Swinburne University of Technology

School of Science, Computing and Engineering Technologies

ASSIGNMENT AND PROJECT COVER SHEET

Subject Code: SWE30003 Unit Title: Software Architectures and Design

Assignment number and title: 2. Object Design Due date: 11 May 2025

Tutorial Day and Time: Wednesday, 12:30pm- 1:30pm Project Group: 2

Tutor: Thakshila Imiya Mohottige

To be completed as this is a group assignment
We declare that this is a group assignment and that no part of this submission has been copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part been written for us by another person

ID Number	Name	Signature
103904561	Jordan Nabulsi	Jordan
103805664	Ch'ng Sze Shuanne	Chloe
104855055	Lehan Lochana Alagedara	Lehan
104546595	Atiya Amiri	Atiya

Marker's comments:

Total Marks:

Extension certification:

This assignment has been given an extension and is now due on

Signature of Convener:

SWE30003 - Software Architectures and Design (Assignment 2)

Object Design

Tutor: Thakshila Imiya Mohottige

Submitted by:
Ch’ng Sze Shuanne
Jordan Nabulsi
Lehan Lochana Alagedara
Atiya Amiri

1. Executive Summary / Document Overview

- Summary of what the document includes

2. Introduction (jordan)

AWE Electronics is transitioning from a local, in-person retail store to a full-featured online platform. The core problem is to design a system that supports all the expected e-commerce functionality being product browsing, cart management, account handling, secure payments, order tracking, and sales reporting in a way that is reliable, scalable, and user-friendly.

This document presents an object-oriented design that addresses these challenges. The proposed system is structured around clearly defined classes with distinct responsibilities, low coupling, and high cohesion. It supports both guest and registered users and provides a clean separation of concerns between business logic and system services such as authentication and reporting. The design is intended to be maintainable and extensible, supporting the store's future growth.

3. Problem Analysis (jordan)

3.1 Design Goals

The main goal of this project is to design a robust object-oriented system for AWE Electronics' online store. The system should:

- Support guest and registered users in browsing products, managing carts, and placing orders.
- Enable admin users to manage product listings and view sales reports.
- Ensure secure handling of payments and personal data.
- Remain performant and usable under high traffic.
- Be designed in a modular and extensible way to accommodate future features.

The design aims to balance functionality with maintainability, keeping the architecture clean and scalable while still covering all the required user tasks.

3.2 Key Assumptions

- Customers may use the platform without registering (guest checkout is supported).
- Only admin users can access sales reports and make changes to the product catalog.
- Payments are handled by a third-party provider (e.g., PayPal), and sensitive financial data is not stored within the system.
- Orders cannot be edited or deleted once placed, to ensure auditability and trust.
- Customers will typically use the system via desktop or mobile web browsers, not through a native app.
- Each registered account can only have one shopping cart associated with it at one time.
- The scope of the basic sales statistics includes: items sold, age of customers, and which items are removed most frequently from shopping carts.
- Customers can only have one account associated with them.

These assumptions shape how responsibilities are assigned to classes and how different parts of the system interact.

3.3 Design Simplifications

To keep the design focused and manageable, some aspects of the system are intentionally simplified:

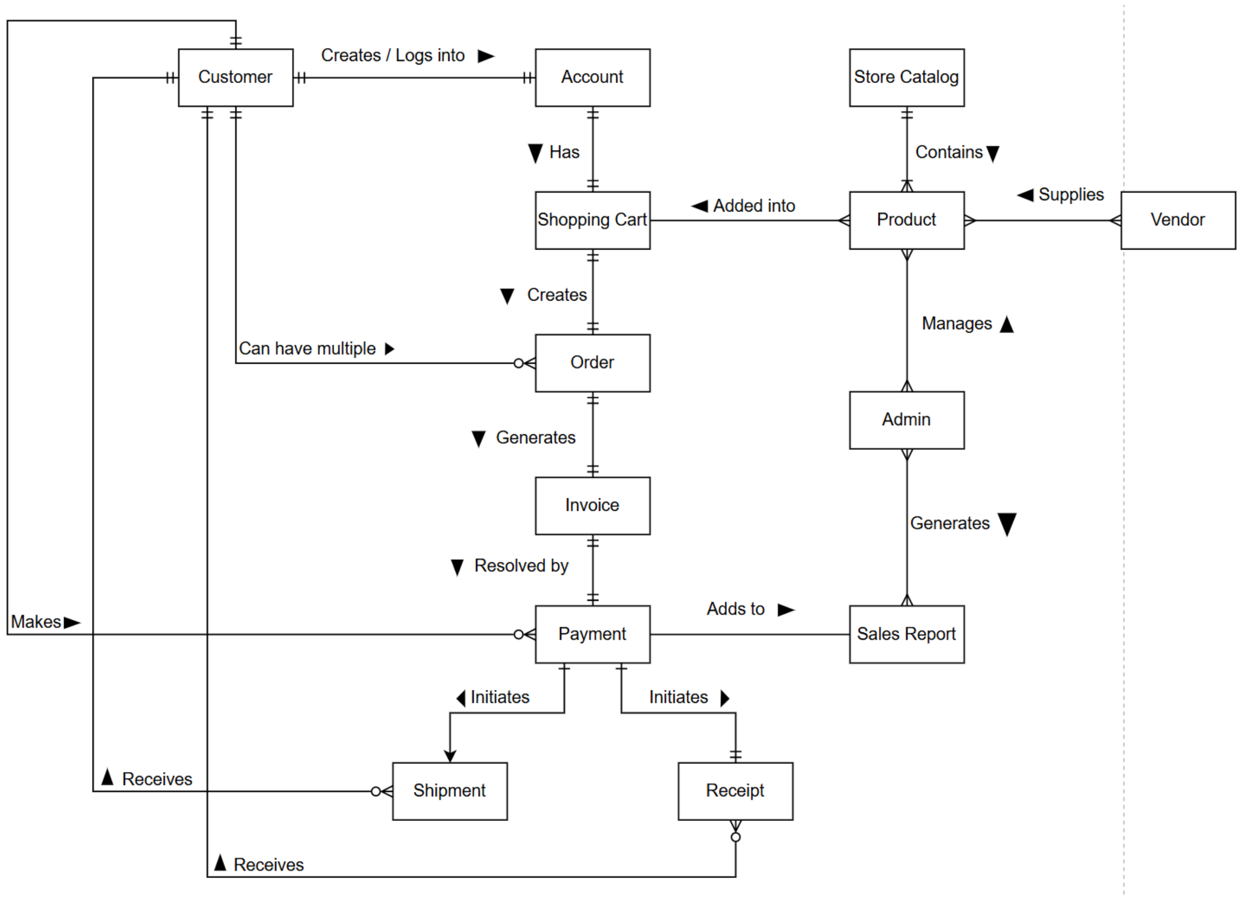
- Invoices and Receipts: Not modeled as separate entities — they are generated dynamically from the Order and Payment objects.
- Inventory Management: Backend stock management beyond price and availability is considered out of scope.
- Session Management: Authentication, login persistence, and user sessions are abstracted into a dedicated AuthService class.
- Guest Users: Treated as Customer objects without an associated Account, allowing for minimal duplication across classes.

These simplifications reduce complexity without compromising core functionality, making the system easier to implement and maintain.

Deviations from First Assignment

1 - Domain Model

After assessing the previous model, we have decided to include a new domain entity “Vendor” that has a many-to-many relationship with the “Products” entity as it supplies the needed products needed for the store to sell to its customers.



2 - Workflows

The following workflows were redefined from the first iteration to showcase a more integrated and complete action from initiation to completion.

Workflow 1:

Task:	**Modifying sales item (previously “store product management”)**
Actor:	Admin/owners
Purpose:	Modify sales items in store catalog
Sub-tasks:	**Example Solution:**
1. Log into admin portal	Dedicated admin login page.
2. View all items on sale in store catalog	Present all on sale products into paginated web pages.
3. Modify on sale item	Option to modify the price or whether an item is on sale or not. Toggle buttons?
4. Log out of admin portal	Saves changes made in the admin portal and exits.

Workflow 2:

Task:	**View sales report**
Actor:	Admin/Owners
Purpose:	Admin users want to see generated ‘report to check sales statistics.
Sub-tasks:	**Example Solution:**
1. Log into admin portal	
2. View total number of items sold	Present data in the form of a bar chart that shows the number of items sold over a period of time that is modifiable.
3. View the average age of customers	Present data in the form of a pie chart to show the customers’ age.
4. View most frequently removed items	Show the top 10 most removed items from

from shopping cart	the shopping cart in a list.
5. Log out of admin portal	

Workflow 3:

Task:	**Customer registration**
Actor:	Customer
Purpose:	Create an account for a new customer
Sub-tasks:	**Example Solution:**
1. Log into admin portal	
2. View total number of items sold	Present data in the form of a bar chart that shows the number of items sold over a period of time that is modifiable.
3. View the average age of customers	Present data in the form of a pie chart to show the customers' age.
4. View most frequently removed items from shopping cart	Show the top 10 most removed items from the shopping cart in a list.
5. Log out of admin portal	

Workflow 4:

Task:	**Customer product purchase**
Actor:	Customer
Purpose:	Customer makes a purchase in online store
Sub-tasks:	**Example Solution:**
1. Customer lands at online store page.	Present option to create account or continue as guest.
2. Create account	
3. Browse store catalogue	If product is not found, system suggests other similar products.
4. Add products into shopping cart **Problem:** Product out of stock.	Show estimated restock time. Suggest another similar product.

5. Place order **Problem:** Payment fails to go through.	Have multiple payment methods available. Order does not go through if payment fails. Customer receives a confirmation of order, receipt of payment and link to track order status.
6. Shipment of order	Warehouse receives notification to pack and send the order via a third-party service. Order status updates with its progress to customer's address.
7. Customer receives order.	

4. Candidate Classes (chloe) refer to lec 5 and 6

- Focus first on physical and conceptual objs
- Class selection rationale (one word for one concept *identify redundant objs*, wary of adjs, implicit objs needed that are identified from sentences, model interfaces to system - don't need to model user explicitly, model values of attributes not attributes themselves - part of something else instead)

● 4.1 Class List
(expand/refine on our current domain model)

- User
    - Customer
    - Admin

- Account
    - Contains customer obj

- Shopping Cart
-

Customer

Description: Represents any user interacting with the store — anonymous or logged-in.
Responsibility:

  Owns a ShoppingCart
  Can place an order (with or without an account)
  Can become an Account holder later

Collaborators:

  ShoppingCart
  Order
  Account (if they register)

Account

Description: Represents a user of the system, either a customer or an admin.
Responsibility:

  Knows their credentials, identity, and role (e.g., admin).
  Can submit orders, and manage their cart.
  Grants permissions for admin actions if applicable.

Collaborators:

  ShoppingCart
  Order
  ReportService

Product

Description: Represents a product listed in the store catalog.
Responsibility:

  Knows its name, description, price, and stock level.
  Knows if it is in stock.
  Can report its basic data to the catalog, cart, or order.

Collaborators:

  ShoppingCart
  Order

ShoppingCart

Description: Holds a temporary list of items a user wants to purchase.
Responsibility:

- Holds products and quantities selected by the user.
- Calculates the subtotal.
- Can be transformed into an Order.

Collaborators:

- Account
- Product
- Order

Order

Description: Represents a finalized purchase transaction.
Responsibility:

- Records a list of purchased products and quantities.
- Stores status, timestamps, shipping info.
- Can be linked to a payment and shipment.

Collaborators:

- Account
- Product
- Payment
- Shipment

Payment

Description: Represents a record of payment for an order.
Responsibility:

- Knows payment status and method (e.g., PayPal, card).
- Knows related order details.
- Logs transaction info from external payment provider.

Collaborators:

- Order
- External: Payment provider API (e.g., PayPal)

ReportService

Description: Generates summaries of sales and activity for admins.
Responsibility:

holds data from orders (e.g., total revenue, top products).
Filters by time period (day/week/month/etc).
Formats summaries for display.

Collaborators:

Order
Product
Account

AuthService

Description: Handles login, authentication, and session management.
Responsibility:

Validates credentials.
Issues tokens or manages sessions.
Verifies user permissions (e.g., admin access).

Collaborators:

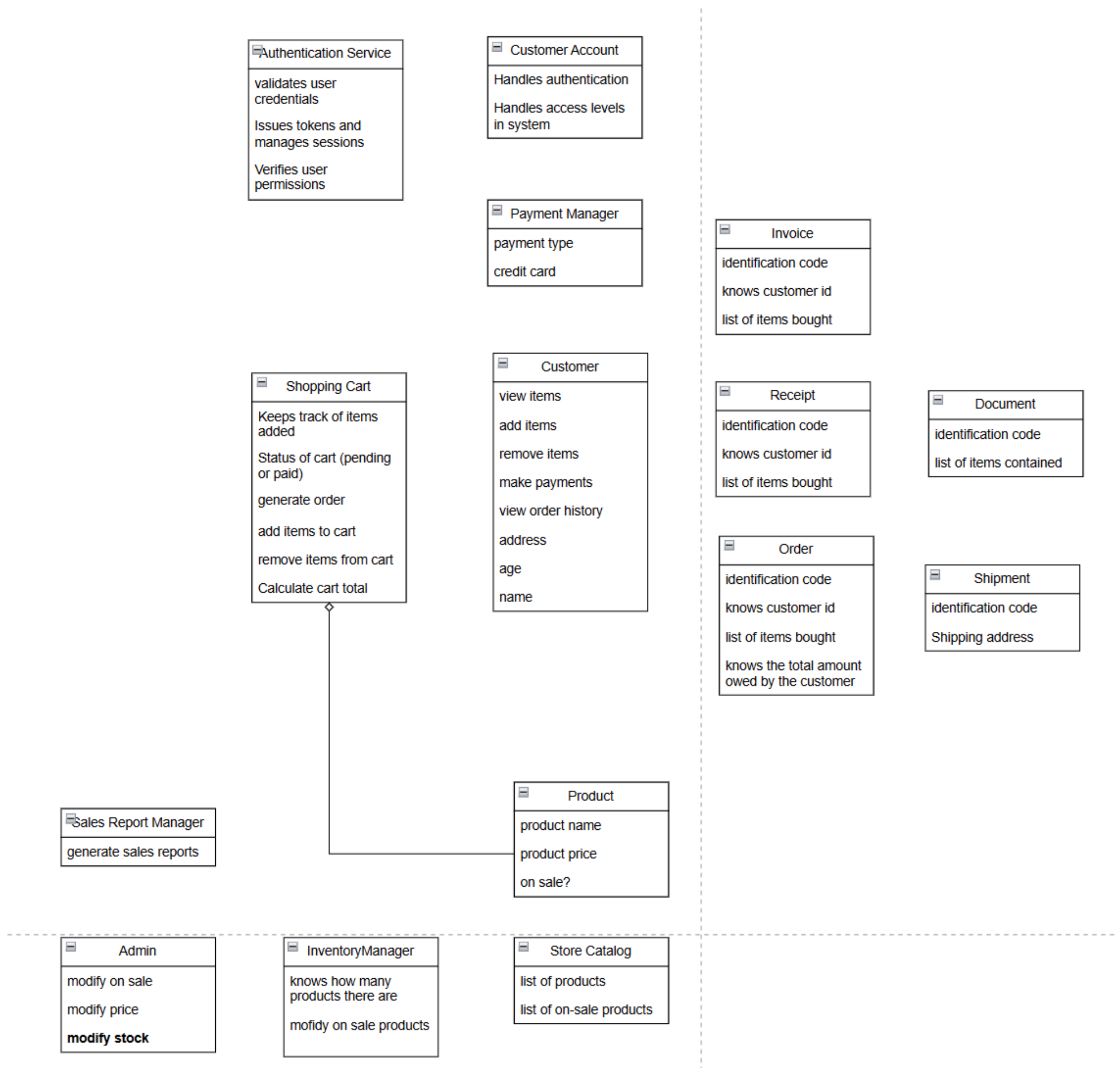
Account

- Justify any discard classes (if any)

  Invoice and receipt can just be generated from order and payment classes.

- 4.2 UML Class Diagram (image or draw.io export)

- 4.3 CRC Cards
  (Each class should have a sub-subsection with its responsibilities and collaborators)
  (refer to lecture 5 on creating CRC cards)



User	**-**
'User' contains the common attributes of the 'Customer' and 'Admin' entities - it cannot exist by itself and must be inherited by either classes mentioned previously. It represents users that will interact with the system.	
Responsibilities:	**Collaborating classes:**
- Knows name and age	NA

Customer	*Subclass of User*
‘Customer’ defines a unique entity that represents each customer that browses the online store. It contains the business logic that a customer is capable of within the system.	
Responsibilities:	**Collaborating classes:**
- Can view items in the store catalogue	StoreCatalog
- Can add items into the shopping cart	Product, ShoppingCart
- Can make payments	PaymentManager
- Knows what is in the shopping cart	ShoppingCart
- Can track shipment status	Shipment
- Knows the name, age, and address of the customer	NA

Admin	*Subclass of User*
‘Admin’ defines a unique entity that represents users with elevated access that can view and modify the system. It contains the business logic that an admin is capable of within the system.	
Responsibilities:	**Collaborating classes:**
- Can modify the store catalog items	StoreCatalog
- Can generate and view sales statistics	SalesReportManager

CustomerAccount	-
‘CustomerAccount’ represents the information of the customers that are using the system	
Responsibilities:	**Collaborating classes:**
- Knows the account id	NA

- Handles credential information (username and password)	Customer
- Handles authentication to the system	Customer, Admin

AuthenticationService	-
‘AuthenticationService’ handles the authentication of users within the system and manages sessions.	
Responsibilities:	**Collaborating classes:**
- Validates user credentials	Account, User
- Issues tokens/manages sessions	Account. User
- Verifies user permissions (e.g. admin access)	NA

Store Catalog	-
‘StoreCatalog’ contains all the products listed by the online store and is accessed by admin users to modify the products within.	
Responsibilities:	**Collaborating classes:**
- Organises and displays the collection of products	Product
- Allows customers to search for products	Customer, Products
- Knows the products availability	InventoryManager

Product	-
‘Product’ stores all the details and information regarding a type of product.	
Responsibilities:	**Collaborating classes:**
- Knows the products details (product id, name, description, price)	NA
- Details of product can be updated by	Admin

admin users	

ShoppingCart	-
‘ShoppingCart’ manages and stores the products added by the customer.	
Responsibilities:	**Collaborating classes:**
- Can store the products the customer wants to purchase	Customer, Product
- Allows products to be modified (added, removed)	Customer, Product
- Can calculate the total of the cart to prepare products for checkout and payment	Product

InventoryManager	-
‘InventoryManager’ manages the stock of the products.	
Responsibilities:	**Collaborating classes:**
- Knows the availability of each product	Product, StoreCatalog

Document	-
‘Document’ represents the business and transactional documents used within the system.	
Responsibilities:	**Collaborating classes:**
- Knows the document id	NA
- Knows the list of items bought	ShoppingCart
- Knows the customer id	Customer

Invoice	Subclass of Document
‘Invoice’ represents the document the store warehouse receives containing the order placed by a customer - which is only generated after a successful payment.	

Responsibilities:	**Collaborating classes:**
- Knows the document id	Document
- Knows the customer id	Customer
- Knows the list of items purchased	Customer, ShoppingCart

Receipt	Subclass of Document
‘Receipt’ is the confirmation of payment that is created and provided to the customer after a successful payment.	
Responsibilities:	**Collaborating classes:**
- Knows the document id	Document
- Knows the customer id	Customer
- Knows the list of items purchased	Customer, ShoppingCart

Order	Subclass of Document
‘Order’ is the document generated when a customer decides to check out their shopping carts. This document shows the customer how much they should pay for the products they have chosen.	
Responsibilities:	**Collaborating classes:**
- Knows the document id	Document
- Knows the customer id	Customer
- Knows the list of items purchased	Customer, ShoppingCart
- Calculates the cart total	Product

SalesReportManager	-
‘SalesReportManager’ generates, manages, and displays sales statistics to admin users.	
Responsibilities:	**Collaborating classes:**
- Knows and can display the system’s sales statistics	Invoice, Customer, Product

- Can modify time period of statistics	Admin

Shipment	-
‘Shipping’ manages the products that will be sent to the customer.	
Responsibilities:	**Collaborating classes:**
- Knows the shipping id	NA
- Knows the list of items its shipping	Invoice
- Knows the status of the shipment	NA

PaymentManager	-
‘PaymentManager’ processes the payment when a customer decides to checkout their shopping cart.	
Responsibilities:	**Collaborating classes:**
- Processes payment from customers	Customer

5. Design Patterns and Heuristics (Tia)

- List of applied design patterns (creational, behavioural, structural)
- Justifications for each

- Heuristics and principles followed (like low coupling, high cohesion)

In the development of the AWE Electronics online shopping system, we applied various design patterns to ensure that the architecture is modular, scalable, and easy to maintain. These patterns provide structured solutions to recurring problems, improving the system’s flexibility and robustness. By leveraging creational, structural, and behavioral patterns, we were able to effectively address challenges such as object creation, system interactions, and the management of complex behaviors. Additionally, we adhered to key software engineering heuristics and principles, ensuring the system is both efficient and user-friendly.

Applied Design Patterns

Creational Patterns

1. Singleton Pattern

- **Applied To:** AuthenticationService, SalesReportService, PaymentService
- **Justification:**
These services must maintain a single point of truth across the system to ensure consistency. For example, having one instance of AuthenticationService ensures that login and session validation are handled uniformly for customers, employees, and admins. Similarly, SalesReportService provides centralized reporting data to avoid discrepancies across dashboards.

2. Factory Method Pattern

- **Applied To:** AccountFactory, OrderFactory
- **Justification:**
Customers can either register or checkout as guests, so a factory is used to create the appropriate type of account without exposing object creation logic. OrderFactory handles different order types (e.g., regular, urgent), making the creation process flexible and modular.

Structural Patterns

3. Facade Pattern

- **Applied To:** ShoppingCartFacade, AdminDashboardFacade
- **Justification:**
Customers interact with shopping cart features like adding/removing products and viewing totals through a unified ShoppingCartFacade, simplifying the complexity of

inventory and pricing logic. Similarly, admins use a single interface (AdminDashboardFacade) to manage orders, products, and employee records without dealing with backend services directly.

4. Adapter Pattern

- **Applied To:** PaymentGatewayAdapter
- **Justification:**
  Enables integration with third-party payment systems such as PayPal or credit card processors. The adapter converts our internal PaymentService methods to match external APIs, ensuring compatibility without changing our core system.

Behavioral Patterns

5. Observer Pattern

- **Applied To:** InventoryService, OrderStatusService, ReportService
- **Justification:**
  Whenever a customer places an order, the inventory and reporting systems must be automatically notified. This decouples these systems and keeps them updated in real-time. Admins and employees can also get notified when order status or product stock changes.

6. Strategy Pattern

- **Applied To:** PaymentStrategy, DiscountStrategy
- **Justification:**
  Different payment methods (e.g., PayPal, credit card) and discount policies (e.g., 10% off, buy one get one) are interchangeable strategies. This allows the customer to choose a preferred payment method, while the system dynamically switches algorithms based on current promotions.

7. Command Pattern

- **Applied To:** AdminActionHandler (e.g., update product, add employee)

- **Justification:**
  Admin tasks are encapsulated into commands that can be executed, queued, or even undone. This allows centralized control and logging of critical operations like deleting products or managing employee access.

Heuristics and Principles Followed

1. Low Coupling and High Cohesion

- Each module is responsible for **a specific functionality** (e.g., authentication, cart, payments).
- Dependencies are minimized using interfaces and service layers, which makes the system more maintainable.

2. Separation of Concerns

- Customer, employee, and admin functionalities are clearly separated across components. For example, only admins access product management, while customers use the shopping interface.

3. DRY (Don't Repeat Yourself)

- Reusable components and services such as validation, database access, and session handling are abstracted to **reduce redundancy**.

4. Open/Closed Principle

- The system is **open to extension** (new payment methods, new admin tasks) but **closed to modification** of core logic, thanks to patterns like Strategy and Command.

5. Principle of Least Knowledge (Law of Demeter)

- Classes communicate only with **closely related objects**, avoiding deep chains of calls. This helps with system decoupling and testing.

6. Fail Fast and Robust Feedback

- Input validations and access checks are implemented at early stages (e.g., during login, product selection) to catch errors quickly and guide the user.

6. Bootstrap Process (jordan)

At startup, the system initializes core services and loads essential data required for operation. The bootstrap process includes:

1. Instantiating the main services:
    - ProductCatalogService
    - AuthService
    - PaymentService
    - OrderService
    - ReportService
2. Loading persistent data:
    - Products (from a database or file)
    - Registered accounts and admin users
    - Sales history for reporting
3. Preparing runtime entities:
    - Empty shopping carts are created when guests or users visit.
    - Admin dashboard is initialized with real-time sales stats if accessed.

TO-DO: sequence diagram

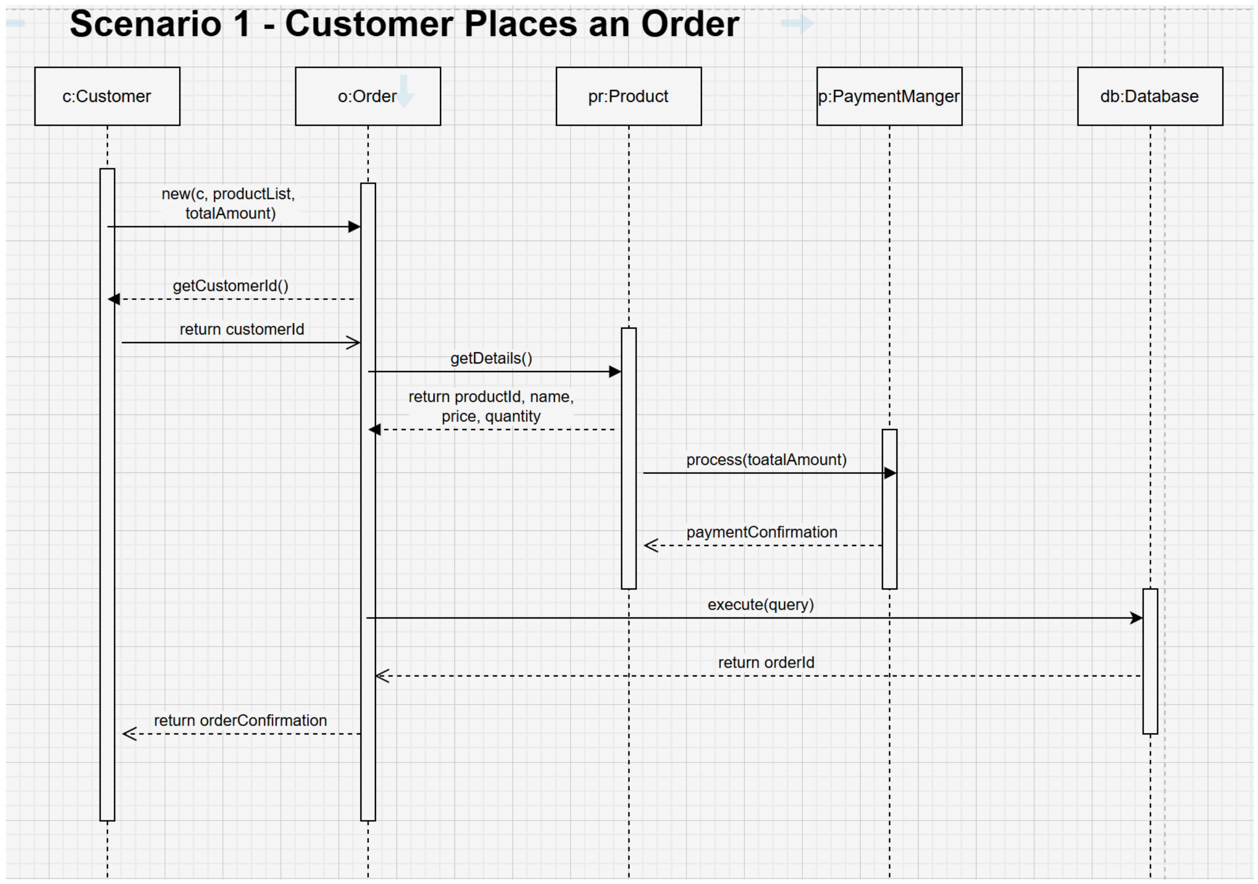
7. Use Case Scenarios - Verification (Chloe)

- Title + description of each scenario (x4)

- Describe how objects interact to fulfill the use case (include diagrams if useful)
- Use a UML sequence diagram (or similar)

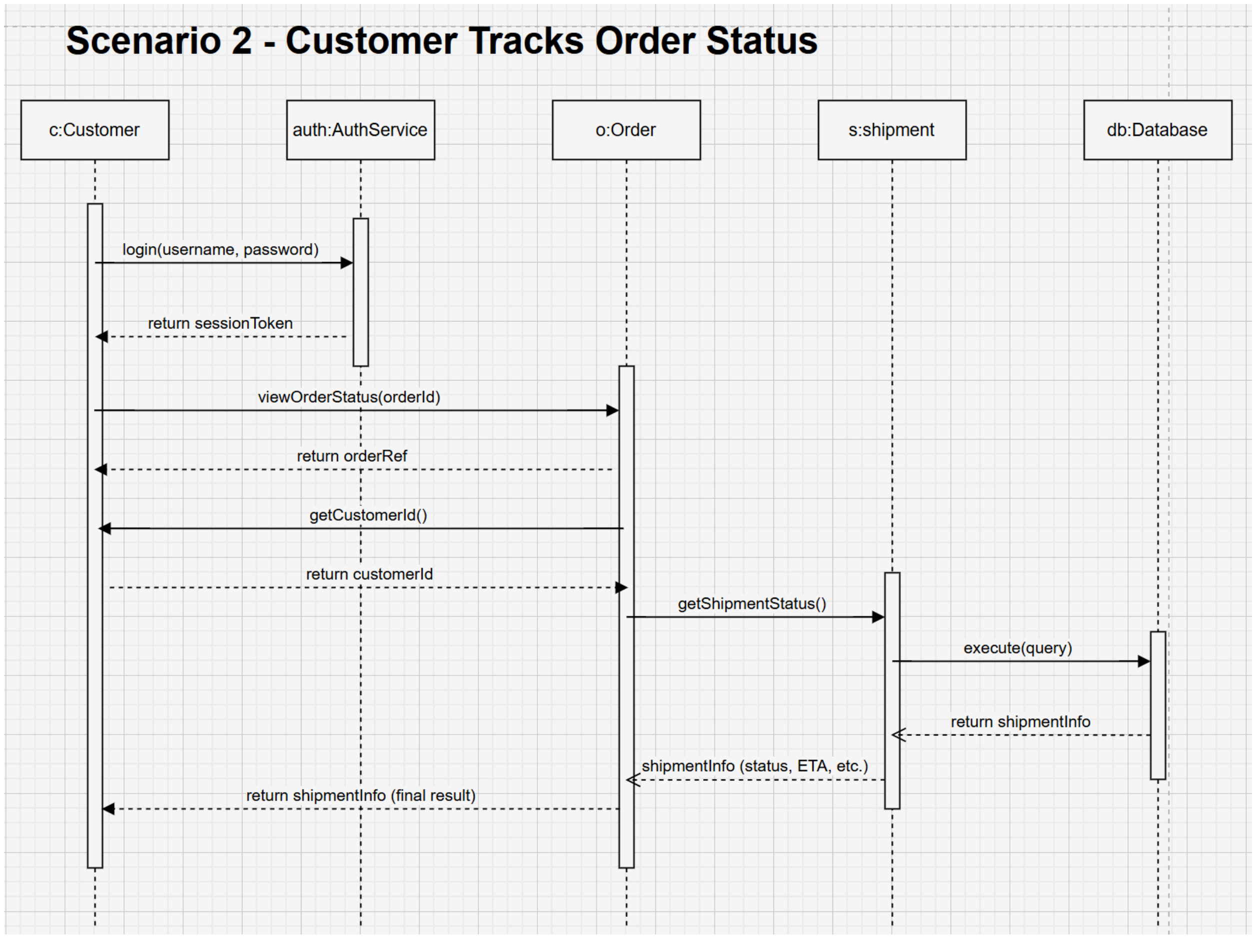
Scenario 1 - Customer makes a purchase

A customer (already registered) completes a purchase from the online store. The system creates a new Order, gathers necessary details from associated objects (Customer, Product, Payment), and records the transaction. This interaction creates a valid order and invoice, which is stored in the system.



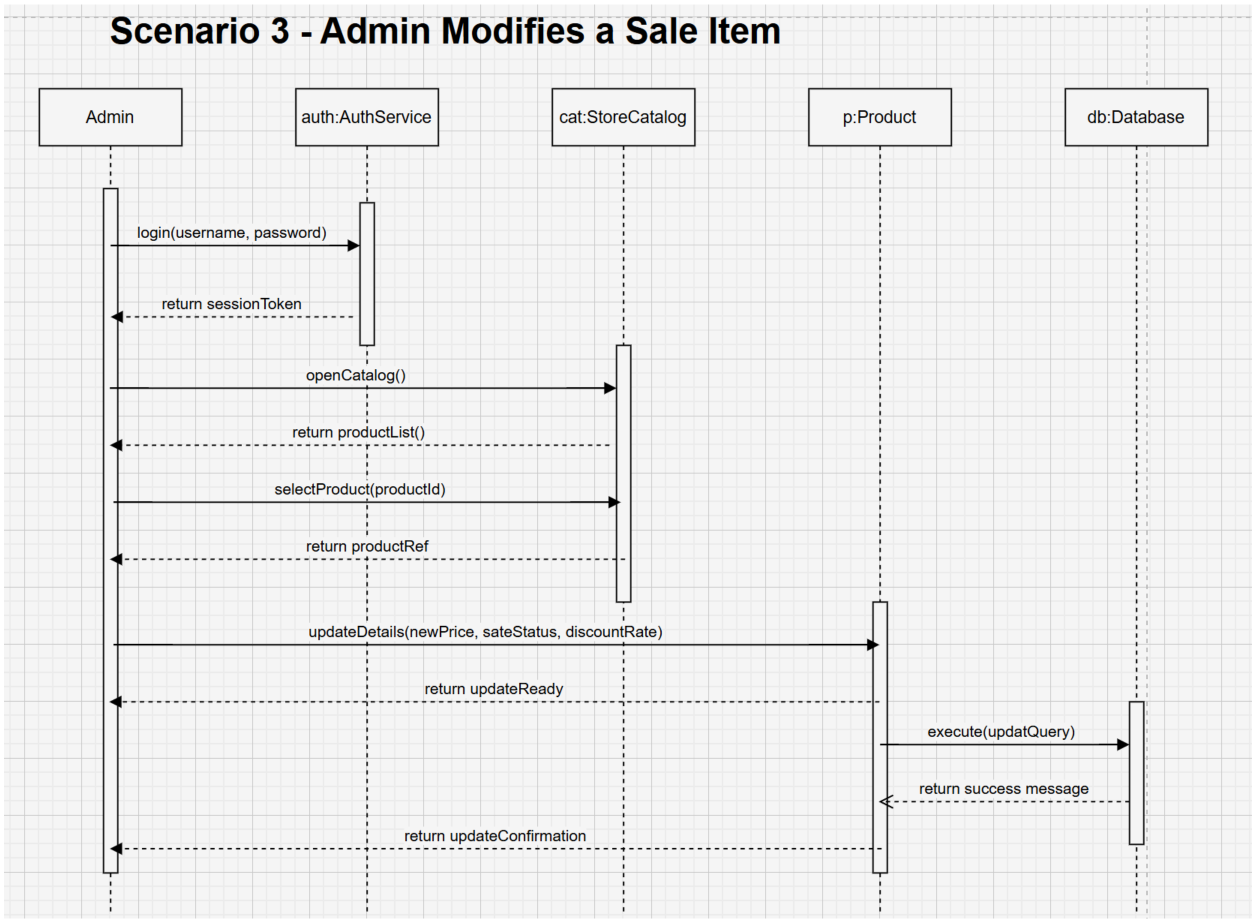
Scenario 2 - Customer Tracks Order Status

A registered customer logs into their account and checks the status of a previously placed order. The system retrieves the order using the customer ID and returns the associated shipment tracking details, such as delivery status and estimated arrival time.



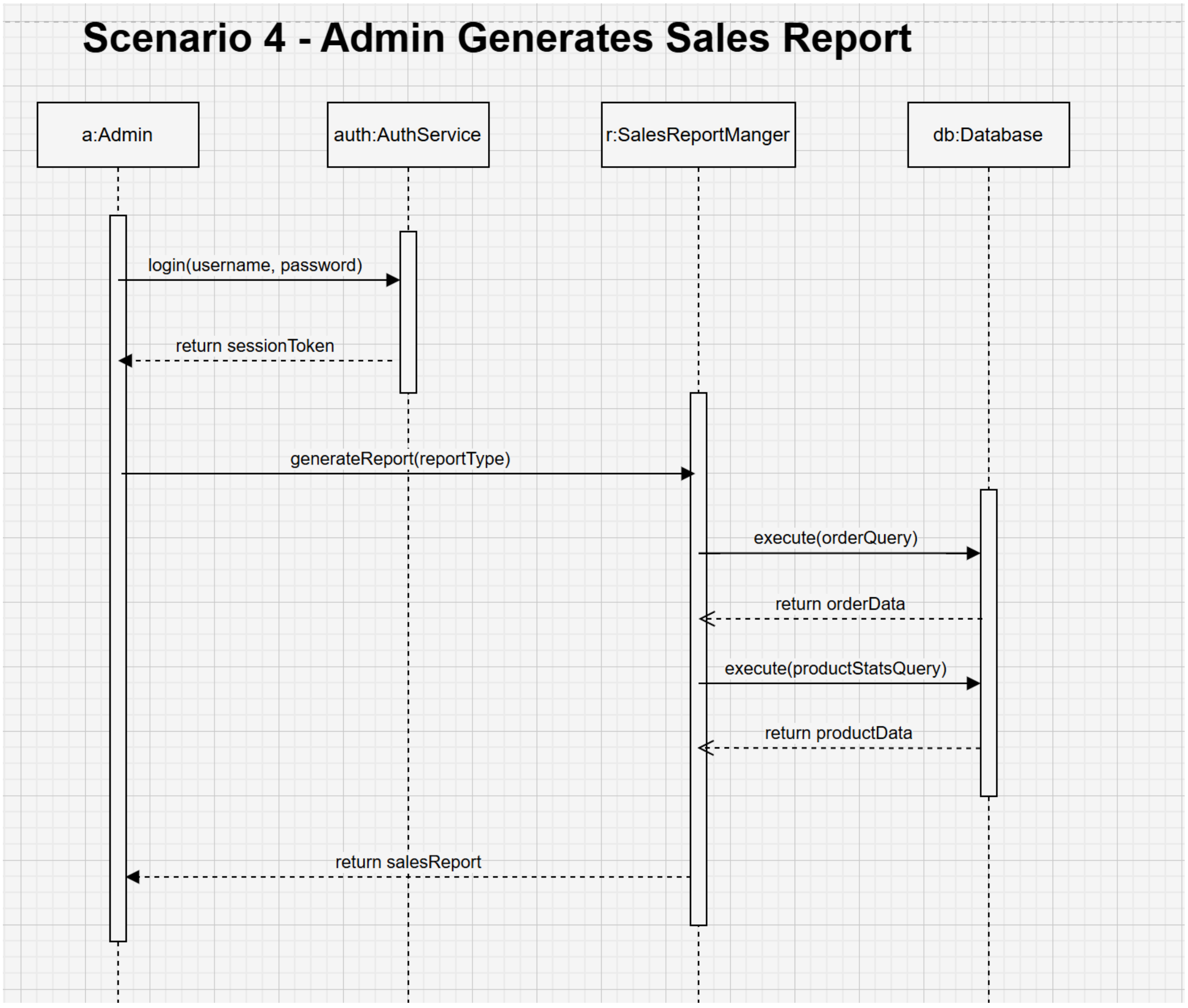
Scenario 3 - Admin Modifies a Sale Item

An admin logs into the system, selects a product from the inventory, updates its price and sale status (e.g., marks it as "On Sale" or changes the discount rate), and submits the updated details. The system validates the request, updates the product record in the database, and returns a confirmation.



Scenario 4 - Admin Generates Sales Report

An admin logs into the system and requests a sales report. The system gathers relevant data including completed orders and product performance by querying the database. Once all data is retrieved, the report is generated and returned to the admin.


Scenario 4 - Admin Generates Sales Report
a:Admin
auth:AuthService
r:SalesReportManger
db:Database
login(username, password)
return sessionToken
generateReport(reportType)
execute(orderQuery)
return orderData
execute(productStatsQuery)
return productData
return salesReport

References

Appendix A – Assignment 1 Submission

Table of Content

1 - Project Overview

AWE Electronic is a small electronics store located on Glenferrie road. Currently the store only has a physical location and sells products to local customers who visit in person. With the increase in online shopping the owners of AWE Electronics want to expand their business by creating an online store. This online store will allow customers from anywhere in Australia to browse products, create accounts, place orders and get their items delivered to their homes. The purpose of this is to build a complete online shopping platform that will help AWE Electronics grow its customers base, make more sales and keep up with other businesses that are already online.

1.2 - Project type

This project is an e-commerce system. It will be a custom solution that includes all the necessary tools for online shopping, such as account creation, shopping cart management, online payment, and delivery tracking. The project will be designed, specially for AWE Electronics based on their current needs and future goals.

1.3 - Goals

The main goal of this project is to help AWE Electronics move from a local store to an online business that can serve customers across Australia. More specifically, the goals are:

- To help AWE resolve their pain points of a physical store.
- Let customers create accounts so they can track their orders.
- Allow customers to browse the store catalogue online.
- Make it easy for customers to add items to their shopping bag and checkout.
- Allow secure online payment.
- Help the store keep track of all orders, payments and deliveries.
- Provide simple reports and statistics so the owners can see which products are selling the most.

1.4 - Objective

The main objective of this project is to build an online store that works smoothly and is easy for both customers and staff to use. The website should make online shopping simple and secure. Customers should be able to find what they need, place an order, and pay online. Staff and store owners should be able to manage the system, update products and see reports without needing technical knowledge.

1.5 - Project incentive

The reason for starting this project is that more and more people now prefer shopping online rather than going to stores in person. AWE Electronics wants to grow as a business and compete with other stores that already sell online. By building an online store, they can attract more customers, make more sales and stay up to date with today's shopping trends. The online store will also make the daily work of managing the business easier by automating many tasks. The system will also aid in business plans as it will give insights to the business' performance through the sales statistics.

1.6 - Context

Right now AWE Electronic is a physical store that only sells products to local customers. But with more people shopping online, the store is at risk of falling behind. To keep up and grow, AWE Electronic needs an online platform that lets them serve customers all across Australia. This new system will work alongside the physical store and help the business grow in both areas.

1.7 - Actors

The actors defined from the case study are:

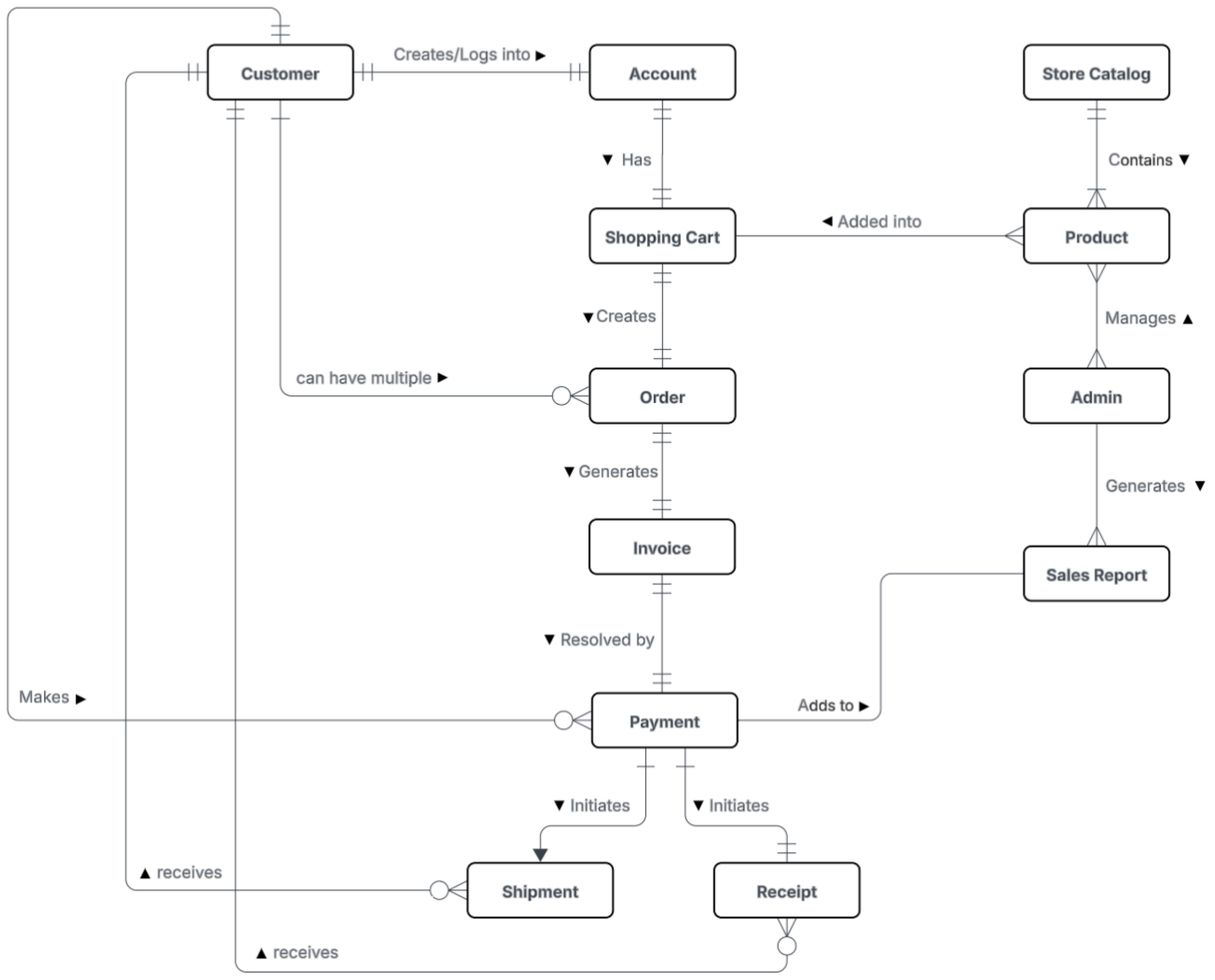
- Customer
- Admin/Owners

1.8 - Assumptions

- Customers will use the website regularly to create accounts and place orders.
- On average, about 5 new customer accounts are created per day.
- On average, 1 customer account is deleted every two weeks,
- Only store owners who manage the website will have admin-level access. This means they can add or remove products, view orders, update order status and see reports.
- Payments will be handled by a trusted third-party provider such as paypal.
- Tacit online store functionalities of admins being able to modify product prices and stock availability, updating store catalogue are included.

2 - Domain Model (Atiya)

The domain model outlines the key entities involved in the online shopping system and how they relate to one another. This is not a database schema, but a high- level representation of system elements.



The entities involved are:

- **Customer:** a shopper who signs up and purchases items from the online store.
- **Account:** Created by the customer to use the platform and shop.
- **Shopping cart:** Holds products the customer wants to buy before placing the order.
- **Product:** Items listed in the online catalog by AWE.
- **Store Catalog:** The full collection of available products.
- **Order:** A list of products a customer buys.
- **Invoice**: receipt generated after order.
- **Receipt:** Proof of purchase generated after a successful payment.
- **Payment:** A business exchange of money and goods between the customer and AWE.

- **Admin:** Person who manages products and reports.
- **Shipment:** Delivery details of the order.
- **Sales Report:** A report keeping track of how many products have sold.

3 - Functional Requirements and Task Descriptions

The functional requirements below define the functional requirements of the system accompanied by their task descriptions. The work area defines the working area of the requirements, which in this case is the online shopping platform.

Work Area: Online Shopping Platform	
- Service online guests 24/7 - Standard online shopping platform behaviour (browsing shop catalogue, add items into cart, handle payments) - Minimal disruptions - Displays items available in-store, online.	
Users:	IT novice
R1: The system shall support tasks 1.1 to 1.8.	

Task 1: Create Account

Task: Create Account	
Purpose:	Create a customer account to allow customers to register to the online store.
Trigger/Precondition:	A customer clicking on a “Sign Up”, “Register”, or “Create Account” button on the online platform. Customer does not have an account.
Frequency:	Average 5 account creations per day.
Critical:	Account created must be valid and avoid duplication.
Subtasks:	**Example solution:**
1. Obtain customer’s personal information	Customers provide their personal details (name, address, age), which is used to create an account.

2. Account creation using entered information	Create a customer's account using provided information.
Variants:	
2a. Invalid/wrong information given.	Suggest suitable changes and create an account when suitable.

Task 2: Delete Account

Task: Delete Account	
Purpose:	Delete a customer’s account from the system’s database.
Trigger/Precondition:	A customer clicking on a “Delete Account” button on the online platform.
Frequency:	Average 1 account deletion every 2 weeks.
Critical:	
Subtasks:	**Example solution:**
1. Delete account from system	Get customer’s confirmation to delete and proceed to delete account from online platform.
Variants:	
1a. Customer changes their mind. Accidentally clicked on the button.	Customer cancels the delete request, account remains active.

Task 3: Browse Store Catalogue

Task: Browse Store Catalogue	
Purpose:	To allow customers to explore the product catalogue and search for specific items.
Trigger/Precondition:	Customers access the online store website.
Frequency:	Multiple times per session, depending on the customer's needs.
Critical:	System performance must remain fast even during peak traffic or when browsing large inventories.
Subtasks:	**Example solution:**
1. Customer navigates product categories	The customer selects a product category from the menu to browse available items.
2. Customer searches for a product	The customer enters a keyword in the search bar and filters products by attributes (price, availability, brand).
3. View product details	The customer clicks on a product to view its full description, images, price, and availability.
Variants:	
2a. Product Not Found	System uses the match algorithm to show related products.

Task 4: Add Products to Cart

Task: Add Products to Cart	
Purpose:	To allow customers to collect products they wish to purchase.
Trigger/Precondition:	Customers have found a product they want to buy.
Frequency:	Multiple times per session, depending on the number of products the customer wants to purchase.
Critical:	Items in the cart must remain persistent across the session until checkout.
Subtasks:	**Example solution:**
1. Customer selects a product	The customer chooses a product and specifies quantity and any variations (e.g., color, size).
2. Customer adds product to cart	The customer clicks 'Add to Cart', and the system updates the cart.
3. Cart updates and displays confirmation	The system confirms the item was added and updates the cart total.
Variants:	
1a. Product has no stock left	System has shown an out-of-stock message, and the item is not added to the cart.

Task 5: Place an Order

Task: Place an Order	
Purpose:	To allow customers to complete a purchase and arrange product delivery.
Trigger/Precondition:	Customers have items in their cart and proceed to checkout.
Frequency:	Once per purchase transaction.
Critical:	Order processing must be secure and reliable to prevent transaction failures.
Subtasks:	**Example solution:**
1. Customer proceeds to checkout	The customer clicks 'Checkout' from the shopping cart page.
2. Customer enters shipping details	The customer provides a shipping address and selects delivery options.
3. Customer selects payment method	The customer chooses a payment method (credit card, PayPal, etc.) and enters payment details.
4. System verifies payment and confirms order	The system processes the payment, verifies details, and generates an order confirmation.
5. Customer receives order confirmation	The system sends a confirmation email with order details and estimated delivery time.
Variants:	
2a. Customer’s account has the address listed.	Automatically use the address associated with the account.
3a. Customer wants to use another address.	Take the new/different address.
4a. Payment method declined.	Request for another form of payment. Order is not successful.

Task 6: Track Order Status

Task: Track Order Status	
Purpose:	Allow customers to monitor the progress and status of their orders after a purchase.
Trigger/Precondition:	Customers have logged in their account and completed a purchase.
Frequency:	Once or twice per order.
Critical:	Accurate, real-time information must be given to keep the customers satisfied.
Subtasks:	**Example solution:**
1. Customer views orders.	System displays a list of past orders and current orders.
2. Customer selects an order.	The system will display tracking information: current status (e.g. “Processed”, “Shipped”, “Delivered”), estimated delivery date and the tracking number of the order.
Variants:	
2a. Order has been delayed or lost	System displays contact customer support option

Task 7: Modify Sale Items

Task: Modify Sale Items	
Purpose:	Allows admin users (store owners/staff) to add, update, or remove special offers on selected products.
Trigger/Precondition:	Admin logs into the admin account and selects the “Sale Items” section.
Frequency:	Used as required
Critical:	Must update prices seamlessly and immediately across the whole platform to prevent pricing errors.
Subtasks:	**Example solution:**
1. Admin selects a product to go on sale	System displays the product details and the current price
2. Admin enters the new sale price and duration	Admin sets the discounted price and sets the start and end dates for the discount.
3. System updates product price	On the customer view, the product with the updated price is tagged “On Sale”.
4. Remove product from sale	Admin selects the product and disables the discount price, then changes the product to the original pricing.
Variants:	
3a. Sale period ends automatically	System changes the product pricing to the original pricing after the set end date/time.

Task 8: Generate Sales Report

Task: Generate Sales Report	
Purpose:	Allow owners to view product sales statistics over various periods.
Trigger/Precondition:	Users with admin-level access clicks on the “View Business Statistics” page.
Frequency:	Daily, weekly, monthly, yearly, or as required
Critical:	
Subtasks:	**Example solution:**
1. Display number of products sold over a specified period**.**	System displays the total number of products sold over a period of time. Could display results in a graph or as a number.
Variants:	
1a. User wants specific details regarding one product.	System tailors results to display information regarding selected product.

4 - Workflows

The following workflows are defined to detail the high level tasks identified in the system.

Workflow 1:

Task:	**1. Customer product purchase**
Actor:	Customer
Purpose:	Customer wants to make a purchase from the store.
Sub-tasks:	**Example solution:**
1. Create account	
2. Browse store catalogue	If a product is not found, system displays, “no result found” and suggests similar or related products.
3. Add Product into shopping cart **Problem:** Product out of stock.	Show estimated restock time. Suggest another similar product.
4. Place order **Problem:** Payment fails to go through.	Have multiple payment methods available. Order does not go through if payment is not received successfully.
5. Track order status	Display order status (e.g.processing order, packing order, out for shipment, received).

Workflow 2:

Task:	**2. Store product management**
Actor:	Admin/owners
Purpose:	Admin/owner wants to modify on-sale items.
Sub-tasks:	**Example solution:**
1. Modify on-sale products	Customisable product details and on-sale duration.

Workflow 3:

Task:	**3. View sales report**
Actor:	Admin/owners
Purpose:	Store wants to generate report to check sales data.
Sub-tasks:	**Example solution:**
1. View statistics for all products.	Show overall sales statistics for all products in a graph/a number. Customisable time period.
2. View statistics for one particular product.	Show product specific details. Customisable time period.

5 - Quality Attributes

Considering the system's application domain and the previously detailed user tasks, four primary quality attributes were identified.

5.1 - Usability

High usability is crucial for persuading potential customers to adopt AWE's new online platform. A user-friendly and intuitive system fosters a more positive user experience, which directly contributes to higher adoption rates.

To achieve this quality attribute, the system shall:

- Strictly adhere to the UI/UX guidelines defined by Swinsoft Consulting.

5.2 - Security

With the global uptick in user data security, ensuring the security of the online platform should be one of our top priorities. The system will have to deal with online payments as well as handling and storing customer information.

To achieve this quality attribute, the system shall:

- Adhere to privacy policies stated in the Australian Privacy Act to handle user data responsibly.
- Encrypt transactions and user data
- Restrict data access according to user roles (only admin users can view business statistics and customers can only view their own invoices and receipts.)
- Comply with PCI DSS (Payment Card Industry Data Security Standard) in the system's payment gateway.

5.3 - Scalability

The system should be able to scale horizontally to handle demands of a larger number of users - given the client's goals of expanding their operations Australia-wide and sudden increase in usage during peak times or sales events.

To achieve this quality attribute, the system shall:

- Be able to scale horizontally to support a maximum of 1000 concurrent users.
- Follow modular design approach to create distinct components that can be scaled independently as the load increases.

5.4 - Performance

The system's ability to respond and retrieve the relevant information in a timely manner is crucial in fostering a positive user experience as well.

To achieve this quality attribute, the system shall:

- Retrieve and display catalog item pages within 3 seconds.
- Retrieve and display individual item information within 1 second.
- Be able to process a maximum 100 concurrent requests within a second.
- Render frontend platform displays within 200ms.

6 - Other Requirements

The online store will include additional product-level and design-level requirements beyond the core tasks outlined previously - using the goal-design scale.

6.1 - Product-Level Requirements

The system shall:

- Store and manage all user and transaction data securely, with proper input validation in place.

- Comply with Australian privacy legislation, with user data only used as outlined in the privacy policy.

- Support reliable and secure online transactions and generate printable receipts and invoices.

- Provide sales tracking and generate relevant reports, such as daily, weekly, monthly, and YTD summaries.

- Load product and catalogue pages within 3 seconds and handle at least 100 concurrent users during peak traffic.

6.2 - Design-Level Requirements

The online store shall:

- Adhere to WCAG 2.1 accessibility standards to support users with disabilities.

- Be compatible with screen readers, support keyboard navigation, and include alternative text for all images.

- Follow modern responsive design practices as outlined in Swinsoft Consulting's UI/UX guidelines to provide a consistent experience across devices.

- Be built to support future internationalization (language and regional formats).

- Display the store's branding consistently across all pages and include legally required pages: Privacy Policy, Terms of Use, and Terms of Trade.

7 - Validation of Requirements (Lehan)

In order to confirm that the AWE Electronic Online Store requirements are valid, accurate, complete and aligned with the user and business needs, the team conducted multiple validation steps.

7.1 - CRUD Check

We analyzed every identified task using CRUD (Create, Read, Update, Delete) operations to check if the main data entities and their relations are captured and managed accurately. This step allows us to confirm that all the necessary interactions with the system are properly managed for both user and admin.

Task	**Entities**						
	Customer	**Product**	**Cart**	**Order**	**Invoice**	**Payment**	**Sale Data**
Create account	CR						
Delete account	D						
Browse store catalogue		R					
Add products to cart		R	CRU				
Place an order	R		U	CR	CR	C	U
Track order status	R			R			
Modify sale items		RU					
Generate sales report		R					CR
Missing?	**U**	**C D**	**D**	**UD**	**UD**	**RUD**	**D**

The CRUD check above is part of the validation step using the completeness checks

Explanation of the Missing CRUD Operations

The last row in the CRUD check table highlights the operations that are not included for some entities. These omissions are intentional and this reflects on how the system is supposed to work in real life.

- Customers (Update): We decided not to include account updates like editing names or preferences, since the main functionalities only require the users to be able to create and delete their accounts.

- Product (Create, Delete): Adding or removing products isn't something customers or even regular staff would do through the online store. These tasks are assumed to be done through a separate internal inventory system.

- Cart (Delete): Carts do not need a delete function as they clear themselves either when an order is placed or if the session times out.

- Order (Update, Delete): Orders are locked in once placed. This avoids confusion, prevents tampering, and keeps the order history reliable for both store and customer.

- Invoice (Update, Delete): Since invoices are tied to payments, they need to stay unchanged to meet legal and financial requirements.

- Payment (Read, Update, Delete): Payment processing is handled externally by a third party provider (e.g. PayPal), so the system does not need to store or display detailed payment data.Once a payment is made, the system just records that it was successful. There is no need for customers or admins to read, update, or delete payment records within the platform itself.

- Sales Data (Delete): Sales data is important for generating reports and understanding business performance, so there is no need to delete them. Keeping this data permanent helps the store track trends over time and maintain accurate records.

7.2 - Scenarios

In order to make sure that all the system functionalities covered, we carried out a scenario based validation. This was done by identifying realistic situations that customers or administrators would run into and checking whether every single scenario is addressed by at least one task in the functional requirements.

Scenario	Covered by task(s)
A customer visits the website and wants to create and account.	Task 1: Create Account
A customer has decided they no longer want to use the service. So they want to delete their account.	Task 2: Delete Account
A customer wants to check out the different available products.	Task 3: Browse Store Catalogue
A customer adds the desired products to the cart and prepares to make the payment.	Task 4: Add Product to Shopping Cart
The customer makes the payment for items selected and receives a payment confirmation for the transaction.	Task 5: Place Order
The customer wants to check the delivery status of an order they made.	Task 6: Track Order Status
Admin wants to update the price of a product for a limited time promotion.	Task 7: Modify on Sale Items
The store manager wants to obtain a report on the total sales made this month.	Task 8: Generate Sales Report

7.3 - Traceability to Business Goals

Traceability makes sure that all the functional and non-functional requirements of the system traces back to the business goals and stakeholder needs which are stated in the case study. This verification step guarantees that the system's objectives are aligned with the primary goals, while avoiding unnecessary features. We established the traceability by checking whether the functional requirements support one or more of the business goals.

Business Goal	Supporting Tasks
Allow Customers to create Accounts and track orders	Task 1: Create Account Task 6: Track Order Status
Allow browsing of the store catalogue	Task 3: Browse Store Catalogue
Making it easy for customers to add items to their shopping cart and checkout	Task 4: Add Product to Cart Task 5: Place Order
Facilitate secure online payments and provide digital receipts	Task 5: Place Order

Allowing store managers to manage product pricing and sales	Task 7: Modify On Sale Items
Provide administrators with reports and sales data for decision making	Task 8: Generate Sales Report
Reach customers across Australia through and online platform	Enabled by Tasks 1 to 6
Streamline admin-levels operations of the business	Task 7: Modify On Sale Items Task 8: Generate Sales Report

8 - Verifiability

Verifiability guarantees that the stated functional and non-functional requirements can be objectively tested and measured during implementation and evaluation. To ensure that the system can be properly verified, we have taken the below mentioned steps.

8.1 - Functional Requirements Verifiability

All the functional requirements are written with:

- **Defined triggers and outcomes**, which makes it easy for validation.
- **Subtasks** which can be directly tested through UI interaction and testing.
- **Variants** which allow for a broader coverage of real world scenarios.

To ensure that the functional requirements are verifiable, the system's functionalities will be verified against all the functional requirements outlined previously throughout the following development process including the design, program, and testing phases.

In order to verify the functional requirements defined, each one of them including their variants shall be simulated. Users will carry out the tasks and its variants using our system to verify that the requirements of the system can be met.

8.2 - Quality Attribute Verifiability

Quality Attribute	**Verification Method**
Usability	- Conduct usability testing sessions with no-technical users to ensure that the system is intuitive and easy to use.

	- Ensure that the User interface follows the UI/UX guidelines
Security	- Conduct tests on the Role-based access control - Use penetration testing to check for vulnerabilities
Performance	- Conduct load tests to check whether: ● Page load time (less than 3s) ● Item detail display time (less than 1s) ● UI rendering speed (less than 200ms)
Scalability	- Create a test script to simulate 1000 users and monitor the system behaviour - Test the database scaling under peak conditions

9 - Possible Solutions

Solution 1: Website-based Approach

A website-based solution is a highly effective and widely adopted strategy and provides a comprehensive platform that can fully satisfy all of the requirements stated.

The website architecture can be logically separated into distinct pages, including a product catalog, individual product detail pages, store information, a customer's shopping cart, a secure payment gateway and an administrative interface to view simple sales statistics.

Upon successful implementation, the database can be populated with product information to be displayed and sold on the website. Admins can proceed with modifying the sale items, and start viewing sales statistics on the products.

Customers will be able to register accounts, browse the product catalog, and populate items to their shopping carts. Once ready, they can proceed through a secure checkout process to complete their purchase. Following a successful payment, an invoice and receipt is generated, and the order is processed for shipment. The order status will then be dynamically updated, providing real-time tracking for the customer.

A web-based approach directly supports the client's goals of potentially expanding their store's reach through benefits like search engine discoverability and accessibility from any web browsers (Safari, Chrome, Bing, etc), without requiring any additional software installations on the user's local device.

Solution 2: E-commerce Platform

Alternatively, leveraging an e-commerce platform like Shopify offers a streamlined approach, eliminating the need for custom website development, server setup, and database management. This solution effectively addresses all project requirements. Shopify's integrated store builder enables comprehensive website customization, encompassing product display, shopping cart functionality, and checkout processes. Furthermore, it provides access to robust business analytics, including site traffic, user behavior, and customer feedback, facilitating future growth and optimization.

This approach achieves a similar result to Solution 1, while significantly reducing the manual setup and maintenance efforts required in building a website from scratch.

Solution 3: Mobile Application-based Approach

A mobile application-based approach offers a user-centric experience and highly personalised approach to building the online store that can satisfy all of the requirements.

This solution would involve developing native applications for both iOS and Android platforms, allowing customers to seamlessly browse the store directly from their mobile devices. Unlike a traditional website, a mobile app offers advantages like push notifications on sale items and potential biometric authentication integration for more secure transactions.

Similar to Solution 1, this implementation would involve creating a backend API to manage product data, user accounts, and order processing. However, the frontend differs and is optimised for mobile interactions, offering a faster performance compared to mobile websites due to resources stored locally.

The app will support integrated mobile payment gateways like Apple Pay and Google Pay, for a streamlined checkout process that many users would be familiar with.

The overall visual structure will maintain a logical page separation, mirroring the previous solutions, to ensure intuitive navigation and information access based on user access levels.

Solution 4: Progressive Web App (PWA)

A Progressive Web App (PWA) is a modern solution that combines the best features of websites and mobile apps. Unlike traditional apps, a PWA does not require installation from an app store. Instead, it can be accessed directly through the web browser, and if preferred, it can even be added to the phone's home screen so it works like an app. PWAs are designed to load quickly, even if your internet connection is slow or unstable, making them very reliable for users in different environments.

One of the main advantages of a PWA is that it offers a smooth, app-like experience without needing to go through the process of downloading an app from an app store. When using a PWA, important notifications can be generated similar to mobile apps, which is great for sending updates about sales or new product arrivals.

Another big benefit is that PWAs are very secure. They use HTTPS to protect your data and make sure online transactions are safe. Because of this, PWAs are ideal for businesses that handle sensitive customer information and online payments.

From a business perspective, PWAs save a lot of time and money. Instead of creating separate versions of an app for different devices like iOS and Android, developers only need to build and update one version of the PWA. This makes maintenance easier and reduces the cost of development. PWAs are also responsive, meaning they automatically adjust their layout to fit the screen size of the device, whether it's a phone, tablet, or desktop computer.

Overall, a Progressive Web App is a great option for businesses that want to provide users with an easy-to-use, fast, and reliable online shopping experience, while keeping development costs lower and maintaining security.