** class
    - Description: The class handles navigation for users.
    - Attribute: None.
    - Method:
      - **showInfo(user_id: uuid)**: displays the user's personal information (full name, address, phone number, etc.).
      - **updateInfo(new_data: dict)**: updates the user's information with new data.

4.12 Component: OrderController (Controller)

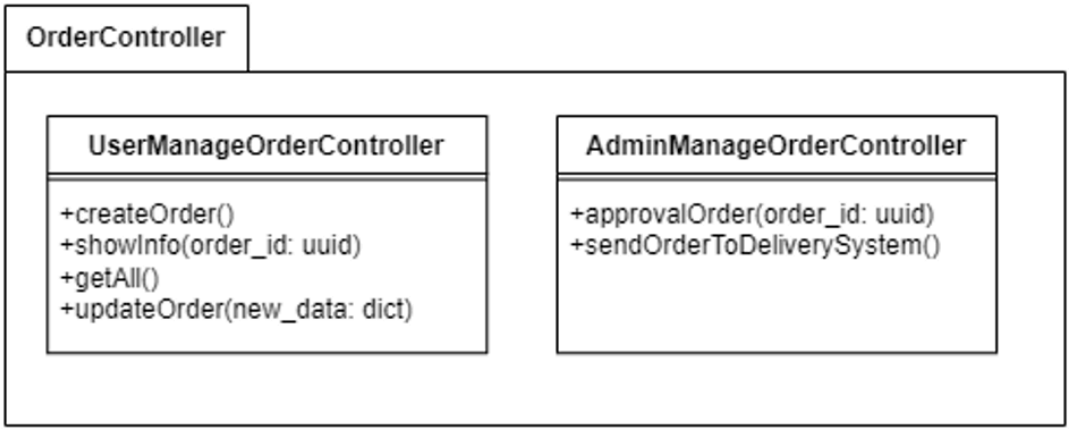

Figure 21. *The OrderController Component*

- Description
  - This component contains order controller-related classes.

- Explanation:
  - The **"UserManageOrderController"** class
    - Description: The class handles order management requests from users.
    - Attribute: None.
    - Method:
      - **createOrder()**: The user creates a new order.
      - **showInfo(order_id: uuid)**: displays the Order information, including the creation time, products in the Order, shipping fees, payment methods, total amount, etc.
      - **getAll()**: displays all Orders in the User's purchase history.
      - **updateOrder(new_data: dict)**: Adjusts some information of the Order (phone number, delivery address, etc.). This must be done before the Order status changes to 'Delivering'.
  - The **"AdminManageOrderController"** class
    - Description: The class handles order management requests from admin.
    - Attribute: None.
    - Method::
      - **approvalOrder(order_id: uuid)**: Admin checks the order information and store inventory, then chooses to approve or reject the order.
      - **sendOrderToDeliverySystem()**: Admin selects a logistics provider and hands over the order.

4.13 Component: CartController (Controller)

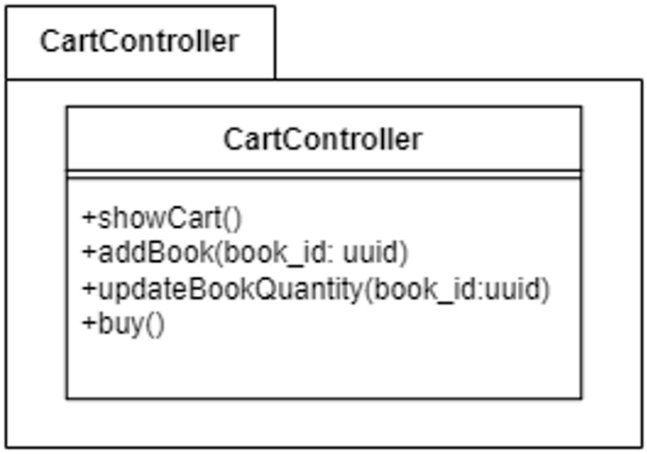

Figure 22. *The CartController Component*

- Description
  - This component contains a cart controller class.

- Explanation
  - The **"CartController"** class
    - Description: The class handles cart management requests from users.
    - Attribute: None.
    - Method:
      - **showCart()**: displays the products and their corresponding quantities in the User's cart.
      - **addBook(book_id: uuid)**: Adds a new product to the user's cart.
      - **updateBookQuantity(book_id:uuid)**: Adjusts the quantity of a product in the cart. If the quantity is reduced to 0, the system automatically removes the product from the cart.
      - **buy()**: Redirects to the payment page. After a successful payment, a new Order is created with the products in the user's cart and the order is sent to the Admin.

4.14 Component: BookController (Controller)

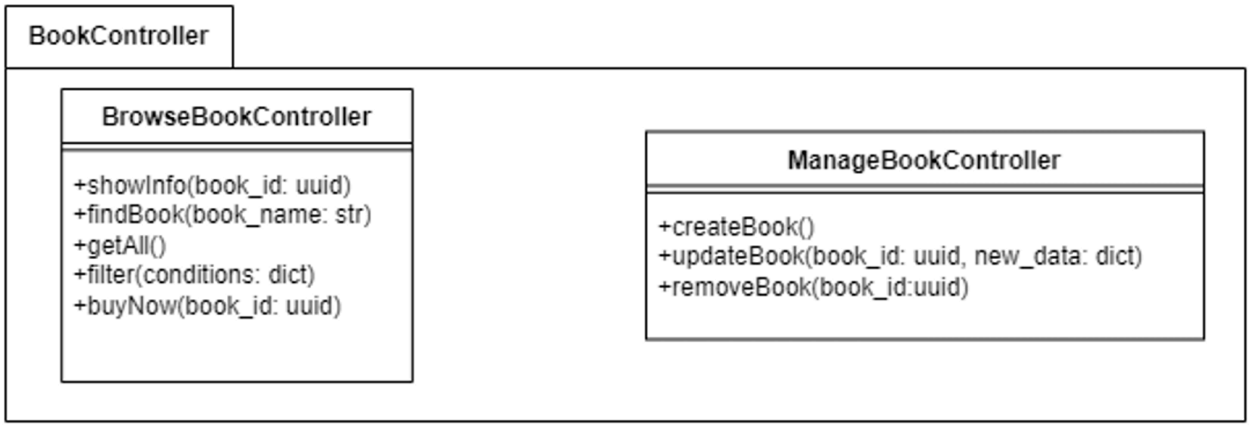

Figure 23. *The BookController Component*

- Description
  - This component contains book controller-related classes.

- Explanation
  - The **"BrowseBookController"** class
    - Description: The class handles book browsing requests from users.
    - Attribute: None.
    - Method:
      - **showInfo(book_id: uuid)**: displays information about a specific book, identified by the book_id stored in the system. The information displayed includes title, author, category, price, description, etc.
      - **findBook(book_name: str)**: searches for a book by its name. If the user enters an approximate name, the system will attempt to find products with similar names.
      - **getAll()**: Returns all books in the system.
      - **filter(conditions: dict)**: Searches for books based on multiple conditions simultaneously.
      - **buyNow(book_id: uuid)**: Redirects from the book viewing page directly to the payment page, by passing the cart.
  - The **"ManageBookController"** class
    - Description: The class handles book management requests from admin.
    - Attribute: None.
    - Method:
      - **createBook()**: Admin adds a new product to the system.
      - **updateBook(book_id: uuid, new_data: dict)**: Admin selects a specific book by its ID, then edits the book's information.
      - **removeBook(book_id:uuid)**: Admin selects a book to remove from the database.

4.15 Component: CommentController (Controller)

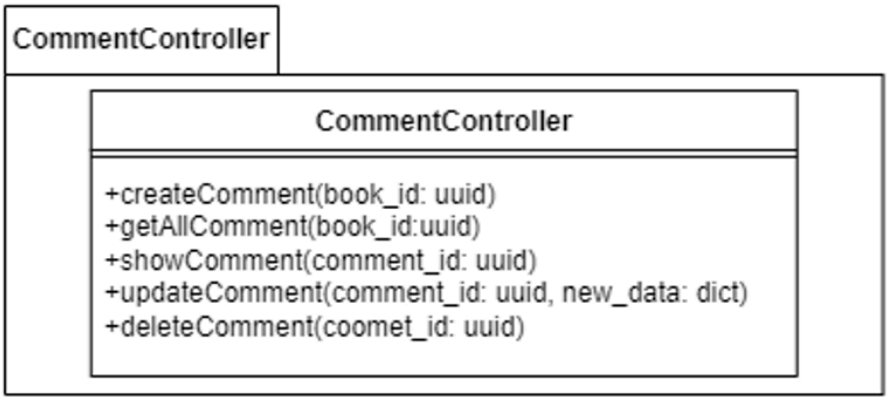

Figure 24. *The CommentController Component*

- Description
  - This component contains a comment controller class.

- Explanation
  - The **"CommentController"** class
    - Description: The class handles comment management requests from users.
    - Attribute: None.
    - Method:
      - **createComment(book_id: uuid)**: creates a new comment for a book.
      - **getAllComment(book_id:uuid)**: retrieves all comments for a book.
      - **showComment(comment_id: uuid)**: displays detailed information about a comment.
      - **updateComment(comment_id: uuid, new_data: dict)**: edits the text in a comment.
      - **deleteComment(coomet_id: uuid)**: deletes a comment from the system.

4.16 Component: PaymentController (Controller)

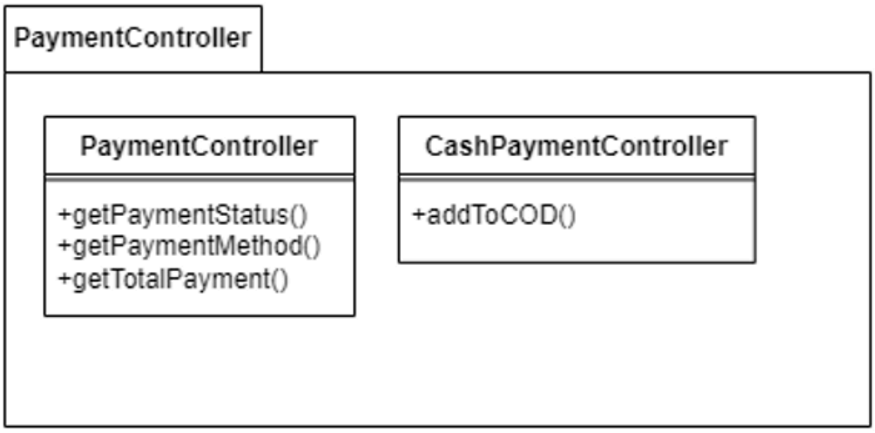

Figure 25. *The PaymentController Component*

- Description
  - This component contains payment controller-related classes.

- Explanation
  - The **"PaymentController"** class
    - Description: The class handles online payment requests from users.
    - Attribute: None.
    - Method:
      - **getPaymentStatus()**: displays the status of the payment (not paid, paid, pending confirmation from Admin/Bank, etc.).
      - **getPaymentMethod()**: displays the payment method chosen by the user (E-wallet, Cash).
      - **getTotalPayment()**: displays the total amount to be paid.
  - The **"CashPaymentController"** class
    - Description: The class handles payment requests from users who want to pay cash.
    - Attribute: None.
    - Method:
      - **addToCOD()**: Adds to the list of orders to be paid upon delivery. Since these orders are not paid at the time of ordering, they can be given lower priority to process the already paid orders first, minimizing the need for refunds when stock runs out.

4.17 Component: AddressController (Controller)

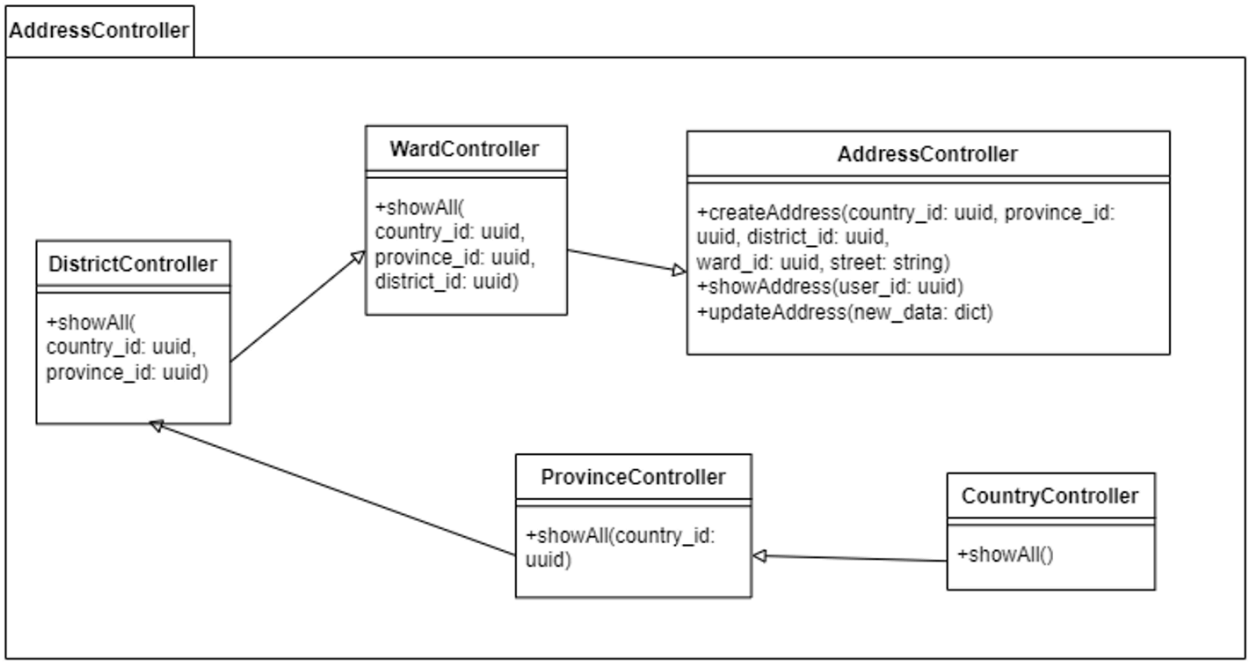

Figure 26. *The AddressController Component*

- Description
  - This component contains address controller-related classes.

- Explanation
  - The **"CountryController"** class
    - Description: The class handles navigation for users.
    - Attribute: None.
    - Method:
      - **showAll()**: displays all countries supported by the system.
  - The **"ProvinceController"** class
    - Description: The class handles navigation for users.
    - Attribute: None.
    - Method:
      - **showAll(country_id: uuid)**: Displays all provinces/cities of the selected country. If no country is selected, it returns nothing.
  - The **"DistrictController"** class
    - Description: The class handles navigation for users.
    - Attribute: None.
    - Method:
      - **showAll(country_id: uuid, province_id: uuid)**: Displays all districts of the province/city belonging to the selected country. If no country or province/city is selected, it returns nothing.
  - The **"WardController"** class

- Description: The class handles navigation for users.
- Attribute: None.
- Method:
  - **showAll(country_id: uuid, province_id: uuid, district_id: uuid)**: Displays all wards of the district in the province/city belonging to the selected country.

- The **"AddressController"** class
  - Description: The class handles navigation for users.
  - Attribute: None.
  - Method:
    - **createAddress(country_id: uuid, province_id: uuid, district_id: uuid, ward_id: uuid, street: string)**: Creates a specific address from the previously selected country, province/city, district, and ward, along with a specific street and house number.
    - **showAddress(user_id: uuid)**: Displays the user's address.
    - **updateAddress(new_data: dict)**: Updates the user's address.

4.18 Component: VoucherController (Controller)

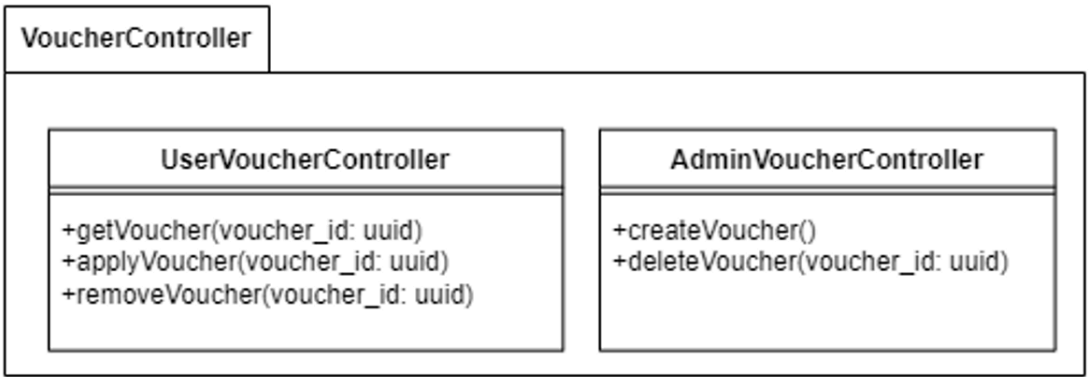

Figure 27. *The VoucherController Component*

- Description
  - This component contains voucher controller-related classes.

- Explanation
  - The **"UserVoucherController"** class
    - Description: The class handles navigation for users.
    - Attribute: None.
    - Method:
      - **getVoucher(voucher_id: uuid)**: User searches for a voucher.
      - **applyVoucher(voucher_id: uuid)**: User applies a voucher to an order.
      - **removeVoucher(voucher_id: uuid)**: User cancel using a voucher in an order.
  - The **"AdminVoucherController"** class

- Description: The class handles navigation for admin.
- Attribute: None.
- Method:
  - **createVoucher()**: Admin creates a new voucher
  - **deleteVoucher(voucher_id: uuid)**: Admin deletes a voucher from the system

4.19 Component: AccountService (Service)

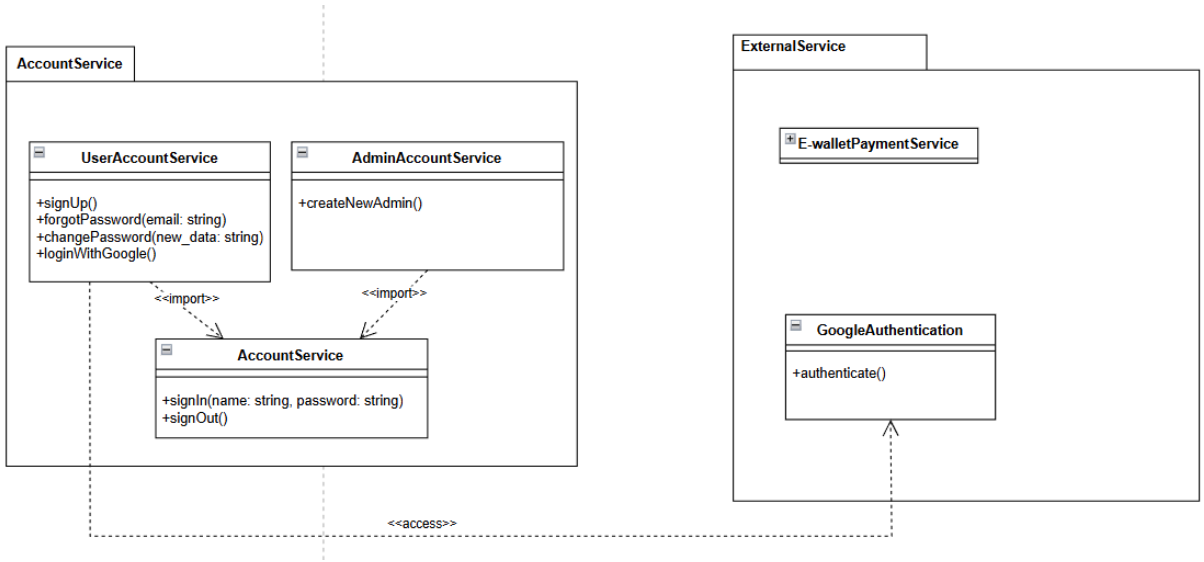

Figure 28. *The AccountService Component*

- Description: These are services related to the Account.
- Explanation
  - The **"AccountService"** class
    - Description: These are common services that both Admin and User can import and use.
    - Attribute: None.
    - Method:
      - **signIn(name: string, password: string)**: checks whether the entered name and password are valid or not, if so, allows login and access to services corresponding to the role in the system.
      - **signOut()**: log out of the system.
  - The **"UserAccountService"** class
    - Description: services for **UserAccount**. Additionally, it includes services imported from **AccountService** and services accessed from the **GoogleAuthentication** component.
    - Attribute: None.
    - Method:
      - **signUp()**: The user registers a new account on the system.

- **forgotPassword(email: string)**: The user enters the email of the account to reset the password. A reset link will be sent to the registered email.
- **changePassword(new_data: string)**: Changes the current password to a new password. This feature can only be performed after logging in or via the reset link from the forgotPassword email.
- **loginWithGoogle()**: Log into the system using a Google account.

○ The **"AdminAccountService"** class

■ Description: Services for AdminAccount, including services imported from AccountService.

■ Attribute: None.

■ Method:

- **createNewAdmin()**: Creates a new Admin account. This can only be performed after authentication (logged in) as an Admin.

4.20 Component: UserService (Service)

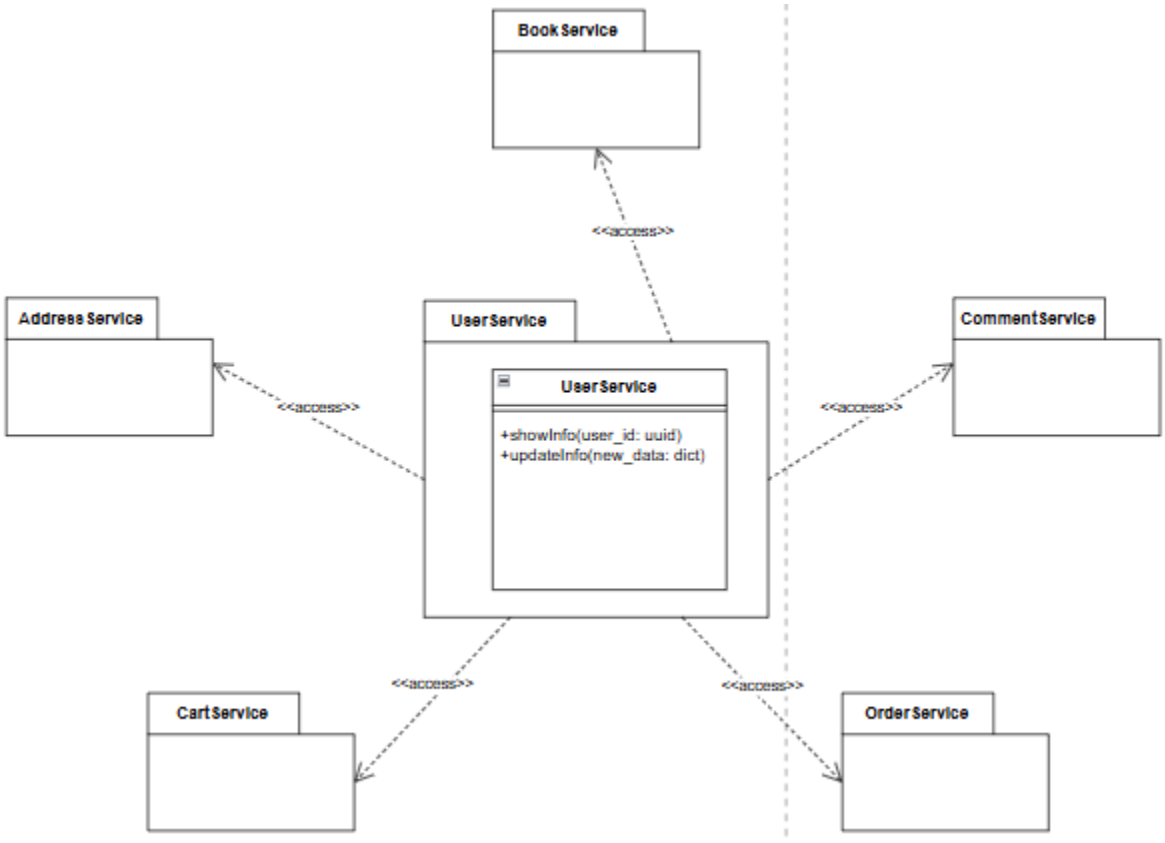

Figure 29. *The UserService Component*

- Description

- These are services for Users. Additionally, there are services accessed from other components.

- Explanation
  - The **"UserService"** class will not have attributes, but it has 2 methods:
    - **showInfo(user_id: uuid)**: displays the user's personal information (Full name, address, phone number, etc.).
    - **updateInfo(new_data: dict)**: updates the user's information with new data.

4.21 Component: OrderService (Service)

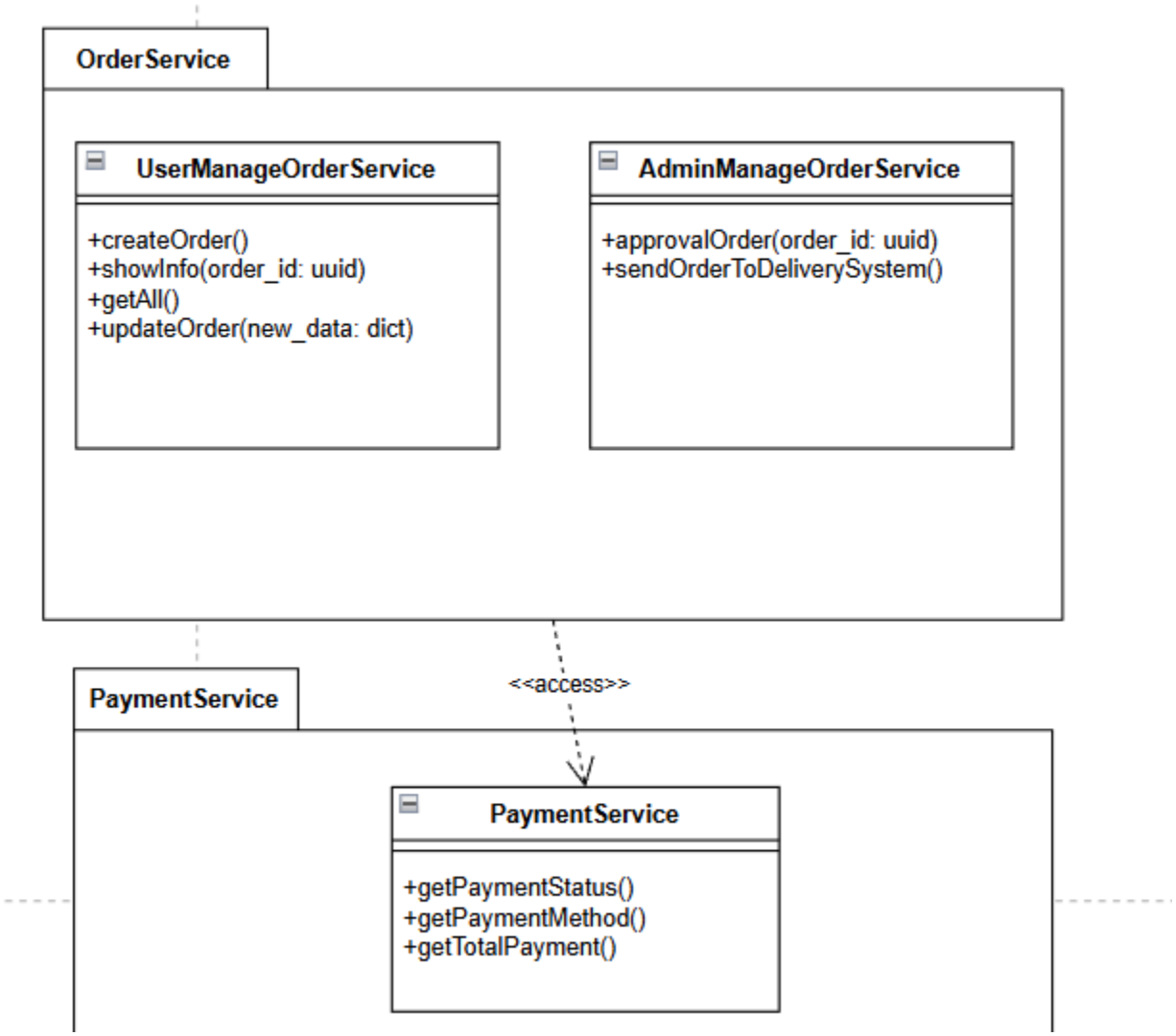

Figure 30. *The OrderService Component*

These are services for the **Order** and services accessed from PaymentService.

1. The **“UserManageOrderService”** class

- Description: Order services in the User role.
- Attribute: None.
- Method:
  - **createOrder()**: Creates a new Order.
  - **showInfo(order_id: uuid)**: Displays the Order information, including the creation time, products in the Order, shipping fees, payment methods, total amount, etc.
  - **getAll()**: Displays all Orders in the User's purchase history.
  - **updateOrder(new_data: dict)**: Adjusts some information of the Order (phone number, delivery address, etc.). This must be done before the Order status changes to 'Delivering'.

2. The **“AdminManageOrderService”** class

- Description: Order services in the Admin role.
- Attribute: None.
- Method:
  - **approvalOrder(order_id: uuid)**: Admin checks the order information and store inventory, then chooses to approve or reject the order.
  - **sendOrderToDeliverySystem()**: Admin selects a logistics provider and hands over the order.

4.22 Component: CartService (Service)

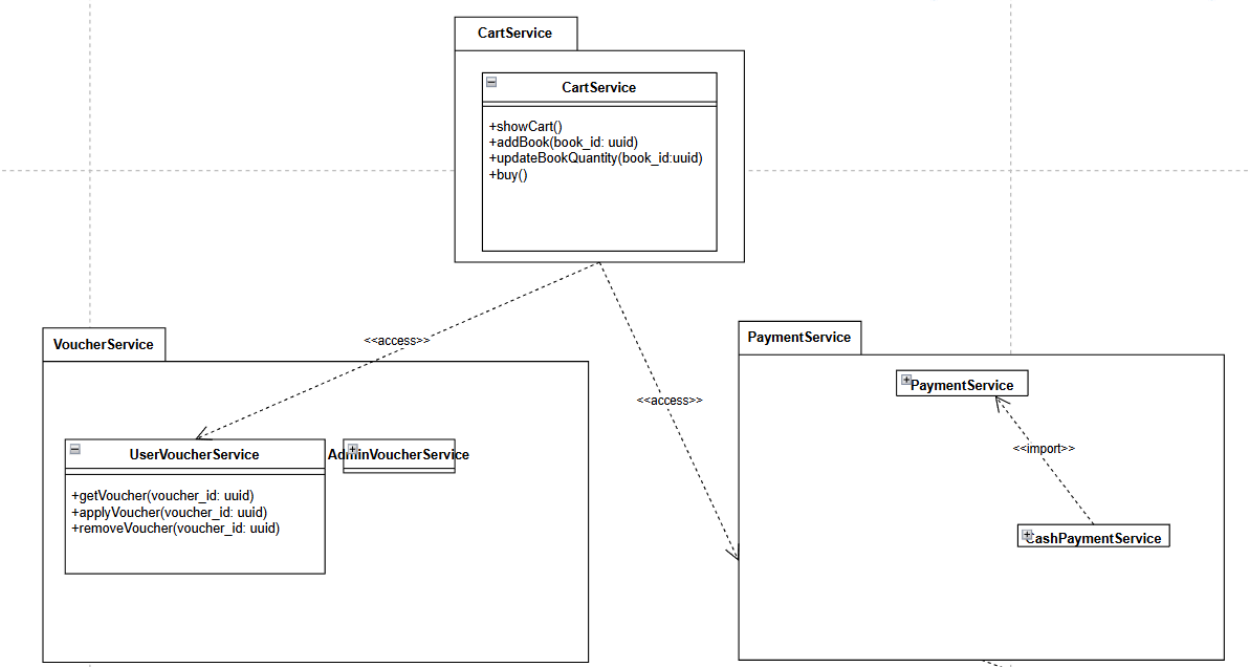

Figure 31. *The CartService Component*

- Description: These are services for the User's Cart and services accessed from the UserVoucherService component.
- Attribute: None.
- Methods:
    - **showCart()**: Displays the products and their corresponding quantities in the User's cart.
    - **addBook(book_id: uuid)**: Adds a new product to the User's cart.
    - **updateBookQuantity(book_id: uuid)**: Adjusts the quantity of a product in the cart. If the quantity is reduced to 0, the system automatically removes the product from the cart.
    - **buy()**: Redirects to the payment page. After a successful payment, a new Order is created with the products in the User's cart and the Order is sent to the Admin.

4.23 Component: BookService (Service)

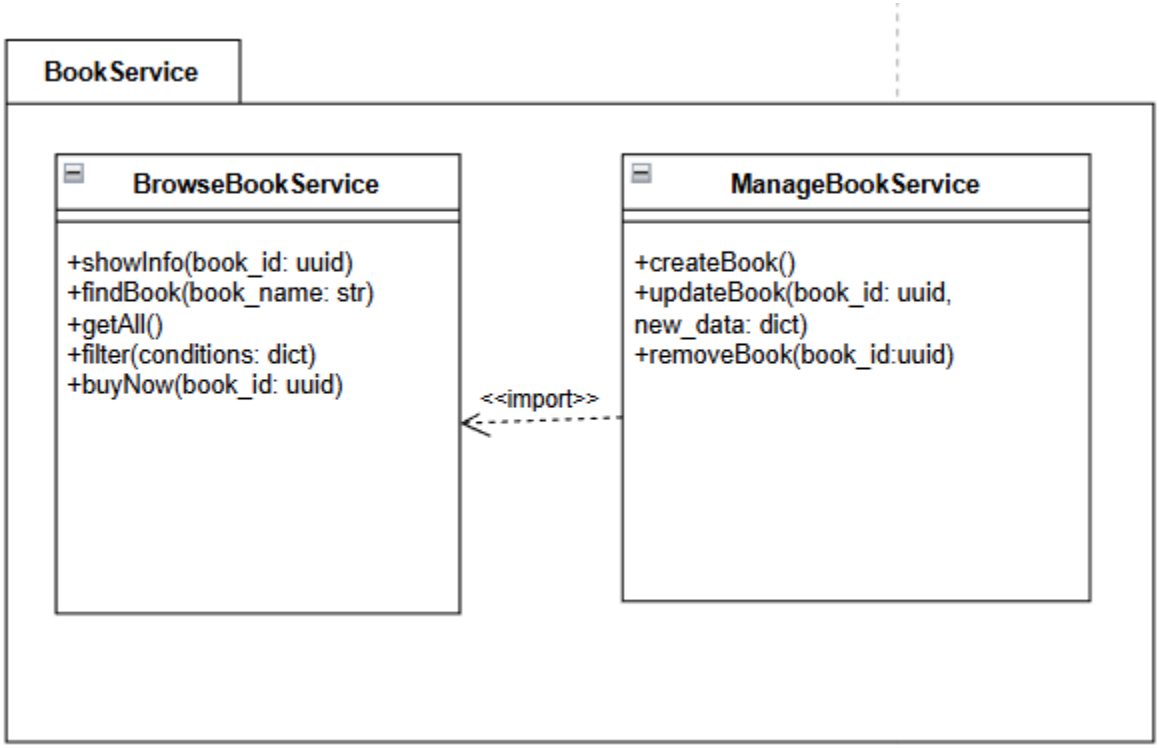

Figure 32. *The BookService Component*

Description: These are services for Books in the Admin and User roles.

1. The **"BrowseBookService"** class

- Description: Book services in the role of User.
- Attribute: None.
- Method:
  - **showInfo(book_id: uuid)**: Displays information about a specific book, identified by the book_id stored in the system. The information displayed includes title, author, category, price, description, etc.
  - **findBook(book_name: str)**: Searches for a book by its name. If the user enters an approximate name, the system will attempt to find products with similar names.
  - **getAll()**: Returns all books in the system.
  - **filter(conditions: dict)**: Searches for books based on multiple conditions simultaneously.
  - **buyNow(book_id: uuid)**: Redirects from the book viewing page directly to the payment page, by passing the cart.

2. The **"ManageBookService"** class

- Description: These are Book services in the role of Admin, including services imported

from the BrowseBookService component.

- Attribute: None.
- Method:
  - **createBook()**: Admin adds a new product to the system.
  - **updateBook(book_id: uuid, new_data: dict)**: Admin selects a specific book by its Id, then edits the book's information.
  - **removeBook(book_id: uuid)**: Admin selects a book to remove from the database.

4.24 Component: CommentService (Service)

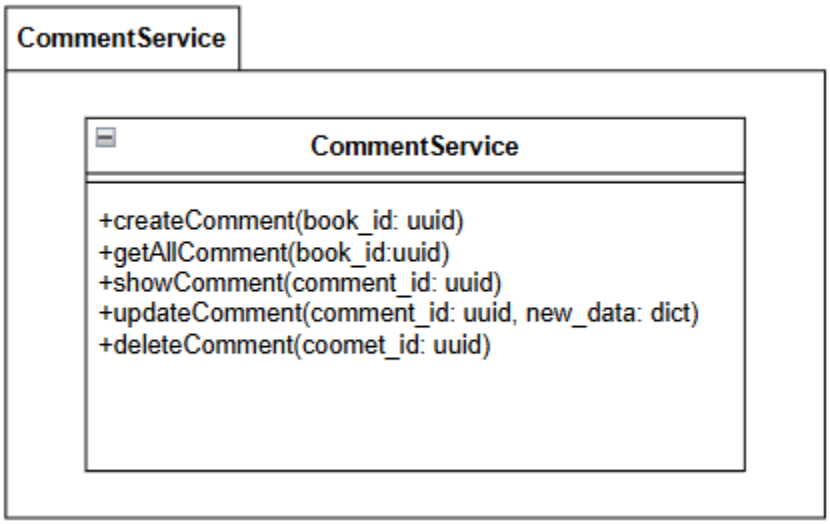

Figure 33. *The CommentService Component*

- Description: services for the **CommentService** component.
- Attribute: None.
- Method:
  - **createComment(book_id: uuid)**: Creates a new comment for a book.
  - **getAllComment(book_id: uuid)**: Retrieves all comments for a book.
  - **showComment(comment_id: uuid)**: Displays detailed information about a comment.
  - **updateComment(comment_id: uuid, new_data: dict)**: Edits the information of a comment.
  - **deleteComment(comment_id: uuid)**: Deletes a comment from the system.

4.25 Component: PaymentService (Service)

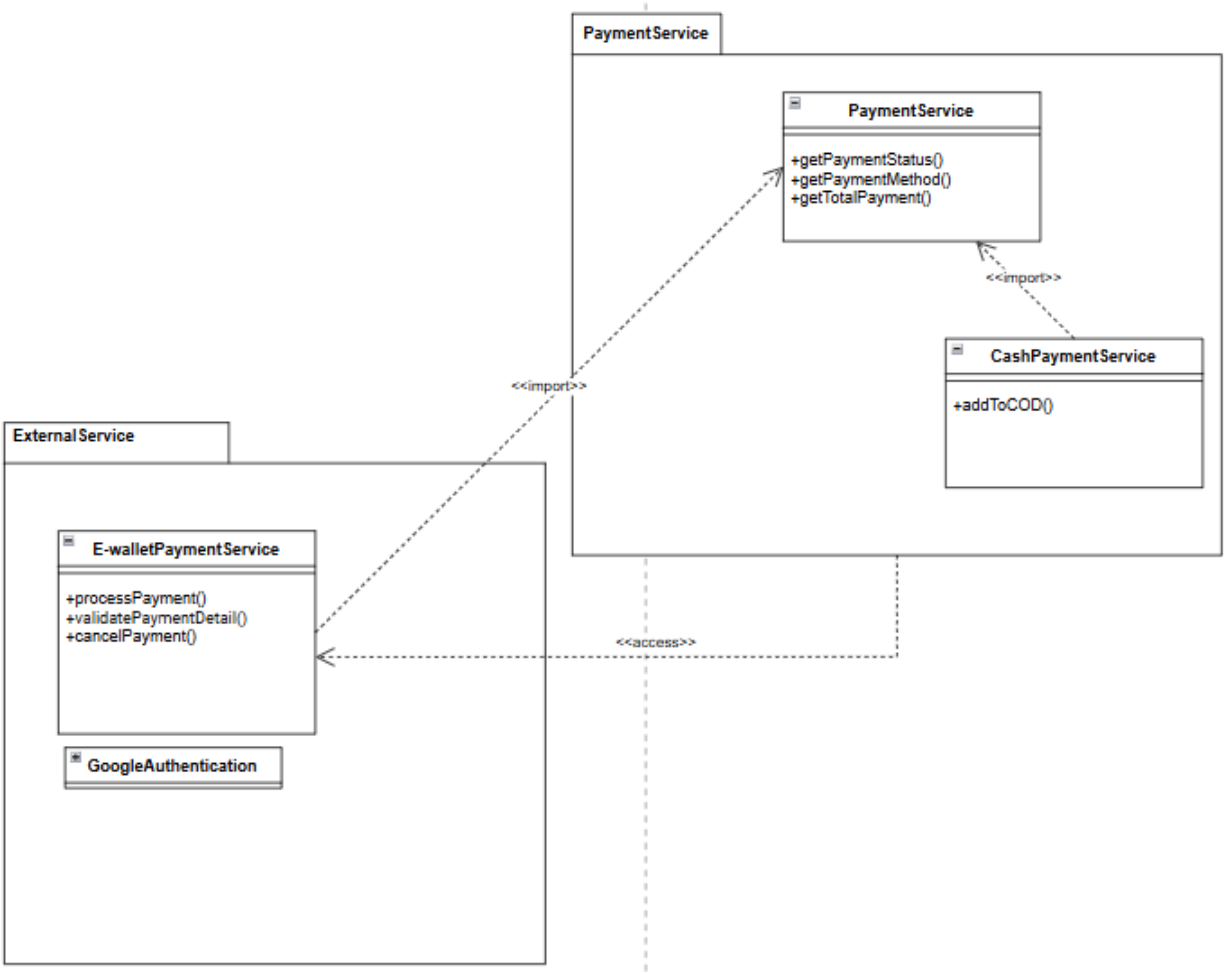

Figure 34. *The PaymentService Component*

- Description
  - These are services for Payment, including **E-walletPaymentService** and **CashPaymentService**.

1. The **"PaymentService"** class

- Description
  - These are common services of **PaymentService**.
- Attribute: None.
- Method:
  - **getPaymentStatus()**: Displays the status of the payment (not paid, paid, pending confirmation from Admin/Bank, etc.).
  - **getPaymentMethod()**: Displays the payment method chosen by the user (E-wallet, Cash).

- **getTotalPayment()**: Displays the total amount to be paid.

2. The **"CashPaymentService"** class

- Description
  - These are services when the user chooses to pay with cash, and it also include services imported from the PaymentService component.
- Attribute: None.
- Method:
  - **addToCOD()**: Adds to the list of orders to be paid upon delivery. Since these orders are not paid at the time of ordering, they can be given lower priority to process the already paid orders first, minimizing the need for refunds when stock runs out.

4.26 Component: AddressService (Service)

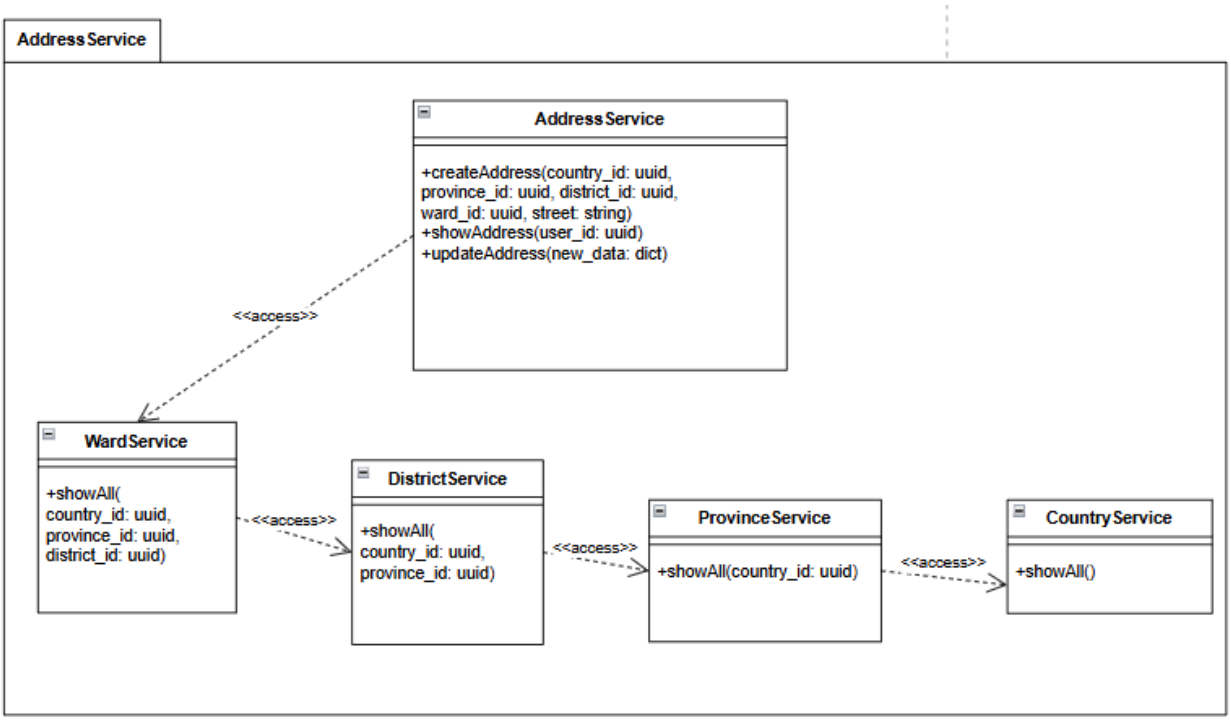

Figure 35. *The AddressService Component*

- Description
    - Below are services related to Addresses, where each lower-level locality can access services from higher-level localities.

1. The **"CountryService"** class

- Attribute: None.
- Method
    - **showAll()**: displays all countries supported by the system.

2. The **"ProvinceService"** class

- Attribute: None.
- Method
    - **showAll(country_id: uuid)**: Displays all provinces/cities of the selected country. If no country is selected, it returns nothing.

3. The **"DistrictService"** class

- Attribute: None.
- Method
    - **showAll(country_id: uuid, province_id: uuid)**: Displays all districts of the province/city belonging to the selected country. If no country or province/city is selected, it returns nothing.

4. The **"WardService"** class

- Attribute: None.
- Method:
    - **showAll(country_id: uuid, province_id: uuid, district_id: uuid)**: Displays all wards of the district in the province/city belonging to the selected country.

5. The **"AddressService"** class

- Attribute: None.
- Method
    - **createAddress(country_id: uuid, province_id: uuid, district_id: uuid, ward_id: uuid, street: string)**: Creates a specific address from the previously selected country, province/city, district, and ward, along with a specific street and house number.
    - **showAddress(user_id: uuid)**: Displays the user's address.
    - **updateAddress(new_data: dict)**: Updates the user's address.

4.27 Component: VoucherService (Service)

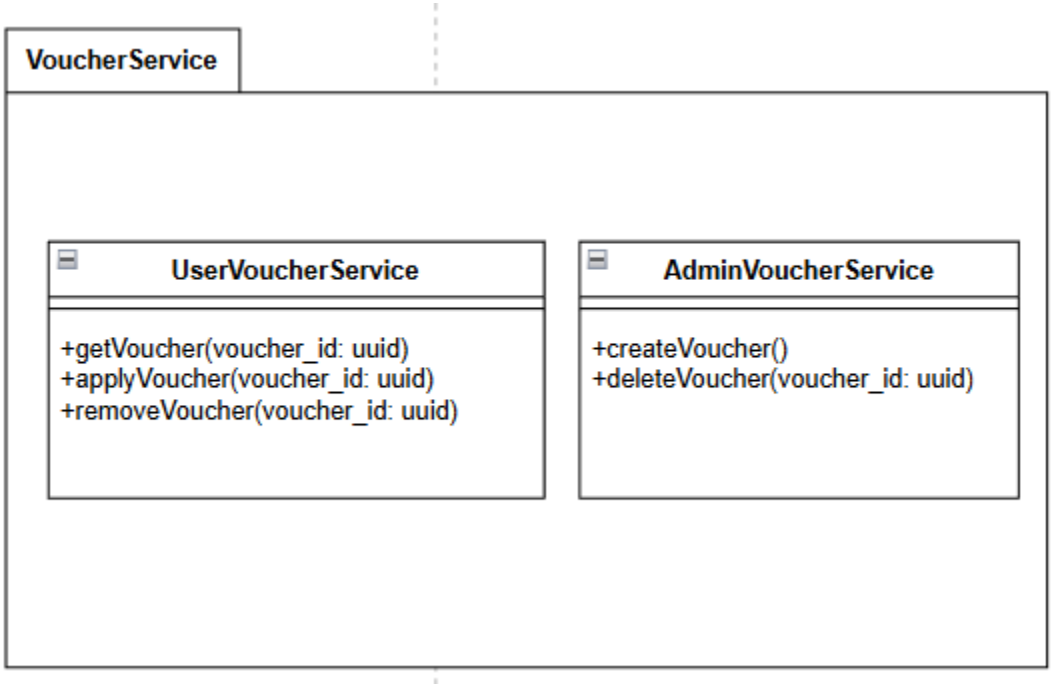

Figure 36. *The VoucherService Component*

These are services related to Vouchers from the perspectives of Admin and User.

1. The **"AdminVoucherService"** class

- Description
  - Services related to Vouchers from the perspective of Admin.
- Attribute: None.
- Method
  - **createVoucher()**: Admin creates a new voucher.
  - **deleteVoucher(voucher_id: uuid)**: Admin deletes a voucher from the system.

2. The **"UserVoucherService"** class

- Description
  - Services related to Vouchers from the perspective of the User.
- Attribute: None.
- Method
  - **getVoucher(voucher_id: uuid)**: Searches for a Voucher in the system.
  - **applyVoucher(voucher_id: uuid)**: Applies a Voucher to an order.
  - **removeVoucher(voucher_id: uuid)**: Removes a selected Voucher from an order.

4.28 Component: E-walletPaymentService (External Service)

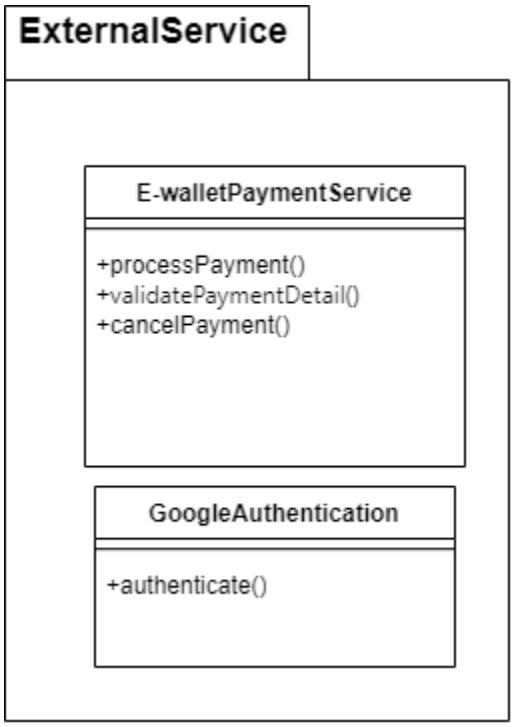

Figure 37. *The E-walletPaymentService Component*

- Description
  - Below are services when the user chooses to pay with an e-wallet. It also includes services imported from the PaymentService component.
- Attribute: None.
- Method
  - **processPayment()**: Processes the transaction by calling the bank, e-wallet, and other APIs.
  - **validatePaymentDetail()**: Checks if the user-entered information is valid (e-wallet account, password, etc.).
  - **cancelPayment()**: Cancels the payment (change of mind, choosing another method, adjusting the cart, etc.).

4.29 Component: GoogleAuthentication (External Service)

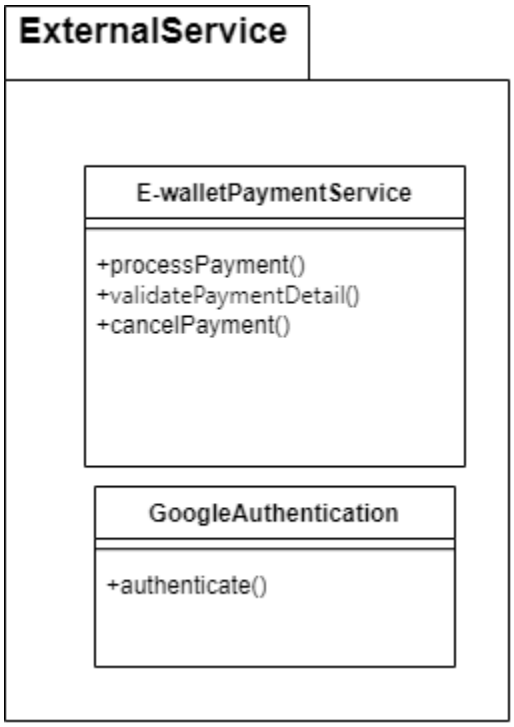

Figure 38. *The GoogleAuthentication Component*

- Description
  - These are services related to account authentication through Google.
- Attribute: None.
- Method
  - **authenticate()**: Uses Google's API to authenticate the account and log in to the system.

4.30 Component: AccountModel (Model)

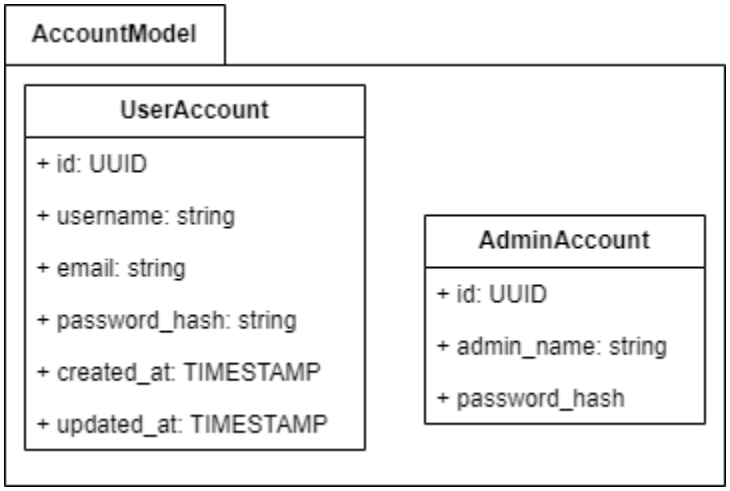

Figure 39. *The AccountModel Component*

- Description
  - This component will encapsulate all attributes of the user account and admin account.
    - **"UserAccount"** class handles general user-related data, such as username, email, and password.
    - **"AdminAccount"** class focuses on administrative users, such as an admin-specific identifier, admin name, and password.

- Explanation
  - The **"UserAccount"** class have some attributes and no methods:
    - **"id"** has type UUID, which is a unique identifier for each user account.
    - **"username"** is the name chosen by the user for signing in.
    - **"email"** address associated with the user account. This is often used for account recovery, communication and sìgn in.
    - **"password_hash"** should be hashed for security reasons to prevent plain text storage.
    - **"created_at"** is the date and time when the user account was created. This can be used for record-keeping purposes and auditing purposes.
    - **"updated_at"** is the date and time when the user account was last updated. This could reflect changes to the user's information, such as a password change or profile update.

  - The **"AdminAccount"** class have some attributes and no methods:
    - **"id"** is a unique identifier for each admin account.
    - **"admin_name"** is the name of the admin account used by the admin for signing in or identifying themselves within the administrative system.
    - **"password_hash"** is the hashed password associated with the admin

account. Security is crucial for admin accounts, so passwords must be hashed.

4.31 Component: UserModel (Model)

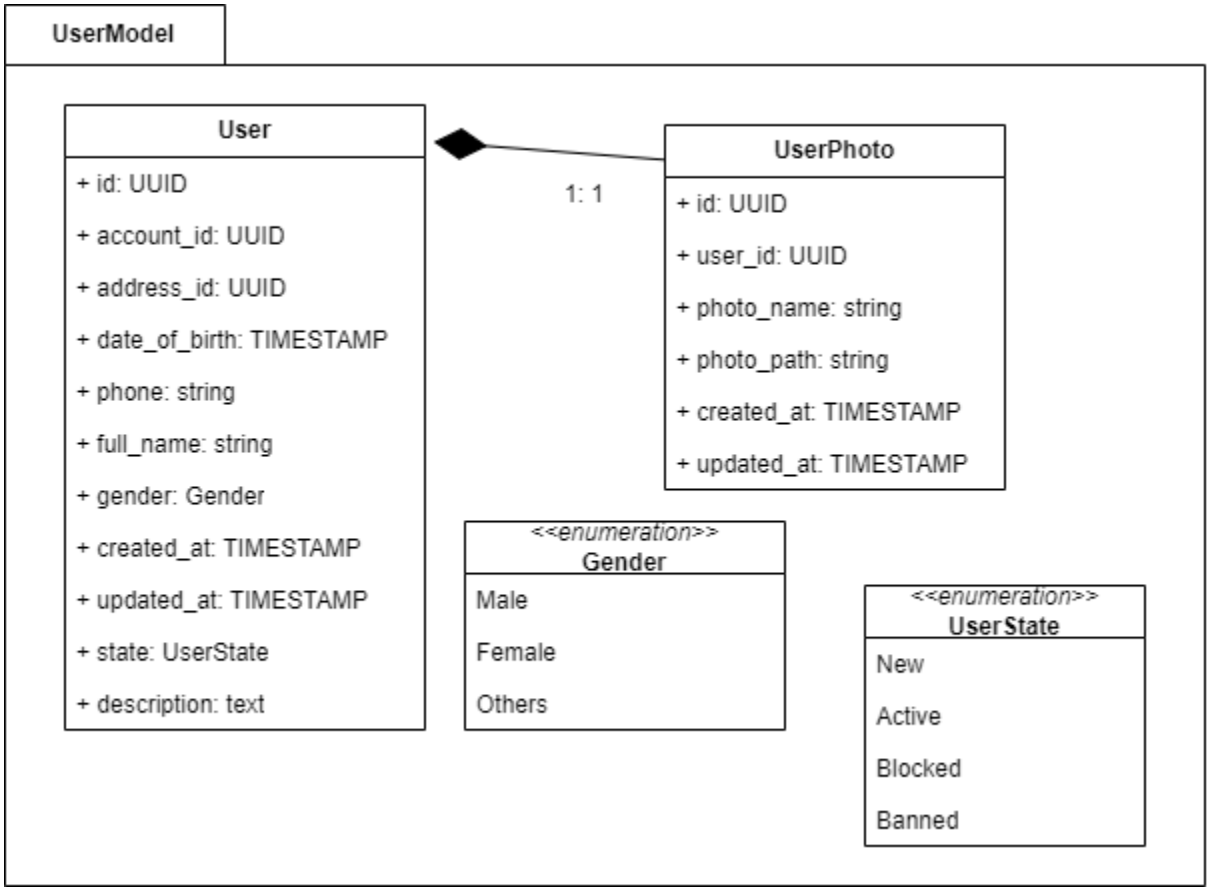

Figure 40. *The UserModel Component*

- Description
  - This component will contain all the information about the user profile in the online bookstore application.
  - The **"UserModel"** component will access the **"AccountModel"** and **"VoucherModel"** components, which represent the user account details and the comments or reviews made by the user.
  - This component will also import the **"AddressModel"**, which represents the address associated with the user.

- Explanation:
  - The **"User"** class represents the primary entity for storing user information. This class have some attributes and no methods:
    - **"id"** - UUID: this unique identifier is crucial for distinguishing each user within the system.

- The **"account_id"** attribute will link the user to their account, which may include login credentials and payment information.
- The **"address_id"** attribute will point to the user's address.
- The **"date_of_birth"** will be used for age verification and personalized marketing.
- **"phone"** is the phone number of the user, which facilitates customer support and order notifications.
- **"full name"** is the complete name of the user, used in personalization and official communication.
- **"gender"** is an enumeration capturing the user's gender for personalized recommendations.
- **"created_at"** records when the user was added to the system.
- **"updated_at"** tracks the last update made to the user's profile.
- **"state"** will indicate the user's status, which affects their access and interaction with the system.
- **"description"** is additional information about the user, such as preferences or special notes.

- The **"UserPhoto"** class manages user profile pictures, enhancing user experience by providing a personalized interface.
  - **"id"** is the unique identifier for each photo entry.
  - **"user_id"** will link the photo to a specific user.
  - **"photo_name"** is the name of the photo file.
  - **"photo_path"** is the file path where the photo is stored.
  - **"created_at"** records when the photo was uploaded.
  - **"updated_at"** tracks the last modification date of the photo.

- These enumerations standardize the values for specific attributes, ensuring consistency across the system.
  - Gender - **"Male"**, **"Female"**, and **"Others"**.
  - UserState:
    - **New** - A user who has recently registered.
    - **Active** - A user who is currently active on the platform.
    - **Blocked** - A user who is temporarily restricted from accessing the platform.
    - **Banned** - A user permanently banned from the platform.

- There is a 1:1 (one-to-one) relationship between **"User"** and **"UserPhoto"**.This means that each user can have one associated photo, and ach photo is linked to one user. This relationship ensures that every user can upload a profile picture, which is uniquely identified and stored with a reference back to the user.

4.32 Component: AddressModel (Model)

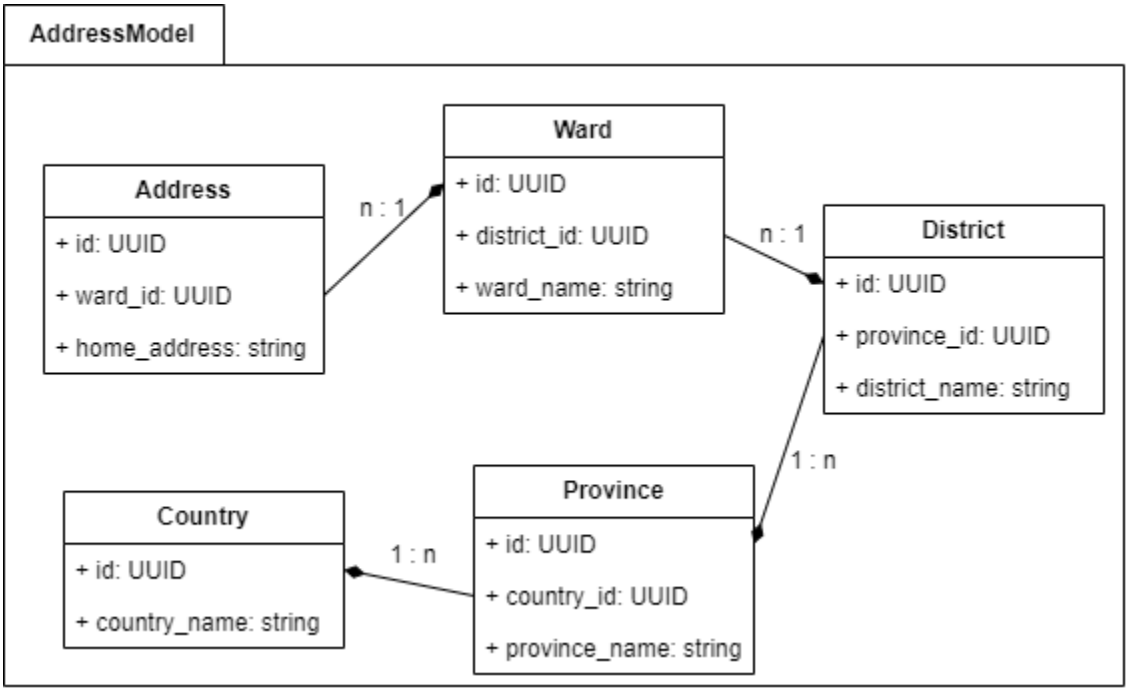

Figure 41. *The AddessModel Component*

- Description
  - This component will store hierarchical address data, including countries, provinces, districts, wards, and specific home addresses. This structured format ensures that addresses are stored in a detailed and organised manner.

- Explanation
  - The **"Address"** class represents the main entity for storing specific address details. One ward can have multiple addresses, but each address belongs to one ward.
    - The **"id"** attribute is a unique identifier for the address entry.
    - The **"ward_id"** is a foreign key linking to the "Ward" class where the address is located.
    - **"home_address"** is the specific home address, such as street name, building number, and apartment number.

  - The **"Ward"** class captures details about the ward, a subdivision of a district. One district can contain multiple wards, but each ward belongs to one district.
    - **"id"** is a unique identifier for the ward.
    - **"district_id"** is a foreign key linking to the District that contains the ward.
    - **"ward_name"** is the name of the ward.

  - The **"District"** class holds information about a district, which is a larger

administrative division containing multiple wards.

- **“id”** is a unique identifier for the district.
- **“province_id”** is a foreign key linking to the Province that includes the district.
- **“district_name”** is the name of the district.

- The **“Province”** class details information about a province, an administrative division containing several districts. One province can contain multiple districts, but each district belongs to one province.
  - **“id”** is a unique identifier for the province.
  - **“country_id”** is a foreign key linking to the **Country** that the province belongs to.
  - **“province_name”** is the name of the province.
- The **“Country”** class represents information about a country. One country can have multiple provinces, but each province belongs to one country.
  - **“id”** is a unique identifier for the country.
  - **“country_name”** is the name of the country.

4.33 Component: OrderModel (Model)

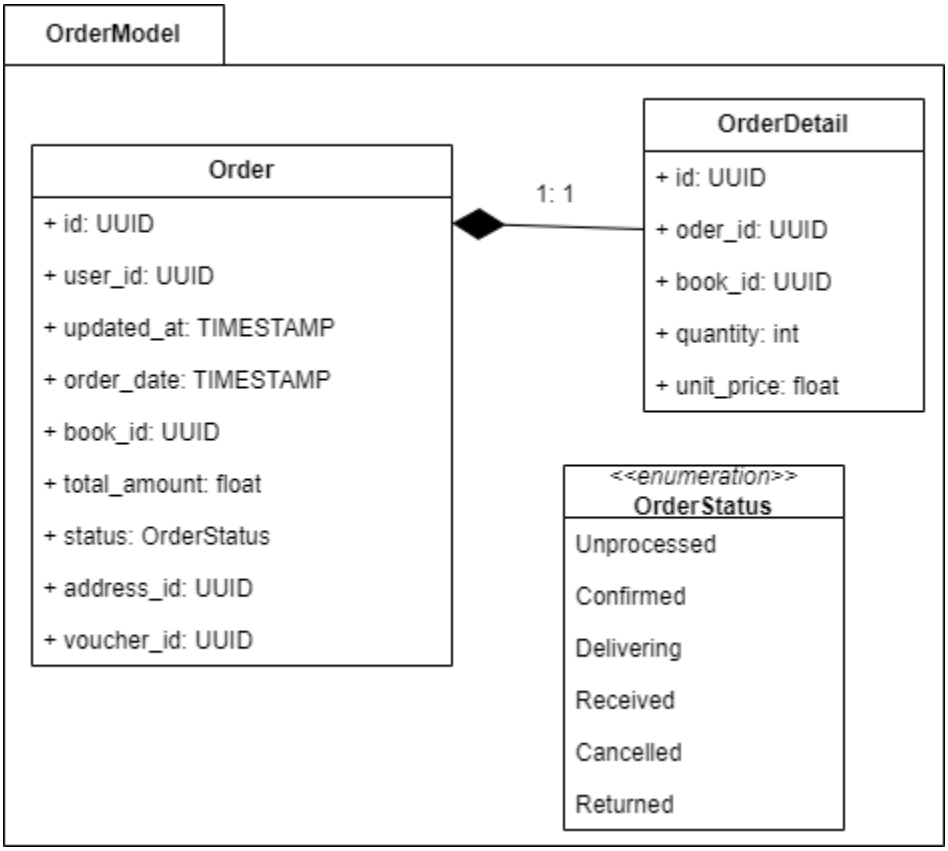

Figure 42. *The OrderModel Component*

- Description
  - This component will contain all the information about each order created by users in the online bookstore.
  - This component will access many components, such as **“CartModel”**, **“VoucherModel”**, **“BookModel”**, and **“PaymentModel”**. It will also import **“AddressModel”** component.

- Explanation
  - The **“Order”** class represents the main entity for storing the order's details.
    - **“id”**: UUID - A unique identifier for the order.
    - **“user_id”**: UUID - ID of the user who created the order.
    - **“updated_at”**: tracks the last update made to the order’s information
    - **“order_date”**: tracks the time the order was created.
    - **“book_id”**: UUID - A foreign key linking to the book the user has ordered.
    - **“total_amount”**: float - The total amount and value of the order.
    - **“status”**: OrderStatus - an enumeration capturing the status of the order.
    - **“address_id”**: UUID - points to the user’s address.
    - **“voucher_id”**: UUID - a foreign key linking to the Voucher used for the order.

  - The **“OrderDetail”** class saves specific details of books in the order.
    - **“id”**: UUID - A unique identifier for this order detail.
    - **“order_id”**: UUID - linking these order details to the specific order.
    - **“book_id”**: UUID - A foreign key linking to the book the user has ordered.
    - **“quantity”**: int - The amount of the book with this book_id above.
    - **“unit_price”**: float - The price of the product per unit.

  - These enumerations standardize the values for specific attributes, ensuring consistency across the system.
    - OrderStatus:
      - **Unprocessed** - An order has been created but not accepted to process yet.
      - **Confirmed** - An order has been accepted, ready to deliver to the user.
      - **Delivering** - An order is in the process of being shipped to the user.
      - **Received** - An order has been successfully delivered to the user.
      - **Cancelled** - An order has been cancelled.
      - **Returned** - An order has been returned to the store.

4.34 Component: BookModel (Model)

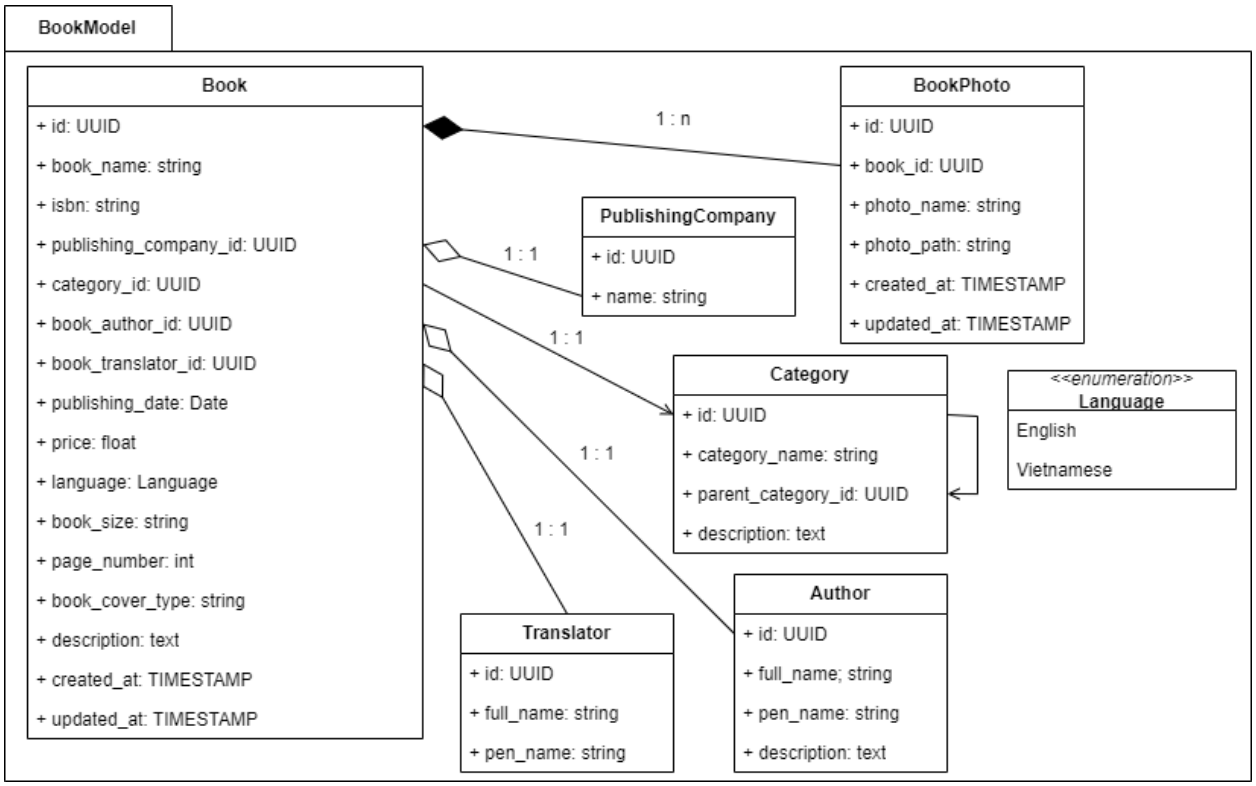

Figure 43. *The BookModel Component*

- Description
  - This component will contain all the information about each book in the bookstore, including basic information like name, photos, ISBN, language, category, author, translator, price, publishing company, etc. This format ensures all books are stored in a detailed and organised structure.
  - This component will import the **"CommentsModel"** component.

- Explanation
  - The **"Book"** class represents the main entity for storing each book.
    - **"id"**: UUID - A unique identifier for the book.
    - **"book_name"**: The name/title of the book.
    - **"ISBN"**: string - The numeric commercial book identifier, unique and constant for each book.
    - **"publishing_company_id"**: UUID - This attribute will link to the publisher who released this book.
    - **"category_id"**: UUID - This attribute will link to the category of this book.
    - **"book_author_id"**: UUID - This attribute will link to the author of this book.
    - **"book_translator_id"**: UUID - This attribute will link to the translator of this book.

- **“publishing_date”:** Date - The release date of this book.
- **“price”**: float - Price of this book per unit.
- **“language”**: Language - An enumeration capturing the Language used in this book.
- **“book_size”**: string - Display the size of the book in all dimensions.
- **“page_number”**: int - Display the number of pages in this book.
- **“book_cover_type”**: Display the type of book cover.
- **“description”**: text - Information and overview of the book’s content.
- **“created_at”**: tracks the time the book is available for sale in the bookshop.
- **“updated_at”**: tracks the last update made to the book’s information.

- The **“BookPhoto”** class saves the illustration image of the specific book.
  - **“id”**: UUID - A unique identifier for the image.
  - **“book_id”**: UUID - linking the picture to the specific book.
  - **“photo_name”**: string - The name of the image file.
  - **“photo_path”**: string - The directory of the image file.
  - **“created_at”**: tracks the time the image is created for the book.
  - **“updated_at”**: tracks the last update made to the book’s image.

- The **“PublishingCompany”** class saves the information of the publisher.
  - “id”: UUID - A unique identifier for the publishing company.
  - “name”: string - The name of the company.

- The **“Author”** class saves the information of the author of the book.
  - **“id”**: UUID - A unique identifier for the author.
  - **“full_name”**: string - Author’s full name.
  - **“pen_name”**: string - Author’s pen name.
  - **“description”**: text - Information and overview about the book’s author.

- The **“Translator”** class saves the information of the translator of the book.
  - **“id”**: UUID - A unique identifier for the translator.
  - **“full_name”**: string - Translator’s full name.
  - **“pen_name”**: string - Translator’s pen name.

- The **“Category”** class saves the information of the category of the book.
  - **“id”**: UUID - A unique identifier for the category.
  - **“category_name”**: string - Name of the category.
  - **“parent_category_id”**: UUID - This attribute will link to the category parent of this category.
  - **“description”**: text - Information and overview about the category.

- These enumerations below standardize the values for specific attributes, ensuring consistency across the system:
  - Language - **“English”** and **“Vietnamese”**

4.35 Component: CartModel (Model)

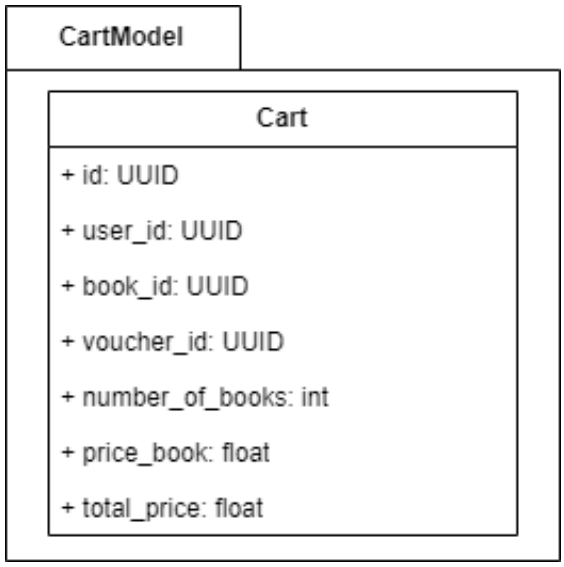

Figure 44. *The CartModel Component*

- Description
  - This component contains information about the user's shopping cart and the products stored in the cart, as well as information about them.
  - This component will import **"BookModel"** and **"UserModel"** components.

- Explanation
  - The **"Cart"** class represents the main entity for the cart.
    - **"id"**: UUID - A unique identifier for the cart.
    - **"user_id"**: UUID - The ID of the user who owns the cart.
    - **"book_id"**: UUID - linking the item in the cart to the specific book.
    - **"voucher_id"**: UUID - linking the item in the cart to the specific voucher.
    - **"number_of_books"**: int - The amount of each book in the cart.
    - **"price_book"**: The price of each book per unit.
    - **"total_price"**: float - The total value of all items in the cart.

4.36 Component: PaymentModel (Model)

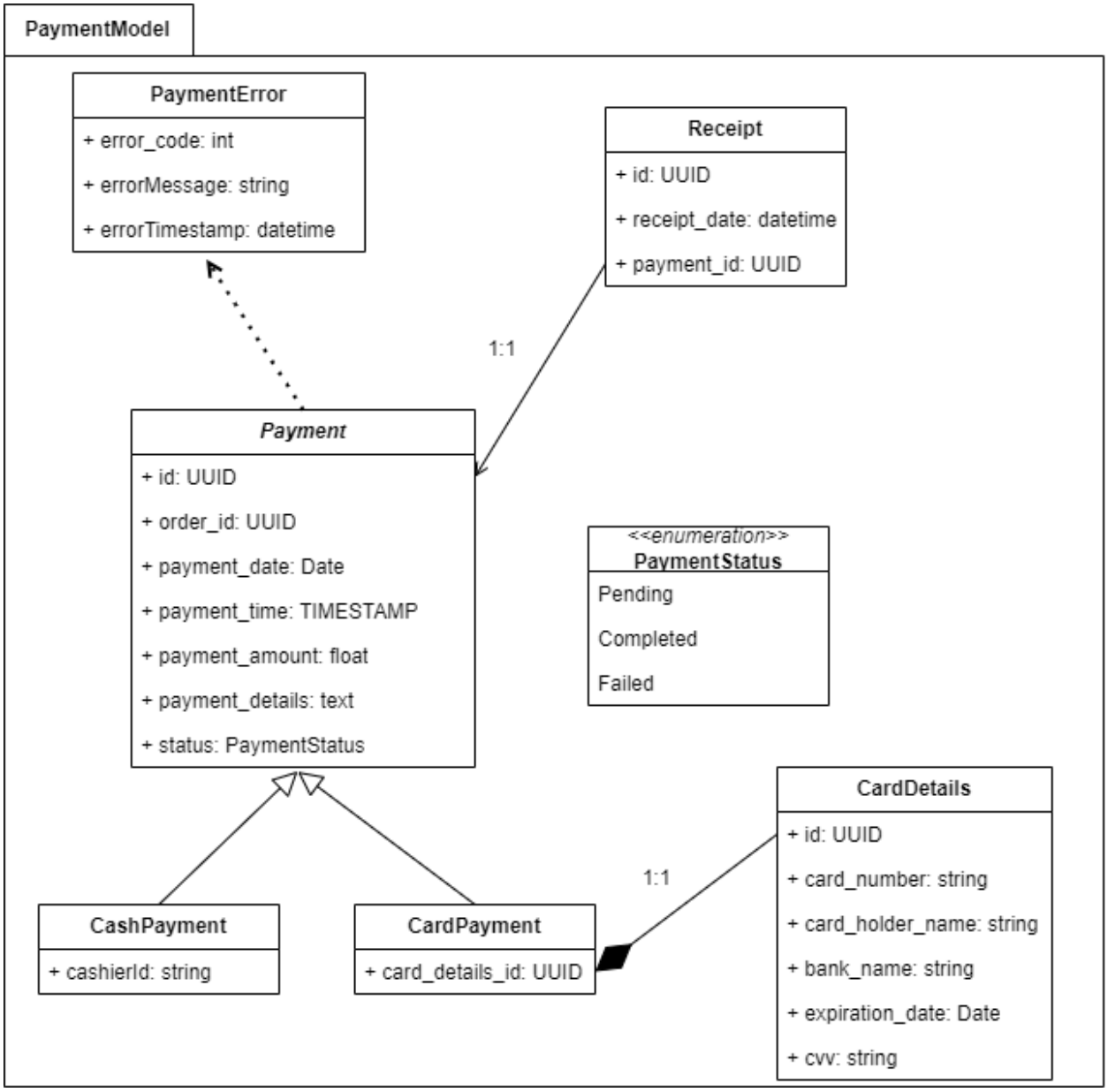

Figure 45. *The PaymentModel Component*

- Description
  - This component encapsulates all attributes of the payment model, including the payment information, status, receipt, methods and details.

- Explanation

  - The **“Payment”** class represents the main entity for storing payment details.
    - **“id”**: UUID - A unique identifier for the payment.

- **"order_id"**: UUID - The ID of the order which makes the payment.
- **"payment_date"**: date - Tracks the date the payment was executed.
- **"payment_time"**: Tracks the time the payment was executed.
- **"payment_amount"**: float - The total amount has to be paid.
- **"payment_details"**: text - Description of the payment action.
- **"status"**: Payment Status - An enumeration capturing the Status used for checking the payment.

- The **"Receipt"** class represents the information about the specific payment.
  - **"id"**: UUID - A unique identifier for the receipt.
  - **"receipt_date"**: date - Tracks the date the receipt was created.
  - **"payment_id"**: UUID - The ID of the payment which owns the receipt.

- The **"CashPayment"** class represents the details about the cash payment method.
  - **"cashierId"**: UUID - The ID of the cashier who executed the payment.

- The **"CardPayment"** class represents the information about the card payment method.
  - **"card_details_id"**: UUID - Unique identifier for the card details.

- The **"CardDetails"** class represents the details about the card payment method.
  - **"id"**: UUID - A unique identifier for the card details.
  - **"card_number"**: string - The card number used for the payment.
  - **"card_holder_name"**: string - The owner's name of the card.
  - **"bank_name"**: string - Name of the bank that manages this card
  - **"expiration_date"**: The expiration date of the card.
  - **"cvv"**: string - Card Verification Value of the card, used for authentication.

- The **"PaymentError"** class stores information about a payment when an error occurs.
  - **"error_code"**: int - The code representing the corresponding error
  - **"errorMessage"**: string - The information about the error code.
  - **"errorTimestamp"**: Track the time the error occurs.

- These enumerations below standardize the values for specific attributes, ensuring consistency across the system:
  - PaymentStatus:
    - **Pending**: The payment is in progress.
    - **Completed**: The payment process is done successfully.
    - **Failed**: The payment process failed.

4.37 Component: CommentsModel (Model)

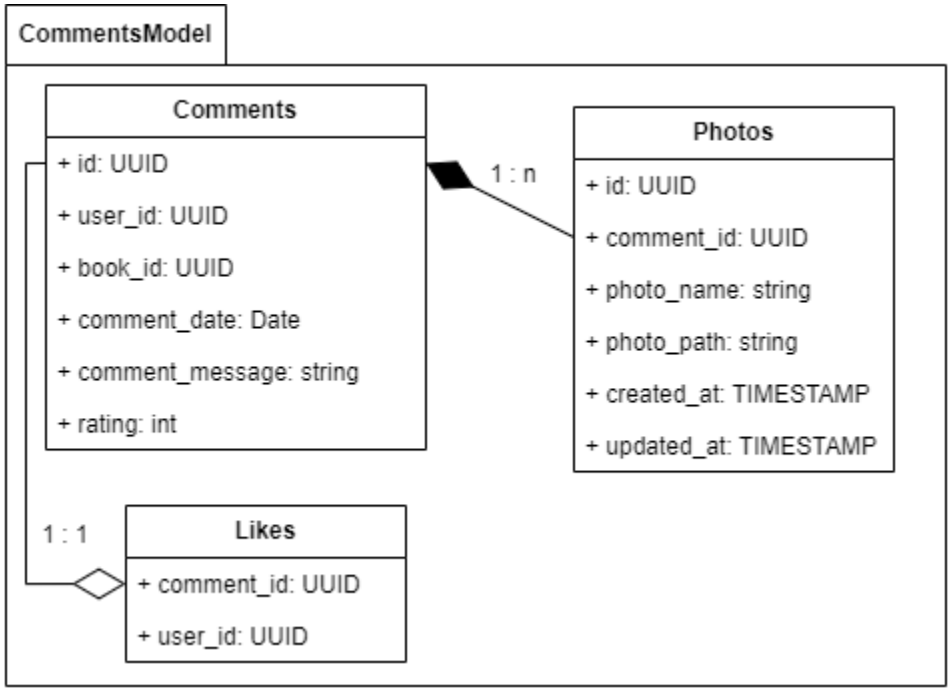

Figure 46. *The CommentsModel Component*

- Description
  - This component contains information about the book's comment section, including the comments, photos and likes.
  - This component will access the **"UserModel"** component.

- Explanation
  - The **"Comment"** class represents the main entity for the comment section.
    - **"id"**: UUID - A unique identifier for the comment.
    - **"user_id"**: UUID - The ID of the user who wrote the comment.
    - **"book_id"**: UUID - The book that contains the comment.
    - **"comment_date"**: date - Tracks the time the comment was written.
    - **"comment_message"**: string - Content of comments by the user.
    - **"rating"**: int - User's rating of the book.

  - The **"Photos"** class represents the photos linked to the specific comment.
    - **"id"**: UUID - A unique identifier for the photos.
    - **"comment_id"**: UUID - The ID of the comment to which the photo is linked to.
    - **"photo_name"**: string - The name of the image file.
    - **"photo_path"**: string - The directory of the image file.
    - **"created_at"**: tracks the time the image is posted for the comment.
    - **"updated_at"**: tracks the last update made to the comment's image.\

- The **“Likes”** class represents the likes a comment gets over time.
  - **“comment_id”**: UUID - The ID of the comment.
  - **“user_id”**: UUID - The ID of the user who wrote the comment.

4.38 Component: VoucherModel (Model)

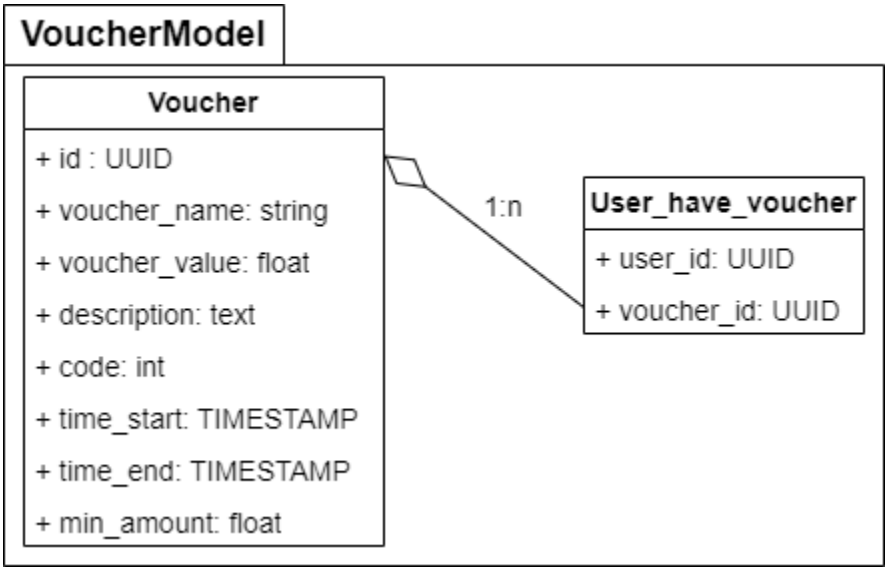

Figure 47. *The VoucherModel Component*

- Description
  - This component will contain all information about the vouchers. It consists of two main classes, **Voucher** and **User_have_voucher**, and it illustrates the relationship between users and vouchers.

- Explanation
  - The **“Voucher”** class captures the details about promotional vouchers that can be used by users. This class has no methods and has some attributes.
    - **“id”**: UUID - A unique identifier for the voucher.
    - **“voucher_name”**: string - The name of the voucher, which can be used to describe the promotion.
    - **“voucher_value”**: float - The monetary value or discount percentage provided by the voucher.
    - **“description”**: text - A detailed description of the voucher, explaining its benefits and conditions.
    - **“code”**: int - A unique code associated with the voucher, which users can apply during checkout.
    - **“time_start”**: TIMESTAMP - The start time from which the voucher becomes valid.
    - **“time_end”**: TIMESTAMP - The end time after which the voucher

expires.
    - **“min_amount”**: float - The minimum purchase amount required to use the voucher.
- The **“User_have_voucher”** class links users with the vouchers they possess, representing a many-to-many relationship between users and vouchers.
    - **“user_id”**: UUID - A unique identifier linking to the user who has the voucher.
    - **“voucher_id”**: UUID - A unique identifier linking to the voucher that the user possesses.

5. Deployment

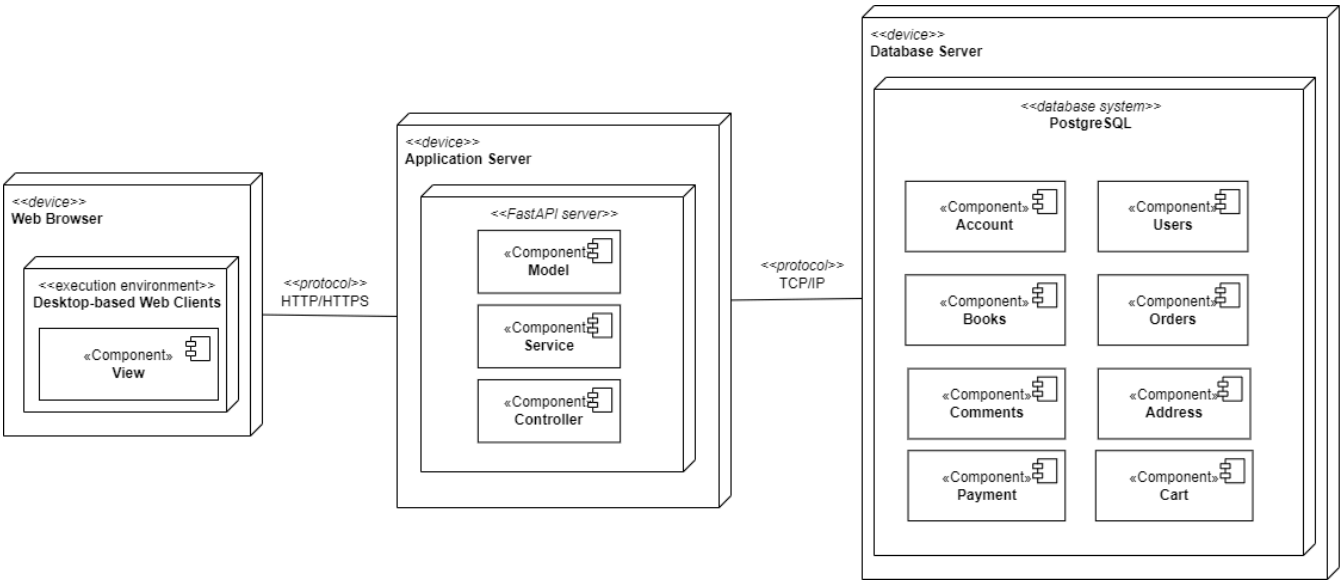

Figure 48. *The Deployment Diagram*

5.1 Node: Web Browser (Device)

This node represents the user's and admin’s client device, which could be a desktop computer, laptop, or any device capable of running a web browser. This web browser is used to interact with the web application. It functions as the front-end client. The "View" component runs in the web browser, representing the UI/UX that users and admins interact with.

5.2 Node: Application Server (Device)

This node represents the server where the application's backend logic resides. It handles processing requests from the client, executing business logic, and interacting with the database server.
The application server is running a FastAPI server,

Components:

- **Model**: This component represents the data models or the core logic of the application, typically interacting with the database and defining how data is structured and accessed.
- **Service**: The service component handles the business logic, processing the data according to the application's requirements.
- **Controller**: This component manages the flow of data between the view and model, handling incoming requests from the user, processing them, and returning the appropriate response.

5.3 Node: Database Server (Device)

This node represents the database server that stores and manages the application’s data.

The database server is running PostgreSQL.

Components:

- **Account**: Manages user accounts, including authentication and user profile information.
- **Users**: Stores detailed user information and handles user-related data.
- **Books**: Stores information about the books available, including titles, authors, prices, and availability.
- **Orders**: Manages the order data, including order history, order status, and transaction details.
- **Comments**: Stores user reviews or comments on books.
- **Address**: Manages user shipping and billing addresses.
- **Payment**: Handles payment processing information, possibly including payment methods and transaction records.
- **Cart**: Stores data related to user shopping carts, tracking items that users intend to purchase.

6. Implementation View

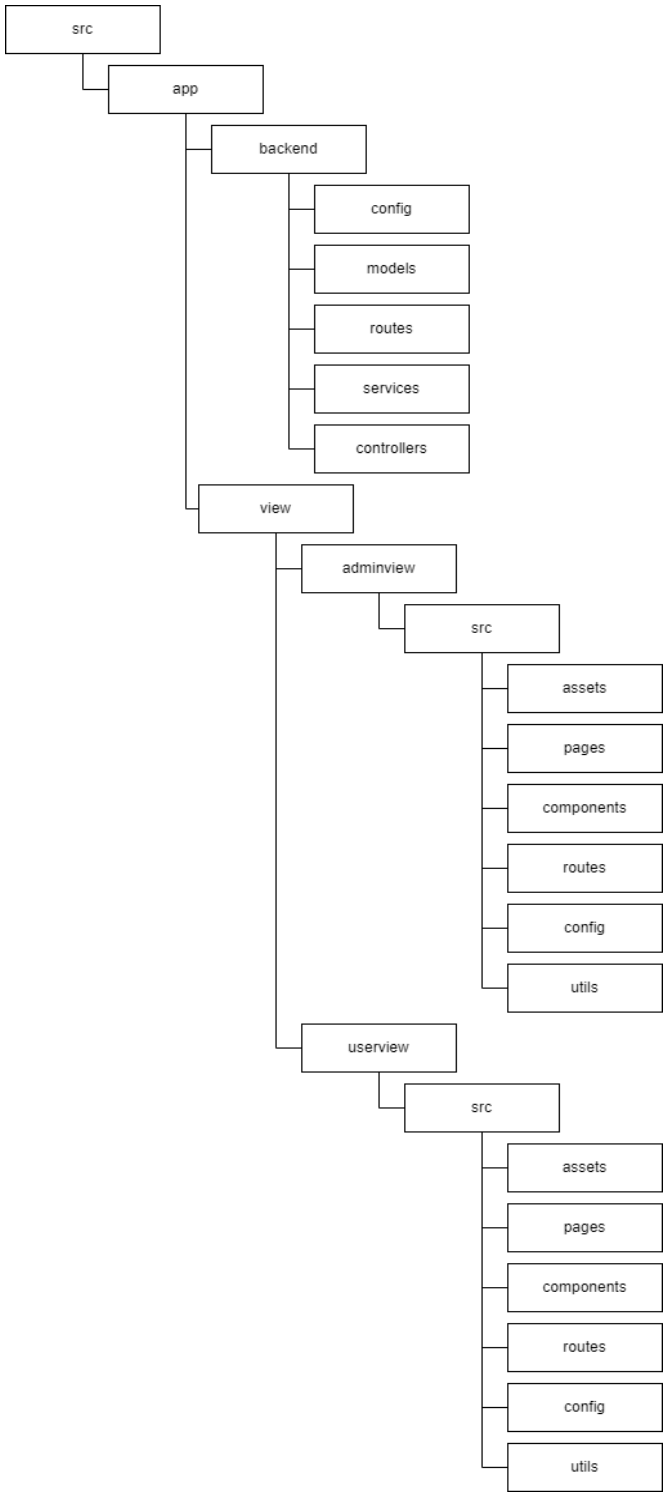

Figure 50. *The folder structure of the system*

The first folder “app” will contain all the folders below (backend and view (frontend)).

6.1 Main folders: backend

This main folder will implement the basic logic and platform of the application server.

6.1.1 Sub-folder: config

This sub-folder will consist of the files (or the sub-folders containing many files) that have the functionality of connecting to the database and configuring the database and the system environment.

6.1.2 Sub-folder: models (implemented the model layers)

This sub-folder will consist of the model file (as described in the Model Layer) that will define the structure of the database and all related functions of the database.

6.1.3 Sub-folder: routes

This sub-folder will contain all files (or sub-folders containing files) as a function of directing requests from the view layer (users or admins) to the corresponding controllers.

6.1.4 Sub-folder: services (implemented the services layers)

This sub-folder will include all functional files (or sub-folders containing files) as described in the part of the Service layer (MVC model) that has the functionality of connecting to the database to retrieve data as requested from the browser.

6.1.5 Sub-folder: controllers (implemented the controllers' layers)

This sub-folder will include all the files (or the sub-folder containing the files) as described in the part of the Controller layer (MVC model) that receive requests from the browser directed by route receive results from the service and then render the view to the browser.

6.2 Main folders: view

This main folder contains all HTML files combined with the public (or the sub-folder containing files) that helps display the user interface as described in the part of the View layer (MVC model).

6.2.1 Sub-folder: userview

This sub-folder contains all classes about all pages for customers (ShopService, Auth, Profile, Home, …) that help display UI and connect to the application server.

6.2.2 Sub-folder: adminview

This sub-folder contains all classes about all pages for administrators (RevenueAnalysis, OrderManagement, AdminAuth, BookManagement, …) that help display UI and connect to the application server.