SWE30003 - SOFTWARE ARCHITECTURES AND DESIGN

ASSIGNMENT 2 – OBJECT DESIGN

TRONG NHAN VO - 104757410

MINH CAO - 104179289

TAHMIDUL HAQUE CHOWDHURY - 103541308

PHUONG BAO MINH NGUYEN - 104339685

WORD COUNT: 2620 WORDS

Executive Summary

AWE Electronics, a mid-sized retailer based in Hawthorn, aims to expand its reach through the development of a comprehensive online shopping platform. This digital transformation will enable customers across Australia to conveniently browse products, place orders, and have them delivered to their doorstep. The proposed system supports customer account management, shopping cart functionality, secure payments, invoice generation, order tracking, and administrative tools for product and sales management.

This document presents the object-oriented design for the Online Electronics Store. It outlines the system's core structure using a Responsibility-Driven Design approach, identifies candidate classes, and illustrates class relationships through a UML class diagram. Additionally, CRC cards are used to define class responsibilities and interactions. Design heuristics and patterns such as Singleton, Factory Method, and Strategy are incorporated to enhance system modularity, scalability, and maintainability.

The design is verified against a series of non-trivial usage scenarios and includes an overview of the system's bootstrap process. Collectively, these components provide a clear and practical blueprint for system implementation that aligns with the goals and requirements defined in the Software Requirements Specification.

Table of Contents

1. Introduction

To support its expansion from a local offline store to a nationwide online and offline retail presence, AWE Electronics requires a robust online shopping platform capable of efficiently managing increased order volumes. The new e-commerce system will enhance customer reach across Australia by delivering a seamless shopping experience, featuring intuitive account management, an easy-to-navigate product catalog, secure cart operations, reliable payment processing, and real-time order tracking. This digital transformation ensures AWE Electronics remains competitive in the growing e-commerce market while maintaining high standards of usability, security, and performance.

This document provides object design for the online shopping platform of AWE Electronics by satisfying the requirements of the store. The object design will identify candidate classes, their associated roles, their messages, and the design of the system will be represented using a UML diagram. It is important to implement functionality for the online shopping platform with four primary components in the system: payment processing, account administration, an online catalog, and shopping cart. The object design has been developed to satisfy the requirements in the “Case Study Online Electronics Store System-2025-s1.doc”, plus operational constraints identified in the case study provided in Assignment 1; therefore, the proposed online shopping platform will be built from a disciplined approach to allow for future upgrades and enhancements in addition to fulfilling operational requirements.

In summary, the document considers the object design, justifies the design patterns, and presents the solution through detailed class description and interaction diagrams. This will provide comprehensive blueprints for all stakeholders and developers involved with the development and implementation of the system.

1.1. Outlook of solution

The solution is designed to work with the store's current database system while keeping different parts organized. The database is only for storing data and handling SQL commands. When the system starts, a lasting connection to the database is set up and stays active as long as the app runs. This connection is shared with any business objects needing database access using constructor injection. This helps in managing resources well and keeps the application and data storage loosely connected. This method has many advantages: it keeps the database independent, makes sharing connections efficient, and allows for testing if needed. It also supports future growth, so the database can be changed or expanded without affecting the business logic.

1.2. Documentation and guidelines for interface

Identifier	Rules	Examples
Classes	Name for classes need to be descriptive and simple. Class names need to be capitalized	class Admin class OrderItem
Interfaces	Interface identifiers need to be capitalized	interface Payment
Variables	Variable names need to be short and meaningful	

2. Problem Analysis

The goal of the system is to expand AWE Electronics' business by developing an online shopping website. Based on the requirements outlined in the Software Requirements Specification (SRS) document, the system will provide a robust e-commerce platform tailored to the store's needs. This will be accomplished by meeting the following requirements:

- Record and manage customer accounts, allow customers to update their details, and provide a guest checkout option
- Maintain a database of product information, enable keyword and category-based search, allow customers to save products to a wishlist, and update stock levels in real time.

- Enable cart management (add/remove/update quantities), discount application, delivery selection, address autofill, and payment preview with prices/taxes/delivery dates.
- Support multiple payment methods, retry failed transactions with alternative options, and automatically email PDF invoices post-purchase.
- Email delivery status updates and allow issue reporting for damaged/missing items.
- Manage products (add/edit/remove), bulk upload via CSV, organize categories, set low-stock alerts, and generate sales reports.

To enhance the system's capabilities, it will integrate with digital marketing services to boost customer engagement and deliver targeted promotions. Additionally, the system will generate real-time sales reports for managers, offering actionable insights into daily website operations. This feature will support data-driven decision-making, enabling more efficient business strategies.

2.1. Assumptions

The following assumptions have been made in the process of creating the object design:

A1 Only electronic devices are sold on the website

A2 A product in this context describes an electronic device

A3 Different products will have unique barcodes and serial numbers

A4 All products will have specific names, descriptions, manufacturer and price

A5 If customers want to make a purchase, they must provide their names, addresses and their bank account details

A6 All products and their details will be registered in the system

A7 All customers have a unique id, name, addresses and phone number

A8 After each payment a receipt is sent to the customer

A9 Integration of the website, including servers, databases and security protocols, will be supported by AWE Electronics

A10 Customer data collected for sales report will be sufficient and accurate

A11 If a customer has not made a purchase before, they will be assigned a unique customer ID for their first purchase

A12 Once the customers' details are registered, they can never be removed in the database

A13 The system resides in a secure environment so that payment information does not have to be encrypted

A14 A customer is permitted to make multiple payment instalments provided the balance is not exceeding the 30th day after purchase.

A15 Inventory updates will reflect in real-time to prevent overselling

A16 Order delivery is completed by third party delivery company

2.2. Simplification

To improve efficiency in the system, these simplifications have been made:

- Once entered in the system, the orders from registered members and guests will be treated in the same manner in the database and therefore will be treated the same in the application

2.3. Design Justification

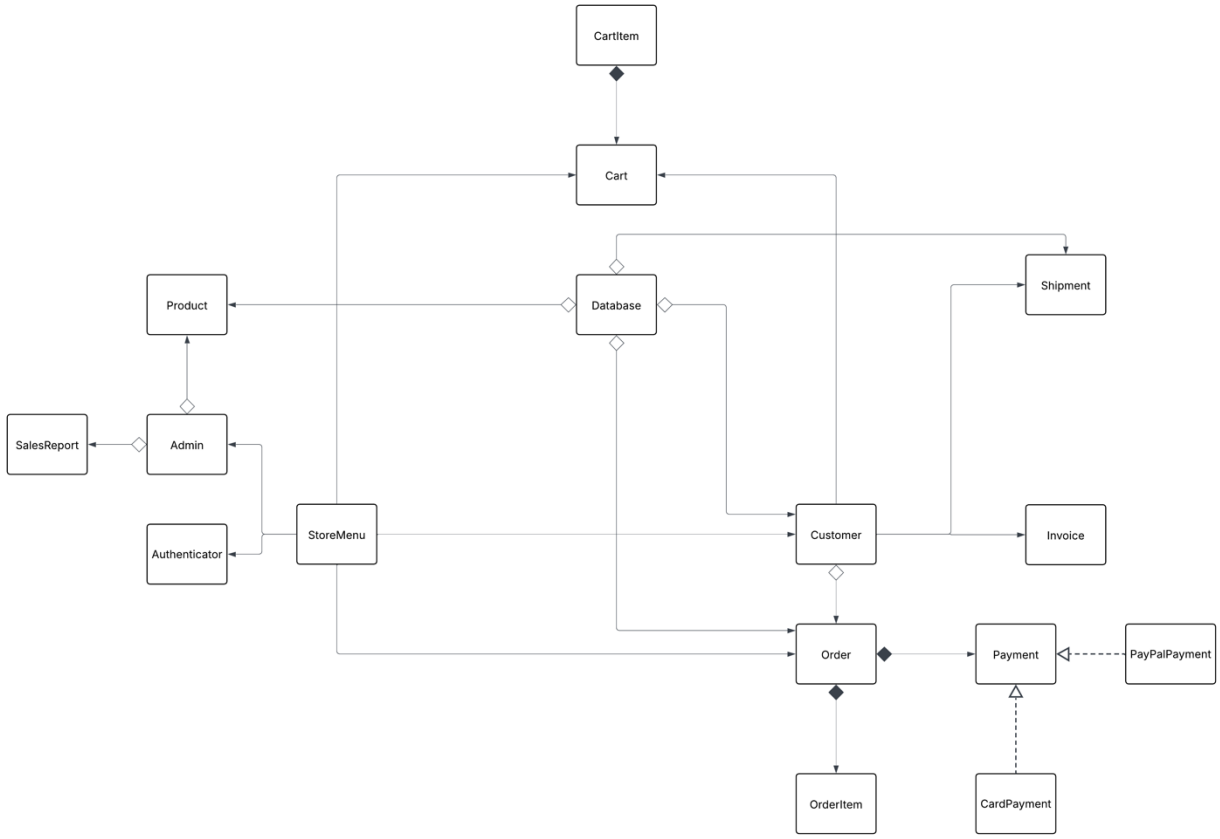
The system is built to connect with the store's existing database. It uses specific classes that control access to the database. Each class keeps its data protected, storing only what it needs and having set permissions to access certain information. This setup allows classes to function efficiently with minimal data exposure. Classes with the right permission can convert raw data into valuable business insights. By controlling how classes interact with the database, the design keeps the application logic separate from data storage. This separation prevents data leaks and ensures operations are secure and efficient. The architecture also supports future growth. If the database structure needs to change, the application layer requires only minor adjustments because each class's data needs and permissions are clearly defined and managed.

3. Candidate Classes

3.1. Candidate class list

- Customer
- Admin
- Product
- Cart
    - CartItem
- Order
    - OrderItem
- Invoice
- Shipment
- Payment
    - CardPayment
    - PayPalPayment
- SalesReport
- Database
- StoreMenu
- Authenticator
- DiscountEngine
- Notifier

3.2. UML Diagram



3.3. CRC Cards

Customer

Attribute	**Details**
Super Class	--
Responsibilities	Create and update accounts, browse products, manage cart, place orders, view invoices and order history
Collaborators	Cart, Order, Invoice, Shipment, StoreMenu, Payment

Product

Attribute	**Details**
Super Class	--
Responsibilities	Store and provide product details (name, description, price, stock), update stock
Collaborators	Admin, CartItem, OrderItem, Database

Cart

Attribute	Details
Super Class	--
Responsibilities	Add/remove/update products, calculate subtotal, clear cart
Collaborators	CartItem, Customer, Product

Order

Attribute	Details
Super Class	--
Responsibilities	Store order details, link to customer, generate invoice, confirm shipment
Collaborators	OrderItem, Payment, Invoice, Shipment, SalesReport

Payment

Attribute	Details
Super Class	Abstract Interface
Responsibilities	Define method to process payment
Collaborators	CardPayment, PayPalPayment, Order

Admin

Attribute	Details
Super Class	--
Responsibilities	Manage product catalogue, view customer data, generate sales reports
Collaborators	Product, SalesReport, Database

CartItem

Attribute	Details
Super Class	--
Responsibilities	Store product and quantity added to cart
Collaborators	Cart, Product

OrderItem

Attribute	Details
Super Class	--
Responsibilities	Store individual product and quantity in an order
Collaborators	Order, Product

Invoice

Attribute	Details
Super Class	--
Responsibilities	Display total cost, taxes and payment confirmation
Collaborators	Order, Customer, Payment

Shipment

Attribute	Details
Super Class	--
Responsibilities	Track delivery status, assign tracking number
Collaborators	Order, Database

CardPayment

Attribute	Details
Super Class	Payment
Responsibilities	Track delivery status, assign tracking number
Collaborators	Order, Database

PayPalPayment

Attribute	Details
Super Class	Payment
Responsibilities	Process PayPal transactions
Collaborators	Order, Customer

SalesReport

Attribute	Details
Super Class	Singleton
Responsibilities	Aggregate order and payment data, generate reports
Collaborators	Admin, Order, Payment

Database

Attribute	Details
Super Class	--
Responsibilities	Persist and retrieve all system data (product, users, orders, etc.)

Collaborators	Product, Customer, Order, Shipment, Admin

StoreMenu

Attribute	**Details**
Super Class	--
Responsibilities	Serve as system entry point, handle navigation to modules
Collaborators	Customer, Admin, Authenticator, Cart, Order

Authenticator

Attribute	**Details**
Super Class	--
Responsibilities	Validate login, register new users, manage credentials
Collaborators	Customer, Admin, StoreMenu

DiscountEngine

Attribute	**Details**
Super Class	--
Responsibilities	Apply promo codes, validate discounts, calculate adjusted price
Collaborators	Cart, Product, Order

Notifier

Attribute	**Details**
Super Class	--
Responsibilities	Send confirmation emails and shipping updates
Collaborators	Customer, Order, Shipment

4. Design Quality

4.1. Design Heuristics

In our design, we followed many important heuristics so that the system is easy to maintain, scalable, and reliable. We will be discussing more about them below:

H1. Each class has one clear responsibility.

Every class handles only one key task. For example, the Cart manages cart operations, while the Notifier is responsible only for sending emails and shipping updates.

H2. Classes hide their internal data.

Each class keeps its data private. For instance, payment details in the Payment, CardPayment, and PayPalPayment classes are not accessible by unrelated classes.

H3. Superclass data stays private.

Our abstract Payment class stores important data privately. Subclasses can use this data only through defined interfaces, avoiding direct access.

H4. Consistent naming and design.

We kept class names, responsibilities, and UML diagrams consistent. This makes the design easy to understand for everyone in the team and future developers.

H5. No god classes.
We avoided creating any "god class." Responsibilities are distributed evenly. No class has too much control or too many tasks.

H6. Abstract classes are used only as base classes.

Our abstract Payment class does not implement any direct behavior. It only serves as a base for payment types like CardPayment and PayPalPayment.

H7. Scalability for future growth.

The design supports scalability by using flexible patterns and separating concerns. New payment methods or discount strategies can be added without changing the core classes.

5. Design Patterns

5.1. Creational Patterns

5.1.1. Factory Method

We used the Factory Method pattern for creating payment types. When a customer chooses a payment method, the system uses a factory method to create either a CardPayment or a

PayPalPayment object. This will allow us to add more payment types in the future without changing the existing code.

5.1.2. Singleton

The SalesReport class uses the Singleton pattern, so there's only one report generator in the system. This avoids duplicate reports and keeps everything consistent.

5.2. Behavioral Patterns

5.2.1. Strategy

The DiscountEngine uses the Strategy pattern, which makes it easy to apply different discount rules without changing the Cart or Product classes. We can also add new discount strategies later without many issues.

6. Bootstrap Process

The bootstrap process of the AWE Electronics Online Store ensures that all essential system components are initialized in a structured order before user interaction begins. It sets up both application logic (classes) and data storage (tables), enabling secure, role-based, and real-time e-commerce operations.

1. **Database Initialization**
   Upon system startup, a persistent database connection is established using constructor injection. This connection is shared across all classes that interact with data (**Product**, **Customer**, **Order**, **Cart**, **Payment**, **Invoice**, **OrderTracking**). The use of constructor injection helps decouple business logic from data storage, enhancing testability and maintainability.

2. **Session Controller Initialization**
   The **SessionController** class is initialized to manage both user authentication and interface routing. It handles login/registration processes and directs authenticated users to the appropriate modules based on their role (**Customer** or **Admin**). This unified controller simplifies session flow and UI redirection logic.

3. **Admin and Customer Class Activation**
   The **Admin** and **Customer** classes are initialized to handle role-specific tasks:

   - **Customer**: interact with products, manage cart, place orders, and view invoice/shipment status

- **Admin**: manage products, view customer data, generate reports, and maintain inventory

4. **Inventory Controller Setup**
   The **InventoryController** is initialized to manage stock levels in real time. It ensures synchronization between business operations and the **product** table, preventing overselling during transactions.

5. **Method Setup for Payments**
   The system configures the **BasePayment** base class and its subclasses (**CardPayment**, **PayPalPayment**) using a factory method. Based on user input, the correct payment subclass is instantiated, and payment data is recorded in the **payment** table.
6. **Invoice Export Service Activation**
   The Invoice module is initialized to generate PDF invoices after successful payments. These invoices are recorded in the **invoice** table and automatically emailed to customers through the **Notifier**.
7. **Tracking order Initialization**
   The **OrderTracking** is activated to assign tracking numbers and update order delivery statuses. It works closely with the tracking tables to provide real-time delivery visibility.

8. **Sales Reporting Generator**
   The **SalesReport** class is loaded as a singleton instance. It aggregates data from the **order**, **payment**, and **customer** tables to generate analytics reports for administrative users.

9. **Discount and Notification Setup**
   The **DiscountEngine** is initialized with pricing strategies used at checkout. Meanwhile, the **Notifier** prepares email templates and delivery protocols to send system notifications such as order confirmations, shipping updates, and invoices.

7. Verification

Use Case 1: User Browsing and Order

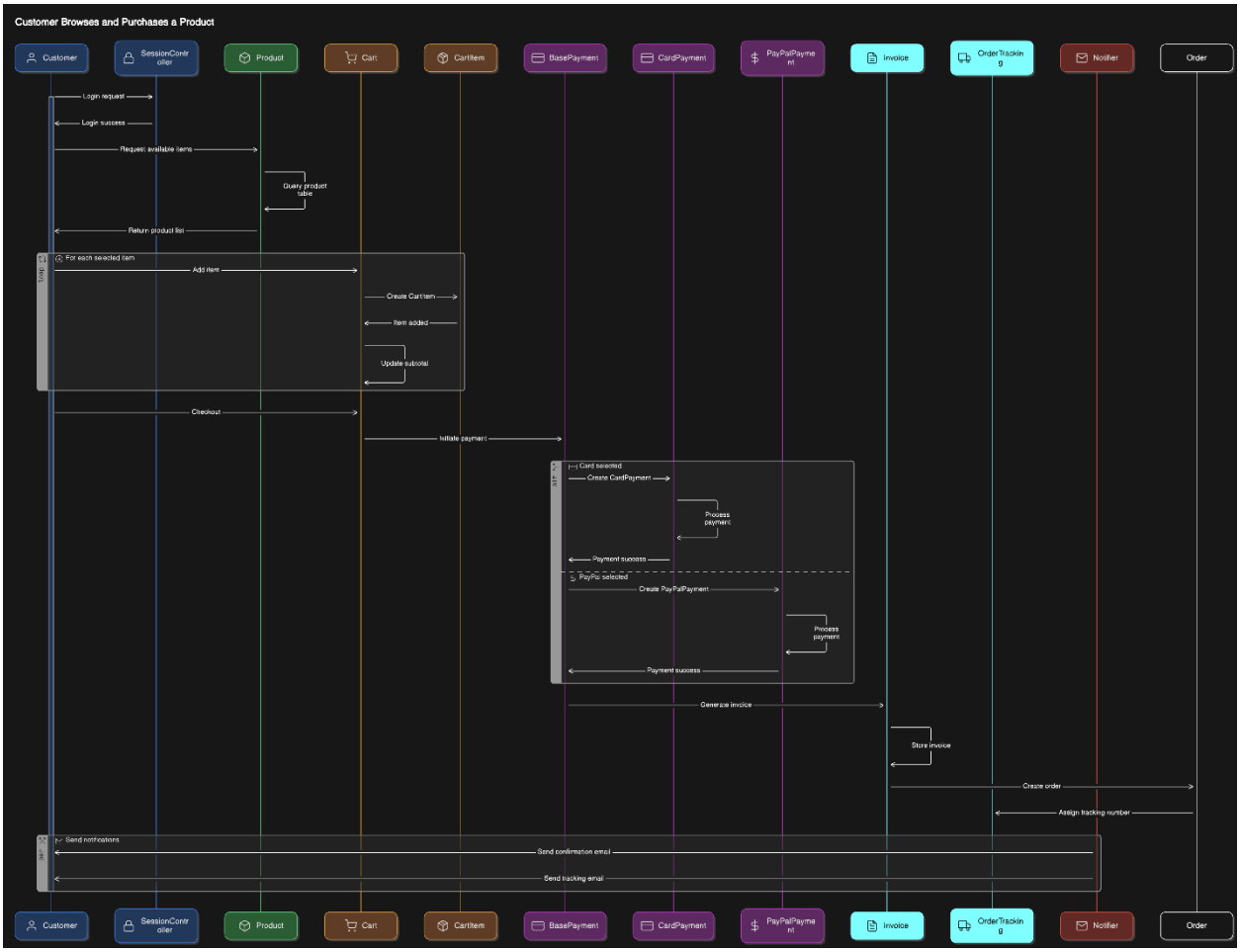
Description: A customer purchases products through the website.
Steps:

1. Customer logs in and browses products.
2. Adds items to the Cart.
3. Proceeds to checkout and selects payment method.
4. Payment is processed, and an Invoice is generated.
5. Order and tracking info are recorded.

6. Notifier sends confirmation email.

Outcome: Order is completed, payment is secured, and the customer receives order details.

Chart Link: Chart 1



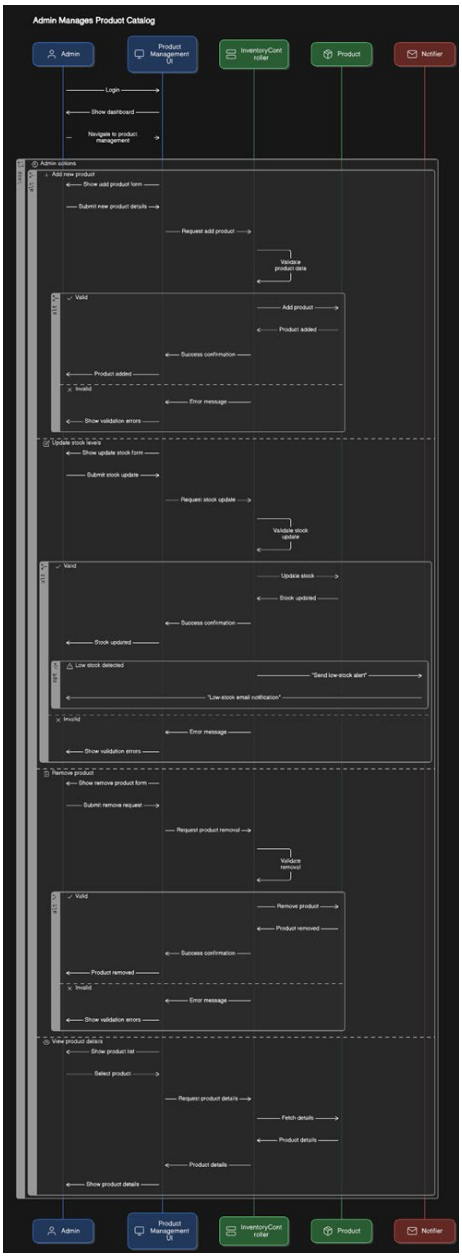
Use Case 2: Admin Managing Product Catalog

Description: Admin updates products and inventory.
Steps:

1. Admin logs in via the website.
2. Views or edits products via the **Product** class.
3. **InventoryController** updates stock levels.
4. Low-stock alerts are triggered if needed.

Outcome: Product data and stock levels are updated accurately in the system.

Chart Link: Chart 2



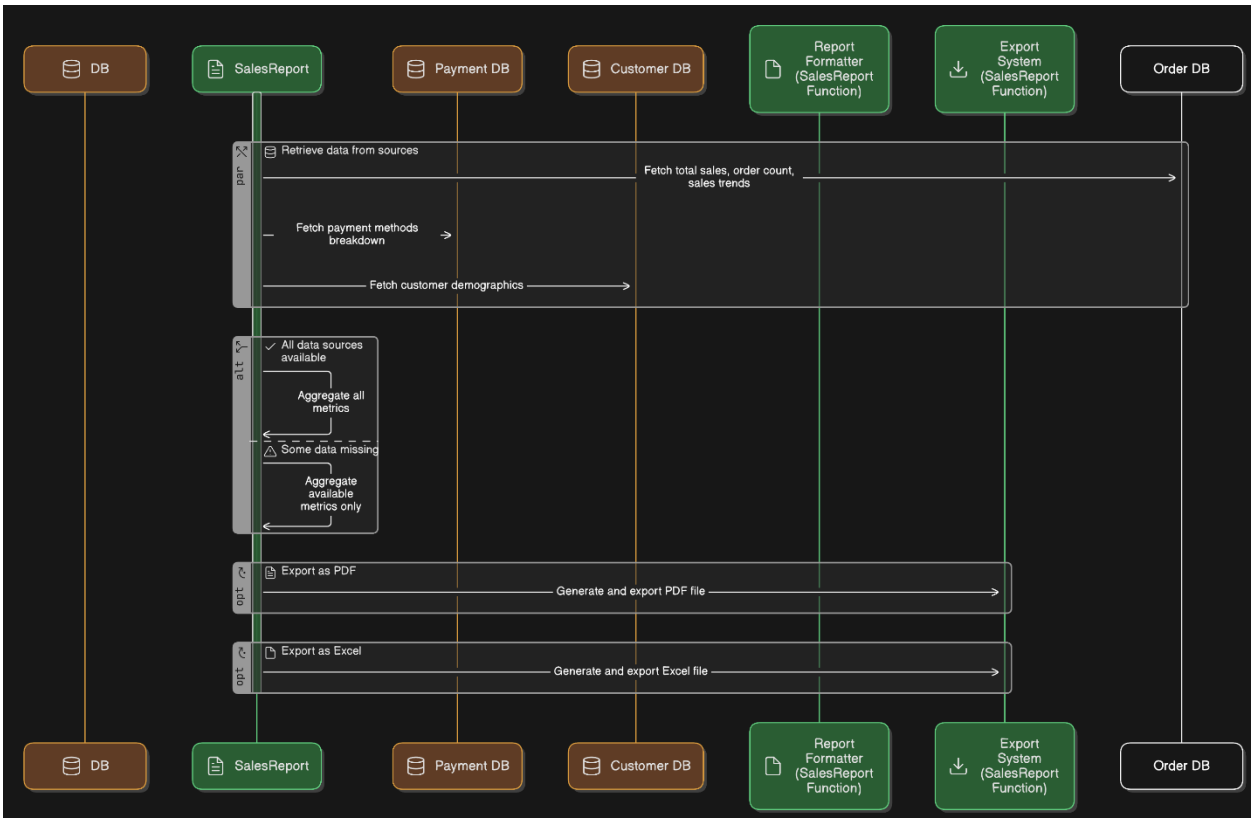
Use Case 3: Generating Sales Reports

Description: Admin reviews performance using **SalesReport**.
Steps:

1. Admin accesses **SalesReport** module.
2. System aggregates data from orders and payments.
3. Report is generated for review.

Outcome: Admin receives up-to-date sales insights for business decision-making.

Chart Link: Chart 3



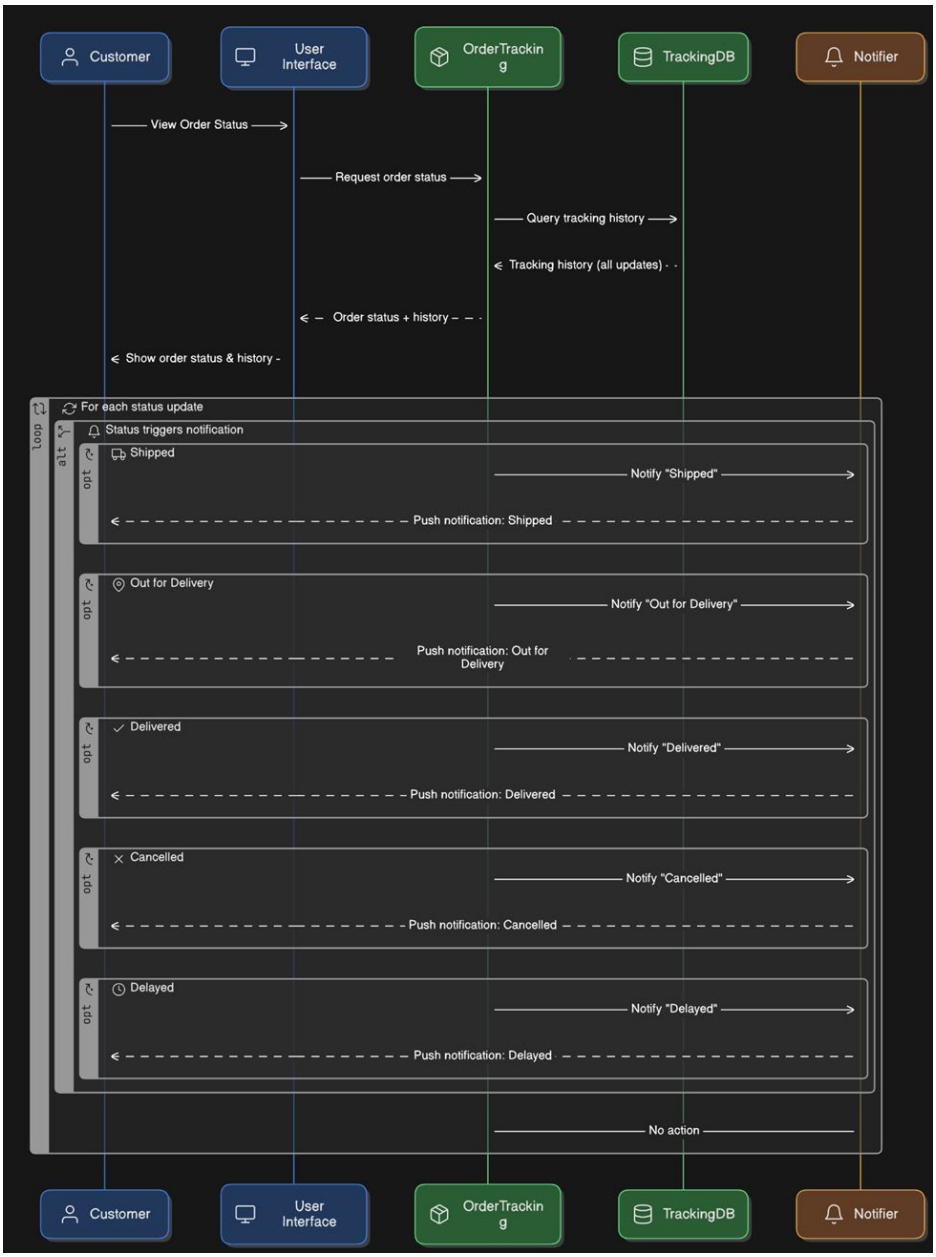
Use Case 4: Tracking an Order

Description: Customer checks the delivery progress of a previous order.
Steps:

1. Customer logs in and selects a past order.
2. **OrderTracking** retrieves the status.
3. Notifier sends update (e.g., "Shipped").

Outcome: Customer receives real-time order status and delivery updates.

Chart Link: Chart 4



8. References