**이룸오더**

**high-level Architecture**

**UML Diagrams**

<팀명: 이룸핑>

<팀원: 2022920028 박수빈

2022920050 이희진

2022920062 최진영

2022920040 양나슬

2022920016 김은지

2022920058 주영은>

변 경 이 력

|  |  |  |  |
| --- | --- | --- | --- |
| 버전 | 일자 | 변경 내역 | 작 성 자 |
| V1 | 2024-11-11 | 프론트엔드 엔티티 단위 클래스 다이어그램 추가 | 박수빈, 이희진, 최진영 |
| V1.1 | 2024-11-14 | 프론트엔드 패키지 단위 클래스 다이어그램 추가 | 박수빈, 이희진 |
| V1.2 | 2024-11-18 | 백엔드 클래스, 시퀀스 다이어그램 추가 및  설명 작성 | 주영은, 양나슬, 김은지 |
| V1.3 | 2024-11-18 | 프론트엔드 클린 아키텍처와 클래스 다이어그램 설명 작성 | 박수빈, 이희진 |
| V2.0 | 2024-12-10 | 프론트엔드 아키텍처 변경 및 다이어그램 수정 삽임 | 이희진 |
| V2.1 | 2024-12-10 | 프론트엔드 클래스 다이어그램 수정 삽임 | 이희진 |
| V2.2 | 2024-12-13 | 백엔드 클래스 다이어그램 수정 삽입 | 최진영 |
|  |  |  |  |

Table of Contents

[1. High-level Architecture 4](#_Toc182840105)

1.1 Frontend

1.2 Backend

[2. Class Diagram 4](#_Toc182840106)

2.1 Frontend

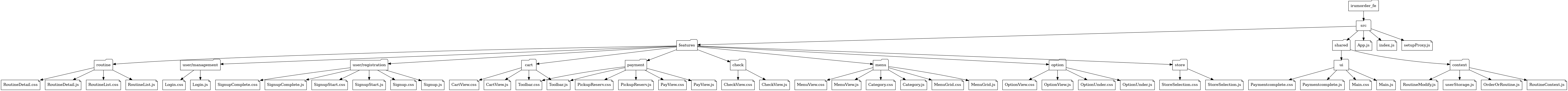
2.2 Backend

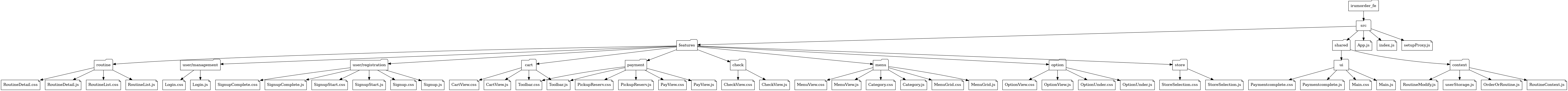
[3. Sequence Diagram 26](#_Toc182840107)

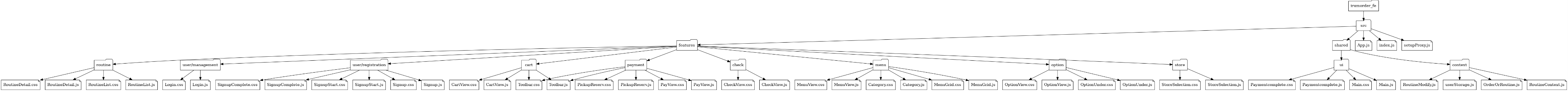
[4. style guide 29](#_Toc182840108)

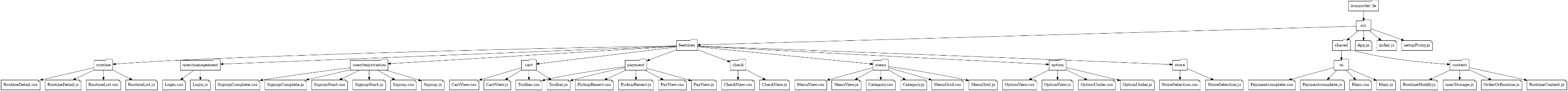
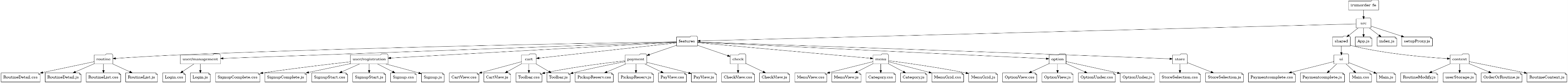
# High-level Architecture

* 1. Frontend - Feature-Based Directory Structure

 ->

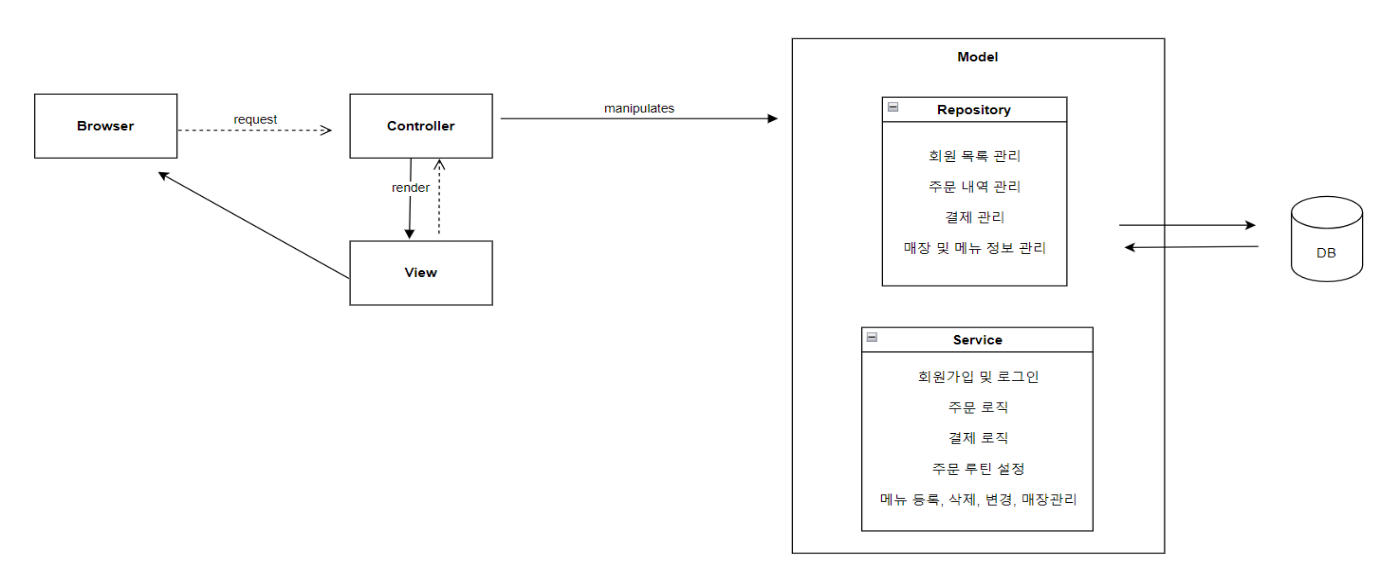
 ->

 ->



프론트엔드는 각 기능 단위의 독립성을 유지하기 위해 Feature-Based Directory Structure를 지향하고자한다. 이 구조는 애플리케이션의 주요 기능별로 파일을 그룹화하여 기능 단위의 모듈화와 독립성을 극대화한다. 각 기능은 features 디렉터리 내에 위치하며, 관련 컴포넌트, 스타일, 로직 등을 하나의 폴더로 묶어 관리한다. 공통적으로 사용되는 컨텍스트, 유틸리티, 공용 UI 컴포넌트는 shared 디렉터리에 배치하여 재사용성을 높였다. 구조적 독립성은 기능 단위로 캡슐화되며, 기능 간 의존성을 최소화한다. 필요한 경우 공통 코드를 참조할 수 있도록 설계되었다. 이를 통해 각 기능의 수정이나 확장 시, 다른 기능에 영향을 주지 않으면서 독립적으로 유지보수와 개발이 가능하다. 새로운 기능 추가 시, 관련 파일을 하나의 폴더로 구성하면 되기 때문에 코드의 가독성과 관리 효율성이 높아진다. 하지만, 현재 프론트엔드 구조는 빠른 프로토타입 개발을 위해서 Monolithic 구조를 포함하고 있다. 이 구조는 유지보수성이 어렵기 때문에 추후 이 프로젝트를 더 디밸롭할시 Feature-Based Directory Structure를 지향한다.

1.2 Backend - MVC architecture

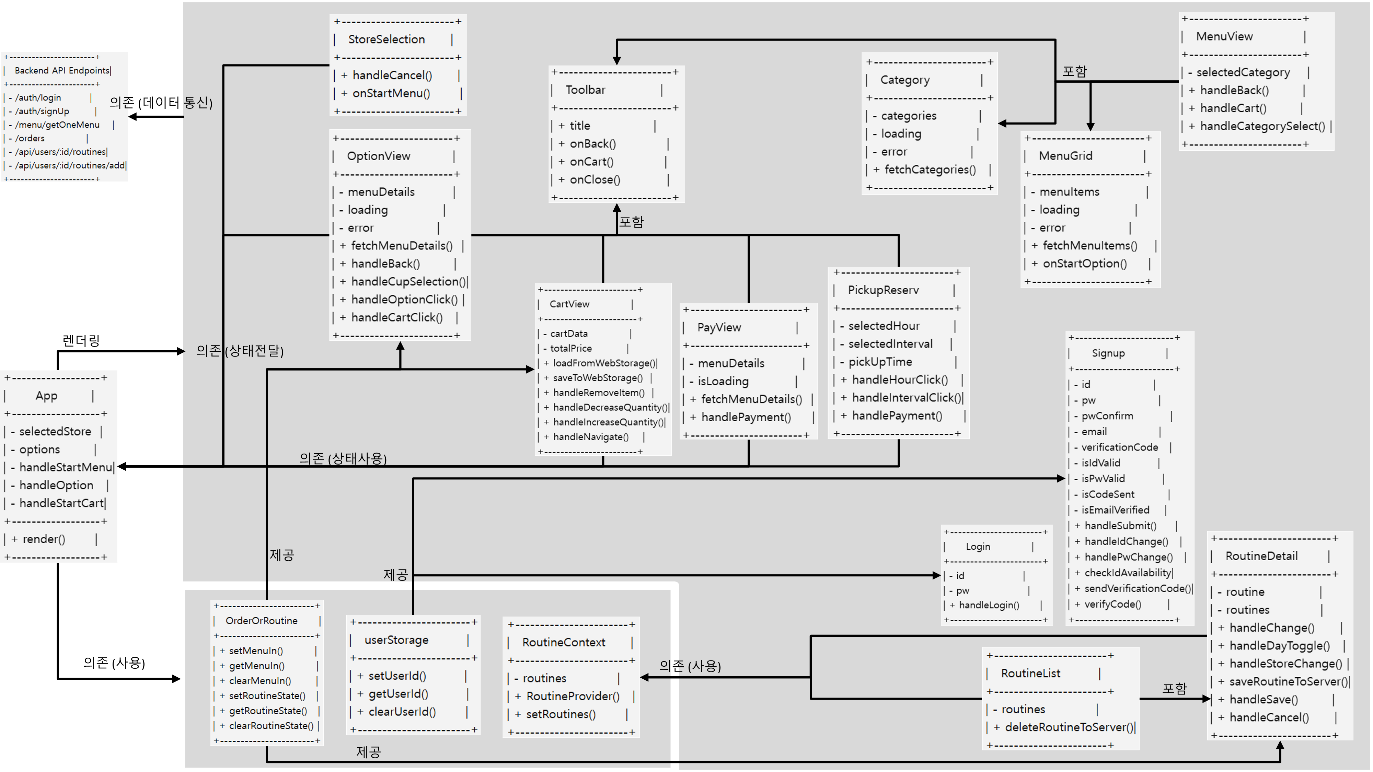


백엔드는 MVC 구조를 사용하여 구현하였다. Browser에서 Controller로 요청을 보내고, Controller는 browser, Model과 상호작용하여 데이터를 조작한다. Repository에서 DB의 데이터 조회, 저장, 수정 작업을 관리한다. Service에서 비즈니스 로직을 담당하여 처리한다.

MVC 구조를 사용하여 각 구성 요소 간의 책임을 명확히 나누어 코드를 모듈화하고, 애플리케이션의 유지보수성과 확장성을 향상시킨다.

# Class Diagram

* 1. Frontend



모듈 단위로 엔티티를 패키지로 묶어 관계를 표현했다.

App

애플리케이션의 루트 컴포넌트로, 모든 주요 기능의 라우팅과 상태 관리를 담당한다.

selectedStore와 options 상태를 관리하고, 화면 전환 핸들러를 통해 사용자 상호작용에 따라 페이지를 이동한다.

Login

기능: 사용자 로그인 화면을 제공하며, 사용자 입력(id, pw)을 서버로 전송하여 인증을 처리한다.

Signup

회원가입 화면 컴포넌트

아이디 중복 확인, 이메일 인증 및 비밀번호 확인 로직을 처리한다.

SignupComplete

회원가입 완료 후 성공 메시지와 다음 동작 안내를 표시

SignupStart

기능: 회원가입 시작 화면으로, 회원가입 프로세스의 초기 화면

Main

기능: 로그인 성공 후 사용자 대시보드 제공

사용자 이름(nickname)을 표시하고 주요 기능 버튼을 제공한다

MenuView

메뉴 선택 화면

선택된 카테고리(selectedCategory)에 따라 메뉴를 필터링하고 표시한다.

OptionView

선택된 메뉴의 옵션 설정 화면

사용자 선택에 따라 옵션 데이터를 업데이트하고 장바구니로 전달한다.

CartView

장바구니 화면 컴포넌트

로컬 스토리지에서 데이터를 로드 및 저장하며 항목 추가, 삭제, 수량 변경 등 장바구니 관리 기능을 제공한다.

PayView

결제 화면

사용자 주문 정보를 확인 및 결제 요청을 처리한다.

PickupReserv

픽업 시간 예약 화면

사용자 선택에 따라 예약 시간을 설정한다.

Toolbar

화면 상단의 내비게이션 컴포넌트

뒤로가기, 장바구니 열기, 닫기 등의 동작을 제공한다.

Category

메뉴 카테고리 표시

사용자 선택에 따라 메뉴를 필터링한다.

MenuGrid

메뉴를 격자 형태로 표시

선택된 카테고리에 따라 메뉴를 보여준다.

RoutineDetail

루틴 세부 정보를 설정하고 저장

루틴 요일, 시간, 메뉴 설정 기능을 제공한다.

RoutineList

사용자의 모든 루틴 목록을 표시

루틴 추가 및 삭제 기능을 제공한다.

RoutineContext

루틴 데이터를 관리하기 위한 React Context

루틴 목록을 전역적으로 공유한다.

StoreSelection

매장 선택 화면

사용자가 주문할 매장을 선택하도록 지원한다.

Paymentcomplete

기능: 결제 완료 후 성공 메시지와 다음 동작 안내를 제공

userStorage

사용자 ID 관리

ID 저장, 조회, 초기화 기능을 제공한다.

OrderOrRoutine

주문 및 루틴 상태 관리

현재 메뉴 ID와 루틴 상태를 저장 및 초기화한다.

Backend API Endpoints

/auth/login: 사용자 로그인 처리

/auth/signUp: 신규 사용자 회원가입

/menu/getOneMenu: 메뉴 세부 정보 조회

/orders: 주문 생성 및 결제 처리

/api/users/:id/routines: 루틴 목록 조회 및 수정

/api/users/:id/routines/add: 신규 루틴 추가

* 1. Backend

백엔드 통합 다이어그램

스케치, 도표, 그림, 평면도이(가) 표시된 사진

자동 생성된 설명

2.2.1 class\_auth

텍스트, 스크린샷, 폰트, 평행이(가) 표시된 사진

자동 생성된 설명

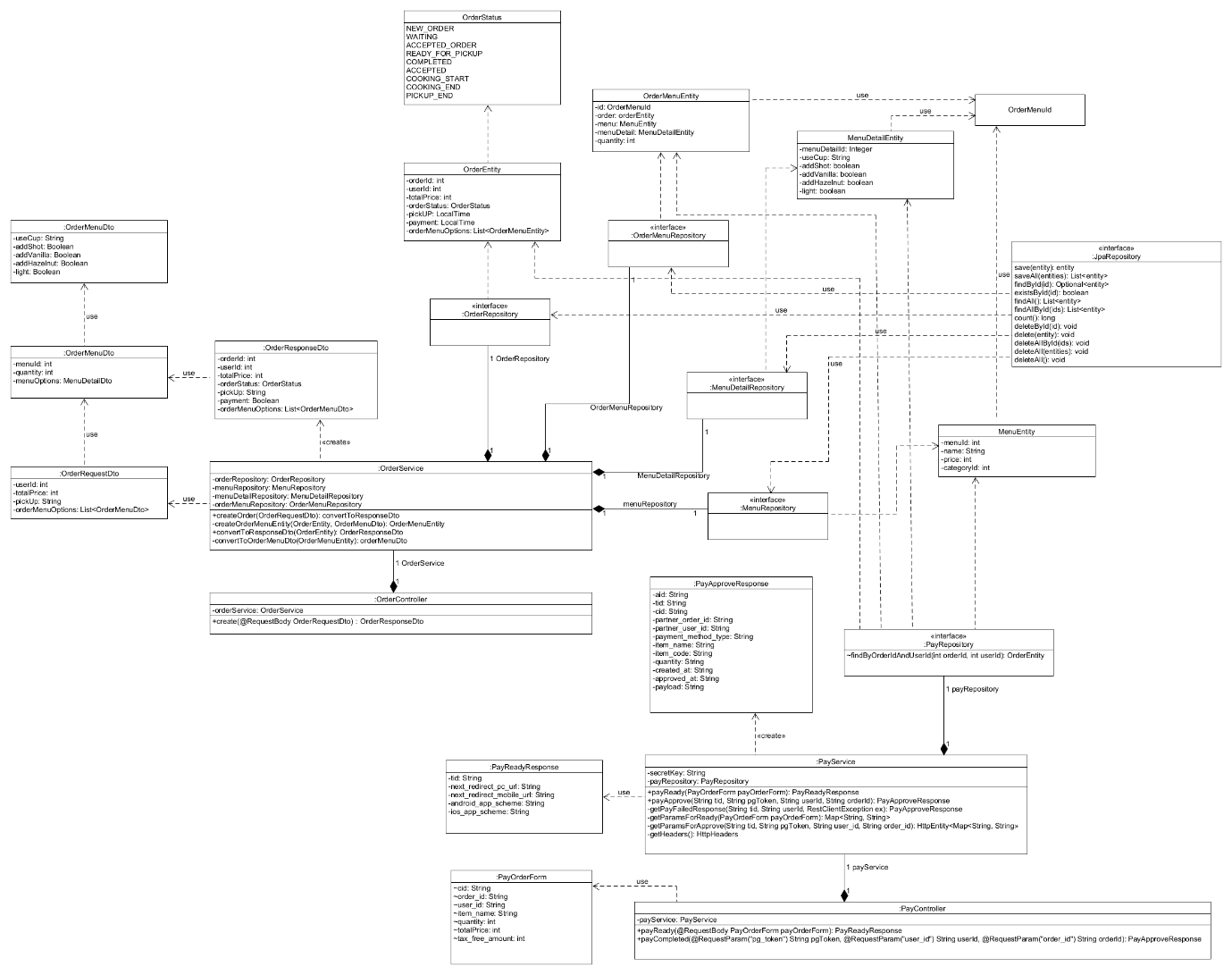
SignUp

아이디, 비밀번호, 이메일을 입력하고 validator를 통해 검증을 거친 후 회원 정보를 DB에 저장하면 회원등록이 완료된다.

Login

회원, 직원, 매장관리자가 아이디와 비밀번호 입력을 통해 로그인한다.

2.2.2 class\_orderPay



OrderPay : 주문과 결제기능을 수행한다.

Order

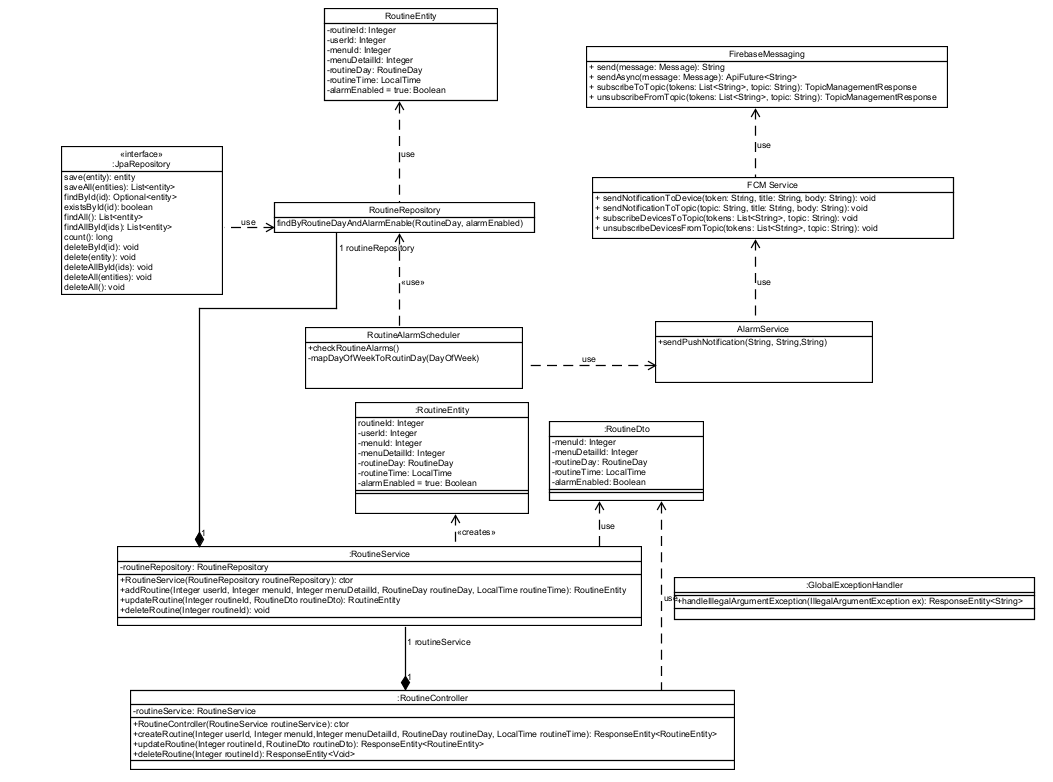
OrderRequestDto에서 받아온 정보들을 OrderEntity, OrderMenuEntity, MenuDetailEntity에 각각 저장해 db에 저장하고 OrderResponseDto를 반환함

Pay

PayController의 payReady 메서드에서 결제를 진행하기 위한 정보를 받고, 이를 PayService의 payReady 메서드로 전송한다. PayService에서 카카오페이 결제 url을 받아 클라이언트에 리턴한다.

클라이언트가 리턴받은 링크에서 결제를 완료한 후 얻은 pg\_token과 tid를 이용하여 결제확인을 수행한다(PayController의 payApprove 메서드, PayService의 payApprove 메서드드). 결제확인이 완료되면 DB에 결제승인 시각을 작성하고(payRepository 사용) 클라이언트에 결제승인 정보를 전송한다.

2.2.3 class\_routineAndAlam



Routine설정 및 Routine 알람기능

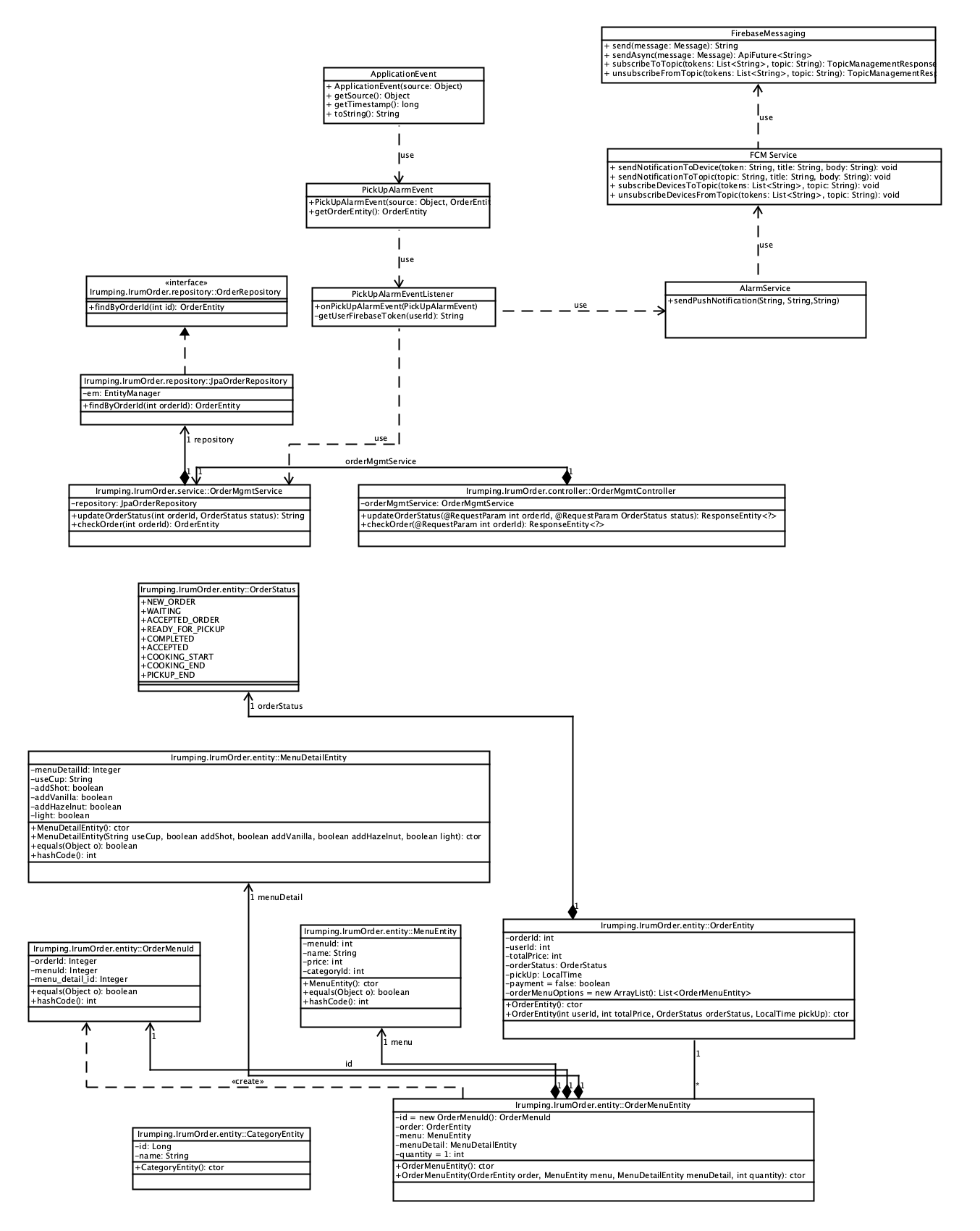
RoutineController로 루틴 추가, 수정, 삭제 요청이 들어온다.

이후 RoutineService 클래스에서 routineRepository를 통해 루틴을 DB에 저장/수정/삭제한다.

Routine Alarm

Scheduler를 이용해 10분마다 디비를 조회하여 픽업 예약시간 2시간전에 user들에게 루틴에 따라 Firebase를 이용하여 주문예약 팝업 알람을 보낸다.

2.2.4 class\_pickUpAlarm+orderManagement



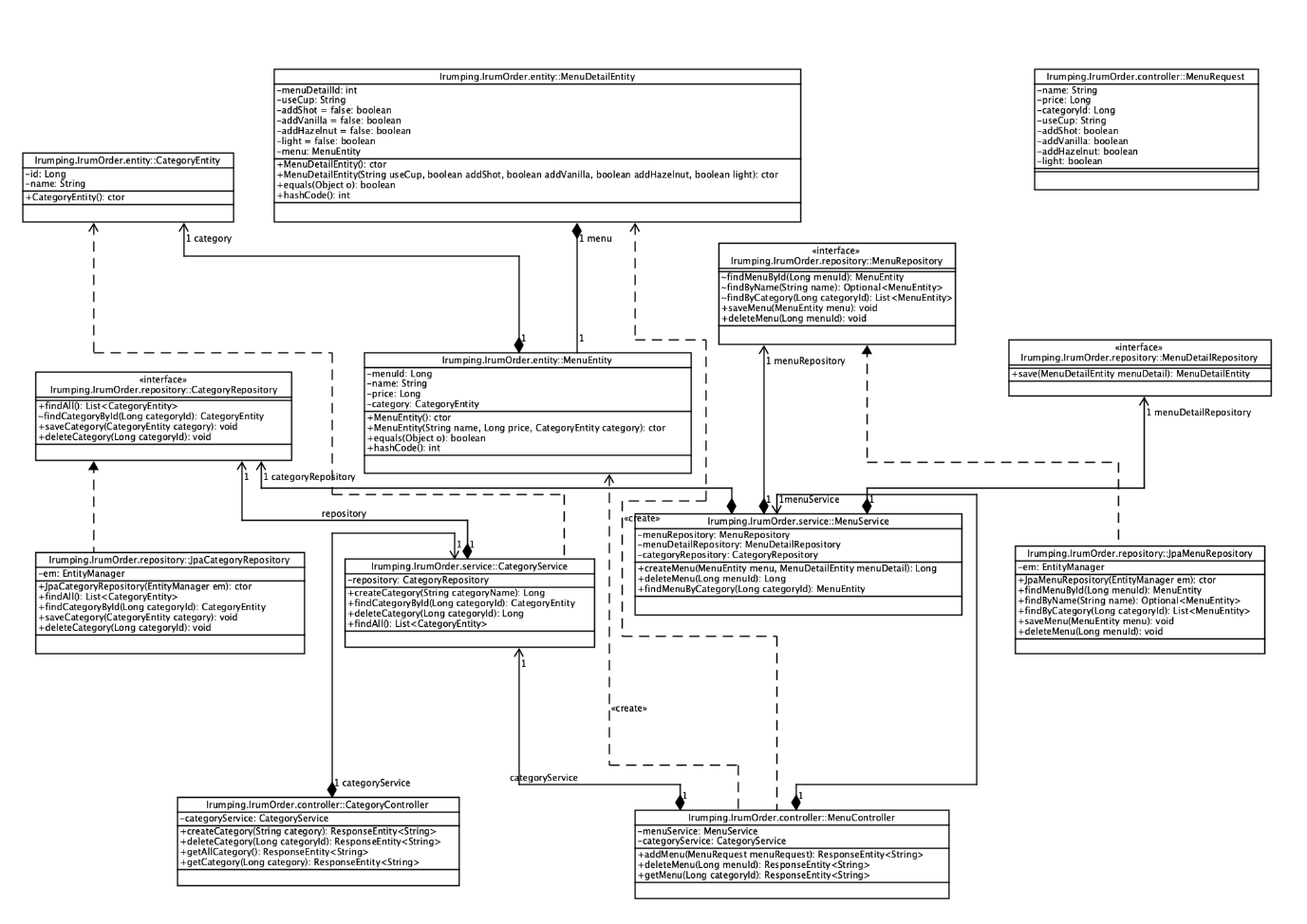
Order 상태 관리

직원이 상태 변경을 요청하면 DB에 변경된 상태를 저장한다.

Pick Up Alarm

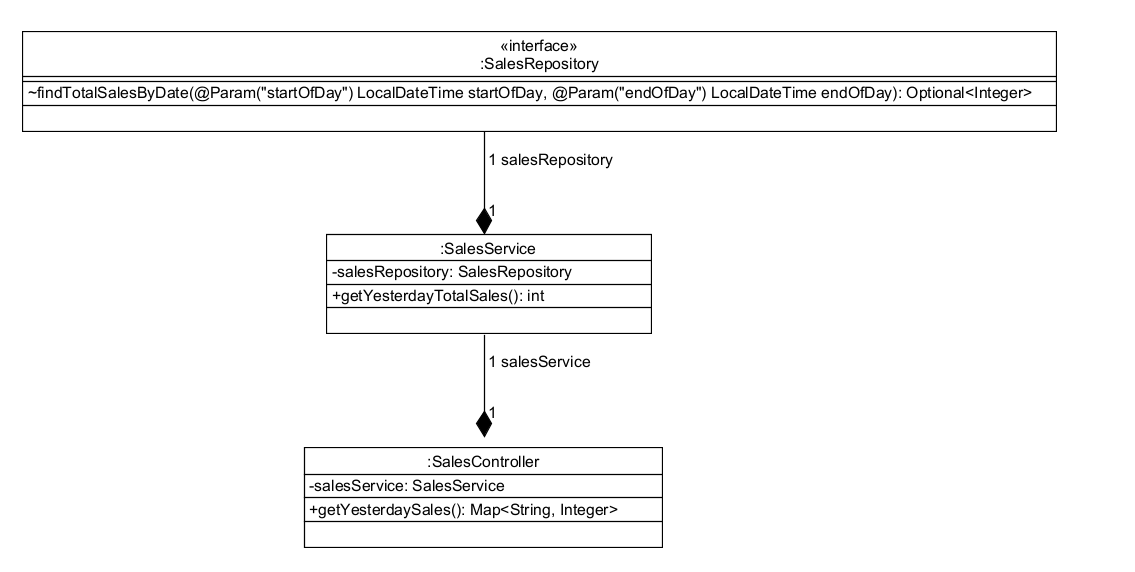
order가 PICKUP\_END 상태가 되면 회원에게 알림을 보낸다.

2.2.5 class\_menu



매장 관리자는 카테고리를 등록, 삭제할 수 있고, 각 카테고리에 대해서 메뉴를 등록, 삭제할 수 있다.

2.2.6 class\_salesCheck

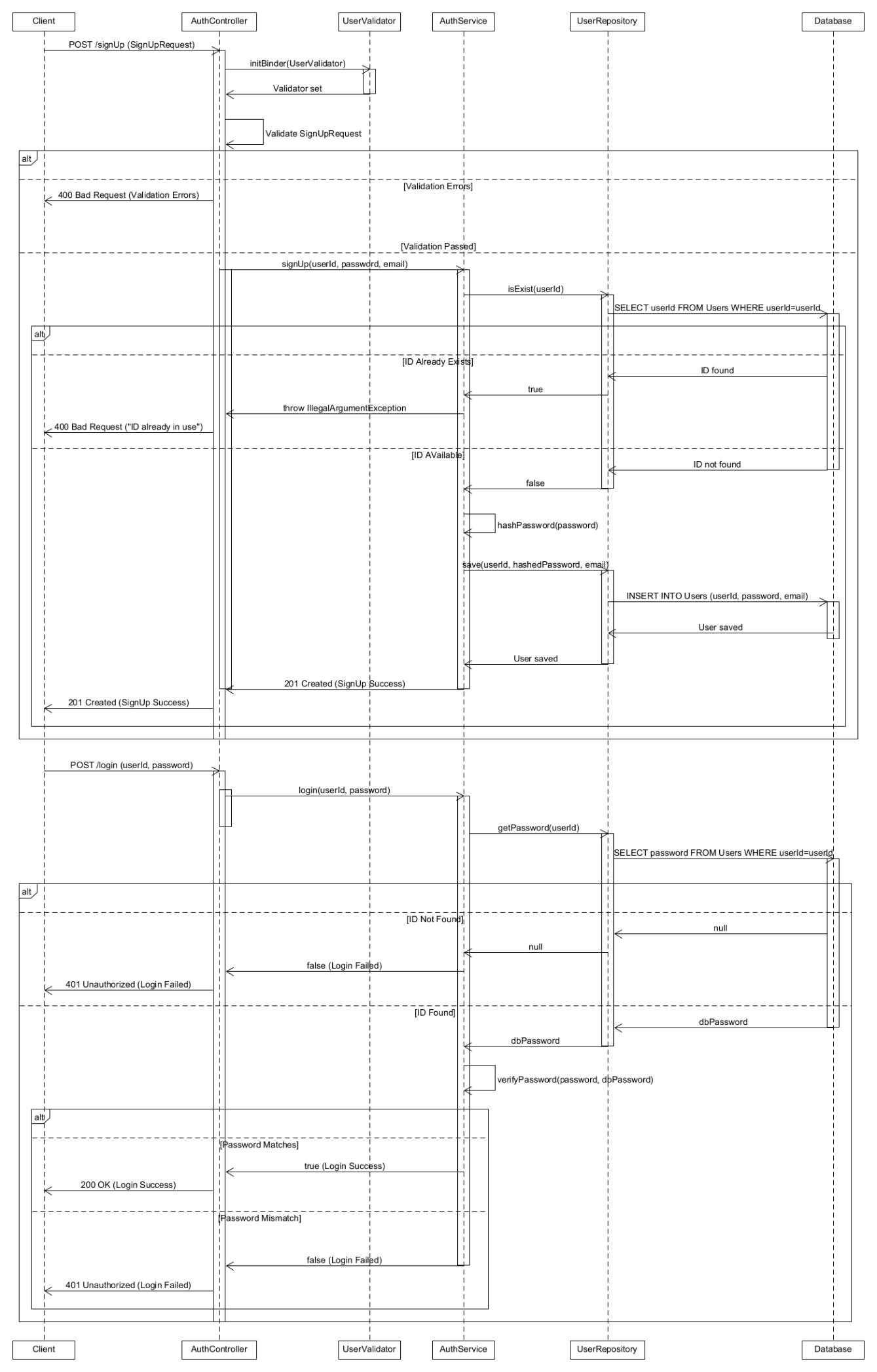


SalesController로 작일 매출정보 열람 요청이 들어오면 SalesService의 getYesterdayTotalSales를 호출하고, salesRepository의 findTotalSalesByDate

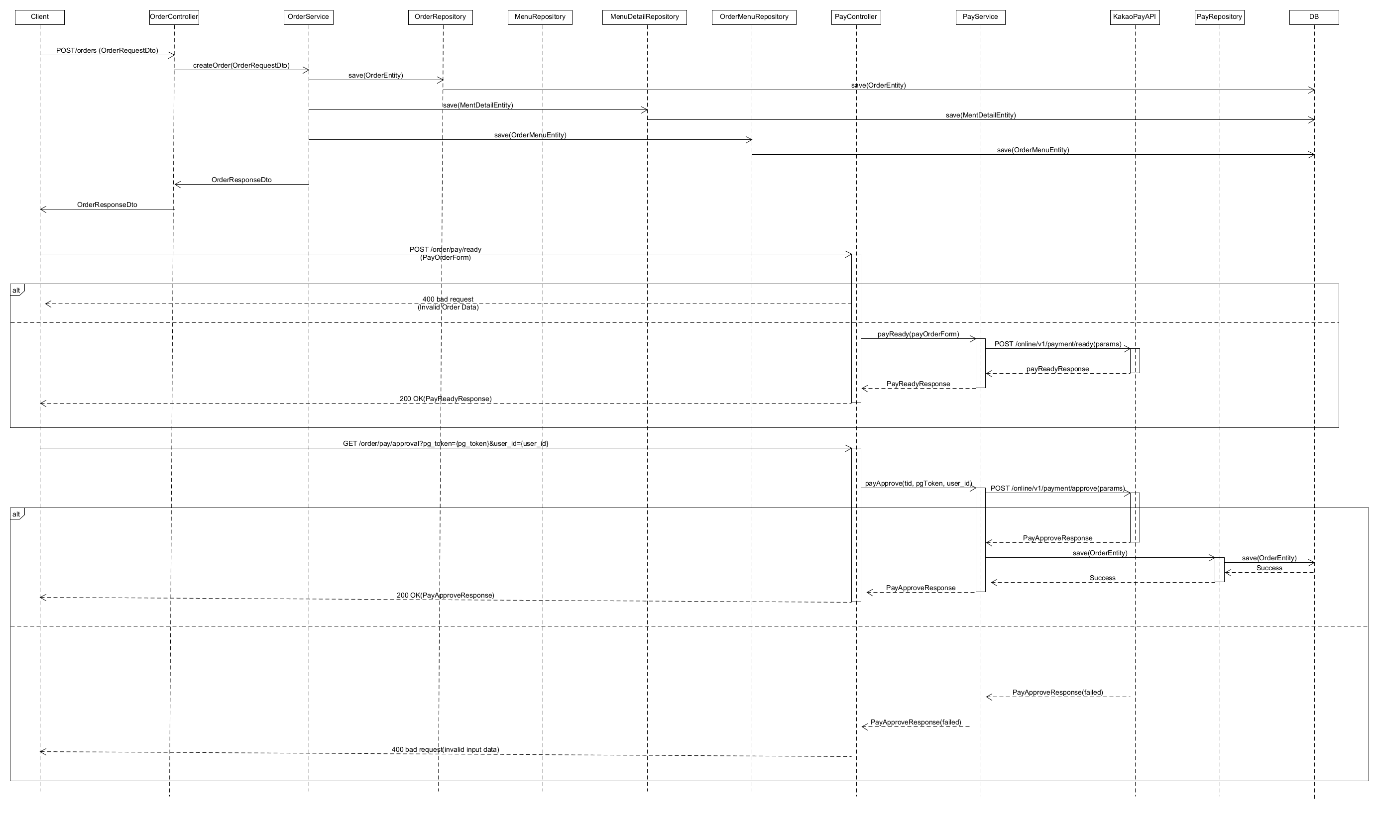
메서드를 통해 작일의 총매출 정보를 DB에서 가져온다. 이후 정보를 클라이언트에 전송한다.

# Sequence Diagram

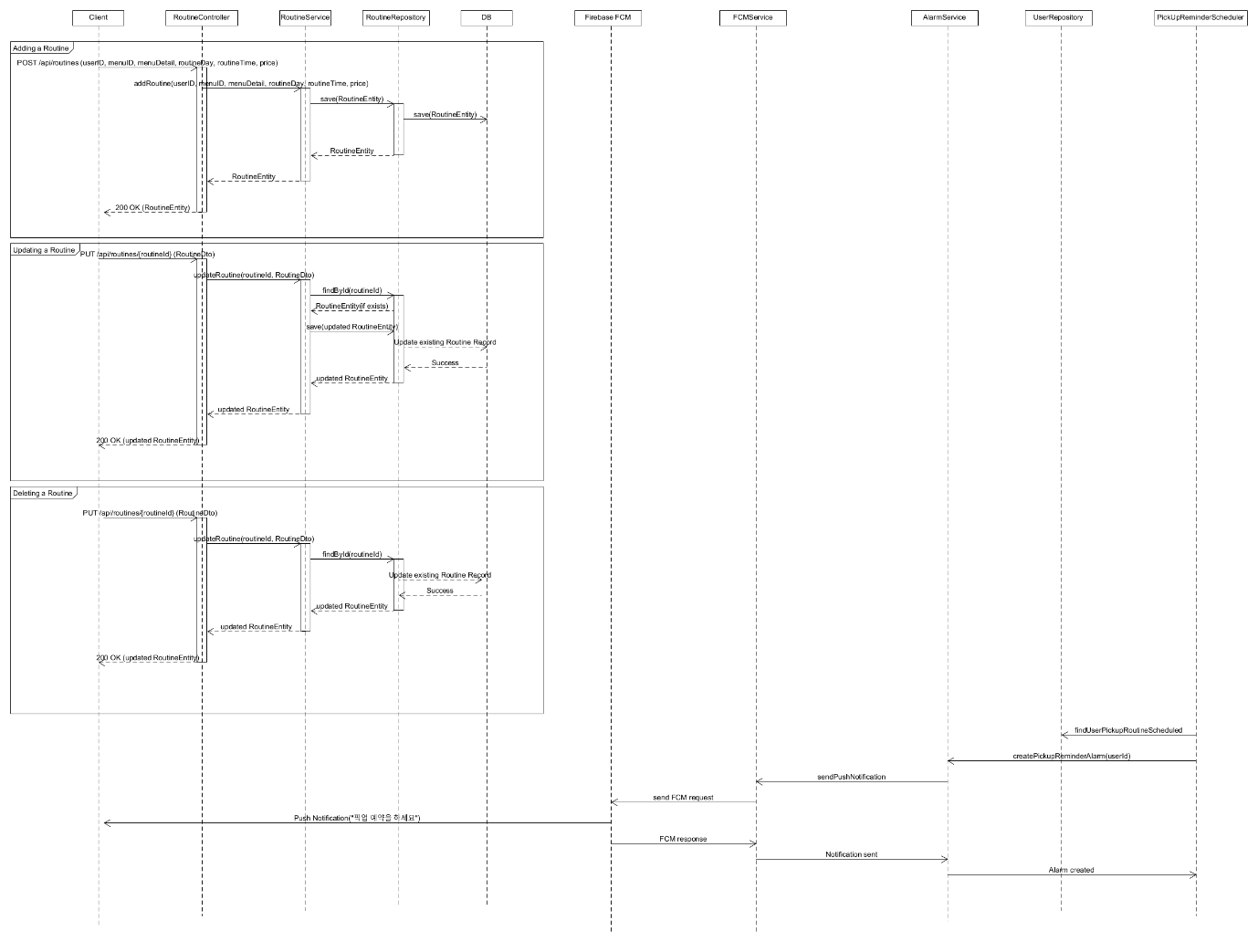
3.1 sequence\_auth



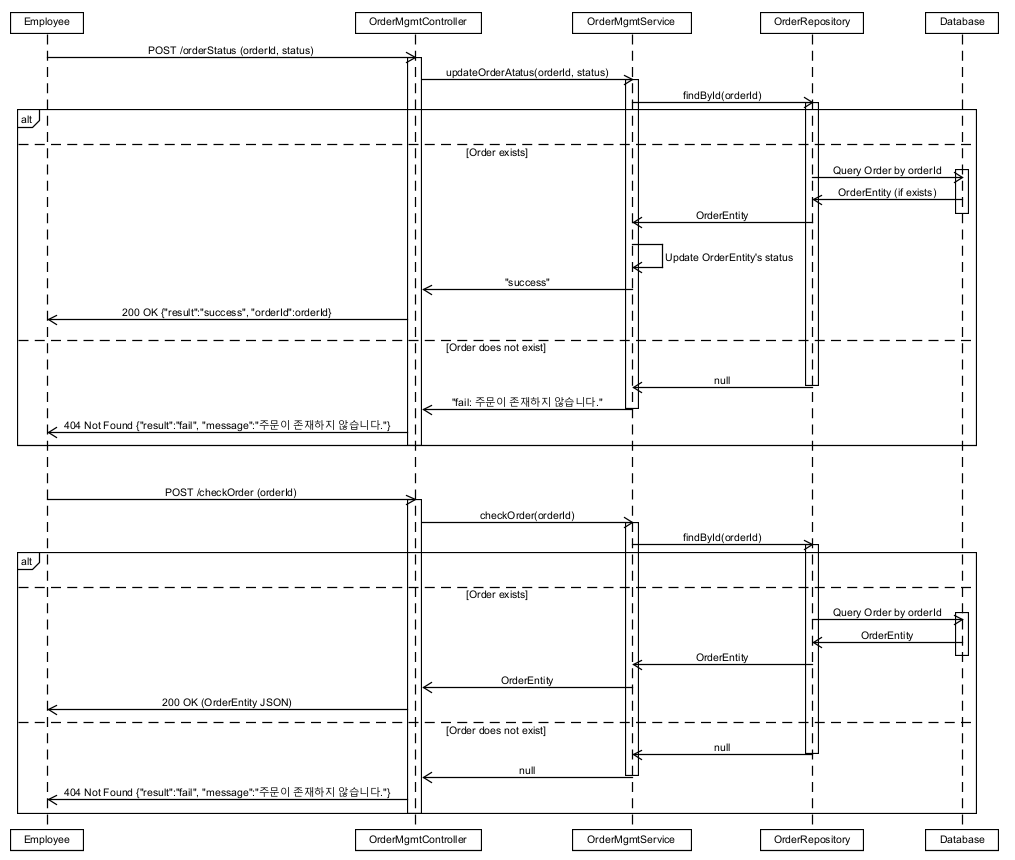
3.2 sequence\_orderPay



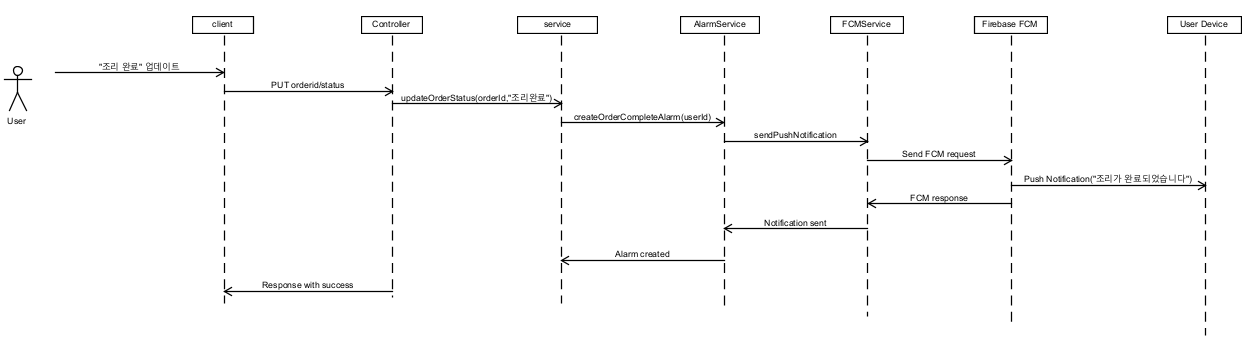
3.3 sequence\_routineAndAlarm



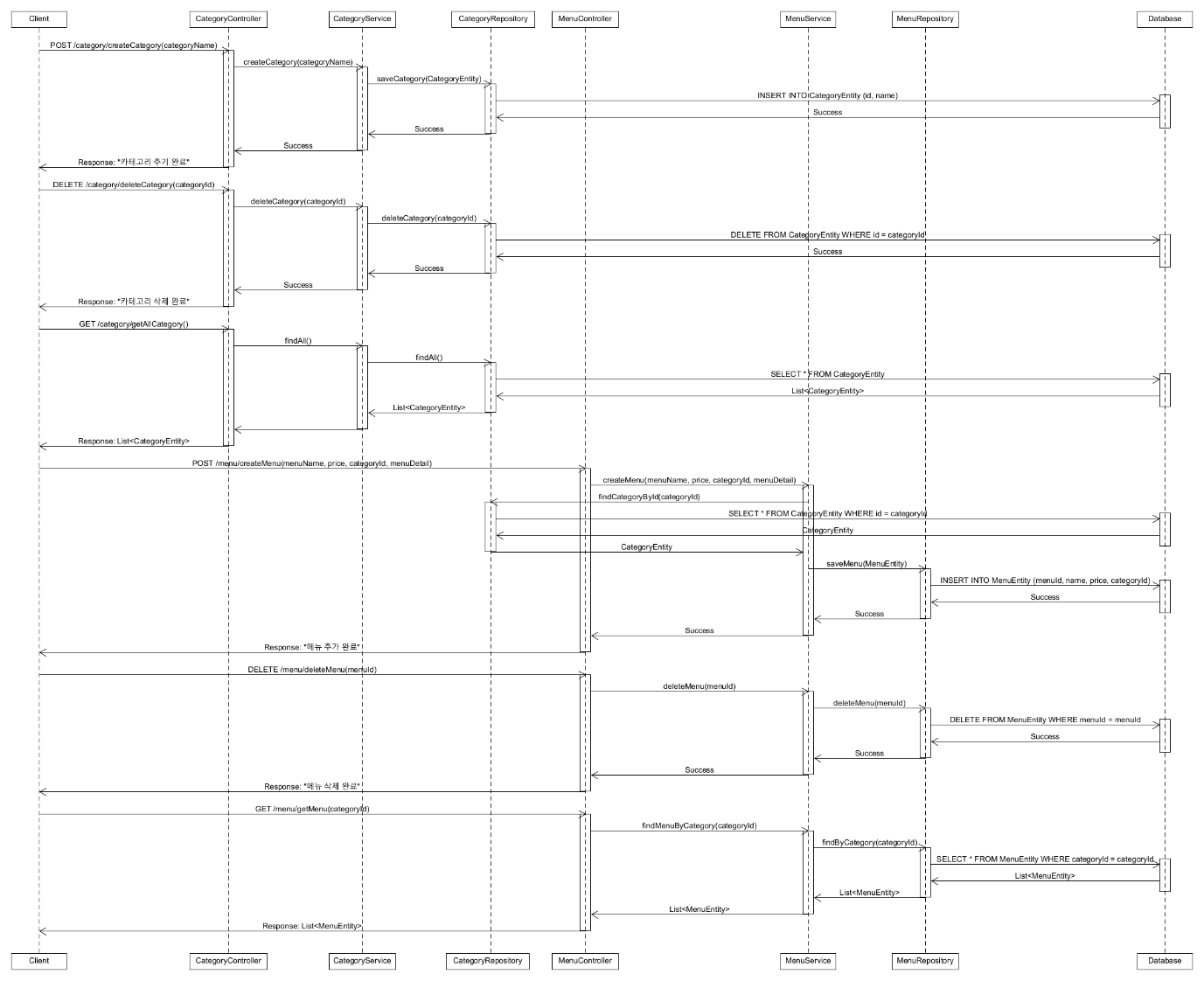
3.4 sequence\_orderManagement



3.5 sequence\_pickUpAlarm



3.6 sequence\_menu



3.7 sequence\_salesCheck

