Project Report
Online Shopping System

---

1. Introduction

The **Online Shopping System** is a Java-based desktop application designed to facilitate secure online shopping. The system enables users to browse products, manage their shopping cart, place orders, and generate invoices. This report provides a detailed overview of the implementation, challenges faced, and future improvements.

2. Project Scope

The system is developed with a strong emphasis on Java technologies and Object-Oriented Programming principles. The key features include:

- **User Registration & Authentication:** Secure login and session management using Java.
- **Product Management:** Users can view, add, and remove products in the shopping cart.
- **Shopping Cart:** A well-structured cart system to manage user-selected items.
- **Order Processing & Invoicing:** Generating order receipts using Java I/O.
- **Admin Controls:** Admin functionality for managing product inventory.
- **Graphical User Interface (GUI):** A Java Swing-based user interface for easy interaction.

3. Technologies Used

The system is entirely built using Java-based technologies, ensuring efficiency and portability:

- **Core Java:** Object-oriented programming (OOP) principles, exception handling, file handling.
- **Java Swing:** GUI design and user interaction.
- **Java Collections Framework (JCF):** Used for managing product lists, user data, and order processing.
- **Java I/O:** Handling file operations for data storage, order history, and invoice generation.
- **JavaFX (Optional):** Enhancing UI with modern styling.
- **JUnit:** Unit testing framework for testing core functionalities.
- **Apache POI (Optional):** For exporting invoices and reports in Excel format.

---

4. Implementation Details

Features Implemented

- **User Registration & Authentication:**
  - Implemented using Java serialization for user data storage.
  - Uses SHA-256 hashing for password security.
- **Product Catalog & Shopping Cart:**
  - Implements `ArrayList` for storing product details dynamically.
  - Provides add, remove, and update functionalities.
- **Order Processing & Invoicing:**
  - Uses Java I/O (`FileWriter`, `BufferedWriter`) to store order history.
  - Generates order summary and invoices as text files.
- **Admin Panel:**
  - Allows admins to add and remove products from the system.
  - Implements `JTable` for product display and inventory management.
- **GUI Components:**
  - Java Swing-based graphical interface with `JFrame`, `JPanel`, and `JButton` elements.
  - Uses `ActionListener` for event-driven interactions.

Features Yet to be Implemented

- **Multi-user Support:** Implementing session management for different user roles.
- **Payment Gateway Simulation:** Implementing a mock payment system for order checkout.
- **Data Persistence:** Migrating from file-based storage to a lightweight embedded database.
- **Graphical Reports:** Enhancing analytics with JavaFX charts for admin insights.

5. System Architecture

The system follows a **modular architecture** with distinct layers:

1. **Presentation Layer (GUI):** Java Swing-based UI for user interactions.
2. **Business Logic Layer:** Core functionality written in Java, including cart operations, order management, and authentication.
3. **Data Layer:** Uses Java serialization for storing user and product data persistently.

The application follows the **Model-View-Controller (MVC) architecture**, ensuring modularity and maintainability.

6. UML Diagrams

To illustrate the system's design and functionality, the following UML diagrams should be included:

6.1 Use Case Diagram

(Insert a diagram showing actors like Customer, Admin, and System interacting with various features.)

6.2 Class Diagram

(Insert a Class Diagram representing classes such as `User`, `Product`, `Cart`, `Order`, `Admin`, and their relationships.)

6.3 Sequence Diagram

(Insert a Sequence Diagram for order placement, from product selection to invoice generation.)

6.4 Activity Diagram

(Insert an Activity Diagram showing user workflows like login, browsing, and checkout.)

---

7. Challenges Faced

7.1 GUI Event Handling

- Managing multiple event listeners for different UI components caused **lag in user response**.
- Implemented **multithreading** to handle heavy UI actions smoothly.
- Used **SwingWorker** to prevent UI from freezing during long operations.

7.2 File Handling & Data Storage

- Java serialization needed careful handling to **prevent data corruption**.
- Implemented **synchronized read-write operations** to avoid file access conflicts.
- Used **BufferedReader & BufferedWriter** for efficient file operations.

7.3 Exception Handling

- Implemented **custom exception classes** for user authentication and cart operations.
- Used **try-catch-finally** blocks extensively to handle input validation errors.

7.4 Performance Optimization

- Initially, loading a large product list caused slowdowns.
- Optimized using **HashMap caching** to store frequently accessed products.
- Used **lazy loading** for fetching data only when needed.

8. Future Directions

- **Advanced UI Enhancements:** Transitioning from Swing to JavaFX for a modernized user interface.
- **Database Integration:** Moving from file-based storage to a more scalable data solution.

- **Automated Testing:** Enhancing reliability through comprehensive JUnit test cases.
- **Cloud Deployment:** Exploring options to deploy the system as a web-based Java application.
- **Barcode Scanner Integration:** Implementing barcode scanning for product management.

---

9. Deployment & System Environment Setup

9.1 System Requirements

- **Operating System:** Windows, Linux, or macOS.
- **Java Development Kit (JDK):** JDK 11 or higher.
- **IDE:** IntelliJ IDEA / Eclipse / NetBeans.
- **Libraries Used:**
  - Apache POI (Optional - for exporting data to Excel)
  - JUnit (for testing)

9.2 Deployment Steps

1. Install **JDK 11 or later**.
2. Clone or download the source code.
3. Open the project in **IntelliJ IDEA / Eclipse**.
4. Compile and run the `Main.java` file.
5. Execute `jar cf OnlineShopping.jar -C bin .` to generate an executable JAR file.
6. Run the application using `java -jar OnlineShopping.jar`.

---

10. Conclusion

The **Online Shopping System** effectively demonstrates Java-based application development using Swing, Java Collections, and file handling. It provides a user-friendly and functional shopping experience. Future enhancements will focus on **improving UI, performance, and scalability**, ensuring the system remains robust and adaptable.

11. Submission Details

The following files are included in the submitted ZIP file:

1. **Source Code:** Java files organized in a structured directory.
2. **README File:** Instructions on setting up and running the project.
3. **Key Screenshots:** Demonstrating major functionalities.
4. **UML Diagrams:** Class, Use Case, Sequence, and Activity diagrams.
5. **Test Cases:** JUnit test cases for critical functionalities.
6. **Executable JAR File:** Compiled version for easy execution.

7. **This Project Report:** Comprehensive documentation of the project.

End of Report