PROJECT REPORT

Object-Oriented Programming (CSL-210)

PROJECT TITLE: ScrunchieSphere

BS (CS) – 2B

Group Members

Name	Enrollment
1. Maria Shamim	02-134242-070
2. Alishba Ahmed	02-134242-006
3. Ayesha Nadeem	02-134242-093

Submitted to:

Ma’am Saba Imtiaz

BAHRIA UNIVERSITY KARACHI CAMPUS

Department of Computer Science

Problem Statement

The key challenges that this program seeks to solve include: The problem this project addresses is the lack of a comprehensive and user-friendly online shopping management system that integrates various business processes, such as product management, order processing, user interaction, and supplier coordination, into a single platform.

While there are many e-commerce systems available today, they often fail to provide a seamless and cohesive experience for all stakeholders — including **users** (customers), **suppliers**, and **administrators**.

Requirement Gathering Approach

Requirement gathering is a crucial phase in software development, particularly for building systems that meet the needs of stakeholders. In this case, the **interview method** was employed to gather detailed insights and specific requirements for the development of the **Online Shopping Management System** (OSMS) for **Scrunchienetix.pk**, a business specializing in e-commerce.

Interviews provide an effective way to directly communicate with business owners, stakeholders, or users to understand their needs, pain points, and expectations from the system. This approach allows the team to collect qualitative data, uncover hidden requirements, and ensure that the final system aligns with the business objectives.

Functional Requirements

1. Product Management:

- ✓ Difficulty in managing the inventory, updating product information, and tracking stock levels.
- ✓ Lack of a streamlined method for suppliers to update product details, prices, and availability.

2. Order Management:

- ✓ Complex and time-consuming process of handling customer orders, payments, and delivery status.
- ✓ Inability to track the status of orders in real-time.

3. User Management:

- ✓ Difficulty in managing user accounts, handling multiple types of users (such as customers and suppliers), and providing personalized experiences.
- ✓ Inadequate handling of user data, such as profile updates and order history.

4. Supplier Coordination:

- ✓ Lack of an effective platform for suppliers to manage their products, update stock, and process deliveries.
- ✓ Poor communication between suppliers and administrators regarding inventory and supply chain management.

5. Transaction Tracking:

- ✓ Inability to easily manage and track transactions, including the status of payments, refunds, and order completion.
- ✓ Difficulty in reconciling payments with orders for auditing and reporting purposes.

Non-Functional Requirements

1. Usability:

- ✓ The system should offer a user-friendly and intuitive interface for all user types.
- ✓ Easy navigation with mobile responsiveness and accessibility compliance (e.g., WCAG).

2. Performance

- ✓ The system should respond to user requests within 2 seconds under normal load.
- ✓ It must support concurrent users without significant performance degradation (e.g., at least 500 users simultaneously).

3. Scalability

- ✓ The platform must be scalable to handle increased numbers of users, products, and transactions over time.

4. Security

- ✓ Implement secure authentication and authorization mechanisms (e.g., JWT, OAuth).
- ✓ Encrypt sensitive user and transaction data (e.g., passwords, credit card information).
- ✓ Role-based access control for different types of users.

5. Reliability & Availability

- ✓ The system should be available at least 99.9% of the time (uptime SLA).
- ✓ Failover mechanisms must be placed for critical services.

6. Maintainability

- ✓ The system codebase should be modular to allow for easy updates and feature enhancements.
- ✓ Logging and error tracking must be implemented for troubleshooting.

7. Compatibility

- ✓ The system must be compatible with major browsers (Chrome, Firefox, Safari, Edge).
- ✓ Should support both desktop and mobile devices.

Entity and Relationship Analysis

Key Entities:

1. Admin:

- ✓ Attributes:
  - adminId (int)
- ✓ Methods:
  - manageProducts()
  - addProduct()
  - removeProduct()
  - updateInventory()

➢ Role: Manages the products in the system.

2. User:

- ✓ Attributes:
  - userId (int)
  - name (string)
  - email (string)
  - password (string)
- ✓ Methods:
  - login()
  - register()
  - browseProducts()
  - viewOrderHistory()
  - updateProfile()

➢ Role: Represents the customer interacting with the system.

3. Supplier:

- ✓ Attributes:
  - supplierId (int)
  - companyName (string)
  - contactInfo (string)
- ➢ Role: Provides products to the system.

4. Product:

- ✓ Attributes:
  - productId (int)
  - productName (string)
  - description (string)
  - price (double)
  - stock (int)
- ➢ Role: Represents the goods available for purchase.

5. Order:

- ✓ Attributes:
  - orderId (int)
  - totalAmount (double)
  - orderDate (string)
- ✓ Methods:
  - placeOrder()
  - viewOrderDetails()
  - updateOrderStatus()
- ➢ Role: Represents a customer's purchase.

6. OrderItem:

- ✓ Attributes:
  - orderId (int)
  - productId (int)
  - quantity (int)
  - price (double)
- ✓ Method:
  - calculateTotal()
- ➢ Role: Represents individual items in an order.

7. Transaction:

- ✓ Attributes:
  - transactionId (int)
  - paymentStatus (string)
  - paymentDate (string)
- ✓ Methods:
  - processPayment()
  - viewTransactionDetails()

➢ Role: Handles the payment for orders.

Relationship between Entities:

1. **User to Order:**
   - ✓ A user can place multiple orders (one-to-many).
2. **Order to OrderItem:**
   - ✓ An order contains multiple order items (composition relationship, one-to-many).
3. **OrderItem to Product:**
   - ✓ Each order item is associated with a product (many-to-one).
4. **Transaction to Order:**
   - ✓ Each transaction corresponds to a single order (one-to-one).
5. **Admin to User:**
   - ✓ Admin can manage users indirectly by managing products they purchase.
6. **Product to Supplier:**
   - ✓ Products are sourced from suppliers (many-to-one).

OOP Application

Encapsulation:

- Definition:
  - ✓ Bundling data and methods that operate on that data within a single unit (class).
- Application:
  - ✓ Each class (like User, Order, Product) encapsulates its data as attributes (e.g., userId, orderDate, price) and related methods (e.g., login(), placeOrder(), calculateTotal()).
  - ✓ Access to data is controlled through methods, promoting data hiding. For instance, user details are private, and methods like updateProfile() manage updates.

Inheritance:

- **Definition:**
  - ✓ Deriving new classes from existing ones to inherit attributes and methods.
- **Application:**
  - ✓ While the diagram does not show explicit inheritance, the structure suggests that some classes (like Admin and User) could share common attributes and methods if they were derived from a more generic Person class.
  - ✓ The absence of inheritance here might indicate a design choice to keep Admin and User separate due to their distinct roles.

Polymorphism:

- **Definition:**
  - ✓ Ability to present the same interface for different underlying data types.
- **Application:**
  - ✓ The methods like placeOrder() and processPayment() can be overridden or extended if new types of orders or payment methods are added in the future.
  - ✓ For example, if the system later supports different types of users (e.g., GuestUser and RegisteredUser), they could share the browseProducts() method while implementing it differently.

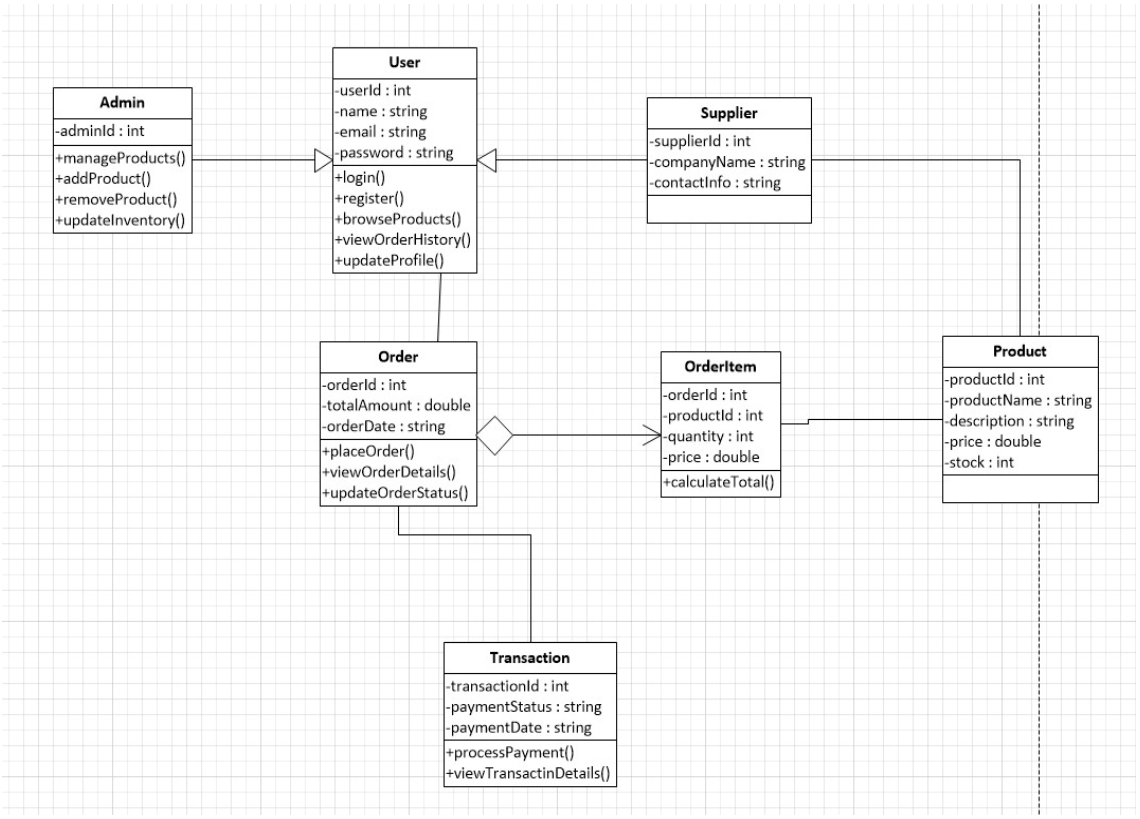
Abstraction:

- **Definition:**
  - ✓ Hiding complex implementation details and showing only essential features.
- **Application:**
  - ✓ The Transaction class abstracts payment processing details from the Order class.
  - ✓ Users interact with methods like placeOrder() without knowing the intricate details of how the order is processed or stored.
  - ✓ The OrderItem class encapsulates the calculation logic through the calculateTotal() method, abstracting away the detailed computation.

Association, Aggregation & Composition:

- **Association:**
  - ✓ A **user** can place multiple **orders** (one-to-many association).
- **Aggregation:**
  - ✓ The **OrderItem** and **Product** relationship is an aggregation because an order item references a product but does not own it.
- **Composition:**
  - ✓ The **Order** and **OrderItem** relationship is a composition (shown as a filled diamond). An order consists of multiple order items, and if an order is deleted, its items are also deleted.

UML Diagram



This UML diagram represents an **E-commerce Management System**. It models the relationships and interactions between various entities, including:

1. **Admin:** Manages products and inventory.
2. **User:** Registers, logs in, browses products, and places orders.
3. **Supplier:** Provides product details and contact information.
4. **Product:** Holds product details like ID, name, description, price, and stock.
5. **Order:** Represents a user's purchase with a total amount and date.
6. **OrderItem:** Contains details of individual items within an order.
7. **Transaction:** Manages payment processing and status.

The diagram effectively applies OOP concepts like **encapsulation** and **association**, showing relationships like **composition** between Order and OrderItem.

Future Development

The future development of the E-commerce Management System can focus on enhancing the system's capabilities, improving user experience, and scaling it for larger user bases. Below are some suggestions for the future development of this program:

1. Enhance User Experience:

- **User Roles and Permissions**: Introduce roles like **Customer**, **Admin**, **Supplier**, and **Guest** with specific permissions.
- **Personalization:** Implement personalized product recommendations based on previous orders.
- **Feedback and Reviews:** Allow users to leave reviews and ratings for products.

2. Advanced Order Management:

- **Order Tracking**: Integrate real-time tracking and notifications for order status changes.
- **Order Cancellation/Modification:** Allow users to modify or cancel orders before shipment.

3. Inventory and Product Management

- **Automated Inventory Updates:** Update product stock automatically upon successful order placement.
- **Low Stock Alerts:** Notify admins when product quantities fall below a threshold
- **Supplier Integration:** Allow suppliers to update product details and availability directly.

4. Payment and Transaction Security:

- **Multiple Payment Options:** Integrate digital wallets (like PayPal, Apple Pay) and cryptocurrency payments.
- **Secure Transactions:** Implement **encryption** and **tokenization** for payment data.
- **Refund and Dispute Handling:** Manage refunds and resolve transaction disputes within the system.

5. Data Analysis and Reporting:

- **Sales Analytics**: Generate reports on sales trends, revenue, and customer behavior.
- **Product Performance Tracking:** Analyze best-selling and slow-moving products.
- **Customer Insights:** Identify loyal customers and offer targeted discounts.

6. Scalability and Performance:

- **Cloud Integration:** Migrate data to cloud databases for scalability.
- **Load Balancing:** Distribute traffic across multiple servers to maintain performance during high demand.
- **Data Backup:** Implement automated backups for critical data.

7. Integrations and Extensions:

- **Third-Party APIs:** Integrate with shipping providers for real-time logistics.
- **Social Media Integration:** Allow users to share purchased products on social platforms.
- **Mobile Application**: Develop a mobile version for convenient access.

Conclusion

This system, based on OOP principles and an effective UML design, serves as a robust foundation for an e-commerce platform. The current model efficiently handles basic functionalities like user management, order processing, and payment handling.

However, to stay competitive and relevant, future development should focus on:

- **Enhancing the user experience** through personalization and efficient order management.
- **Improving scalability** and security to handle larger customer bases and secure transactions.
- **Integrating analytics** to make data-driven decisions for business growth.

By incorporating these improvements, the system will evolve into a full-featured, scalable, and secure e-commerce solution, catering to both small businesses and large enterprises.