Online Restaurant

Introduction

- **Purpose: To provide an online platform for users to browse, order, and review meals from the restaurant.**
- **Target Audience: Customers looking for an easy way to order food online and the restaurant staff managing orders.**

Features

- **User Registration and Login: Allow users to create accounts and log in using email or social media accounts.**
- **Menu Browsing: Display the restaurant's menu with categories and detailed descriptions of each item.**
- **Order Placement: Enable users to add items to their cart and place orders for delivery or pickup.**
- **Payment Integration: Integrate with a payment gateway for secure online payments.**
- **Order Tracking: Provide real-time updates on the status of orders, from preparation to delivery.**
- **Admin Dashboard: Provide a dashboard for restaurant staff to manage orders, menu items, and user reviews.**

User Roles

- **Customer: Can browse the menu, place orders, track orders, and leave reviews.**
- **Admin: Manages menu items, reviews, and orders through the admin dashboard.**

Technology Stack

- **Frontend: Angular**
- **Backend: Java Spring Boot**
- **Database: Sequel Server**
- **Payment: Paypal**
- **Authentication: JWT (JSON Web Tokens)**

System Architecture

- **Client-Server Model: Frontend communicates with backend API to fetch data and place orders.**
- **Database Integration: Sql Server used to store user information, menu items, orders, and reviews.**
- **Payment Gateway: Integration with Paypal for processing online payments securely.**

User Interface

- **Homepage: Displays featured items, promotions, and menu categories.**
- **Cart Page: Shows items in the cart with the option to edit or remove items.**
- **Order Tracking: Provides real-time updates on order status and estimated delivery time.**

Functional Requirements

User Management:

Admin:

Can add, edit, and delete users.

Menu Management:

Admin:

Can add, edit, and delete menu items.

Can categorise menu items

Customers:

Can view the menu.

Can search for menu items.

Can filter menu items by category.

Ordering System:

Customers:

Can add items to their cart.

Can view order history.

Delivery Tracking:

Customers:

Can track the status of their delivery orders (e.g., out for delivery, delivered).

Can view estimated delivery time.

Integration with Third-party Services:

Payment Gateways:

Integrates with third-party payment gateways for secure online transactions.

Non-Functional Requirements

Performance:

The system is able to handle a large number of concurrent users

The system can respond to user interactions within 3 seconds.

Reliability:

The system is available 24/7, with minimal downtime for maintenance (less than 1 hour per month).

The system has backup and recovery mechanisms in place to prevent data loss.

Usability:

The user interface is intuitive and easy to navigate.

The product is easy to use on the first attempt by a member of the public without training.

Security:

The system encrypted sensitive data (e.g., user information, payment details) to protect against unauthorised access.

The system should have measures in place to prevent SQL injection and other common security threats.

Scalability:

The system is able to scale up to accommodate an increasing number of users and menu items.

The system is able to integrate with third-party services (e.g., payment gateways, delivery services) as needed.

Maintainability:

The system is modular and well-documented to facilitate future updates and maintenance.

The system has a version control system in place to track changes and revisions.

LOOK AND FEEL REQS:

The system uses only two colours.

SRS

Use Case Diagram

Use Case of online restaurant

Client
Admin

register
login
add admin
search item
edit users
update cart
edit items
place order
edit order status
track order
payment
logout
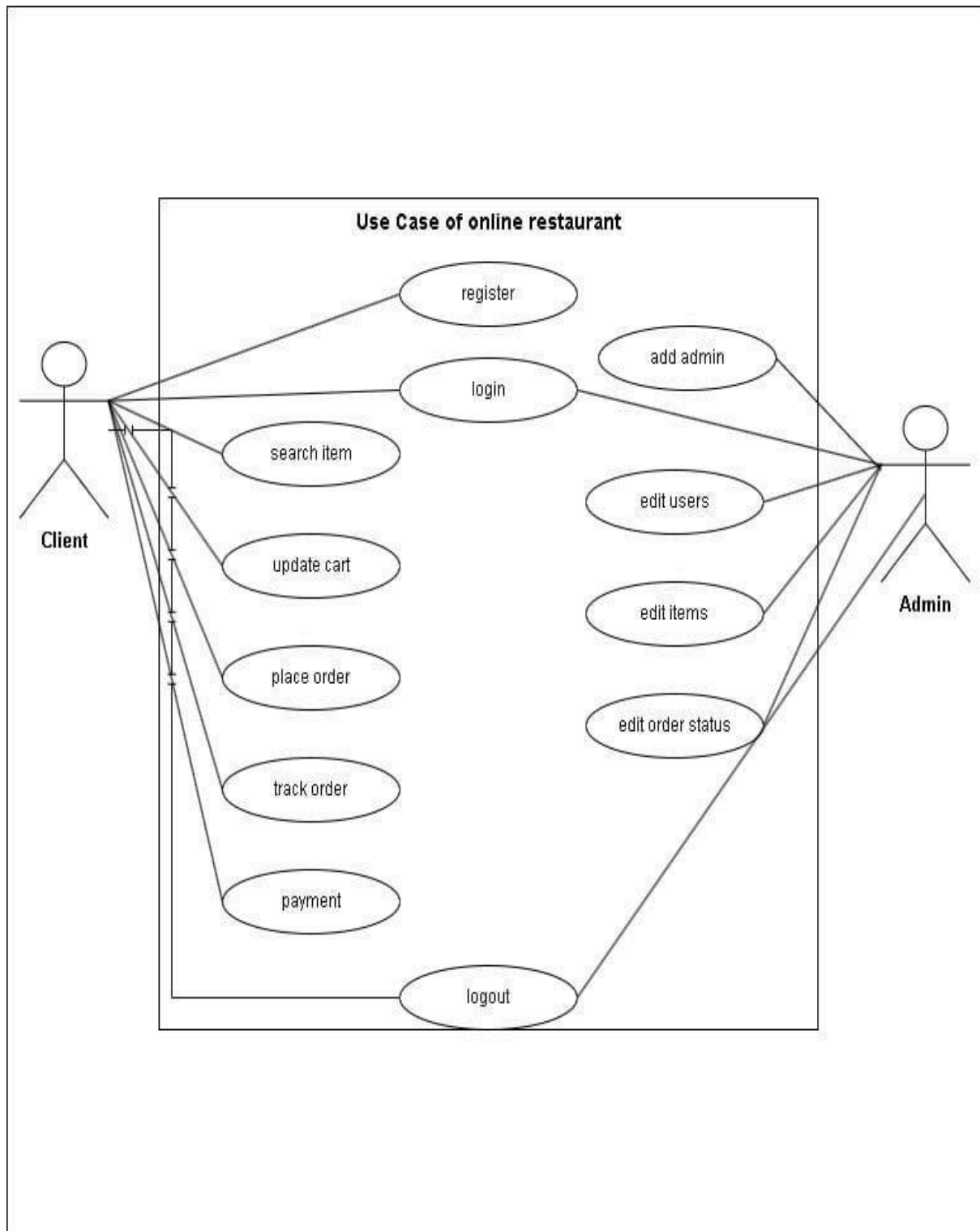
User Activity Diagram
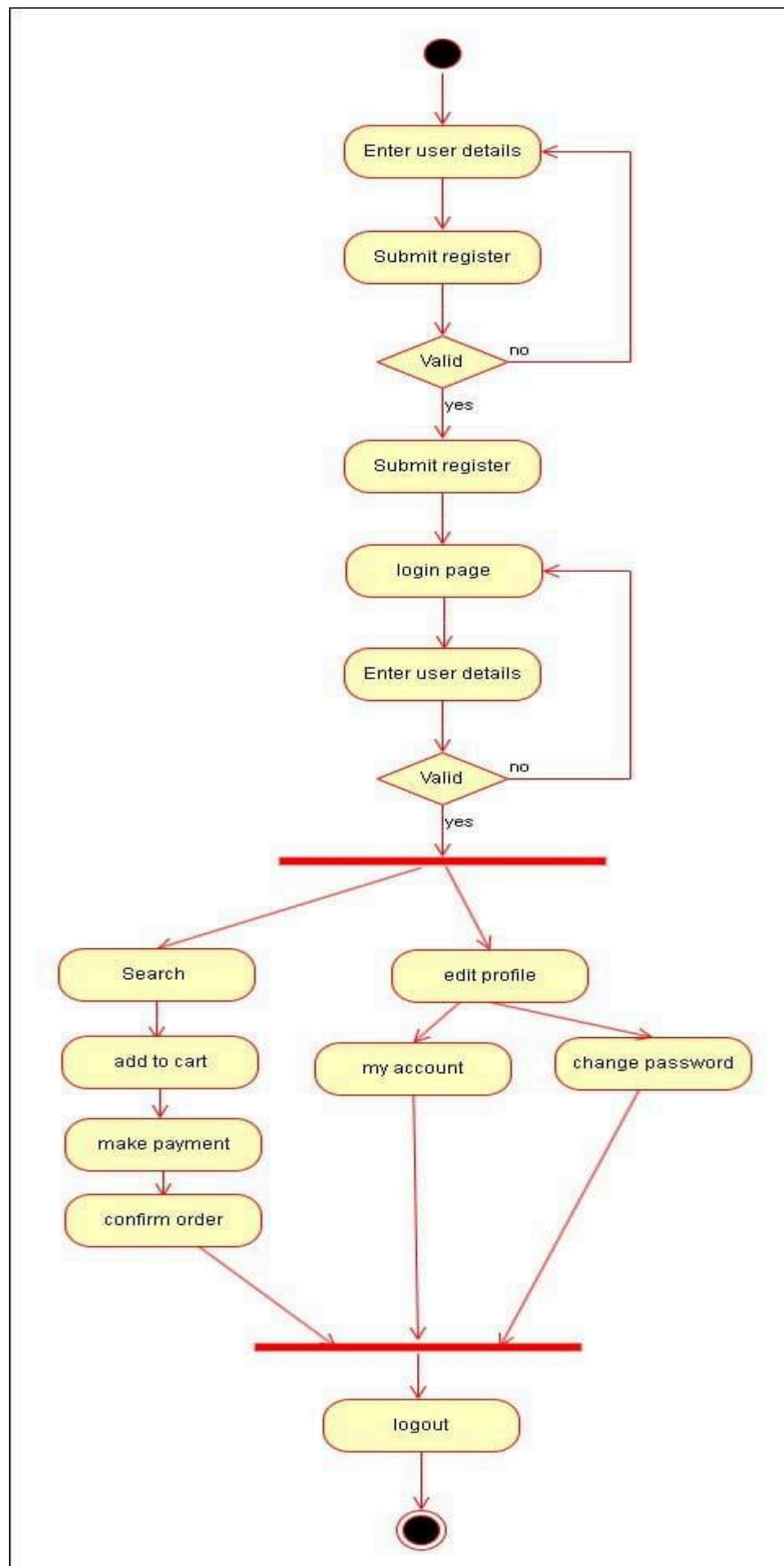
Admin Activity Diagram
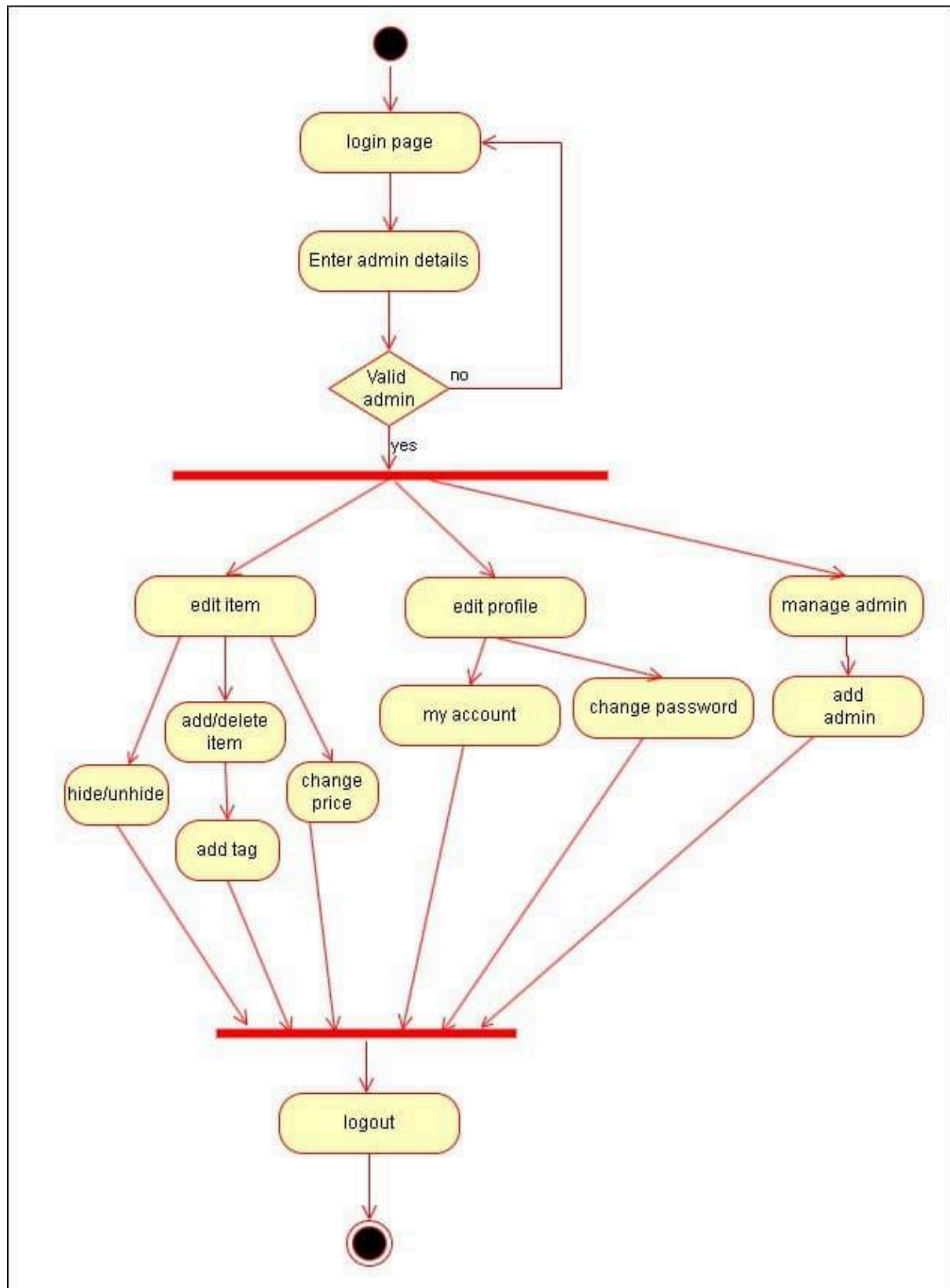
Sequence Diagrams

Login
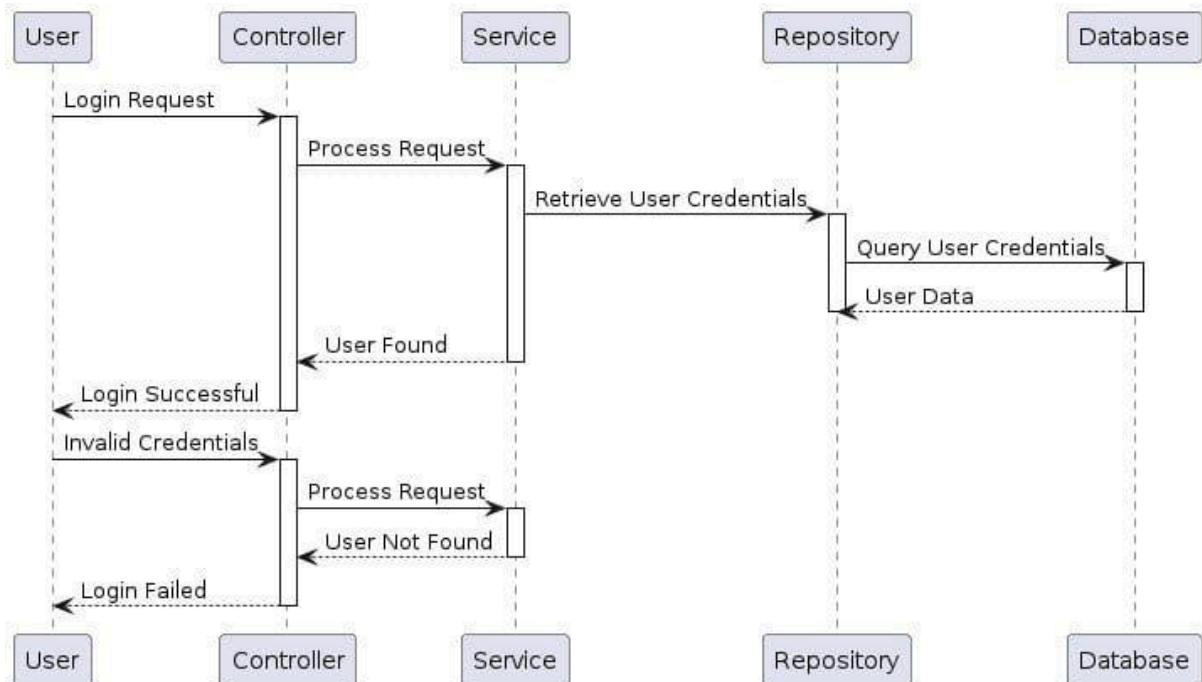
Register
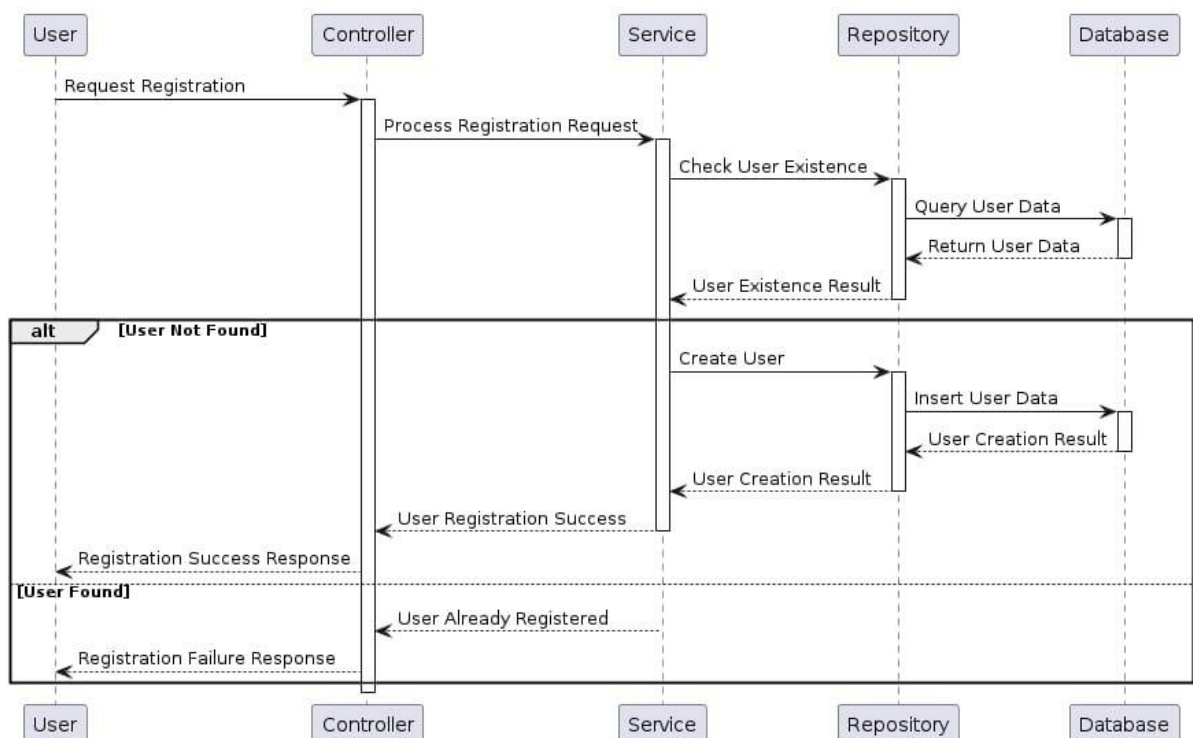
Administration
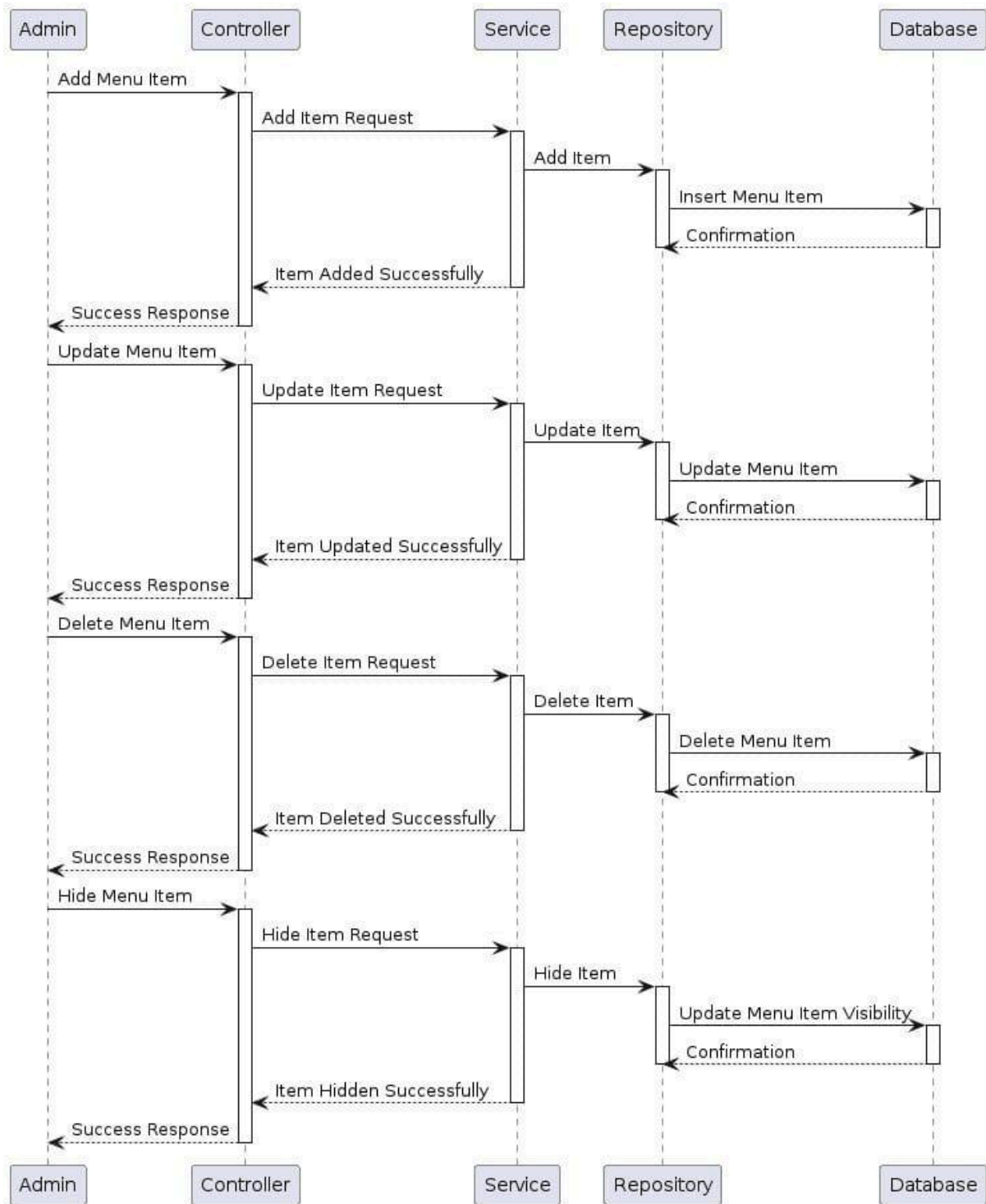
Update Information
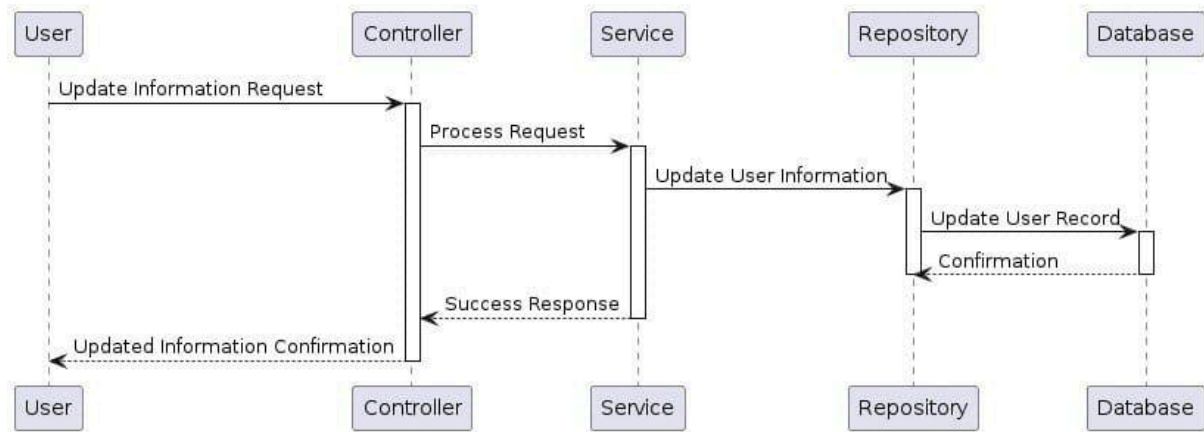
Search

ERD

Class Diagram

Client

-id: long
-firstName: String
-lastName: String
-address: String
-email: String
-password: String
-banned: boolean

+gettersAndSetters()
+saveClientWithoutChangingPass(client: Client): userDetails
+saveClient(client: Client): UserDetails
+banClient(client: Client): UserDetails
+unbanClient(client: Client): UserDetails
+listClients()
+getClientByID(id)
+deleteClientByID(id)

Authority

-id: long
-authName: String

+gettersAndSetters()
+getAuthorityById(id: long): Authority

Food

-id: long
-name: String
-price: Double
-imageUrl: String
-origins: List<String>
-cookTime: String
-hidden: boolean

+gettersAndSetters()
+save(Food food)
+findAllVisible(): List<Food>
+findByTags(tagNames)
+findAll(): List<Food>
+findByPrice(): String name
+hideFood(Food food)
+unhideFood(Food food)
+findByName(): String name

Tag

-name: String
-count: long

+gettersAndSetters()

Order

-totalPrice: long
-name: String
-address: String
-createdAt: Date
-updatedAt: Date

+gettersAndSetters()
+saveOrder(order: Order)
+getAllOrderByDate(String email)
+findOrdersById(id: long)
+cancelOrder(Order order)
+shipOrder(Order order)

OrderItem

-id: long
-price: Double
-quantity: int

+gettersAndSetters()

OrderResponseDTO

-id: long
-totalPrice: Double
-name: string
-address: string
-createdAt: Date
-updatedAt: Date
-paymentId

+toOrderDTO(orders: List<Order>)
+gettersAndSetters()

LatLng

-lat: String
-lng: String

+gettersAndSetters()

«enumeration»
OrderStatus

NEW
PAYED
SHIPPED
CANCELED
REFUNDED

OCL

Order

Order

- -id : Long
- -name : String
- -totalPrice : Long
- -address : String
- -status : String
- -createdAt : Date
- -updatedAt : Date

- +gettersAndSetters()
- +saveOrder(Order order)
- +getAllOrderByDate(String email)
- +cancelOrder(Order order)
- +findOrderById(Long orderId)
- +shipOrder(Order order)

context Order
inv: self.id <> null
inv: self.totalPrice >0
inv: self.status <> null
inv: self.createdAt <> null
inv: self.updatedAt >self.createdAt

context Order::saveOrder(order: Order)
pre: order.validOrderObject()
post: result = savedOrder() else result = null

context Order::getAllOrdersByDate(email: String)
post: result = sortedListAllOrders() else result = emptyList()

context Order::cancelOrder(order: Order)
pre: order <> null
post: result = true else result = false

context Order::findOrderById(orderId: Long)
pre: orderId.validLong()
post: result = orderById() else result = null

context Order::shipOrder(order: Order)
pre: order <> null
post: result = true else result = false

Food

Food

- -id : Long
- -name : String
- -price : Double
- -imageUrl : String
- origins : List<String>
- -cookTime : String
- -hidden : Boolean

- +settersAndGetters()
- +save(Food food)
- +findByName():String name
- +findByPrice() : String name
- +hideFood(Food food)
- +unhideFood(Food food)
- +findAllVisible():List<Food>
- +findAll(): List<food>
- +findByTags():List<Food>

context Food
inv: self.validString(name)

inv: self.price >0

context Food::save(food: Food)
pre: food.validFoodObject()
post: result = savedFood() else result = null

context Food::findByName(name: String)
pre: name.validString()
post: result = listFoundByName() else result = emptyList()

context Food::findByPrice(price: Double)
pre: price.validDouble()
post: result = listFoundByPrice() else result = emptyList()

context Food::hideFood(food: Food)
pre: food <> null
post: result = true else result = false

context Food::unhideFood(food: Food)
pre: food <> null
post: result = true else result = false

context Food::findAllVisible()
post: result = listAllVisible() else result = emptyList()

context Food::findAll()
post: result = sortedListAll() else result = emptyList()

context Food::findByTags(tags: List)
post: result = listFoundByTags() else result = emptyList()

Client

Class invariant

Client

- -id :Long
- -firstName : String
- -lastName : String
- -address : String
- -email : String
- -password : String
- -banned : Boolean

- +gettersAndSetters()
- +saveClient(client : Client) : UserDetails
- +saveClientWithoutchangingPass(client : Client) : UserDetails
- +banClient(client : Client) : UserDetails
- +unbanClient(client : Client) : UserDetails
- +listClients()
- +getClientById(id)
- +deleteClientByID(id)

context Client
inv : self.firstName <> null
inv : self.lastName <> null
inv : self.email <> null
inv : self.password <> null

context Client::saveClient(client: Client)
pre: client.notNullFields()

context Client::saveClientWithoutChangingPass(client: Client)
pre: client.notNullFields()

context Client::banClient(client: Client)
pre: client.notNullFields()

context Client::unbanClient(client: Client)
pre: client.notNullFields()

context Client::listAllClients()
pre: true

context Client::getClientById(id: String)
pre: id.validClientId()

context Client::deleteClientById(id: String)
pre: id.validClientId()

context Client::saveClient(client: Client)
post: result = null implies errorOccurred()

context Client::saveClientWithoutChangingPass(client: Client)
post: result = null implies errorOccurred()

context Client::banClient(client: Client)
post: result = true implies bannedSuccessfully() else not bannedSuccessfully()

context Client::unbanClient(client: Client)
post: result = true implies
unbannedSuccessfully() else not unbannedSuccessfully()

context Client::listAllClients()
post: result = listOfClients()

context Client::getClientById(id: String)
post: result = clientById() else result = null

context Client::deleteClientById(id: String)
post: result = true implies deletedSuccessfully() else not deletedSuccessfully()