CSE241: Object-Oriented Computer Programming

Desktop E-commerce

Project Video Link:

https://drive.google.com/file/d/1NBGH-jjXdPAsWfYtU5fKhiYffOLvPHcV/view

Group 14: Name (ID)

Islam Nashaat Ahmed Mohammed (23P0077)

Ahmed Hesham Mohammed (23P0167)

Youssef Farid Haddad (23P0113)

Marwan Wagih Mohamed Mohamed (23P0247)

Ahmed Sadik Mobrak (23P0140)

Semester (Fall 2024)

Desktop E-commerce

This project involves the design and implementation of an e-commerce application, featuring distinct roles for administrators and customers. The primary functionalities include user management, product management, and order management.

Administrators have the ability to manage product and category information, including adding, updating, and deleting products and categories. They also have access to view all customer and order details, ensuring they can oversee the entire operations smoothly. The admin dashboard includes intuitive screens for these tasks, making it easy for administrators to navigate and perform their duties efficiently.

Customers can create accounts, browse products, add products to their carts, place orders, and manage their personal information. The application offers a user-friendly graphical user interface (GUI) using JavaFX to facilitate a seamless shopping experience. The GUI includes screens for user registration and login, product browsing, cart management, and order placement.

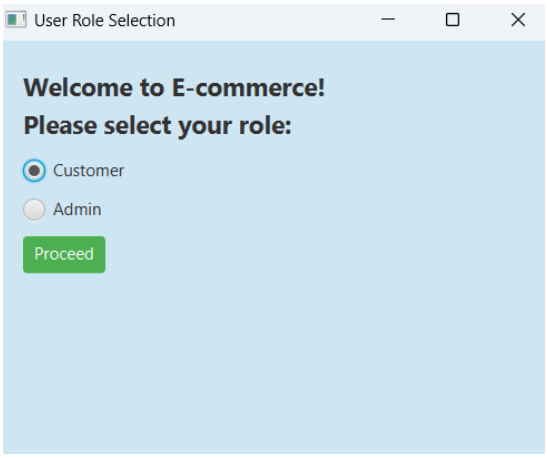
Additionally, the application includes robust validation checks to ensure data integrity and security, covering aspects like user credentials, product details, and transaction information. This helps in maintaining the reliability and accuracy of the data stored in the system.

By integrating multiple data access objects (DAOs) and services, the system ensures efficient data handling and business logic execution. The use of enums and validation classes further enhances the robustness and reliability of the application.

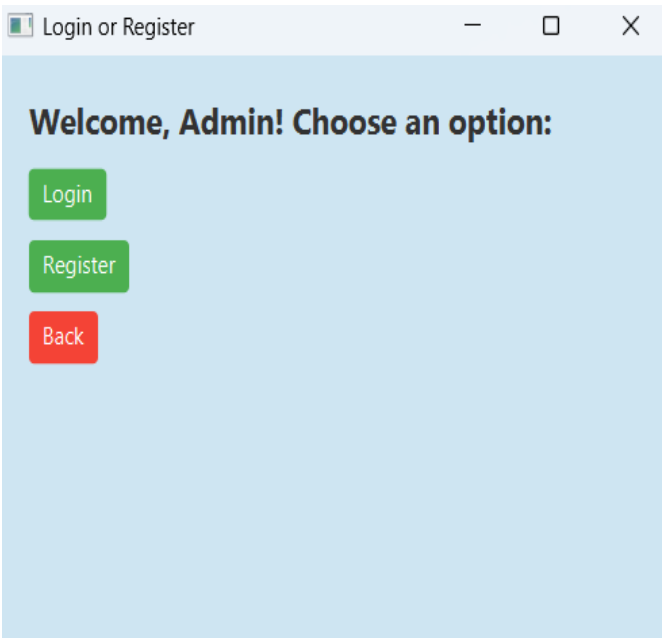
Overall, this project aims to deliver a comprehensive and user-friendly e-commerce platform, catering to both administrative tasks and customer shopping experiences.

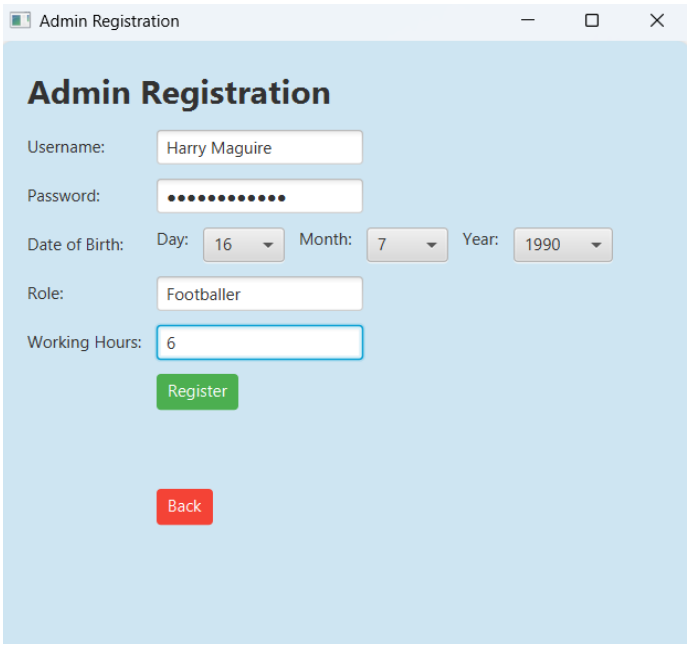
Steps of Gui

Login/Registration

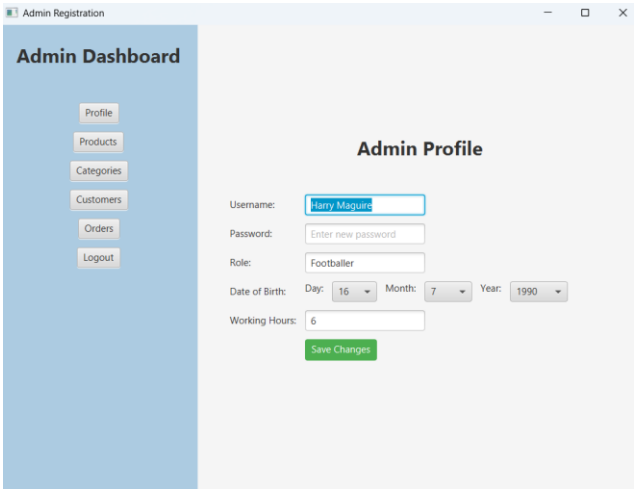


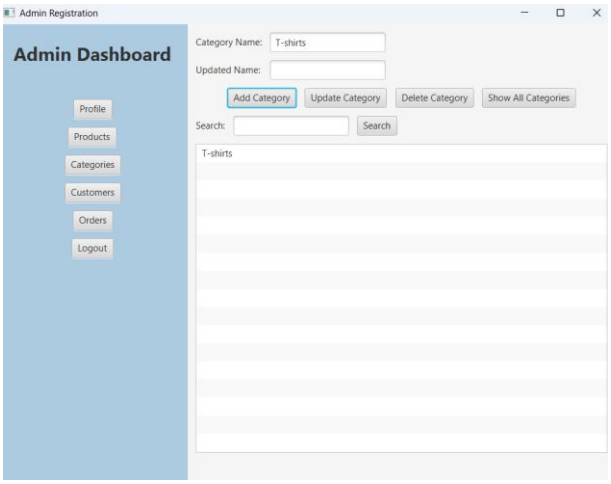
Admin Registration

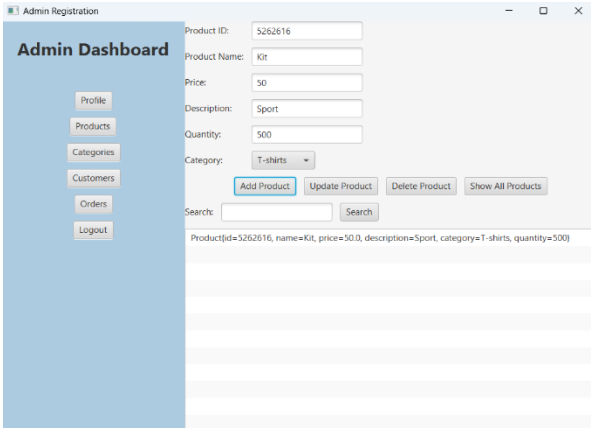


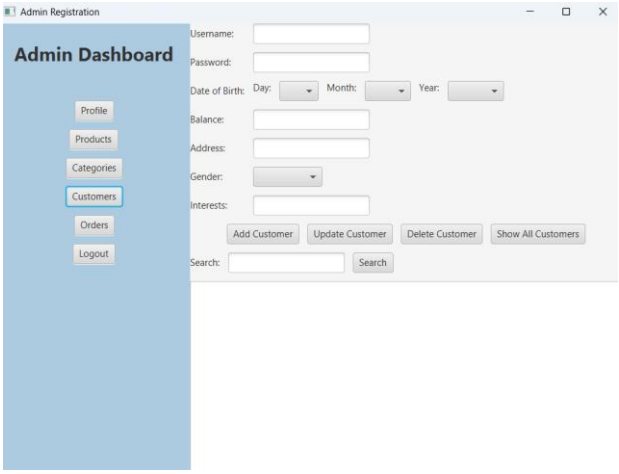


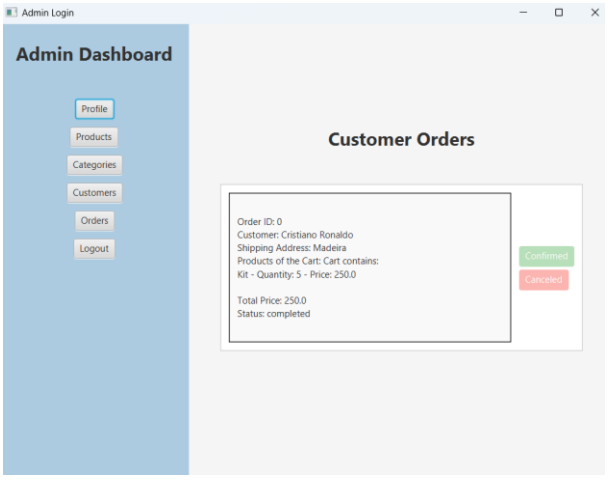
Admin Dashboard



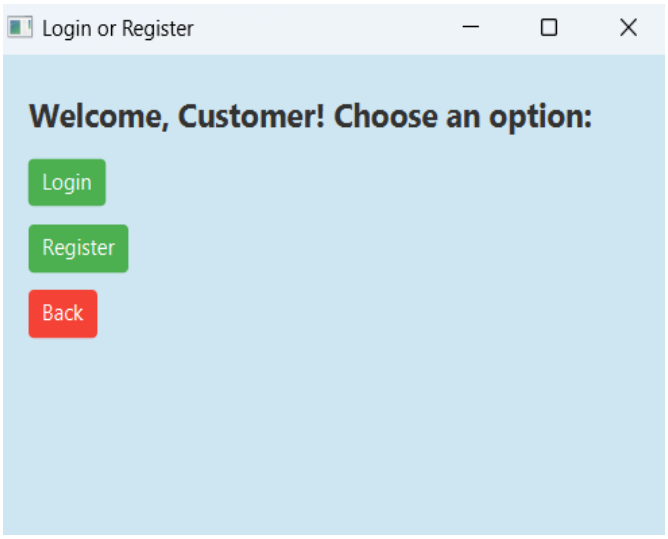


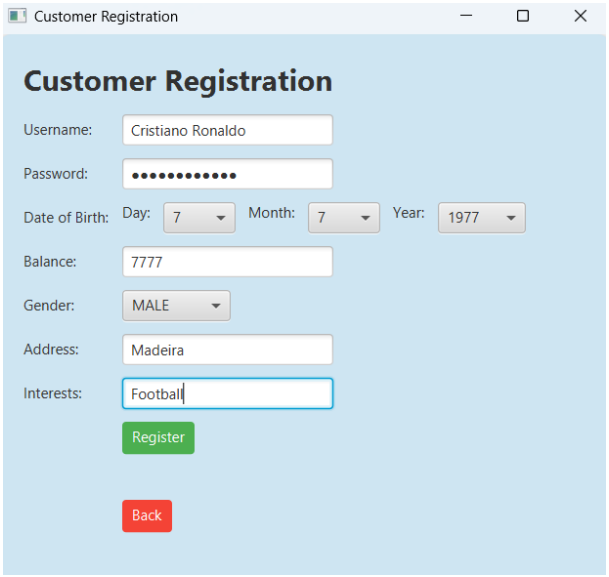




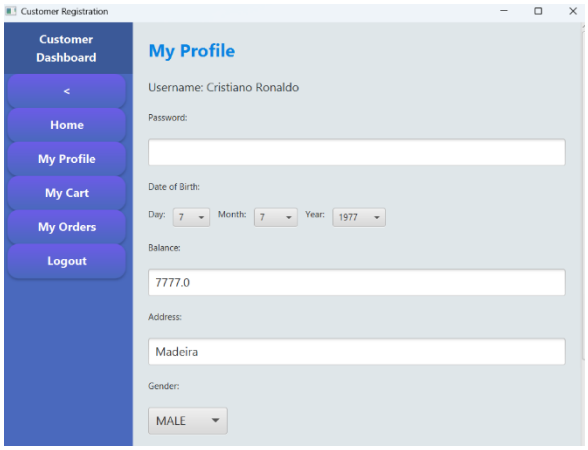


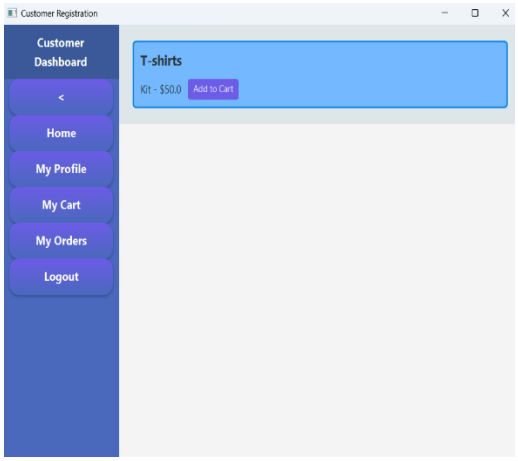
Customer Registration

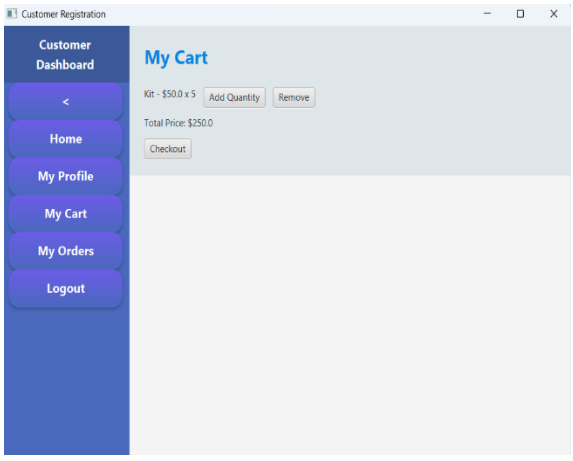


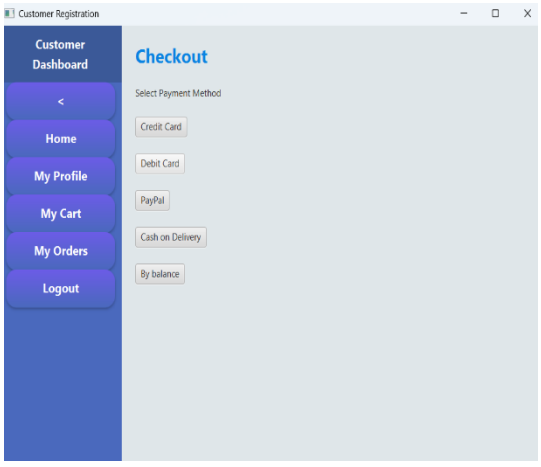


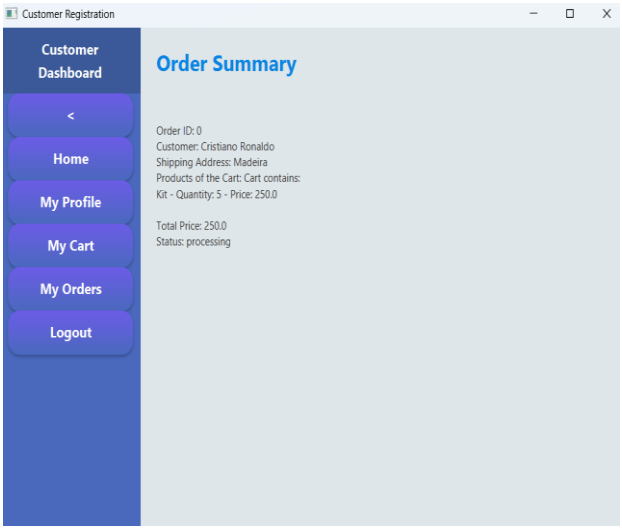
Customer Dashboard

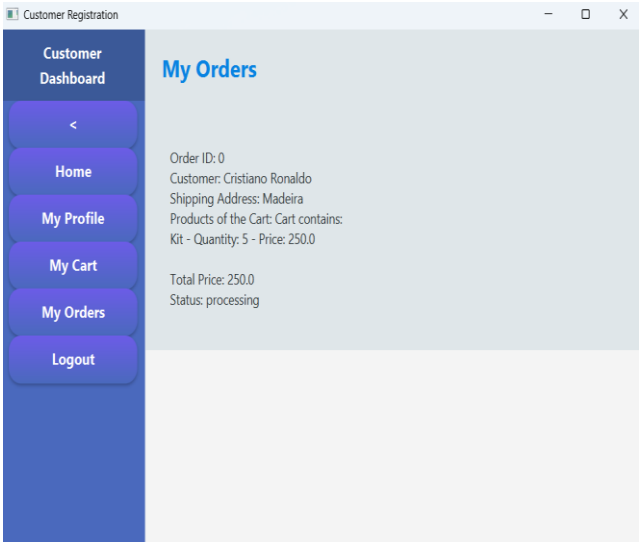












UML

The UML (Unified Modeling Language) diagram of this project provides a comprehensive visual representation of the various classes, their attributes, methods, and the intricate relationships between them. It outlines the core components such as Admin, Customer, Product, Category, and Order, alongside the associated services and data access objects (DAOs). The diagram details the inheritance hierarchy with AbstractUser being the superclass for both Admin and Customer. It also depicts how enums like Gender are used within classes. Associations highlight interactions between classes like AdminService with AdminDAO and CustomerService with CustomerDAO. Aggregations and compositions are illustrated, showing how Cart aggregates Product, and Order composes Customer and Cart. The use of interfaces, such as DAO, and their implementations by classes like AdminDAO, CustomerDAO, ProductDAO, and CategoryDAO are clearly defined. Additionally, dependencies on the ValidationCheck class ensure data integrity across the application. Overall, the UML diagram encapsulates the structural blueprint of the e-commerce application, detailing the interactions and dependencies that drive its functionality.

Relations:

Inheritance:

- Admin inherits from AbstractUser.
- Customer inherits from AbstractUser.

Implementation:

- AdminDAO implements DAO<Admin>.
- CustomerDAO implements DAO<Customer>.
- ProductDAO implements DAO<Product>.
- CategoryDAO implements DAO<Category>.

Associations:

- AdminService uses AdminDAO.
- CustomerService uses CustomerDAO.
- Order uses Customer and Cart.
- Admin interacts with ProductDAO.
- Admin interacts with CategoryDAO.
- Customer interacts with Cart.
- Product uses Category.
- AdminDAO manages Admin instances.
- CustomerDAO manages Customer instances.
- ProductDAO manages Product instances.
- CategoryDAO manages Category instances.
- Customer uses Gender.

Aggregations:

- Cart aggregates Product.

- Category aggregates Product.

- Database aggregates ProductDAO, CustomerDAO, AdminDAO, CategoryDAO.

Compositions:

- Order composes Customer.

- Order composes Cart.

- Database composes Customer.

- Database composes Admin.

- Database composes Category.

- Database composes Product.

Dependencies:

- ValidationCheck methods are used by Admin, Customer, AdminService, CustomerService.

- Order depends on PaymentMethod for payment processing.

Review of Classes and Relations:

- AbstractUser has no specific relationships.

- Admin inherits from AbstractUser, uses ProductDAO, uses CategoryDAO, and is associated with AdminDAO.

- Customer inherits from AbstractUser, uses Cart, uses Gender, and is associated with CustomerDAO.

- AdminService uses AdminDAO.

- CustomerService uses CustomerDAO.

- AdminDAO implements DAO<Admin> and manages Admin.

- CustomerDAO implements DAO<Customer> and manages Customer.

- ProductDAO implements DAO<Product> and manages Product.

- CategoryDAO implements DAO<Category> and manages Category.

- Cart aggregates Product and is composed by Customer.

- Category aggregates Product.

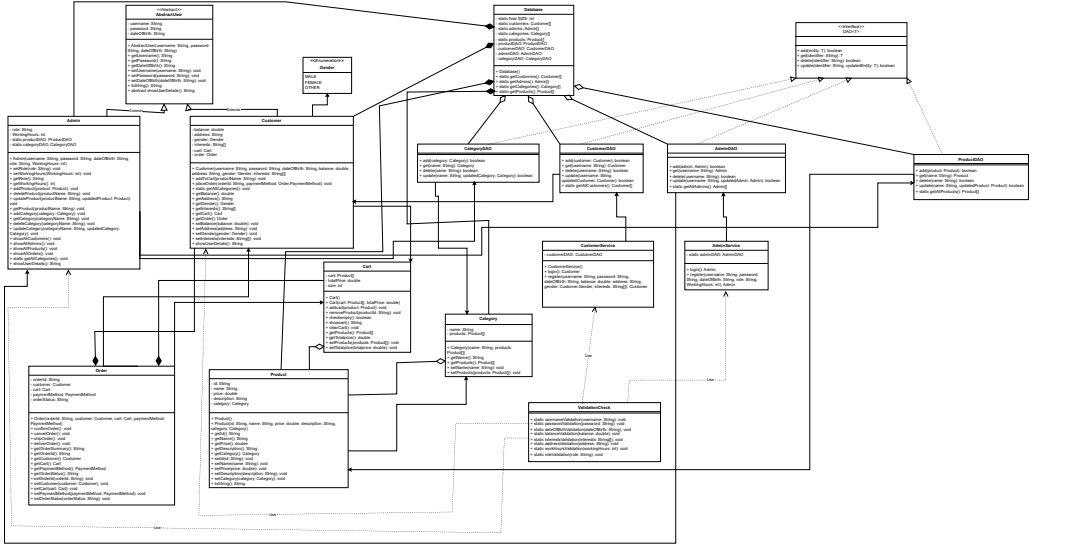
- Product is used by Category.

- Order uses Customer, uses Cart, and is composed by Customer. It also uses PaymentMethod for payment processing.

- Database aggregates ProductDAO, CustomerDAO, AdminDAO, CategoryDAO, and composes Customer, Admin, Category, and Product.

- ValidationCheck methods are used by Admin, Customer, AdminService, and CustomerService.

- Gender is an enum used by Customer.


«Abstract»
AbstractUser
- username: String
- password: String
- dateOfBirth: String
+ AbstractUser(username: String, password: String, dateOfBirth: String)
+ getUsername(): String
+ getPassword(): String
+ getDateOfBirth(): String
+ setUsername(username: String): void
+ setPassword(password: String): void
+ setDateOfBirth(dateOfBirth: String): void
+ toString(): String
+ abstract showUserDetails(): String
«Enumeration»
Gender
MALE
FEMALE
OTHER
Database
- static final SIZE: int
- static customers: Customer[]
- static admins: Admin[]
- static categories: Category[]
- static products: Product[]
- static adminDAO: AdminDAO
- customerDAO: CustomerDAO
- productDAO: ProductDAO
- categoryDAO: CategoryDAO
+ Database()
+ static getCustomers(): Customer[]
+ static getAdmins(): Admin[]
+ static getCategories(): Category[]
+ static getProducts(): Product[]
«Interface»
DAO<T>
+ add(entity: T): boolean
+ delete(entity: T): boolean
+ delete(identifier: String): boolean
+ update(identifier: String, updatedEntity: T): boolean
Admin
- role: String
- WorkingHours: int
- static productDAO: ProductDAO
- static categoryDAO: CategoryDAO
+ Admin(username: String, password: String, dateOfBirth: String, role: String, WorkingHours: int)
+ setRole(role: String): void
+ setWorkingHours(WorkingHours: int): void
+ getRole(): String
+ getWorkingHours(): int
+ addProduct(product: Product): void
+ deleteProduct(productName: String): void
+ updateProduct(productName: String, updatedProduct: Product): void
+ addCategory(category: Category): void
+ getCategory(category: String): void
+ deleteCategory(categoryName: String): void
+ updateCategory(categoryName: String, updatedCategory: Category): void
+ showAllCustomers(): void
+ showAllAdmins(): void
+ showAllOrders(): void
+ showAllOrders(): void
+ static getAllCategories(): void
+ showUserDetails(): String
Customer
- balance: double
- address: String
- gender: Gender
- friends: String[]
- cart: Cart
- order: Order
+ Customer(username: String, password: String, dateOfBirth: String, balance: double, address: String, gender: Gender, friends: String[])
+ addToCart(product: Product): void
+ placeOrder(cardId: String, paymentMethod: OrderPaymentMethod): void
+ static getAllCategories(): void
+ getBalance(): double
+ getAddress(): String
+ getGender(): Gender
+ getFriends(): String[]
+ getCart(): Cart
+ getOrder(): Order
+ setBalance(balance: double): void
+ setAddress(address: String): void
+ setGender(gender: Gender): void
+ setFriends(friends: String[]): void
+ showUserDetails(): String
CategoryDAO
+ add(category: Category): boolean
+ delete(name: String): boolean
+ update(name: String, updatedCategory: Category): boolean
CustomerDAO
+ add(customer: Customer): boolean
+ delete(customer: Customer): boolean
+ delete(username: String): boolean
+ update(username: String, updatedCustomer: Customer): boolean
+ static getAllCustomers(): Customer[]
AdminDAO
+ add(admin: Admin): boolean
+ getUsername(name: String): Admin
+ delete(username: String): boolean
+ update(username: String, updatedAdmin: Admin): boolean
+ static getAdmins(me): Admin[]
ProductDAO
+ add(product: Product): boolean
+ getName(name: String): Product
+ delete(name: String): boolean
+ update(name: String, updatedProduct: Product): boolean
+ static getAllProducts(): Product[]
CustomerService
- customerDAO: CustomerDAO
+ CustomerService()
+ login(): Customer
+ register(username: String, password: String, dateOfBirth: String, balance: double, address: String, gender: Customer.Gender, friends: String[]): Customer
AdminService
- static adminDAO: AdminDAO
+ login(): Admin
+ register(username: String, password: String, dateOfBirth: String, role: String, WorkingHours: int): Admin
Cart
- cart: Product[]
- totalPrice: double
- size: int
+ Cart()
+ CartCart: Product[], totalPrice: double)
+ addToCart(product: Product): void
+ removeProduct(productName: String): void
+ checkempty(): boolean
+ showcart(): String
+ clearCart(): void
+ getCart(): Product[]
+ getTotalItem(): double
+ getProductAt(productIndex: Product[]): void
+ setTotalItem(totalPrice: double): void
Category
- name: String
- products: Product[]
+ Category(name: String, products: Product[])
+ getName(): String
+ getProducts(): Product[]
+ setName(name: String): void
+ setProducts(products: Product[]): void
Order
- orderId: String
- customer: Customer
- cart: Cart
- paymentMethod: PaymentMethod
- orderStatus: String
+ Order(orderId: String, customer: Customer, cart: Cart, paymentMethod: PaymentMethod)
+ confirmOrder(): void
+ cancelOrder(): void
+ shipOrder(): void
+ deliverOrder(): void
+ getOrderSummary(): String
+ getOrderId(): String
+ getCustomer(): Customer
+ getCart(): Cart
+ getPaymentMethod(): PaymentMethod
+ getOrderStatus(): String
+ setOrderId(orderId: String): void
+ setCustomer(customer: Customer): void
+ setCart(cart: Cart): void
+ setPaymentMethod(paymentMethod: PaymentMethod): void
+ setOrderStatus(orderStatus: String): void
Product
- id: String
- name: String
- price: double
- description: String
- category: Category
+ Product()
+ Product(id: String, name: String, price: double, description: String, category: Category)
+ getId(): String
+ getName(): String
+ getPrice(): double
+ getDescription(): String
+ getCategory(): Category
+ setId(id: String): void
+ setName(name: String): void
+ setPrice(price: double): void
+ setDescription(description: String): void
+ setCategory(category: Category): void
+ toString(): String
ValidationCheck
+ static usernameValidation(username: String): void
+ static passwordValidation(password: String): void
+ static dateOfBirthValidation(dateOfBirth: String): void
+ static balanceValidation(balance: double): void
+ static idInvalidValidation(idInvalid: String[]): void
+ static addressValidation(address: String): void
+ static workHoursValidation(workingHours: int): void
+ static roleValidation(role: String): void
Use
Use
Use
Use
Use
Use