INSIDE CAMPUS

Software Design Specification

2021.05.16

Team 13(Inside Campus)

Team Leader	황 석진
Team Member	김 규용
Team Member	김 승호
Team Member	신 승환
Team Member	천 주형
Team Member	한 지명

목차

그림 목차

표 목차

1. Preface

이 챕터는 INSIDE CAMPUS 소프트웨어 디자인 명세서의 독자층, 범위, 목적, 구조에 대한 정보를 포함하고 있습니다.

1.1. Readership

이 소프트웨어 디자인 명세서는 다양한 소제목을 포함하고 있으며, 총 10가지 부분으로 나누어져 있습니다. 그리고 이 명세서의 메인 독자층은 Team13이고, 추가적으로 교수님과 조교님들, 그리고 소프트웨어 공학 개론을 수강하고 있는 학생들 또한 메인 독자층이 될 수 있습니다.

1.2. Scope

이 디자인 명세서는 캠퍼스 지도를 일반적인 도로와 건물 위치뿐만 아니라 건물 내부까지 표시해 주고, 위치 검색 및 원하는 위치까지의 경로를 표시하는 시스템을 구현하기 위해 필요한 소프트웨어 엔지니어링 및 소프트웨어 품질 엔지니어링에서 사용됩니다.

1.3. Objective

이번 소프트웨어 디자인 명세서의 본 목적은 INSIDE CAMPUS 앱의 기술적 디자인 측면에 대한 설명을 제공하는 것입니다. 이 문서는 INSIDE CAMPUS를 구현하는 데에 필요한 소프트웨어 아키텍쳐와 소프트웨어 디자인을 설명합니다. 또한 시스템의 다른 측면을 묘사하기 위한 시스템의 구조 개요를 제공합니다.
추가적으로 SRS문서 (요구사항 명세서)에서 논의된 일부 모듈의 구조와 설계를 명시하고, 프로그래밍 팀이 특정 모듈을 구현하는 방법을 보여주는 클래스 다이어그램을 포함하여 순차 및 활동 다이어그램으로 변환된 일부 사용 사례를 표시합니다.
이 명세서의 열람이 가능한 사람은 INSIDE CAMPUS 앱의 이해관계자, 개발자, 설계자 및 소프트웨어 테스터입니다.

1.4. Document Structure

- 1.Preface: 이 챕터는 독자층과 명세서의 범위, 시스템의 목적, 그리고 명세서의 구조를 설명합니다.
- 2.Introduction: 이 챕터는 이 명세서에 사용된 몇가지 도구와, 명세서와 레퍼런스에 사용된 몇가지 다이어그램, 그리고 이 프로젝트의 목적을 설명합니다.
- 3.Overall System Architecture: 이 챕터는 컨텍스트 다이어그램, 시퀀스 다이어그램, 유즈 케이스 다이어그램을 이용해 시스템의 전체적인 아키텍처를 설명합니다.
- 4.System Architecture - Frontend: 이 챕터는 클래스 다이어그램과 시퀀스 다이어그램을 이용해 프론트엔드 시스템의 아키텍처를 설명합니다.

- 5.System Architecture - Backend: 이 챕터는 클래스 다이어그램과 시퀀스 다이어그램을 이용해 백엔드 시스템의 아키텍처를 설명합니다.
- 6.Protocol Design: 이 챕터는 클라이언트와 서버의 커뮤니케이션을 위한 몇 가지 프로토콜 디자인을 설명합니다.
- 7.Database Design: 이 챕터는 ER 다이어그램과 SQL DDL을 이용한 데이터베이스 디자인을 설명합니다.
- 8.Testing Plan: 이 챕터는 시스템의 테스트 계획을 설명합니다.
- 9.Development Plan: 이 챕터는 시스템을 개발하기 위해 사용된 도구와 제약 조건, 가정 및 종속성을 설명합니다.
- 10.Supporting Information: 이 챕터는 이 명세서의 기록과 기준을 설명합니다.

2. Introduction

이 프로젝트는 캠퍼스 내의 상세한 지도를 표현함으로써 신입생들이나 학교를 처음 방문하는 사람들에게는 보다 정확한 위치를 표시해주고 현재 위치에서 원하는 위치까지의 경로 및 시간을 표시해줍니다. 또한 기존 재학생들과 학교 관계자분들, 혹은 캠퍼스에 자주 방문하신 분들에게도 생소한 위치나 건물 내부의 강의실 위치, 해당 강의실의 강의 시간 등을 자세히 알려줍니다. 이 디자인 명세서에는 프로젝트 구현에 사용되거나 사용될 설계가 나와 있습니다. 설명된 디자인은 프로젝트를 위해 미리 준비한 소프트웨어 요구사항 명세서에 지정된 요구사항을 따릅니다.

2.1. Objectives

이 챕터에서는 디자인 단계에서 이 프로젝트에 적용한 다양한 도구와 도표를 설명합니다.

2.2. Applied Diagrams

- UML (통합 모델링 언어)
  소프트웨어 공학에서 사용되는 표준화된 범용 모델링 언어이다. 이 표준은 UML을 고안한 객체 관리 그룹에서 관리하고 있다. UML은 데이터 모델링(개체-관계 모델)과 비즈니스 모델링(업무 흐름), 객체 모델링, 부품 모델링의 최선의 기술을 조합한다. UML은 소프트웨어 개발 프로세스뿐만 아니라 다른 구현 기술의 모든 공정에서 사용될 수 있다. UML은 Booch 방법론의 객체 모델링 기법(OMT)와 객체 지향 소프트웨어 공학(OOSE)을 광범위하게 사용할 수 있는 단일한 공통 모델링 언어로 통합한다. UML의 목표는 동시적 분산 시스템을 모델링하는 표준 언어다. UML은 산업의 실질적 표준으로서, 객체 관리 그룹(OMG)에 의해 개선되고 있다.

초기에 OMG가 엄격한 소프트웨어 모델링 언어를 만들기 위해 객체 지향 방법론적인 통지를 요청했고, 많은 산업 선구자가 UML 표준 제작을 돕기 위해 진지하게 응답하였다. UML 모델은 객체 관리 그룹이 지원하는 QVT와 같은 변환 언어 등을 이용해 다른 표현(예를 들면 자바)으로 자동적으로 변환된다. UML은 확장할 수 있으며 커스터마이제이션을 위한 메커니즘인 프로파일 (UML), 스테레오타입 (UML)을 제공한다. 프로파일을 이용한 확장의 의미는 UML 2.0에서 개선되었다.

- Use case Diagram

  유스 케이스 다이어그램은 사용자, 그리고 사용자가 수반한 다른 유스 케이스 간의 관계를 보여주는 사용자-시스템 간 상호작용의 표현이다. 유스 케이스 다이어그램은 각기 다른 종류의 시스템 사용자와 각기 다른 유스 케이스를 식별할 수 있으며 다른 유형의 다이어그램이 수반되기도 한다. 유스 케이스는 원이나 타원으로 표현된다.

- Sequence Diagram

  시퀀스 다이어그램은 컴퓨터 과학 커뮤니티뿐만 아니라 비즈니스 애플리케이션 개발을 위한 설계 레벨 모델로서도 아마도 가장 중요한 UML 다이어그램일 것입니다. 최근에는 시각적 자기 설명적 특성 때문에 비즈니스 프로세스를 묘사하는 데 인기를 얻고 있습니다. 이름에서 알 수 있듯이, 시퀀스 다이어그램은 행위자와 객체 간에 발생하는 메시지 및 상호 작용의 순서를 설명합니다. 행위자 또는 사물은 필요하거나 다른 사물과 통신하려는 경우에만 활성화될 수 있습니다. 모든 통신은 시간 순으로 표시됩니다.

- Class Diagram

  UML의 클래스 다이어그램은 시스템의 클래스, 속성, 작업(또는 메서드) 및 개체 간의 관계를 보여줌으로써 시스템의 구조를 설명하는 정적 구조 다이어그램의 한 유형입니다. 클래스는 객체의 구성 블록이므로 클래스 다이어그램은 UML의 구성 블록입니다. 클래스 다이어그램의 다양한 구성 요소는 실제로 프로그래밍될 클래스, 주 객체 또는 클래스와 객체 사이의 상호 작용을 나타낼 수 있습니다. 클래스 모양 자체는 세 개의 행이 있는 직사각형으로 구성됩니다. 맨 위 행에는 클래스의 이름이 포함되고 가운데 행에는 클래스의 속성이 포함되며 맨 아래 섹션에는 클래스에서 사용할 수 있는 메서드 또는 작업이 표시됩니다. 클래스와 하위 클래스가 함께 그룹화되어 각 개체 간의 정적 관계를 표시합니다.

- Context Diagram
  시스템 컨텍스트 다이어그램(레벨 0 DFD)은 데이터 흐름 다이어그램에서 가장 높은 레벨이며 모델링할 시스템의 컨텍스트와 경계를 설정하는 전체 시스템을 나타내는 하나의 프로세스만 포함합니다. 이는 시스템과 외부 실체(즉, 행위자) 사이의 정보 흐름을 식별합니다. 상황별 다이어그램은 일반적으로 요구 사항 문서에 포함되어 있습니다. 이 내용은 모든 프로젝트 관계자가 읽어야 하므로 이해 관계자가 항목을 이해할 수 있도록 쉬운 언어로 작성해야 합니다. 시스템 컨텍스트 다이어그램의 목적은 전체 시스템 요구 사항 및 제약 조건 집합을 개발할 때 고려해야 하는 외부 요인 및 이벤트에 주의를 집중하는 것입니다. 시스템 컨텍스트 다이어그램은 프로젝트 초기에 조사 대상 범위를 결정하는 데 종종 사용됩니다. 따라서, 문서 내에 있습니다. 시스템 컨텍스트 다이어그램은 시스템과 상호 작용할 수 있는 모든 외부 엔터티를 나타냅니다. 전체 소프트웨어 시스템이 단일 프로세스로 표시됩니다. 이러한 다이어그램은 내부 구조에 대한 세부 정보가 없는 중앙의 시스템을 모든 외부 도면요소, 상호 작용하는 시스템 및 환경으로 표시합니다.

- ERD (Entity Relationship Diagram)
  ERD는 데이터베이스에 저장된 엔티티 세트의 관계를 보여줍니다. 이 컨텍스트에서 엔티티는 개체이며 데이터의 구성 요소입니다. 엔티티 집합은 유사한 엔티티의 컬렉션입니다. 이러한 엔티티는 속성을 정의하는 속성을 가질 수 있습니다. 엔티티, 속성을 정의하고 개체 간의 관계를 표시함으로써 ER 다이어그램은 데이터베이스의 논리적 구조를 보여줍니다. 엔티티 관계 다이어그램은 데이터베이스 설계를 스케치하는 데 사용됩니다.

2.3. Applied Tools

- Draw.io
  다이어그램을 그리는 데에 사용한 사이트 주소입니다. 전반적인 다이어그램을 그리는 데에 필요한 모양이나 선 등이 준비되어 있으며, 사이트에서 작성된 다이어그램은 사이트 내에서 열 경우, 언제 어디서든 누구나 수정이 가능하게끔 되어 있습니다.

- Aquerytool.com
  ERD를 보다 쉽게 표현할 수 있는 사이트 주소입니다. 웹 기반이라 인터넷만 있으면 시간과 장소에 제한이 없으며, 작성된 파일을 가진 누구나 수정이 가능합니다.

- Gitmind.com
  웹 기반 마인드맵핑 사이트입니다. 다양한 아이콘을 사용해 보다 직관적인 다이어그램을 그릴 수 있습니다.

2.4. Project Scope

INSIDE CAMPUS는 캠퍼스 어디에 무엇이 있는지 잘 모르거나, 캠퍼스 내에서 강의실 혹은 특정 위치를 가고 싶을 때, 실제 거리와 문 위치, 복도 사진, 강의실 위치 등을 보여주면서 빠르고 정확하게 찾아가게끔 도와줍니다. 로드뷰 방식의 지도 데이터는 데이터베이스 서버에 저장하여 불러오는 방식을 채택할 것입니다.

2.5. References

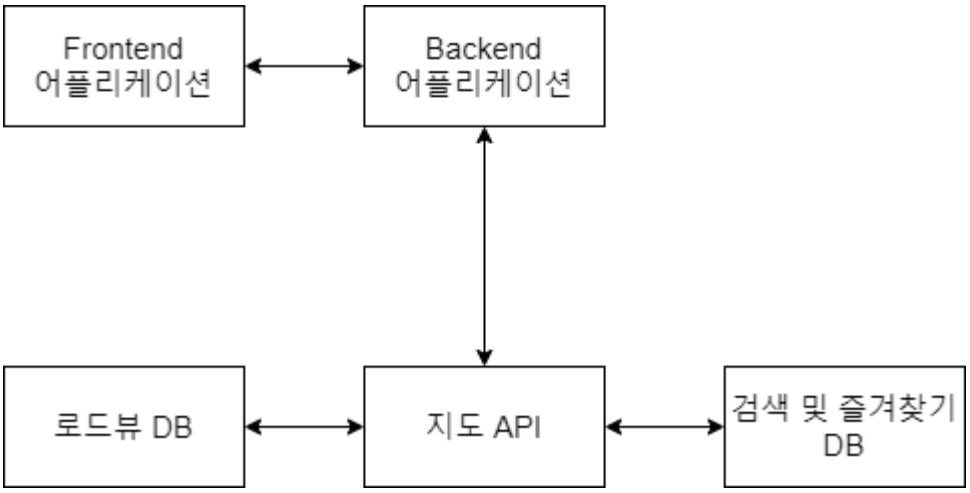
- Team 1, 2020 Spring, Software Design Document, SKKU

3. System Architecture – Overall

3.1. Objectives

프론트엔드 디자인부터 백엔드 디자인까지의 시스템 구성을 설명합니다.

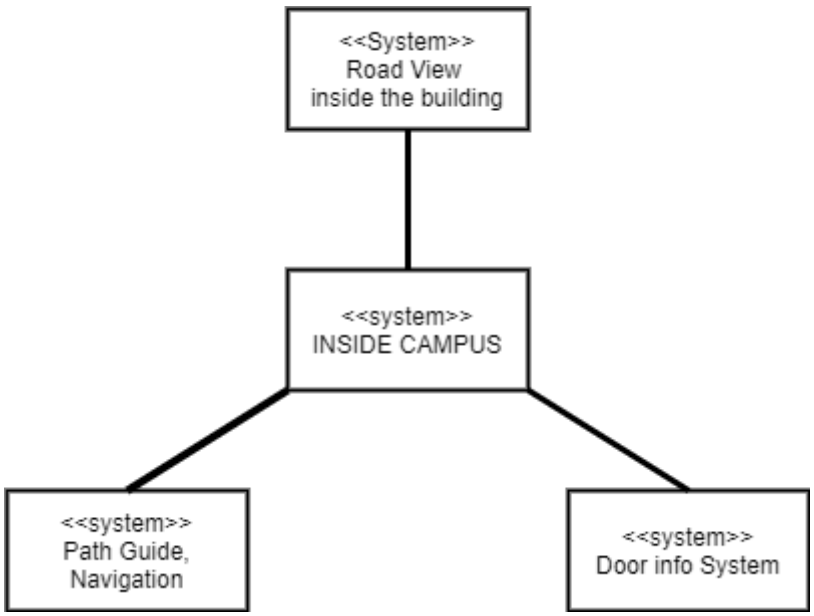
3.2. System Organization



[Figure 1] system architecture - overall

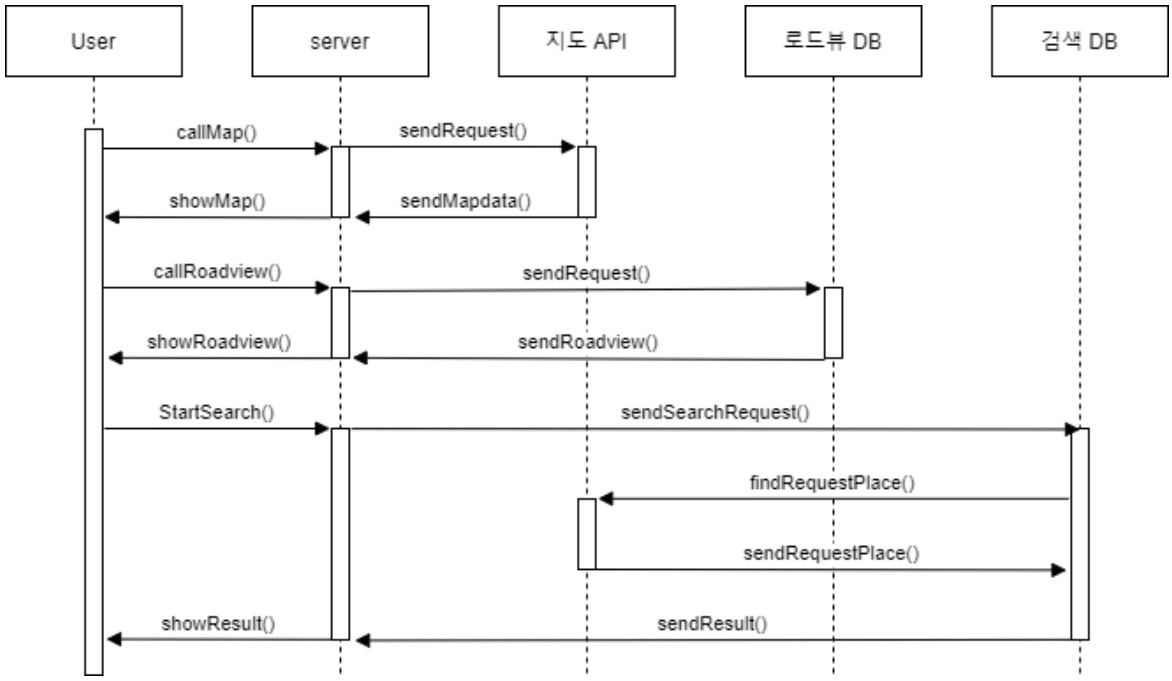
이 서비스는 client-server 모델을 적용시킨 시스템으로 디자인할 것입니다. Frontend 어플리케이션에서 이 앱의 모든 기능을 유저들에게 제공할 것이고, Backend 어플리케이션에선 Frontend 어플리케이션의 기능들에 사용되는 모든 데이터를 관리할 것입니다. Frontend와 Backend 어플리케이션은 JSON을 기반으로 데이터를 주고 받습니다.

3.2.1. Context Diagram



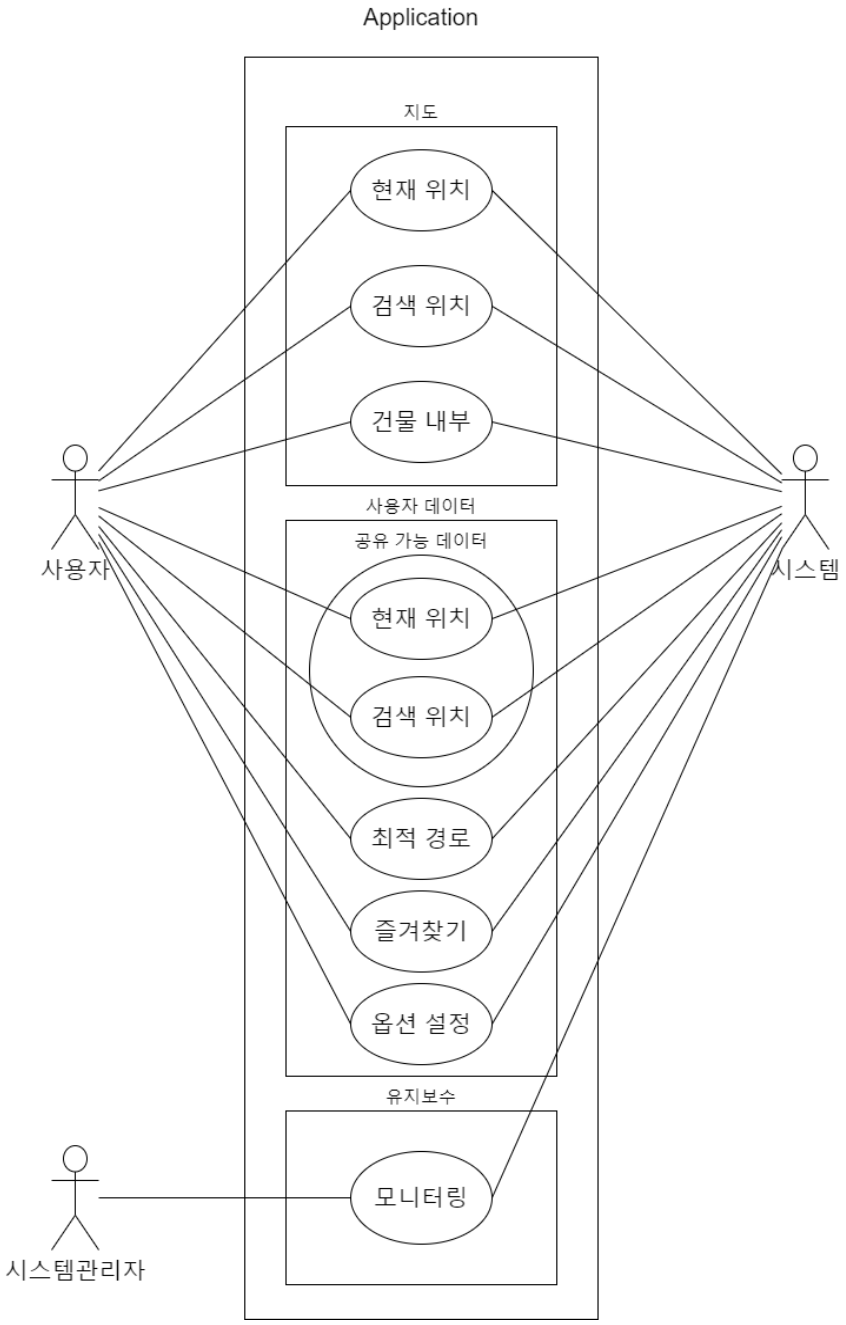
[Figure 2] Context Diagram

3.2.2. Sequence Diagram



[Figure 3] Sequence Diagram

3.2.3. Use Case Diagram



[Figure 4] Use Case Diagram

4. System Architecture - Frontend

4.1. Objectives

이 장에서는 frontend system의 구조, 특성, 기능을 소개하고 각 요소 간의 관계를 묘사한다

4.2. Subcomponents

4.2.1. Map Visualization

Map class는 앱을 실행 시 혹은 검색을 완료한 뒤에 경로를 설정할 시에 지도를 보여주는 class이다.

4.2.1.1. attribute

Map class는 다음과 같은 attribute를 가지고 있다.

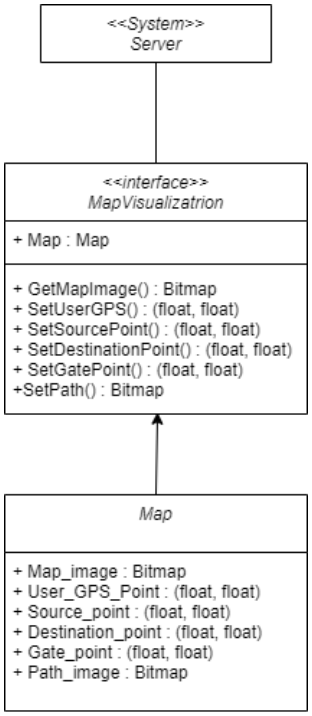
- Map_image : API로 받아오는 지도 이미지이다.
- User GPS : 현재 user의 위치이다.
- Start_point : 경로를 탐색할 때 지도에 표시할 시작 지점이다.
- Goal_point : 경로를 탐색할 때 지도에 표시할 목표 지점이다.
- Gate_point : 경로 탐색 시 해당 건물의 출입이 가능한 문 중 가장 가까운 문의 위치이다.
- Path_image : User GPS와 Goal을 잇는 경로의 이미지이다.

4.2.1.2. Methods

Map class는 위와 같은 methods를 가지고 있다.

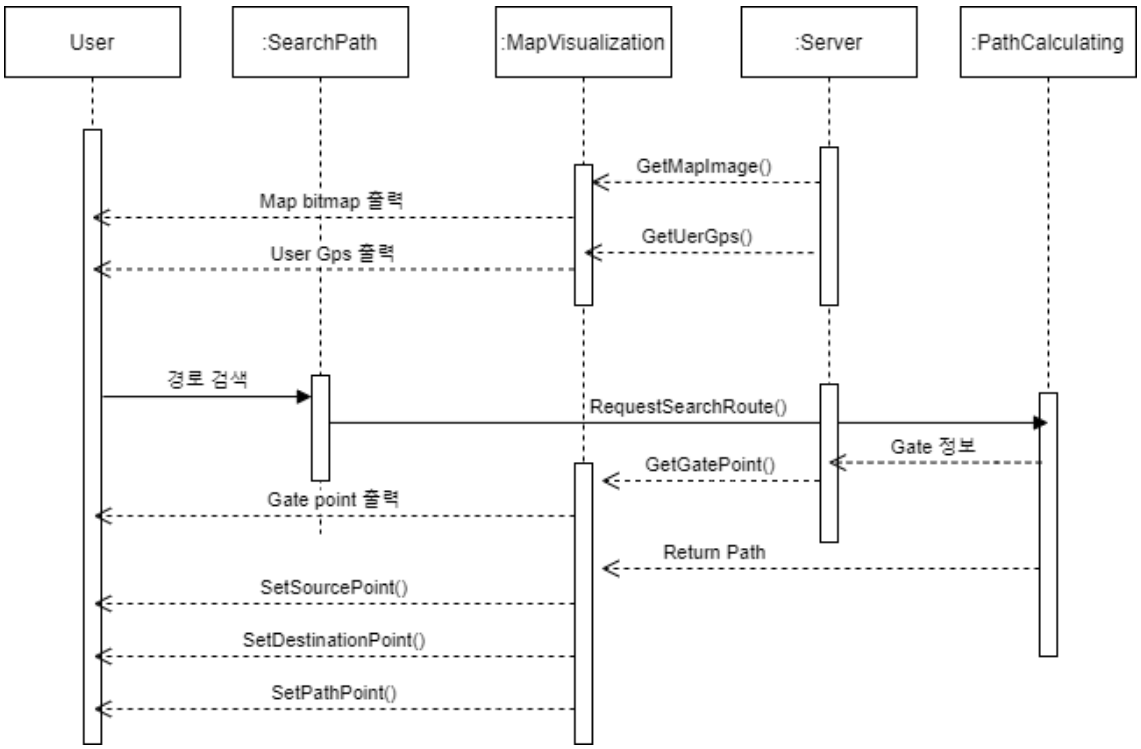
- GetMapImage()
- SetUserGPS()
- SetSourcePoint()
- SetDestinationPoint()
- SetGatePoint()
- SetPath()

4.2.1.3. Class Diagram



[Figure 5] Class diagram - MapVisualization

4.2.1.4. Sequence Diagram



[Figure 6] Sequence diagram – MapVisualization

4.2.2. roadview

roadview는 앱 내에서 2d로 표현된 지도가 아닌 파노라마 사진으로 주변을 자세히 보고 싶을 때 혹은 건물 내부의 사진을 보고 싶을 때 주위의 사진을 보여주는 class이다.

4.2.2.1. attribute

roadview class는 다음과 같은 attribute를 가지고 있다.

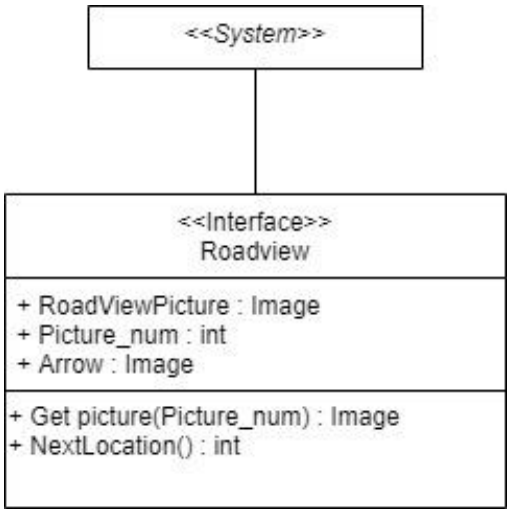
- Picture : roadview에 필요한 파노라마 이미지이다.
- Picture_num : 데이터 중 사진의 번호이다.
- Arrow : 다음에 표시할 roadview의 방향을 지정한다.

4.2.2.2. Methods

roadview class는 다음과 같은 methods를 가지고 있다.

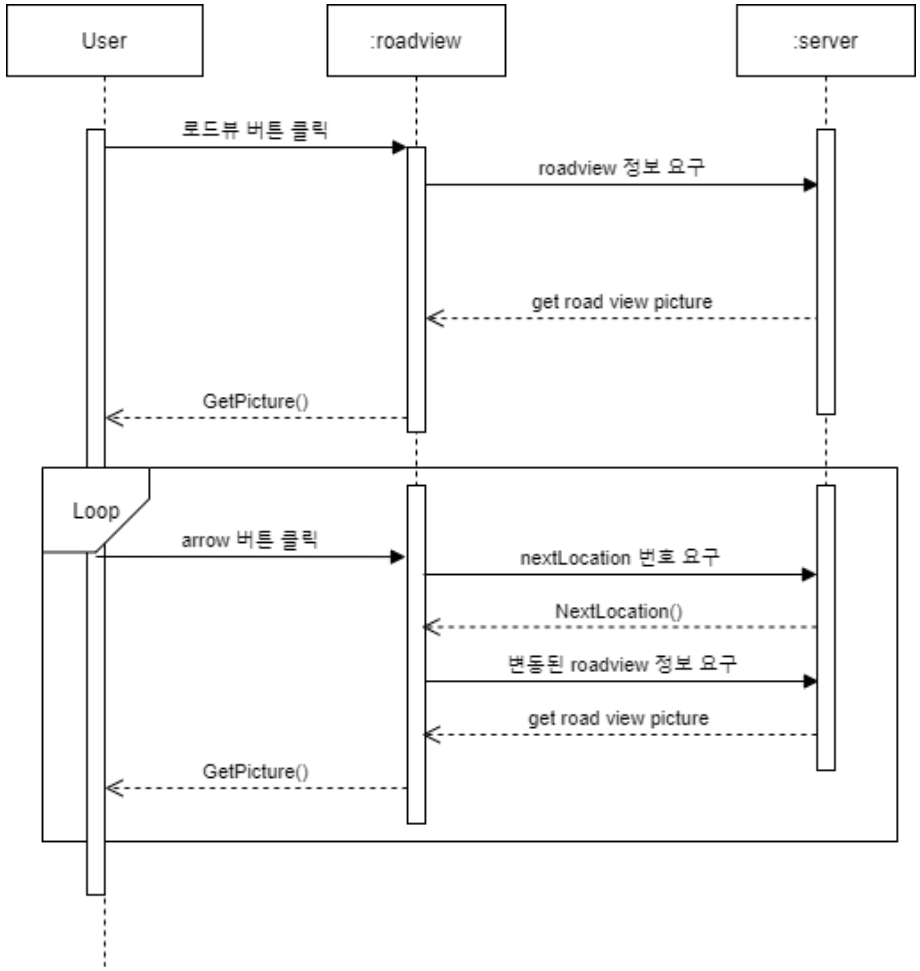
- GetPicture()
- NextLocation()

4.2.2.3 Class Diagram



[Figure 7] Class diagram – roadview

4.2.2.4. Sequence Diagram



[Figure 8] Sequence diagram – roadview

4.2.3. BuildingInfo

BuildingInfo는 지도에 있는 건물 혹은 검색을 실행한 뒤에 건물을 탭 했을 시 건물에 대한 자세한 정보를 나타내 주는 class이다.

4.2.3.1. attribute

BuildingInfo는 다음과 같은 attribute를 가지고 있다.

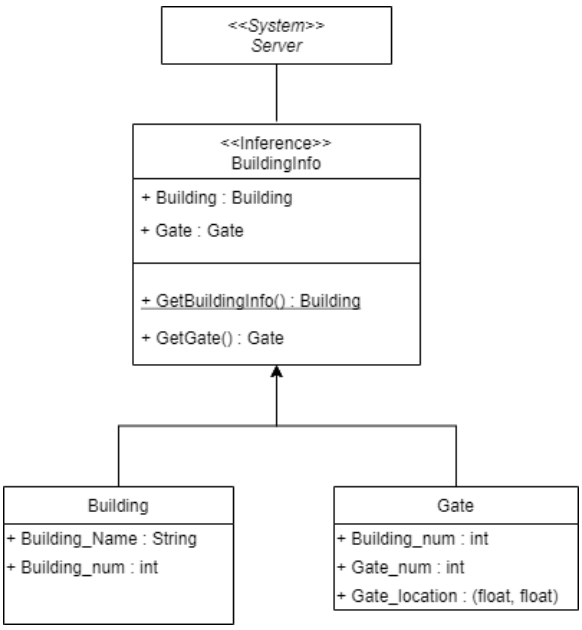
- Building name : 건물의 이름이다.
- Building num : 건물의 번호이다.
- Gate : 해당 건물의 출입이 가능한 문 중 가장 가까운 문의 위치이다.

4.2.3.2. Methods

BuildingInfo는 다음과 같은 methods를 가지고 있다.

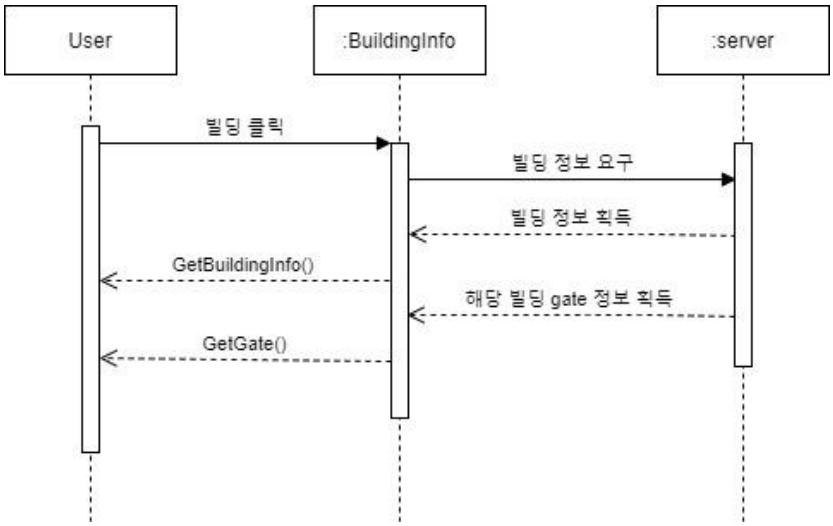
- GetBuildingInfo()
- GetGate()

4.2.3.3 Class Diagram



[Figure 9] Class diagram - BuildingInfo

4.2.3.4 Sequence diagram



[Figure 10] Sequence diagram - BuildingInfo

4.2.4. SearchPath

SearchPath class에서는 사용자가 찾고자 하는 경로를 검색하도록 하는 class이다. 사용자로부터 출발지와 목적지를 가져와 pathInfo 객체를 Path Calculating System으로 보낸다.

4.2.4.1. Attributes

pathInfo class는 다음과 같은 attribute를 가지고 있다.

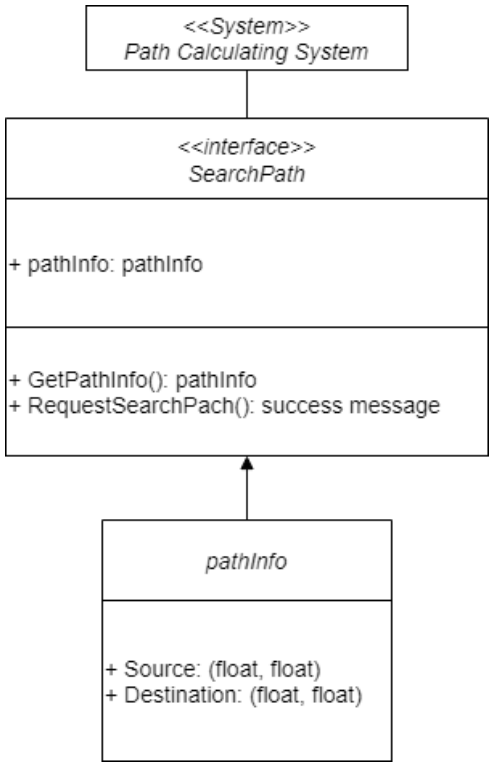
- **Source:** 사용자가 검색하려고 하는 경로의 출발지. 기본값은 사용자의 현재 위치이며, 특정 건물에서의 특정 위치 또는 GPS상의 좌표로 표현된다.
- **Destination:** 사용자가 검색하려고 하는 경로의 도착지. 특정 건물에서의 특정 위치 또는 GPS상의 좌표로 표현된다.

4.2.4.2. Methods

SearchPath class는 다음과 같은 methods를 가지고 있다.

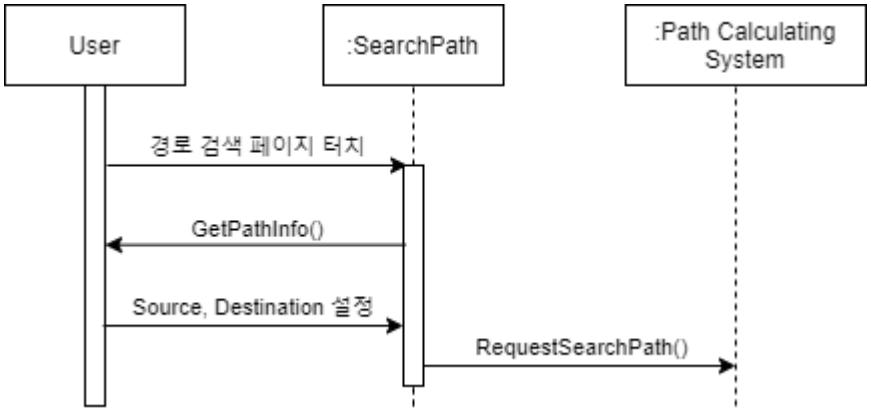
- **GetPathInfo()**
- **RequestSearchPath()**

4.2.4.3. Class Diagram



[Figure 11] Class diagram - SearchPath

4.2.4.4. Sequence Diagram



[Figure 12] Sequence diagram - SearchPath

4.2.5. MyFavorite

MyFavorite class는 특정 건물이나 강의실을 사용자가 즐겨찾기에 추가하거나 삭제하도록 하는 class이다. 사용자가 즐겨찾기 버튼을 터치했을 때 해당 장소가 즐겨찾기 데이터에 없다면 추가한다. 이미 즐겨찾기 데이터에 있는 장소라면 즐겨찾기 데이터에서 지운다.

4.2.5.1. Attributes

favorite class는 다음과 같은 attribute를 가지고 있다.

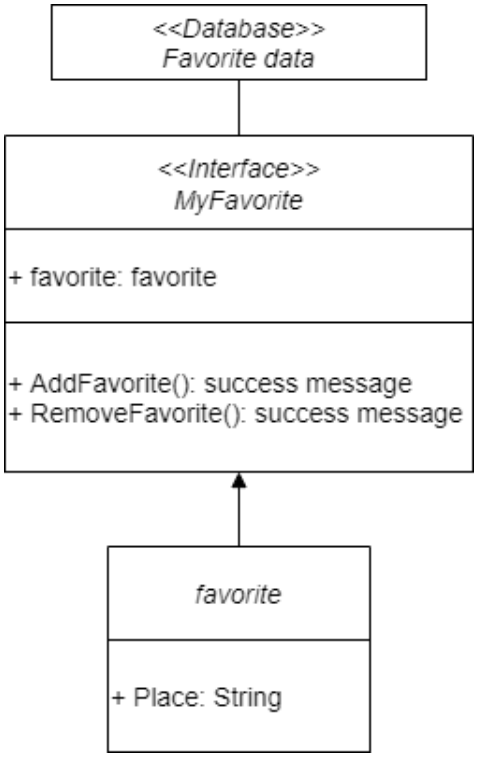
- **Place:** 즐겨찾기에 추가하려는 위치. 건물이나 강의실의 번호를 담고 있다.

4.2.5.2. Methods

MyFavorite class는 다음과 같은 methods를 가지고 있다.

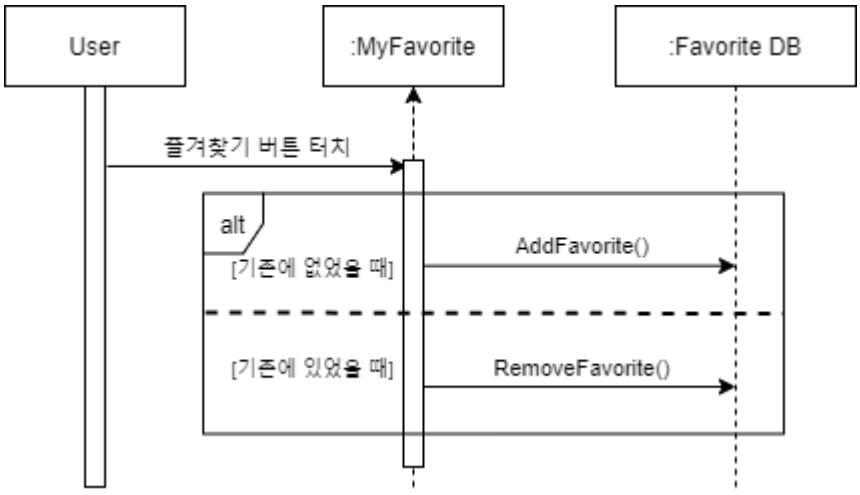
- **AddFavorite()**
- **RemoveFavorite()**

4.2.5.3. Class Diagram



[Figure 13] Class diagram - MyFavorite

4.2.5.4. Sequence Diagram



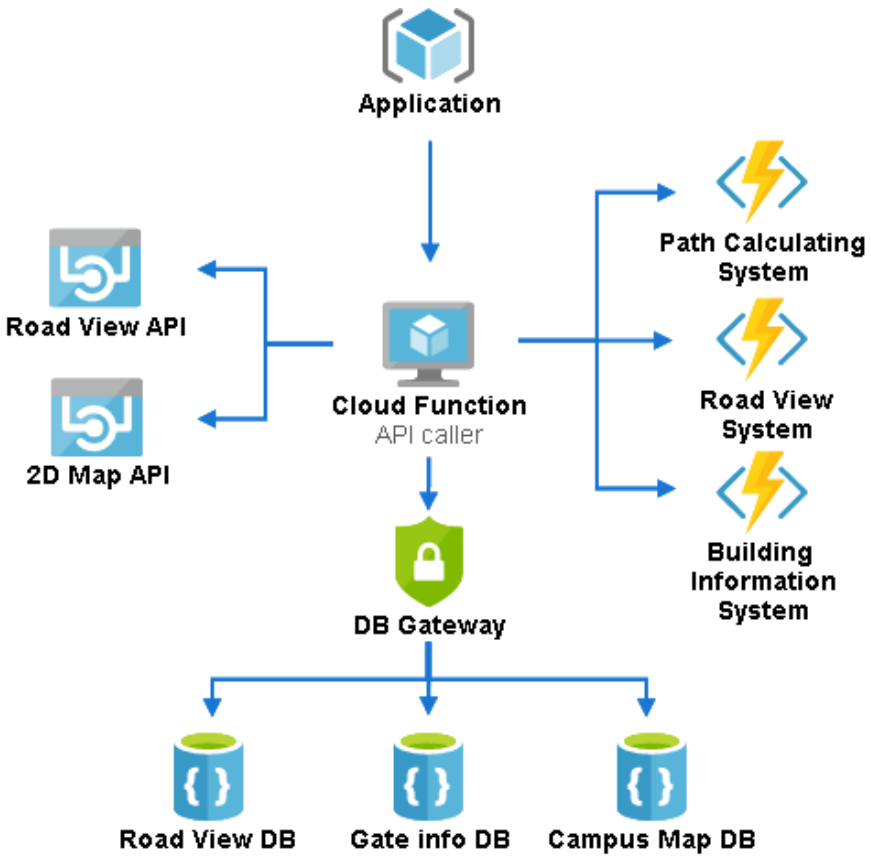
[Figure 14] Sequence diagram - MyFavorite

5. System Architecture – Backend

5.1. Objectives

이 챕터에서는 데이터베이스와 API 클라우드를 포함하는 백엔드 시스템의 구조를 표현한다.

5.2. Overall Architecture

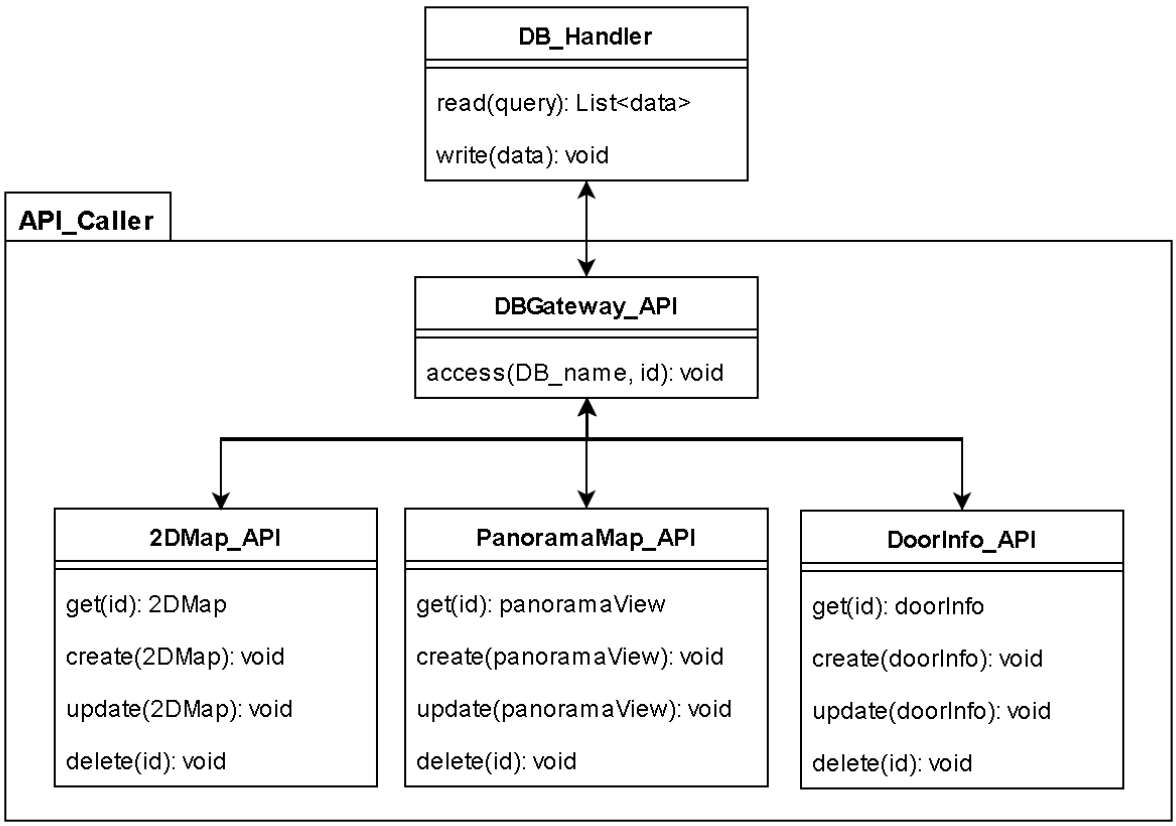


[Figure 15] Overall architecture

백엔드 시스템의 전체적인 구조는 위의 그림과 같다. 프론트엔드로부터 온 요청사항에 대해 Cloud Funtion(API caller)는 알맞은 API를 요청한다. API 요청을 통해 서브 시스템에 접근하게 되고, 서브 시스템의 요청에 따라 데이터베이스에 접근하여 데이터를 불러온다. 데이터베이스의 접근은 보안을 위해 자체 게이트웨이를 사용한다.

5.3. Subcomponents

5.3.1. Cloud Function



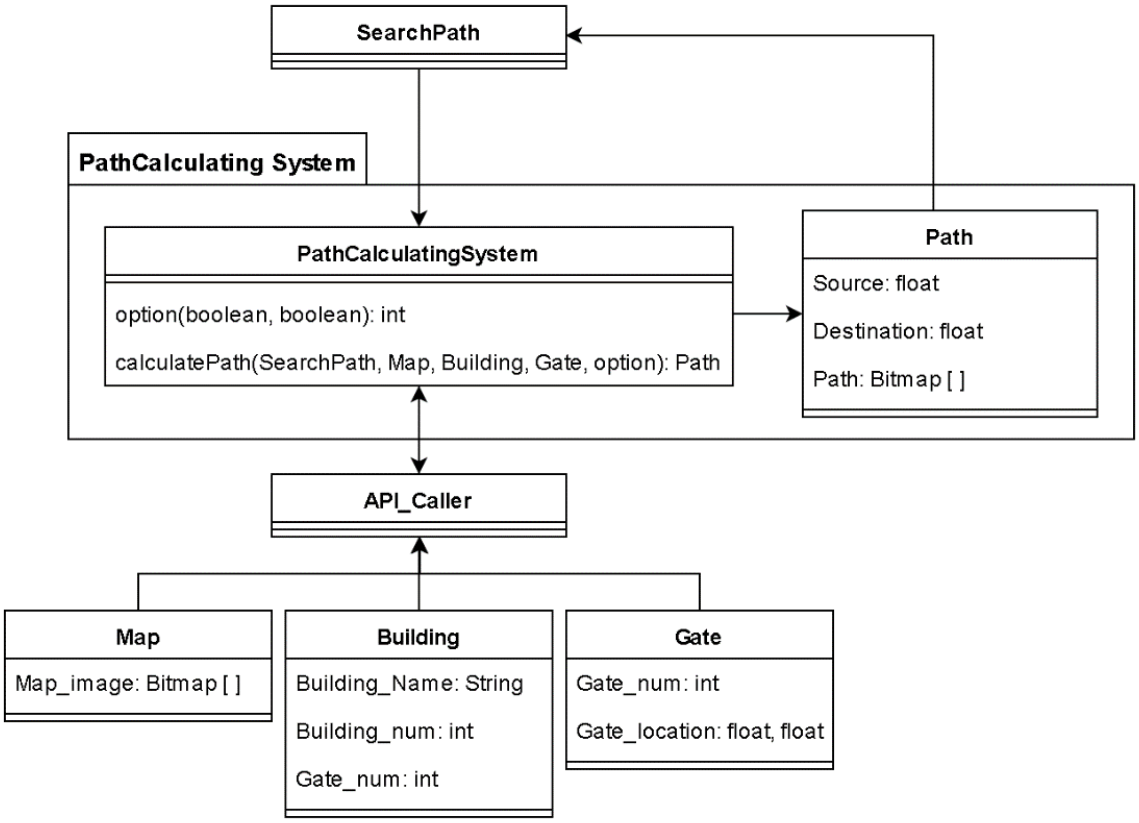
[Figure 16] Class diagram – Cloud functions

5.3.1.1. DB_Handler Class

데이터베이스와 연결되는 인터페이스이다. 보안을 위해 DBGateway를 통해 데이터를 주고받으며, 지도, 출입구 데이터베이스에 접근해 데이터의 생성, 수정, 삭제 등의 역할을 수행한다.

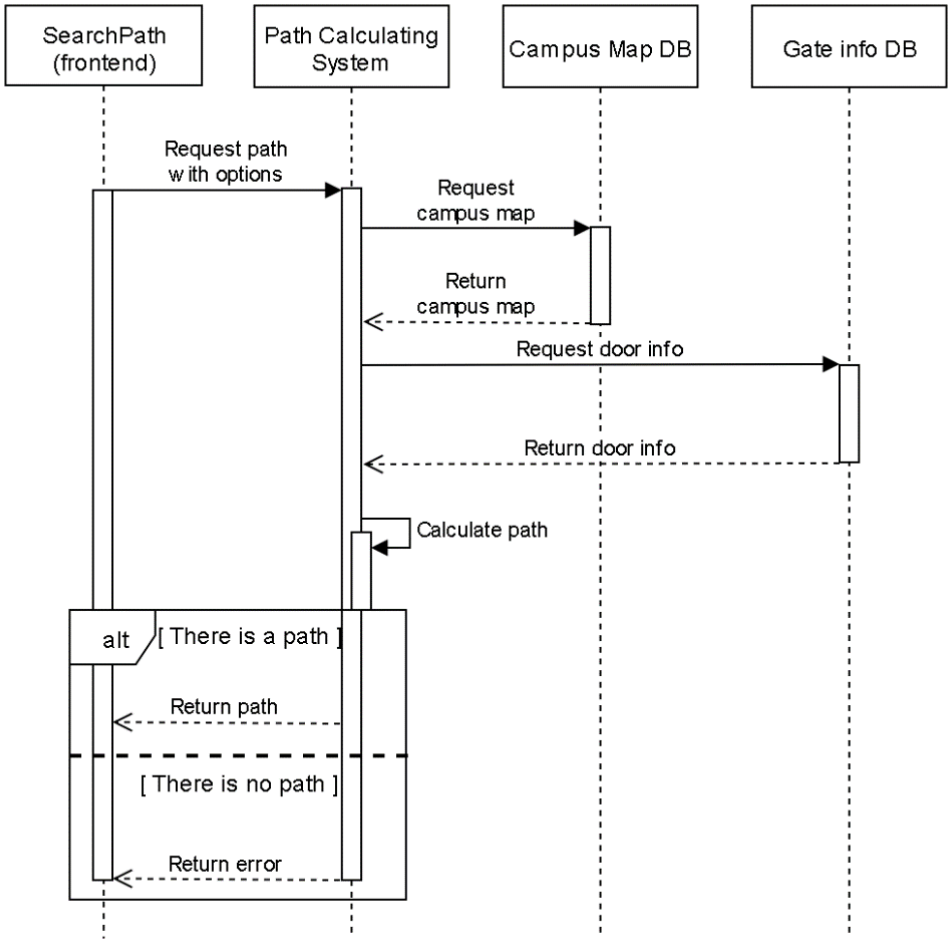
5.3.2. Path Calculating System

5.3.2.1. Class Diagram



[Figure 17] Class diagram – Path calculating system

5.3.2.2. Sequence Diagram

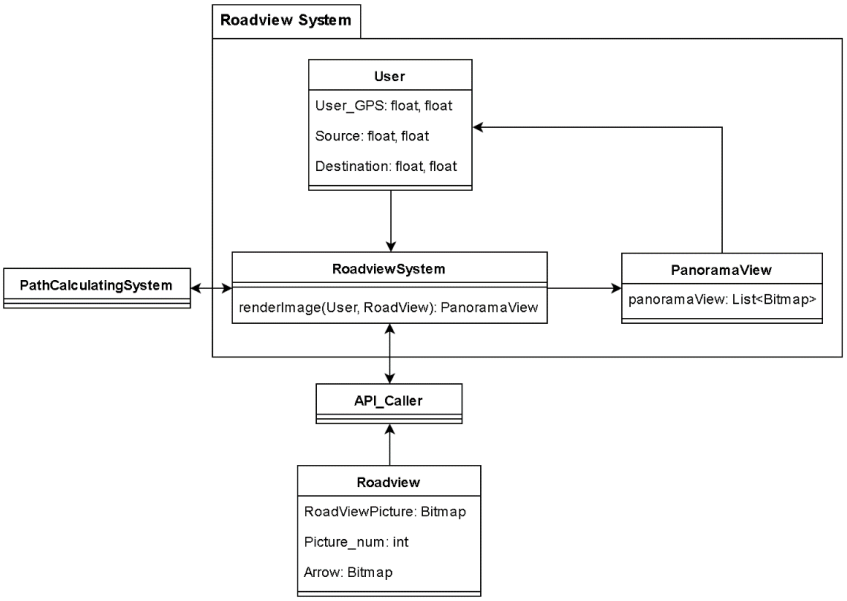


[Figure 18] Sequence diagram – Path calculating system

- **Description**
  - ✓ 프론트엔드로부터의 경로 탐색 요청을 처리하는 시스템이다. 자체 엔진으로 경로를 탐색하며, 그것을 다시 프론트엔드로 전달한다.
  - ✓ 코로나로 인해 강의실이 폐쇄중인 경우, 자동차 경로로 요청했으나 너무 짧은 경로인 경우 등과 같이 경로를 나타내기 어려울 때에는 에러 메세지를 전달한다.

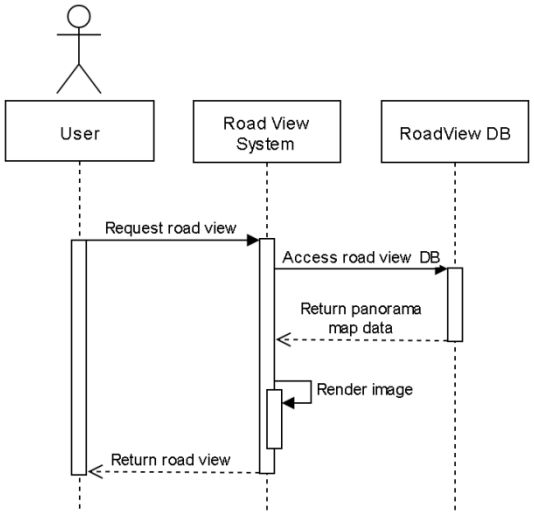
5.3.3. Roadview System

5.3.3.1. Class Diagram



[Figure 19] Class diagram – Roadview system

5.3.3.2. Sequence Diagram

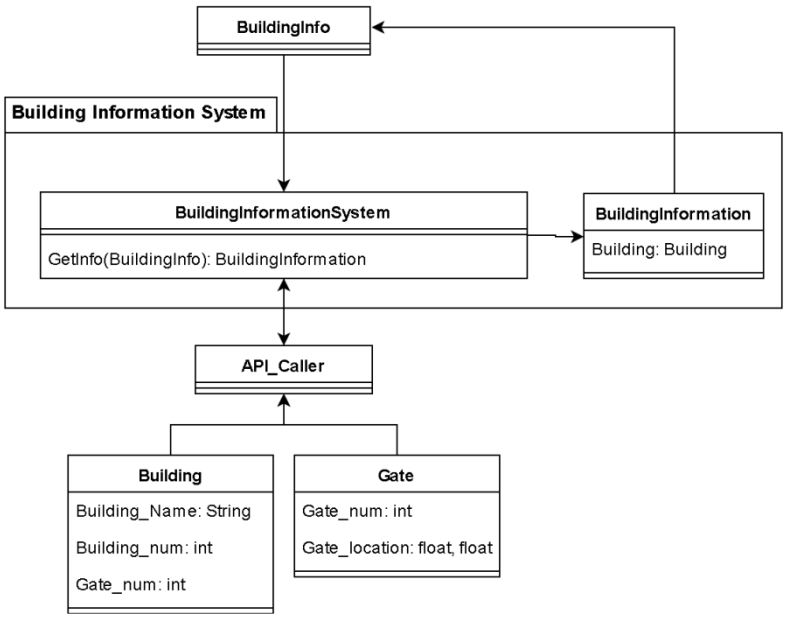


[Figure 20] Sequence diagram – Roadview system

- **Description**
  - ✓ 유저의 로드뷰를 요청하면 로드뷰 데이터베이스에 접근하여 유저가 요청한 지역의 로드뷰 데이터를 가져온다. 이를 렌더링하여 프론트엔드로 전달한다.

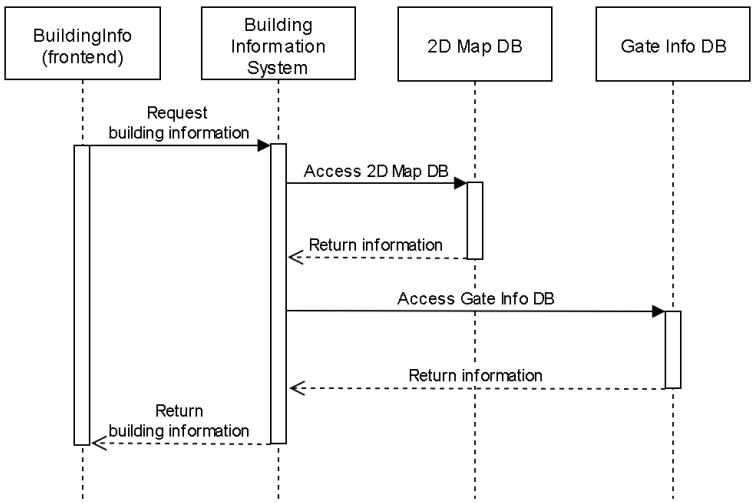
5.3.4. Building Information System

5.3.4.1. Class Diagram



[Figure 21] Class diagram – Building information system

5.3.4.2. Sequence Diagram



[Figure 22] Sequence diagram – Building information system

- **Description**
  - ✓ 건물, 건물 내부, 출입구의 정보에 대한 요청이 들어오면 데이터베이스에 접근하여 관련 정보를 가져오고, 이를 프론트엔드로 전달한다.

6. Protocol Design

6.1. Objectives

이 목차에서는 전체 System을 구성하는 Sub-system들 또는 Component들 사이에서 일어나는 상호작용에서 어떠한 Protocol이 쓰이는지에 대해서 다룬다. 또한 어떠한 데이터들이 어떻게 전송되는지 다룬다.

6.2. JSON

JSON은 JavaScript Object Notation의 약자로써, 기준 형식으로써, 사람이 읽을 수 있는 문자열로 키:값 대응을 이루고 있다. 데이터 교환에 자주 쓰이는 형식이다. Json을 바로 넣을 수 있는 DB도 존재하지만, DB구성상 관계형 데이터베이스에 데이터를 저장하기 때문에, Json 파일을 바로 넣을 수 없으므로, Parsing하는 과정을 거쳐서 입력해야 한다.

6.3. GPS Navigation System

6.3.1. Get Current GPS

- Request

[표 1] Table of Get Current GPS

Attribute	Detail	
Method	GET	
URI	/map/gps	
Header	Stations Address Array	Array containing nearby Base Station Address
Header	WIFI AP Address Array	Array containing nearby WIFI AP Address

- Response

[표 2] Table of Get Current GPS

Attribute	Detail	
Success Code	200 OK	
Failure Code	HTTP error code = 404 (Not Found)	
Success Response Body	GPS Info	GPS Object (current location, coordinates…)
Failure Response Body	Message	Request Error Message / Internal Server Error

6.3.2. Get Shortest Path

- Request

[표 3] Table of Get Shortest Path

Attribute	Detail	
Method	GET	
URI	/map/path	
Parameter	Source	Source Coordinates
Parameter	Destination	Destination Coordinates

- Response

[표 4] Table of Get Shortest Path

Attribute	Detail	
Success Code	200 OK	
Failure Code	HTTP error code = 404 (Not Found)	
Success Response Body	Path	Return Shortest Path
Failure Response Body	Message	Request Error Message / Internal Server Error

6.3.3. Get Location Info

- Request

[표 5] Table of Get Location Info

Attribute	Detail	
Method	GET	
URI	/map/location/info	
Parameter	Coordinates	Coordinates of Requested location

- Response

[표 6] Table of Get Location Info

Attribute	Detail	
Success Code	200 OK	
Failure Code	HTTP error code = 404 (Not Found)	
Success Response Body	Info	Info Object(Id, feature, …)
Failure Response Body	Message	Request Error Message / Internal Server Error

6.4. Road View System

6.4.1. Get Road View Data

- Request

[표 7] Table of Get Road View Data

Attribute	Detail	
Method	GET	
URI	/roadview	
Parameter	Location	Location you want to look up
Parameter	Direction	Specific direction you want to look up(ex. Front, Back, Left, Right)

- Response

[표 8] Table of Get Road View Data

Attribute	Detail	
Success Code	200 OK	
Failure Code	HTTP error code = 404 (Not Found)	
Success Response Body	Panorama Data	Rendered Map data (Data Rendered from Panorama Map Picture)
Failure Response Body	Message	Request Error Message / Internal Server Error

7. Database Design

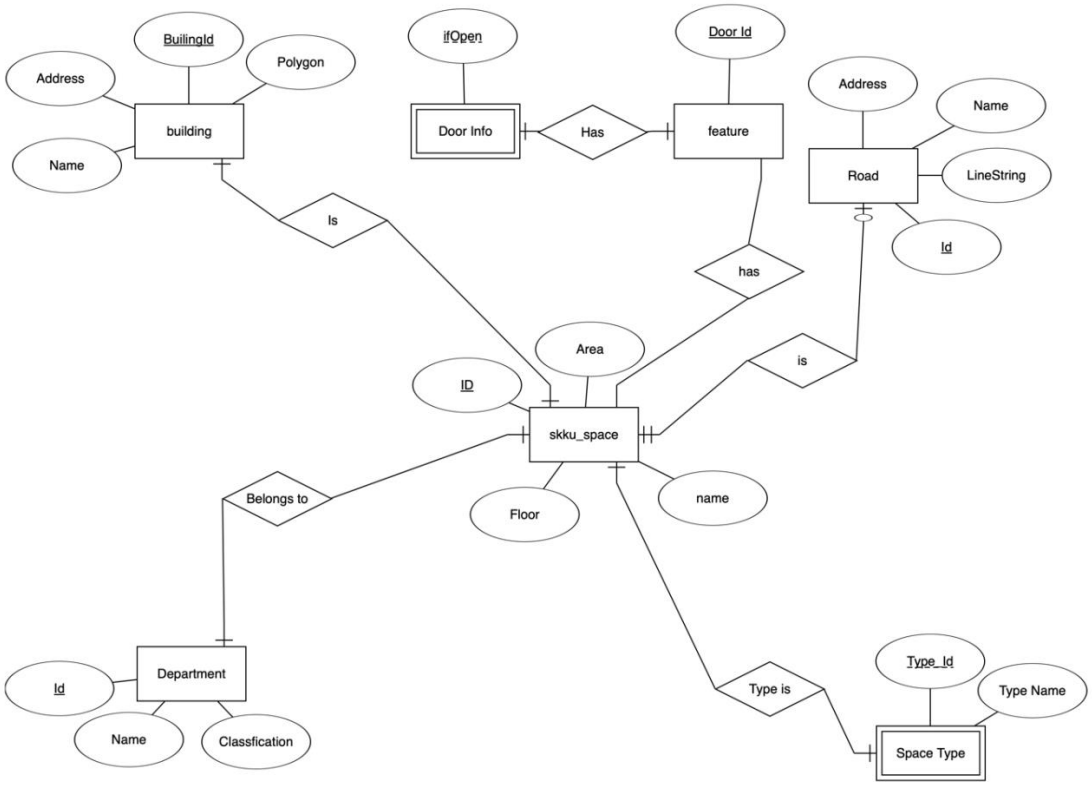
7.1. Objectives

이 부분에서는 System의 서비스들이 사용하는 Data 구조와 이것을 Database에 어떠한 구조로 저장 하는 지 기술한다. 다음과 같은 3단계로 진행된다.

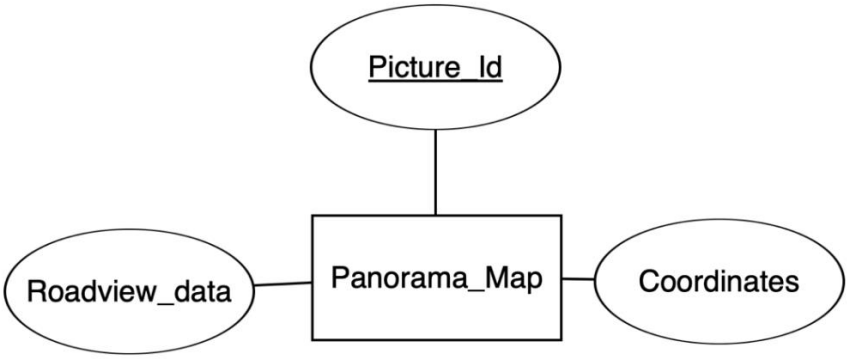
- 1 단계: ER-diagram (Entity Relationship diagram)으로 Entity 와 relationship 을 식별한다.
- 2 단계: 1 단계에서 식별한 ER-diagram 으로 Relational Schema 를 도출한다.
- 3 단계: SQL DDL(Data Description Language)을 통해서 DB 를 구성한다.

7.2. ER Diagram

이 시스템은 직사각형으로 표현된 8개의 entity를 가진다. Space라고 하는 총괄적인 entity가 존재 하고, Space Entity와 여러 관계를 갖는 5개의 Entity들(Building, Feature, Road, Space Type, Department)이 있다. 각 5개의 Entity들은 Space와 1대1관계를 가지기 때문에, two Crossed line로 구성하였다.(+ 모양) 또한 이외에 Road View를 지원하기 위해서 Panorama 사진을 저장하는 DB도 구성하였다. 각 entity에 속하는 attribute들은 타원으로 표시하였다.



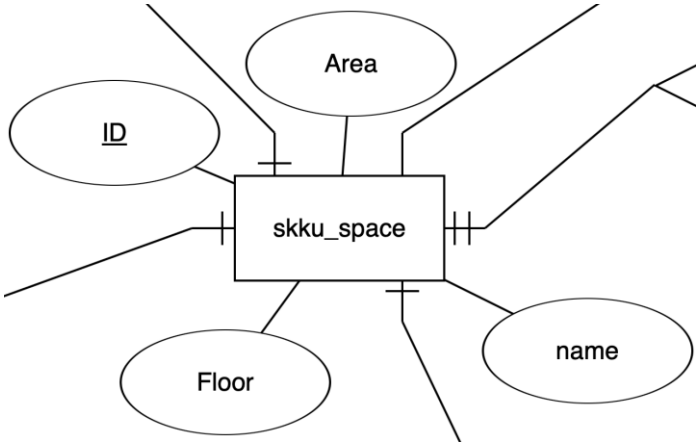
[Figure 23] ER-diagram1



[Figure 24] ER-diagram 2

7.2.1. Entities

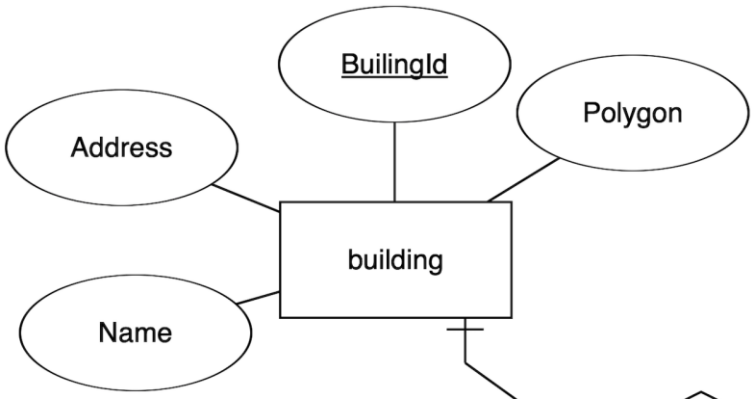
7.2.1.1. Space



[Figure 25] ER diagram, Entity, Space

Space Entity는 모든 종류의 공간을 이야기한다. Atrribute로 ID, Area, Floor, Name을 가지고 있다. ID Attribute는 Primary key로 작용을 한다. 다른 Entity들과 많은 관계를 가지면서 다양한 공간 정보들을 표현한다.

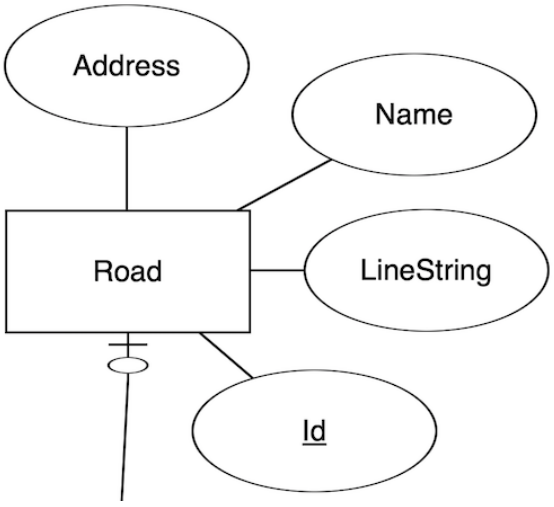
7.2.1.2. Building



[Figure 26] ER diagram, Entity, Building

Building Entity는 Space Entity들 중에서 건물에 대한 정보를 표현한다. 건물의 정보를 표현할 수 있는 Building_Id, Address, Name, Polygon 과 같은 정보들을 Attribute로 가지고 있다. 이때 이 Polygon은 건물의 위치정보를 포함하는 공간 데이터베이스의 Polygon 자료형을 이용한다.

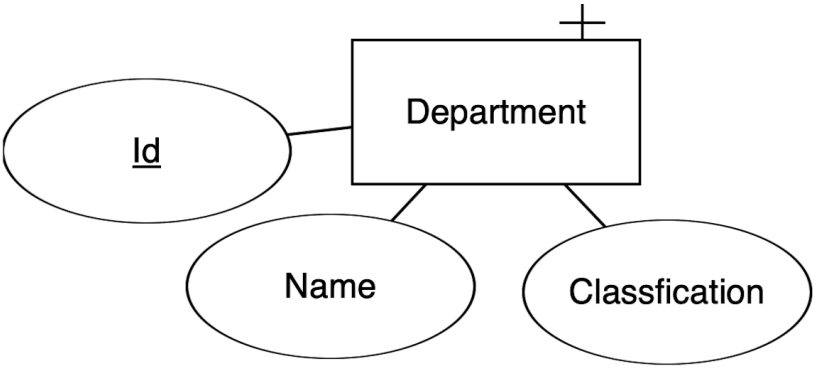
7.2.1.3. Road



[Figure 27] ER diagram, Entity, Road

Road Entity는 Space Entity들의 Tuple 중에서 도로에 대한 정보를 담고 있다. 도로의 정보를 표현할 수 있는 정보들을 Attribute로 포함한다. Id, Address, Name, LineString과 같은 정보들이다. 이때 LineString은 공간 데이터베이스의 자료형으로써, LineString 자료형을 사용한다.

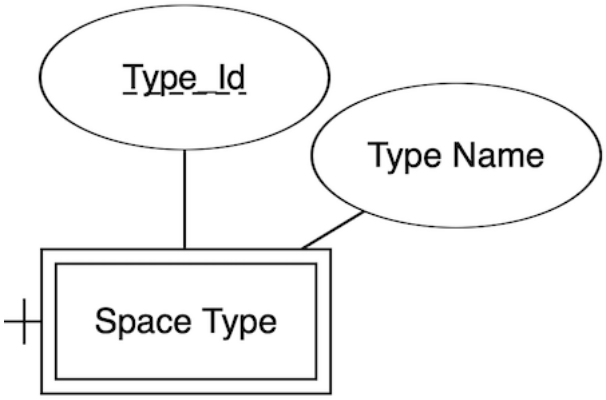
7.2.1.4. Department



[Figure 28] ER diagram, Entity, Department

Department Entity는 Space Entity들 중에서 관리 주체 또는 소속을 표시할 정보들을 담고 있다. Id, Name, Classfication이라는 Attribute를 가지며, 이때 Classfication은 다른 Entity를 참조하여, 관리 주체 또는 소속을 가리킨다.

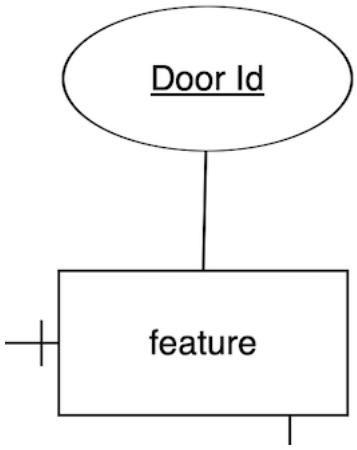
7.2.1.5. Space Type



[Figure 29] ER diagram, Entity, Space Type

Space Type Entity는 해당 공간의 Type을 명시한다. Attribute로 Type_id, Type name을 가진다.

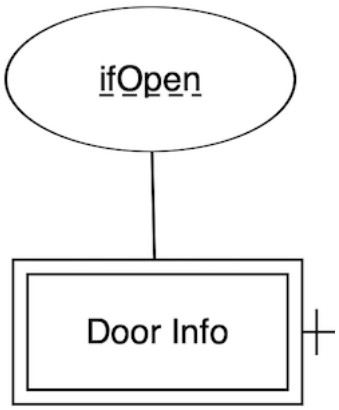
7.2.1.6. Feature



[Figure 30] ER diagram, Entity, Feature

Feature Entity는 해당 공간의 특징들을 명시한다. 2021 상반기에는 코로나로 인하여, 건물의 특정 출입구 개폐 여부를 판단할 수 있는 Entity로써 그 목적이 있다. 다른 특징들이 생긴다면(ex. 도로 공사로 인해 도로 우회중인 정보), 새로운 특징들을 Attribute로 추가할 수 있다.

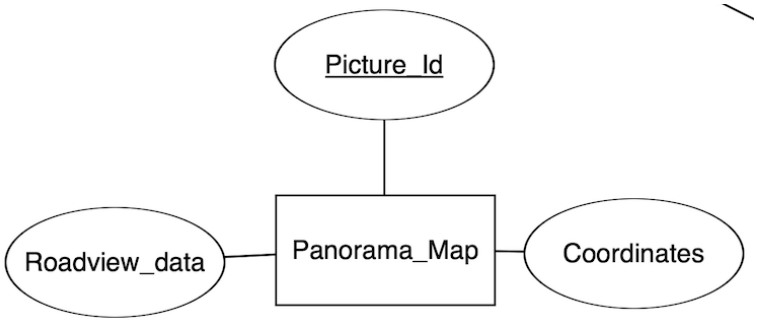
7.2.1.7. Door Info



[Figure 31] ER diagram, Entity, DoorInfo

Door Info Entity는 Weak Entity로써, 2021 상반기에는 코로나로 인하여, 건물의 특정 출입구 개폐 여부를 판단할 수 있는 Entity로 목적이 있다.

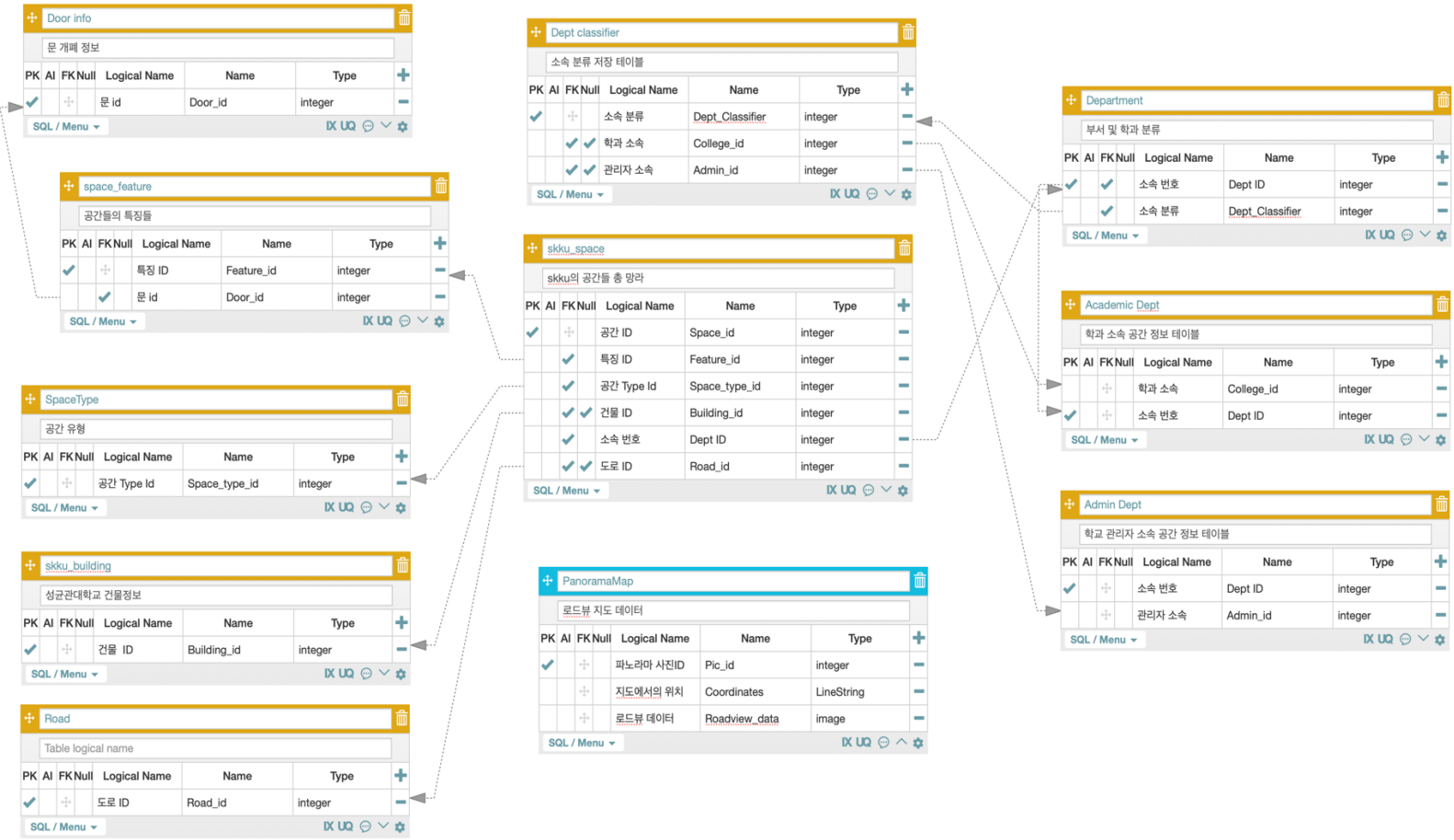
7.2.1.8. Panorama_Map



[Figure 32] ER diagram, Entity, Panorama_Map

Panorama_Map Entity는 로드뷰 기능 제공 시에 필요한 로드뷰 사진들을 표현한다. 해당 사진 데이터들은 Picture_id, RoadView_data, Coordinates를 Attribute로 사용하여서 표현한다.

7.3. Relational Schema



[Figure 33] Relational Schema

7.4. SQL DDL

7.4.1.SKKU_Space

```
CREATE TABLE skku_space
(
    Space_id         integer        NOT NULL,
    Space_name       varchar(50)    NOT NULL,
    Area             varchar(50)    NOT NULL,
    Floor            varchar(50)    NULL,
    Feature_id       integer        NOT NULL,
    Space_type_id    integer        NOT NULL,
    Building_id      integer        NULL,
    Dept ID          integer        NOT NULL,
    Road_id          integer        NULL,
    CONSTRAINT  PRIMARY KEY (Space_id)
);


ALTER TABLE skku_space
    ADD CONSTRAINT FK_skku_space_Building_id_skku_building_Building_id FOREIGN
KEY (Building_id)
        REFERENCES skku_building (Building_id);


ALTER TABLE skku_space
    ADD  CONSTRAINT  FK_skku_space_Feature_id_space_feature_Feature_id  FOREIGN
KEY (Feature_id)
        REFERENCES space_feature (Feature_id);


ALTER TABLE skku_space
    ADD CONSTRAINT FK_skku_space_Dept ID_Department_Dept ID FOREIGN KEY (Dept
ID)
        REFERENCES Department (Dept ID);


ALTER TABLE skku_space
    ADD CONSTRAINT FK_skku_space_Space_type_id_SpaceType_Space_type_id FOREIGN
KEY (Space_type_id)
```

```
        REFERENCES SpaceType (Space_type_id);


ALTER TABLE skku_space
    ADD CONSTRAINT FK_skku_space_Road_id_Road_Road_id FOREIGN KEY (Road_id)
        REFERENCES Road (Road_id);
```

7.4.2. Space_Type

```
CREATE TABLE SpaceType
(
    Space_type_id      integer        NOT NULL,
    Space_type_name    varchar(50)    NOT NULL,
    CONSTRAINT  PRIMARY KEY (Space_type_id)
);
```

7.4.3. Space_Feature

```
CREATE TABLE space_feature
(
    Feature_id    integer    NOT NULL,
    Door_id       integer    NOT NULL,
    CONSTRAINT  PRIMARY KEY (Feature_id)
);

ALTER TABLE space_feature
    ADD CONSTRAINT FK_space_feature_Door_id_Door info_Door_id FOREIGN KEY
(Door_id)
        REFERENCES Door info (Door_id);
```

7.4.4. Building

```
CREATE TABLE skku_building
(
    Building_id         integer         NOT NULL,
    Building_address    varchar(128)    NOT NULL,
    Building_name       varchar(40)     NOT NULL,
    Building_polygon    polygon         NOT NULL,
    CONSTRAINT  PRIMARY KEY (Building_id)
);
```

7.4.5. Road

```
CREATE TABLE Road
(
    Road_id            integer          NOT NULL,
    Road_linestring    LineString       NOT NULL,
    Road_address       varchar(128)     NOT NULL,
    Road_name          varchar(128)     NOT NULL,
    CONSTRAINT  PRIMARY KEY (Road_id)
);
```

7.4.6. Dept_classifier

```
CREATE TABLE Dept classifier
(
    Dept_Classifier    integer    NOT NULL,
    College_id         integer    NULL,
    Admin_id           integer    NULL,
    CONSTRAINT  PRIMARY KEY (Dept_Classifier)
);

ALTER TABLE Dept classifier
    ADD CONSTRAINT FK_Dept classifier_College_id_Academic Dept_College_id
FOREIGN KEY (College_id)
        REFERENCES Academic Dept (College_id);

ALTER TABLE Dept classifier
    ADD CONSTRAINT FK_Dept classifier_Admin_id_Admin Dept_Admin_id FOREIGN KEY
(Admin_id)
        REFERENCES Admin Dept (Admin_id);
```

7.4.7. Department

```
CREATE TABLE Department
(
    Dept ID            integer        NOT NULL,
    Dept_name          varchar(50)    NOT NULL,
    Dept_Classifier    integer        NOT NULL,
    CONSTRAINT  PRIMARY KEY (Dept ID)
);

ALTER TABLE Department
    ADD CONSTRAINT FK_Department_Dept_Classifier_Dept
classifier_Dept_Classifier FOREIGN KEY (Dept_Classifier)
        REFERENCES Dept classifier (Dept_Classifier);

ALTER TABLE Department
    ADD CONSTRAINT FK_Department_Dept ID_Academic Dept_Dept ID FOREIGN KEY
(Dept ID)
        REFERENCES Academic Dept (Dept ID);
```

7.4.8. Academic Dept

```
CREATE TABLE Academic Dept
(
    College_id    integer    NOT NULL,
    Dept ID       integer    NOT NULL,
    CONSTRAINT  PRIMARY KEY (Dept ID)
);
```

7.4.9. Admin_Dept

```
CREATE TABLE Admin_Dept
(
    Dept ID     integer    NOT NULL,
    Admin_id    integer    NOT NULL,
    CONSTRAINT  PRIMARY KEY (Dept ID)
);
```

7.4.10. Door_Info

```
CREATE TABLE Door info
(
    Door_id      integer    NOT NULL,
    Door_open    boolean    NULL,
    CONSTRAINT  PRIMARY KEY (Door_id)
);
```

7.4.11. Panorama_Map

```
CREATE TABLE PanoramaMap
(
    Pic_id           integer       NOT NULL,
    Coordinates      LineString    NOT NULL,
    Roadview_data    image         NOT NULL,
    CONSTRAINT  PRIMARY KEY (Pic_id)
);
```

8. Testing Plan

8.1. Objectives

이 문단에서는 소프트웨어 개발 및 배포 단계에서 필요한 여러 테스트에 대해 설명한다. 수행하는 테스트는 크게 4개로, **컴포넌트 테스팅, 시스템 테스팅, 인수 테스팅, 릴리즈 테스팅**이다. 이 테스트들은 공통적으로 소프트웨어의 에러나 결점을 미리 발견하여 수정할 수 있게 만들어 준다. 그래서 궁극적으로, 안정적이고 믿을만한 소프트웨어를 시장 및 소비자에게 배포할 수 있게 해준다.

8.2. Testing Policy

이 문단에서는 문단 8.1에서 언급한 4가지 테스트에 대해 설명하고자 한다. 이 중에서 컴포넌트 테스팅, 시스템 테스팅, 인수 테스팅은 개발 단계에서 이루어진다.

8.2.1. Component Testing

컴포넌트라는 것은 소프트웨어의 기본 단위로, 서로 관련이 있는 함수 혹은 객체들을 모아놓은 것을 말한다. 컴포넌트 테스팅이란 개별 컴포넌트를 다른 컴포넌트와 합치지 않고 독립적으로 테스트를 수행하는 것을 말한다. 이 테스트를 수행하면 소프트웨어가 더 복잡해지기 전에 버그를 빠르게 찾을 수 있어 개발 비용을 줄이는데 큰 도움이 된다.

8.2.2. System Testing

시스템 테스팅은 마지막 컴포넌트까지 통합하고 나서 만들어진 최종적인 시스템을 테스트하는 것이다. 이때 신뢰성(reliability), 성능(performance), 가용성(availability) 등 창발적 속성(emergent property)이 잘 드러나는지 테스트하는 것이 특히 중요하다. 또한 시스템의 기능을 꼼꼼하게 테스트할 수 있도록, 실제로 시스템에서 처리될 것으로 예상되는 데이터를 토대로 테스트 케이스가 작성되어야 한다.

8.2.3. Acceptance Testing

인수 테스팅은 시스템이 요구사항 명세서에서 제시된 사용자의 요구를 만족하는지 테스트하는 과정이다. 이 테스트는 실제 사용자의 데이터를 가지고 진행되어야 한다.

8.2.4. Release Testing

많은 시간과 비용을 들여서 만든 소프트웨어를 실제로 시장에 배포하고, 소비자가 사용할 수 있게 하는 것은 전체 프로세스 중에서도 꽤 중요한 단계라고 할 수 있다. 지금까지의 테스트에서 문제가 없었더라도, 실제로 소프트웨어를 배포했을 때 예상치 못한 문제점이 생길 수도 있다. 릴리즈 테스팅은 새로운 버전의 소프트웨어가 출시되기 전에 문제점이나 결점이 없는지 테스트를 하는 것을 의미한다. 그리하여 소프트웨어가 출시되었을 때, 문제점이 없도록 하는 것이 목적이 된다.

또한 소프트웨어 사용자들에게 주기적으로 설문조사 혹은 리뷰를 요청해야 한다. 실제로 소프트웨어를 사용하는 유저들이 개발 단계에서는 몰랐던 문제점이나 아이디어를 발견할 수 있기 때문이다.

9. Development Plan

9.1. Objectives

이 문단에서는 소프트웨어 개발에 사용되는 여러 기술 및 개발환경에 대해 설명한다.

9.2. Frontend Environment

9.2.1. Flutter

[Figure 34] 플러터 로고

플러터는 크로스 플랫폼 앱개발을 지원하는 UI 프레임워크로, 이것으로 컴파일된 프로그램은 안드로이드와 IOS 기기에서 동시에 실행이 가능하다. 플러터에 사용되는 언어는 구글에 의해 만들어진 다트(Dart)라는 언어다. 플러터는 UI를 자체 렌더엔진인 스키아(Skia)로 직접 렌더링하여 뛰어난 성능을 보여준다. 여기서 플러터의 목적은 앱의 전체적인 인터페이스를 제작하는 것으로, 모든 유저의 입출력이 플러터를 거쳐가게 된다.

9.2.2. Adobe Photoshop

[Figure 35] 어도비 포토샵 로고

어도비 포토샵은 이미지의 합성과 편집에 특화된 프로그램으로, 픽셀을 기본 단위로 하는 비트맵 방식을 사용한다. 포토샵을 이용하여, 소프트웨어에 사용되는 직관적이면서도 아름다운 로고 및 아이콘을 디자인한다.

9.2.3. Adobe Xd

[Figure 36] 어도비 Xd 로고

어도비 Xd는 UI/UX 디자인에 쓰이는 프로그램으로, 디자인에서 바로 프로토타입을 만들고 공유할 수 있어 원활한 커뮤니케이션을 돕고 빠른 소프트웨어 생산에 기여한다. 어도비 Xd를 이용하여, 유저가 어렵지 않게 프로그램을 사용할 수 있도록 직관적이고 깔끔한 UI/UX를 디자인한다.

9.3. Backend Environment

9.3.1. Firebase

[Figure 37] 파이어베이스 로고

파이어베이스는 고품질의 모바일 앱 및 웹사이트를 쉽게 만들 수 있게 도와주는 개발 플랫폼으로, 매우 다양한 기능을 제공한다. 대표적으로 로그인 인증, 클라우드 스토리지, 실시간 처리가 가능한 데이터베이스, 호스팅 등 백엔드 개발에 필요한 기능을 제공한다. 특히 실시간 데이터베이스를 통해, 모든 클라이언트에서 데이터가 동기화되어 안정적인 기능 제공이 가능하다.

9.3.2 Github

[Figure 38] 깃허브 로고

깃허브는 코드 버전 관리 및 협업을 지원하는 웹호스팅 서비스다. 즉, 하나의 프로젝트를 여러 팀원과 수행할 때 안정적인 협업을 도움으로써 서로 다른 소프트웨어 컴포넌트 간의 원활한 통합을 돕는다.

9.3.3 Docker

[Figure 39] 도커 로고

도커는 컨테이너를 기반으로 한 오픈소스 가상화 플랫폼이다. 도커는 다양한 프로그램 및 실행 환경을 하나의 컨테이너로 추상화한다. 대부분의 소프트웨어 서비스 환경은 컨테이너로 추상화될 수 있으며, 이들은 AWS, Google Cloud 등 어디에서든 도커만 설치되어 있다면 실행이 가능하다.

9.4. Constraints

이 시스템은 이 문서에 언급된 컨텐츠를 바탕으로 디자인, 구현된다. 세부적인 디자인, 구현은 개발자에 의해 정해진다. 다음의 사항들은 세부적인 디자인, 구현에서의 제약 사항들이다.

- 경로 탐색의 경우 5초 이상 걸리면 안된다.
- 비용(저작권 등)을 지불해야 하는 소프트웨어는 가능하다면 사용을 피한다.
- 가능하다면 오픈소스 소프트웨어를 사용한다.
- 시스템 비용과 유지 비용을 고려한다.
- 유저 친화적인 소프트웨어를 개발한다.
- 미래의 기술 발전을 고려하여 확장성을 높인 시스템을 개발한다.
- 개발 환경 OS는 윈도우 10 이다.
- 개발 툴은 안드로이드 스튜디오 4.1.3 버전과 Xcode 12.5 버전이다.
- 구동을 위해 필요한 최소 안드로이드 버전은 Android version 6.0 (API 23), iOS 버전은 iOS 10이다.
- 테스트를 위한 에뮬레이터 버전은 Android version 10 (API 29)이다.

9.5. Assumptions and Dependencies

이 문서는 모든 시스템이 안드로이드, ios 디바이스 및 오픈 소스를 이용하여 디자인, 구현된다는 가정을 토대로 작성되었다. 구동을 위해 필요한 버전은 안드로이드의 경우 Android version 6.0(API 버전 23) 이상, ios의 경우 ios 10 이상이다. 테스트에 쓰이는 에뮬레이터의 버전은 Android version 10(API 버전 29)다. 이 외의 다른 운영체제나 버전에서는 정상적으로 작동하지 않을 수 있다.

10. Supporting Information

10.1. Software Design Specification

이 소프트웨어 디자인 명세서는 IEEE Recommendation (IEEE Recommended Practice for Software Design Description, IEEE-Std-1016)에서 제시하는 기준을 준수하여 작성되었다.

10.2. Document History

[표 9] 문서 히스토리

일자	버전	설명	작성자
2021.05.06	0.1	문서 디자인, 목차	천주형
2021.05.12	1.0	8, 9 추가	천주형
2021.05.13	1.1	1, 2, 3 추가	김규용
2021.05.13	1.2	4.1, 4.2.1, 4.2.2, 4.2.3 추가	신승환
2021.05.13	1.3	4.2.4, 4.2.5 추가	한지명
2021.05.14	1.4	5 추가	김승호
2021.05.14	1.5	6, 7 추가	황석진
2021.05.15	1.6	7 수정	황석진
2021.05.15	2.0	모든 항목 병합	천주형
2021.05.15	2.1	모든 파트 검토	팀원 전원
2021.05.16	2.2	10 추가	천주형