Khaing Pwint Wai Tun(s23840)

User Requirements

Restaurant Management System

The restaurant management system is a platform that is designed for the restaurant's owner to manage their business more efficiently. With this system, the owner can create and manage multiple restaurants in one place. The customers will also be able to make reservation or order from the restaurants easily with this system. The employees from the restaurants will also be able to manage reservations and orders more efficiently by using the system.

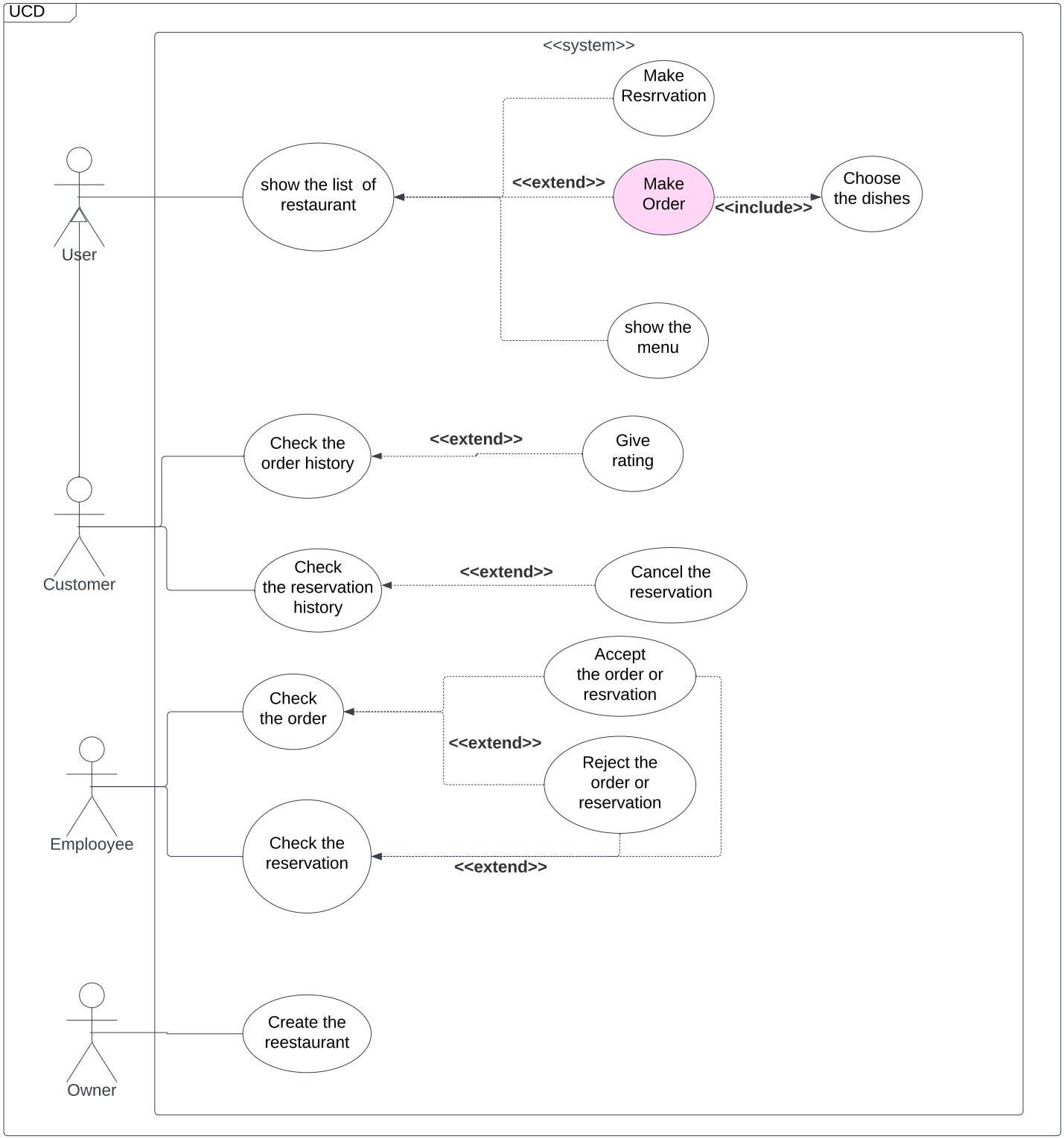
In the system, a lot of restaurants are registered. Restaurants can be divided based on the type. They can be café, bar, normal restaurants. And they can be different types at the same time.

Restaurants will have the menu and the menu will have different dishes. The system should allow the user to check different restaurants and check the menus. The user will be able to order or make the reservation at the restaurant they chose. To make order, the user must choose the dishes. After choosing the dishes, they can choose the order type such as with pick up or delivery, standard or express order.

To make the reservation the users should fill the form with required details and send the request. After they make the order, they will be able to choose the payment method such as online payment or cash. To check the order history and reservation history, the user must be log in and be in the database.

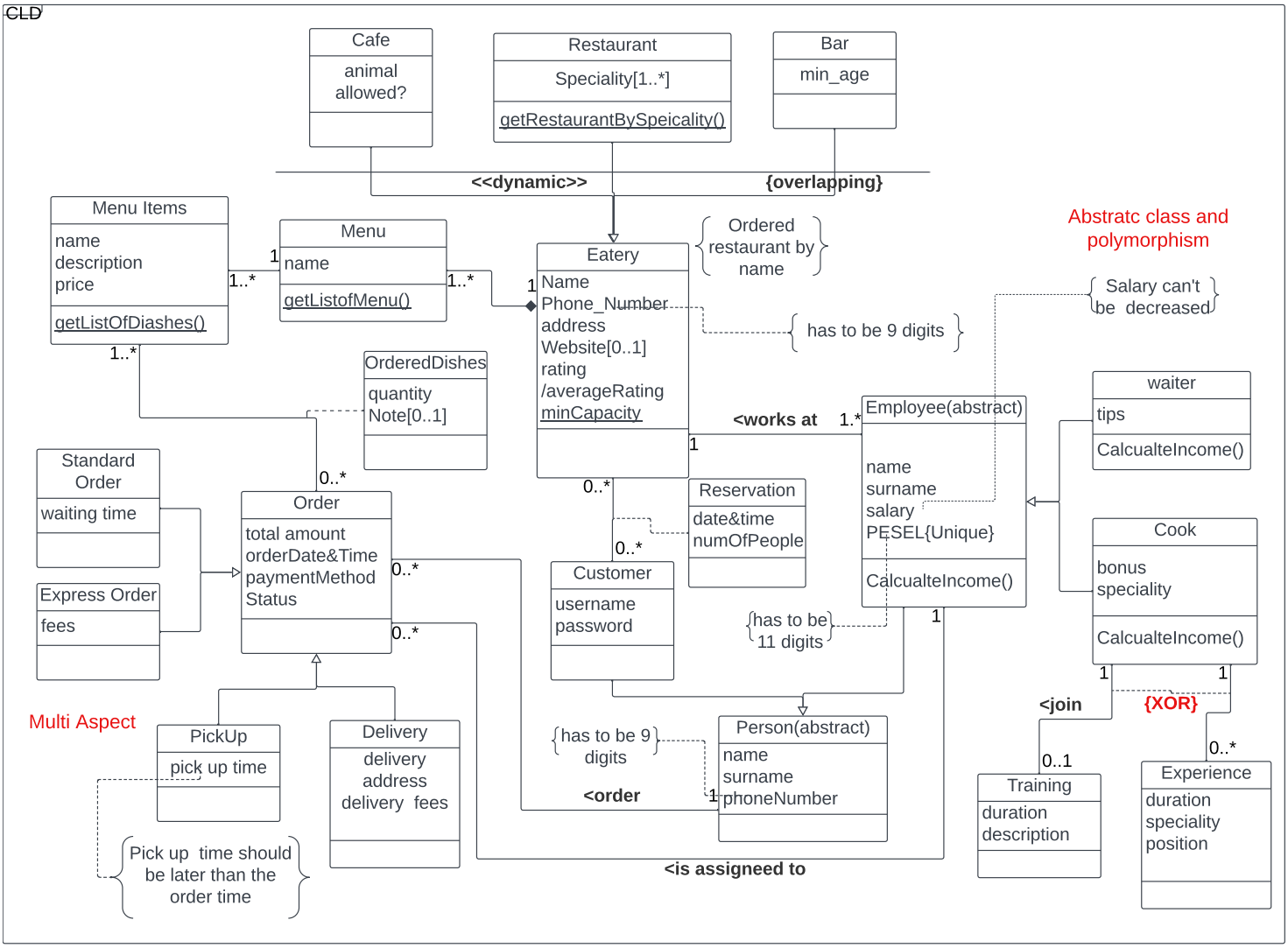
The employees from the restaurant must be able to check the orders and reservation request from the customer. They can also cancel or accept the order or reservation.

The Use Case Diagram

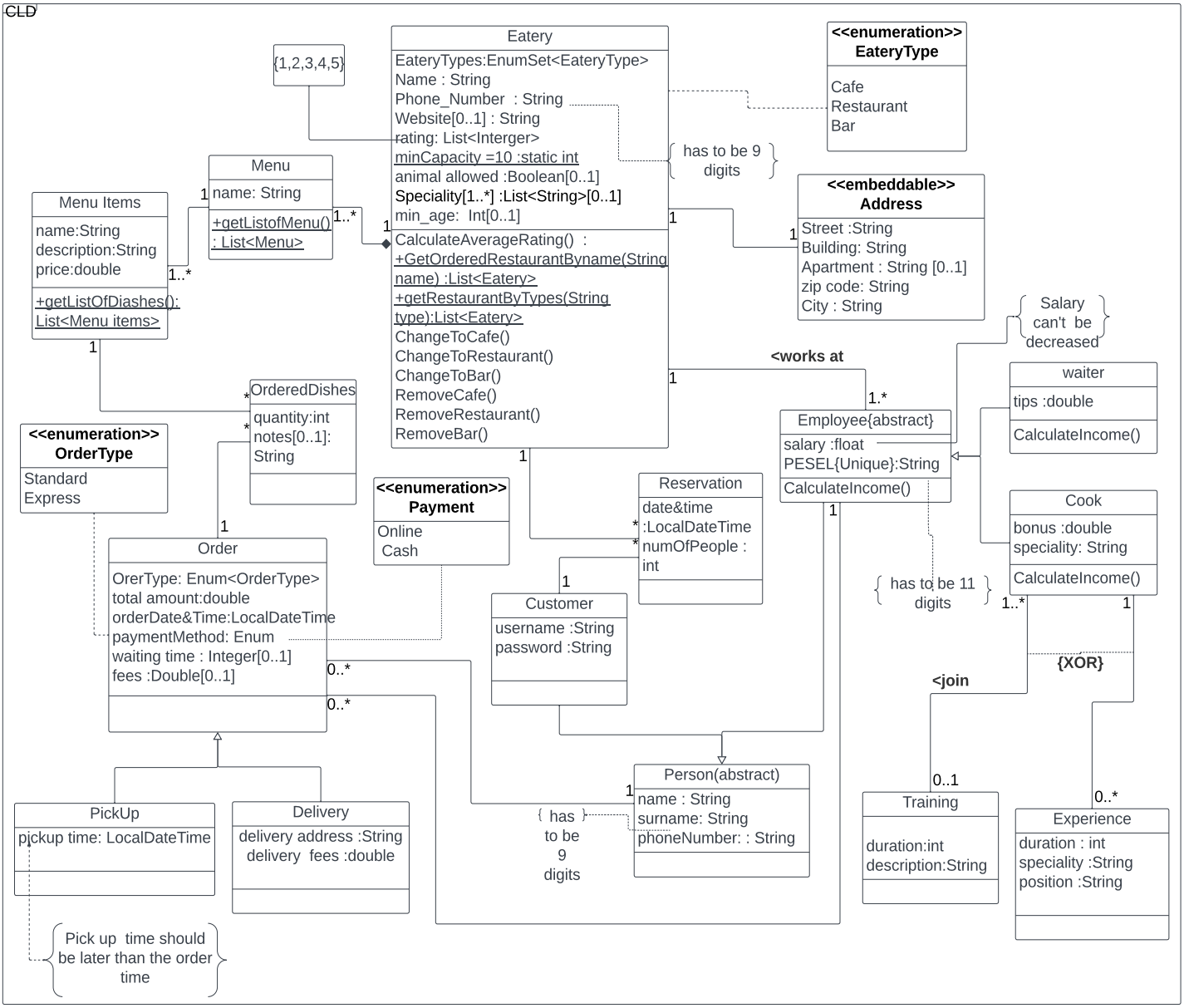


Khaing Pwint Wai Tun(s23840)

The Class Diagram (Analytical)



The Class Diagram (Design)



Khaing Pwint Wai Tun(s23840)

The Scenario of Selected Use Case (as text)

Use case: Make order

Actor: User

Pre-conditions:

1. To make order, the actor selected a desired restaurant from the lists of restaurants that the system shows.

Basic flow of events:

1. The actor clicks on a button “Order” and starts a use case make order.
2. The system shows the list of menu.
3. The actor selects the desired menu.
4. The system shows the list of dishes from selected menu.
5. The actor chooses the desired dishes and the quantity and clicks “check out” button.
6. The system shows the form with the availability of order types, the dishes which the actor chose, the total amount and payment method and the option to edit, cancel and order.
7. The actors select the order type (standard or express) and click “order”.
8. The system shows the form to confirm the order.
9. The actor confirms the order.
10. The system added the order to the database and shows the form with the option to check order history or to make payment and go back to the main page.

Alternative path:

7a. The actor selects the option to edit the order.

7aa. The system returns to the step 4.

9a. The actor selects the option to cancel order.

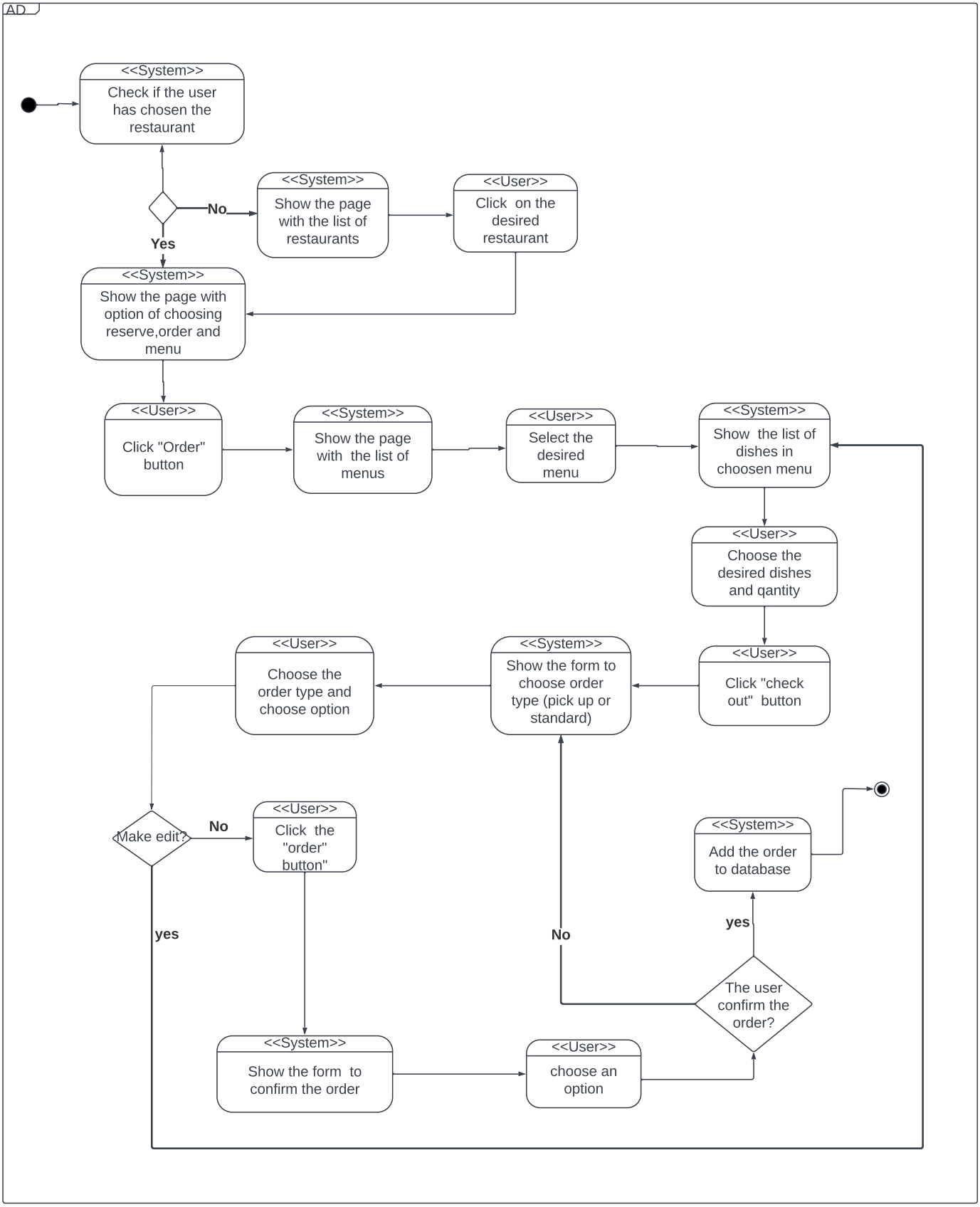
9aa. The system goes back to the step 7.

Post-conditions:

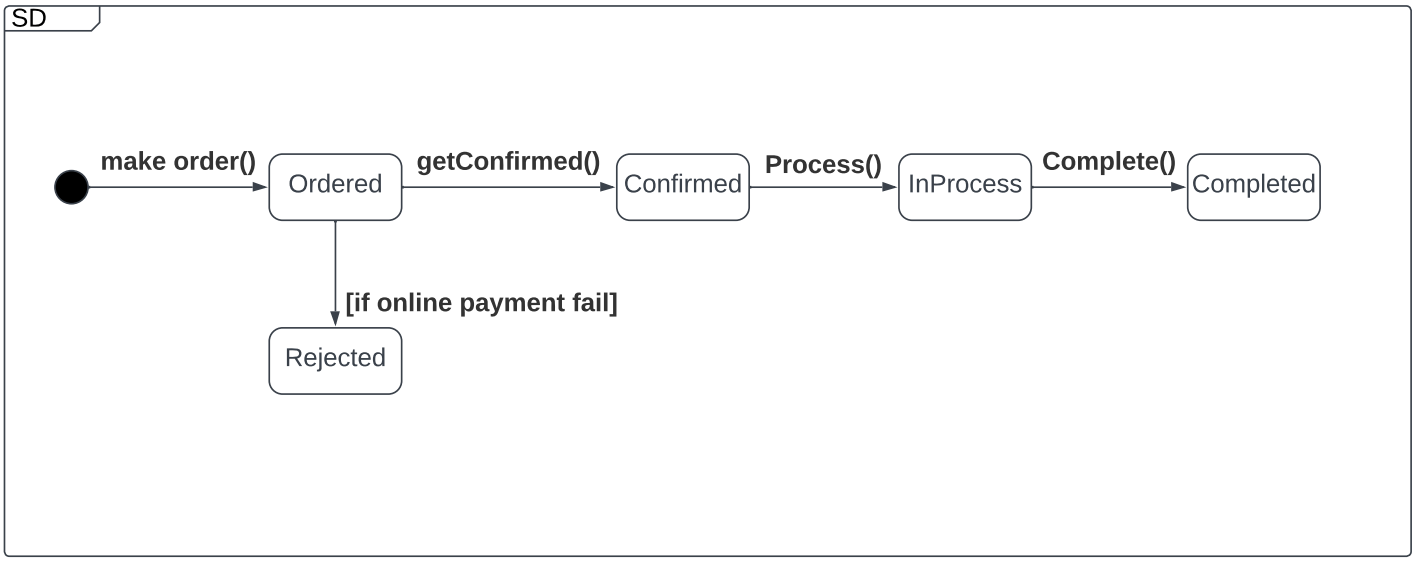
The user successfully made the order.

Khaing Pwint Wai Tun(s23840)

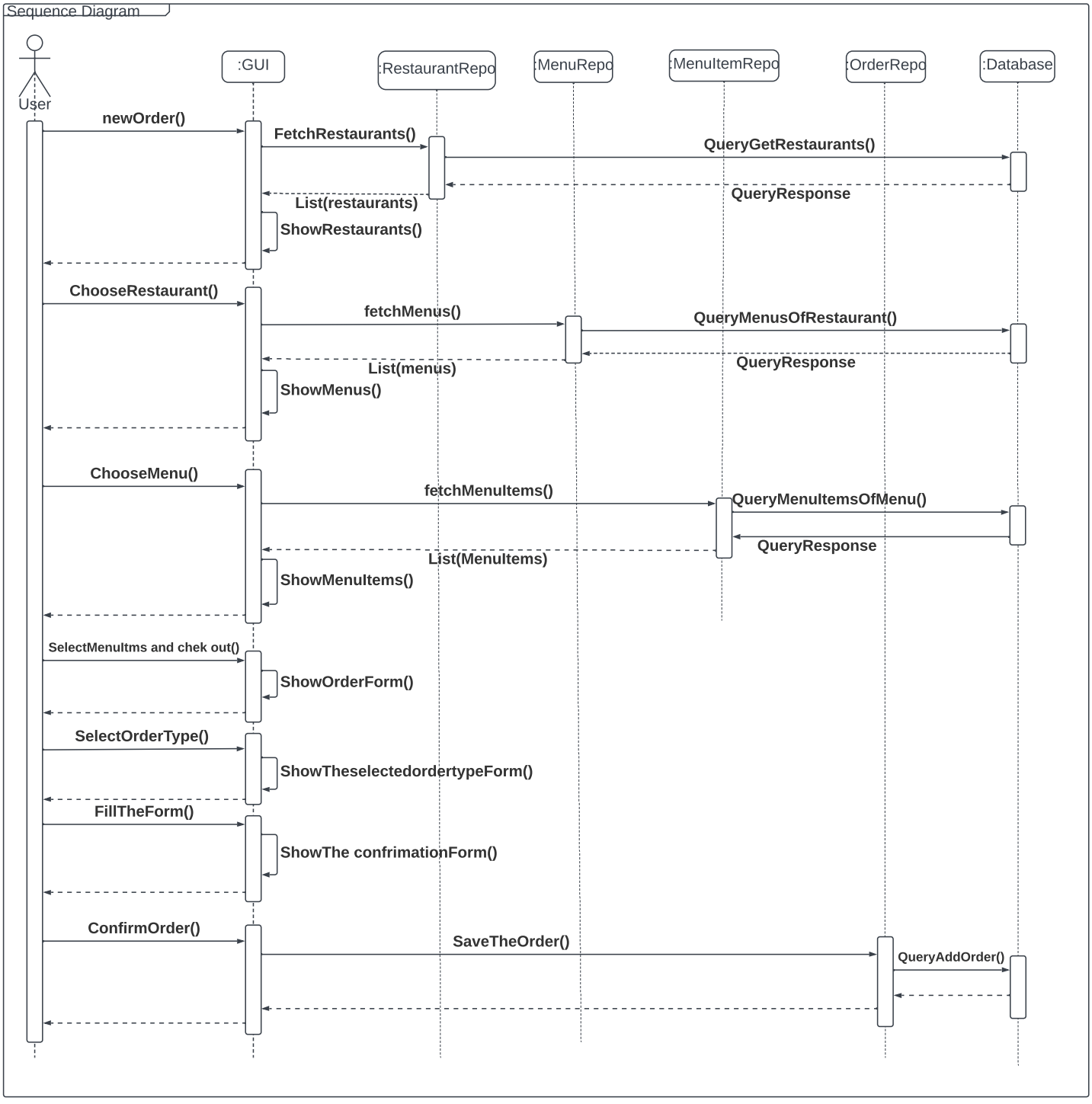
The Activity Diagram Picked Use Case



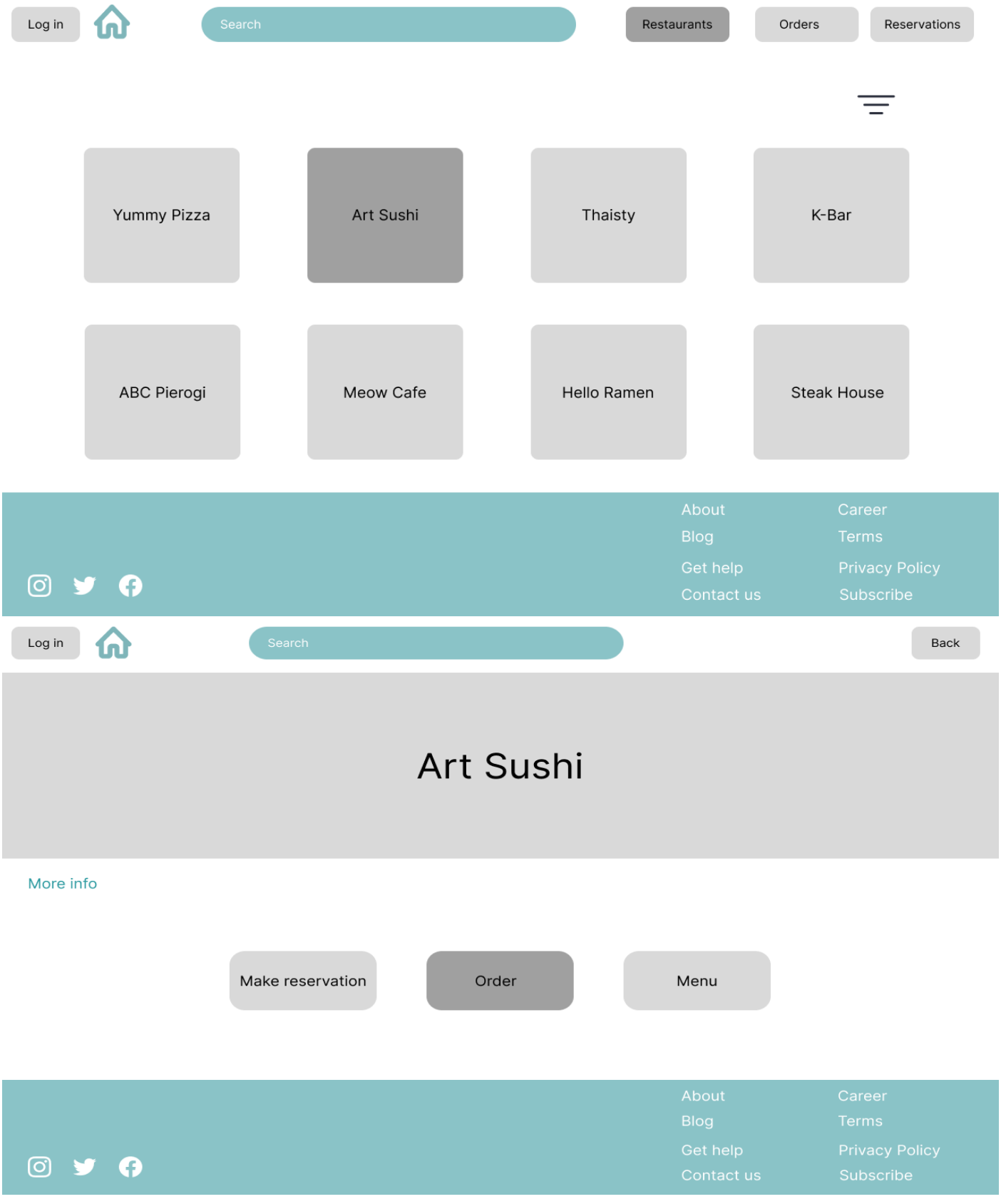
The State Diagram for Selected Class(Order class).



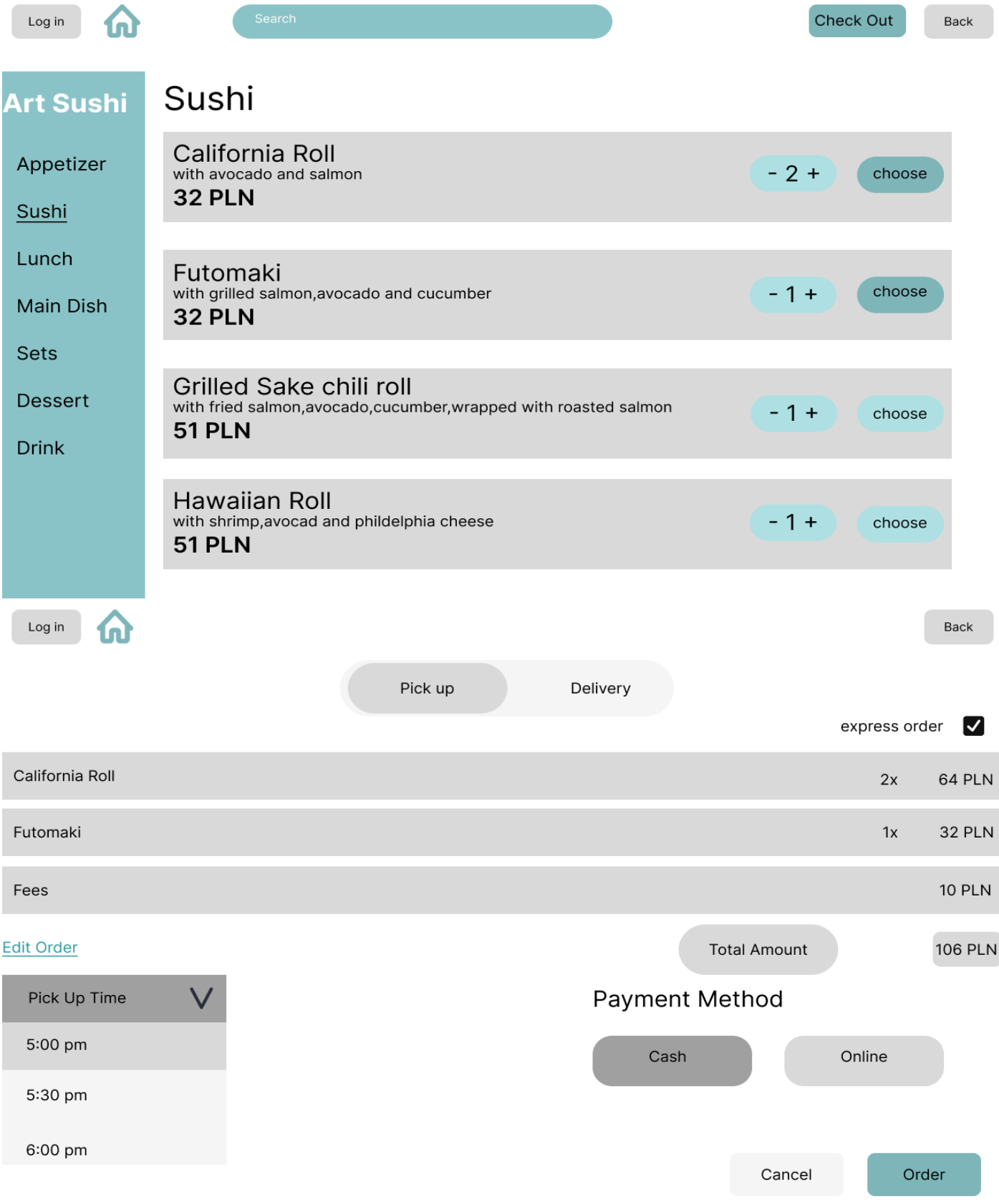
The Interaction (sequence) Diagram for Selected Use Case



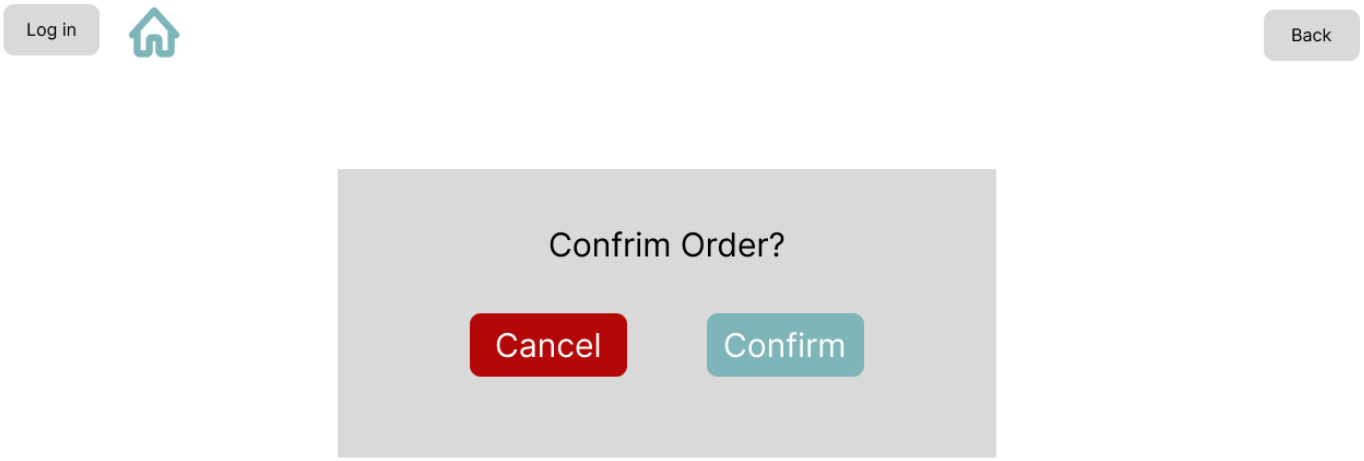
The GUI Design

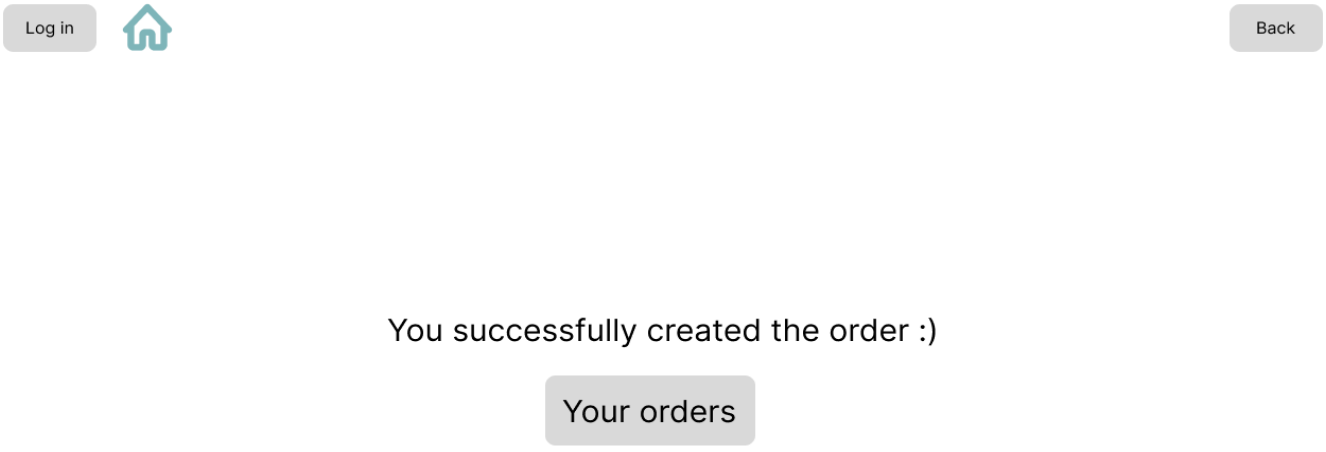


Khaing Pwint Wai Tun(s23840)



Khaing Pwint Wai Tun(s23840)





Khaing Pwint Wai Tun(s23840)

The Discussion of Design Decisions and the Effect of Dynamic analysis

As classes will be implemented using hibernate entities, ID will be generated in each class.

Complex Attributes

The complex attribute will be implemented using @Embeddable and @Embeded annotations to map it. In this project, Address is the complex attribute and In address class, @Embeddable annotations will be used and Eatery class includes @Embeded attribute of type Address. Hibernate will automatically map the individual properties of the Address class to columns in the Employee table.

Multi-value Attributes

To implement multi-value attributes, collection mapping method @ElementCollection annotations will be used. Specialty of restaurant in this case. The restaurant can have many specialized cuisines such as Chinese, Japanese, Italian, American etc.

Optional Attributes

To implement optional attributes, the attributes can be marked by @Column (nullable = true) annotation as hibernate will consider all attributes are mandatory by default. Website in Eatery. Class and Note in order dish class as not every restaurants has website and not every ordered dish has note.

Derived Attributes

Derived attributes can be implemented by using @Transient annotations and calculating method ,in this project, there is average rating as derived attribute which can be calculating from the list of ratings.

Association with Attribute and Bag/history association

There are reservation class with date and time and number of people attributes and orderdish class with quantity and note. Those classes exit between 2 related classes and has many to one. association with each related class. To implement those classes, @ManyToOne , @JoinColumn annotations are used. In the case of OrderDish class, it has many to one association with Order and Dish class. So @ManyToOne , @JoinColumn annotations are used in of OrderDish class and @OneToMany in order and dish class. The same implementation method is used for reservation.

Overlapping Inheritance and Dynamic Inheritance

To implement overlapping inheritance and dynamic, the flattening class hierarchy is used. All attributes and methods for subclasses are transferred to the base class which is Eatery in my case. Then, there is the Enum EateryType which has Café, Restaurant and Bar. In the base class, Enumset<EateryType> will be introduced. Getter and setter for each attribute will also be implemented and all the necessary validation will be checked. Moreover, the exception will be thrown if the methods which aren't not related to the indicated role is used.

The constructor will take Enumset<EateryType> and all the attributes as parameters. In the constructor, the check for current roles will be implemented which means the attributes of role will be assigned based on the current roles. As it also dynamic inheritance, the types of restaurant can be changed by using methods such as ChangeToCafe (),ChangeToRestaurant(), ChangeToBar() will be used.

Multi-Aspect Inheritance

Multi-Aspect inheritance is implemented using the normal inheritance for one aspect and the other aspect is implemented using the flattening class hierarchy. In my case, Pick up and Delivery extends the base class which is Order. For the other aspect, I created the Enum Order Type with Standard and Express. In the order class, the Enum<OrderType> will be implemented. It is similar to overlapping inheritance except the fact that the type can't be overlap. That's why Enum is used instead of EnumSet in this case.

Abstract class and Polymorphism

Employee class has the abstract method calculate income. The subclasses which are Waiter and cook extends from employee class. For waiter the income will be calculates by summing up salary and tips and by summing up salary and bonus for Cook.

Composition

Between Eatery and Menu class, composition association is created which means menu can't exit without eatery. @OneToMany, @ManyToOne annotations will be used in related class. As menu will be removed if the eatery is deleted, cascade = CascadeType.REMOVE will be used in @OneToMany.

XOR

In this project, cook can either have experience or has to join training but not both simultaneously. That means while one association exited, the other isn't allowed to exit. To ensure this, necessary validation will be implemented.