High-level Architecture Document – UOSense

(시대생 맛집 지도)

2019920034 이도권
2019920015 김세윤
2019920018 김준호
2019920045 장규민
2021930028 최수아

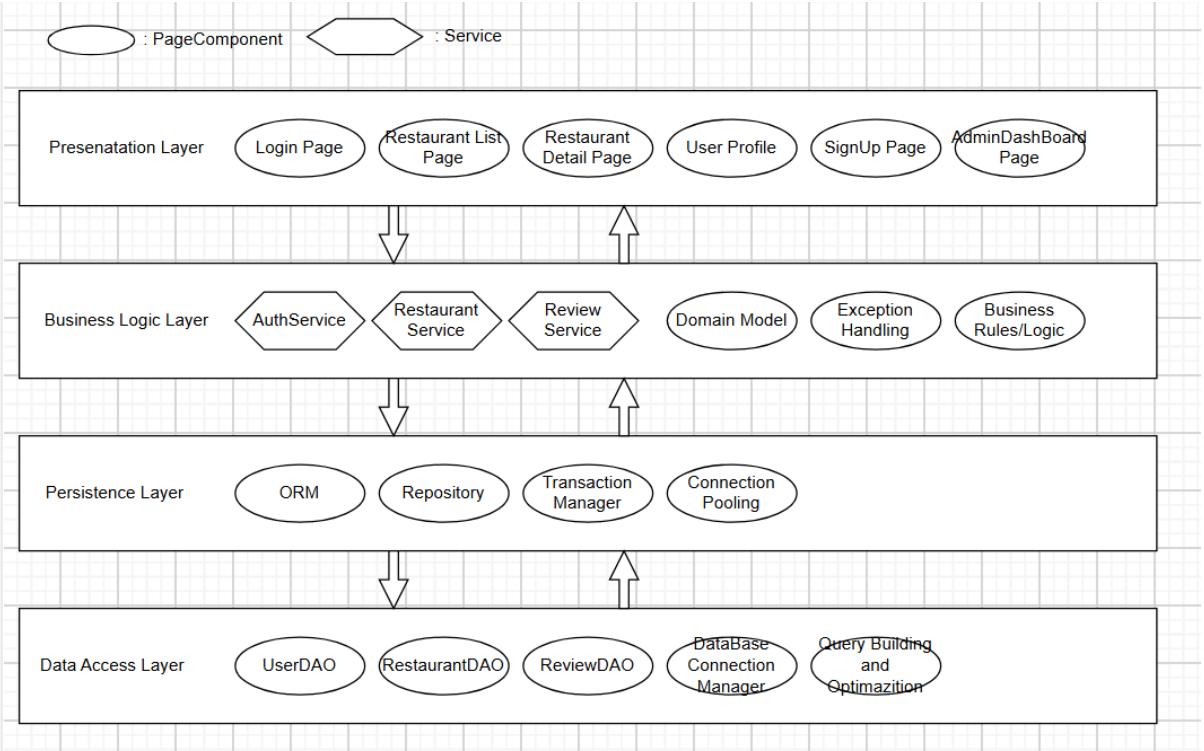
목차

Revision History(변경 이력)

버전	일자	변경 내역	작 성 자
Ver 1.0	**2024/11/17**	**Initial version**	김세윤
Ver 2.0	**2024/12/14**	실제 프로젝트와 동기화	최수아, 김준호

High-level Architecture UML Diagrams Document

1. High-level Architecture
   1.1 Layered Architecture (FRONT)



1.1.1 상세 내용

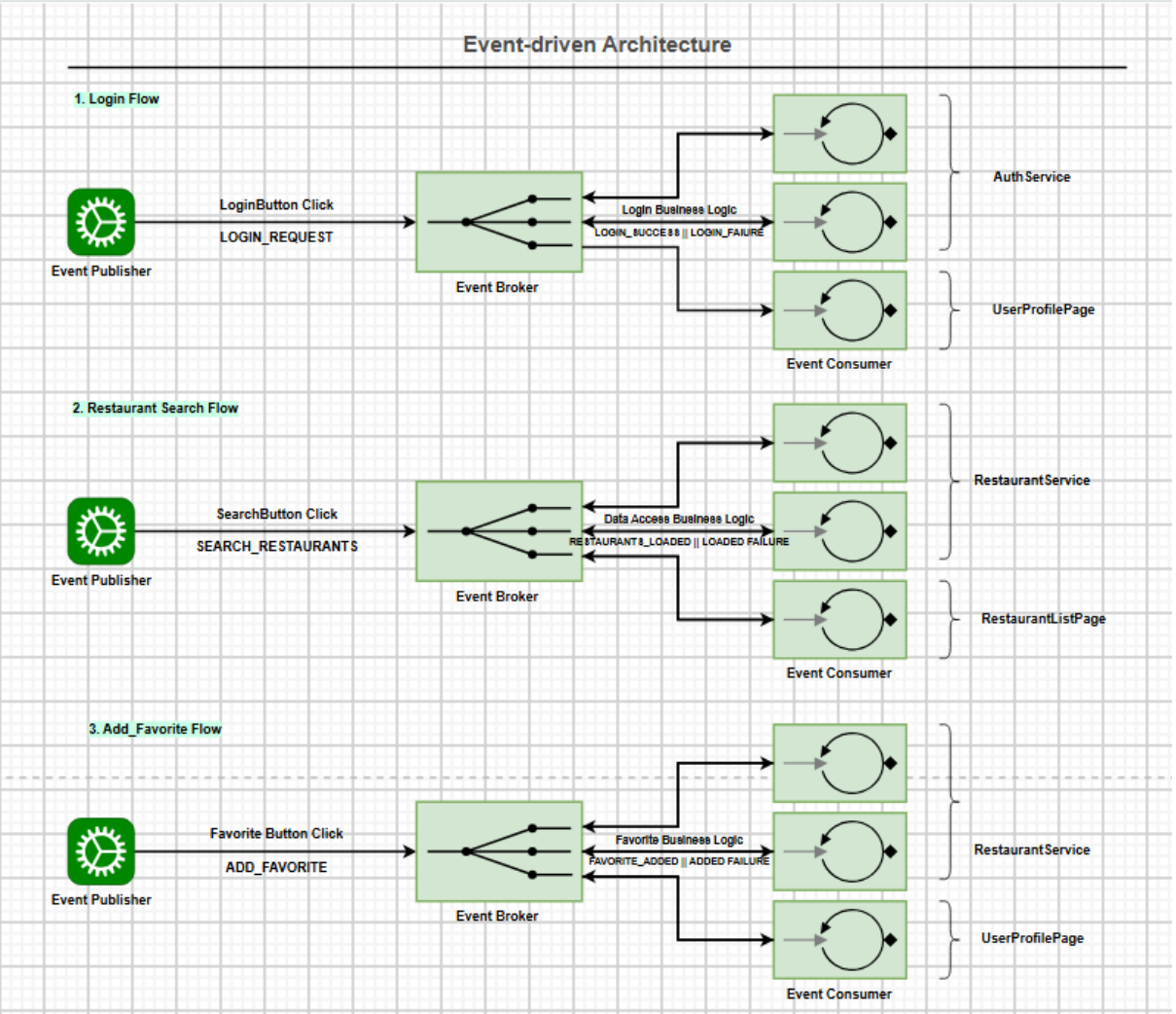
- Presentation Layer: 사용자와 직접 상호작용하는 레이어로, 왼쪽에서부터 각각 로그인, 식당 목록, 식당 세부 정보, 회원가입, 사용자 정보, 관리자의 행동을 보여주는 페이지 컴포넌트로 구성되어 있음.

- Business Logic Layer: 비즈니스 로직 레이어로, 실제 도메인 로직을 포함하여 시스템 내 주요 비즈니스 규칙을 처리하는 레이어임. 구성 요소로는 비즈니스 규칙을 처리하는 핵심 구성 요소인 서비스 클래스(AuthService, Restaurant Service, Review Service), 앱의 핵심 엔터티(User, Restaurant, Review, Favorite)를 정의하는 도메인 모델, 비즈니스 로직에서 발생할 수 있는 오류를 처리하는 예외 처리, 서비스 클래스에서 구현되는 로직인 비즈니스 규칙/논리(리뷰 작성 규칙 등)가 있음.

- Persistence Layer: 데이터의 영속성을 관리하는 레이어로 구성 요소로는 객체와 DB 간의 매핑을 관리하는 라이브러리인 ORM, 비즈니스 로직과 DAO 간의 중간 레이어로 특정 도메인에 맞는 데이터 접근을 제공하는 Repository, 데이터 일관성을 위해 여러 작업을 하나의 트랜잭션으로 처리하는 Transaction Management, 데이터베이스 연결을 효율적으로 관리하여 성능을 높이는 Connection Pooling이 있음.

- Data Access Layer: 데이터와 비즈니스 로직 레이어 간의 안전하고 효율적인 통신을 보장하는 레이어로, 구성 요소로는 DB의 CRUD(생성, 읽기, 수정, 삭제)작업을 담당하는 객체인 DAO(UserDAO, RestaurantDAO, ReviewDAO), DB연결을 관리하고 최적화하는 Database Connection Management, 특정 비즈니스 요구에 맞는 쿼리를 작성하고 성능을 최적화하는 Query Building and Optimization이 있음.

1.2 Event-driven Architecture



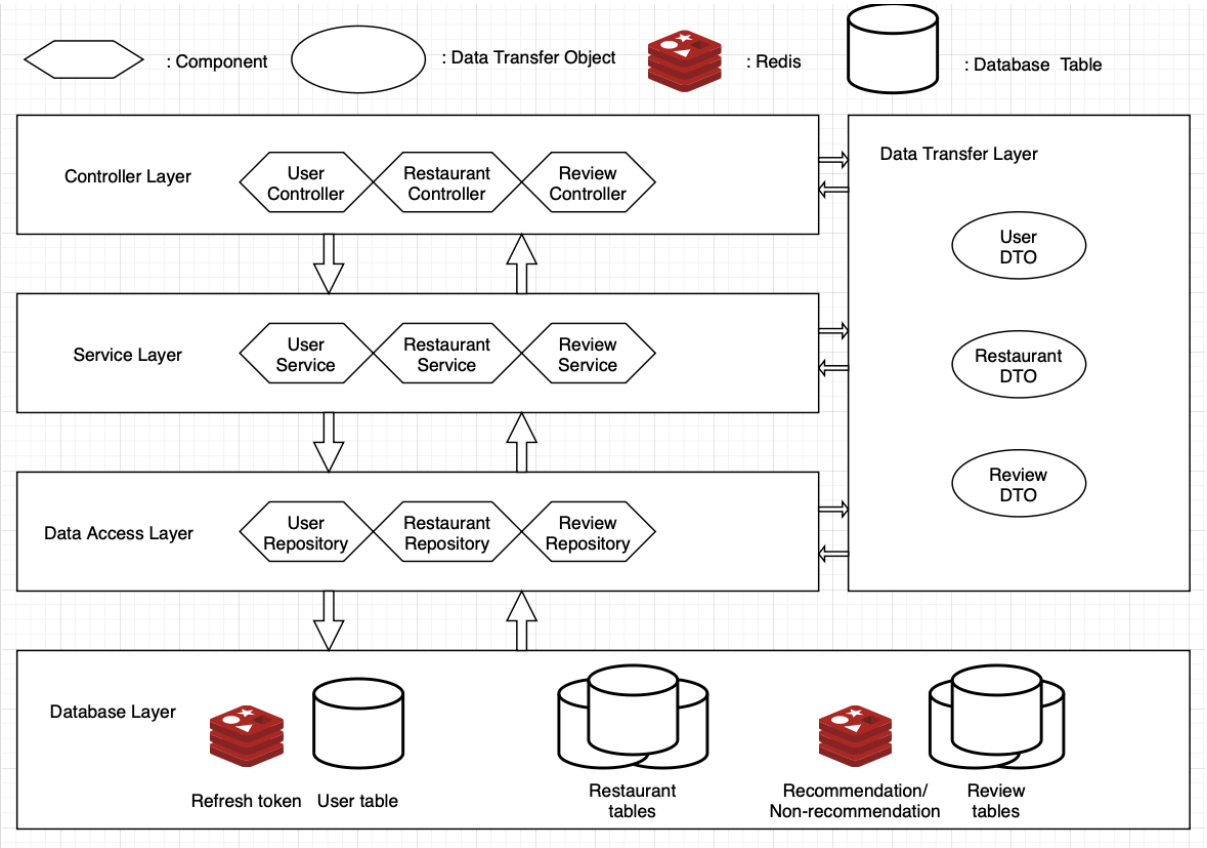
1.2.1 상세 내용

- Login Flow: 로그인 과정으로, 로그인 버튼을 클릭하면 로그인 요청 이벤트가 생성되어 브로커에게 전송되고, 브로커는 이 이벤트를 수신한 뒤 해당 이벤트를 구독 중인 인증 서비스에 전달, 인증 서비스는 비즈니스 로직을 처리하고 로그인 성공 여부에 따라 로그인 성공 또는 로그인 실패 이벤트를 새로 생성하여 브로커로 전달, 이때, 유저 프로필 페이지와 같은 추가 소비자가 이 이벤트를 받아 필요한 작업을 수행

- Restaurant Search Flow: 맛집 검색 과정으로, 사용자가 검색을 시작하면 레스토랑 검색 이벤트가 생성되어 브로커에 전달, 브로커는 이 이벤트를 레스토랑 서비스에 전달하며, 레스토랑 서비스는 데이터 접근 성공 여부에 따라 결과 이벤트를 생성하여 브로커에 전달, 이 검색 이벤트의 추가 소비자는 레스토랑 리스트 페이지로, 해당 페이지는 결과를 받아 사용자에게 보여줌.

- Add_Favorite Flow: 즐겨찾기 추가 과정으로, 사용자가 즐겨찾기에 특정 레스토랑을 추가하면 즐겨찾기 추가 이벤트가 생성되어 브로커에 전달, 브로커는 이를 즐겨찾기 서비스에 전달, 서비스는 추가 성공 여부에 따라 결과 이벤트를 다시 브로커에 전달함.

1.3 Layered Architecture (BACK)



1.3.1 상세 내용

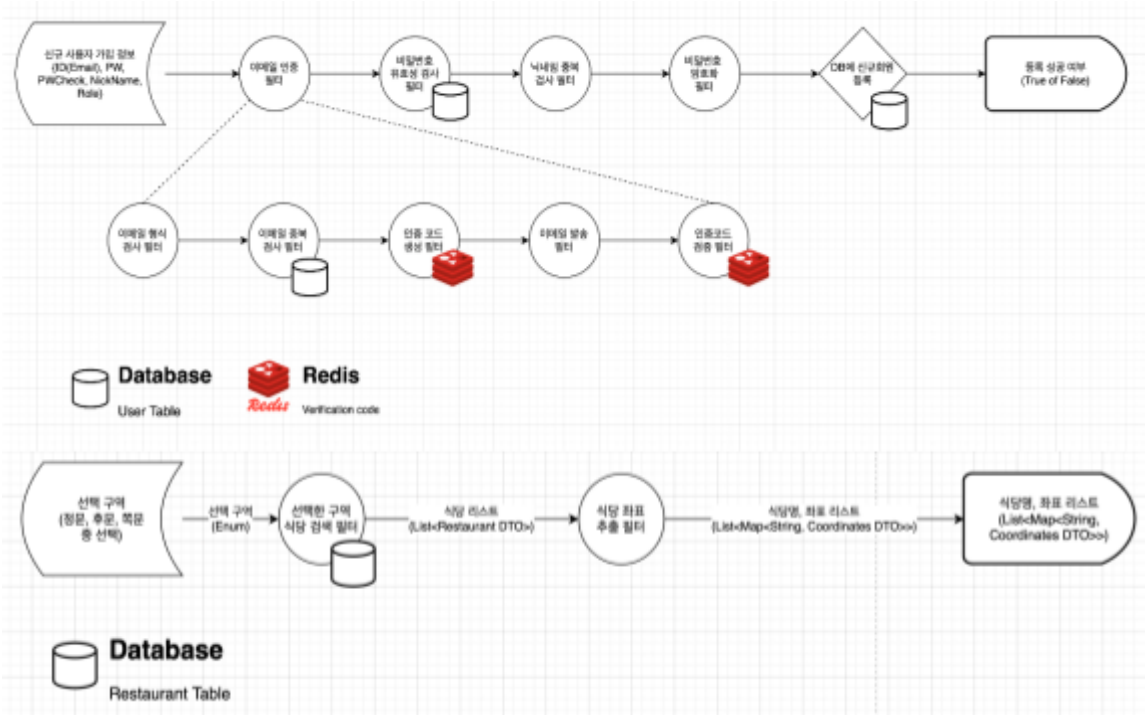
- Controller Layer: User, Restaurant, Review 세 가지 컨트롤러가 있고, 이 레이어는 사용자 요청을 처리하고 필요한 데이터를 서비스 레이어에 전달.

- Service Layer: 각 컨트롤러에 해당하는 UserService, RestaurantService, ReviewService가 있으며, 비즈니스 로직을 수행하고 필요한 데이터를 Data Access Layer에서 받아와 컨트롤러에 전달.

- Data Access Layer: DB와 직접 연결된 Repository들이 포함되며, 이 레이어의 UserRepository, RestaurantRepository, ReviewRepository는 DB와의 CRUD 작업을 수행.

- Data Transfer Layer: 이 레이어는 UserDTO, RestaurantDTO, ReviewDTO가 위치하며, 각 레이어 간 데이터가 안전하고 일관되게 전달되도록 중개 역할을 함.

- DataBase Layer: Redis와 MySQL을 사용하여 구성한 레이어로, User Table, Restaurant Table, Review Table과 같은 테이블들이 있으며, Refresh Token, Recommendation/Non-Recommendation과 같은 데이터를 Redis에 저장하여 빠른 액세스를 지원함.



1.4 Pipe-filter Architecture

1.4.1 상세 내용

- 두 입력 소스가 입력될 때 상황을 나타냈고 두 상황이 존재

- 입력 소스가 신규 사용자 가입 정보인 상황: 이메일 인증 과정을 압축한 이메일 인증 필터를 거치고, 이후 비밀번호 유효성 검사, 닉네임 중복 검사, 비밀번호 암호화 필터를 거치고, 모든 필터를 수행하고 나면 마지막으로 DB에 신규 회원 정보를 추가하고 최종적으로 등록 성공 여부를 반환.

- 입력 소스가 정문, 후문, 쪽문 중 구역을 선택한 선택 구역인 상황: 입력 소스가 선택 구역 식당 검색 필터를 거치면서 식당 DB 내에 특정 구역에 해당하는 식당 리스트로 변환되고, 식당 리스트는 식당 좌표 추출 필터를 거쳐 식당명, 좌표 리스트로 변환되고, 최종적으로 식당명, 좌표 리스트가 반환됨.

1.5 Tradeoff를 통한 소프트웨어 아키텍처의 선택

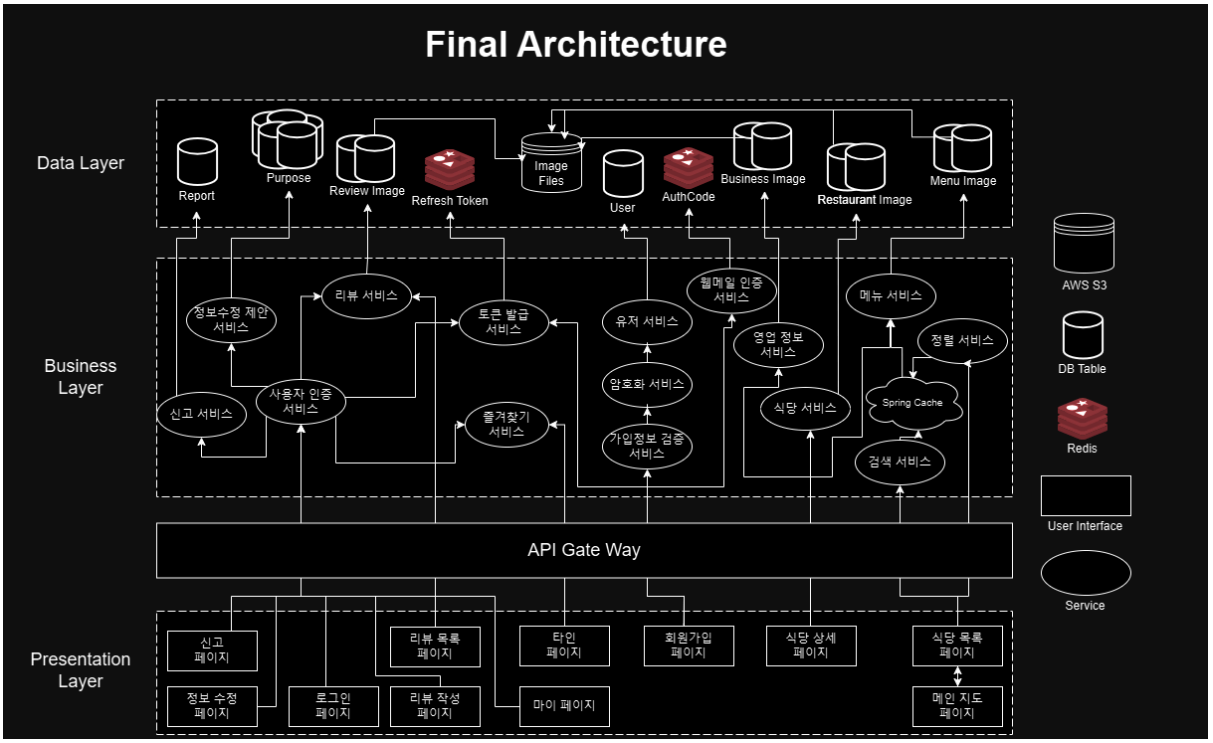
- Layered Architecture(Frontend/Backend): 장점으로는 유지보수성(레이어 간 독립성이 높아 특정 레이어만 수정해도 나머지 레이어에 영향이 적음), 역할 분리(UI, 비즈니스 로직, 데이터 접근 등을 구분해 역할과 책임 구체화 가능), 재사용성(비즈니스 로직이나 데이터 접근 레이어 등 특정 레이어를 다른 시스템에서도 재사용 가능), 테스트 용이성(각 레이어를 독립적으로 테스트할 수 있어 단위 테스트가 용이)가 있고, 단점으로는 성능 저하(각 요청이 여러 레이어를 거치기 때문에 성능 저하 우려), 복잡도 증가(각 레이어에 독립적 로직이 분산되어 레이어 간 통신이 복잡), 느슨한 결합 부족(레이어들 간 강한 결합이 필요할 수 있어, 특정 레이어 변경 시 다른 레이어에 영향 미칠 수 있음)이 있음.

- Event-Driven Architecture(Frontend): 장점으로 높은 확장성(비동기적으로 이벤트를 처리하므로 시스템 확장이 용이), 느슨한 결합(이벤트를 통해 서로 독립적인 컴포넌트들이 통신하므로, 서로 간의 결합도가 낮아 유지보수가 용이), 실시간 반응성(이벤트가 발생할 때 즉시 처리가 가능하여 실시간 반응성 증가), 확장성 높은 구조(새로운 이벤트와 이벤트 소비자를 쉽게 추가할 수 있어 변화에 유연하게 대처 가능)이 있고, 단점으로는 디버깅 어려움(이벤트 흐름이 복잡해지면 이벤트 간의 관계를 추적하고 디버깅하는 것이 어려울 수 있음), 데이터 일관성 문제(비동기

이벤트 처리 특성으로 인해 데이터의 일관성을 유지하는 데 주의가 필요), 복잡한 이벤트 관리(이벤트의 수와 이벤트를 처리하는 핸들러가 많아질수록 복잡성이 증가하며, 관리가 어려워질 수 있음), 기술 의존성(메시지 큐나 브로커 같은 추가 기술에 대한 의존성이 발생)이 있음.

- Pipe-Filter Architecture(Backend): 장점으로는 모듈화 및 재사용성(각 필터가 독립적인 작업을 수행하므로 개별 필터를 재사용하거나 교체하기가 용이), 데이터 흐름의 가독성(파이프를 통해 데이터를 처리하는 일련의 과정을 명확히 정의할 수 있어 데이터 변환 과정을 쉽게 파악), 병렬 처리 용이(각 필터는 독립적이기 때문에 병렬 처리나 분산 처리에도 적합), 단순한 확장(새로운 필터를 추가하거나 기존 필터를 교체하여 기능을 확장 가능)이 있고, 단점으로는 데이터 변환 부담(각 필터가 데이터를 다른 형식으로 변환하는 경우가 많아 데이터의 변환 부담 높음), 복잡한 오류 처리(파이프가 직렬로 연결되어 있어 특정 필터에서 오류가 발생하면 전체 파이프라인이 영향 받음), 고정된 데이터 흐름(파이프-필터 아키텍처는 고정된 데이터 흐름을 필요로 하기 때문에 동적인 데이터 흐름 처리 어려움), 성능 문제(파이프-필터 방식의 반복적인 데이터 변환과 복사가 성능 저하 유발 가능)이 있음.

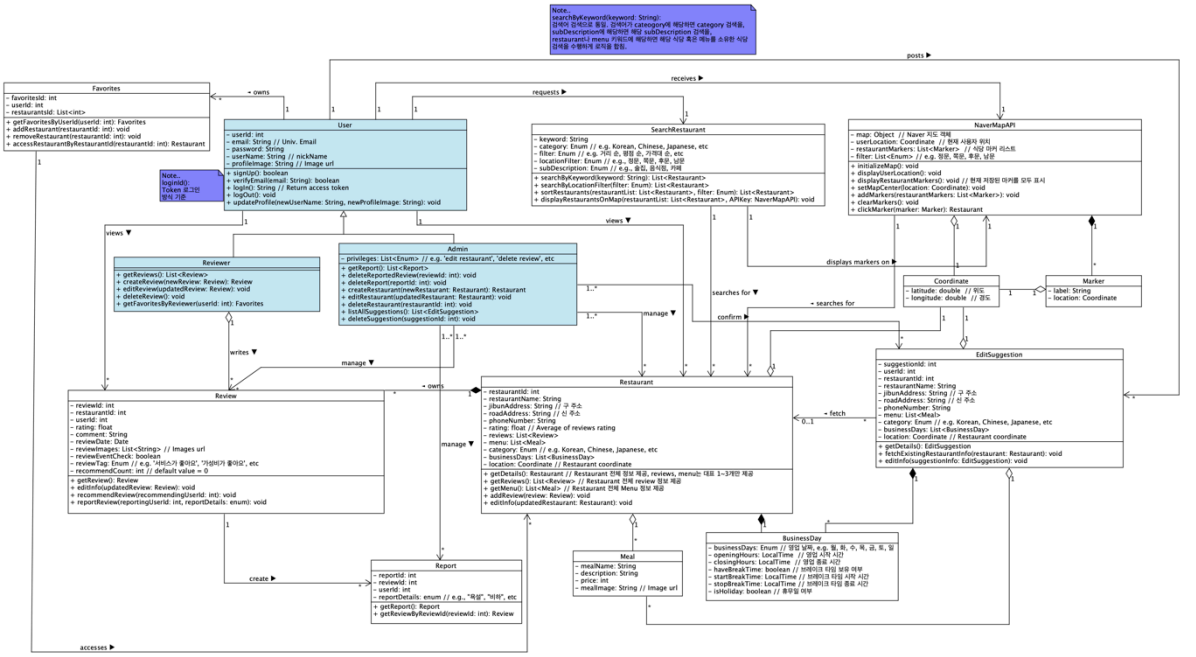
1.6 Final Architecture



1.6.1 상세내용

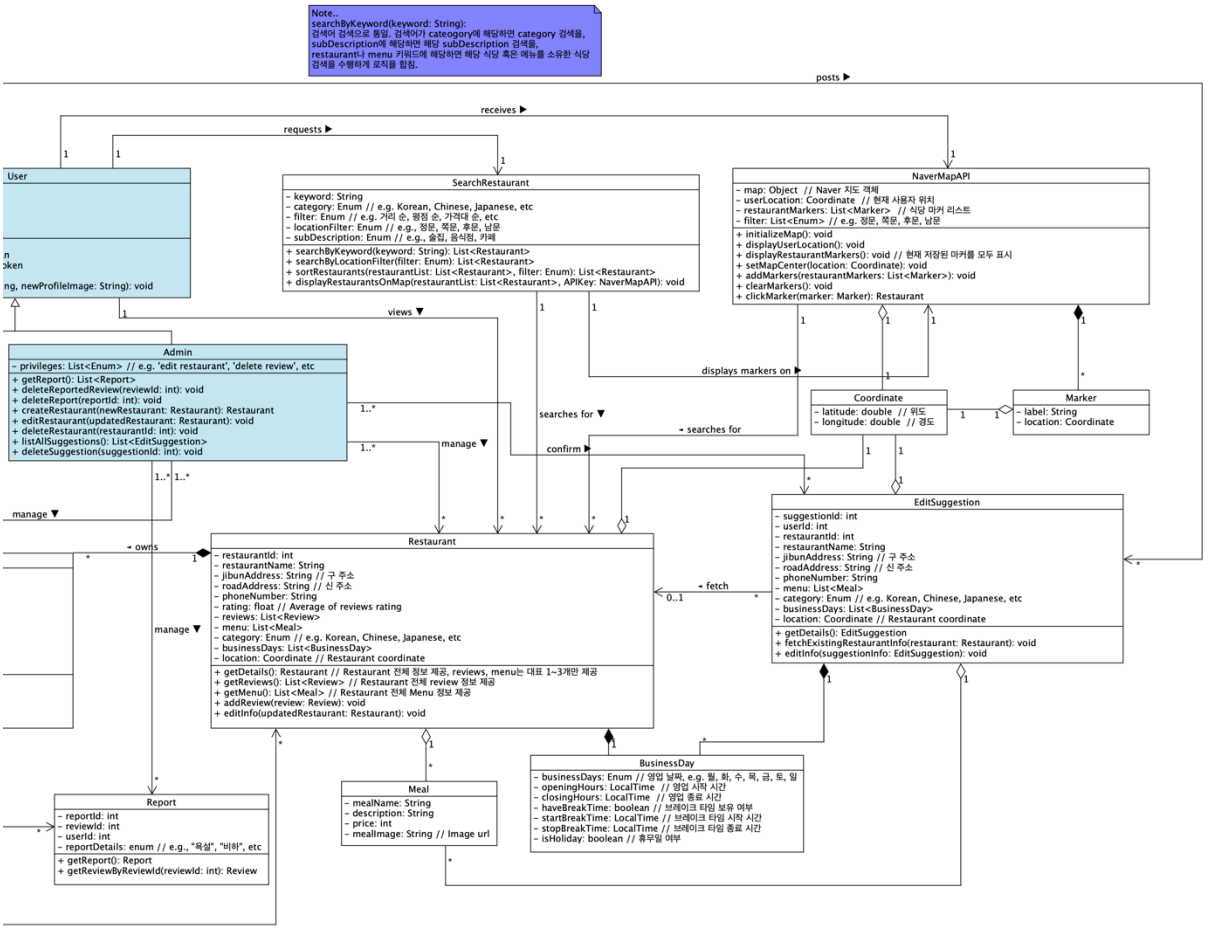
- Layered Architecture+Microservice Architecture: Layered Architecture와 Microservice Architecture의 특징을 결합한 구조로, Layered Architecture의 안정성과 Microservice Architecture의 유연성을 모두 반영하고자 함. 먼저, 전체 시스템은 Data Layer, Business Layer, Presentation Layer의 3계층으로 나뉘어 Layered 아키텍처의 분리된 책임 원칙을 따름.Data Layer는 Redis, MySQL DB, AWS S3 등의 저장소 내의 데이터를 관리하고, Business Layer는 사용자 인증, 리뷰, 검색 등 비즈니스 로직을 처리하며,Presentation Layer는 프론트엔드 화면과 사용자 경험을 제공함.또한, 각 서비스는 최대한 독립적인 기능 단위로 설계하여 결합도를 낮추고 확장성과 유지보수성을 높이고자 함

2. Class Diagram



<Class Diagram>

<Class Diagram - Left>



<Class Diagram – Right>

2.1 Class Diagram 설명

A. User: 일반 사용자(actor)를 나타내는 클래스로, Reviewer와 Admin을 자식 클래스로 둔 부모 클래스이다. 사용자 계정 생성, 로그인, 로그아웃, 이메일 인증, 프로필 변경 등을 수행한다.

B. Reviewer: User의 하위 클래스로, 리뷰어(actor)를 나타내며 리뷰에 관련된 전반적인 기능을 담당한다. 리뷰 생성(작성), 리뷰 수정, 리뷰 삭제, Reviewer의 즐겨찾기 목록 반환 등을 수행한다.

C. Admin: User의 하위 클래스로, 관리자(actor)를 나타낸다. 리뷰 신고 관리, 식당 정보 관리(생성, 수정, 삭제), 정보 수정 제안 조회 등을 수행한다.

D. Restaurant: 식당 정보를 저장하는 클래스로, 식당의 전반적인 세부 정보와 여러 Reviewer가 남긴 리뷰를 저장하고 관리한다. 식당 세부 정보 제공, 리뷰 제공, 메뉴 정보 제공 등의 기능을 수행한다.

   i. Meal: 단일 메뉴 정보를 저장하기 위한 클래스로, Restaurant의 부분 클래스이다. 식사 이름, 설명, 가격, 이미지 정보 등을 저장한다.

   ii. BusinessDay: 식당 영업 정보를 저장하기 위한 Restaurant의 부분 클래스이다. 영업 날짜, 오픈 시간, 마감 시간, 브레이크 타임 여부, 브레이크 타임 시작, 종료 시간, 휴무일 여부를 저장한다.

E. Review: Reviewer가 남긴 리뷰 정보를 저장하고 관리하는 클래스이다. reviewId, restaurantId, userId 등의 Id 정보와, 별점, comment, 작성 날짜, 이미지, 태그, 추천 수 등의 review 정보를 저장하고 관리한다. 또한 reportReview()를 통해 해당 리뷰를 신고할 수 있으며, 리뷰를 신고할 경우 그에 해당하는 Report 객체가

생성된다.

F. Report: 신고 정보를 저장하는 클래스로, Review 클래스의 reportReview() 메소드를 통해 생성된다. 신고 Id, 해당 reviewId, 신고한 유저 Id, 신고 내용(Category를 통해 선택) 등의 정보를 저장하고 관리한다.

G. NaverMapAPI: Naver Map API라는 외부 API를 통해 User에게 지도 정보를 제공하는 클래스이다. 기본적인 지도 제공 외에도 User 실시간 위치, Marker 기능(지도에 Marker 표시, 제거, Click event) 등의 기능을 제공한다.

   i. Coordinate: NaverMapAPI와 Restaurant의 부분 클래스로, 위도, 경도 좌표를 저장하는 클래스이다.

   ii. Marker: NaverMapAPI의 부분 클래스로, 지도에 마커를 표시하기 위해 사용하는 클래스이다. Coordinate를 부분 클래스로 두어 위치 정보를, label로 마커 이름을 저장한다.

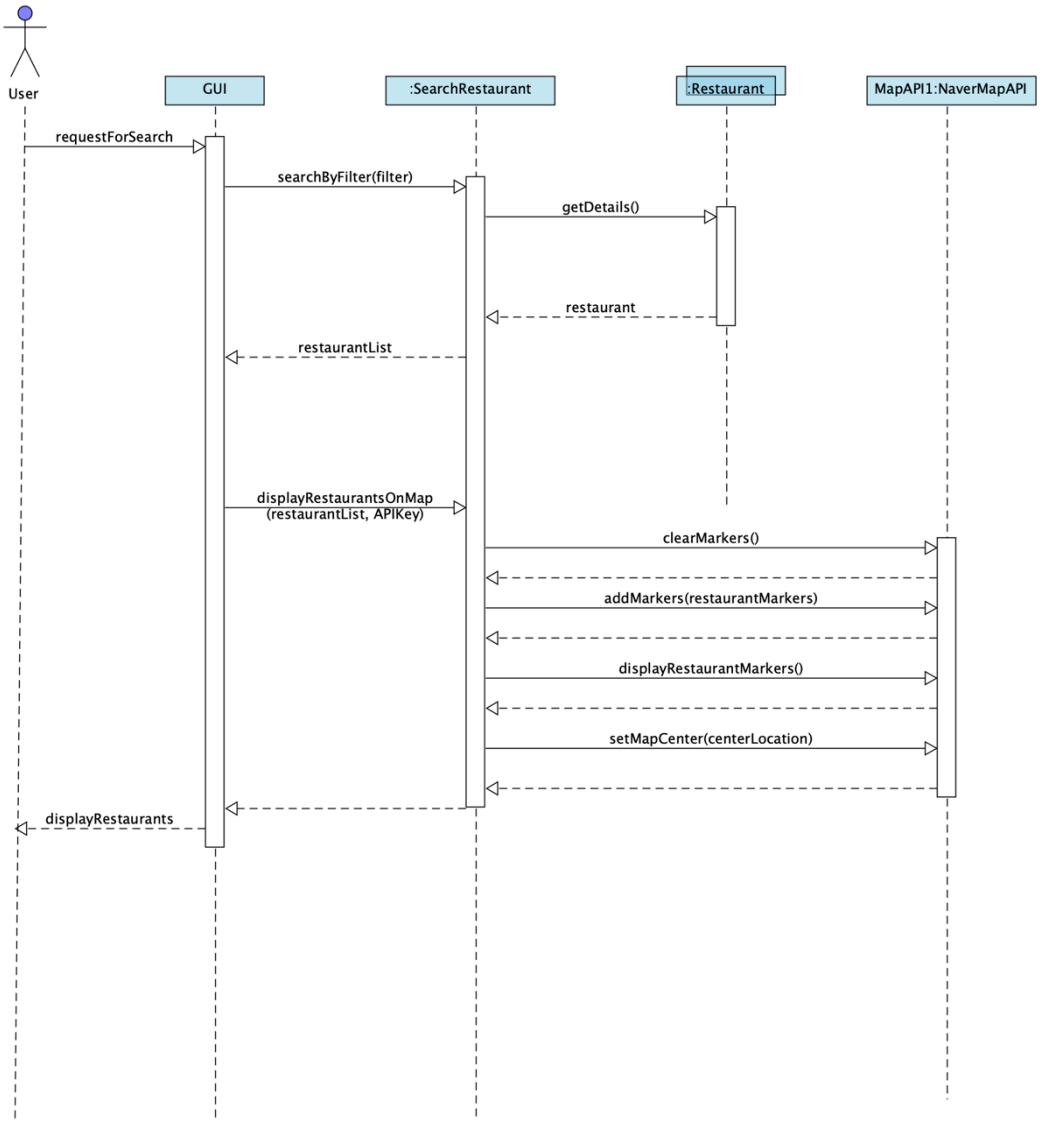
H. SearchRestaurant: 식당 검색을 위한 클래스로 category, subDescription 검색도 가능한 검색어를 통한 식당 검색, locationFilter를 통한 식당 검색 등의 식당 검색 기능과 검색된 식당 리스트 filter에 맞게 정렬, 지도에 마커로 표시 등의 기능을 제공한다.

I. EditSuggestion: 식당 정보 수정 제안을 위한 클래스로, 기존 식당의 정보를 수정하고자 할 때 사용된다. 기존 식당 정보 가져오기, 제안 정보 수정하기 등의 기능을 수행한다.

J. Favorites: User의 즐겨찾기 목록을 저장하는 클래스이다. 즐겨찾기 목록 반환, 식당 추가, 제거, 해당 식당 정보 제공 등의 기능을 수행한다.

3. Sequence Diagram

3.1 Use case 1(검색어 기반 식당 검색)

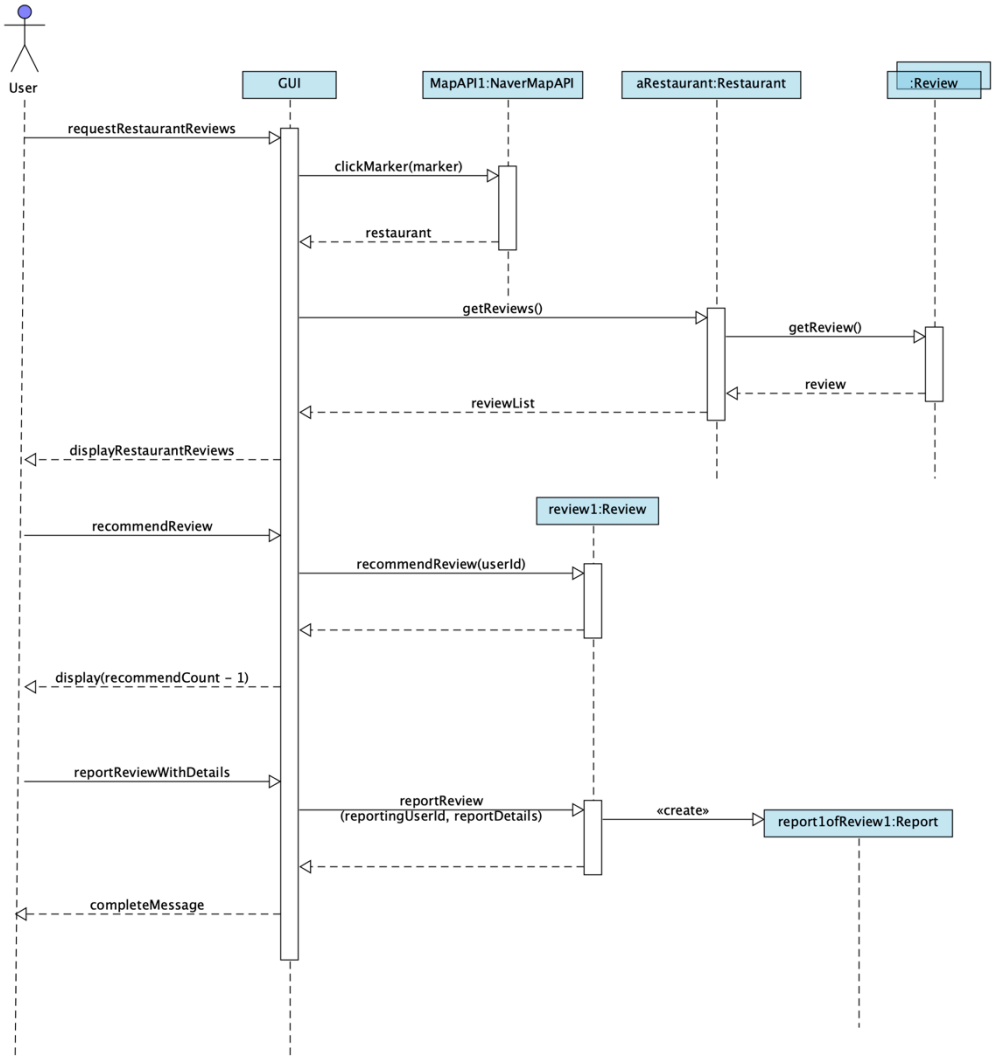


3.1.1 Use case 진행 과정:

i. User(actor)가 GUI에게 검색 요청을 한다.
ii. GUI는 SearchRestaurant의 인스턴스의 searchByFilter(filter) 메소드를 호출해 filter(e.g., 정문, 쪽문, 후문, 남문)에 맞는 식당 리스트를 반환할 것을 요청한다.
iii. SearchRestaurant의 인스턴스(이하 SearchRestaurant)는 Restaurant 인스턴스 모임의 getDetails() 메서드를 호출하고, 각 인스턴스에게 restaurant(Restaurant의 인스턴스)를 반환 받는다.
iv. SearchRestaurant는 GUI에게 restaurantList(List<Restaurant)를 반환한다.
v. GUI는 SearchRestaurant의 displayRestaurantsOnMap(restaurantList, APIKey) 메소드를 호출해 식당들을 마커로 표시할 것을 요청한다.
vi. SearchRestaurant는 NaverMapAPI의 인스턴스 MapAPI1의clearMarkers()(기존 마커 초기화), addMarkers(restaurantMarkers)(새로운 식당 마커 추가) 메소드를 호출한다.
vii. 그 후 저장된 마커를 모두 표시하는 displayRestaurantMarkers() 메소드를 호출한다.

viii. 마지막으로, 지도의 중심 위치를 표시한 마커들의 중간으로 맞추는 setMapCenter(centerLocation) 메소드를 호출한다.
ix. GUI는 User에게 결과를 보여주고 UseCase를 마친다.

3.2 Use case 2(리뷰 열람 및 상호작용)

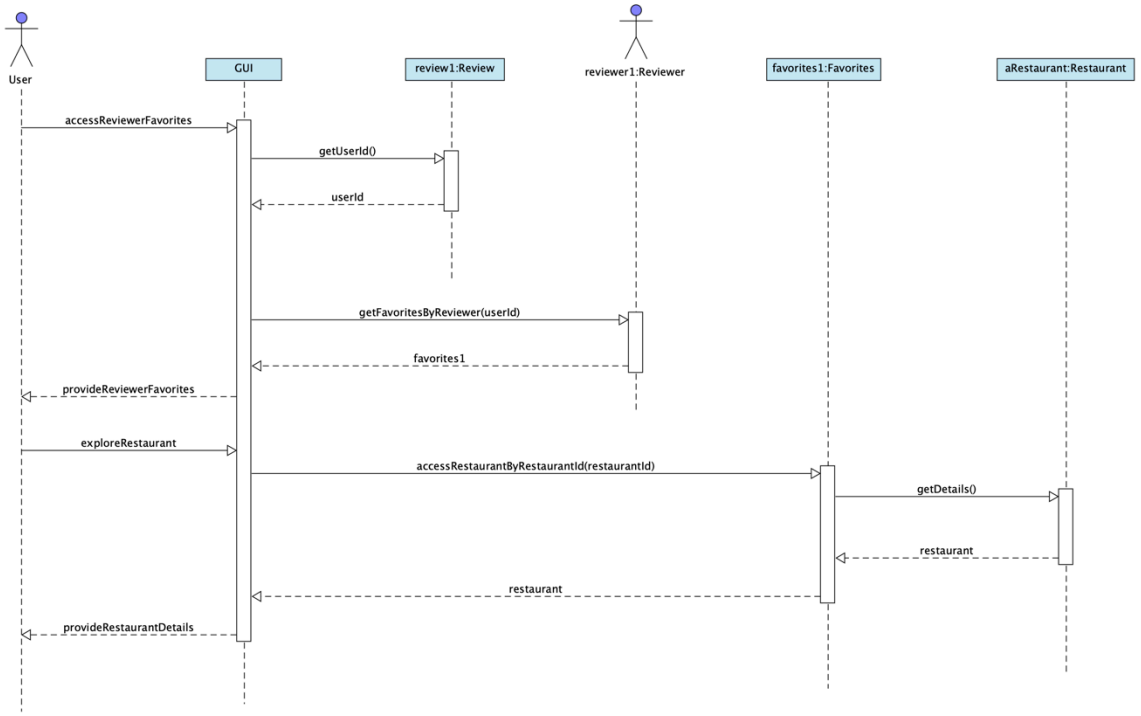


3.2.1 Use case 진행 과정:

i. User가 GUI에게 특정 식당(aRestaurant)의 리뷰 목록을 요청한다.
ii. GUI는 NaverMapAPI의 인스턴스 MapAPI1에게 해당 식당의 마커를 클릭하는 clickMarker(marker) 메소드를 호출한다. MapAPI1은 반환값으로 해당 식당을 나타내는 인스턴스 restaurant를 반환한다.
iii. GUI는 aRestaurant의 getReviews() 메소드를 호출한다.
iv. aRestaurant는 자신이 소유한 Review 인스턴스 모임에게 getReview() 메소드를 호출한다. Review 인스턴스 모임은 review 인스턴스를 반환한다.
v. aRestaurant는 getReviews()의 반환값으로 자신이 소유한 Review 인스턴스들의 모임 reviewList(=List<Review>)를 반환한다.
vi. GUI는 User에게 해당 식당의 리뷰 목록을 제시한다.
vii. User는 리뷰 목록 중 특정 리뷰에 '좋아요'를 누른다.

viii. GUI는 review1(해당 리뷰)의 recommendReview(userId) 메소드를 호출한다.
ix. review1의 recommendCount가 1 증가하고, GUI는 이를 사용자의 화면에 피드백한다.
x. User는 해당 리뷰를 신고한다. 이때 신고 내용 역시 포함된다(e.g., 욕설).
xi. GUI는 review1의 reportReview(reportingUserId, reportDetails) 메소드를 호출한다.
xii. review1은 해당 리뷰의 신고 인스턴스인 report1ofReview1 인스턴스(Report 클래스의 인스턴스)를 생성한다.
xiii. GUI는 사용자에게 완료 메시지를 표시한다.

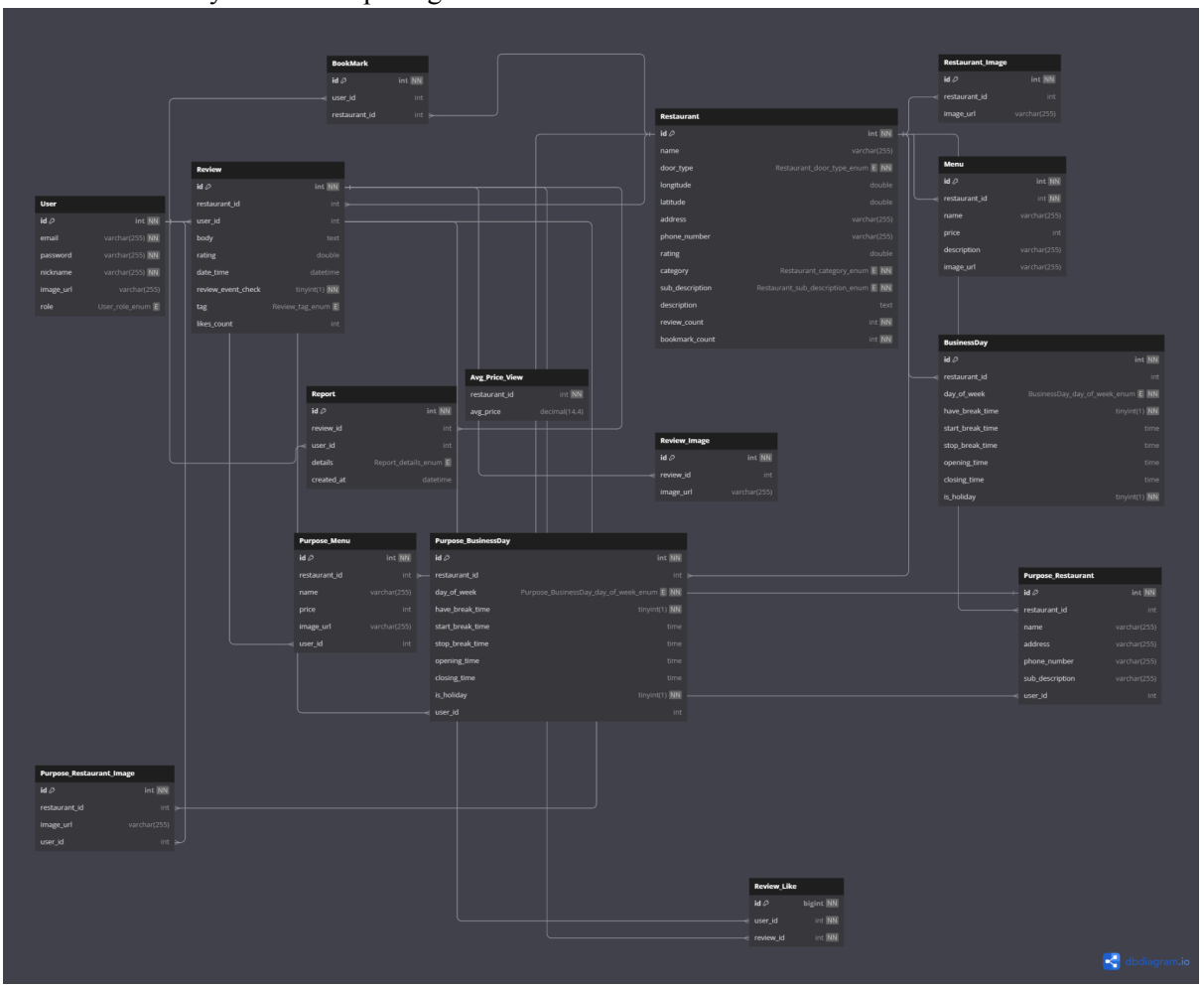
3.3 Use case 3(다른 사용자의 즐겨찾기 목록 열람)



3.3.1 Use case 진행 과정:

i. User는 GUI에게 특정 리뷰어의 즐겨찾기 목록에 접근할 것을 요청한다.
ii. GUI는 review1(해당 리뷰어의 리뷰)의 getUserId() 메소를 호출해 userId를 반환 받는다.
iii. GUI는 해당 userId를 갖는 reviewer1(Reviewer의 인스턴스)의 getFavoritesByReviewer(userId) 메소드를 호출해 해당 리뷰어의 즐겨찾기 목록 favorites1(Favorites의 인스턴스)를 반환 받는다.
iv. GUI는 요청 리뷰어의 즐겨찾기 목록을 User에게 제공한다.
v. User는 GUI에게 즐겨찾기 목록 중 특정 식당의 정보를 요청한다.
vi. GUI는 favorites1의 accessRestaurantByRestaurantId(restaurantId) 메소드를 호출해 해당 식당 정보를 요청한다.
vii. favorites1은 aRestaurant(해당 식당 인스턴스)의 getDetails() 메소드를 호출해 식당 정보 restaurant를 반환 받는다.
viii. favorites1은 GUI에게 restaurant를 반환한다.
ix. GUI는 User에게 요청 식당 정보를 제공한다.

4. Final Entity Relationship Diagram



4.1 Entity 설명

- User: 사용자 정보.

- id (PK): 사용자 ID
- email: 사용자 이메일
- password: 비밀번호
- nickname: 닉네임
- image_url: 프로필 이미지 URL
- role: 사용자 역할 (일반 사용자, 관리자)

- Restaurant: 음식점 정보

- id (PK): 음식점 ID
- name: 음식점 이름
- door_type: 출입구 (예: 정문, 쪽문, 후문, 남문)
- address: 주소
- longitude: 위도
- latitude: 경도
- phone_number: 전화번호
- rating: 평점
- category: 음식점 카테고리 (예: 한식, 중식 등)
- description: 음식점 설명
- sub_description: 음식점 추가 설명 (예: 음식점, 술집, 카페)

- review_count: 리뷰 개수
- bookmark_count: 북마크 수

- Restaurant_Image: 음식점 사진

- id (PK): 고유 ID
- restaurant_id (FK): 해당 이미지의 음식점 ID
- image_url: 이미지 URL

- Review: 리뷰 정보

- id: 리뷰 ID
- restaurant_id (FK): 해당 리뷰의 음식점 ID
- user_id (FK): 리뷰 작성자 ID
- rating: 평점 (0~5)
- body: 리뷰 내용
- review_event_check: 이벤트 참여 여부
- likes_count: 좋아요 수
- tag: 음식점 특징 태그 (예: “사장님이 친절해요”)
- date_time: 작성일자

- Review_Like: 리뷰 좋아요(추천) 수

- id (PK): 고유 ID
- review_id (FK): 좋아요를 누른 리뷰 ID
- user_id (FK): 좋아요를 누른 사용자 ID

- Review_Image: 리뷰 사진

- id (PK): 고유 ID
- review_id (FK): 해당 이미지의 리뷰 ID
- image_url: 이미지 URL

- Bookmark: 즐겨찾기 정보

- id (PK): 북마크 ID
- user_id (FK): 북마크한 사용자 ID
- restaurant_id (FK): 북마크한 음식점 ID

- Report: 신고

- id (PK): 신고 ID
- review_id (FK): 신고된 리뷰 ID
- user_id (FK): 신고한 사용자 ID
- details: 신고 상세 내용 (예: 욕설, 비하 발언, 광고성 리뷰)
- created_at: 신고 생성일자

- Avg_Price_View: 음식점 별 평균 가격

- restaurant_id (FK): 음식점 ID
- avg_price: 음식점 평균 가격

- Menu: 메뉴

- id (PK): 메뉴 ID
- restaurant_id (FK): 해당 메뉴의 음식점 ID
- name: 메뉴 이름
- price: 가격
- description: 설명

- image_url: 메뉴 이미지 URL

- BusinessDay: 영업일 정보

- id (PK): 고유 ID
- restaurant_id (FK): 음식점 ID
- day_of_week: 요일
- opening_time: 영업 시작 시각
- closing_time: 영업 종료 시각
- have_break_time: 휴식 시간 유무
- start_break_time: 휴식 시작 시간
- stop_break_time: 휴식 종료 시간
- is_holiday: 휴무 여부

- Purpose_Restaurant: 음식점 기본정보 수정 제안 이력

- id (PK): 고유 ID
- restaurant_id (FK): 음식점 ID
- name: 음식점 이름
- address: 음식점 주소
- phone_number: 음식점 전화번호
- sub_description: 음식점 추가 설명 (예: 음식점, 술집, 카페)
- user_id (FK): 해당 정보 제안을 등록한 사용자 ID

- Purpose_Restaurant_Image: 음식점 기본정보 수정 제안 이력 사진

- id (PK): 고유 ID
- restaurant_id (FK): 음식점 기본정보 수정 제안 이력 ID
- image_url: 음식점 이미지 URL
- user_id (FK): 해당 이미지를 등록한 사용자 ID

- Purpose_BusinessDay: 음식점 영업일정보 수정 제안 이력

- id (PK): 고유 ID
- restaurant_id (FK): 음식점 ID
- day_of_week: 요일 (예: 월요일, 화요일 등)
- have_break_time: 휴식 시간 존재 여부
- start_break_time: 휴식 시작 시간
- stop_break_time: 휴식 종료 시간
- opening_time: 영업 시작 시간
- closing_time: 영업 종료 시간
- is_holiday: 휴무 여부
- user_id (FK): 해당 정보 제안을 등록한 사용자 ID

- Purpose_Menu: 음식점 메뉴 정보 수정 제안 이력

- id (PK): 고유 ID
- restaurant_id (FK): 메뉴가 속한 음식점 ID
- name: 메뉴 이름
- price: 메뉴 가격
- image_url: 메뉴 이미지 URL
- user_id (FK): 해당 정보 제안을 등록한 사용자 ID