**Development Documentation Outline**

1. **Javadocs**
   * Describe how to generate Javadocs.
   * Provide a summary of the main classes and methods.
2. **Source Code Directory Structure**
   * Outline the directory structure of the project.
   * Describe what each directory contains.
3. **Build Process**
   * Provide instructions on how to compile and run the project.
   * Include any necessary build tools or commands.
4. **Compiler Time Dependencies**
   * List the dependencies required during compilation.
   * Mention any external libraries or tools used.
5. **Development Standards**
   * Describe coding standards (e.g., naming conventions, code formatting).
   * Outline version control practices.
6. **Database Setup**
   * Provide steps to set up the PostgreSQL database for development.
   * Include any necessary scripts or configuration files.
7. **Source Code Repository**
   * Explain how to obtain the source code from the repository.
   * Include instructions for cloning the repository and setting up the development environment.

**Class: App**

**Description:** The main class of the application that sets up the database connection, initializes DAOs and services, and starts the console menu.

**Methods:**

* **public static void main(String[] args):**  
  The main method that serves as the entry point of the application. It connects to the database, initializes DAOs and services, starts the console menu, and closes resources.
* **private static void connectToDatabase():**  
  Establishes a connection to the PostgreSQL database using JDBC.
* **private static void closeConnection():**  
  Closes the database connection if it is open.

**Class: ProductDAO**

**Description:** Provides data access methods for the Product entity, including adding, retrieving, updating, and deleting products in the database.

**Constructor:**

* **public ProductDAO(Connection connection):**  
  Initializes the ProductDAO with a database connection.

**Methods:**

* **public void addProduct(Product product) throws SQLException:**  
  Adds a new product to the database.
* **public Product getProduct(int itemId) throws SQLException:**  
  Retrieves a product by its ID from the database.
* **public void updateProduct(Product product) throws SQLException:**  
  Updates an existing product in the database.

**Class: UserDAO**

**Description:** Provides data access methods for the User entity, including adding, retrieving, updating, and deleting users in the database.

**Constructor:**

* **UserDAO(Connection connection):**  
  Initializes the UserDAO with a database connection.

**Methods:**

* **void saveUser(User user):**  
  Adds a new user to the database.
* **User getUserByUsername(String username):**  
  Retrieves a user by their username from the database.
* **void updateUser(User user):**  
  Updates an existing user in the database.
* **void deleteUser(int userId):**  
  Deletes a user by their ID from the database.
* **User getUserById(int userId):**  
  Retrieves a user by their ID from the database.
* **List<User> getAllUsers():**  
  Retrieves all users from the database.

**Class: DatabaseConnection**

**Description:** Handles the connection to a PostgreSQL database.

**Fields:**

* **URL:** The URL of the PostgreSQL database.
* **USER:** The username for the database.
* **PASSWORD:** The password for the database.

**Methods:**

* **main(String[] args):**  
  The entry point of the application. It loads the PostgreSQL JDBC driver, establishes a connection to the database, prints a success message, and then closes the connection. Handles ClassNotFoundException and SQLException.

**Class: AdminMenu**

**Description:** Provides an administrative menu for managing users and products.

**Fields:**

* **Scanner scanner:** An instance of Scanner for reading user input.
* **UserService userService:** An instance of UserService for managing user-related operations.
* **ProductService productService:** An instance of ProductService for managing product-related operations.

**Constructor:**

* **AdminMenu(Scanner scanner, UserService userService, ProductService productService):**  
  Initializes the AdminMenu with the provided Scanner, UserService, and ProductService.

**Methods:**

* **static void displayMenu(Scanner scanner, UserService userService, ProductService productService):**  
  Static method to display the menu and start the interaction.
* **void start():**  
  Starts the menu loop, displaying options and handling user choices.
* **void viewAllUsers():**  
  Displays a list of all users (implementation not shown in the excerpt).
* **void deleteUser():**  
  Deletes a user based on input (implementation not shown in the excerpt).
* **void viewAllProducts():**  
  Displays a list of all products (implementation not shown in the excerpt).

**Class: BuyerMenu**

**Description:** Provides a menu interface for buyers to interact with the product service.

**Fields:**

* **Scanner scanner:** Used to read user input.
* **ProductService productService:** Service to interact with product data.

**Constructor:**

* **BuyerMenu(Scanner scanner, ProductService productService):**  
  Initializes the BuyerMenu with a Scanner and ProductService.

**Methods:**

* **static void displayMenu(Scanner scanner, ProductService productService):**  
  Static method to display the menu and start the interaction.
* **void start():**  
  Starts the menu loop, displaying options and handling user input.
* **private void viewProducts():**  
  Displays all products by calling the ProductService.
* **private void searchProducts():**  
  (Not fully shown in the excerpt) Presumably searches for products based on user input.

**Class: ConsoleMenu**

**Description:** Provides a console-based menu for user interactions, such as registering and logging in users.

**Fields:**

* **Scanner scanner:** Used for reading user input from the console.
* **UserService userService:** Service for handling user-related operations.
* **ProductService productService:** Service for handling product-related operations.

**Constructor:**

* **public ConsoleMenu(Scanner scanner, UserService userService, ProductService productService):**  
  Initializes the ConsoleMenu with the provided Scanner, UserService, and ProductService.

**Methods:**

* **public void start():**  
  Starts the console menu loop, displaying options for registering, logging in, and exiting.
* **private void registerUser():**  
  Prompts the user to enter a username for registration.
* **private void loginUser():**  
  (Not shown in the excerpt) Presumably handles user login functionality.

**Class: SellerMenu**

**Description:** Manages the seller's menu options and interactions.

**Methods:**

* **private void viewProducts():**  
  Displays all products associated with the seller.
* **private void updateProduct():**  
  Updates the details of a product based on user input.
* **private void deleteProduct():**  
  Deletes a product based on the provided product ID.

**Class: ProductService**

**Description:** Provides services related to product management, acting as an intermediary between the application and the data access layer.

**Methods:**

* **public void addProduct(Product product) throws SQLException:**  
  Adds a new product to the database.
* **public Product getProduct(int itemId) throws SQLException:**  
  Retrieves a product by its ID.
* **public void updateProduct(Product product) throws SQLException:**  
  Updates the details of an existing product.
* **public void deleteProduct(int itemId) throws SQLException:**  
  Deletes a product by its ID.
* **public List<Product> getProductsBySeller(int sellerId):**  
  Retrieves a list of products associated with a specific seller.
* **public List<Product> getAllProducts():**  
  Retrieves a list of all products.
* **public Product getItemById(int itemId) throws SQLException:**  
  Fetches a single product by its ID.
* **public List<Product> searchProducts(String itemName) throws SQLException:**  
  Searches for products by name. Throws IllegalArgumentException if the item name is null or empty.

**Class: UserService**

**Description:** Responsible for handling user-related operations such as retrieving, deleting, registering, and logging in users. It interacts with the UserDAO class to perform database operations.

**Constructor:**

* **UserService(UserDAO userDAO):**  
  Initializes the UserService with a UserDAO instance.

**Methods:**

* **User getUserById(int userId) throws SQLException:**  
  Retrieves a user by their ID.
* **void deleteUser(int userId) throws SQLException:**  
  Deletes a user by their ID.
* **List<User> getAllUsers() throws SQLException:**  
  Retrieves a list of all users.
* **void registerUser(User user) throws SQLException:**  
  Registers a new user by encrypting their password using BCrypt and saving the user to the database.
* **User loginUser(String username, String password) throws SQLException:**  
  Logs in a user by verifying the username and password against the stored hashed password.

**Class: PasswordUtil**

**Description:** Provides utility methods for hashing and verifying passwords using the BCrypt library.

**Methods:**

* **public static String hashPassword(String plainTextPassword):**  
  Hashes a plaintext password using BCrypt.
* **public static boolean checkPassword(String plainTextPassword, String hashedPassword):**  
  Verifies that a plaintext password matches a hashed password.

**Class: Admin**

**Description:** Represents an admin in the system.

**Constructors:**

* **Admin():**  
  Default constructor.
* **Admin(int userId, String username, String password, String email):**  
  Initializes an admin with the given details.

**Methods:**

* **void deleteUser(User user):**  
  Deletes a user.
* **void viewAllUsers():**  
  Views all users.
* **void viewAllProducts():**  
  Views all products.
* **String toString():**  
  Returns a string representation of the admin.

**Class: Buyer**

**Description:** Represents a buyer in the system, inheriting from the User class.

**Constructors:**

* **Buyer(Connection connection):**  
  Default constructor that initializes the buyer with a database connection.
* **Buyer(int userId, String username, String password, String email, Connection connection):**  
  Parameterized constructor that initializes the buyer with user details and a database connection.

**Methods:**

* **void browseProducts():**  
  Method to browse available products.
* **void viewProductDetails(int productId) throws SQLException:**  
  Method to view details of a specific product by its ID.

**Class: Product**

**Description:** Represents a product in the system.

**Constructor:**

* **Product(int itemId, String itemName, String itemType, String itemDescription, int sellerId):**  
  Initializes a product with the given details.

**Methods:**

* **int getItemId():**  
  Gets the item ID.
* **void setItemId(int itemId):**  
  Sets the item ID.
* **String getItemName():**  
  Gets the item name.
* **void setItemName(String itemName):**  
  Sets the item name.
* **String getItemType():**  
  Gets the item type.
* **void setItemType(String itemType):**  
  Sets the item type.
* **String getItemDescription():**  
  Gets the item description.
* **void setItemDescription(String itemDescription):**  
  Sets the item description.
* **int getSellerId():**  
  Gets the seller ID.
* **void setSellerId(int sellerId):**  
  Sets the seller ID.
* **String toString():**  
  Returns a string representation of the product.

**Class: Seller**

**Description:** Represents a seller in the system.

**Constructor:**

* **Seller(int userId, String username, String password, String email):**  
  Initializes a seller with the given details.

**Methods:**

* **String getStoreName():**  
  Gets the store name.
* **List<Product> getProducts():**  
  Gets the list of products.
* **void addProduct(Product product):**  
  Adds a product to the seller's list of products.
* **void removeProduct(int productId):**  
  Removes a product from the seller's list of products by product ID.
* **void updateProduct(Product product):**  
  Updates a product in the seller's list of products.

**Class: User**

**Description:** Represents a user in the system.

**Constructors:**

* **User():**  
  Default constructor.
* **User(int userId, String username, String password, String email, String role):**  
  Initializes a user with the given details.

**Methods:**

* **int getUserId():**  
  Gets the user ID.
* **void setUserId(int userId):**  
  Sets the user ID.
* **String getUsername():**  
  Gets the username.
* **void setUsername(String username):**  
  Sets the username.
* **String getPassword():**  
  Gets the password.
* **void setPassword(String password):**  
  Sets the password.
* **String getEmail():**  
  Gets the email.
* **void setEmail(String email):**  
  Sets the email.
* **String getRole():**  
  Gets the role.
* **void setRole(String role):**  
  Sets the role.

project-root/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── main/

│ │ │ ├── util/

│ │ │ │ └── PasswordUtil.java

│ │ │ └── [other packages and classes]

│ │ └── resources/

│ │ └── [resource files]

│ ├── test/

│ │ ├── java/

│ │ │ └── main/

│ │ │ ├── util/

│ │ │ │ └── PasswordUtilTest.java

│ │ │ └── [other test packages and classes]

│ │ └── resources/

│ │ └── [test resource files]

├── pom.xml

└── [other project files]

**src/:** The main source directory for the project.

**main/:** Contains the main application code and resources.

**java/:** Contains the Java source files.

**main/:** The base package for the main application code.

**util/:** Contains utility classes, such as PasswordUtil.java.

[other packages and classes]: Other packages and classes for the main application.

**resources/:** Contains resource files needed by the main application (e.g., configuration files, templates).

**test/:** Contains the test code and resources.

**java/:** Contains the Java test source files.

**main/:** The base package for the test code.

**util/:** Contains test classes for utility classes, such as PasswordUtilTest.java.

[other test packages and classes]: Other packages and classes for testing.

**resources/:** Contains resource files needed for testing (e.g., test data, mock files).

**pom.xml:** The Maven Project Object Model file, which contains project configuration details, dependencies, and build instructions.

[other project files]: Any additional files related to the project (e.g., README, .gitignore).

**JBCrypt:**

Group ID: org.mindrot

Artifact ID: jbcrypt

Version: 0.4

**PostgreSQL JDBC Driver:**

Group ID: org.postgresql

Artifact ID: postgresql

Version: 42.7.3

External Libraries or Tools Used

**JBCrypt:** A library for hashing passwords using the BCrypt algorithm.

**PostgreSQL JDBC Driver:** A driver for connecting Java applications to PostgreSQL databases.

Naming Conventions:

Classes: Use PascalCase (e.g., DatabaseConnection).

Methods: Use camelCase (e.g., connectToDatabase).

Variables: Use camelCase (e.g., databaseUrl).

Constants: Use UPPER\_SNAKE\_CASE (e.g., MAX\_CONNECTIONS).

Code Formatting:

Indentation: Use 4 spaces per indentation level.

Line Length: Limit lines to 80-100 characters.

Braces: Use K&R style for braces.

Comments: Use Javadoc for public methods and classes. Use inline comments sparingly.

**Database Setup**

Install PostgreSQL.

Start the PostgreSQL service.

Create a new database and user.

Set up the database schema using a SQL script.

Configure your Java application to connect to the PostgreSQL database using a properties file.

Clone the repository using git clone.

Navigate to the project directory.

Install Java and Maven.

Set up PostgreSQL and create the necessary database and user.

Run the schema script to set up the database schema.

Build the project using Maven.

Run the application using Maven.