Swinburne University of Technology

School of Software and Electrical Engineering

ASSIGNMENT AND PROJECT COVER SHEET

Subject Code: SWE30003 Unit Title: Software Architectures and Design

Assignment number and title: 2, Object Design Due date: 11:59pm, 28th Oct 2024

Tutorial Day and time: 13:00 Tuesday Project Group: 5

Tutor: Mr. Trung Pham

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
104513735	Thai Duong Bao Tan	Tan
1041171133	Le Huu Nhan	Nhan
103805253	Dang Quynh Chi	Chi

Marker's comments:

Total Mark:

Extension certification:

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

Signature of Convener:

Executive Summary

Relaxing Koala, a café and restaurant, is preparing for a significant expansion. However, it faces a number of issues as most of the restaurant's processes, such as recording customer orders or accounting, is still manual. To improve these daily tasks, the restaurant has engaged Swinsoft Consulting to design a comprehensive restaurant information system.

This system will automate key processes like reservations, invoice generation, and basic accounting. It will include a database to track essential metrics on menu items, guests, and orders. Additionally, an online menu feature will allow customers to view options and place takeaway orders directly. The system is designed with scalability in mind, allowing for seamless integration of future features as Relaxing Koala continues to grow.

By implementing this new system, Relaxing Koala is expected to increase its customer capacity from 50 to 150 as planned. The owners can now manage and plan future strategies more efficiently with the restaurant's statistics tracking ability. From the customers' perspective, their experience will also be much better as the menu can be easily accessed online and menu items can be delivered right to their homes.

This document provides an analysis of the problem, details on the classes with CRC cards, UML diagrams, and outlines the bootstrap and verification processes.

Table of Contents

Problem Analysis

Assumptions

A1. Relaxing Koala has enough staff to cover all roles.
A2. Each staff member sticks to their assigned role.
A3. Each order has a unique ID.
A4. Food and beverages at Relaxing Koala are not pre-prepared.
A5. When an order is placed, the cashier will receive it and notify the kitchen
A6. An order is marked as completed/done only when it has been served and payment is received. This applies to both dine-in and takeaway orders.
A7. Customers are allowed to customize their orders (e.g., less ice in a drink).
A8. Once food or drinks are ready, the kitchen staff will notify the waiter directly.
A9. The waiter knows which table to deliver food to for dine-in customers.
A10. When delivery personnel (DP) deliver an order to a customer, they take a picture of the customer receiving the order. The DP then uploads this picture to the system as proof of delivery.
A11. There is a designated specialist who will review the uploaded image of DP to ensure it clearly shows the customer receiving the order. The order will be marked as completed only after the specialist confirms that the image is acceptable.
A12. Each table has an unique ID.
A13. Customers are required to provide their name, contact information (phone or email), party size, and time when making a reservation.
A14. Relaxing Koala has reservationist(s) to receive customer reservation requests, update and check table availability, and confirm the requests.
A15. Customers must pay a deposit if the party size is larger than 10 people. The deposit from customers when making a reservation is processed manually by the reservationist in person.
A16. Customers can use different payment methods when settling an invoice.
A17. The system tracks data on menu items ordered and order frequency to generate sales statistics and popular menu items reports.

Simplifications

To reduce system complexity and lower operational costs for the Relaxing Koala, the system has been designed as a semi-automated solution. This means that the system's primary function is to receive requests only, while staff handle the actual processing of those requests. The simplifications include:

- Although there are two types of orders (Dine-in and Takeaway), they will not be separated into distinct classes under a single superclass because the differences are minor and can be managed entirely by staff.
- The system will only receive delivery requests, the assignment of delivery personnel will be manually managed by a specialist.
- The system will not track the delivery route or distance traveled. Delivery personnel will use tools like Google Maps for navigation, and any live tracking would need to be handled through third-party apps if necessary.
- While invoices and receipts serve different purposes, printing or generating both for each order would be inefficient. Instead, they are combined into a single document to improve efficiency and reduce paper consumption.

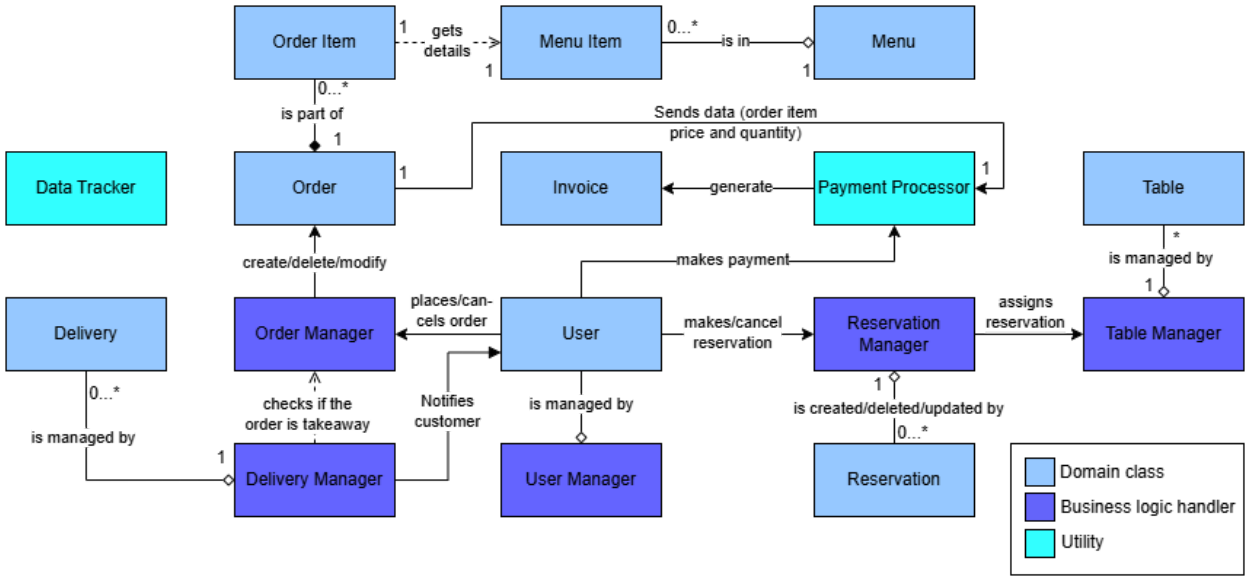
Candidate Classes

Overview

Candidate Class List

- Data Tracker
- Delivery
- Delivery Manager
- Invoice
- Menu
- Menu Item
- Order
- Order Item
- Order Manager
- Payment Processor
- Reservation
- Reservation Manager
- Table
- Table Manager
- User
- User Manager

UML Class Diagram



Justification

After careful consideration, several candidate classes were removed in order to select those that are believed to be truly relevant to what the Restaurant Information System needs to manage. These include: viewing the menu, placing orders, making reservations, processing payments, managing deliveries, and tracking order data.

Plus, this design assumes that the system will operate as a semi-automated solution, where human staff work alongside the system, handling tasks that don't require full automation.

As shown in the UML class diagram, there are three main groups: Domain Classes, Business Logic Handlers, and Utility Classes.

- Domain Classes represent the key entities, like Orders, Menu Items, and Reservations, that the system needs to manage.
- Business Logic Handlers are the classes that will manage the primary functions and operations of the system.
- Utility Classes provide functionalities that support the system's operations without holding core business data, providing essential services such as tracking data and facilitating payment processing.

Regarding the candidate classes that were excluded, it was first realized that including classes for Customer, Staff and its subclasses would complicate the system greatly. For example, managing staff information would require the implementation of a staff

scheduler. As a result, all of these were combined into a single unified class called User, which simplifies the structure considerably.

CRC Cards

Data Tracker

Class name: Data Tracker **Superclass:** None	
Monitors and records system data, primarily used for analysis and reporting on orders.	
Responsibilities	Collaborators
Knows menu item sales data Generate reports or statistics	

Delivery

Class name: Delivery **Superclass**: None	
Represents an order's delivery details, such as delivery address and status.	
Responsibilities	Collaborators
Knows delivery status Knows delivery address	

Delivery Manager

Class name: Delivery Manager **Superclass:** None	
Manages all delivery operations, such as creating delivery tasks, assigning deliveries to drivers, and notifying customers about delivery status.	
Responsibilities	Collaborators
Knows list of deliveries Create new delivery Update delivery status Notify customers	Delivery Order Manager User

Invoice

Class name: Invoice **Superclass**: None	
Contains billing information for an order, including itemized charges, and total price.	
Responsibilities	Collaborators
Knows date Knows order ID Knows order items' names Knows order items' quantity Knows price per item Knows total price	Payment Processor

Menu

Class name: Menu **Superclass**: None	
Displays a list of all available menu items for customers to view and order.	
Responsibilities	Collaborators
Knows list of menu items Add menu item Remove menu item Fetch specific items based on search	Menu Item

Menu Item

Class name: Menu Item **Superclass**: None	
Represents individual items on the menu, with attributes such as name, description, price, and availability.	
Responsibilities	Collaborators
Knows menu item's name Knows menu item's price Knows menu item's availability Update item price Update item availability	

Order

Class name: Order **Superclass**: None	
Manages a list of customer's ordered items.	
Responsibilities	Collaborators
Knows ID Knows list of Order Items Knows type Knows order status	Order Item

Order Item

Class name: Order Item **Superclass**: None	
Represents a single item within an order, including its details, quantity, and any special requests.	
Responsibilities	Collaborators
Knows quantity Knows specific Menu Item associated with Knows customizations or special instructions Knows price	Menu Item

Order Manager

Class name: Order Manager **Superclass**: None	
Handles the creation, modification, and cancellation of orders, ensuring that they are properly recorded and processed.	
Responsibilities	Collaborators
Receive order request Create new orders Update orders Delete orders Knows the list of orders being processed	Order

Payment Processor

Class name: Payment Processor **Superclass**: None	
Handles payment transactions, including generating invoices, processing payments, and updating payment status.	
Responsibilities	Collaborators
Knows payment status Knows payment methods Calculate total price Process payments Generate invoices	Invoice Order

Reservation

Class name: Reservation **Superclass**: None	
Manages the details of a customer's reservation, such as the date, time, and number of guests.	
Responsibilities	Collaborators
Knows customer name Knows customer email/phone Knows party size Knows time Knows assigned table Knows reservation status	

Reservation Manager

Class name: Reservation Manager **Superclass**: None	
Manage the reservation process, allowing customers to make, update, or cancel reservations, and allow reservationists to assign tables.	
Responsibilities	Collaborators
Knows all reservations Update reservation status Cancel a reservation	Reservation User Table Manager

Create a new reservation Receive reservation request	

Table

Class name: Table **Superclass**: None	
Represents a dining table in the restaurant, including its ID and current status.	
Responsibilities	Collaborators
Knows ID Knows availability Update status	

Table Manager

Class name: Table Manager **Superclass**: None	
Manages the availability and assignment of tables.	
Responsibilities	Collaborators
Knows availability of all tables Assigns tables to reservations	Table Reservation Manager

User

Class name: User **Superclass**: None	
Represents both customers and staff in the system, storing details like name, contact information, and role.	
Responsibilities	Collaborators
Knows name Knows phone number Knows email Knows role Knows login credentials Places order Makes reservations	Order Manager Reservation Manager Payment Processor

Makes payment	

User Manager

Class name: User Manager **Superclass**: None	
Manages user accounts, allowing for the creation, update, and deletion of user information.	
Responsibilities	Collaborators
Knows all registered users Assign user roles Creates users Updates users' info Delete users	User

Design Quality

Design Heuristics

H1. A class should capture one and only one key abstraction: All the key abstractions in the restaurant information system like order, reservation, table, menu are implemented in individual classes to allow for independent modifications or improvements when necessary.

H2. Do not create god classes/objects: This design clearly does not have any behavioral god classes, as most of the decision-making responsibilities are distributed among the business logic handlers. Similarly, the domain classes such as Reservation, Delivery, Invoice contain different information instead of concentrating in one data god class.

H3. All data should be hidden within its class: Attributes in each class are made private and accessed only through specific methods.

H4. Keep related data and behavior in one place: For instance, much of the information of a user like name, phone number, email is placed in the User class or behaviors like assigning user’s roles, creating users, updating users’ info is in the User Manager class

H5. Eliminate irrelevant classes from the design: Classes like viewing the menu, placing orders, and making reservations have been excluded, as it is assumed that many parts of the system are semi-automated with human’s participation

H6. Spin off non related information into another class: The Order class potentially includes delivery-specific details relevant only to takeaway orders, such as delivery address and status. However, they are not relevant to dine-in order, so a separate Delivery class has been created to manage these details.

H7. Global data or functions should not be used for bookkeeping information: No global data or functions are present in the system, as all necessary variables and methods are assigned to their respective classes.

Design Patterns

Singleton Design Pattern

This design pattern is suitable for classes that require only one instance with a globally accessible point of access. In this case, the Data Tracker class is a great candidate for a singleton because it is primarily responsible for database interactions such as connecting, querying, logging, and data handling. By ensuring only one instance, thread safety is improved, and memory usage is minimized, especially since multiple instances may result in conflicting access or redundant memory usage.

Strategy Design Pattern

This design pattern is useful for classes that require multiple interchangeable algorithms or methods, as it allows clients to change these methods without affecting the underlying code. The class Payment Processor is a good example here, as it handles various payment methods, such as cash or credit cards. Implementing a strategy pattern for payment processing enables seamless switching between payment methods, giving the customer the flexibility to choose their preferred method.

Mediator Design Pattern

This design pattern defines an intermediary that manages and restricts communication between objects, centralizing it within the mediator class. In this system, the Order Manager class can serve as a mediator. It facilitates interactions between the User and Order classes. This mediation facilitates efficient communication for tasks such as placing, modifying, and deleting orders. This centralization reduces dependencies between classes and streamlines the flow of interactions.

MVC Design Pattern

Model-View-Controller (MVC) is ideal for organizing the system into three main components: Model, View, and Controller. This pattern makes it easier to update, fix bugs, or add new features:

- Model: The data layer is represented by domain classes such as Table, User, and Menu. They manage and store data, enforce business rules, and respond to requests from the View and Controller layers.
- View: This layer is represented by the front-end of the website or mobile app, where customers view the menu, make reservations, or order takeout, and restaurant staff can check orders and track statistics.
- Controller: Business logic handler classes such as Order Manager, Reservation Manager, and Table Manager interpret user input, update the Model as needed, and adjust the View to reflect Model changes.

Bootstrap Process

Initialization

Menu initialization:

- The system automatically initializes the Menu class and fetches menu data from the database.
- The menu is viewable by all users but can only be modified by the admin.

User initialization:

- The system checks if the username and password provided by the user match those stored in the database.
- If valid, a User class instance is created.
- Users can create, edit, or cancel new and existing orders.

Order Manager initialization:

- The system initializes the Order Manager class.
- The Order Manager can handle the creation, editing, and cancellation of orders initiated by either customers or admin.

Reservation Manager initialization:

- The system initializes the Reservation Manager class.
- The Reservation Manager can create new reservations and assign them to tables.

Delivery Manager initialization:

- The system initializes the Delivery Manager class

- It manages the creation of deliveries and notifies users of their delivery status.

Payment Processor initialization:

- The system initializes the Payment Processor.
- The Payment Processor can create invoices and update the invoice payment status.

Process

Customer Accessing the Menu

- System initializes the **Menu** class
- **Menu** class creates instances of **Menu** Items.
- **Menu Item** instances fetch data from the database.
- **Menu Item** instances display the menu data in a viewable interface for the customer.

Customer Creating an Order

- **User** interacts with the **Order Manager** to place an order.
- **Order Manager** creates an **Order** instance.
- **Order** instance generates **Order Item** instances based on the items selected by the customer.
- **Order Item** instances fetch item details from **Menu Item** and add customer-specified quantities.
- **Order** instance sends order data, including item details and quantities, to the **Payment Processor**.
- **Payment Processor** calculates the total price based on the data received from the **Order**.
- **Payment Processor** generates **Invoice**.

Delivery Order Processing

- **Delivery Manager** sends a request to the **Order Manager** to check if the order is marked as takeaway.
- If the order is takeaway, the **Delivery Manager** creates and manages the **Delivery**.
- **Delivery Manager** notifies the customer of the delivery status.

Reservation Processing

- **User** interacts with the **Reservation Manager** to make a reservation.

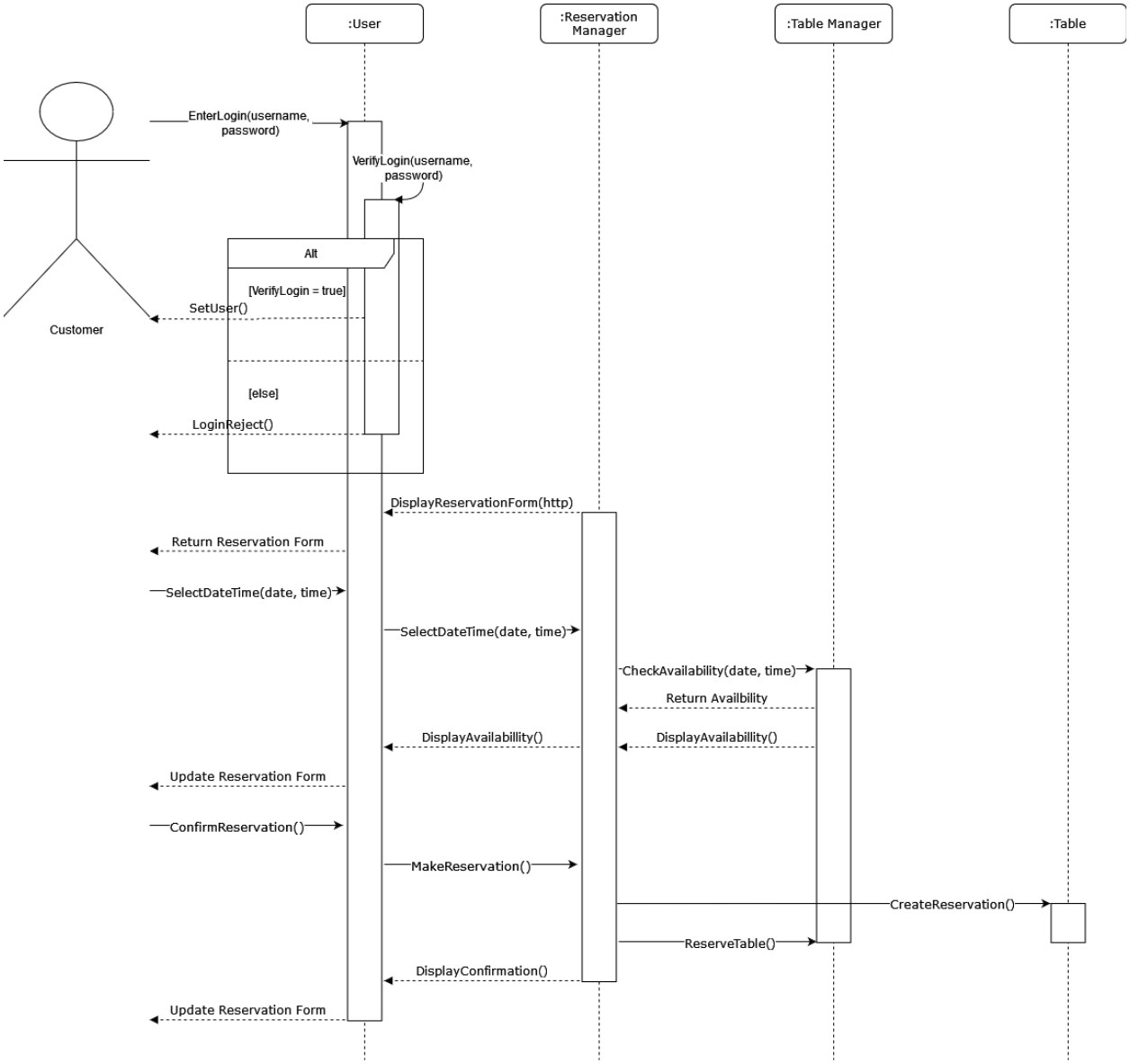
- **Reservation Manager** creates a new **Reservation** instance.
- **Reservation Manager** requests table assignment from the **Table Manager**.
- **Table Manager** assigns an available **Table** to the **Reservation**.

Order Completion and Payment

- **User** sends a completion signal to the **Order Manager**.
- **Order Manager** updates the **Order** status to "complete."
- Necessary data of the completed **Order** is sent to the database.
- **Data Tracker** retrieves and analyzes data from the database.
- **User** makes payment via the **Payment Processor**.

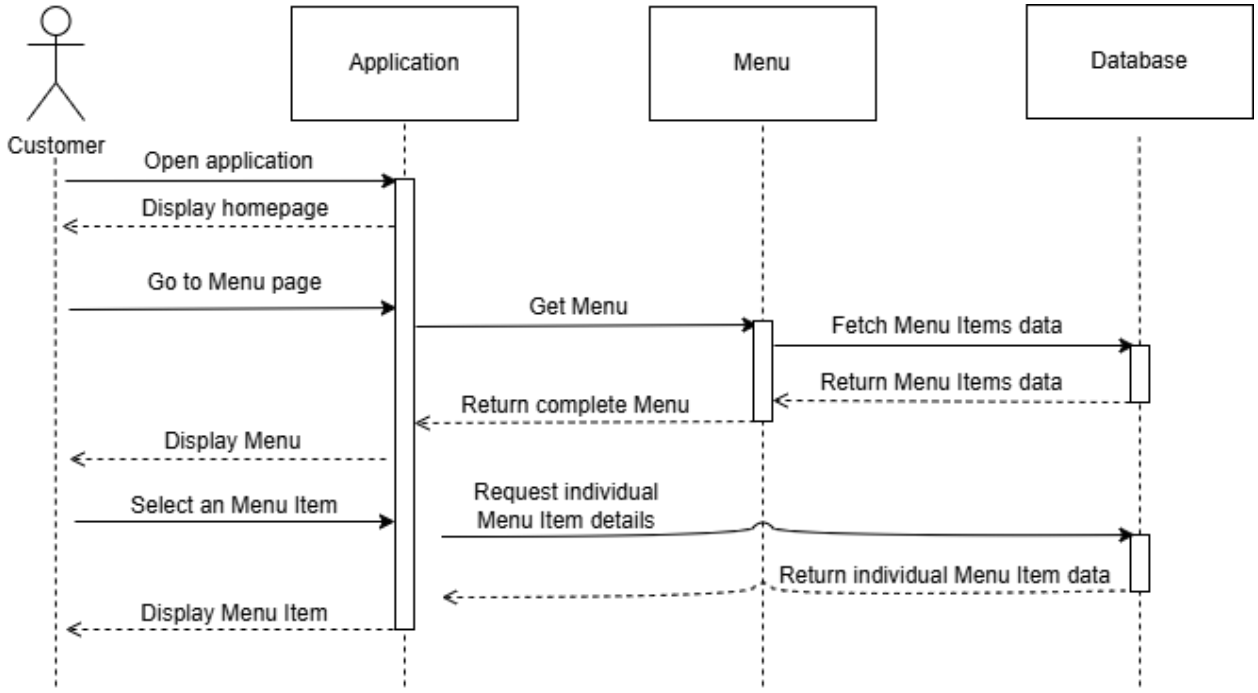
Verification

Making Reservation



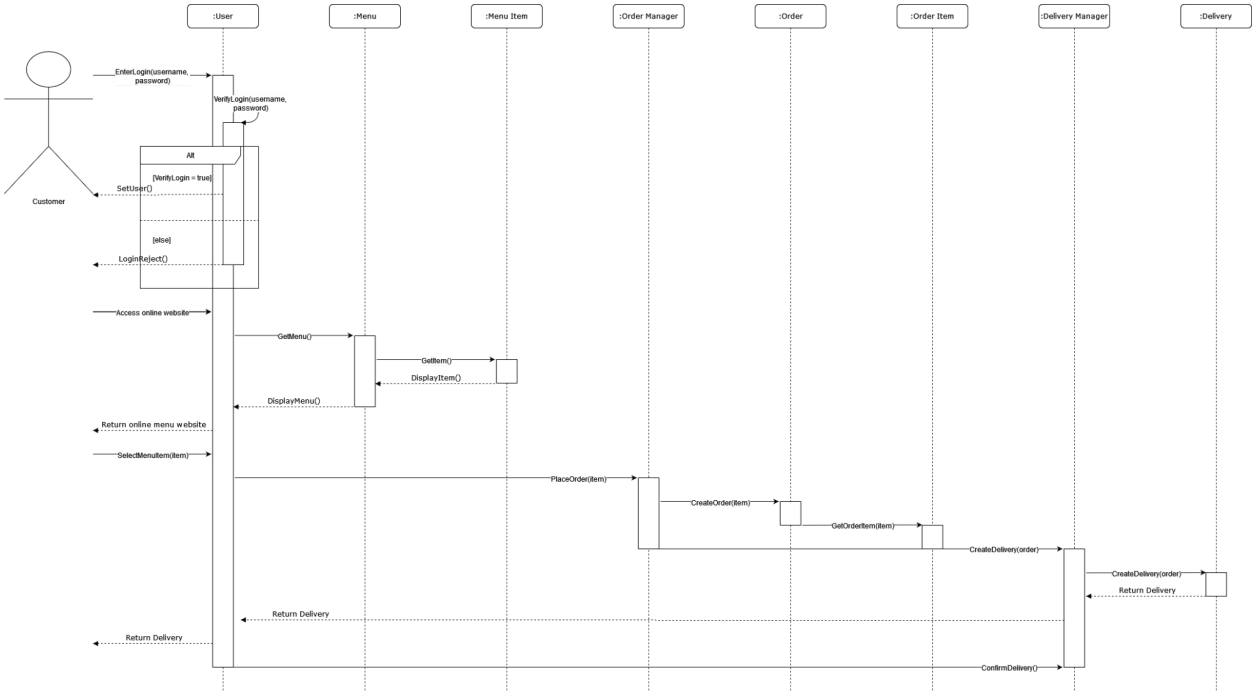
To make a reservation at Relaxing Koala, the customer needs to log in with their username and password credential first. Upon successful authentication, the user interacts with the Reservation Manager to select a desired date and time for their reservation. The Reservation Manager then communicates with the Table Manager to check table availability for the specified date and time. If a table is available, the user confirms the reservation, which prompts the Reservation Manager to instruct the Table Manager to create the reservation with the Table class. After that, the Reservation Manager displays the confirmation to the user, finalizing the reservation process.

Viewing Online Menu



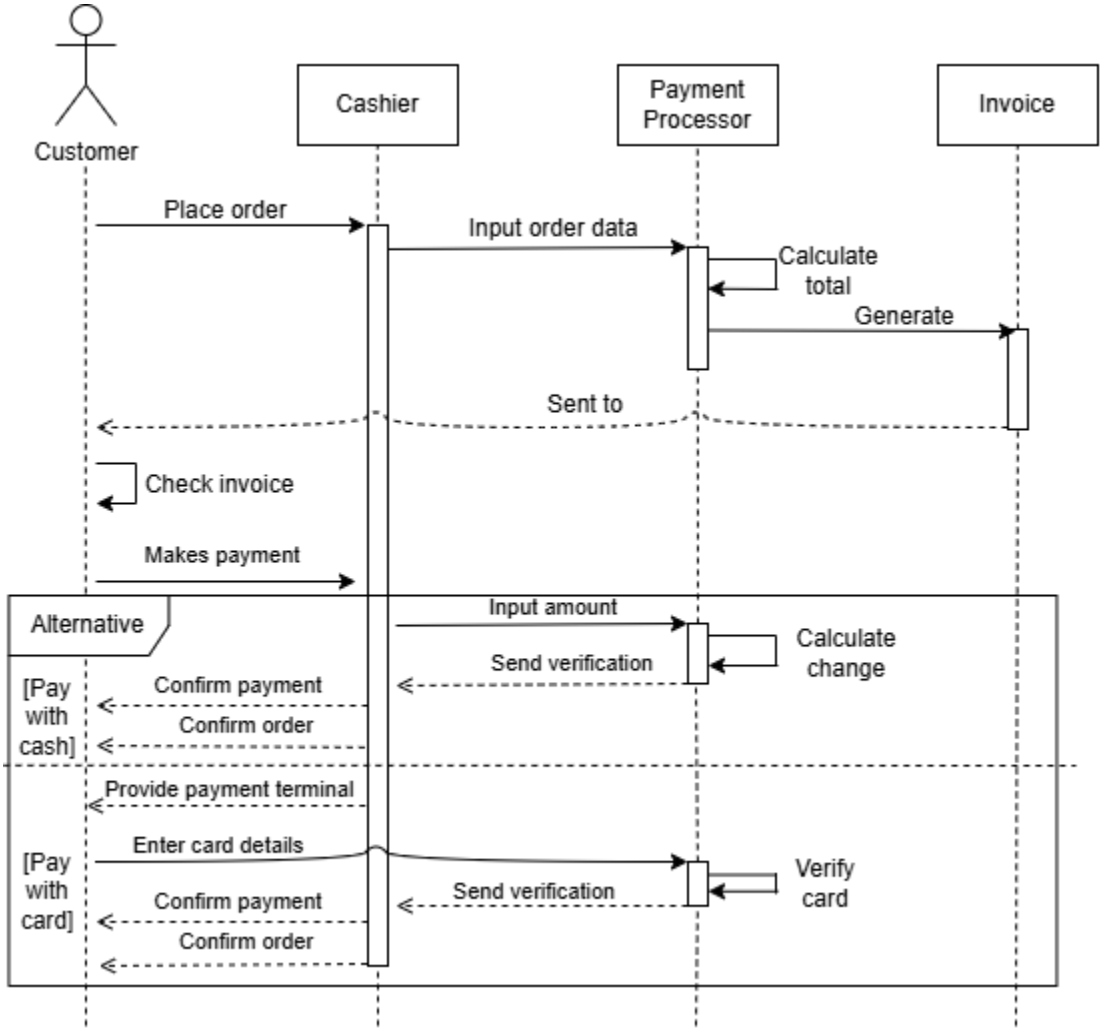
This sequence diagram shows how the restaurant information system's application initializes and displays the menu. The process begins when the *Customer* accesses the Menu page of the *Application*, which then makes a request to access the *Menu*. This request causes the *Menu* to begin its own initialization process, querying the *Database* to retrieve the necessary menu data. When the *Database* receives this request, it retrieves the relevant menu item information and returns it to the *Menu*. With the menu data available, the *Menu* can now display the items, allowing the *Application* to present the user with a complete and interactive menu. This interaction enables customers to view the available options directly, which serves as the foundation for order placement and further system engagements.

Ordering Takeaway Food



The UML sequence diagram illustrates the process of ordering takeaway food, beginning with the customer logging in the application with username and password like the making reservation process. After that, the customer can view the menu in the website with the DisplayMenu() method in the Menu class. The customer then selects one menu item fetched from the MenuItem class and the Order Manager will create a delivery order with the Delivery Manager after the order is done. The customer then receives the delivered food or drink returned from the Deliver class and then confirms it with the Delivery Manager.

Dine In Paying



This sequence diagram illustrates the dine-in payment process within the restaurant information system. The process begins when the *Cashier* initiates a payment request, which is directed to the *Payment Processor*. The *Payment Processor* then generates an *Invoice* and retrieves the necessary payment information. Once these details are obtained, the *Customers* have the option of paying with cash or card.

If the *Customer* chooses to pay in cash, the *Cashier* collects the cash and confirms the transaction after the *Payment Processor* sends verification to the *Cashier*. Alternatively, if the *Customer* chooses to pay by card, the *Cashier* provides a payment terminal for the *Customer* to enter their card details. These details are then sent to the *Payment Processor* for verification. If the payment is valid, the *Cashier* confirms the payment.

References

Assignment 1 - Requirements Specification |
SWE30003_Assignment1_Group5_new-1.pdf