8/9/2024

# User Documentation

Edward Spurrell I Charity Smith I Dawson Bursey

# 1. Application Overview

- **Application Name:** JAVA_Final_Sprint "Vintique"
- **Purpose of the Application:**
  *The purpose of this application is a console-based E-Commerce platform for the vintage store "Vintique" using Java and PostgreSQL*.

---

- **Key Features:**
  *List and briefly describe key features*.
  1. Users can register as Buyers and Sellers
  2. The Sellers can list products for sale, view the database of products, edit and update their products and also delete/remove products
  3. The buyers are able to search for products, and view products stored on the platform including product details and descriptions.
  4. The Admins are able to view all users, delete users, and view all products.

## 2. Class Descriptions

### **Class Name:** App

**Purpose:** The App class serves as the main entry point for the application. It is responsible for setting up the database connection, initializing necessary services and DAOs, and launching the console menu for user interaction. It acts as the orchestrator for the application's flow.

**Methods:**

1. **Method Name: main**
    - **Description:** The main method initializes the application, sets up database connections, and creates the necessary objects for user and product management. It starts the console menu for interacting with the user.
    - **Parameters:**
        - String[] args: Command-line arguments (not used in this case).
    - **Returns:**
        - void: Does not return any value.

2. **Method Name: connectToDatabase**
    - **Description:** Establishes a connection to the PostgreSQL database using JDBC. If the connection fails, it prints an error message and exits the program.
    - **Parameters:**
        - None.
    - **Returns:**
        - void: Does not return any value.

3. **Method Name: closeConnection**
    - **Description:** Closes the database connection if it is open, handling any potential SQLException.
    - **Parameters:**
        - None.
    - **Returns:**
        - void: Does not return any value.

**Attributes:**

1. **Attribute Name: connection**
    - **Type:**
        - Connection (from java.sql)
    - **Description:**
        - Holds the database connection object used for executing SQL statements

**Associations:**

- **UserDAO and UserService:**
  - The App class creates an instance of UserDAO, passing the database connection to it, and then creates an instance of UserService using the UserDAO. This association allows UserService to perform business logic operations using the UserDAO.

- **ProductDAO and ProductService:**
  - Similarly, App instantiates ProductDAO and ProductService, enabling product-related operations through these classes.

- **ConsoleMenu:**
  - App creates a ConsoleMenu object, passing in the Scanner, UserService, and ProductService instances. The console menu interacts with the user, allowing them to use the services provided by the application

## Additional Classes

UserDAO, UserService, ProductDAO, ProductService, and ConsoleMenu.

# Class Name: ProductDAO

**Purpose:** The ProductDAO class is responsible for handling all database operations related to products. It provides methods to add, retrieve, update, delete, and search for products in the database. The class interacts with the database using JDBC (Java Database Connectivity) and is designed to encapsulate the logic for managing product data.

**Attributes:**

1. **Attribute Name: connection**
   - **Type:**
     - Connection (from java.sql)
   - **Description:**
     - This attribute holds the JDBC Connection object, which is used to execute SQL statements and manage transactions with the database.

**Methods:**

1. **Method Name: addProduct**
   - **Description:**
     - This method adds a new product to the Products table in the database. It uses a PreparedStatement to safely insert the product data.
   - **Parameters:**
     - Product product: The Product object containing details like item name, item type, item description, and seller ID.
   - **Returns:**
     - void: This method does not return a value but throws SQLException if any database error occurs.
   - **Exceptions:**
     - SQLException: This exception is thrown if there is an issue executing the SQL query.

2. **Method Name: getProduct**
   - **Description:**
     - Retrieves a product from the database using its unique itemId. It queries the Products table and returns a Product object if found.
   - **Parameters:**
     - int itemId: The unique identifier of the product to be retrieved.
   - **Returns:**
     - Product: A Product object populated with the retrieved data, or null if no matching product is found.
   - **Exceptions:**
     - SQLException: This exception is thrown if there is an issue executing the SQL query.

3. **Method Name: updateProduct**
   - **Description:**
     - Updates the details of an existing product in the database. It uses a PreparedStatement to modify the specified fields for the given product ID.
   - **Parameters:**
     - Product product: A Product object with updated details.
   - **Returns:**
     - void: This method does not return a value but throws SQLException if any database error occurs.
   - **Exceptions:**
     - SQLException: This exception is thrown if there is an issue executing the SQL query.

4. **Method Name: deleteProduct**
   - **Description:**
     - Deletes a product from the database using its itemId. It uses a PreparedStatement to ensure safe execution.
   - **Parameters:**
     - int itemId: The ID of the product to be deleted.
   - **Returns:**
     - void: This method does not return a value but throws SQLException if any database error occurs.
   - **Exceptions:**
     - SQLException: This exception is thrown if there is an issue executing the SQL query.

5. **Method Name: getProductsBySeller**
   - **Description:**
     - Retrieves all products associated with a specific seller ID from the Products table. It returns a list of Product objects.
   - **Parameters:**
     - int sellerId: The ID of the seller whose products are to be retrieved.
   - **Returns:**
     - List<Product>: A list of products belonging to the specified seller. An empty list is returned if no products are found.
   - **Exceptions:**
     - Handles SQLException internally and prints stack trace.

6. **Method Name: getAllProducts**
   - **Description:**
     - Retrieves all products from the Products table. It returns a list of Product objects representing every product in the database.
   - **Parameters:**
     - None.

- **Returns:**
  - List<Product>: A list of all products available in the database. An empty list is returned if no products are found.
- **Exceptions:**
  - Handles SQLException internally and prints stack trace.

7. **Method Name: searchProductsByName**
   - **Description:**
     - Searches for products whose names contain the specified string. It performs a case-insensitive search using a LIKE SQL clause and returns matching products.
   - **Parameters:**
     - String itemName: The search string used to match product names.
   - **Returns:**
     - List<Product>: A list of products whose names match the search criteria. An empty list is returned if no matches are found.
   - **Exceptions:**
     - Handles SQLException internally and prints stack trace.

**Associations:**

- **Database Connection:**
  - Utilizes the Connection object passed during construction to interact with the database. This connection is critical for executing SQL statements and is managed by the App class or other configuration.
  - 
- **Product:**
  - The class heavily relies on the Product class to represent and manage product data. All CRUD operations involve creating, returning, or manipulating Product objects.

# Class Name: UserDAO

**Purpose:** The UserDAO class is a Data Access Object (DAO) that encapsulates all the database operations related to users in the system. This class handles CRUD (Create, Read, Update, Delete) operations on user data stored in the database, using JDBC to execute SQL queries.

**Methods:**

1. **Method Name: saveUser**
   - **Description:**
     - Inserts a new user into the users table in the database.
   - **Parameters:**
     - User user: The User object containing details like username, password, email, and role.
   - **Returns:**
     - void: This method does not return a value but throws SQLException if any database error occurs.

2. **Method Name: getUserByUsername**
   - **Description:**
     - Retrieves a user from the database based on their username.
   - **Parameters:**
     - String username: The username of the user to be retrieved.
   - **Returns:**
     - User: A User object if a matching user is found, or null if no user is found.

3. **Method Name: updateUser**
   - **Description:**
     - Updates the details of an existing user in the database.
   - **Parameters:**
     - User user: The User object containing updated user information.
   - **Returns:**
     - void: This method does not return a value but throws SQLException if any database error occurs.

4. **Method Name: deleteUser**
   - **Description:**
     - Deletes a user from the database based on their user ID.
   - **Parameters:**
     - int userId: The ID of the user to be deleted.
   - **Returns:**

- void: This method does not return a value but throws SQLException if any database error occurs.

5. **Method Name: getUserById**
   - **Description:**
     - Retrieves a user from the database based on their user ID.
   - **Parameters:**
     - int userId: The ID of the user to be retrieved.
   - **Returns:**
     - User: A User object if a matching user is found, or null if no user is found.

6. **Method Name: getAllUsers**
   - **Description:**
     - Retrieves all users from the database.
   - **Parameters:**
     - None.
   - **Returns:**
     - List<User>: A list of User objects representing all users in the database.

**Attributes:**

1. **Attribute Name: connection**
   - **Type:**
     - Connection (from java.sql)
   - **Description:**
     - This attribute holds the JDBC Connection object used to execute SQL statements and manage transactions with the database.

**Associations:**

- **Database Connection:**
  - The UserDAO class is directly associated with the Connection class from the java.sql package. This connection is critical for executing SQL statements and is typically managed externally by other classes or application configuration, such as the App class.

- **User Classes:**
  - The class interacts with several user-related classes (User, Buyer, Seller), demonstrating polymorphism in user handling. The getUserByUsername, getUserById, and getAllUsers methods instantiate Buyer or Seller objects based on the user's role. This implies that Buyer and Seller are likely subclasses of User, with possible unique methods or attributes specific to their roles.

# Class Name: DatabaseConnection

**Purpose:** The DatabaseConnection class is designed to establish a connection to a PostgreSQL database using JDBC. It serves as a simple demonstration of how to connect to a database, load the appropriate JDBC driver, and manage the connection lifecycle within a Java application.

**Methods:**

1. **Method Name: main**
   - **Description:**
     - The main method is the entry point of the program. It demonstrates how to load the PostgreSQL JDBC driver, establish a connection to the database, and handle any exceptions that may occur during this process.
   - **Parameters:**
     - String[] args: Command-line arguments passed to the program. However, these arguments are not used in this implementation.
   - **Returns:**
     - void: This method does not return a value.

**Attributes:**

1. **Attribute Name: URL**
   - **Type:**
     - String
   - **Description:**
     - The JDBC URL used to specify the database connection details. This URL includes the type of database (postgresql), the host (localhost), and the port (5432) as well as the specific database name (vintique).

2. **Attribute Name: USER**
   - **Type:**
     - String
   - **Description:**
     - The username required to authenticate the connection to the database. In this case, the username is postgres.

3. **Attribute Name: PASSWORD**
   - **Type:**
     - String
   - **Description:**
     - The password associated with the database user to authenticate the connection. Here, the password is set to Keyin2021.

**Associations:**

- **JDBC Driver:**
  - The DatabaseConnection class uses the PostgreSQL JDBC driver (org.postgresql.Driver) to facilitate communication between the Java application and the PostgreSQL database. The driver must be available in the classpath for this connection to work.

- **Connection Class:**
  - The DatabaseConnection class interacts with the Connection class from the java.sql package to create and manage a connection to the database. This includes opening the connection using the DriverManager.getConnection method and closing it after use.

- **DriverManager Class:**
  - The DriverManager class is used to obtain a connection to the database using the specified URL, username, and password.

# Class Name: AdminMenu

**Purpose:** The AdminMenu class is designed to provide an interactive console-based menu for administrative users. It allows administrators to perform tasks such as viewing all users, deleting users, and viewing all products along with their seller information. The class interacts with the UserService and ProductService to perform these operations.

**Methods:**

1. **Method Name: AdminMenu (Constructor)**
   - **Description:**
     - Initializes an instance of AdminMenu with the provided Scanner, UserService, and ProductService objects, which are used for input and interaction with user and product data.
   - **Parameters:**
     - Scanner scanner: Used to read input from the console.
     - UserService userService: Provides services related to user management.
     - ProductService productService: Provides services related to product management.
   - **Returns:**
     - AdminMenu: A new instance of the AdminMenu class.

2. **Method Name: displayMenu**
   - **Description:**
     - A static method that creates an instance of AdminMenu and starts the menu. This method provides an entry point for displaying the admin menu.
   - **Parameters:**
     - Scanner scanner: Used for console input.
     - UserService userService: Used to access user-related services.
     - ProductService productService: Used to access product-related services.
   - **Returns:**
     - void: This method does not return a value.

3. **Method Name: start**
   - **Description:**
     - Initiates the admin menu loop, presenting options to the user and executing corresponding actions based on user input. This is the core method for interaction.
   - **Parameters:**
     - None
   - **Returns:**
     - void: This method does not return a value.

4. **Method Name: viewAllUsers**
   - **Description:**
     - Fetches and displays a list of all users from the database, printing each user's details.
   - **Parameters:**
     - None
   - **Returns:**
     - void: This method does not return a value.
       - 
5. **Method Name: deleteUser**
   - **Description:**
     - Prompts the user for a user ID and deletes the corresponding user from the database.
   - **Parameters:**
     - None
   - **Returns:**
     - void: This method does not return a value.

6. **Method Name: viewAllProducts**
   - **Description:**
     - Retrieves and displays all products along with the associated seller information. It queries the user database to fetch the seller's name for each product.
   - **Parameters:**
     - None
   - **Returns:**
     - void: This method does not return a value.

**Attributes:**

1. **Attribute Name: scanner**
   - **Type:**
     - Scanner
   - **Description:**
     - Used for reading user input from the console to navigate and interact with the admin menu.

2. **Attribute Name: userService**
   - **Type:**
     - UserService
   - **Description:**
     - Provides operations and services related to user management, such as retrieving and deleting users.

3. **Attribute Name: productService**
    - **Type:**
        - ProductService
    - **Description:**
        - Provides operations and services related to product management, such as retrieving products.

**Associations:**

- **UserService Class:**
    - The AdminMenu class interacts with the UserService to perform operations like fetching all users and deleting a user. This association allows the admin menu to manage user data effectively.
- **ProductService Class:**
    - The AdminMenu class uses the ProductService to retrieve product information. This association enables the admin menu to list products and provide information about their associated sellers.
- **User Class:**
    - The viewAllUsers method interacts with the User class by retrieving and displaying user objects, which are fetched via the UserService.
- **Product Class:**
    - The viewAllProducts method interacts with the Product class by retrieving and displaying product objects, which are fetched via the ProductService.
- **Scanner Class:**
    - The AdminMenu uses the Scanner class for reading user input from the console, enabling interaction with the admin menu options.

# Class Name: BuyerMenu

**Purpose:** The BuyerMenu class provides an interactive console-based menu interface for buyers in the application. It allows buyers to view all available products or search for specific products by name. The class utilizes a Scanner for user input and interacts with the ProductService to access product information.

**Methods:**

1. **Method Name: BuyerMenu (Constructor)**
   - **Description:**
     - Initializes a new instance of the BuyerMenu class with a given Scanner and ProductService. This setup is required to facilitate user interactions and product retrieval.
   - **Parameters:**
     - Scanner scanner: Used for reading user input from the console.
     - ProductService productService: Provides methods to retrieve and search products.
   - **Returns:**
     - BuyerMenu: A new instance of the BuyerMenu class.

2. **Method Name: displayMenu**
   - **Description:**
     - A static method that creates a new instance of the BuyerMenu and initiates the start of the buyer menu loop. It serves as the entry point for displaying the buyer menu.
   - **Parameters:**
     - Scanner scanner: Used for reading user input from the console.
     - ProductService productService: Provides methods for managing products.
   - **Returns:**
     - void: This method does not return a value.

3. **Method Name: start**
   - **Description:**
     - Initiates the buyer menu loop, presenting options for viewing products, searching for products, or exiting the menu. This method handles user choices and delegates actions to specific methods based on input.
   - **Parameters:**
     - None
   - **Returns:**
     - void: This method does not return a value.

4. **Method Name: viewProducts**
   - **Description:**
     - Retrieves and displays all products available in the system using the ProductService. The method iterates over the list of products and prints each one to the console.
   - **Parameters:**
     - None
   - **Returns:**
     - void: This method does not return a value.

5. **Method Name: searchProducts**
   - **Description:**
     - Prompts the user to enter a product name and searches for matching products using the ProductService. Displays the search results or a message indicating no products were found.
   - **Parameters:**
     - None
   - **Returns:**
     - void: This method does not return a value.

6. **Method Name: finalize**
   - **Description:**
     - Overrides the finalize method to ensure the Scanner is closed when the object is garbage-collected. This helps to release system resources associated with the Scanner.
   - **Parameters:**
     - None
   - **Returns:**
     - void: This method does not return a value.

**Attributes:**

1. **Attribute Name: scanner**
   - **Type:**
     - Scanner
   - **Description:**
     - Used for reading input from the console, allowing the buyer to interact with the menu and make selections.
2. **Attribute Name: productService**
   - **Type:**
     - ProductService
   - **Description:**
     - Provides access to methods for retrieving and searching for products in the system.

**Associations:**

- **ProductService Class:**
  - The BuyerMenu class interacts with the ProductService class to retrieve product information. This association allows the buyer menu to access product data, enabling features like viewing all products and searching for specific products by name.
- **Product Class:**
  - The viewProducts and searchProducts methods interact with the Product class by retrieving and displaying product objects, which are managed by the ProductService.
- **Scanner Class:**
  - The BuyerMenu uses the Scanner class to capture user input from the console. This interaction is essential for navigating the menu and performing actions based on user choices.

# Class Name: SellerMenu

**Purpose:** The SellerMenu class provides a user interface for sellers to manage their products within a system. It allows sellers to perform various actions related to their products, such as adding, viewing, updating, and deleting products. This class operates in a console-based environment, interacting with the user via standard input and output.

**Methods:**

1. **Method Name:** displayMenu
   - **Description:** This static method initializes an instance of SellerMenu and starts the menu interaction. It's a convenient entry point for displaying the menu to the user.
   - **Parameters:**
     - Scanner scanner: The scanner object for reading user input.
     - Seller seller: The seller object representing the current user.
     - ProductService productService: The service object responsible for product operations.
   - **Returns:** void

2. **Method Name:** start
   - **Description:** This method begins the menu loop, presenting options to the user and processing their choices. It continually prompts the user until they choose to exit.
   - **Parameters:** None
   - **Returns:** void

3. **Method Name:** addProduct
   - **Description:** This method handles the addition of a new product by collecting product details from the user and passing them to the ProductService to be saved.
   - **Parameters:** None
   - **Returns:** void

4. **Method Name:** viewProducts
   - **Description:** This method retrieves and displays all products associated with the current seller by querying the ProductService.
   - **Parameters:** None
   - **Returns:** void

5. **Method Name:** updateProduct
   - **Description:** This method facilitates updating an existing product by allowing the user to input new details for a specified product ID and then sending these updates to the ProductService.
   - **Parameters:** None
   - **Returns:** void

6. **Method Name:** deleteProduct
    - **Description:** This method allows the user to delete a product by specifying its ID, which is then processed by the ProductService.
    - **Parameters:** None
    - **Returns:** void

**Attributes:**

1. **Attribute Name:** scanner
    - **Type:** Scanner
    - **Description:** An instance of Scanner used for reading user input from the console.

2. **Attribute Name:** productService
    - **Type:** ProductService
    - **Description:** A service class responsible for performing operations related to products, such as adding, updating, and deleting products.

3. **Attribute Name:** seller
    - **Type:** Seller
    - **Description:** Represents the current seller interacting with the menu, providing access to seller-specific information like user ID.

**Associations:**

- **Interaction with Seller Class:**
    - The SellerMenu class uses an instance of the Seller class to obtain the seller's ID. This ID is crucial for performing operations related to the seller's products, such as adding, viewing, updating, and deleting products.

- **Interaction with ProductService Class:**
    - The ProductService class is used to handle CRUD (Create, Read, Update, Delete) operations for products. The SellerMenu class calls methods from ProductService to perform these operations on behalf of the seller.
- **Interaction with Product Class:**
    - The SellerMenu class creates instances of the Product class to represent the product data being manipulated. These instances are then passed to ProductService for processing.

# Class Name: ConsoleMenu

**Purpose:** The ConsoleMenu class serves as the primary interface for user interactions within a console-based application. It provides functionality for user registration and login, and serves as the entry point for users to access various features based on their role (buyer, seller, admin). It manages user input and delegates business logic to the appropriate service classes.

**Methods:**

1. **Method Name:** start
   - **Description:** This method initiates the main loop of the console menu, presenting options to the user and processing their choices. It continues to prompt the user until they choose to exit.
   - **Parameters:** None
   - **Returns:** void

2. **Method Name:** registerUser
   - **Description:** Handles user registration by collecting necessary details from the user (username, email, password, and role) and then creating a User object based on the role. It delegates the task of saving the user to UserService.
   - **Parameters:** None
   - **Returns:** void

3. **Method Name:** createUserByRole
   - **Description:** Creates a User object based on the provided role (buyer, seller, admin). It constructs the appropriate subclass of User (such as Buyer, Seller, or Admin) and returns it.
   - **Parameters:**
     - String username: The username of the user.
     - String password: The password of the user.
     - String email: The email of the user.
     - String role: The role of the user (buyer, seller, admin).
   - **Returns:** User (or null if the role is invalid)

4. **Method Name:** loginUser
   - **Description:** Handles user login by prompting the user for their username and password. It validates these credentials using UserService and, if successful, provides feedback to the user and handles role-specific actions.
   - **Parameters:** None
   - **Returns:** void

**Attributes:**

1. **Attribute Name:** scanner
    - **Type:** Scanner
    - **Description:** Used for reading user input from the console.

2. **Attribute Name:** userService
    - **Type:** UserService
    - **Description:** Provides methods for user registration and login. This service interacts with the underlying data store to manage user accounts.

3. **Attribute Name:** productService
    - **Type:** ProductService
    - **Description:** Provides methods related to product management. Although not directly used in the ConsoleMenu class methods, it is passed to user role handlers for potential product-related actions.

**Associations:**

- **Interaction with UserService Class:**
    - ConsoleMenu uses UserService to handle user registration and login. The UserService class provides methods to register new users and authenticate existing users by validating their credentials.

- **Interaction with ProductService Class:**
    - While ConsoleMenu does not directly manipulate products, it provides a ProductService instance to the user role handlers. This is useful for cases where roles like sellers or admins might need to perform product-related operations.

- **Interaction with User Class and its Subclasses (Buyer, Seller, Admin):**
    - The ConsoleMenu class creates instances of User or its subclasses based on the user's role during registration. It relies on these user subclasses to handle role-specific logic and interactions.

# Class Name: Admin

**Purpose:**

The Admin class extends the User class and represents an administrative user with higher privileges in the system. An admin has the ability to manage other users and view all products in the system. This class provides specific functionality tailored to administrative roles, such as deleting users and viewing all users and products.

**Methods:**

1. **Method Name:** Admin()
    - **Description:** Default constructor that initializes an Admin object with default values and sets the role to "admin."
    - **Parameters:** None
    - **Returns:** Admin object with the role set to "admin."
2. **Method Name:** Admin(int userId, String username, String password, String email)
    - **Description:** Parameterized constructor that initializes an Admin object with specified values for userId, username, password, and email, and sets the role to "admin."
    - **Parameters:**
        - int userId: The ID of the admin user.
        - String username: The username of the admin user.
        - String password: The password of the admin user.
        - String email: The email of the admin user.
    - **Returns:** Admin object with the specified values and role set to "admin."
3. **Method Name:** deleteUser(User user)
    - **Description:** Method to delete a specified user. This might involve interacting with a data access object (DAO) or service layer to remove the user from the database.
    - **Parameters:**
        - User user: The user object to be deleted.
    - **Returns:** void
4. **Method Name:** viewAllUsers()
    - **Description:** Method to retrieve and display all users in the system. This might involve calling a service layer method to fetch user data and then displaying it.
    - **Parameters:** None
    - **Returns:** void
5. **Method Name:** viewAllProducts()
    - **Description:** Method to retrieve and display all products in the system. This might involve calling a service layer method to fetch product data and then displaying it.
    - **Parameters:** None

    - **Returns:** void
6. **Method Name:** toString()
    - **Description:** Override of the toString method from the User class to provide a string representation of the Admin object, including user ID, username, and email.
    - **Parameters:** None
    - **Returns:** String - A string representation of the Admin object.

**Attributes:**

*Inherits attributes from the User class:*

1. **Attribute Name:** userId
    - **Type:** int
    - **Description:** The unique identifier for the admin user.
2. **Attribute Name:** username
    - **Type:** String
    - **Description:** The username of the admin user.
3. **Attribute Name:** password
    - **Type:** String
    - **Description:** The password of the admin user.
4. **Attribute Name:** email
    - **Type:** String
    - **Description:** The email address of the admin user.
5. **Attribute Name:** role
    - **Type:** String
    - **Description:** The role of the user, set to "admin" in this case.

**Associations:**

- **Interaction with User Class:**
    - Admin extends User, inheriting attributes and methods related to user information. It can utilize methods from the User class and may override them to add or modify functionality specific to administrative roles.
- **Interaction with UserDAO or UserService:**
    - The deleteUser and viewAllUsers methods may interact with a UserDAO or UserService to manage user data. The UserDAO would handle database operations, while the UserService would provide higher-level methods to manage users.
- **Interaction with ProductService:**
    - The viewAllProducts method may interact with ProductService to retrieve and display product information. This service would provide access to product data, which the admin can view.

# Class Name: UserService

**Purpose:** The UserService class is a service layer responsible for handling user-related business logic and interactions. It acts as an intermediary between the application and the data access layer (represented by UserDAO), providing functionality for user registration, login, deletion, and retrieval of user information. It also handles password encryption and verification.

**Methods:**

1. **Method Name:** getUserById
    - **Description:** Retrieves a user based on their ID by delegating the call to UserDAO.
    - **Parameters:**
        - int userId: The ID of the user to be retrieved.
    - **Returns:** User - The user object associated with the given ID.
    - **Throws:** SQLException - If there is an issue accessing the database.
2. **Method Name:** deleteUser
    - **Description:** Deletes a user from the system based on their ID by delegating the call to UserDAO.
    - **Parameters:**
        - int userId: The ID of the user to be deleted.
    - **Returns:** void
    - **Throws:** SQLException - If there is an issue accessing the database.
3. **Method Name:** getAllUsers
    - **Description:** Retrieves a list of all users from the system by calling UserDAO.
    - **Parameters:** None
    - **Returns:** List<User> - A list of all users.
    - **Throws:** SQLException - If there is an issue accessing the database.
4. **Method Name:** registerUser
    - **Description:** Registers a new user by encrypting their password and saving the user details to the database using UserDAO.
    - **Parameters:**
        - User user: The user object containing registration details.
    - **Returns:** void
    - **Throws:** SQLException - If there is an issue accessing the database.
    - **Note:** Utilizes BCrypt for password encryption.
5. **Method Name:** loginUser
    - **Description:** Authenticates a user by verifying their username and password. Uses UserDAO to retrieve user details and BCrypt to check the password.
    - **Parameters:**
        - String username: The username of the user attempting to log in.
        - String password: The password provided by the user.

    - **Returns:** User - The authenticated user object if credentials are valid; otherwise, null.
    - **Throws:** SQLException - If there is an issue accessing the database.
6. **Method Name:** logoutUser
    - **Description:** Placeholder method for handling user logout. Currently, it returns null as no specific logout functionality is implemented.
    - **Parameters:** None
    - **Returns:** User - Always returns null.

**Attributes:**

1. **Attribute Name:** userDAO
    - **Type:** UserDAO
    - **Description:** The data access object (DAO) responsible for database operations related to users. It provides methods to interact with the database for tasks such as saving, retrieving, and deleting user records.

**Associations:**

- **Interaction with UserDAO Class:**
    - UserService relies on UserDAO to perform database operations related to users. It delegates tasks such as retrieving user information, saving new users, and deleting users to the UserDAO.
- **Interaction with User Class:**
    - UserService interacts with the User class to manage user data. This includes creating User instances during registration, retrieving user details for login, and performing user-related operations.
- **Interaction with BCrypt Library:**
    - UserService uses the BCrypt library for password encryption and verification. It ensures that user passwords are securely hashed and compared during authentication.

# Class Name: Product

**Purpose:** The Product class represents an item being sold in the system. It encapsulates the details of the product, such as its ID, name, type, description, and the ID of the seller who is offering the product. This class is used to manage and display product information within the application.

**Methods:**

1. **Method Name:** Product(int itemId, String itemName, String itemType, String itemDescription, int sellerId)
    - **Description:** Constructor that initializes a new Product object with the specified values for item ID, name, type, description, and seller ID.
    - **Parameters:**
        - int itemId: The unique identifier for the product.
        - String itemName: The name of the product.
        - String itemType: The type or category of the product.
        - String itemDescription: A description of the product.
        - int sellerId: The ID of the seller offering the product.
    - **Returns:** Product object initialized with the provided values.
2. **Method Name:** getItemId
    - **Description:** Retrieves the ID of the product.
    - **Parameters:** None
    - **Returns:** int - The unique identifier for the product.
3. **Method Name:** setItemId
    - **Description:** Sets the ID of the product.
    - **Parameters:**
        - int itemId: The new ID for the product.
    - **Returns:** void
4. **Method Name:** getItemName
    - **Description:** Retrieves the name of the product.
    - **Parameters:** None
    - **Returns:** String - The name of the product.
5. **Method Name:** setItemName
    - **Description:** Sets the name of the product.
    - **Parameters:**
        - String itemName: The new name for the product.
    - **Returns:** void
6. **Method Name:** getItemType
    - **Description:** Retrieves the type of the product.
    - **Parameters:** None
    - **Returns:** String - The type of the product.
7. **Method Name:** setItemType
    - **Description:** Sets the type of the product.

- **Parameters:**
  - String itemType: The new type for the product.
- **Returns:** void

8. **Method Name:** getItemDescription
   - **Description:** Retrieves the description of the product.
   - **Parameters:** None
   - **Returns:** String - The description of the product.
9. **Method Name:** setItemDescription
   - **Description:** Sets the description of the product.
   - **Parameters:**
     - String itemDescription: The new description for the product.
   - **Returns:** void
10. **Method Name:** getSellerId
    - **Description:** Retrieves the ID of the seller offering the product.
    - **Parameters:** None
    - **Returns:** int - The ID of the seller.
11. **Method Name:** setSellerId
    - **Description:** Sets the ID of the seller offering the product.
    - **Parameters:**
      - int sellerId: The new ID for the seller.
    - **Returns:** void
12. **Method Name:** toString
    - **Description:** Provides a string representation of the Product object, including its ID, name, type, description, and seller ID.
    - **Parameters:** None
    - **Returns:** String - A formatted string describing the product.

**Attributes:**

1. **Attribute Name:** itemId
   - **Type:** int
   - **Description:** The unique identifier for the product.
2. **Attribute Name:** itemName
   - **Type:** String
   - **Description:** The name of the product.
3. **Attribute Name:** itemType
   - **Type:** String
   - **Description:** The type or category of the product.
4. **Attribute Name:** itemDescription
   - **Type:** String
   - **Description:** A description of the product.
5. **Attribute Name:** sellerId
   - **Type:** int
   - **Description:** The ID of the seller offering the product.

**Associations:**

- **Interaction with Seller Class:**
  - The Product class has an association with the Seller class through the sellerId attribute. This ID represents the seller who offers the product. The Seller class would have a corresponding userId that matches this sellerId.
- **Interaction with ProductService Class:**
  - The Product class is used by the ProductService class to perform various operations such as adding, updating, or retrieving product information. The ProductService would handle business logic related to products, while the Product class provides the data structure.
- **Interaction with ProductDAO Class:**
  - The Product class is used by ProductDAO for database operations involving products. The ProductDAO would use the Product class to save, retrieve, update, or delete product records in the database.

# Class Name: Seller

**Purpose:** The Seller class represents a user with seller-specific attributes and behaviors in the application. It extends the User class, inheriting common user properties while adding functionality specific to sellers, such as managing their store and products. The class allows sellers to add, remove, and update products they are offering.

**Methods:**

1. **Method Name:** Seller(int userId, String username, String password, String email)
   - **Description:** Constructor that initializes a Seller object with the specified user ID, username, password, and email. The store name is set to a default value ("Vintique"), and the product list is initialized as an empty list.
   - **Parameters:**
     - int userId: The unique identifier for the seller.
     - String username: The username of the seller.
     - String password: The password for the seller's account.
     - String email: The email address of the seller.
   - **Returns:** Seller object initialized with the provided values and default attributes.

2. **Method Name:** getStoreName
   - **Description:** Retrieves the name of the seller's store.
   - **Parameters:** None
   - **Returns:** String - The store name of the seller. It is fixed to "Vintique" and cannot be changed.

3. **Method Name:** getProducts
   - **Description:** Retrieves the list of products associated with the seller.
   - **Parameters:** None
   - **Returns:** List<Product> - The list of products managed by the seller.

4. **Method Name:** addProduct
   - **Description:** Adds a product to the seller's product list.
   - **Parameters:**
     - Product product: The product to be added to the list.
   - **Returns:** void

5. **Method Name:** removeProduct
   - **Description:** Removes a product from the seller's product list based on the product ID.
   - **Parameters:**
     - int productId: The ID of the product to be removed.
   - **Returns:** void

6. **Method Name:** updateProduct
   - **Description:** Updates the details of an existing product in the seller's product list.
   - **Parameters:**
     - Product product: The product object with updated details.
   - **Returns:** void

**Attributes:**

1. **Attribute Name:** storeName
   - **Type:** String
   - **Description:** The name of the seller's store. This is set to a default value "Vintique" and is not intended to be changed.

2. **Attribute Name:** products
   - **Type:** List<Product>
   - **Description:** A list of Product objects representing the products that the seller is managing. This list allows the seller to add, remove, and update products.

**Associations:**

- **Interaction with User Class:**
  - The Seller class extends the User class, inheriting common user attributes (like user ID, username, password, and email) and methods. It specializes the User class with additional attributes and methods specific to sellers.

- **Interaction with Product Class:**
  - The Seller class has a one-to-many relationship with the Product class. Each seller manages a list of products, which are instances of the Product class. Methods in the Seller class (addProduct, removeProduct, updateProduct) operate on Product objects to manage the seller's inventory.

# Class Name: User

**Purpose:** The User class represents a user in the system with attributes common to all users, such as an ID, username, password, email, and role. It serves as a base class for different types of users (e.g., buyers, sellers, admins) and provides methods for managing user details and handling user-specific functionality based on their role.

**Methods:**

1. **Method Name:** User()
   - **Description:** Default constructor that initializes a User object without setting any attributes.
   - **Parameters:** None
   - **Returns:** User object with default values.

2. **Method Name:** User(int userId, String username, String password, String email, String role)
   - **Description:** Parameterized constructor that initializes a User object with specified values for user ID, username, password, email, and role.
   - **Parameters:**
     - int userId: The unique identifier for the user.
     - String username: The username of the user.
     - String password: The password for the user's account.
     - String email: The email address of the user.
     - String role: The role of the user (e.g., buyer, seller, admin).
   - **Returns:** User object initialized with the provided values.

3. **Method Name:** getUserId
   - **Description:** Retrieves the user ID.
   - **Parameters:** None
   - **Returns:** int - The unique identifier for the user.

4. **Method Name:** setUserId
   - **Description:** Sets the user ID.
   - **Parameters:**
     - int userId: The new ID for the user.
   - **Returns:** void

5. **Method Name:** getUsername
   - **Description:** Retrieves the username.
   - **Parameters:** None
   - **Returns:** String - The username of the user.

6. **Method Name:** setUsername
   - **Description:** Sets the username.
   - **Parameters:**
     - String username: The new username for the user.
   - **Returns:** void

7. **Method Name:** getPassword
   - **Description:** Retrieves the password.
   - **Parameters:** None
   - **Returns:** String - The password for the user's account.

8. **Method Name:** setPassword
   - **Description:** Sets the password.
   - **Parameters:**
     - String password: The new password for the user.
   - **Returns:** void

9. **Method Name:** getEmail
   - **Description:** Retrieves the email address.
   - **Parameters:** None
   - **Returns:** String - The email address of the user.

10. **Method Name:** setEmail
    - **Description:** Sets the email address.
    - **Parameters:**
      - String email: The new email address for the user.
    - **Returns:** void

11. **Method Name:** getRole
    - **Description:** Retrieves the role of the user.
    - **Parameters:** None
    - **Returns:** String - The role of the user (e.g., buyer, seller, admin).

12. **Method Name:** setRole
    - **Description:** Sets the role of the user.
    - **Parameters:**
      - String role: The new role for the user.
    - **Returns:** void

13. **Method Name:** handleRole
    - **Description:** Handles role-specific functionality by displaying the appropriate menu based on the user's role.
    - **Parameters:**
        - ProductService productService: Service to manage product-related operations.
        - UserService userService: Service to manage user-related operations.
    - **Returns:** void
    - **Details:** The method uses a switch statement to determine the appropriate menu to display based on the user's role (buyer, seller, admin). It interacts with the respective menu classes (e.g., BuyerMenu, SellerMenu, AdminMenu).

14. **Method Name:** toString
    - **Description:** Provides a string representation of the User object, including user ID, username, email, and role.
    - **Parameters:** None
    - **Returns:** String - A formatted string describing the user.

**Attributes:**

1. **Attribute Name:** userId
    - **Type:** int
    - **Description:** The unique identifier for the user.

2. **Attribute Name:** username
    - **Type:** String
    - **Description:** The username of the user.

3. **Attribute Name:** password
    - **Type:** String
    - **Description:** The password for the user's account.

4. **Attribute Name:** email
    - **Type:** String
    - **Description:** The email address of the user.

5. **Attribute Name:** role
    - **Type:** String
    - **Description:** The role of the user, which can be "buyer," "seller," "admin," etc.

**Associations:**

- **Interaction with BuyerMenu, SellerMenu, and AdminMenu Classes:**
  - The handleRole method in the User class interacts with these menu classes to display the appropriate user interface based on the user's role. It instantiates the relevant menu class and passes necessary services to manage user-specific functionalities.
- **Interaction with ProductService and UserService Classes:**
  - The handleRole method also uses ProductService and UserService to perform operations related to products and users, respectively. These services are passed as parameters to the handleRole method.

# Class Name: ProductService

**Purpose:** The ProductService class provides a layer of abstraction between the application and the data access layer for product-related operations. It handles business logic and interactions related to products, such as adding, updating, deleting, and retrieving product information. The class interacts with the ProductDAO to perform database operations and ensures that product-related data is managed effectively.

**Methods:**

1. **Method Name:** ProductService(ProductDAO productDAO)
   - **Description:** Constructor that initializes the ProductService with a ProductDAO instance.
   - **Parameters:**
     - ProductDAO productDAO: The data access object used to interact with the product database.
   - **Returns:** ProductService object with the provided ProductDAO.

2. **Method Name:** addProduct
   - **Description:** Adds a new product to the database.
   - **Parameters:**
     - Product product: The product object to be added.
   - **Returns:** void
   - **Throws:** SQLException if an error occurs while interacting with the database.

3. **Method Name:** getProduct
   - **Description:** Retrieves a product from the database by its ID.
   - **Parameters:**
     - int itemId: The ID of the product to be retrieved.
   - **Returns:** Product - The product object corresponding to the provided ID.
   - **Throws:** SQLException if an error occurs while interacting with the database.

4. **Method Name:** updateProduct
   - **Description:** Updates an existing product in the database.
   - **Parameters:**
     - Product product: The product object with updated information.
   - **Returns:** void
   - **Throws:** SQLException if an error occurs while interacting with the database.

5. **Method Name:** deleteProduct
   - **Description:** Deletes a product from the database by its ID.
   - **Parameters:**
     - int itemId: The ID of the product to be deleted.
   - **Returns:** void

- **Throws:** SQLException if an error occurs while interacting with the database.

6. **Method Name:** getProductsBySeller
   - **Description:** Retrieves all products associated with a specific seller.
   - **Parameters:**
     - int sellerId: The ID of the seller whose products are to be retrieved.
   - **Returns:** List<Product> - A list of products associated with the specified seller.

7. **Method Name:** getAllProducts
   - **Description:** Retrieves all products from the database.
   - **Parameters:** None
   - **Returns:** List<Product> - A list of all products.

8. **Method Name:** getItemById
   - **Description:** Fetches a single product by its ID (duplicate of getProduct).
   - **Parameters:**
     - int itemId: The ID of the product to be fetched.
   - **Returns:** Product - The product object corresponding to the provided ID.
   - **Throws:** SQLException if an error occurs while interacting with the database.

9. **Method Name:** searchProducts
   - **Description:** Searches for products by their name.
   - **Parameters:**
     - String itemName: The name of the product to search for.
   - **Returns:** List<Product> - A list of products that match the provided name.
   - **Throws:** IllegalArgumentException if the itemName is null or empty.
   - **Throws:** SQLException if an error occurs while interacting with the database.

**Attributes:**

1. **Attribute Name:** productDAO
   - **Type:** ProductDAO
   - **Description:** The data access object used for performing database operations related to products. It provides methods to add, retrieve, update, delete, and search for products in the database.

**Associations:**

- **Interaction with ProductDAO:**
  - The ProductService class uses ProductDAO to perform all database interactions related to products. It delegates tasks such as adding, updating, deleting, and retrieving product information to the ProductDAO.

# Class Name: UserService

**Purpose:** The UserService class is responsible for managing user-related operations and business logic. It acts as an intermediary between the application's business layer and the data access layer for user management. This class provides functionalities for user registration, login, retrieval, deletion, and other user-related operations, ensuring that all user data is handled securely and efficiently.

**Methods:**

1. **Method Name:** UserService(UserDAO userDAO)
   - **Description:** Constructor that initializes the UserService with a UserDAO instance.
   - **Parameters:**
     - UserDAO userDAO: The data access object used to interact with the user database.
   - **Returns:** UserService object with the provided UserDAO.

2. **Method Name:** getUserById
   - **Description:** Retrieves a user from the database by their ID.
   - **Parameters:**
     - int userId: The ID of the user to be retrieved.
   - **Returns:** User - The user object corresponding to the provided ID.
   - **Throws:** SQLException if an error occurs while interacting with the database.

3. **Method Name:** deleteUser
   - **Description:** Deletes a user from the database by their ID.
   - **Parameters:**
     - int userId: The ID of the user to be deleted.
   - **Returns:** void
   - **Throws:** SQLException if an error occurs while interacting with the database.

4. **Method Name:** getAllUsers
   - **Description:** Retrieves all users from the database.
   - **Parameters:** None
   - **Returns:** List<User> - A list of all users.
   - **Throws:** SQLException if an error occurs while interacting with the database.

5. **Method Name:** registerUser
   - **Description:** Registers a new user by saving their details to the database after encrypting their password.
   - **Parameters:**
     - User user: The user object containing registration details.
   - **Returns:** void
   - **Throws:** SQLException if an error occurs while interacting with the database.

6. **Method Name:** loginUser
   - **Description:** Authenticates a user by checking their username and password.
   - **Parameters:**
     - String username: The username of the user attempting to log in.
     - String password: The password of the user attempting to log in.
   - **Returns:** User - The authenticated user object if credentials are valid; otherwise, null.
   - **Throws:** SQLException if an error occurs while interacting with the database.

7. **Method Name:** logoutUser
   - **Description:** Logs out a user (this implementation is a placeholder and does not perform any actual logout operations).
   - **Parameters:** None
   - **Returns:** User - null in this implementation, as the method does not perform any logout actions.

**Attributes:**

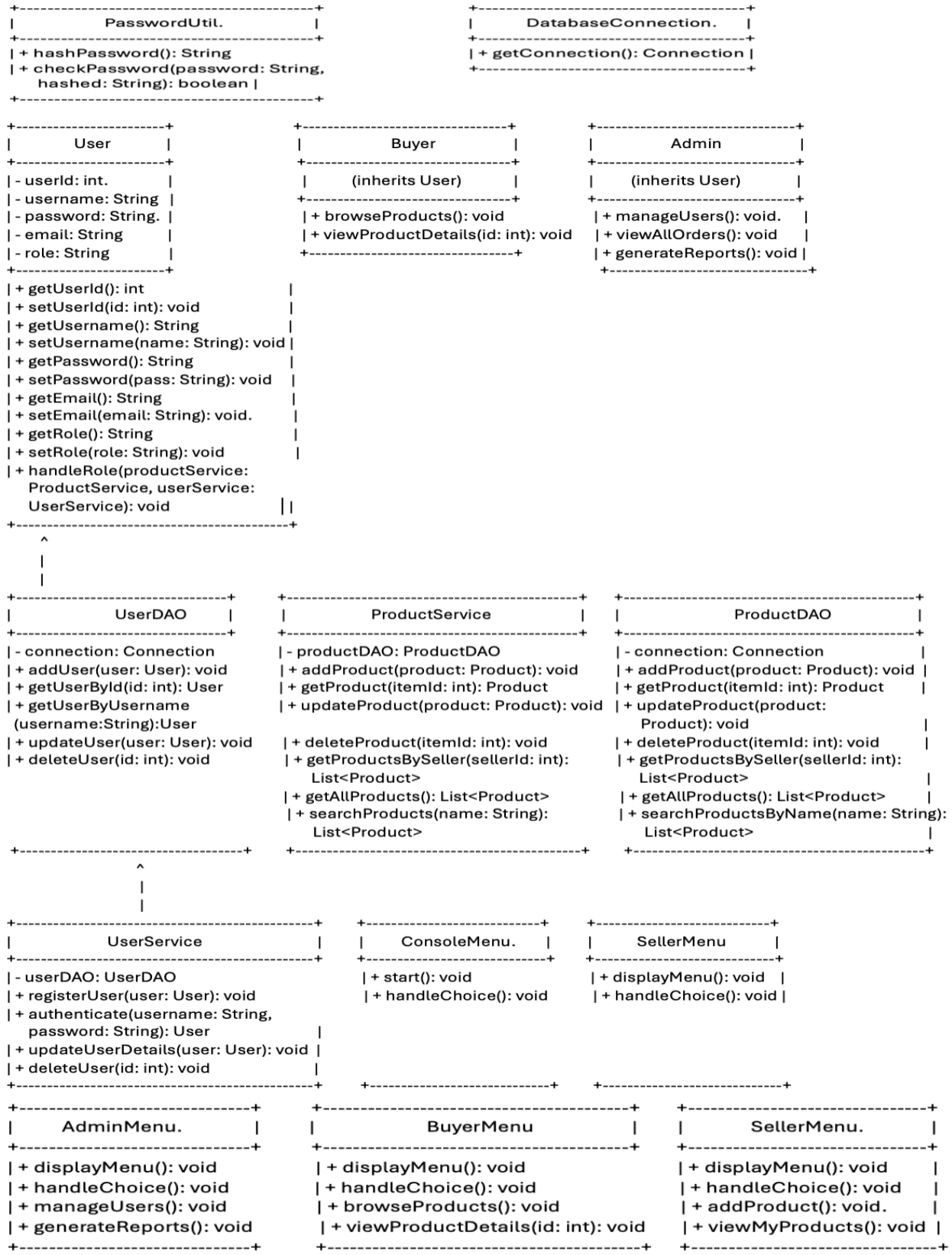
1. **Attribute Name:** userDAO
   - **Type:** UserDAO
   - **Description:** The data access object used for performing database operations related to users. It provides methods to retrieve, save, delete, and update user data in the database.

**Associations:**

- **Interaction with UserDAO:**
  - The UserService class relies on UserDAO to handle all database interactions related to users. It delegates tasks such as retrieving, saving, and deleting user data to the UserDAO. This separation of concerns allows the UserService to focus on business logic and security (such as password encryption) while the UserDAO handles the direct data access.
  -
- **Interaction with BCrypt:**
  - The UserService class uses BCrypt for password encryption and validation. During user registration, it hashes the user's password before saving it to the database. During login, it checks the provided password against the hashed password stored in the database to authenticate the user.

---

## 3. Class Diagram



## 4. Installation

Install PostgreSQL and pgAdmin

1. Download and install PostgreSQL from the official PostgreSQL website
   - https://www.postgresql.org/download/
2. During installation set a password for the 'postgres' user (superuser)
3. Install pgAdmin
   - https://www.pgadmin.org/download/

Set Up PostgreSQL Database

1. Open pgAdmin and connect to your PostgreSQL server using the 'postgres' user.
2. Create a new database for the application:
   - Name: 'vintique'
3. Create the necessary tables using the provided SQL scripts or manually in pgAdmin
4. Insert initial data using provided SQL statements

Install Visual Studio Code (VSCode)

1. Download and install VSCode from the official website
   - https://code.visualstudio.com/
2. Install relevant extensions:
   - Java Extension Pack
     - # Update package index
     - sudo apt update
     -
     - # Install OpenJDK (replace 'openjdk-11-jdk' with the desired version)
     - sudo apt install openjdk-11-jdk
     -
     - # Verify the installation
     - java -version
   - Maven for Java
     - # Install Maven
     - sudo apt install maven
     -

        - # Verify the installation
        - mvn -version
    - PostgreSQL
        - # Install PostgreSQL
        - sudo apt install postgresql postgresql-contrib
        -
        - # Start PostgreSQL service
        - sudo service postgresql start
        -
        - # Access the PostgreSQL command line
        - sudo -u postgres psql

Clone the Project Repository

1. Open a terminal or a command prompt.
2. Clone the project repository using Git:
    - git clone https://github.com/espurrell/JAVA_Final_Sprint.git
3. Navigate to the project directory:
    - cd vintique-ecommerce

Configure the Application

1. Open the project in VSCode
2. Configure the database connection to the application:
    - Open the App.java class
    - Set the database URL, username and password:
        - spring.datasource.url=jdbc:postgresql://localhost:5432/vintique
        - spring.datasource.username=postgres
        - spring.datasource.password=Keyin2021

Build and Run the Application

1. Open a terminal in VSCode
2. Use Maven to build the project:
    - mvn clean install
3. Run the application:
    - mvn exec:java -Dexec.mainClass="App"
4. Access the application via the terminal.

5. Troubleshooting
    - Database Connection Issues: Verify that PostgreSQL is running and the database credentials are correct
    - Maven Build Failures: Ensure all dependencies are correctly defined in the ‘pom.xml’ file.

6. Common Errors and Solutions
    - Error: ‘java.sql.SQLException: No suitable driver found’
        - Solution: Ensure the PostgreSQL JDBC driver is included in your project dependencies
    - Error: ‘ERROR: relation “users” does not exist’
        - Solution: Ensure the ‘Users’ table is created with the ‘vintique’ database