

# K-Software Empowerment Boot Camp

## AI Project

### 프로젝트 결과 보고서

#### AI 도로 시설물 파손 감지 및 관리 사용자 서비스

팀명 : SAVEWAY

#### 팀 구성 및 역할

순번	이름	역할	담당업무	학교
1	팽준호	팀장	앱 개발, AI 모델	인하대학교
2	김재영	팀원	데이터 처리, AI 모델	성균관대학교
3	박근태	팀원	데이터 처리, AI 모델	인하대학교
4	오수만	팀원	BackEnd, AI 모델	인하대학교
5	최민혁	팀원	BackEnd, AI 모델	인하대학교
6	정지연	팀원	FrontEnd, 데이터 처리	성균관대학교
7	박서영	팀원	FrontEnd, PM	인하대학교

# 목차

<b>I. 초록</b>	..... p.3
<b>II. 개발 배경</b>	..... p.4
<b>III. 개발 계획</b>	..... p.6
1. 예상 프로세스	
2. 데이터셋 구축	
3. 서비스 구현	
A. Application	
B. Back-End	
C. Web Page	
<b>IV. 개발 내용</b>	..... p.11
1. 프로젝트 구조	
2. 데이터셋 구축	
3. AI 모델 - YOLOV8-S	
4. AI 모델 - Real-ESRGAN	
5. AI 모델 - EfficientNet	
6. Application Service	
7. Back-End	
8 .Web-Page Service	
<b>V. 결과</b>	..... p. 39
1. 서비스 결과	
2. 평가 의견 및 느낀점	

# I. 초록

## 1. 문제 제기

한국에서는 도로 파손, 특히 포트홀 문제로 인해 차량 손상 및 사고가 증가하고 있으며, 이는 운전자의 수리비 부담과 안전 위협을 초래하고 있다. 도로 시설물(시선유도봉, 라바콘, PE 안내봉 등)의 파손 또한 운전자의 안전을 위협하며, 이를 신속히 인식하고 수리, 제거, 교체하는 과정이 필요하다. 이와 함께 복잡한 피해 보상 절차가 운전자와 도로 관리 기관에 큰 부담을 주고 있다.

## 2. 프로젝트 목표

본 프로젝트의 목표는 도로 및 도로 시설물의 파손을 실시간으로 감지하고 관리자가 신속하게 대응할 수 있는 스마트폰 앱 기반 도로 파손 모니터링 시스템을 개발하는 것이다. 이 시스템은 AI 기술을 활용하여 도로 파손 및 시설물 파손을 자동으로 감지하고, 감지된 파손사진, 위치 정보 데이터를 실시간으로 수집하여 도로 상태를 신속하게 파악할 수 있도록 지원한다.

## 3. 개발 내용

어플리케이션은 YOLOv8 모델을 사용하여 도로 파손물을 실시간으로 감지하고, 서버로 해당 데이터를 전송한다. 서버는 Real-ESRGAN 을 통해 이미지 화질을 개선하고, EfficientNet 을 사용하여 도로 파손의 종류를 분류한다. 수집된 데이터는 데이터베이스에 저장되며, 관리자는 웹 페이지를 통해 데이터베이스의 정보를 열람하여 파손물의 위치, 감지 일자, 사진 데이터 등을 확인하고, 지도에서 파손물의 정보를 시각적으로 확인할 수 있다.

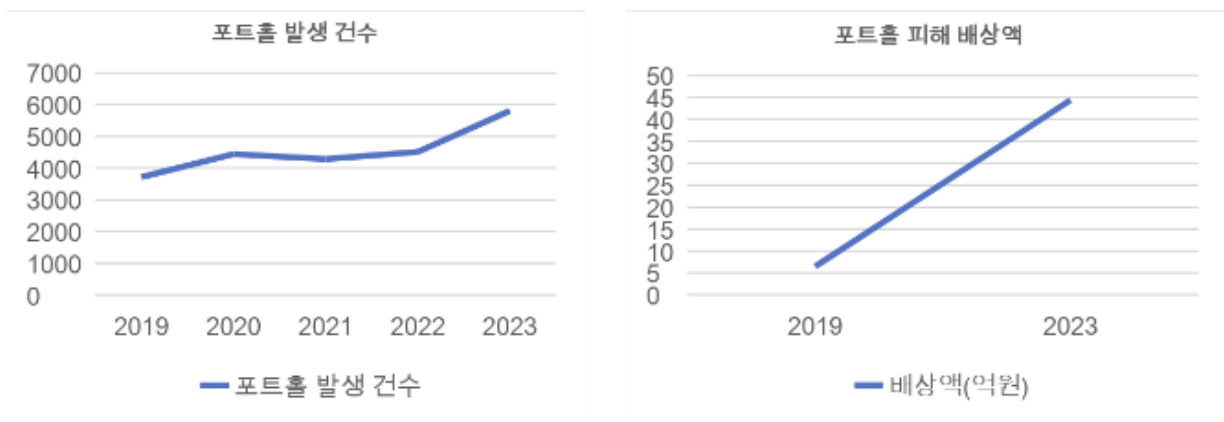
## 4. 결과 및 기대 효과

이 시스템은 도로 관리의 효율성을 높이고, 지자체 및 민간 차량의 피해 보상 문제를 완화하여 운전자의 안전을 강화할 것으로 기대할 수 있다. 또한, 다양한 도로 파손물에 대한 신속한 감지와 대응을 통해 정보 기관 및 지자체의 도로 관리 효율성을 증대시킬 것으로 예상된다.

## II. 개발 배경

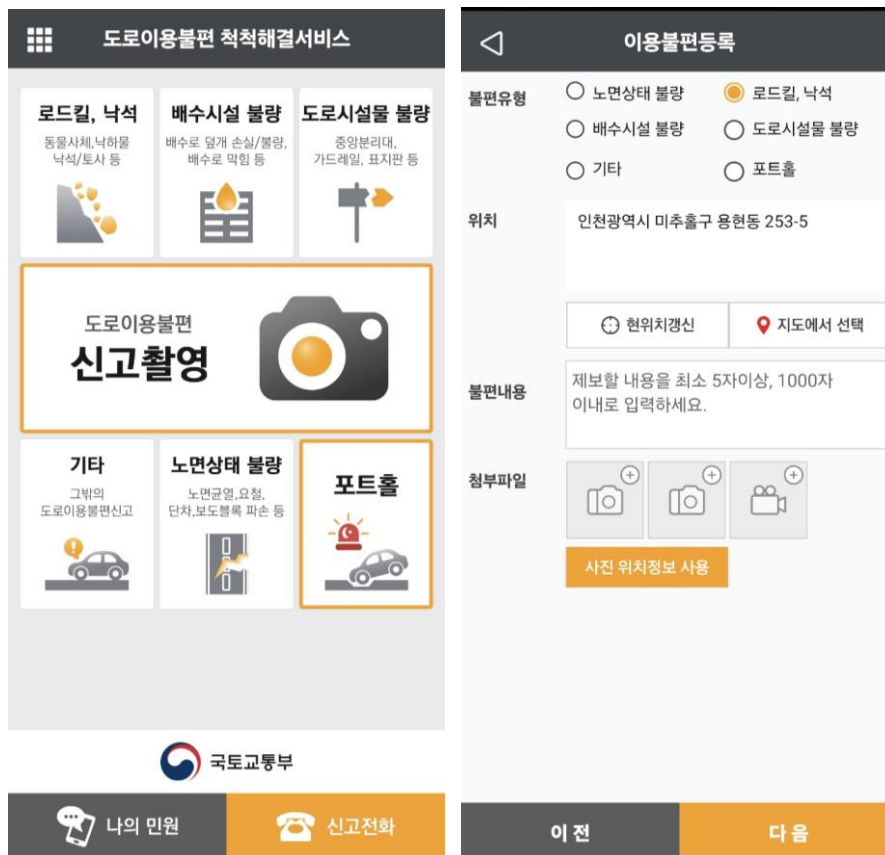
### 1. 도로 파손 문제의 심각성

최근 한국에서는 도로 파손, 특히 포트홀과 같은 문제로 인해 차량 손상과 사고가 빈번하게 발생하고 있다. 이상기후로 인한 잦은 비와 기타 기상 변화로 도로 파임 피해가 급증하면서 운전자의 안전을 크게 위협하고 있는 상황이다. 이로 인해 관련 민원이 급증하여 지자체는 큰 곤란을 겪고 있으며, 민원이 접수될 때마다 즉시 보수 작업을 시도하지만, 포트홀의 증가 속도를 따라가지 못해 피해가 계속 확대되고 있다. 포트홀 외에도 시선유도봉, 라바콘, PE 안내봉 등의 도로 시설물의 파손이 증가하고 있으며, 이는 운전자의 안전을 더욱 위협하고 있다. 따라서 도로 파손물에 대한 신속한 수리나 제거가 필요하며, 이를 위해 정확하고 빠른 대응이 필수적이다.



### 2. 기존 도로 관리 시스템의 한계

기존의 도로 관리 시스템은 도로 파손을 신속하게 감지하고 대응하는 데 한계를 보이고 있다. 도로 파손물이 발생한 후 이를 발견하고 수리까지 이루어지는 과정은 시간과 비용이 많이 소요되고, 이로 인해 도로 이용자들은 지속적인 불편을 겪으며, 피해 보상 절차 또한 복잡하고 시간이 오래 소요된다. 특히 도로 관리 기관과 지자체는 도로 파손물의 효율적인 관리와 수리에 있어 어려움을 겪고 있다. 민원 처리 방식으로는 파손물의 위치를 효율적으로 관리하기 어렵고, 도로 파손물을 직접 탐지할 경우에는 파손물의 종류와 위치를 직접 기록해야 하는 수고스러움이 존재한다. 현재 국토교통부의 "도로 이용 불편 척척해결 서비스" 앱은 여러 종류의 불편 사항을 버튼으로 구분하여 신고하지만, 이 과정이 번거롭고 실사용에서의 신속한 신고가 어려운 문제를 가지고 있다고 한다. 따라서 이러한 단점을 개선할 수 있는 사용자 중심의 도로 관리 시스템의 필요성이 부각되고 있고, 따라서 본 팀은 SAVEWAY 프로젝트를 통하여 신속하고 편리한 도로 관리 시스템을 개발하고자 한다.



국토교통부에서 관리하는 "도로 이용 불편 척척해결 서비스" 앱 화면

### 3. 인공지능을 활용한 도로 파손 모니터링의 필요성

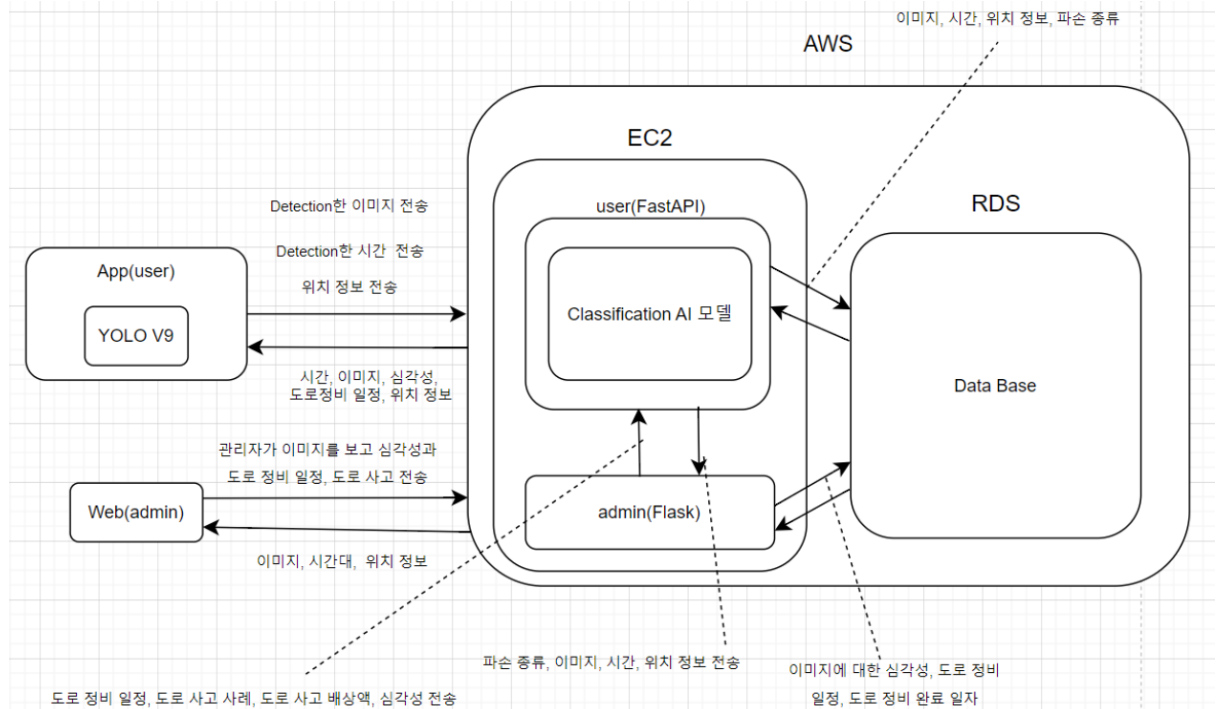
이러한 문제를 해결하기 위해 인공지능(AI)을 활용한 실시간 도로 파손 모니터링 시스템의 필요성이 대두되었다. AI 기술을 활용하면 번거로운 절차 없이 도로 파손물의 자동 감지가 가능해지며, 이를 통해 관리자는 신속하게 대응할 수 있다. 또한 AI 기반의 분류 시스템을 통해 파손물의 종류를 구분하고, 수집된 데이터를 바탕으로 효율적인 도로 관리가 가능하다. 도로 파손물로 인한 피해보상을 위한 신고 시에도 GPS 값과 파손정보가 연동되어 전송되기 때문에 지자체에서 보상을 위한 검토 시 진위여부 확인에 용이할 것으로 예상된다.

### 4. 본 프로젝트의 목표와 방향성

본 프로젝트는 도로 파손물의 실시간 감지와 신속한 대응을 목표로 하는 스마트폰 앱 기반의 모니터링 시스템을 개발하는 데 중점을 두고 있다. YOLOv8 모델을 활용하여 도로 파손물을 실시간으로 감지하고, Real-ESRGAN 을 통해 이미지 화질을 개선하며, EfficientNet 을 사용해 파손물을 분류한다. 수집된 데이터를 서버로 전송하여 데이터베이스로 관리하며, 관리자가 이를 쉽게 확인하고 도로 파손물의 위치, 감지일자, 사진데이터 등을 종합적으로 관리할 수 있는 웹 페이지를 제공한다. 이러한 시스템을 통해 도로관리의 효율성을 극대화하고 운전자들의 안전 강화를 목표로 한다.

## Ⅲ. 개발 계획

### 1. 예상 프로세스



#### ● 전체 설계

아래 세 파트로 나누어 설계하였다.

##### 1) 핸드폰 어플리케이션 | App(user)

- 어플리케이션 상에서 YOLOv9 을 이용하여 도로 파손물을 감지한다. (최종적으로 YOLOv8 사용)
- 감지한 이미지, 감지한 시간, 위치 정보를 서버로 전송한다.

##### 2) 서버 | AWS

- 어플리케이션에서 전송한 데이터를 EC2 에서 Bounding Box Crop 을 진행한다.
- FastAPI 에서 Classification AI 모델을 사용하여 파손 종류 구분한다.
- DataBase 에 앱, 웹을 통해 들어온 데이터 저장한다.

##### 3) 웹 페이지 | Web(admin)

- 관리자가 AWS 서버에서 받은 데이터를 확인하고 검토한다.
- 관리자가 검토하여 수정 및 삭제한 데이터를 서버로 전송한다.

## 2. 데이터셋 구축

도로 시설물 파손 감지를 위해 AI에 학습시킬 파손된 도로 시설물 이미지 수집이 필요하다. 수집할 이미지 데이터 구분, 이미지 수집 방법과 수집된 이미지 데이터 정제 방법은 아래와 같다.

### ● 이미지 데이터 구분

포트홀, 도로 시설물 (시선유도봉, PE 안내봉, 라바콘, 방호벽)

### ● 이미지 데이터 수집 방법

1) AI 통합 플랫폼 이용 : AI-Hub, kaggle 등 AI 통합 플랫폼의 공개 데이터를 이용하여 도로 파손물 이미지 데이터를 수집한다.

2) 웹 크롤링 : Bing, Google 등에서 웹 크롤링을 통해 이미지 데이터를 수집한다.

### ● 이미지 데이터 관리

1) 사용 플랫폼 : Roboflow

2) 이미지 데이터 관리 : 파손물 종류에 따라 클래스 별로 구분하여 관리한다. 클래스 별 이미지 데이터를 수집한 후 데이터 라벨링을 진행하고 Data Augmentation(데이터 증강)을 사용하여 강건한 데이터셋을 구축한다.

## 3. 서비스 구현

### 3-1. Application

#### ● App 서비스 개발 목적

1) 도로 파손 정보의 실시간 전달: 사용자들이 스마트폰 앱을 통해 도로 파손 정보를 쉽고 빠르게 수집하고 전달할 수 있도록 한다. 이를 통해 도로 상태에 대한 모니터링이 가능해져, 도로 관리자가 빠르게 파손을 파악하고 대응할 수 있다.

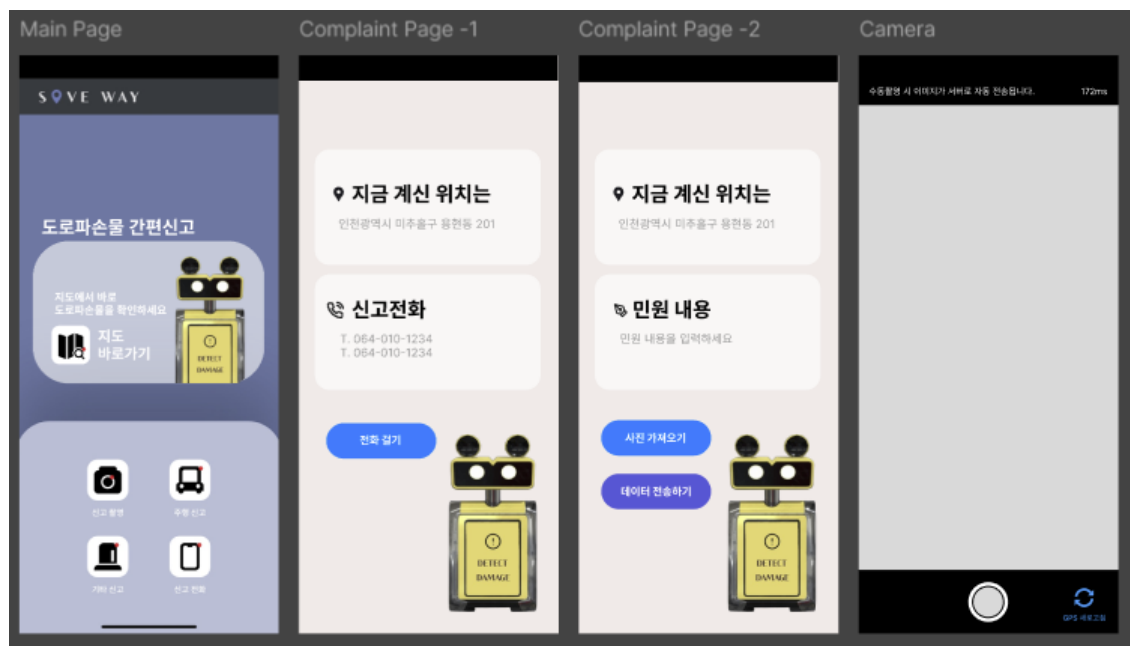
2) 사용자 편의성 증대: 사용자가 도로 관련 정보를 쉽고 빠르게 제공받을 수 있도록 하여, 도로 파손으로 인한 불편함을 최소화한다. 또한, 앱을 통해 간편하게 민원을 제기할 수 있다.

## ● App 주요 기능

기능	구현 사항
1. 신고 촬영	<ul style="list-style-type: none"> <li>- Camera X 사용하여 "신고 촬영 버튼" 클릭 시 카메라 화면으로 전환</li> <li>- Yolo v8S 클래스 부분의 도로 파손 부분을 감지하여 Bounding Box 생성</li> <li>- 캡처 버튼 클릭 시 현재 위치 정보(위도, 경도), 캡처한 사진, CropAreas 좌표, 기능 Type 을 서버로 전송</li> </ul>
2. 주행 신고	<ul style="list-style-type: none"> <li>- "주행 신고 버튼" 클릭 시 카메라 화면으로 전환</li> <li>- Record 버튼 클릭 후 주행 중 도로 파손 객체 감지 시 이미지 캡처 및 데이터(위도, 경도, CropAreas 좌표, 기능 Type) 서버 전송</li> <li>- Record 버튼 클릭 시 주행 신고 기능 종료</li> </ul>
3. 기타 신고	<ul style="list-style-type: none"> <li>- "기타 신고 버튼" 클릭 시 기타 신고 페이지로 전환</li> <li>- 사용자의 현재 위치 표시 및 민원 내용 입력 기능</li> <li>- 갤러리에서 사진 가져오기 기능 제공</li> <li>- "데이터 전송하기" 버튼 클릭 시 위치 정보(위도, 경도), 민원 내용, 이미지, 기능 Type 을 서버로 전송</li> </ul>
4. 신고 전화	<ul style="list-style-type: none"> <li>- 사용자의 현재 위치 표시</li> <li>- 위치에 따른 도로 관할 부서 전화번호 표시</li> <li>- "전화 걸기" 버튼 클릭 시 통화 키패드로 이동하여 전화 연결 가능</li> </ul>
5. 웹 페이지로 바로가기 배너	<ul style="list-style-type: none"> <li>- 관리자가 정제한 데이터가 표시되는 지도 페이지로 이동하여 도로 파손 데이터를 확인</li> </ul>

## ● 앱 디자인 기획

디자인 툴 Figma 를 활용하여 앱의 레이아웃과 요소들의 배치를 시각적으로 디자인하였다.





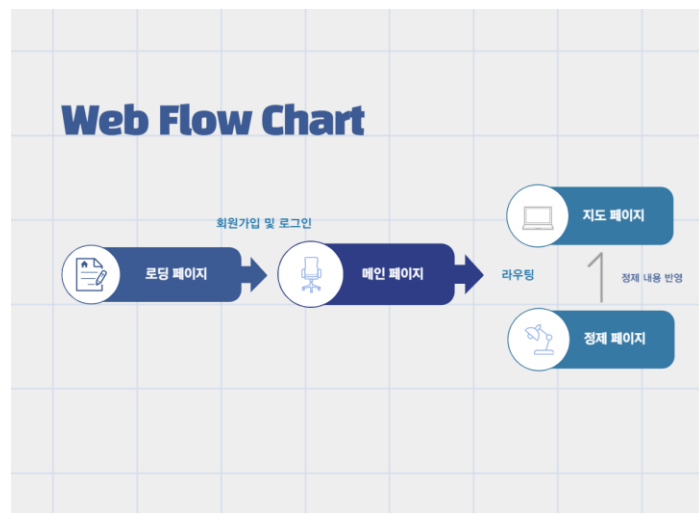
### 3-3. Web Page

#### ● 웹 서비스의 개발 목적

- 1) 데이터 정제 및 유지보수: Application 단계에서 Object Detection 및 Image Classification 과정을 거친 이미지 데이터의 정확성 및 유지보수를 위하여 웹 서비스를 구현한다. 관리자 차원에서 파손 종류와 정도를 판단하여 실시간으로 서버에 저장된 데이터를 정제할 수 있다.
- 2) 데이터 관리: 한국도로공사 등의 공공기관 관리자에게 정제된 도로 파손의 위치와 정비 현황을 한 눈에 보여준다. 지도 API 를 이용하여 파손 정보를 나타낸다.
- 3) 시각적 파손 정보 제공: 지도 API 를 활용하여 도로 파손물의 위치를 시각적으로 표현하고, 관리자가 필요한 정보를 손쉽게 확인할 수 있도록 한다.

#### ● 웹 서비스의 주요 기능

##### 1) Flow Chart



웹페이지 플로우 차트

##### 2) 페이지 별 기능 설계

페이지	기능
1. 로딩 페이지	<ul style="list-style-type: none"> <li>- 사용자가 회원가입을 완료한 후, 해당 계정으로 로그인을 수행</li> <li>- 로그인 성공 시 메인 페이지로 라우팅</li> </ul>
2. 메인 페이지	<ul style="list-style-type: none"> <li>- 로그인 후 사용자의 정보를 확인</li> <li>- 정제 페이지와 지도 페이지로 이동할 수 있는 요소 배치</li> <li>- 두 개의 페이지로 라우팅 가능</li> </ul>
3. 정제 페이지	<ul style="list-style-type: none"> <li>- 서버에서 전송된 이미지를 GET 요청으로 받아옴</li> <li>- 이미지 원본 파일을 날짜별로 그룹화</li> </ul>

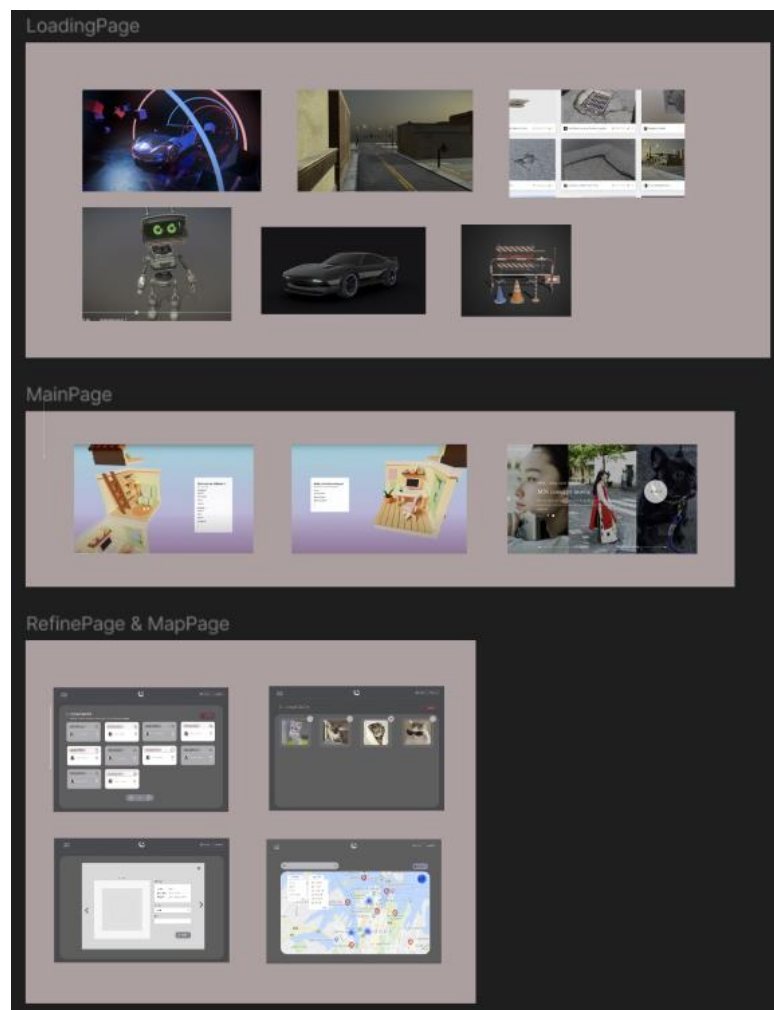
	<ul style="list-style-type: none"> <li>- 날짜 그룹을 선택하여 개별 이미지를 확인 및 팝업창에서 세부 정보 확인</li> <li>- 팝업창에서 파손의 정도와 종류를 수정하고 저장</li> </ul>
<b>4. 지도 페이지</b>	<ul style="list-style-type: none"> <li>- 카카오맵 API 를 이용하여 정제 페이지에서 관리자가 저장한 데이터를 지도에 Marker 로 표시</li> <li>- Marker 는 지역, 파손 종류, 파손 정도별로 필터링하여 확인 가능</li> </ul>

### 3) 서버와의 통신 데이터

데이터 항목	설명
이미지 메타 데이터	파일명, 이미지파일(Blob), 파손구분, 이미지 ID, 업로드날짜, GPS, 기능 유형
이미지 원본 데이터	이미지의 실제 원본 데이터

### ● 웹 페이지 디자인 기획

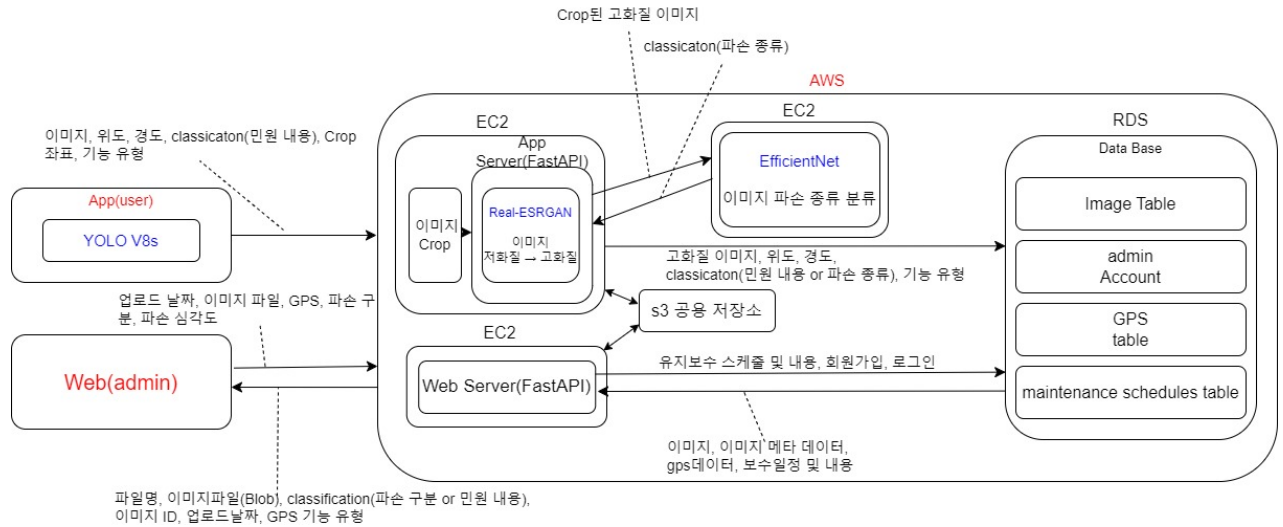
- Loading Page 와 Main Page 는 3D 로 구현하기 위해 레퍼런스를 참고하여 기획하였고, Refine Page 와 Map Page 는 디자인 툴 Figma 를 활용하여 기능 구현 중심으로 레이아웃과 요소들의 배치를 시각적으로 디자인하였다.



## IV. 개발 내용

### 1. 프로젝트 구조

#### ● 프로젝트 다이어그램



#### ● 개발 환경

No	구분	내용
1	Languages	[Android Studio앱 개발] Kotlin [웹 프론트엔드 개발] JavaScript, HTML, CSS, Three.js [서버 및 백엔드 개발] Python
2	Framework	[앱 개발] Android SDK [웹 개발] React, React-Three-Fiber
3	Library	pandas numpy matplotlib FastApi sqlalchemy OS GSAP (웹 애니메이션) react-three/fiber react-three/drei
4	DB	MySQL
5	AI	YOLOv8 (도로 파손 객체 인식) Real-ESRGAN (이미지 화질개선) EfficientNet (도로 파손물 분류)
6	etc	AWS EC2 (서버 호스팅) AWS S3 (파일 저장소) AWS RDS (관계형 데이터베이스 서비스) Docker (컨테이너 관리) Sketchfab (웹 3D 모델 오픈소스)

## 2. 데이터셋 구축

### ● 데이터 분류

: 데이터 수집 및 아래 5 개 클래스로 데이터 분류하였다.

CLASS NAME	구분
barrier	PE 방호벽
direct	PE 안내봉
porthole	포트홀
rubbercone	라바콘
tubular	시선유도봉

### ● 데이터 수집

#### 1) AI 통합 플랫폼 'AI Hub'의 공개 데이터 다운로드

- 지자체 도로부속시설물 파손 데이터 : 도로 파손물 이미지 수집
- 부산광역시 항만도로 컨테이너 차량에 의한 파손 이미지 데이터 : 도로 파손물 이미지 수집
- 이륜자동차 안전 위험 시설물 데이터 : 포트홀 이미지 수집

#### 2) 웹 크롤링을 통한 이미지 데이터 다운로드

- bing 이미지 다운로드



```
pip install bing-image-downloader

Collecting bing-image-downloader
  Using cached bing_image_downloader-1.1.2-py3-none-any.whl (5.9 kB)
Installing collected packages: bing-image-downloader
Successfully installed bing-image-downloader-1.1.2

from bing_image_downloader.downloader import download

query_string = 'rear fender damage' # 여기에 손상종류랑 차량부위를 쓰고
number_of_images = 10
output_directory = "rear_fender_damage_images" # 디렉토리도 손상종류랑 차량부위에 따라서 다르게

download(query_string, limit = number_of_images, output_dir= output_directory , adult_filter_off =True,
         force_replace = False, timeout = 60, verbose = True, filter = 'jpg')

[%] Downloading images to /content/rear_fender_damage_images/rear fender damage
```

- google 이미지 다운로드

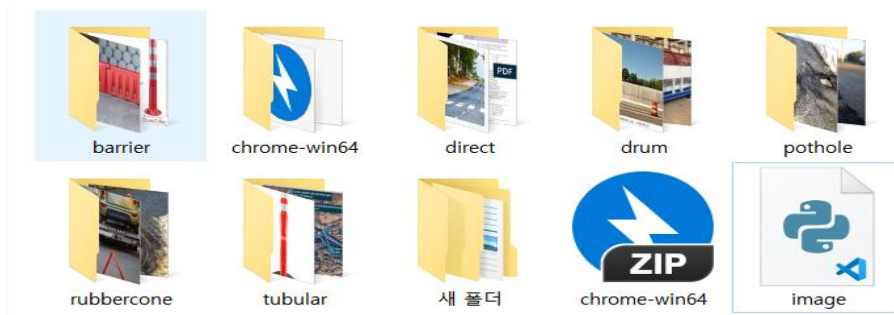
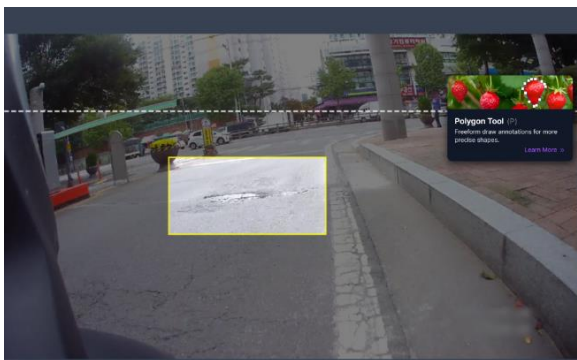


image.py 를 통한 웹 크롤링 결과 (코드 별첨)

## ● 데이터 라벨링

수집된 이미지 데이터 약 10,000 장 중 중복데이터 삭제 후 약 5,000 장을 roboflow 에 클래스 별로 업로드하였다. Bounding Box Tool 을 사용하여 라벨링을 1 차 진행하였으나, AI 모델 학습률을 더욱 증가시키기 위해 Polygon Tool 을 이용하여 2 차 라벨링을 진행하여 정확도를 높였다.

유효한 데이터만 라벨링을 진행하여 라벨링이 완료된 데이터는 총 4,500 장 정도이다.



Bounding Box Tool 을 사용



Polygon Tool 을 사용

## ● 데이터 증강 및 추출

더 정확도가 높은 AI 모델 학습을 위해 roboflow 에서 데이터 증강을 진행하였다. 데이터 증강 진행 후 Train Set(훈련 세트) 9,135 장, Valid Set(검증 세트) 877 장, Test Set(테스트 세트) 427 장이 준비되었다. 이렇게 총 학습 데이터는 11,000 개 가량 준비되었다.

Dataset Split	Preprocessing		
	Auto-Orient: Applied Static Crop: 0-100% Horizontal Region, 20-85% Vertical Region Resize: Stretch to 640x640 Auto-Adjust Contrast: Using Contrast Stretching		

TRAIN SET	VALID SET	TEST SET
9135 Images	877 Images	427 Images

### 3. AI 모델 – YOLOV8-S 모델 사용

#### ● 모델 선택 배경

##### 1) YOLOv9 모델 검토

안드로이드 애플리케이션에서 도로 파손물 및 포트홀을 실시간으로 감지하기 위해 **YOLOv8-S** 모델을 채택하였다. 프로젝트 초기에는 가장 최근에 출시된 YOLOv9 모델의 'pt'파일을 'tflite'로 변환하여 사용하려 했으나, 'pt'파일은 정상적으로 작동하지만 'tflite'파일로 변환이 되지 않는 문제가 발생하였다. 파일 변환이 되지 않으면 안드로이드에서 작동을 하지 못하기 때문에 YOLOv9 모델 사용을 포기하였다. 제약된 프로젝트 기간 내 빠른 해결책이 필요했기 때문에 기존의 레퍼런스가 많고, 정확도와 속도 면에서 YOLOv9 와 근접한 **YOLOv8** 모델을 대신 사용하기로 결정하였다.

##### 2) YOLOv8 모델 비교 및 최종 선택

앱에서는 정확성과 더불어 실시간성(속도)이 중요하기 때문에 **YOLOv8n** 과 **YOLOv8s** 두 가지 모델을 비교하였다. **YOLOv8n** 은 **YOLOv8s** 에 비해 추론 속도가 약 2 배 빠른 것으로 나타났다. 안드로이드 앱에서 도로 파손 감지가 빠르게 되어야 하기 때문에, 파라미터가 가장 적고 스피드가 빠른 YOLOv8n 을 채택하려 했었다. 그러나 객체 인식이 잘 되지 않는 성능적인 문제가 발생하였다. 그래서 YOLOv8n 보다 스피드는 느리지만 객체 인식이 더 정확한 YOLOv8s 를 채택하게 되었다.

Model	size (pixels)	mAP <sup>val</sup> 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
<a href="#">YOLOv8n</a>	640	37.3	80.4	0.99	3.2	8.7
<a href="#">YOLOv8s</a>	640	44.9	128.4	1.20	11.2	28.6
<a href="#">YOLOv8m</a>	640	50.2	234.7	1.83	25.9	78.9
<a href="#">YOLOv8l</a>	640	52.9	375.2	2.39	43.7	165.2
<a href="#">YOLOv8x</a>	640	53.9	479.1	3.53	68.2	257.8

YOLOv8 성능 비교

#### ● 모델 학습 및 평가

- **모델 학습 코드** : YOLOv8s 모델을 학습시키기 위해 작성한 코드이다.

모델을 학습시킬 때 yolov8s.pt 파일을 우리의 데이터셋으로 파인튜닝 하였다. yolov8s.pt 는 COCO 데이터셋과 같은 대규모 데이터셋에서 사전 학습된 모델이다. 이 모델은 다양한 객체 인식 작업에서 좋은 성능을 보이도록 사전 학습된 가중치를 가지고 있다. 이 가중치를 사용하면 모델이 초기 단계에서 무작위로 시작하는 것보다 훨씬 빠르고 정확하게 학습할 수 있다. 또한 yolov8s.pt 파일은 다양한 데이터에서 학습되었기 때문에 일반화 성능이 뛰어나다. 이를 기반으로 새로운 데이터셋에 맞게 파인튜닝 하면 모델이 우리의 데이터셋에도 잘 일반화될 수 있다. 이러한 이유들로 yolov8s.pt 모델을 파인튜닝 하는 방법을 사용했다.

```

from ultralytics import YOLO

# Load a model
model = YOLO('yolov8s.pt') # load a pretrained model (recommended for training)

# Train the model with 2 GPUs
results = model.train(data='data.yaml', epochs=250, imgsz=640, device=0)

```

- **모델 평가 코드** : 학습된 모델의 성능을 평가하기 위해 평가 코드를 작성하였다.

```

# Validate model

from ultralytics import YOLO

# Load a model
model = YOLO('/content/best250.pt') # load a custom model

# Validate the model
metrics = model.val() # no arguments needed, dataset and settings remembered
metrics.box.map       # map50-95
metrics.box.map50     # map50
metrics.box.map75     # map75
metrics.box.maps      # a list contains map50-95 of each category

```

- **모델 평가 지표** : YOLOv8n 및 YOLOv8s 모델 각각에 대해 150 epoch에서의 평가 지표를 비교하였으며, YOLOv8n의 추론 속도는 0.5ms, YOLOv8s의 추론 속도는 1.1ms로 YOLOv8n이 2배 가량 빠른 것으로 나타난다.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95)
all	999	965	0.761	0.788	0.81	0.632
barrier	37	38	0.845	0.858	0.941	0.676
direct	17	17	0.692	0.765	0.786	0.734
porthole	699	836	0.665	0.609	0.629	0.347
rubbercone	35	36	0.897	0.917	0.957	0.856
tubular	29	38	0.706	0.789	0.737	0.549

Speed: 0.1ms preprocess, 0.5ms inference, 0.0ms loss, 1.3ms postprocess per image  
Results saved to runs/detect/train

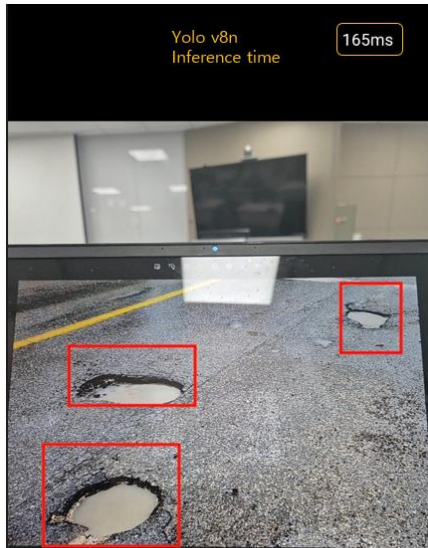
Class	Images	Instances	Box(P	R	mAP50	mAP50-95)
all	999	965	0.765	0.808	0.803	0.632
barrier	37	38	0.839	0.895	0.934	0.676
direct	17	17	0.727	0.824	0.793	0.734
porthole	699	836	0.65	0.584	0.595	0.347
rubbercone	35	36	0.901	0.917	0.968	0.856
tubular	29	38	0.706	0.822	0.725	0.549

Speed: 0.1ms preprocess, 1.1ms inference, 0.0ms loss, 0.8ms postprocess per image  
Results saved to runs/detect/val

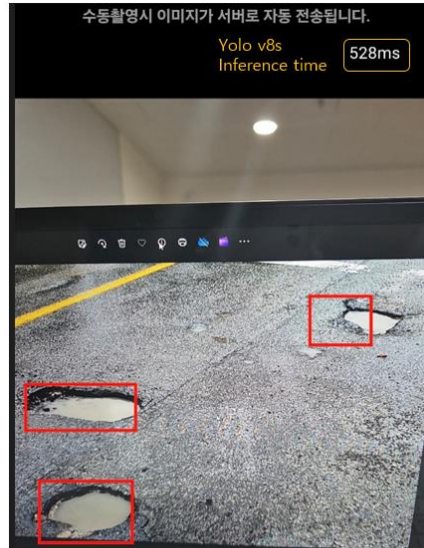
(위) YOLOv8n epoch : 150 평가 지표 (아래) YOLOv8s epoch : 150 평가 지표

- **추론 속도 in Application** : 학습된 YOLOv8n 및 YOLOv8s 모델의 'pt'파일을 'tflite'파일로 변환하여 앱에 적용시키고 앱에서 추론을 진행시켰다. YOLOv8n은 165ms의 추론 시간을, YOLOv8s는 528ms의 추론시간을 기록했다.





Yolo v8n 객체 인식 결과

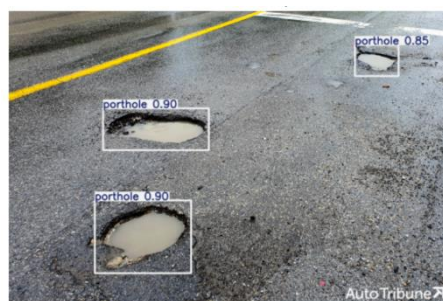


Yolo v8s 객체 인식 결과

- **모델 추론 정확도** : YOLOv8s 모델의 객체 인식 확률이 20%~30% 정도 더 높게 나타났다.



Yolo v8n 객체 인식 결과



Yolo v8s 객체 인식 결과

- **학습 결과** : YOLOv8s 모델을 250 epoch 으로 학습한 결과, 학습 수치가 낮다고 판단되는 포트홀 객체의 탐지가 원활히 감지되는 것을 확인할 수 있다. 이에 따라 YOLOv8s 모델을 앱에서 사용하기로 결정하였다.

Class	Images	Instances	Box(P	R	mAP50	mAP50-95)
all	999	965	0.765	0.808	0.803	0.63
barrier	37	38	0.839	0.895	0.934	0.693
direct	17	17	0.727	0.824	0.793	0.719
pothole	699	836	0.65	0.584	0.595	0.321
rubbercone	35	36	0.901	0.917	0.968	0.874
tubular	29	38	0.706	0.822	0.725	0.543



(좌) YOLOv8s epoch : 250 평가 지표 (우) 포트홀 Detection 결과

## ● pt to tflite 변환

앱에서 모델을 사용할 수 있도록 학습된 YOLOv8s 모델의 pt 파일을 tflite 파일 형식으로 변환하였다. 변환 과정에서 필요한 코드를 작성하고, 이를 통해 모델을 성공적으로 변환하여 앱에서 실행하였다.



```
# Export model to tflite

from ultralytics import YOLO

# Load a model
model = YOLO('/content/runs/detect/train3/weights/best.pt') # load a custom trained model

# Export the model
model.export(format='tflite')
```

## 4. AI 모델 – Real-ESRGAN 모델 사용

### ● 모델 선택 배경

AWS EC2 인스턴스에서 애플리케이션을 통해 감지된 이미지를 서버로 전송하고, 해당 이미지를 크롭하여 분류하려는 시도를 하였을 때, 이미지의 해상도(화질)가 크게 저하되어 분류 과정에서 문제를 겪게 되었다. 저해상도로 인해 도로 시설물의 파손 종류를 정확하게 분류하기 어려웠으며, 이러한 문제를 해결하기 위해 이미지의 해상도를 높이는 upscaling 과정을 도입하기로 결정하였다. 이 작업을 위해 **Real-ESRGAN** 모델을 사용하였다.

### ● 모델 비교 및 최종 선택

#### 1) 품질 향상

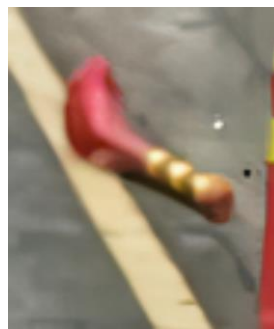
Real-ESRGAN은 현실적인 이미지 품질을 제공하며, 디테일과 질감을 잘 보존한다. 기존의 SRCNN 및 VDSR과 같은 딥러닝 기반 업스케일링 모델은 Real-ESRGAN에 비해 품질이 낮고, 인공적인 흔적이 나타나는 경우가 많았다. 따라서 Real-ESRGAN의 결과물이 파손 종류를 분류하는 데 충분한 품질을 제공한다고 판단하여 이를 사용하기로 결정했다.



[original picture]



[ Real -ESRGAN ]



[SRCNN]



[VDSR]



[original picture]



[ Real -ESRGAN ]



[SRCNN]



[VDSR]

## 2) 속도

Real-ESRGAN 은 기존의 ESRGAN 에 최신 기술을 접목시켜 연산효율성을 높이면서 높은 품질의 이미지를 생성한다. ESRGAN 과 동일한 super resolution 모델을 사용했다. 하지만 기존의 ESRGAN 과 비교했을때, pixel-unshuffle 을 적용해, 공간적인 사이즈를 줄이며 채널 수를 늘려서 사용했고, 결과적으로 GPU memory 소모량을 줄일 수 있어, 타 모델들에 비해 추론속도가 비교적 빠르다. AWS EC2 서버에서 Real-ESRGAN 모델을 직접 구동해 본 후, 타 모델들에 비하여 빠른 추론 속도를 확인할 수 있었다.

## 3) 비용

초기에 GPEN 모델을 고려했지만, GPEN 을 구동하려면 상당한 리소스를 가진 EC2 서버가 필요하며, 이에 따른 시간당 비용이 크게 증가할 것으로 예상되었다. 반면, 도로 파손 분류에 필요한 해상도는 매우 고정밀도를 요구하지 않기 때문에, 현실적인 비용을 고려하여 Real-ESRGAN 을 선택하게 되었습니다. Real-ESRGAN 은 고품질의 upscaling 모델로서, 경제적이면서도 필요한 성능을 충분히 발휘할 수 있는 모델이다.

### ● 이슈 대응 및 해결

#### 1) 시간 초과 (무한로딩)

로컬 환경에서 GPU 를 사용하여 테스트에 성공한 후, EC2 서버의 p2.xlarge 인스턴스에서 Real-ESRGAN 을 실행하였으나 첫 이미지 처리에서 무한 로딩 문제가 발생하였다. 오류 메시지를 통해 GPU 를 제대로 활용하지 못하고 있다는 것을 확인하였고, 문제의 원인을 탐색하였다. Tesla K80 GPU 와 PyTorch 의 버전이 호환되지 않는 문제를 확인한 후, PyTorch 버전을 **CUDA 11.4** 와 호환되는 버전으로 다운그레이드하여 문제를 해결하였다.

```
ch.load' with weights_only=False (the current default value), which uses the default pickle Mo
implicitly. It is possible to construct malicious pickle data which will execute arbitrary code
ing unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
details). In a future release, the default value for 'weights_only' will be flipped to 'True'. Th
limits the functions that could be executed during unpickling. Arbitrary objects will no longer be
lowed to be loaded via this mode unless they are explicitly allowlisted by the user via 'torch.se
ization.add_safe_globals'. We recommend you start setting 'weights_only=True' for any use case whe
you don't have full control of the loaded file. Please open an issue on GitHub for any issues rela
to this experimental feature.
loadnet = torch.load(model_path, map_location=torch.device('cpu'))
isting 0 test1
```

[사진 1 로딩 에러]

#### 2) Python 패키지 import 오류

basrcr 패키지를 설치 후, ModuleNotFoundError: No module named 'torchvision.transforms.functional\_tensor' 오류가 발생하였습니다. 초기에 torchvision 패키지 문제로 인식하고 여러 번 업그레이드, 재설치, 다운그레이드를 시도했으나 문제는 해결되지 않았다. 직접 라이브러리 경로에서 문제를 탐색한 결과, degradations.py 파일 내에서 torchvision.transforms.functional\_tensor 를 import 하는 과정에서 호환되지 않는 버전을 사용하고 있다는 것을 확인하였고, 코드를 수정하여 문제를 해결할 수 있었다.

#### 3) Python 스크립트 소멸 문제

pth 파일을 이용해 Real-ESRGAN 을 잘 사용하고 있었으나, 예상치 못한 ModuleNotFoundError: No module named 'version.py' 오류가 발생하였다. 로컬 환경에서 진행할 때는 문제가 없던 파일이 서버로 전송되는 과정에서 손실된 것으로 파악되었고, 스크립트 파일이 사라져 있었다. 본 팀은 서버 디렉토리에서 touch 명령어를 통해 version.py 파일을 생성하고, 해당 문제를 해결하였다.

## 5. AI 모델 – EfficientNet 모델 사용

### ● 모델 로드 및 클래스 수정

도로 파손물의 분류 작업을 위해 **EfficientNet-b3** 모델을 활용하였다. 이 모델은 YOLO 로 감지된 사진을 정확하게 분류하여 데이터베이스에 저장한다. 또한 더 많은 클래스를 학습시켜 YOLO 에서 탐지하지 못한 파손 부분도 분류하는 역할을 한다. EfficientNet 은 적은 데이터 셋으로도 높은 효율성을 보이며, ViT, BASIC-L 등과 같은 모델들에 비해 가볍다는 장점이 있다.

- **사전 학습된 모델 로드:** EfficientNet-b3 모델을 로드한다. 이 버전은 적은 데이터셋으로도 우수한 성능을 발휘하기 때문에 최종 선택한 모델이다..
- **클래스 수 정의:** 처음에는 6 개의 클래스로 분류할 계획이었으나, 최종적으로 15 개의 클래스로 수정하였다.
- **모델 수정:** 사전 학습된 모델의 마지막 Fully Connected 층을 15 개의 클래스로 수정하여 도로 파손물의 정확한 분류를 가능하게 하였다.
- **평가 모드 전환:** model.eval()을 사용하여 모델을 평가 모드로 전환하였다. 이는 추론 시 가중치 업데이트를 방지하고 모델을 최적의 상태로 유지하게 한다.

```
class_names = {  
    "0": "barrier",      #!  
    "1": "no parking",  #!  
    "2": "roadkill",    # !  
    "3": "도로 요철 균열", #!  
    "4": "표지판 파손",  #!  
    "5": "배수 시설 불량", # !  
    "6": "가드레일",    # !  
    "7": "pe drum",     #!  
    "8": "porthole",    #!  
    "9": "rubbercone",  #!  
    "10": "tubular",    #!  
    "11": "낙석",       #!  
    "12": "노면 균열",  #!  
    "13": "중앙 분리대", # !  
    "14": "보도블럭 파손" #!  
}
```

```
# 사전 학습된 EfficientNet 모델 로드
model = EfficientNet.from_pretrained('efficientnet-b3')
num_classes = 15 # 분류할 클래스 수에 맞게 설정
in_features = model._fc.in_features
model._fc = torch.nn.Linear(in_features, num_classes)
#학습된 pt파일 load
model.load_state_dict(torch.load('./president_model.pt', map_location=torch.device('cpu')))
model.eval()
```

## ● 이미지 전처리 및 추론

appServer 에서 API 호출을 통해 S3 URL 로 전달받은 이미지를 불러온 후, 분류 작업을 위해 이미지를 전처리한다. 이 과정에서 이미지 크기를 224\*224 로 리사이즈 한 후, tensor 로 변환하고 정규화한다.

```
@app.post("/predict")
async def predict(file_url: str = Form(...)):
    global model
    print("effi")
    # S3 클라이언트 생성
    s3_client = boto3.client('s3')

    # 이미지 로드 및 전처리
    # S3에서 파일 다운로드
    BUCKET_NAME = 'kebbucket'
    image_filename = file_url.split('/')[-1]
    local_path = f"/tmp/{image_filename}"

    s3_client.download_file(BUCKET_NAME, image_filename, local_path)

    # 다운로드한 파일 열기
    input_image = Image.open(local_path).convert("RGB")
    input_image = input_image.resize((224, 224))

    preprocess = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    ])

    input_tensor = preprocess(input_image)
    input_batch = input_tensor.unsqueeze(0)

    if torch.cuda.is_available():
        input_batch = input_batch.to('cuda')
        model = model.to('cuda')
    else:
        input_batch = input_batch.to('cpu')
        model = model.to('cpu')

    # 분류 예측 수행
    with torch.no_grad():
        output = model(input_batch)

    _, predicted = torch.max(output, 1)
    classification_result = str(predicted.item())

    return {"classification_result": classification_result}
```

- **이미지 전처리:** 이미지를 tensor 로 변환하고 정규화하는 파이프라인을 구성하여, 이를 input\_tensor 에 저장한다. 이 과정은 모델이 이미지를 올바르게 이해하고 처리할 수 있도록 한다.
- **배치 차원 추가:** input\_tensor.unsqueeze(0)를 통해 배치 차원을 추가하여 모델이 여러 이미지를 동시에 처리할 수 있도록 한다.

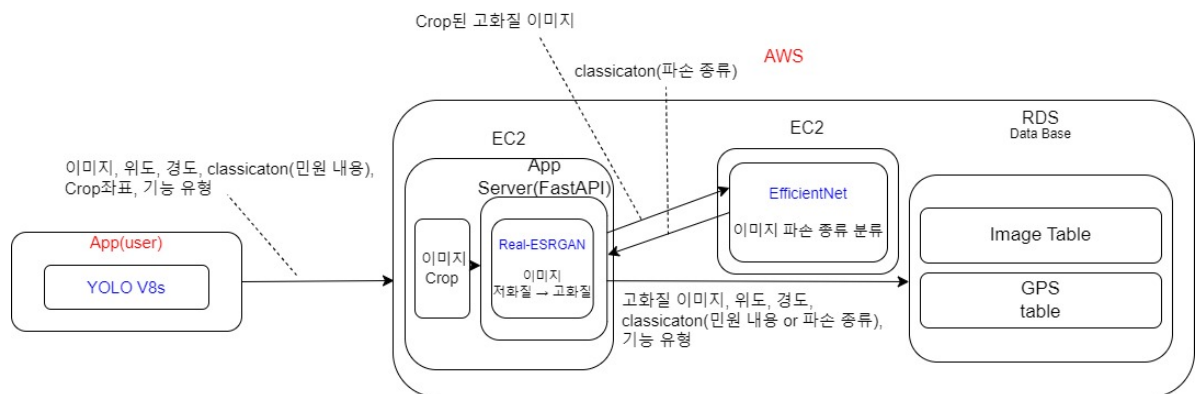
- **GPU 사용 여부 확인:** GPU 사용 가능 여부를 확인하는 조건문을 추가하여, 가능할 경우 GPU 에서 추론을 진행함으로써 성능을 최적화한다.
- **추론:** with torch.no\_grad()를 사용하여 추론 모드로 설정하고, 가중치 업데이트를 방지하며 추론을 수행한다. 추론 결과는 최종적으로 데이터베이스에 저장된다.

## 6. Application Service

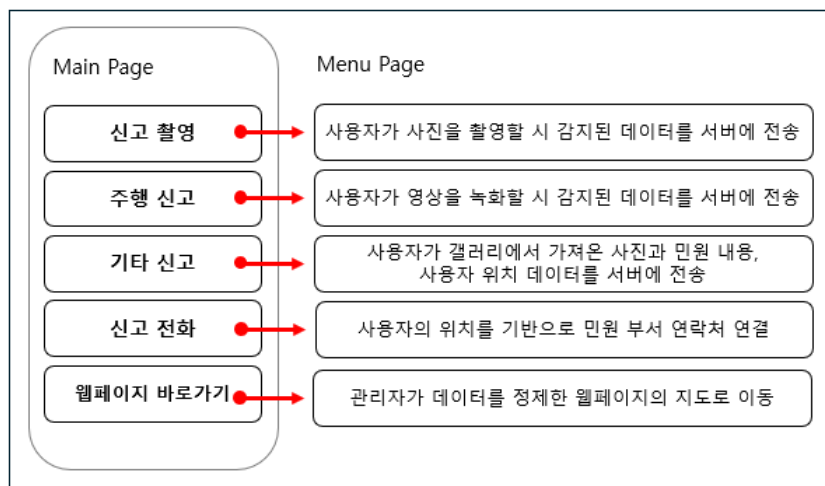
### ● 개발 환경

- Android Studio : koala 버전
- Language : Kotlin
- Design tool : Figma
- App Communication : FastApi (retrofit 통신)

### ● Application 구조



### ● Application 기능



● Application 개발 내용

기능	구성	구현방법
신고촬영	카메라	카메라 X 가 실행되면서 도로 파손부분이 인식이 바로 시작되며, Bounding Box 들을 그려준다.
	데이터 전송	<p>촬영 버튼을 클릭 시 데이터가 바로 서버로 전송된다. 전송되는 데이터의 종류는 아래와 같다.</p> <p><b>1) Bounding Box 인식 시</b> 이미지 안에서 인식된 바운딩 박스의 크롭 좌표가 전송된다.</p> <p><b>2) Bounding Box 미인식 시</b> 이미지 비트맵의 전체 좌표(0, 0, bitmap.width, bitmap.height)로 서버로 전송된다.</p> <p>크롭 좌표는 서버에서 크롭되어 Real-ESRGAN 을 통해 고화질로 변환되고, EffientNet 이 이미지의 파손 종류를 분류한다. EffientNet 의 성능이 높아 Bounding Box 를 인식하지 않아도 파손 종류를 구분할 수 있기 때문에 이러한 방식으로 설계되었다.</p>
	AI 모델 작동 확인	객체 인식 inference time(객체 인식을 해서 Bounding Box 를 그려주는 데까지 걸린 시간)이 우측 상단에 표시된다. 이 기능으로 AI 모델이 잘 작동되고 있는지 확인할 수 있다.
	GPU 새로고침	TensorFlow Lite 모델을 재시작하여 GPU 를 사용할 지 여부에 따라 TensorFlow Lite 인터프리터를 다시 설정하고 초기화한다. (YOLO 모델 재시작 버튼)
주행 신고	카메라	주행신고 버튼 클릭 시 카메라 X 가 실행된다.
	데이터 전송	촬영 버튼 클릭 시 상시 촬영으로 변경되고, 5 초동안 바운딩 박스가 그려질 시 자동으로 **데이터가 서버에 전송된다.
	AI 모델 작동 확인	객체 인식 inference time(객체 인식을 해서 Bounding Box 를 그려주는 데까지 걸린 시간)이 우측 상단에 표시된다. 이 기능으로 AI 모델이 잘 작동되고 있는지 확인할 수 있다.
	GPU 새로고침	TensorFlow Lite 모델을 재시작하여 GPU 를 사용할 지 여부에 따라 TensorFlow Lite 인터프리터를 다시 설정하고 초기화한다. (YOLO 모델 재시작 버튼)
기타 신고	위치 표시	사용자의 GPS 값을 가져와 현재 위치를 표시한다.
	민원 내용 입력	사용자가 민원 내용을 직접 입력할 수 있다.
	사진 가져오기	[사진 가져오기] 버튼 클릭 시 갤러리에서 사진을 가져올 수 있다.
	데이터 전송	[데이터 전송하기] 버튼 클릭 시 데이터를 서버가 서버에 전송된다.
신고 전화	위치 표시	사용자의 GPS 값을 가져와 현재 위치를 표시한다.
	담당자 정보 표시	사용자의 GPS 값으로 현재 위치를 관리하는 담당 부서의 연락처를 연결한다.
	전화 연결	[전화 걸기] 버튼 클릭 시 전화 어플로 바로 전화를 걸 수 있도록 전화번호가 연동된다.
웹페이지 바로가기	웹 링크	[지도 바로 가기] 이미지 버튼을 클릭하면, 관리자가 웹에서 정리한 현재 도로 파손물 현황을 지도 형식으로 사용자가 볼 수 있다.

## ● Application 개발 이슈사항

### 1) “신고 촬영” 기능 개발 시 이슈사항

#### - 이슈 1

Android Studio 에서 제공하는 기본 카메라 기능이 한정되어 있어, 앱 내에서 도로 파손을 객체로 감지하는 기능 구현에 어려움이 있었다. 특히, 기본 카메라 API 는 객체 감지와 같은 고급 기능을 지원하지 않기 때문에, 이미지 처리와 분석에 필요한 고성능 카메라 기능이 필요하였다.

⇒ **해결방안** : Android Studio 에서 제공하는 카메라 X API 를 도입하였다. CameraX 는 Android 의 최신 카메라 기능을 활용할 수 있도록 설계된 라이브러리로, 하드웨어와의 호환성이 높고, 고성능을 발휘할 수 있는 기능들을 제공한다. CameraX 는 ML Kit 과 같은 머신러닝 라이브러리와도 쉽게 통합되어, 실시간 객체 감지와 같은 고급 기능 구현이 가능하다. 따라서 CamaraX 를 도입함으로써, 앱 내에서 실시간으로 도로 파손을 감지하고, 해당 영역을 Bounding Box 로 표시할 수 있게 되었다.

```
val cameraxVersion = "1.4.0-alpha04"
implementation("androidx.camera:camera-camera2:${cameraxVersion}")
implementation("androidx.camera:camera-lifecycle:${cameraxVersion}")
implementation("androidx.camera:camera-view:${cameraxVersion}")
```

#### - 이슈 2

카메라 작동 중에 Yolo v8S 모델을 활용하여 도로 파손을 실시간으로 감지하고, 해당 객체를 화면에 바운딩 박스로 표시하는 데 어려움이 있었다. 이는 카메라와 객체 감지 모델 간의 상호작용을 실시간으로 구현해야 하는 복잡한 작업이었으며, 이를 통해 사용자에게 도로 파손 정보를 시각적으로 제공해야 했다.

⇒ **해결방안**: 문제를 해결하기 위해, 카메라에서 객체를 감지하고 시각적으로 표현할 수 있도록 세 가지 주요 클래스를 구현하였다:

##### 1. BoundingBox.kt

감지된 객체에 대한 정보를 담는 데이터 클래스를 정의하였다. 이 클래스는 바운딩 박스의 좌표(x1, y1, x2, y2), 중심 좌표(cx, cy), 너비(w), 높이(h), 감지 신뢰도(cnf), 클래스 ID(cls), 클래스 이름(clsName) 등의 정보를 포함하여 객체 감지 결과를 구조화된 형태로 관리한다.

##### 2. Detector.kt

이 클래스는 YOLO v8S 모델과 상호작용하여 객체 감지를 수행하는 역할을 한다. TensorFlow Lite 의 Interpreter 를 사용해 YOLO 모델을 로드하고, 이미지 전처리(ImageProcessor)를 통해 입력 이미지를 모델에 적합하게 변환한 후, 모델을 실행하여 바운딩 박스를 생성한다. 또한, GPU delegate 를 활용하여 감지 속도를 향상시키고, 감지된 객체의 클래스 라벨을 관리한다.

##### 3. OverlayView.kt



감지된 객체를 화면에 시각적으로 표시하는 역할을 하는 클래스이다. 이 클래스는 Android 의 View 를 확장하여 onDraw 메서드를 오버라이드하고, 바운딩 박스와 클래스 라벨을 화면에 그린다. 이를 통해 사용자는 카메라 작동 중에 실시간으로 감지된 객체를 확인할 수 있다.

⇒ **전체적인 프로세스:** 이미지가 카메라로 캡처된 후, Detector 클래스가 YOLO 모델을 통해 객체를 감지하고 바운딩 박스를 생성한다. 그 후, 이 바운딩 박스는 OverlayView 를 통해 화면에 시각적으로 표시된다. 이로써 사용자는 실시간으로 도로 파손 객체를 확인할 수 있게 되었다.

### - 이슈 3

사용자가 캡처 버튼을 눌렀을 때, 캡처된 이미지와 관련 데이터를 즉시 서버로 전송하는 기능을 구현하는 데 어려움이 있었다. 특히, 캡처된 이미지와 함께 위치 정보, 바운딩 박스 좌표 등의 다른 데이터를 하나로 묶어 서버로 전송하는 방법에 대해 명확한 해결책이 필요했다.

⇒ **해결방안:** 이 문제를 해결하기 위해 Retrofit2 를 활용하여 데이터를 서버로 전송하는 방식을 구현하였다. Retrofit2 는 Android 에서 HTTP 요청을 처리하는 데 널리 사용되는 라이브러리로, 파일 전송 및 다양한 데이터 형식을 지원한다

#### .1. RequestBody 형식 사용

캡처된 이미지와 다른 데이터를 서버로 전송하기 위해, 모든 데이터를 Retrofit2 의 RequestBody 형식으로 변환하였다. MultipartBody.Part 를 사용하여 이미지를 파일로 처리하고, 나머지 데이터는 RequestBody 로 각각 포장하여 서버로 전송할 수 있도록 구성하였다.

#### 2. Retrofit 인터페이스 정의

서버와의 통신을 위해 Retrofit 인터페이스를 정의하고, @Multipart 어노테이션을 사용하여 여러 개의 RequestBody 를 하나의 HTTP 요청에 포함시킬 수 있도록 하였다. 이를 통해, 이미지와 기타 데이터를 함께 묶어 서버로 전송하는 기능을 구현하였다.

#### 3. 데이터 전송

사용자가 캡처 버튼을 누르면, 이미지와 다른 데이터를 RequestBody 형식으로 변환한 후, Retrofit2 의 enqueue() 메서드를 사용하여 비동기 방식으로 데이터를 서버에 전송하였다. 이로써, 사용자가 캡처한 이미지와 해당 이미지에 관련된 모든 데이터를 즉시 서버로 전송할 수 있게 되었다.

### 2) “주행 신고” 기능 개발 시 이슈사항

#### - 이슈 1

주행 중 도로 파손을 감지할 때, 차량의 위치가 계속해서 빠르게 변경되어 위치 정보를 정확하게 업데이트하고 서버로 전송하는 데 어려움이 있었다. 이로 인해 위치 정보가 과도하게 자주 변경되어 서버에 불필요한 데이터를 전송하게 될 가능성이 있었다.



⇒ **해결방안** : 위치 업데이트 주기를 설정하고, 필요에 따라 위치 업데이트를 시작하거나 중지할 수 있는 방법을 구현하였다.

#### 1. 위치 업데이트 주기 설정

FindLocation.kt 파일에서 Handler 와 Looper 를 사용하여 위치 업데이트 간격을 조절하였다. refreshInterval 변수를 통해 위치 업데이트 간격을 5000 밀리초(5 초)로 설정하여, 위치 정보가 너무 자주 변경되지 않도록 하였다. 이로써 위치 업데이트 주기를 제어할 수 있게 되었다.

#### 2. 위치 업데이트 메서드

startLocationUpdates() 메서드는 Handler 를 사용하여 설정된 간격(refreshInterval)마다 위치를 업데이트하도록 구현되었다. 위치 정보는 fusedLocationClient 를 통해 가져오며, 이는 Google Play 서비스의 위치 API 를 사용하여 정확한 위치 데이터를 제공한다. handler.postDelayed 를 사용하여 위치 업데이트를 주기적으로 반복 수행할 수 있도록 하였다.

#### 3. 위치 정보 갱신 및 서버 전송

위치가 성공적으로 업데이트되면, 해당 위치 정보를 사용하여 필요한 작업, 예를 들어 서버로의 위치 데이터 전송을 수행할 수 있다. 이 과정에서 불필요한 위치 업데이트를 줄이기 위해 주기적으로만 위치를 새로 고치도록 설정하였다.

#### 4. 위치 업데이트 제어

AfterCarFunction.kt 파일에서는 토글 버튼을 통해 위치 업데이트를 제어할 수 있게 하였다. 토글 버튼이 활성화되면 startLocationUpdates() 메서드가 호출되어 위치 업데이트가 시작되고, 비활성화되면 stopLocationUpdates() 메서드가 호출되어 위치 업데이트가 중지된다. 이로써 사용자는 필요에 따라 위치 업데이트를 제어할 수 있게 되었으며, 주기적인 위치 업데이트는 설정된 5 초 간격으로 이루어지게 된다.

### 3) “기타 신고” 기능 개발 시 이슈사항

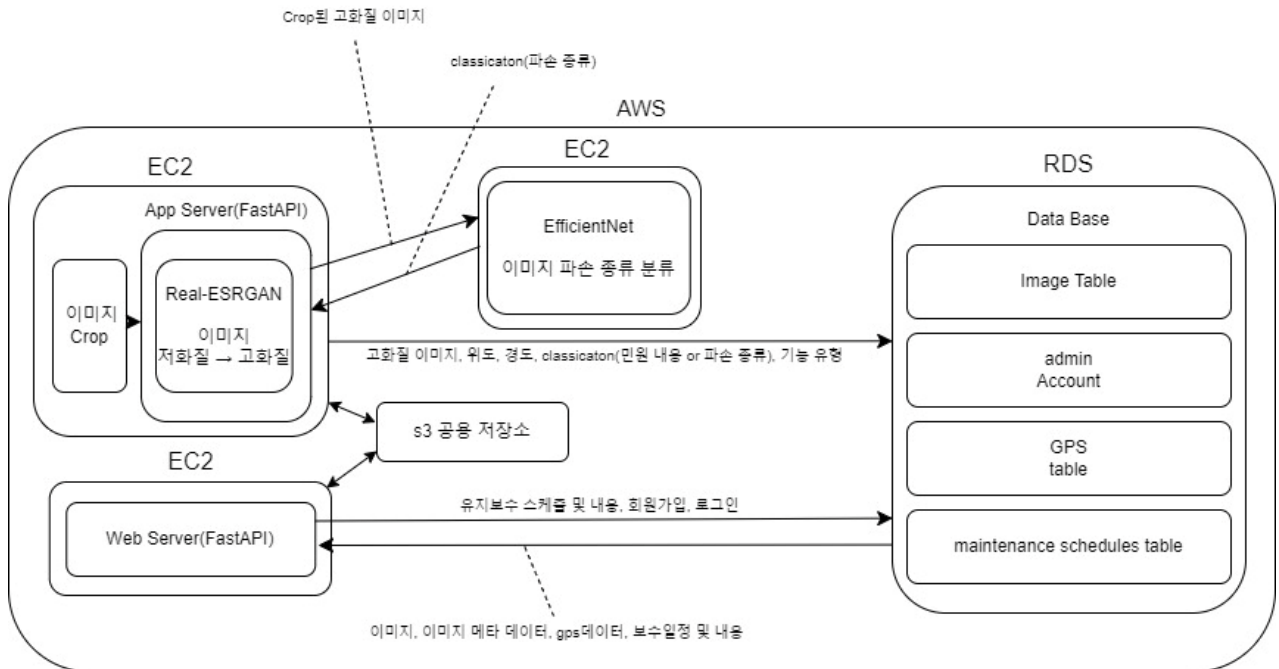
#### - 이슈 1

위치 정보를 거의 모든 기능에서 사용해야 했는데, 별도의 클래스로 정리하지 않고 중복된 코드를 사용함으로 인해 앱의 성능이 저하되고 속도가 느려지는 문제가 발생하였다. 중복된 위치 처리 로직이 코드 유지보수와 성능 최적화에 있어서도 비효율적이었다.

⇒ **해결방안** : 위치 정보를 관리하는 별도의 Kotlin 클래스인 FindLocation.kt 를 작성하여 위치 관련 기능을 모두 이 클래스에서 처리하도록 구현하였다.

## 7. Back-End

### 7-1. 아키텍처 다이어그램



#### ● 개발 환경

- 사용할 프레임워크 : FastAPI
- API 서버 구조 : appServer, webServer
  - appServer : Post method (이미지, gps 값 등 데이터 처리)
  - webServer : Post method (유지보수 스케줄 처리), Get method (이미지, gps 값 등 데이터 조회), login 기능, Delete method (데이터 삭제)
- database : Mysql
  - 테이블 : image table

#### ● 주요 기술 스택

**1) FastAPI :** Python 기반의 백엔드 프레임워크인 FastAPI 를 사용하여 서버를 구축하였다. FastAPI 는 뛰어난 성능, 자동 문서화, 직관적이고 간결한 코드로 인해 개발 생산성을 크게 향상시킬 수 있다. 또한, Uvicorn, Gunicorn 과 함께 사용되어 배포와 확장성이 용이하며, API 서버 구축에 최적화된 환경을 제공한다.

**2) Docker :** Docker 를 활용하여 각 서버를 컨테이너화하였고, 이를 통해 일관된 개발 환경을 보장하고 종속성 관리를 용이하게 하였다. Docker 는 빠른 배포와 확장성을 제공하며, CI/CD

파이프라인 구축에도 유리하다. 각 서버는 Docker 이미지를 통해 EC2 인스턴스에 쉽게 배포할 수 있게 된다.

**3) AWS :** AWS 의 EC2 인스턴스를 활용하여 서버를 배포하였다. 또한, RDS 를 사용하여 데이터베이스 연결을 손쉽게 구현하였고, S3 스토리지를 통해 서로 다른 서버들이 동일한 저장소에 접근할 수 있도록 하였다. VPC, 서브넷, 보안 그룹 설정을 통해 데이터 보호를 강화하였으며, 리소스 과부하 문제를 해결하기 위해 총 3 개의 EC2 인스턴스를 사용하여 서버를 분산 배치하였다.

**4) Database :** 데이터베이스는 사용자 친화적인 인터페이스와 읽기 작업에서 효율적인 성능을 제공하는 MySQL 을 사용하여 설계되었다. 다음과 같은 세 가지 주요 테이블을 포함하고 있다:

- Image Table: 이미지 데이터가 저장되는 테이블로, 도로 파손 이미지와 관련된 정보 저장.
  - GPS Table: 도로 파손 이미지와 관련된 GPS 정보 저장.
  - Maintenance Schedules Table: 도로 유지보수와 관련된 일정 및 작업 내용을 관리하는 테이블
- 또한, `imageId` 를 외래키로 사용하여 `gps table` 및 `maintenance_schedules table` 과 `image table` 을 연결하여, 이미지와 관련된 모든 정보를 통합 관리할 수 있도록 하였다.

#### ● 전체적인 사용 흐름

- 데이터 수집 : 도로 파손 정보가 어플리케이션을 통해 수집되고, AppServer 로 전송된다.
- 데이터 처리 : AppServer 는 수집된 데이터를 S3 와 데이터베이스에 저장하며, AI 모델을 통해 데이터를 처리한다.
- 데이터 관리 : WebServer 를 통해 관리자는 수집된 데이터를 조회하고, 필요한 유지보수 작업을 관리한다.

## 7-2. AppServer

#### ● Post method

appServer 의 Post method 는 다음과 같은 파라미터를 받는다.

: 파일(이미지), 위도 및 경도(GPS 좌표), 크롭을 위한 bounding box 좌표, 기타 신고를 위한 분류 결과, 신고 종류

```
# Background task for upscaling and classification
if type != 2:
    background_tasks.add_task(upscaling_and_classification, s3_url, db_image.id, db)

@app.post("/upload")
async def upload_image(
    background_tasks: BackgroundTasks,
    file: UploadFile = File(...),
    latitude: float = Form(...),
    longitude: float = Form(...),
    crop_areas: str = Form(...),
    classification_result: str = Form(...),
    type: int = Form(...),
    db: Session = Depends(database.get_db),
):
```

이 메서드는 전달된 'crop\_areas'를 기반으로 이미지를 크롭한 후 잘려진 이미지를 순차적으로 S3 및 데이터베이스에 저장한다. 이후, 백그라운드에서 Real-ESRGAN 모델을 사용하여 이미지를 업스케일링한다. 신고종류가 "기타신고(2)"인 경우, 사용자가 직접 신고 내용을 입력하므로 모델을 실행하지 않으며, 나머지 신고 유형에 대해서만 모델이 실행된다.

```
root@ip-172-31-0-191:~# docker logs e0129b3764b9
INFO: Started server process [1]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
crop ok
s3 save ok
db save ok
INFO: 165.246.159.2:50793 - "POST /upload HTTP/1.1" 200 OK
filename: c284e06c-a77f-41b3-adc8-37621c164191_0.jpg
path: /app/Real-ESRGAN/inputs/c284e06c-a77f-41b3-adc8-37621c164191_0.jpg
upscale_filename : c284e06c-a77f-41b3-adc8-37621c164191_0_out.jpg
upscale_file_path : /app/Real-ESRGAN/results/c284e06c-a77f-41b3-adc8-37621c164191_0_out.jpg
Running command: python3 inference_realesrgan.py -n RealESRGAN_x4plus
Execution successful!
Testing 0 c284e06c-a77f-41b3-adc8-37621c164191_0

Upscaled image uploaded to S3: upscaled_c284e06c-a77f-41b3-adc8-37621c164191_0.jpg
Image filepath updated in DB: https://kebbucket.s3.amazonaws.com/upscaled_c284e06c-a77f-41b3-adc8-37621c164191_0.jpg
호출 완료
Response text: {"classification_result": "4"}
finished to update db!
```

#### 이미지 처리 및 업스케일링 서버 로그

### ● appServer 개발 이슈사항

#### - 이슈 1 : GPU 사용 오류

GPU를 사용할 수 있는 EC2 인스턴스를 생성했음에도 불구하고, GPU가 인식되지 않는 오류가 발생하였다.

⇒ **해결방안** : sudo apt-get update, sudo apt-get install -y nvidia-docker2, sudo systemctl restart docker, sudo apt install nvidia-driver-450 명령어를 사용하여 직접 NVIDIA 드라이버를 설치함으로써 문제를 해결하였습니다.

## - 이슈 2 : Docker 에서 GPU 사용 문제

정상적인 로그가 출력되지 않으며, GPU 를 사용하지 못하는 에러가 지속적으로 발생했다. 이는 Docker 컨테이너에서 EC2 인스턴스의 GPU 를 인식하지 못하는 문제였음을 확인했다.

⇒ **해결방안** : Docker 에서 GPU 를 사용하기 위해 `--gpus all` 옵션을 추가하여 문제를 해결하였다. 이후, Real-ESRGAN 이 정상적으로 동작하였고, 업스케일링 작업을 완료한 후 EfficientNet 서버의 API 를 호출하였다. 최종적으로 분류가 완료된 후에는 image table 의 classification\_result 속성을 업데이트하였다.

## - 이슈 3: 서버 간 API 통신 문제

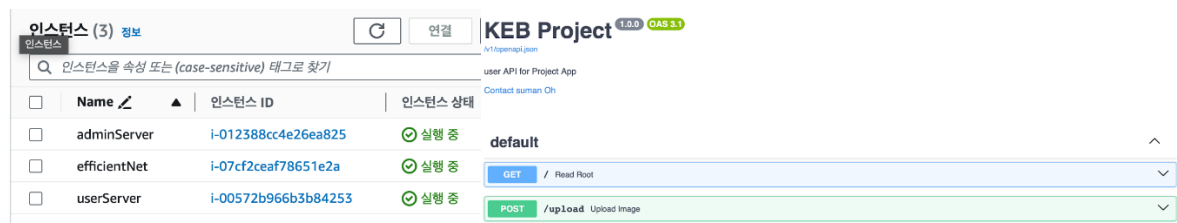
서로 다른 EC2 인스턴스에 있는 서버 간 API 통신에서 requests 라이브러리를 사용했으나, 통신이 제대로 이루어지지 않았고, 에러 메시지도 출력되지 않았다.

⇒ **해결방안** : FastAPI 는 비동기 기반이기 때문에 동기 방식의 requests 라이브러리를 사용할 수 없었습니다. 이 문제를 해결하기 위해, 비동기 요청을 지원하는 httpx 라이브러리로 교체하여 통신을 성공적으로 구현하였다.

## - 이슈 4: 모델 분리 문제

원래 하나의 인스턴스에서 두 개의 모델(Real-ESRGAN 과 EfficientNet)을 실행할 계획이었으나, Cuda Runtime Error 가 지속적으로 발생하였다. 이 오류는 EC2 인스턴스의 리소스 한계로 인해 발생한 것으로 판단된다.

⇒ **해결방안** : 최종적으로 두 모델을 각각 다른 EC2 인스턴스로 분리하여 실행함으로써 문제를 해결했다. 모델을 분리함으로써 각 인스턴스의 리소스 부담이 줄어들었고, 시스템의 안정성과 성능이 크게 향상되었다.



Name	인스턴스 ID	인스턴스 상태
adminServer	i-012388cc4e26ea825	실행 중
efficientNet	i-07cf2ceaf78651e2a	실행 중
userServer	i-00572b966b3b84253	실행 중

**KEB Project** 1.0.0 QAS 3.1  
user API for Project App  
Contact suman Ch  
**default**  
GET / Read Root  
POST /upload Upload Image

## 7-3. WebServer

### ● 주요 메서드 및 기능

WebServer에서는 이미지 메타데이터, 이미지 파일, 유지보수 스케줄 데이터를 관리하고, 사용자 인증 기능을 제공한다.

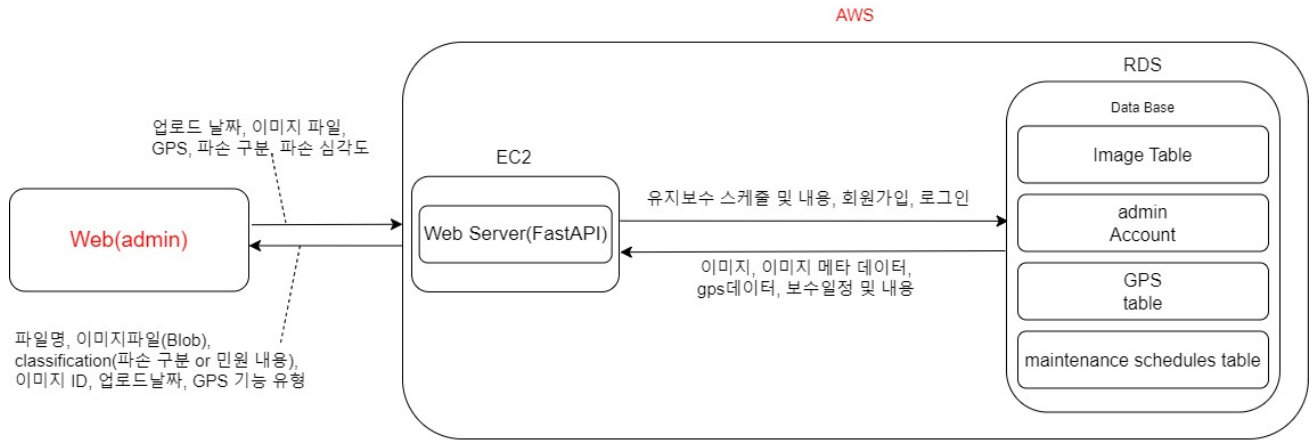
- 1) get("/images/total/") : image table에 저장된 모든 이미지의 메타데이터를 조회하여 반환
- 2) get("/images/{image\_id}") : image\_id에 해당하는 이미지의 메타데이터를 조회하여 반환
- 3) get("/images/get/{image\_id}") : image\_id에 해당하는 이미지 파일 자체를 반환
- 4) get("/imagesCount/totalCount") : image table에 저장된 이미지의 총 개수를 조회하여 반환
- 5) get("/getDetails/") : maintenance-schedule 테이블에 저장된 모든 데이터를 조회하여 반환.  
details는 도로 파손 정도를 의미하며, 관리자가 파손 정보를 저장하고 불러오는데 사용한다.
- 6) delete("/images/{image\_id}") : 특정 이미지 데이터를 삭제
- 7) post("/maintenance-schedule") : 유지보수 스케줄을 추가하고, 파손 정도(Details) 및 EfficientNet이 잘못 분류한 결과를 수정
- 8) signup, login 메서드 : 회원가입 및 로그인 기능을 제공

## 8. Web-Page Service

### ● 개발 환경

- Language : JavaScript, Three.js
- Front-End: React, React-Three-Fiber
- MAP API : Kakao Map API
- Design tool : Figma
- Server Communication: FastAPI, Postman
- Version Control : Github

## ● Wep 구조

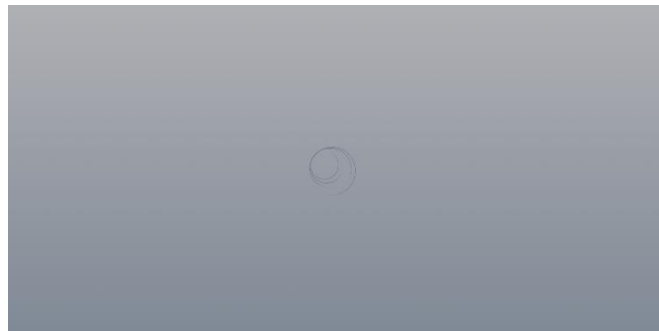


## 8-1. 3D Web-Page

로딩 페이지와 메인 페이지는 사용자 친화성을 위해 3D 컴포넌트를 사용하여 입체감 있게 구현하였다. 웹 사이트의 주제를 표현하고 사용자와 인터랙티브하게 상호작용 할 수 있도록 구성하였다. 3D 컴포넌트 소스는 오픈소스 사이트인 Sketchfab 를 사용하였다.

### ● 로딩 페이지

#### 1) Splash



- **React-Spinners:** 로딩 페이지의 3D 컴포넌트의 렌더링이 진행될 동안, 사용자에게 로딩이 진행되고 있다는 사실을 전달하기 위하여 React 의 spinners 라이브러리를 이용하여 애니메이션 효과를 추가한 Splash 화면을 제작하였다.

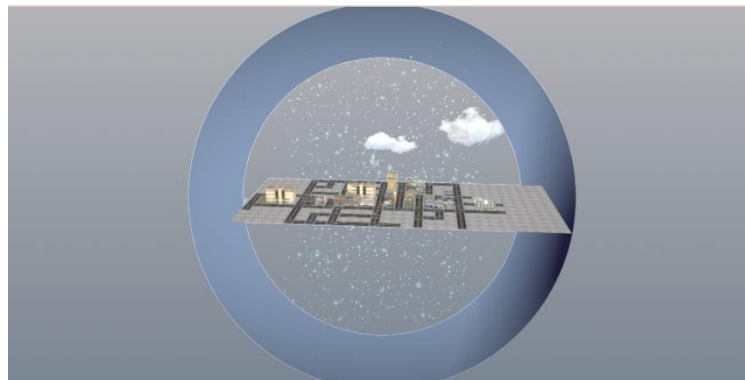
#### 2) 배경 제작

- **GLTF 파일 커스터마이징:** SketchFab 을 통해 오픈 소스 컴포넌트들을 GLTF 파일로 저장한 후, JSX 파일로 변환하여 웹페이지의 테마에 맞게 커스터마이징 하였다. 아래 컴포넌트들을 활용하여 로딩 페이지의 전체 배경을 제작하였다.



로딩 페이지 배경 컴포넌트

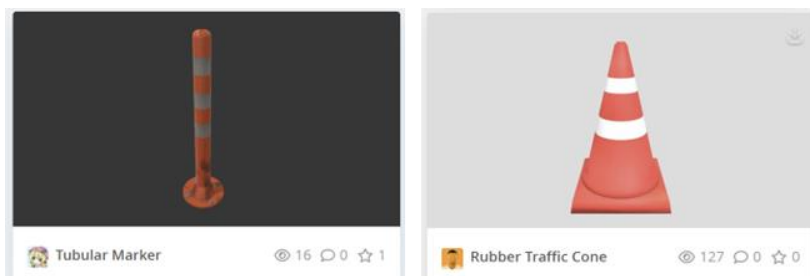
- **Three.js의 Orbit Control 사용** : Three.js의 Orbit Control을 이용해 로딩 페이지의 초기 화면에 구를 생성하고, 사용자가 마우스 드래그 및 줌 기능을 통해 카메라를 회전 및 확대/축소할 수 있도록 구현하였다.



- **하늘 표현 및 조명 추가** : BackGround 함수와 구름 컴포넌트를 조합하여 하늘을 표현하였으며, 배경 색상의 그라데이션과 Sparkle을 추가하여 밤하늘의 별을 표현하였다. 그러나, 3D 공간 상에 조명이 없어 모든 컴포넌트가 검게 보이는 문제가 발생하였다. 이를 해결하기 위해 ambient Light와 directional Light를 추가하고, intensity 파라미터를 조정하여 3D 컴포넌트들이 정상적으로 보이도록 구현하였다.

### 3) 사물 배치

- **지자체 도로부속시설물 (라바콘 및 시선 유도봉)** : 프로젝트의 주제인 도로 시설물 파손 감지와 관련하여 웹사이트 테마에 적합한 라바콘 및 시선 유도봉 모델을 도로 위에 배치하였다. XYZ 축의 좌표 값을 이용하여 이 부속물들을 적절하게 배치하여, 실제 도로 환경을 시각적으로 표현하였다.





- **포트홀** : 프로젝트의 메인 주제인 포트홀 감지를 강조하기 위하여 포트홀 이미지를 이용하여 도로에 배치하였다.



- **배치 결과** : 총 3 가지 종류의 포트홀을 사용하였으며, 라바콘과 시선 유도봉을 도로 위에 배치하여 실제 도로파손 감지 상황에 맞는 화면을 제작하였다.

- **로봇 컴포넌트** : 웹사이트의 사용자 주체인 관리자를 표현하기 위해 로봇 3D 컴포넌트를 추가하였다. 다양한 로봇이나 캐릭터, 자동차를 후보 모델로서 고려하였으나 프로젝트 주제에 가장 부합하는 모델을 선택하였다.



이미지 배치 결과

로봇 컴포넌트

#### 4) 카메라 제어 및 애니메이션 효과 추가

- **Camera Controls 컴포넌트 사용** : Three.js 의 Camera Controls 컴포넌트를 이용하여 카메라의 위치, 시야 및 애니메이션을 제어하였다. 이 컴포넌트를 통해 사용자가 마우스 드래그와 줌 기능을 활용하여 웹사이트 상의 3D 환경을 직관적으로 탐색할 수 있도록 하였다.

- **라이브러리 활용** : 카메라 제어 및 애니메이션 효과를 구현하기 위해 react-three/fiber 와 react-three/drei 라이브러리를 사용하였다. 또한, R3F 애니메이션을 위해 gsap 라이브러리를 사용하여 부드럽고 자연스러운 애니메이션 효과를 추가하였다.

- **초기 카메라 시야 설정** : 로딩 페이지가 처음 렌더링될 때, 멀리서 아래를 내려다보는 시야에서 시작하여, 이후에 가까운 거리에서 정면을 보는 시야로 전환되도록 카메라의 FOV(Field of View) 값을 80 에서 40 으로 감소시켜 자연스러운 시야 전환을 설정하였다.

- **카메라 초점 이동** : 사용자의 몰입감을 높이기 위해 페이지 로딩 후 10 초 동안, 카메라의 초점이 Z 축 기준으로 양수 방향으로 이동하도록 설정하여 점진적으로 가까워지는 시야가 보여지게 하였다.

- **로봇 애니메이션 추가** : 카메라의 초점 이동에 맞추어 로딩 페이지에 배치된 로봇이 도로를 주행하는 애니메이션 효과를 추가하였다.



카메라 초점 이동 화면



로봇 애니메이션 효과

## 5) 로그인 및 회원가입 팝업

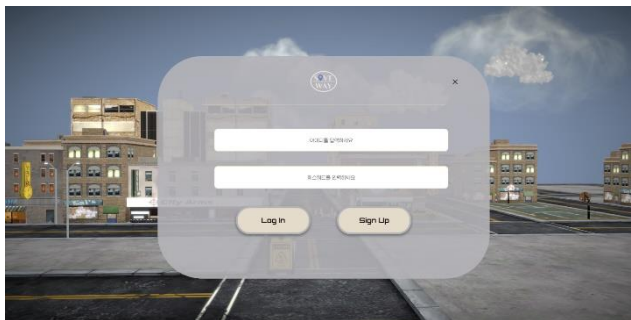
카메라 이동 애니메이션이 끝난 후 사용자에게 로그인 및 회원가입 팝업 창이 표시된다. 이 팝업 창은 2D React 로 구현되었으며 사용자 인증을 요구한다.

- **로그인 우선순위** : 로봇과 카메라 이동 애니메이션이 종료된 후, 로그인 팝업이 우선적으로 표시된다. 회원가입이 완료된 사용자는 이 창에서 로그인할 수 있고, 회원가입이 되지 않은 사용자는 회원가입 절차를 먼저 진행해야 한다.

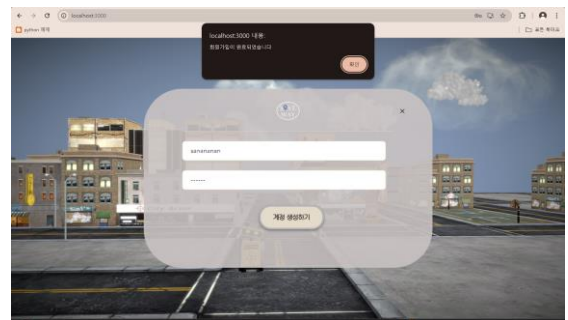
- **서버 통신** : 로그인과 회원가입 과정에서 서버와의 통신을 위해 POST 메소드를 사용하였다. Fetch API 를 활용하여 서버와 데이터를 주고받으며, 사용자 입력 데이터를 데이터베이스에 저장할 수 있도록 구현하였다.

- **계정 생성** : 웹사이트 사용자인 관리자가 ID 와 비밀번호를 입력한 후, 계정 생성하기 버튼을 누르면 서버의 API 엔드포인트로 POST 요청이 전송된다. 요청이 성공하면 해당 정보가 admin\_account 테이블에 저장되며, 회원가입이 정상적으로 처리된 경우, 사용자에게 alert 창으로 알림이 표시된다.

- **회원가입 조건 및 로그인 처리** : 회원가입 시 ID 와 비밀번호는 모두 영문 8 자 이상으로 설정해야 한다. 회원가입이 완료된 사용자가 로그인하면, 메인 페이지로 이동할 수 있고, 회원가입이 완료되지 않은 사용자가 로그인하려고 하면, 재로그인을 요구하는 알림창이 표시된다.



로그인 팝업 화면



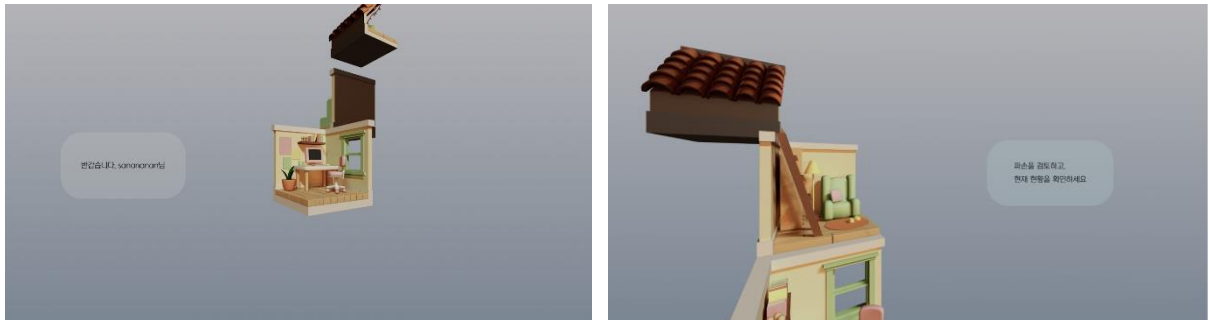
회원가입 팝업 화면

## ● 메인 페이지

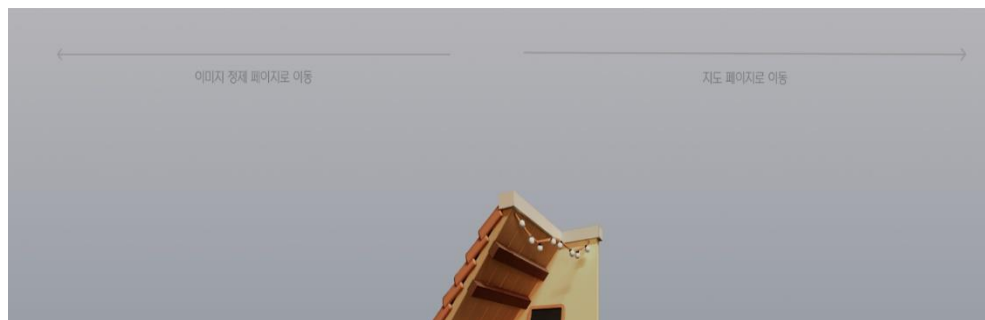
로그인이 완료되어 메인 페이지로 이동하면 사용자 ID 가 뜬다. 처음 이용하는 웹사이트 사용자를 위하여 웹페이지에 대한 소개를 넣어 구성하였다.

- **Scroll 컴포넌트** : 내부에서 3 개의 Section 컴포넌트를 렌더링하도록 구성하였다. 각 섹션은 스크롤 위치에 따라 투명도가 변화하며, 첫번째 섹션은 사용자의 ID 와 인사말을 표시하고 두번째 섹션은

웹사이트에 대한 간략한 소개를 제공, 세번째 섹션은 이미지 정제 페이지와 지도 페이지로 이동할 수 있는 기능이 배치되었다.



메인페이지 실행 화면 : 웹 소개



메인페이지 실행 화면 : 페이지 이동

## ● 3D Web-Page 개발 이슈사항

### 1) 로딩페이지 개발 시 이슈사항

#### - 이슈 1 : 포트홀 배치 시 이미지 컴포넌트의 색상 불일치

로딩 페이지의 도로에 포트홀을 추가하려 했으나, SketchFab 에서 적절한 포트홀 모델의 GLTF 색상 코드가 배경의 도로 색상과 일치하지 않는 문제가 발생하였다. 기존의 RGB 값을 사용하는 방법이 아닌, texture image 가 존재하여 색상을 직접 변경하는 것이 불가하였다.

⇒ **해결방안** : 여러 가지 색상의 포트홀 모델을 조합하여 자연스럽게 표현하고, 도로 색상과 조화를 이루는 모델을 선택하여 3D 공간 상에 배치하였다. XYZ 축 좌표를 이용해 포트홀을 도로 위에 자연스럽게 배치하여 실제 도로 파손 모습을 반영하였다.

## 8-2. 2D Web-Page

정제페이지와 지도페이지는 기능을 사용자 친화적으로 구현하는 것이 가장 중요한 요소이기에 2D 구조를 사용하여 높은 접근성을 가지도록 구성하였다.

### ● 이미지 정제 페이지

**1) dateview\_page** : 서버에서 받아 온 이미지 데이터를 날짜와 신고유형 별로 그룹화하여 사용자에게 보여주는 역할을 하는 페이지이다. 사용자가 특정 날짜에 업로드 된 이미지를 확인할 수 있다. DateView 컴포넌트를 통해 그룹화 된 데이터를 화면에 나타나게 하였다.

- DateView 컴포넌트 : fetchData 함수를 사용하여 서버에서 데이터를 가져온다. 날짜와 신고유형별로 그룹화하여 날짜, 이미지 개수, 해당 날짜에 속한 이미지 목록을 포함하는 그룹을 생성한다.

- 페이지네이션 : 한 페이지에 총 아이템 수 이상으로 표시되지 않도록 하고 다음 페이지로 이동하도록 한다.

- 이미지 불러오기 : 사용자가 특정 날짜를 선택하면 해당 날짜의 모든 이미지에 대해 서버로부터 원본 데이터를 가져온다. 이 때 이미지는 Blob 형식으로 받아온 뒤 Base64 로 변환하여 상태에 저장된다. 여기서 json 파일로 불러왔으나, 이미지 파일인 원본 데이터를 가져오지 못하여 Blob 형식으로 받아온 후 base64 로 변환하는 방식을 사용하였다.

- 데이터 삭제 기능 : 사용자가 선택한 날짜에 속한 모든 이미지를 삭제하는 기능이다.

**2) refine\_page**: 서버에서 가져온 이미지들을 검토하고, 필요에 따라 이미지를 삭제하거나 세부 정보를 수정하도록 하는 페이지이다.

- 이미지 표시 기능 : 페이지에 이미지를 표시할 이미지를 선택하고 화면에 보여준다.

- 이미지 선택 및 삭제 : 이미지에 체크박스를 설정하고, 체크박스를 선택하여 이미지를 서버에서 삭제할 수 있도록 하였다. 위 dateview 페이지와 같이 서버의 DELETE 메소드를 사용하여 서버 데이터 삭제 요청을 보내는 방식으로 구현하였다.

- 팝업 창 관리 : 사용자가 이미지를 클릭하면, 이미지의 상세 정보를 확인할 수 있는 팝업 창이 열리도록 구현하였다.

- 페이지네이션 : dateview 페이지와 같이 한 페이지에 나타나는 이미지 수를 제한하고, 다음 페이지로 이동할 수 있도록 탐색 버튼을 생성하였다.

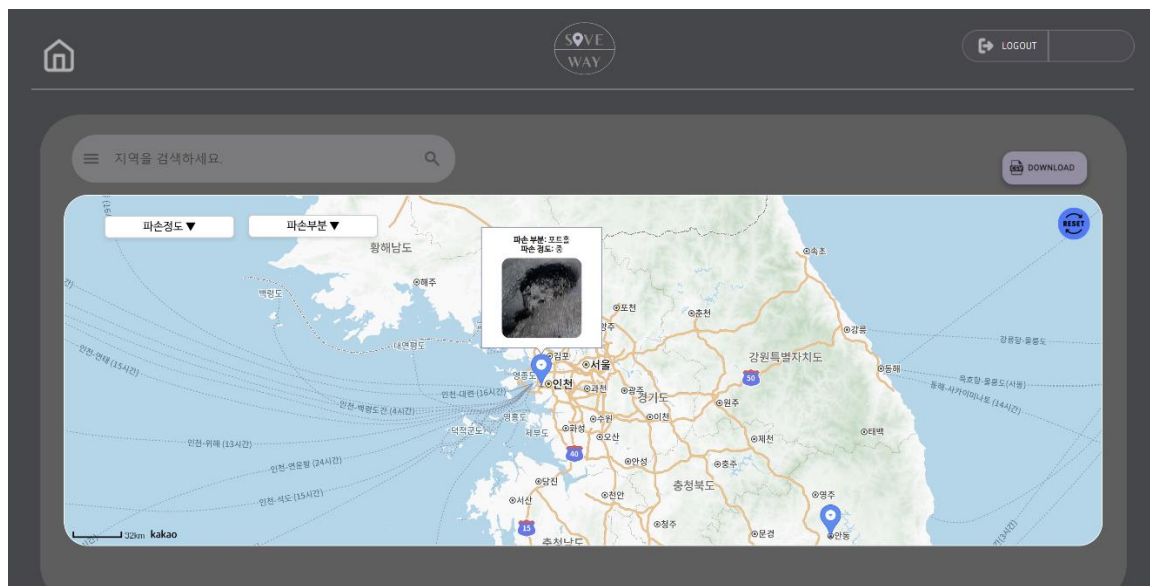
**3) image\_popup** : refine page 에서 전체 이미지 중 한 개를 선택하여 클릭할 경우, 개별 이미지에 대한 팝업창을 띄운다. 기본 레이아웃이 되는 PopupInside 와 파손 종류를 관리하기 위한 CustomSelect, 파손 정도를 관리하기 위한 CustomD 컴포넌트로 구성하였다. 서버에서 올바르게 데이터를 불러오기 위하여 GET/POST 메소드를 사용하여 통신하였다. 이 팝업창을 통해 관리자는 어느 정도의 파손인지 판별하고 저장하여 서버로 전송한다.

- 파손 종류 classification : 분류가 올바르게 된 경우 파손 종류에 classification 된 데이터가 뜨도록 하였고, 분류가 올바르게 되지 않은 경우 default 값으로 placeholder 로 '파손 종류를 선택하세요' 라는 문구가 뜨도록 설계하였다.

- 데이터 정제 후 저장 : SAVE 버튼을 눌러 이미지에 대한 정보를 저장하였을 시, 데이터베이스로 전송되며 refine\_page 에서 이미지를 선택하여 팝업창이 떴을 때 파손 종류, 파손 정도가 저장되어 입력되어 있는 상태가 된다.

## ● 지도 페이지

지도 페이지는 도로 파손 정보를 지도 상에 시각적으로 표시한다. 이 페이지에서는 카카오맵 API 를 활용하여 사용자가 선택한 지역의 파손 정보를 지도에 마커로 표시하고 이를 필터링 할 수 있는 기능을 제공한다.



### 1) MapPage

- SearchBar : 사용자가 특정 지역을 검색할 수 있도록 입력 창을 제공하고, 입력된 지역명을 'place' 상태로 관리하며 'KakaoMap' 컴포넌트에 전달한다.

- KakaoMap : 전달된 지역명을 기반으로 카카오맵 API 를 통해 해당 위치로 지도를 이동시키고, 데이터베이스에서 가져온 파손 정보를 지도에 마커로 표시한다.

### 2) KakaoMap

- 데이터베이스 연동 및 마커 데이터 처리 : 서버에서 데이터를 가져와 지도에 표시할 마커 데이터로 처리하도록 구현하였다. [getDetails/](#) 엔드포인트를 통해 각 이미지의 파손 정도 데이터를 서버에서 받아오고, [images/total](#) 엔드포인트를 호출하여 모든 이미지의 메타데이터와 위치 정보를 가져왔다. 마커 데이터 구성은 각각의 이미지 ID 를 기준으로 결합되어 파손유형, 파손정도, GPS 정보 등을 포함한 마커 데이터로 구성하였다. 이 데이터는 markderData 상태에 저장된다.

- 지도에 마커 표시 : markerData 를 기반으로 지도에 마커를 표시하고, 사용자가 특정 지역을 검색할 때 지도의 중심을 해당 위치로 이동시킨다. 마커를 표시하고 지도를 초기화, 포커싱하는 기능을 구현하여 하여 사용자가 데이터를 지도에서 확인 가능하도록 하였다.

- 검색 및 필터링 : 사용자가 파손 정보(파손정도, 파손부분)를 필터링하고 검색 기능을 통해 특정 지역으로 이동할 수 있도록 기능을 구현하였다.

- 맵 리셋 기능 : 지도를 초기 상태로 리셋하는 기능을 구현하여, 모든 필터를 초기화하고 지도를 초기 상태로 돌리는 리셋 버튼을 생성하였다.

## ● 2D Web-Page 개발 이슈사항

### 1) image\_popup 페이지 개발 시 이슈사항

#### - 이슈 1 : 422 Unprocessable Entity 에러 발생

로그인과 회원가입 시, POST 메소드로 데이터를 전송할 때 application/json 형식으로 모든 데이터를 하나로 묶어 보냈으나, maintenance-schedule 엔드포인트로 동일한 방식으로 POST 요청을 보냈을 때, 422 Unprocessable Entity 에러가 발생하는 문제가 발생하였다. Request Body 에 존재하는 속성과 속성의 타입 형식에 맞추어 작성하였음에도 불구하고 에러가 지속되었다.

⇒ **해결방안** : http://3.39.6.45:8000/v1/docs# 를 확인한 결과, 이 에러는 body 로 모든 속성을 한꺼번에 묶어서 보낼 때 발생하는 것으로 확인되었다. 이 문제를 해결하기 위해, FormData 를 사용하여 속성을 개별적으로 묶어 전송하였을 때 데이터가 정상적으로 전송됨을 확인할 수 있었다.

### 2) map\_page 개발 시 이슈사항

#### - 이슈 2 : 파손 정도 및 파손 종류별 필터링 시 배열 초기화 문제 발생

map\_page 내에서 파손 정도와 파손 종류를 선택한 후 Submit 버튼을 클릭할 때, 서버의 'get Details' 및 'images/total' 엔드 포인트에 GET 메소드를 전송하여 'details' (파손 정도)와 'classification result' (파손 종류) 정보를 얻어와 카카오 맵 상의 마커를 필터링하여 나타내야 한다. 그러나 1 번 이상 필터링을 진행할 경우, Kakao map 컴포넌트에서 Filter box 컴포넌트로 props 전달 시 이전에 선택하였던 파손 정도와 종류가 남아있어 제대로 필터가 되지 않는 현상이 발생하였다. 또한, Reset 버튼을 클릭할 때도 마찬가지로 예전 선택 사항이 저장되어 초기화되지 않는 문제가 발생하였다.

⇒ **해결방안** : Reset 버튼의 클릭 이벤트 발생 시에 현재 선택된 파손 종류와 정도가 저장되어 있는 배열인 'damage Parts'와 'damage Levels'를 빈 배열로 초기화해주는 코드를 추가하였다. Submit 버튼의 클릭 이벤트 발생 시에 현재 선택된 파손 종류와 정도를 서버에 넘겨주고, 그 이후에 배열을 초기화하여 비워주었다.

결과적으로 여러 옵션을 선택하고 Submit 을 할 경우에는 예전 선택이 보존되지만, Submit 과 Reset 버튼을 클릭할 시에는 temporary 하게 저장된 배열이 비워지기 때문에 이전 선택이 보존되지 않아 필터링이 올바르게 수행된다.

## V. 결과

### 1. 서비스 결과

어플리케이션, 서버, 웹 페이지가 모두 성공적으로 연동되어 원활히 작동하는 것을 확인할 수 있다. 어플리케이션에서 수집된 도로 파손 데이터는 서버로 성공적으로 전송되며, AI 모델이 이를 처리하여 데이터베이스에 저장된다. 웹 페이지에서는 이 데이터를 기반으로 사용자가 필요한 정보를 가공하고 활용할 수 있다.

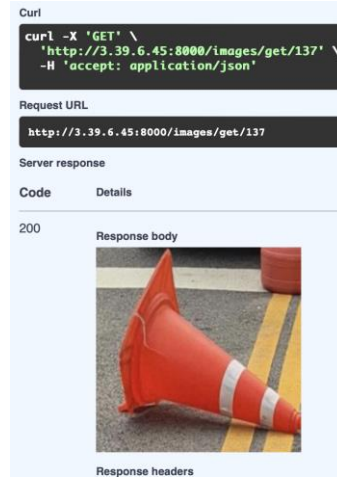
#### ● 어플리케이션 실행 결과

1) **Splash 화면, 메인 화면** : 어플리케이션 첫 실행 시 스플래시 화면이 뜨고, 기획하였던 신고촬영, 주행신고, 기타신고, 신고전화, 웹 페이지 바로가기 기능이 포함된 메인 화면이 실행된다.





2) 신고 촬영 기능 : 메인 화면에서 신고 촬영 아이콘 클릭 시 카메라 기능 화면으로 전환되며, 파손물 감지 시 바운딩 박스가 나타나고 촬영 버튼 클릭 시 서버로 데이터가 전송된다. GPS 새로고침 버튼 클릭 시 현재 위치가 새로고침 된다.



신고 촬영 실행 화면

서버 업로드 결과

3) 주행 신고 기능 : 메인 화면에서 주행 신고 아이콘 클릭 시 영상 녹화 화면으로 전환되며, 녹화 시작 후 파손물 감지 시 바운딩 박스가 나타나고, 자동으로 서버로 데이터가 전송된다. GPS 새로고침 버튼 클릭 시 현재 위치가 새로고침 된다.



앱 실행 화면

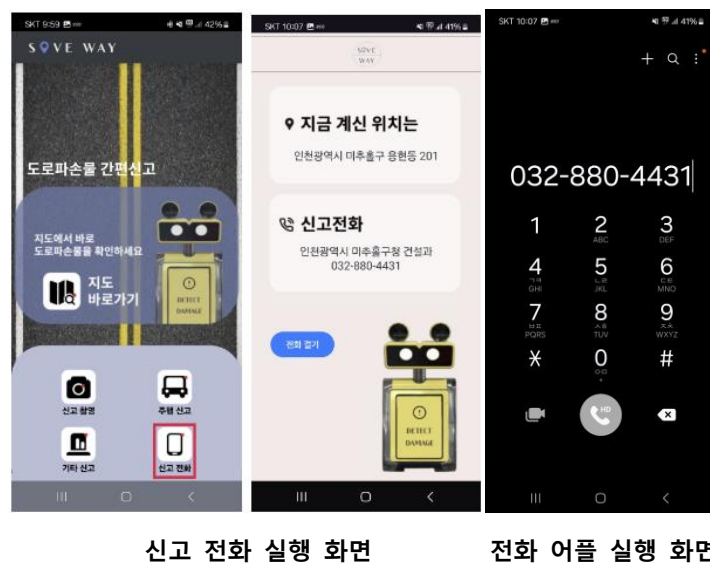
서버 업로드 결과

4) 기타 신고 기능 : 메인 화면에서 기타 신고 아이콘 클릭 시 민원 내용을 입력하고 갤러리에서 사진을 불러와 데이터를 서버로 전송할 수 있다. 이 화면에서는 민원인의 현재 위치가 표시된다.

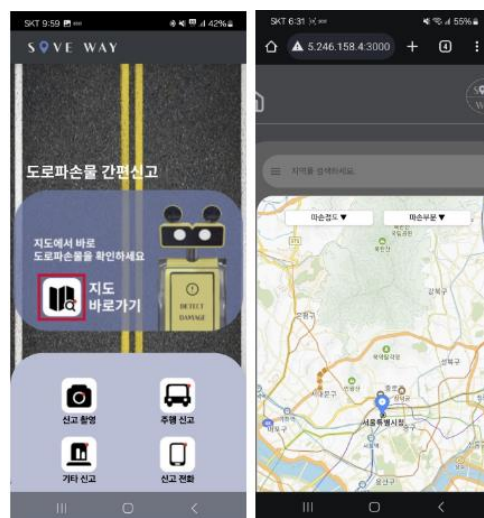




5) 신고 전화 기능 : 메인 화면에서 신고 전화 아이콘 클릭 시 현재 위치를 기반으로 담당 부서와 연락처를 알 수 있고, 전화 걸기 버튼 클릭 시 휴대폰의 전화 기능이 실행된다.



6) 웹 페이지(지도) 바로가기 : 지도 바로가기 배너 클릭 시 도로 파손물 정보를 가지고 있는 웹 페이지의 관리자 지도에 접근할 수 있다.



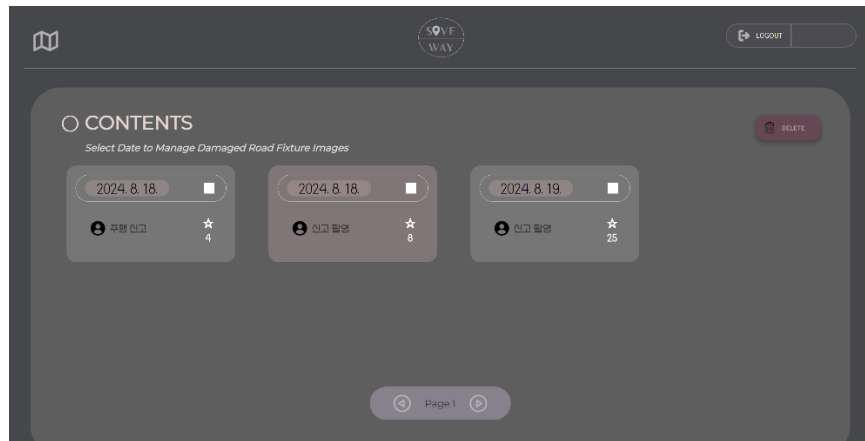
## ● 웹페이지 실행 결과

1) 로딩 페이지, 메인 페이지 : 메인 페이지 3D 애니메이션 실행 후 로그인/회원가입 창이 뜬다. 로딩 페이지에서는 로그인 된 관리자가 기능 페이지로 이동할 수 있도록 안내한다.



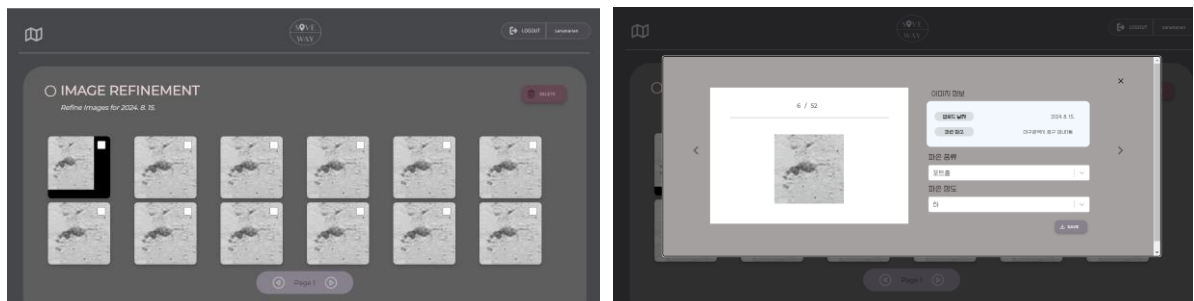
## 2) 정제 페이지

2-1) 이미지 관리 페이지 : 어플리케이션을 통해 서버로 넘어온 데이터가 날짜 별, 신고 유형 별로 선택되도록 화면에 그룹화 되어있다. 이 그룹을 기준으로 삭제, 혹은 이미지 정제 페이지로 이동할 수 있다.



2-2) 이미지 정제 페이지 : 2-1 에서 선택한 그룹에 대해 이미지를 삭제하거나 이미지 정보를 수정, 추가할 수 있다. IMAGE REFINEMENT 에서 이미지를 선택하면 팝업창이 생성되어 이미지의

GPS 정보, 업로드 날짜, 파손구분을 확인할 수 있고 파손정도를 선택하여 저장하면 지도페이지의 지도에 마커가 생성된다.



3) 지도 페이지 : 이미지 정제 페이지에서 저장한 도로시설물 파손 데이터의 위치가 지도 위에 마커로 띄워진다. 지역을 검색하면 해당 지역으로 지도가 포커싱된다. 필터 기능을 통해 원하는 데이터만 선택하여 조회할 수 있다.

## 2. 평가 및 느낀점

### ● 성과 및 기대효과

- 본 프로젝트를 통해 개발한 도로 시설물 파손 모니터링 시스템은 스마트폰 애플리케이션과 웹 페이지의 연동을 통해 도로 파손에 대한 파손 정도, 파손 종류, 파손 위치 정보를 수집하고 관리하는데 성공적으로 설계되었다. 어플리케이션에서 수집된 데이터는 웹 페이지에서 효과적으로 가공되어 관리자가 도로 상태를 한눈에 파악하고 신속하게 대응할 수 있게 된다.

본 시스템의 도입으로 인해 도로 관리의 효율성이 높아지고, 소모되는 비용과 시간도 줄어들 것을 기대할 수 있다. 또한, 도로 파손으로 인해 발생하던 차량 피해와 사고 등이 줄어들 것을 예측할 수 있고, 최종적으로 운전자의 안전과 원활한 교통 유지에도 도움이 될 것으로 기대할 수 있다.

### ● 자체 평가 의견

- **기획 의도와 의 부합 및 달성도:** 도로 파손을 감지하고 효과적으로 관리할 수 있는 도구를 제공하여 도로 관리의 효율성을 높이고 운전자의 안전을 강화하고자 했던 기획 의도에 충분히 부합하다.

- **실무 활용 가능 정도:** 스마트폰 애플리케이션과 웹 페이지가 원활하게 연동되며 데이터베이스와의 통합도 잘 이루어졌다. 그러나 도로 시설물 데이터가 포트홀만큼 충분하지 않아 파손 정도에 따른 인식률이 비교적 낮다. 포트홀에 대한 인식은 90%이상 되었기 때문에 yolo 모델은 제대로 작동하고 있으며, 타 파손에 대한 추가적인 데이터 보강 시 도로 시설물의 인식도 원활할 것으로 예상할 수 있다. 따라서, 파손종류의 인식에 대한 정확도를 높이기 위해서 훈련을 위한 데이터셋에 추가적인 보강이 필요할 것으로 판단된다. 또한, gps의 오차범위가 존재하기 때문에, 이를 최소화 하기 위한 추가적인 방법도 필요할 것으로 보인다.

- **완성도:** 개발 과정에서 발생한 다양한 기술적 문제를 해결하였으며, 최종 시스템은 안정적으로 작동하는 것을 확인하였다. 실시간 데이터 수집과 시각화, 사용자 편의를 고려한 다양한 기능이 효과적으로 구현되었으며, 서버 간 통신과 모델 분리 등 복잡한 기술적 문제도 잘 해결되어 시스템이 원활하게 동작하는 것을 확인하였다.

## 3. 첨부

### ● 개발코드

#### 1) GIT


- 백엔드 docker : <https://hub.docker.com/repository/docker/somany/saveway/general>

- 웹 프론트엔드 : [https://github.com/sanananan3/KEB3\\_frontend.git](https://github.com/sanananan3/KEB3_frontend.git)

## 2) 구글드라이브 : 웹 / 앱 / 백엔드 개발코드

- [https://drive.google.com/drive/folders/1IKZtY7DL5oc1ZOIEwvn0FjC\\_N-JIBWsW?usp=sharing](https://drive.google.com/drive/folders/1IKZtY7DL5oc1ZOIEwvn0FjC_N-JIBWsW?usp=sharing)

---

 ★KEB\_SAVEWAY 제출파일

---

### ● 시연영상

: 구글드라이브 혹은 유튜브 링크를 통해 접속 가능하다.

#### 1) 앱

구글드라이브 : [앱 시연영상 구글드라이 링크](#)

유튜브 : <https://youtu.be/YUbR1JZXTd8>

#### 2) 웹

구글드라이브 : [웹 시연영상 구글드라이브 링크](#)

유튜브 : [https://youtu.be/q8J\\_hO4LqJM](https://youtu.be/q8J_hO4LqJM)