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: 17th March 2024

Tutorial Day and Time: Tuesday 9:00AM Project Group: 9

Tutor: Ms. Mandeep Dhindsa

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
103803891	Tran Quoc Dung	Dung
103843282	Pham Hoang Duong	Duong
103806531	Nguyen Thai Son	NTSon
103792724	Nguyen Manh Duc	ManhDuc

Marker's comments:

Total Mark:_____

Extension certification:

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

Signature of Convener:_____

Contents

Executive Summary

The objective of this initial object-oriented design for the Online Healthy Foods Store, tailored for the All Your Healthy Foods food retailer, is to provide a structured blueprint for the system's development. The design process begins with the identification of key classes and their interrelationships, depicted through a basic UML class diagram following the Responsibility Driven Design approach.
The class definition is followed by an illustration of the initialization process that shows the classes that create instances of others and the order in which initialization occurs. To further ensure that the design is effective in managing user inputs, validation, and application to business processes, four common interaction patterns or scenarios are described. By following defined design concepts and patterns, the Online Healthy Foods Store complies with the functional needs of the All Your Healthy Foods food retailer. The structured approach seeks to bring clarity and consistency to the design.

1. Problem Analysis

1.1 Goals

- Expanding the food store from local to Australia-wide by establishing an online presence
- Developing an online store with all features of the physical stores, moreover, allowing customers to browse the store catalogue, ordering healthy foods online, and receive it by delivery services.
- The online store enables online users to create and manage accounts.
- Customers can adjust their Cart.
- Supporting the creation of invoices and receipts, handle payments and managing the packaging, shipments of goods.
- Provide basic statistics about the goods sold over various period.

1.2 Assumptions

A1 - Our target audience will value the convenience of having healthy food delivered directly to their doorstep.
A2 - Customer appreciate a diverse selection of healthy food products across different categories.
A3 - Some customer will have a specific dietary need.

A4 - Because we are shipping Australia-wide, product will likely need to have a long shelf life and require a minimal refrigeration.
A5 - Shipping perishable or temperature-controlled items can be expensive.
A6 - All customer will have unique name, address and phone number.
A7 - Store only sell fresh and quality food.
A8 - Packaging needs to be secure and ensure product integrity during transit across varying distances and climates.
A9 - The system resides in a secure environment so that payment information does not have to be encrypted.
A10 - After a customer pay for a product, they will receive a phone call to confirm about their address.

1.3 Simplifications

The All Your Healthy Foods online store can be simplified by narrowing the project scope, or implementing features in phases instead of simultaneously. Here are some possible simplifications:

- Start with a limited number of products in the online store, so that Product Catalogue can be reduced.
- Customers could purchase and check out as guests, instead of creating accounts for more other preferences in database.
- Regardless of adding and removing products, the Shopping cart can simply add products and check out.
- Basic manual sales reports can be written instead of providing detailed sales analytics based on the database.

1.4 Discarded Class List

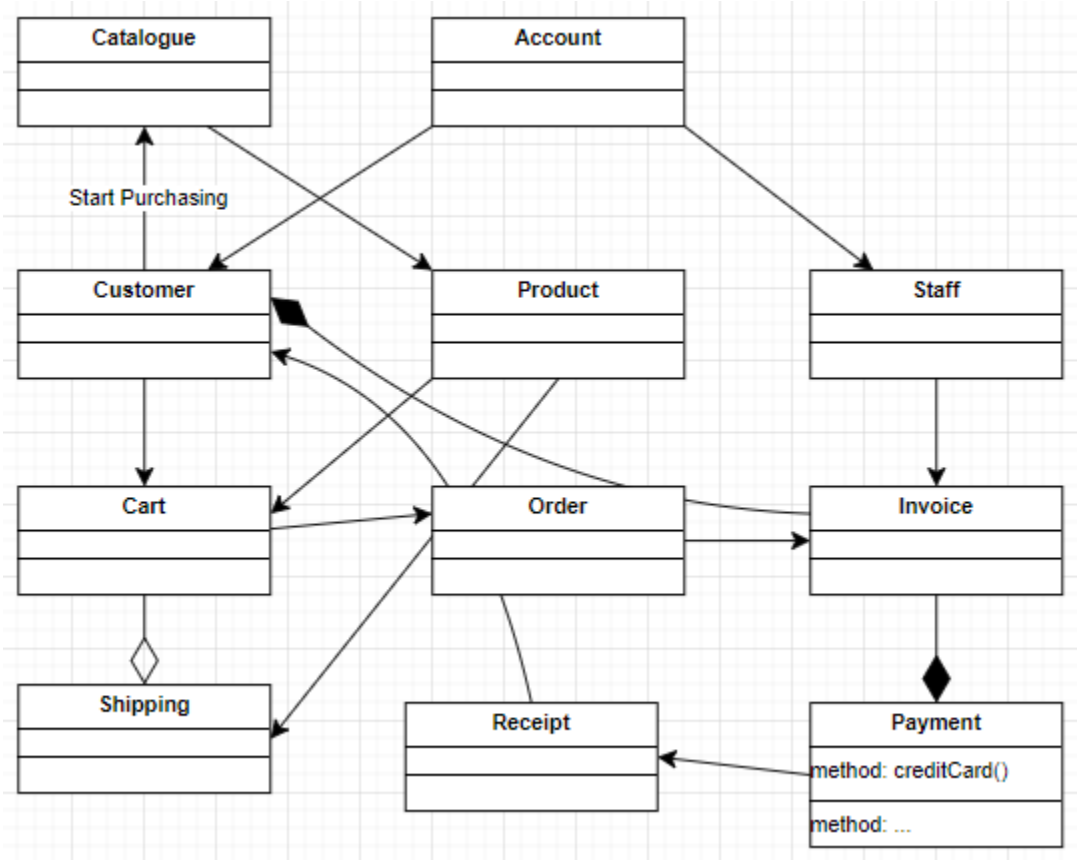
- Cart: The Cart Class was supposed to appear along with the other pages, until we realized this feature can also be programmed in back-end language in the Order page as well. As the Cart feature can also be observed in our reports (adjusting products), we believe this Class List is not necessary in our design model.

2. Candidate Classes

2a) Candidate Class List

1. Customer
2. Staffs
3. Product
4. Order
5. Invoice
6. Receipt
7. Payment
8. Shipping
9. Database
10. Account
11. Menu

2b) UML Diagram



Basically, this UML illustrates how the classes in our webpage are connected. To elaborate, there are two types of relation: *Directional* and

Database is the back-end components of the system, while the foregoing classes appear as the interfaces of the database.

From the customer's perspective, the Customer creates an Order, which includes Products that they can start choosing from the Catalogue. After that, an Invoice created by the Staffs is appropriately displayed for them. Then the Customer opts for their payment method (in Payment Class) to complete their purchase.

After this step, a receipt is sent to the Customer, simultaneously the Order and Products are arranged to be received by the Shipping services. Finally, the Shipping services deliver the products, according to the Order, to the Customer, complete an online purchase.

2c) CRC Cards

Class Name: Customer **Super Class: -**	
Customer class has the same responsibility of the account of a real Customer	
Responsibilities	**Collaborators**
• **Know how much money to pay for purchase**	**Invoice**
• **Know if product is paid or not**	**Invoice**

Class Name: Product **Super Class: -**	
Product holds information about the product	
Responsibilities	**Collaborators**
• **Knows product name, id, cost and general details**	**NA**

Class Name: Catalogue **Super Class: -**
Catalogue shows a list of product types available for customers to choose

Responsibilities	Collaborators
• **Contains list of product**	**NA**

Class Name: Staff **Super Class: -**	
Staffs account manages majority of operations made by the system	
Responsibilities	**Collaborators**
• **Can update/delete product and their information**	**NA**
• **Knows sale reports**	**NA**

Class Name: Order **Super Class: -**	
Order holds current customer's order for items in the store	
Responsibilities	**Collaborators**
• **Holds a list of purchased product**	**NA**

Class Name: Invoice **Super Class: -**	
Invoice contains purchases from customer that hasn't been paid yet	
Responsibilities	**Collaborators**
• **Knows what customer has invoice**	**Customer**
• **Knows PaymentID**	**Payment**

Class Name: Receipt **Super Class: -**	
Receipt process payment details after a purchase has been successfully paid for	
Responsibilities	**Collaborators**
• **Contains payments details (who pays for what and how much)**	**NA**

Class Name: Payment **Super Class: -**	
Payment processes invoice of customers when it is paid	
Responsibilities	**Collaborators**
• **Knows how much money purchase costs**	**Invoice**

• **Can check if a payment is success or not**	**NA**
• **Can validate payment method**	**NA**

Class Name: Cart **Super Class: -**	
Cart holds product that may or may not be shipped	
Responsibilities	**Collaborators**
• **Knows product info**	**NA**
• **Knows which product will be processed for shipping**	**Shipping**

Class Name: Shipping **Super Class: -**	
Shipping contains product and order of customers	
Responsibilities	**Collaborators**
• **Contains shipping fees**	**NA**
• **Contains customers address**	**NA**

• **Know list of product purchase by customer**	**Cart**

Class Name: Database **Super Class: -**	
Database holds record of purchase and all available information of the store's products	
Responsibilities	**Collaborators**
• **Records receipt of past purchases**	**NA**
• **Can update product information**	**NA**
• **Can remove products**	**NA**
• **Analyze sales statistics, trending products**	**NA**

Class Name: Account **Super Class: -**	
Accounts represent the user account on the website	
Responsibilities	**Collaborators**
• **Knows account details like name, address, etc**	**NA**

3. Design Heuristics

H1 Place common data and behaviors as high as possible in the inheritance hierarchy.

H2 Encapsulation data within classes.

H3 A class should not manage an excessive number of objects.

H4 Utilize class variables and methods instead of global variables and functions to store class-specific information.

H5 A derived class should not override inherited methods with empty implementations.

H6 All base classes should be abstracted classes.

H7 Avoiding creating 'God Classes' that control too many aspects of the program.

H8 Foster reusability within the codebase

H9 Do not create a class for an action; reserve classes for objects or entities.

Heuristics	Task	Class
H1	Create and manage customer accounts	Account, Menu
H2	All tasks	All classes
H3	• Browse the store catalogue. • Manage goods packaging and shipment. • Create invoices and receipts for customers. • Record statistics of goods sold based on receipts over various periods.	Catalogue, Shipping, Staff, Database
H4	All tasks	All classes
H5	• Create and manage customer accounts. • Create invoices and receipts for customers.	Customer, Staff
H6	Create and manage customer accounts	Account, Menu
H7	All tasks	All classes
H8	• Create invoices and receipts for customers. • Manage goods packaging and shipment.	Invoice, Product, Order
H9	All tasks	All classes

4. Design Patterns

4.1. Creational Patterns

4.1.1. Factory Method Pattern:

Purpose: This pattern provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.

Application: In the context of the "All Your Healthy Foods" online store, the Factory Method pattern can be easily utilized for diving the 2 types of different accounts, which are Customer accounts and Staff accounts. With this pattern, the user account can easily alter the types of objects created by the class Account.

Moreover, it can be applied to the creation of different types of products based on their categories. For example, it can be used to create different types of healthy food products such as fruits, vegetables, dairy, etc., each with its own factory method responsible for creating instances of products belonging to that category. By implementing this pattern, it allows for future expansion and addition of new product categories, enhancing the flexibility and scalability of the online store system.

4.1.2. Builder Pattern:

Purpose: Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.

Application: Used in the online store to construct complex objects like Product or Invoice, since they relate to numerous classes. For instance, the ProductBuilder class can handle the product types to be inserted in correct catalogue, also providing flexibility in being added or removed from Cart. Moreover, while buying several numbers of products at a time, discount might also be applied to customers, therefore applying this pattern is an appropriate solution.

4.2. Behavioral Patterns

4.2.1 Observer Pattern:

Purpose: Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Application: Implemented in the online store to notify customers when their orders are processed, shipped, or delivered. For instance, the Order class can act as the subject, and registered customers can subscribe to receive notifications about their orders' status changes; or the Shipping class can start their process once the Payment class is implemented.

4.3. Structural Patterns

4.3.1. Composite Pattern:

Purpose: Composes objects into tree structures to represent part-whole hierarchies. It allows clients to treat individual objects and compositions of objects uniformly.

Application: Utilized in the online store to represent the Payment class. Each payment is a composite object containing individual Invoice as leaf nodes, enabling secured payment process of every online users when purchasing the products.

4.3.2. Model-View-Controller (MVC) Pattern

Purpose: Separates an application into three main components: Model, View, and Controller.

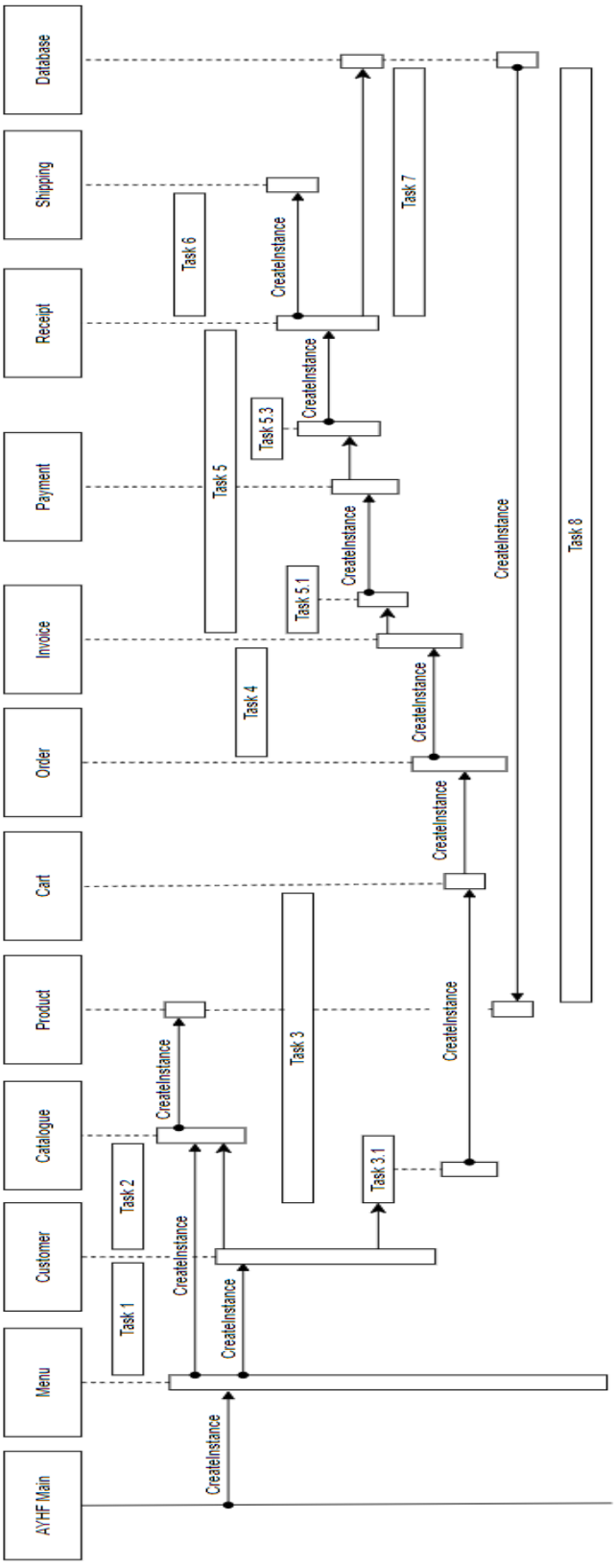
Application: In the "All Your Healthy Foods" online store, the MVC pattern can be applied to separate the application into three main components:

- **Model:** Represents the data model of the online store, including products, customers, orders, etc.

- **View:** Represents the presentation layer of the online store, including the user interface for browsing products, managing shopping carts, etc.

- **Controller:** Represents the business logic layer of the online store, including handling user requests, processing orders, etc. This architectural pattern facilitates a workflow through each of the online store's functions, enhancing maintainability and scalability.

5. Bootstrap Process



Note:

Task	Subtask (if included)
Task 1: Create and manage customer account	
Task 2: Browse the store catalogue	
Task 3: Manage the Shopping Cart	Task 3.1: Customer creates order
Task 4: Create invoices and receipts for customers	
Task 5: Handle payments securely	Task 5.1: Payment detail
	Task 5.3: Print out bills for customers
Task 6: Manage goods packaging and shipment	
Task 7: Record statistics of goods sold based on receipts over various periods	
Task 8: Update the available products as new items become available	

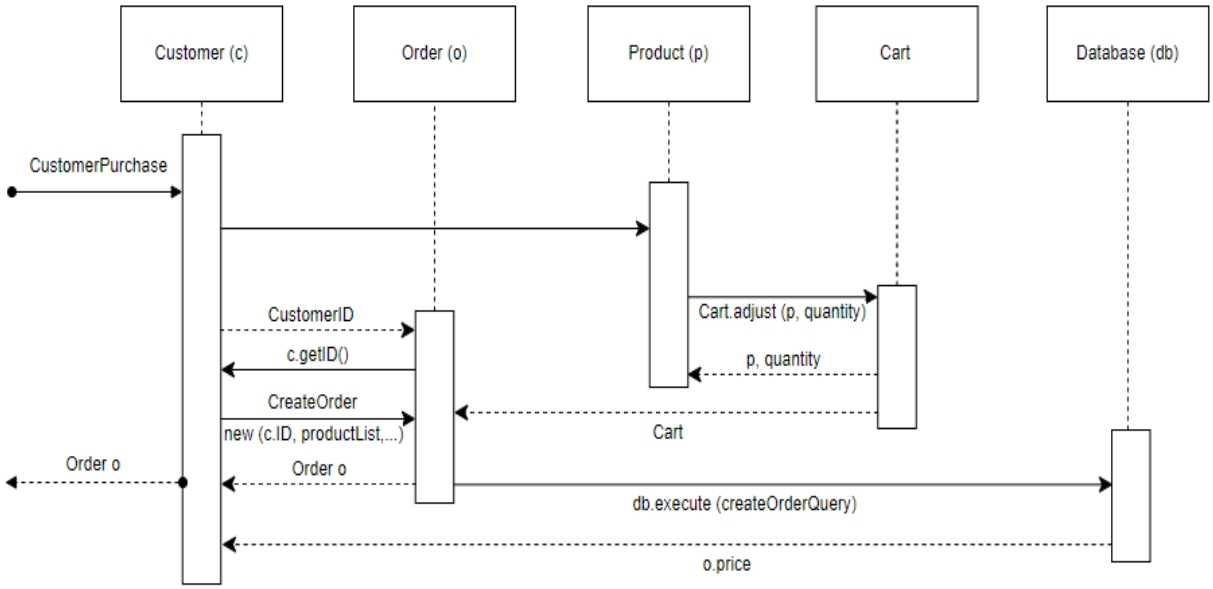
This bootstrap process outlines the sequence of actions involved in a customer browsing, selecting, purchasing, and complete their purchase from the All Your Healthy Foods online store, facilitated by the core classes identified in the case study.

- **Main initializes the Menu class**: Main creates an instance of the Menu class to provide navigation for customers and staffs within the online store.
- **Customer creates an Account for online shopping**: Customers access through Menu, create accounts securely using their email addresses to access the online store and make purchases.
- **Customer browses Product catalog**: Customers navigate through the Menu to browse and select products from the Product catalog.
- **Customer places an Order**: Upon selecting desired products, the Customer class creates an Order representing the customer's request to purchase.

- **Invoice initiates Payment**: The Invoice initiates the Payment process, representing the transaction to complete the purchase securely.
- **Payment completes transaction and generates Receipt**: The Payment class completes the transaction securely and generates a Receipt for the customer's purchase.
- **Shipping prepares order for delivery**: The Shipping class handles the packaging and preparation of the order for delivery, integrating with delivery systems for shipment.
- **Receipt generated for completed transaction**: Upon successful delivery, a Receipt is generated for the completed transaction, providing proof of purchase to the customer.
- **Database records goods sold statistics:** Receipts are recorded in database after being printed out from customer’s payment.

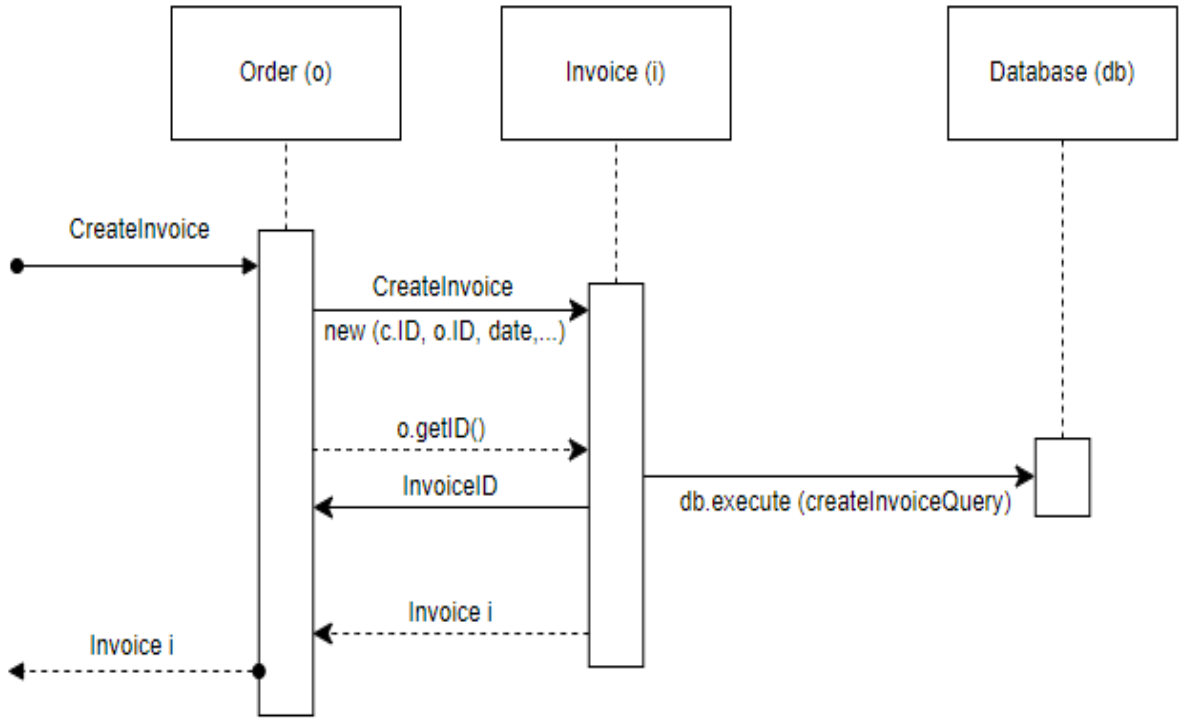
6. Verifications

6.1. Manage the Shopping Cart (task 3):



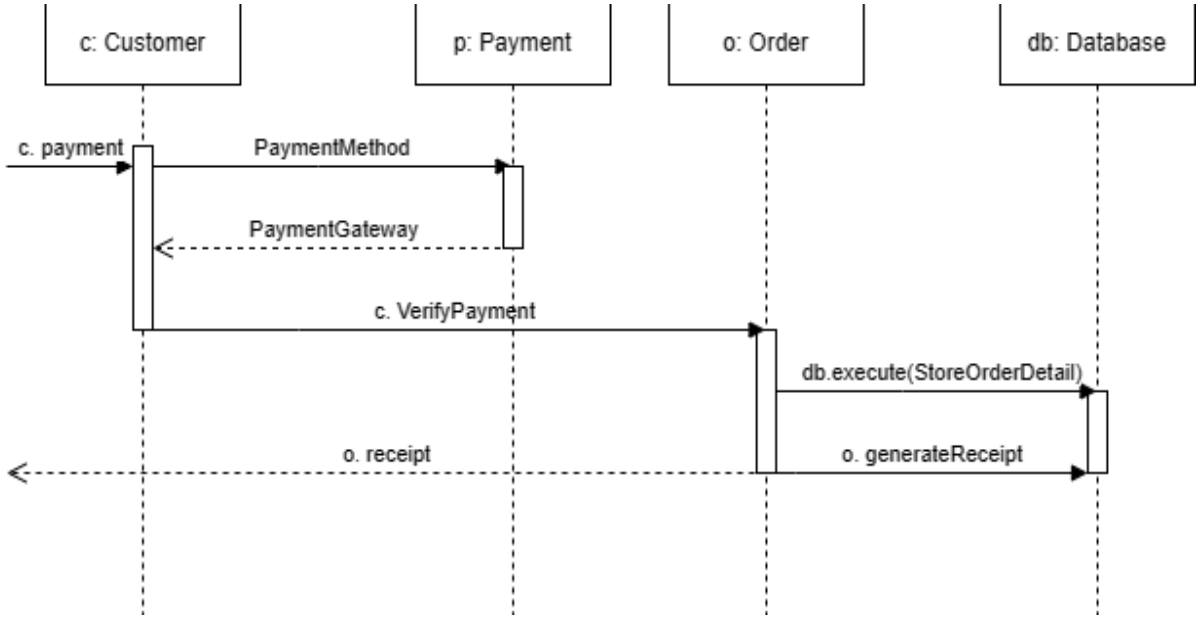
The customer creates new order. The database is informed with the Order creation, using the foregoing back-end implementations to build the query. After that, the customer can start selecting products, and add them to the temporary Cart. After the Cart selection is completed, it returns the information to the Order class, containing products information. Then, the total price of order is calculated by the database, then returned to the Order. Finally, the Order displays desired products with appropriate price, then returned itself to continue following tasks.

6.2. Create invoices and receipts for customers (task 4):



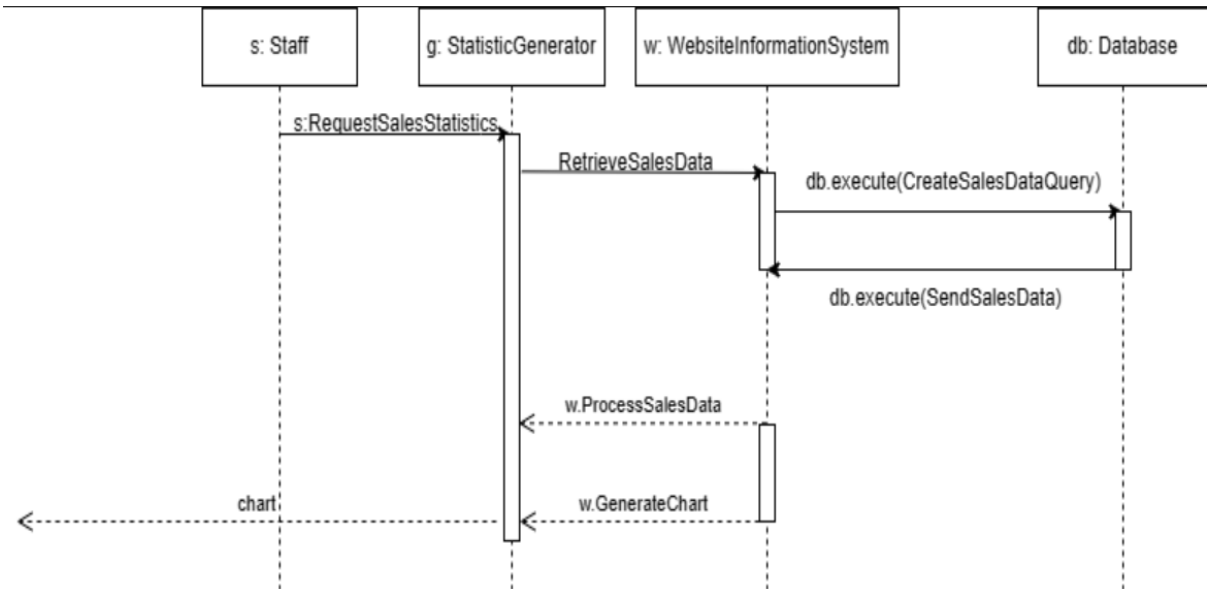
A new Invoice is created using the information of the Order, such as OrderID, CustomerID, Date, etc). Then, the database receive the fundamental information of the invoice, so that a new invoice can be generated, to be sent back to the Customer.

6.3. Handle payment securely (task 5):



The customer interacts with the Payment object, placing an order and choosing a payment method. Then the Payment object becomes active and displays a payment details form (PaymentGateway) to the customer. The customer fills in the necessary payment details. After that, the Order object stores the order details (without payment information) in the Database. Customers receive a receipt after the order is finished.

6.4. Record statistics of goods sold based on receipts over various periods (task 7):



The Staff interacts with the StatisticsGenerator through the website or app to request sales statistics for multiple periods. Then StatisticsGenerator activates and interacts with the WebsiteInformationSystem to retrieve sales data for the requested period. WebsiteInformationSystem then queries the Database based on the request. After that, Database sends sales data to the WebsiteInformationSystem. StatisticsGenerator processes the retrieved sales data, likely calculating statistics for the entire period and generates a chart representing the sales statistics (e.g., number of sales vs. time). The chart now will be displayed through the website.

7. References

Design Heuristics tutorial: https://swinburne.instructure.com/courses/57782/pages/lecture-6-object-design-part-ii?module_item_id=3910455

Design Patterns: https://swinburne.instructure.com/courses/57782/pages/lecture-7-design-patterns?module_item_id=3910463

UML Drawing Tool: https://www.draw.io