

# 자바 스프링 부트 프로젝트와 파이썬 AI 프로젝트 연결하기

Java to Python AI Server  
AJAX, MQTT  
Real-time Streaming



# 과정 안내

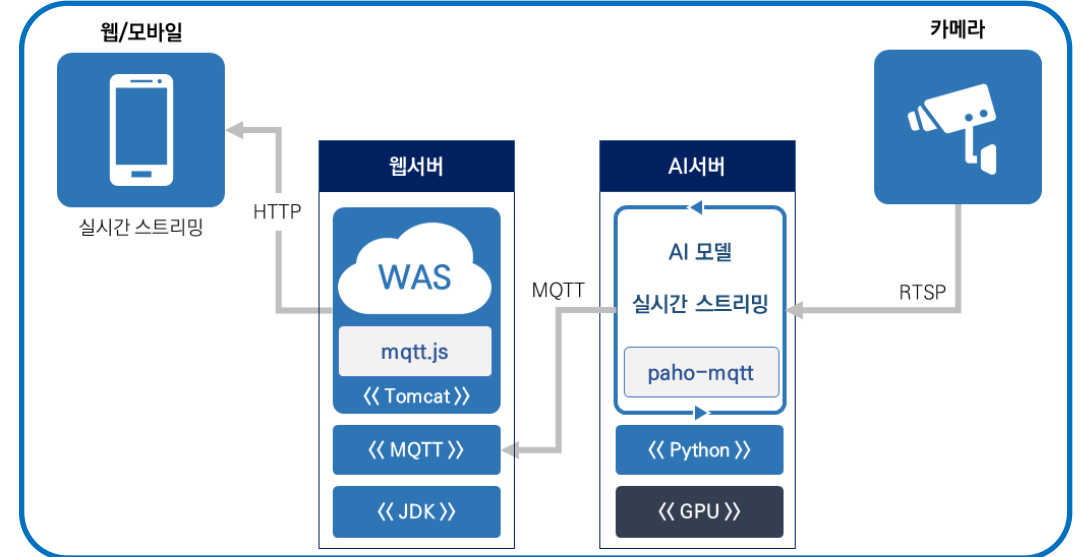
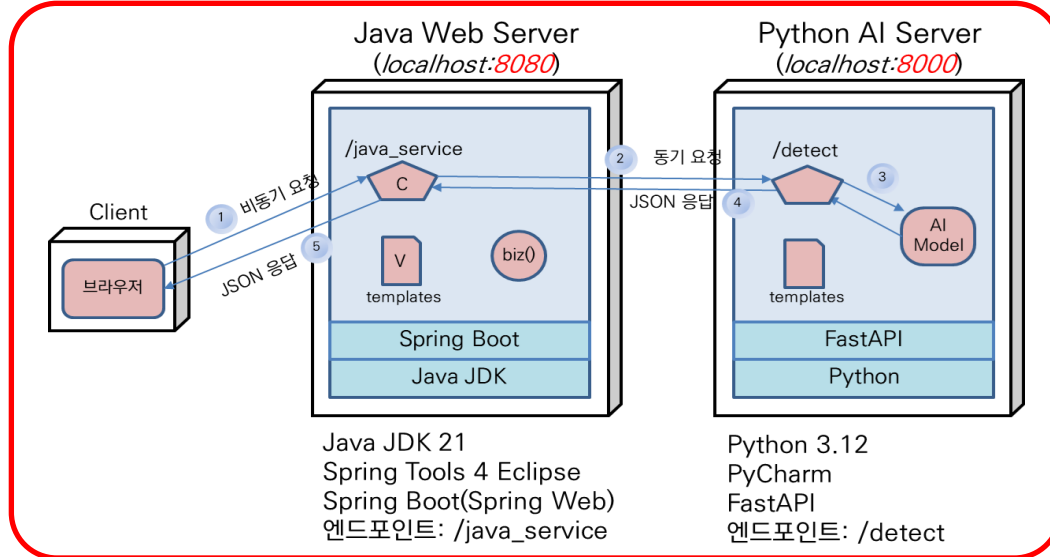
자바 스프링 부트 프로젝트와 파이썬 AI 프로젝트 연결하기

인프런 강의영상: <https://inf.run/izXES>

## 자바 스프링 부트 프로젝트와 파이썬 AI 프로젝트 연결하기

- ▶ 자바 스프링 부트 프로젝트와 파이썬 Fast API를 이용한 웹 애플리케이션 개발 방법을 설명합니다.
- ▶ 자바에서 파이썬 AI 서버에 데이터를 보내고 받는 방법을 설명합니다.
- ▶ 영상에서 객체를 탐지하고, 결과를 MQTT로 서버에 전송하여 자바 웹 어플리케이션에서 즉시 시각화합니다.

## 주요 아키텍처



# 1. 웹 애플리케이션 프로젝트 만들기

자바 스프링 부트 프로젝트와 파이썬 AI 프로젝트 연결하기

## ✓ FastAPI를 이용한 파이썬 웹 애플리케이션

- ▶ 파이썬 개발환경 구성
- ▶ FastAPI를 이용한 웹 애플리케이션

## ✓ Spring Boot를 이용한 자바 웹 애플리케이션

- ▶ JDK와 이클립스를 이용한 개발 환경 구성
- ▶ 스프링 부트를 이용한 웹 애플리케이션

# 1절. FastAPI를 이용한 파이썬 웹 애플리케이션

## 1장. 웹 애플리케이션 프로젝트 만들기



# 1.1. 개발환경

## 1. 웹 애플리케이션 프로젝트 만들기 / 1절. FastAPI를 이용한 파이썬 웹 애플리케이션

### ✓ Python 인터프리터

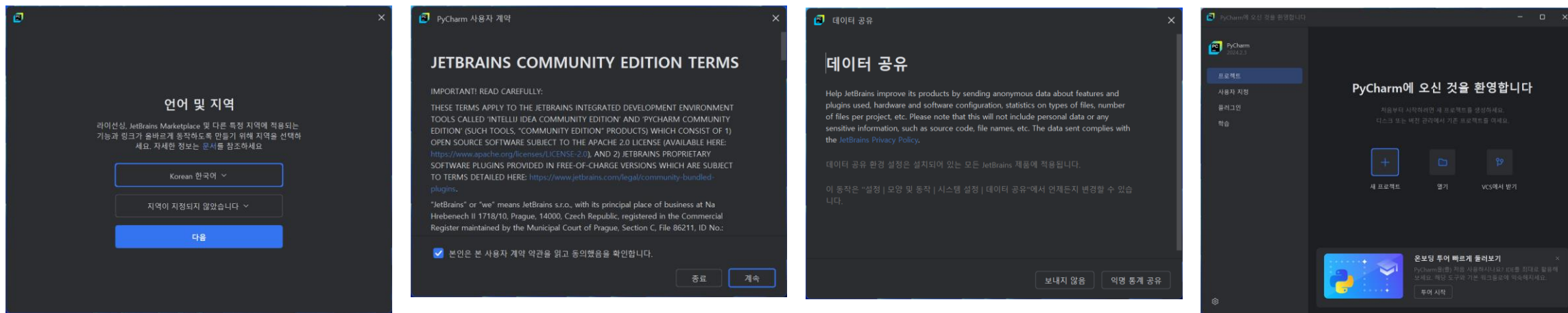
<https://www.python.org>

설치 시 "Add Python to PATH" 옵션을 선택

### ✓ PyCharm

<https://www.jetbrains.com/> -> Developer Tools -> PyCharm -> PyCharm Community Edition

<https://www.jetbrains.com/pycharm/> -> PyCharm Community Edition



## 1.2. FastAPI 설치

1. 웹 애플리케이션 프로젝트 만들기 / 1절. FastAPI를 이용한 파이썬 웹 애플리케이션

```
pip install fastapi uvicorn
```

### ✓ ASGI

- ASGI(Asynchronous Server Gateway Interface)
- 파이썬에서 비동기 웹 서버와 웹 애플리케이션 간의 인터페이스 표준.
- ASGI는 기존 WSGI(Web Server Gateway Interface)의 비동기 버전임.
- 파이썬에서 비동기 처리를 지원하는 웹 애플리케이션을 구축하기 위해 설계되었음.

### ✓ ASGI의 주요 특징

- **비동기 지원:** ASGI는 비동기 코드 실행을 지원하여 높은 성능과 동시성을 제공합니다. 이는 특히 웹소켓이나 서버 푸시와 같은 비동기 통신이 필요한 애플리케이션에 유용합니다.
- **범용성:** ASGI는 HTTP뿐만 아니라 WebSocket, gRPC와 같은 다른 프로토콜도 지원합니다.
- **유연성:** ASGI 애플리케이션은 다양한 서버 및 프레임워크와 호환되며, 모듈식으로 구성할 수 있습니다.

### ✓ FastAPI와 ASGI

- FastAPI는 ASGI 표준을 따르는 웹 프레임워크
- FastAPI 애플리케이션은 비동기 처리를 기본으로 하며, Uvicorn과 같은 ASGI 서버를 사용하여 높은 성능을 제공함

# 1.3. FastAPI 애플리케이션 생성

1. 웹 애플리케이션 프로젝트 만들기 / 1절. FastAPI를 이용한 파이썬 웹 애플리케이션

main.py

```
1 from fastapi import FastAPI
2
3 app = FastAPI()
4
5
6 @app.get("/")
7 async def read_root():
8     return {"Hello": "World"}
9
10
11 @app.get("/items/{item_id}")
12 async def read_item(item_id: int, q: str = None):
13     return {"item_id": item_id, "q": q}
14
```



'/': 루트 경로에 대한 GET 요청을 처리합니다.

- '/items/{item\_id}': 동적 경로 매개변수(Path Parameters) item\_id를 사용하여 특정 아이템을 조회합니다. 경로 매개변수는 중괄호 '{}' 안에 변수명을 적어 정의합니다.



FastAPI는 경로 매개변수와 쿼리 매개변수를 함께 사용할 수 있습니다. 경로 매개변수는 URL 경로의 일부로 사용되고, 쿼리 매개변수는 URL의 쿼리 문자열로 전달됩니다.

- item\_id: 경로 매개변수입니다.
- q: 쿼리 매개변수(기본값은 None)입니다.



# 1.4. 데이터 모델링

1. 웹 애플리케이션 프로젝트 만들기 / 1절. FastAPI를 이용한 파이썬 웹 애플리케이션

main.py

```
1 from fastapi import FastAPI
2 from pydantic import BaseModel
3
4 app = FastAPI()
5
6
7 class Item(BaseModel):
8     name: str
9     description: str = None
10    price: float
11    tax: float = None
12
13
14 @app.post("/items/")
15 async def create_item(item: Item):
16     return item
17
```



- BaseModel은 pydantic 라이브러리의 핵심 클래스입니다.
- pydantic은 데이터 유효성 검사와 설정 관리에 사용되는 파이썬 라이브러리로, 데이터 모델링을 쉽고 강력하게 할 수 있도록 도와줍니다.
  - FastAPI는 pydantic을 사용하여 데이터 유효성 검사를 수행합니다.
  - 7라인에 정의된 클래스를 15라인의 앱 함수 매개변수로 설정하면 FastAPI가 요청 본문을 자동으로 Item 모델로 변환하고, 유효성 검사를 수행합니다. 잘못된 데이터가 입력되면 자동으로 '422 Unprocessable Entity' 응답을 반환합니다.



# 1.5. FastAPI 문서화

## 1. 웹 애플리케이션 프로젝트 만들기 / 1절. FastAPI를 이용한 파이썬 웹 애플리케이션

- FastAPI는 자동으로 API 문서를 생성합니다.
- 애플리케이션을 실행한 후 브라우저에서 다음 URL을 방문하여 API 문서를 확인할 수 있습니다.
  - Swagger UI: <http://127.0.0.1:8000/docs>
  - ReDoc: <http://127.0.0.1:8000/redoc>

main.py

```
1 from fastapi import FastAPI
2
3 app = FastAPI(
4     title="My API",
5     description="This is a sample API",
6     version="1.0.0",
7     docs_url=None,          # Swagger UI 비활성화
8     redoc_url=None         # ReDoc 비활성화
9 )
```



Swagger UI와 Redoc는 API 문서화를 위한 도구입니다. 이들은 RESTful API의 문서와 사용 방법을 시각적으로 표현하여 개발자들이 쉽게 이해하고 테스트할 수 있도록 도와줍니다.



FastAPI는 기본적으로 Swagger UI와 ReDoc을 활성화합니다. 하지만 원하는 경우 왼쪽의 코드처럼 이를 비활성화하거나 커스터마이징할 수 있습니다.

# 1.6. FastAPI 미들웨어

1. 웹 애플리케이션 프로젝트 만들기 / 1절. FastAPI를 이용한 파이썬 웹 애플리케이션



## FastAPI 미들웨어

- 요청(request)과 응답(response) 사이에서 특정 작업을 수행하는 데 사용되는 함수 또는 클래스
- 모든 요청에 대해 실행되며, 요청을 처리하기 전에 또는 응답을 반환하기 전에 특정 작업을 수행할 수 있음
- 예를 들어, 로깅, 인증, CORS 처리, 압축 등이 미들웨어를 통해 구현될 수 있음

```
5 app = FastAPI()
6
7
8 class LoggingMiddleware(BaseHTTPMiddleware):
9     async def dispatch(self, request, call_next):
10         logging.info(f"Req: {request.method} {request.url}")
11         response = await call_next(request)
12         logging.info(f"Status code: {response.status_code}")
13         return response
14
15 app.add_middleware(LoggingMiddleware)
```

# 1.7. FastAPI 종합 예제

1. 웹 애플리케이션 프로젝트 만들기 / 1절. FastAPI를 이용한 파이썬 웹 애플리케이션

main.py

```
1  from fastapi import FastAPI, HTTPException
2  from pydantic import BaseModel
3
4  app = FastAPI()
5
6  class Item(BaseModel):
7      name: str
8      description: str = None
9      price: float
10     tax: float = None
11
12     items = {}
13
14     @app.get("/")
15     async def read_root():
16         return {"Hello": "World"}
17
18     @app.get("/items/{item_id}")
19     async def read_item(item_id: int):
20         if item_id not in items:
21             raise HTTPException(status_code=404, detail="Item not found")
22         return items[item_id]
23
24     @app.post("/items/")
25     async def create_item(item: Item):
26         item_id = len(items) + 1
27         items[item_id] = item
28         return {"item_id": item_id, **item.dict()}
```

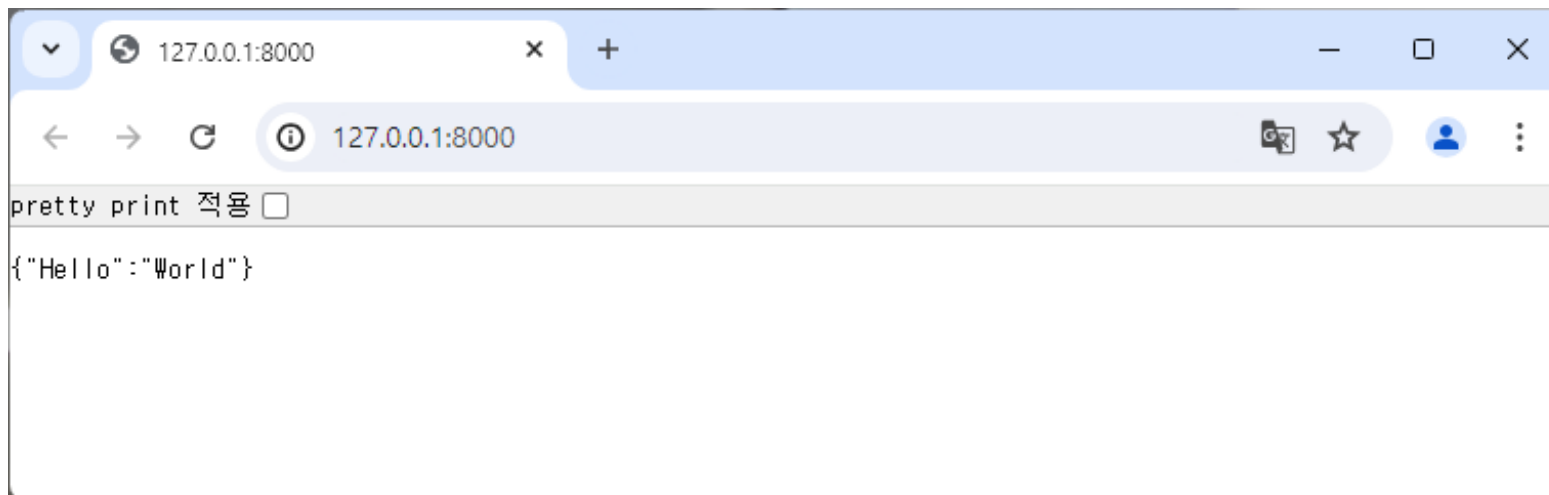


더 많은 기능과 세부 설정은 FastAPI 공식 문서(<https://fastapi.tiangolo.com/>)를 참고하세요.

# 1.8. 애플리케이션 실행

1. 웹 애플리케이션 프로젝트 만들기 / 1절. FastAPI를 이용한 파이썬 웹 애플리케이션

```
uvicorn main:app --reload
```



서버의 실행되는 포트를 바꾸고 싶다면 --port 옵션을 추가하세요.

```
uvicorn main:app --host 0.0.0.0 --port 8001
```

## 2절. Spring Boot를 이용한 자바 웹 애플리케이션

### 1장. 웹 애플리케이션 프로젝트 만들기

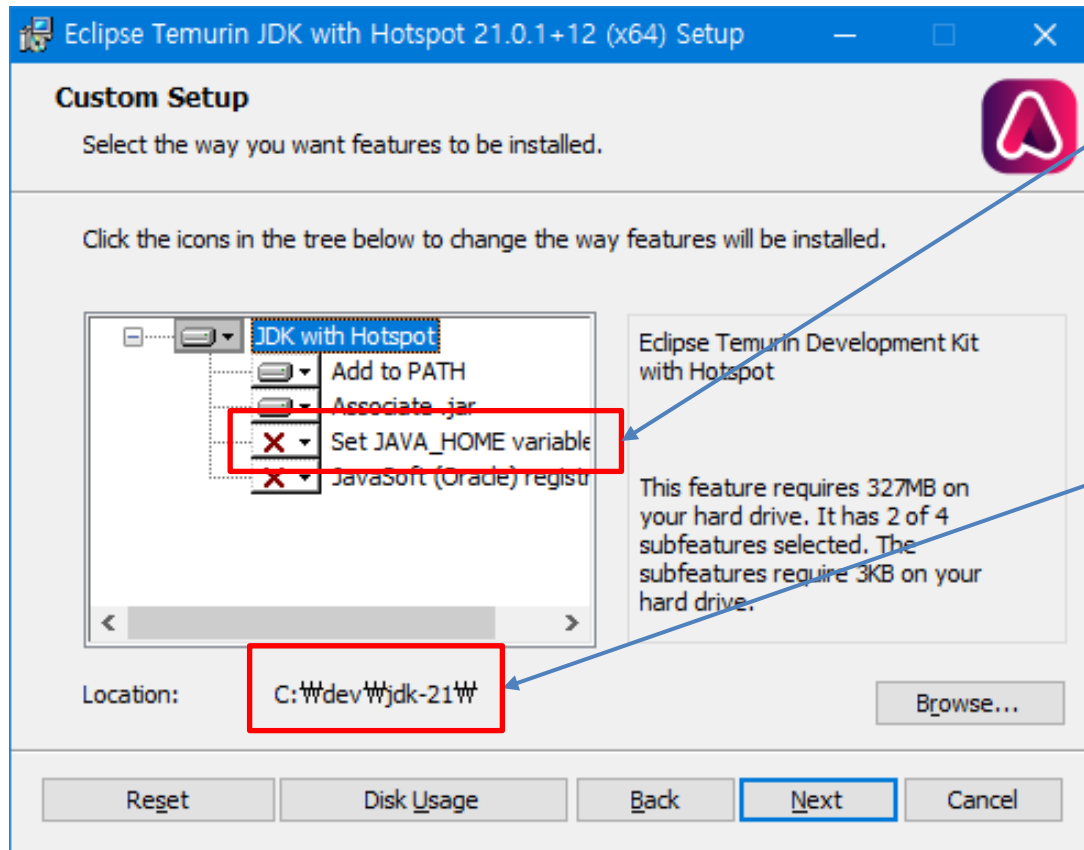


# 2.1. JDK 설치

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

## ✓ JDK

<https://adoptium.net/> -> Latest LTS Release 다운로드  
<https://adoptium.net/temurin/releases/?version=21>

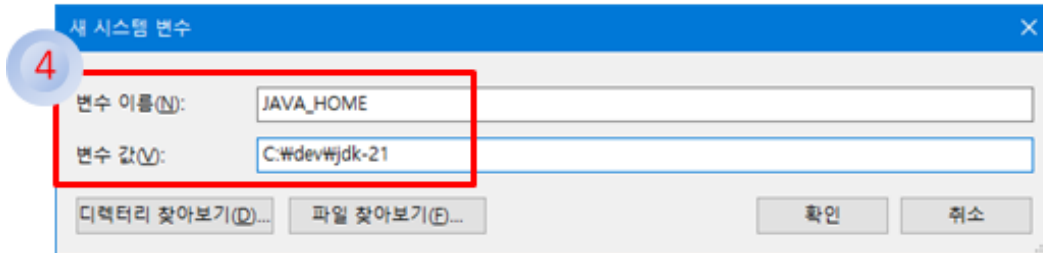
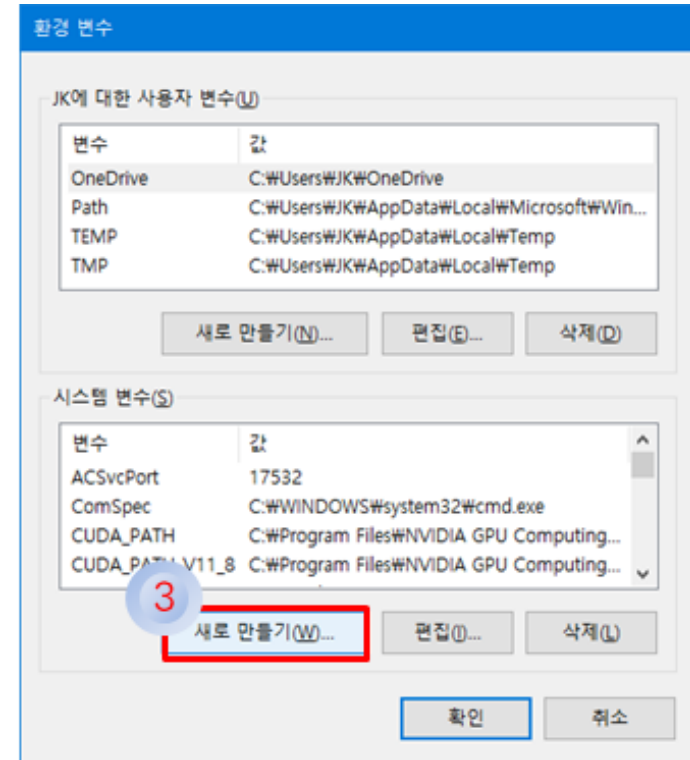
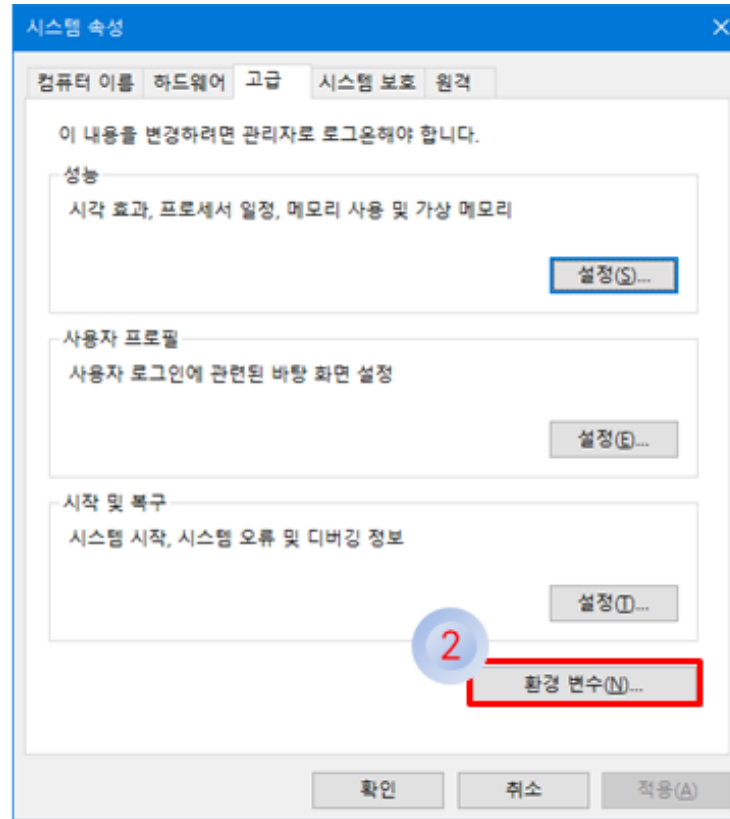
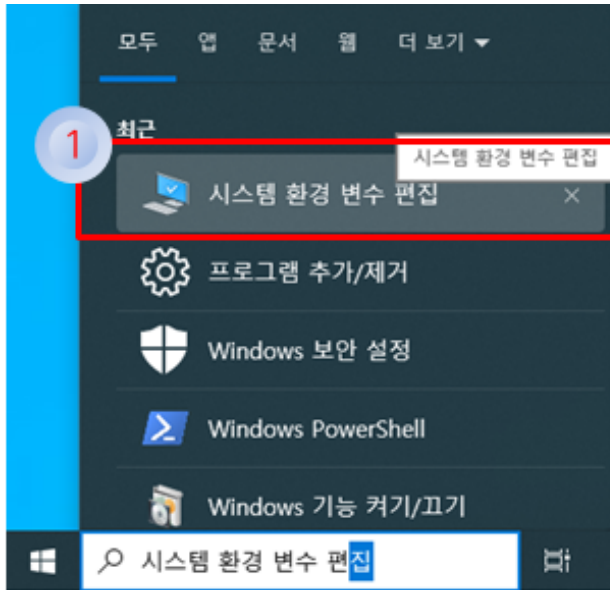


Set JAVA\_HOME variable 항목을 선택하면 JAVA\_HOME 환경변수가 자동으로 설정됩니다.

JDK를 설치하는 디렉토리가 반드시 'C:\dev\jdk-21'일 필요는 없습니다. JDK를 찾기 쉬운 디렉토리에 설치해 놓는다면 JDK가 있어야 하는 여러 상황에 유용하게 사용됩니다.

## 2.1. JDK 설치 – JAVA\_HOME 환경변수 설정

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

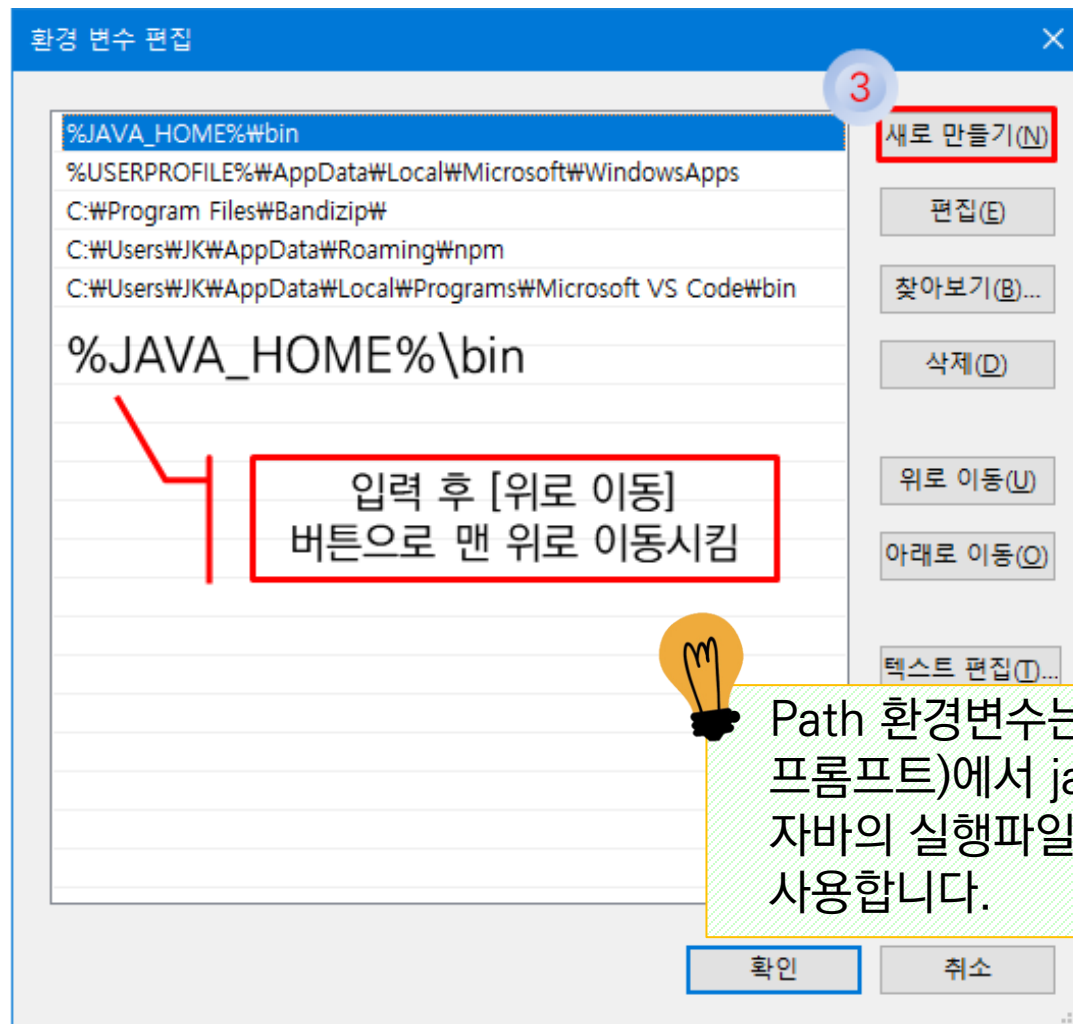
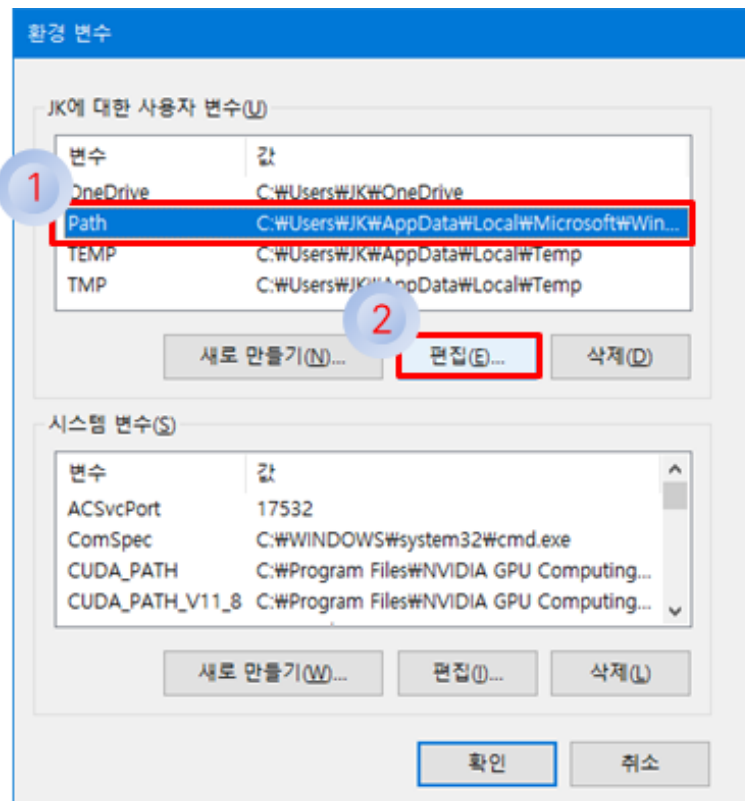


JAVA\_HOME 환경변수는 자바 JDK를 필요로 하는 다른 애플리케이션이 사용합니다.



## 2.1. JDK 설치 – Path 환경변수 설정

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션



## 2.1. JDK 설치 – 자바 버전 확인

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

```
C:\Users\JK>javac -version  
javac 21.0.1
```

```
C:\Users\JK>java -version  
openjdk version "21.0.1" 2023-10-17 LTS  
OpenJDK Runtime Environment Temurin-21.0.1+12 (build  
21.0.1+12-LTS)  
OpenJDK 64-Bit Server VM Temurin-21.0.1+12 (build 21.0.1+12-LTS,  
mixed mode, sharing)
```

## 2.2. 이클립스를 이용한 자바 개발 – STS4 설치하기

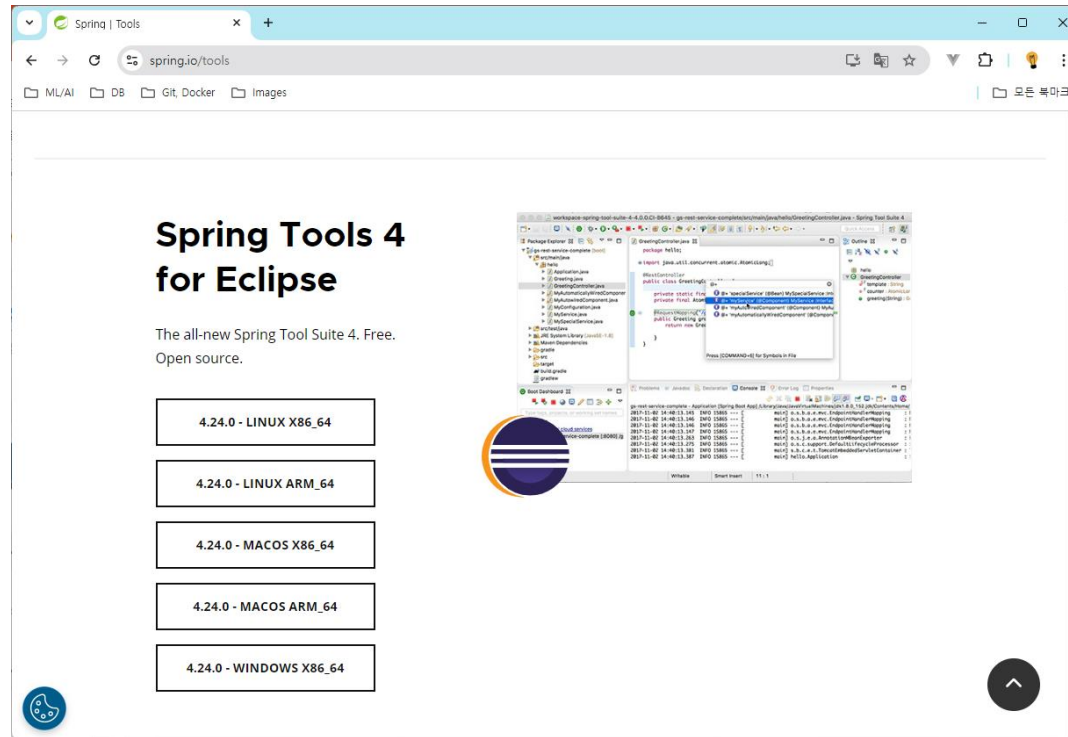
1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

### 이클립스를 이용한 스프링 프로젝트 개발환경 구성 방법

- www.egovframe.go.kr에서 제공하는 전자정부표준프레임워크 개발환경
- **spring.io에서 제공하는 Spring Tools 4**
- eclipse.org에서 제공하는 기본 이클립스에 Spring Tools 플러그인 설치

### Spring Tools 4

<https://spring.io/tools>



내려받은 파일은 확장자가 '.jar'파일일 경우 JDK가 설치되어 있고 Path 환경변수가 설정되어 있다면 더블클릭만 하면 '.jar' 파일이 있는 디렉토리에 압축이 풀립니다.

- 확장자가 .zip 이면 압축만 푸세요.

# 2.2. 이클립스를 이용한 자바 개발 – 스프링 스타터 프로젝트

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

✓ 스프링 프로젝트 만들기

- 이클립스 메뉴에서 File > New > Spring Starter Project 선택

New Spring Starter Project

Service URL

https://start.spring.io

Name

demo

☒ Use default location

Location

C:\dev\WeGovFrameDev-4.1.0-64bit\workspace\demo

Browse

Type

Maven

Packaging

War

Java Version

17

Language

Java

Group

com.example

Artifact

demo

Version

0.0.1-SNAPSHOT

Description

Demo project for Spring Boot

Package

com.example.demo

Working sets

☐ Add project to working sets

New...

Working sets

Select...

?

< Back

Next >

Finish

Cancel



스프링 부트 프로젝트를 만들 때 선택할 수 있는 항목과 값들입니다.

항목	기본값	설명
Service URL	http://start.spring.io	스프링 프로젝트의 서비스 URL입니다.
Name	demo	프로젝트의 이름으로 만들어집니다.
Type	Maven / Gradle	라이브러리 의존성 관리 도구를 지정합니다.
Packaging	Jar / War	패키징할 기본 형식을 지정합니다.
Java Version	17	자바의 버전을 지정합니다.
Language	Java / Kotlin / Groovy	사용할 언어를 지정합니다.
Group	com.example	그룹 이름을 지정합니다. 주로 도메인 이름까지 사용합니다.
Artifact	demo	아티팩트는 메이븐 빌드의 결과로 얻을 수 있는 일반적인 jar 나 war 또는 여타의 실행 파일의 이름으로 지정됩니다.
Version	0.0.1-SNAPSHOT	버전을 지정합니다.
Description	Demo project for Spring Boot	이 프로젝트의 설명을 기록합니다.
Package	com.example.demo	기본 패키지 이름을 지정합니다.

## 2.3. 부트 프로젝트 생성

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

### ✓ 스프링 프로젝트 만들기

- 이클립스 메뉴에서 File > New > Spring Starter Project 선택

Name:	myapp		
Type:	Maven	Packaging:	War
Java Version:	21	Language:	Java
Group:	com.example		
Artifact:	myapp		
Version:	1.0		
Description:	인공지능 서비스		
Package:	com.example.myapp		

### ✓ 부트 버전 및 프로젝트 의존성 설정

- Spring Boot Version: 3.3.2
- Developer Tools
  - Spring Boot DevTools
- Template Engines
  - Thymeleaf
- Web
  - Spring Web
  - Spring Reactive Web



이 프로젝트는 타임리프(Thymeleaf) 뷰 템플릿을 사용합니다.

- 프로젝트 생성 시 부트 버전과 의존성을 설정하세요.

## 2.4. 스프링 설정파일

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

✓ application.properties

`spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration`



이 프로젝트는 데이터베이스 연결을 사용하지 않습니다.

- src/main/resources/에 있는 스프링 설정 파일에 데이터소스 자동 설정을 제외하는 설정을 추가하세요.

## 2.5. 홈 컨트롤러와 프로젝트 실행 - 컨트롤러

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

### HomeController.java

```
1  package com.example.myapp.controller;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.GetMapping;
5
6  @Controller
7  public class HomeController {
8
9      @GetMapping("/")
10     public String home() {
11         return "index";
12     }
13 }
```



스프링에서 컨트롤러는 요청을 처리합니다. 이 컨트롤러는 루트 컨텍스트(/)에 요청 시 index.html 파일을 화면에 출력합니다.



## 2.5. 홈 컨트롤러와 프로젝트 실행 - 뷰

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

index.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Welcome</title>
6 </head>
7 <body>
8 <h1>Welcome Java</h1>
9 </body>
10 </html>
```

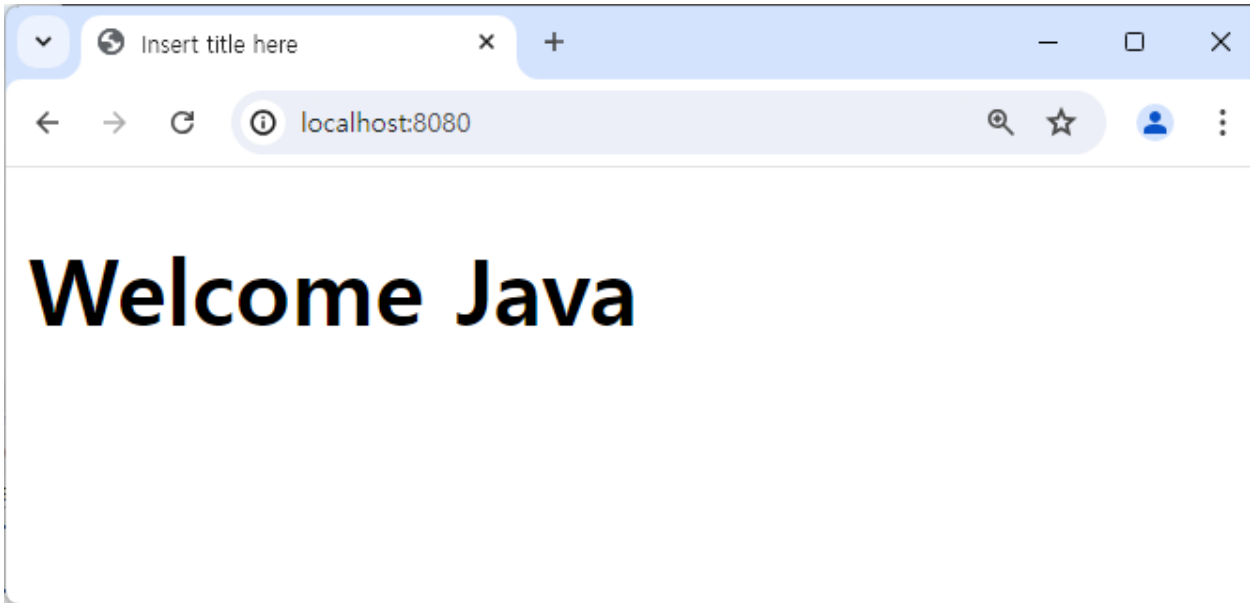


src/main/resources/  
아래의 templates/  
디렉토리에 index.html  
파일을 추가하세요.

## 2.5. 홈 컨트롤러와 프로젝트 실행 - 실행

1. 웹 애플리케이션 프로젝트 만들기 / 2절. Spring Boot를 이용한 자바 웹 애플리케이션

- [Run] > [Run AS] > [Spring Boot App]
- 브라우저를 실행한 후 localhost:8080으로 접속 후 테스트



'Spring Boot App' 메뉴가 없다면  
프로젝트에 MyAppApplication 파일을  
선택한 후 [Run] > [Run As] > [Java  
Application] 메뉴를 선택하면 스프링 부트  
프로젝트를 실행할 수 있습니다.

## 2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리

자바 스프링 부트 프로젝트와 파이썬 AI 프로젝트 연결하기

### ✓ 비동기 웹서비스 아키텍처와 개발 환경

- ▶ 아키텍처
- ▶ 개발환경

### ✓ 파이썬 FastAPI 프로젝트

- ▶ FastAPI 앱 구현 및 실행
- ▶ 이미지 객체 탐지 서비스 구현

### ✓ 자바 스프링 부트 프로젝트

- ▶ 스프링 부트 프로젝트와 스프링 설정파일
- ▶ WebClient 빈 설정
- ▶ 비동기 요청 컨트롤러 구현

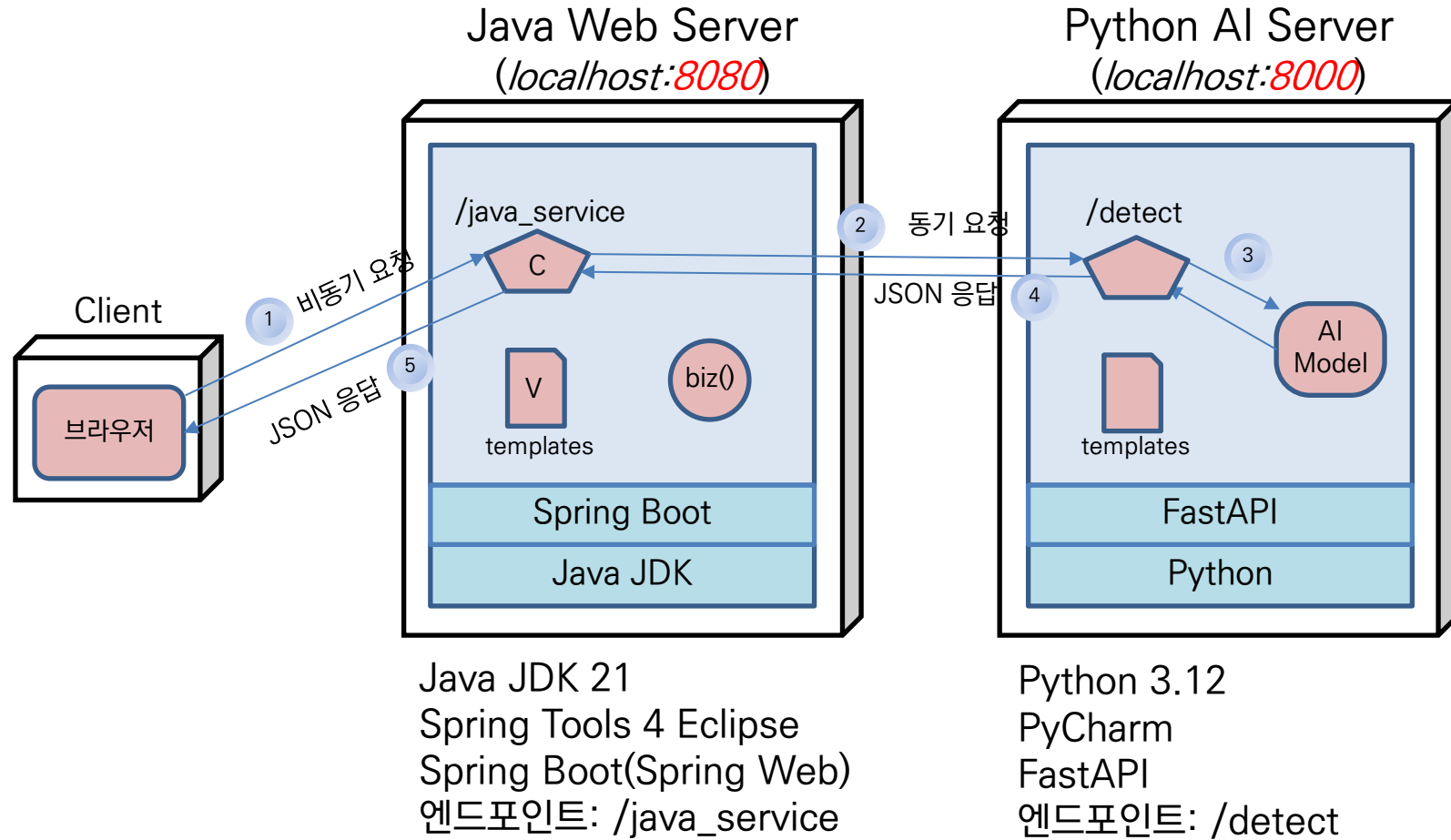
# 1절. 비동기 웹서비스 아키텍처와 개발 환경

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리



# 1.1. 시스템 아키텍처

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 1절. 비동기 웹서비스 아키텍처와 개발 환경



# 1.2. 개발 환경

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 1절. 비동기 웹서비스 아키텍처와 개발 환경

## 자바

도구 및 라이브러리	버전	다운로드
JDK	21 LTS	<a href="https://adoptium.net/">https://adoptium.net/</a>
Spring Tools 4 for Eclipse	4.24.0	<a href="https://spring.io/tools">https://spring.io/tools</a>
Spring Boot	3.3.2	스프링 부트 버전은 2.5 이상이어야 합니다.

## 파이썬

도구	버전
파이썬	3.12
PyCharm community edition	2024.2

## 파이썬 라이브러리

라이브러리	버전	주요 기능
fastapi	0.111.1	비동기 웹 프레임워크, 자동 OpenAPI 문서 생성
uvicorn	0.30.1	고성능 비동기 서버, ASGI 표준 지원
pydantic	2.7.1	데이터 검증 및 직렬화, 타입 힌팅, 설정 관리
pillow	10.3.0	이미지 열기, 저장, 변환, 다양한 이미지 처리 작업
numpy	1.24.4	수치 계산, 배열 및 행렬 연산, 다양한 수학 함수
requests	2.32.3	간단한 HTTP 요청 및 응답 처리
ultralytics	8.2.58	YOLOv8 객체 탐지 모델 제공
opencv-python	4.10.0	이미지 및 비디오 처리, 컴퓨터 비전 기능
python-multipart	0.0.9	멀티파트 폼 데이터를 파싱하기 위해 사용

## 2절. 파이썬 FastAPI 프로젝트

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리





## 2.1. 테스트를 위한 RestAPI 구현

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 2절. 파이썬 FastAPI 프로젝트

main.py


```
1  from fastapi import FastAPI
2
3  app = FastAPI()
4
5  @app.get("/")
6  async def read_root():
7      return {"message": "Hello FastAPI"}
8
9  if __name__ == "__main__":
10     import uvicorn
11     uvicorn.run(app, host="0.0.0.0", port=8000)
```



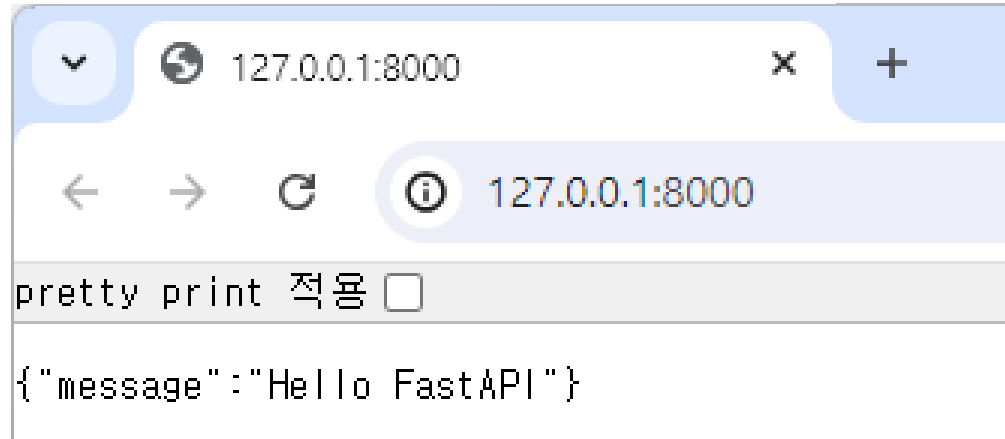
GET 방식 요청을 테스트하기 위해서 "Hello FastAPI"라는 메시지를 JSON 형식으로 반환 기본 엔드포인트를 추가합니다.

## 2.2. RestAPI 앱 실행

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 2절. 파이썬 FastAPI 프로젝트



```
uvicorn main:app --reload
```



--reload 옵션은 파일이  
수정되면 서버를 자동으로  
재시작합니다.

## 2.3. 이미지 객체 탐지 서비스 구현

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 2절. 파이썬 FastAPI 프로젝트

### ✓ 1) 라이브러리 및 모듈 импорт

main.py

```
1 from fastapi import FastAPI, UploadFile, File, Form
2 from fastapi.responses import JSONResponse
3 from pydantic import BaseModel
4 import io
5 import base64
6 from PIL import Image
7 import numpy as np
8 from ultralytics import YOLO
9 import cv2
10
```



POST 요청을 통해 이미지가 전송되면 인공지능 객체 탐지 모델을 이용해서 객체를 탐지하고 그 결과 이미지를 base64 인코딩된 문자열로 반환하는 서비스를 구현합니다.

## 2.3. 이미지 객체 탐지 서비스 구현

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 2절. 파이썬 FastAPI 프로젝트

### ✓ 2) FastAPI 애플리케이션 인스턴스 생성

main.py

```
11 app = FastAPI()
12
```

### ✓ 3) YOLOv8 모델 로드

main.py

```
13 model = YOLO('yolov8n.pt')
14
15
```



yolov8n.pt는 YOLO 8 버전의 nano 모델 가중치입니다.

### ✓ 4) 데이터 모델 정의

main.py

```
16 # 데이터 모델 정의
17 class DetectionResult(BaseModel):
18     message: str
19     image: str
20
21
```

## 2.3. 이미지 객체 탐지 서비스 구현

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 2절. 파이썬 FastAPI 프로젝트

### ✓ 5) 객체 탐지 함수

main.py

```
22 def detect_objects(image: Image.Image):
23     img = np.array(image) # 이미지를 numpy 배열로 변환
24     results = model(img) # 객체 탐지
25     class_names = model.names # 클래스이름 저장
26
27     # 결과를 바운딩 박스, 클래스이름, 정확도로 이미지에 표시
28     for result in results:
29         boxes = result.boxes.xyxy # 바운딩 박스
30         confidences = result.boxes.conf # 신뢰도
31         class_ids = result.boxes.cls # 클래스 이름
32         for box, confidence, class_id in zip(boxes, confidences, class_ids):
33             x1, y1, x2, y2 = map(int, box) # 좌표를 정수로 변환
34             label = class_names[int(class_id)] # 클래스 이름
35             cv2.rectangle(img, (x1,y1), (x2,y2), (255,0,0), 2)
36             cv2.putText(img, f'{label} {confidence:.2f}', (x1,y1),
37                         cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255,0,0), 2)
38
39     result_image = Image.fromarray(img) # 결과 이미지를 PIL로 변환
40     return result_image
41
```

### ✓ 6) 기본 엔드포인트

main.py

```
42 @app.get("/")
43 async def index():
44     return {"message": "Hello FastAPI"}
45
46
```

## 2.3. 이미지 객체 탐지 서비스 구현

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 2절. 파이썬 FastAPI 프로젝트

### 7) 객체 탐지 엔드포인트

main.py

```
47 @app.post("/detect", response_model=DetectionResult)
48 async def detect_service(message: str = Form(...), file: UploadFile = File(...)):
49     # 이미지를 읽어서 PIL 이미지로 변환
50     image = Image.open(io.BytesIO(await file.read()))
51
52     # 알파 채널 제거하고 RGB로 변환
53     if image.mode == 'RGBA':
54         image = image.convert('RGB')
55     elif image.mode != 'RGB':
56         image = image.convert('RGB')
57
58     # 객체 탐지 수행
59     result_image = detect_objects(image)
60
61     # 이미지 결과를 base64로 인코딩
62     buffered = io.BytesIO()
63     result_image.save(buffered, format="JPEG")
64     img_str = base64.b64encode(buffered.getvalue()).decode("utf-8")
65
66     return DetectionResult(message=message, image=img_str)
67
68
```

### 8) 애플리케이션 실행을 위한 정의

main.py

```
69 if __name__ == "__main__":
70     import uvicorn
71     uvicorn.run(app, host="0.0.0.0", port=8000)
72
```



decode('utf-8')을 포함한 이유는  
base64인코딩 후 인코딩된 데이터가  
바이트 객체를 반환하기 때문에 바이트  
객체를 문자열 형태로 사용하려면  
디코딩이 필요하기 때문입니다.

## 2.4. 전체 코드

### 2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 2절. 파이썬 FastAPI 프로젝트

main.py

```
1 from fastapi import FastAPI, UploadFile, File, Form
2 from fastapi.responses import JSONResponse
3 from pydantic import BaseModel
4 import io
5 import base64
6 from PIL import Image
7 import numpy as np
8 from ultralytics import YOLO
9 import cv2
10
11 app = FastAPI()
12
13 model = YOLO('yolov8n.pt') # YOLOv8 모델 로드
14
15
16 # 데이터 모델 정의
17 class DetectionResult(BaseModel):
18     message: str
19     image: str
20
21
22 def detect_objects(image: Image.Image):
23     img = np.array(image) # 이미지를 numpy 배열로 변환
24     results = model(img) # 객체 탐지
25     class_names = model.names # 클래스이름 저장
26
27     # 결과를 바운딩 박스, 클래스이름, 정확도로 이미지에 표시
28     for result in results:
29         boxes = result.boxes.xyxy # 바운딩 박스
30         confidences = result.boxes.conf # 신뢰도
31         class_ids = result.boxes.cls # 클래스
32         for box, confidence, class_id in zip(boxes, confidences, class_ids):
33             x1, y1, x2, y2 = map(int, box) # 좌표를 정수로 변환
```

```
34         label = class_names[int(class_id)] # 클래스 이름
35         cv2.rectangle(img, (x1,y1), (x2,y2), (255,0,0), 2)
36         cv2.putText(img, f'{label} {confidence:.2f}', (x1,y1),
37                     cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255,0,0), 2)
38
39     result_image = Image.fromarray(img) # 결과 이미지를 PIL로 변환
40     return result_image
41
42 @app.get("/")
43 async def index():
44     return {"message": "Hello FastAPI"}
45
46
47 @app.post("/detect", response_model=DetectionResult)
48 async def detect_service(message: str = Form(...), file: UploadFile = File(...)):
49     # 이미지를 읽어서 PIL 이미지로 변환
50     image = Image.open(io.BytesIO(await file.read()))
51
52     # 알파 채널 제거하고 RGB로 변환
53     if image.mode == 'RGBA':
54         image = image.convert('RGB')
55     elif image.mode != 'RGB':
56         image = image.convert('RGB')
57
58     # 객체 탐지 수행
59     result_image = detect_objects(image)
60
61     # 이미지 결과를 base64로 인코딩
62     buffered = io.BytesIO()
63     result_image.save(buffered, format="JPEG")
64     img_str = base64.b64encode(buffered.getvalue()).decode("utf-8")
65
66     return DetectionResult(message=message, image=img_str)
67
68
69 if __name__ == "__main__":
70     import uvicorn
71     uvicorn.run(app, host="0.0.0.0", port=8000)
```



전체 코드는 아래의 깃허브 주소에서 볼 수 있습니다.

- <https://github.com/hjk7902/java2ai>  
-> main.py



## 2.5. 서비스 실행 확인

## 2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 2절. 파이썬 FastAPI 프로젝트

## 1) FastAPI 앱 실행

```
uvicorn main:app --reload
```

## 2) 비동기 요청 테스트

```
1 import requests
2
3 url = "http://127.0.0.1:8000/detect"
4 message = "Test message"
5 file_path = "sample.jpg"
6
7 with open(file_path, "rb") as file:
8     response = requests.post(url, data={"message": message},
9                               files={"file": file})
10 print(response.json())
```



파이썬 명령으로 test.py 파일을 실행하면 POST 방식으로 비동기 요청을 테스트할 수 있습니다.

```
python test.py
```

[illegible]

## 3절. 자바 스프링 부트 프로젝트

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리



# 3.1. 부트 프로젝트 생성

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 3절. 자바 스프링 부트 프로젝트



## 스프링 부트 프로젝트 생성

[File] > [New] -> [Spring Starter Project]

Name:	myapp		
Type:	Maven	Packaging:	War
Java Version:	21	Language:	Java
Group:	com.example		
Artifact:	myapp		
Version:	1.0		
Description:	인공지능 서비스		
Package:	com.example.myapp		



## 스프링 부트 버전 및 의존성

Spring Boot Version: 3.3.2

Developer Tools

- Spring Boot DevTools

Template Engines

- Thymeleaf

Web

- Spring Web
- Spring Reactive Web

## 3.2. 스프링 설정파일

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 3절. 자바 스프링 부트 프로젝트

✓ application.properties

`spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration`



- 이 프로젝트는 데이터베이스 연결을 사용하지 않습니다.
- src/main/resources/ 아래에 있는 스프링 설정 파일에 데이터소스 자동 설정을 제외하는 설정을 추가하세요.

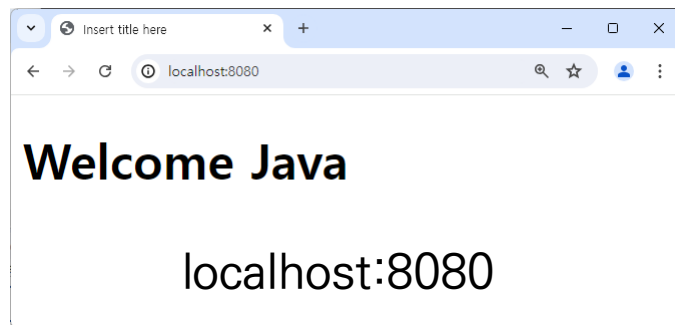
## 3.3. 홈 컨트롤러와 프로젝트 실행

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 3절. 자바 스프링 부트 프로젝트

### ✓ 컨트롤러

HomeController.java

```
1 package com.example.myapp.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5
6 @Controller
7 public class HomeController {
8
9     @GetMapping("/")
10    public String home() {
11        return "index";
12    }
13 }
```



### ✓ 뷰

index.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Welcome</title>
6 </head>
7 <body>
8 <h1>Welcome Java</h1>
9 </body>
10 </html>
```

### ✓ 실행

[Run] > [Run AS] > [Spring Boot App]

## 3.4. WebClient 빈 설정

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 3절. 자바 스프링 부트 프로젝트

WebClientConfig.java

```
1 package com.example.myapp.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.reactive.function.client.ExchangeStrategies;
6 import org.springframework.web.reactive.function.client.WebClient;
7
8 @Configuration
9 public class WebClientConfig {
10
11     @Bean
12     WebClient webClient() {
13         return WebClient.builder()
14             .exchangeStrategies(ExchangeStrategies.builder()
15                 .codecs(configurer -> configurer.defaultCodecs()
16                     .maxInMemorySize(-1)) // unlimited
17                 .build())
18             .baseUrl("http://localhost:8000")
19             .build();
20     }
21 }
```



WebClient는 메모리에 버퍼링할 수 있는 최대 크기를 무제한으로 설정하고, 기본 URL을 AI 서버의 주소(<http://localhost:5000>)로 설정하여 모든 요청에 해당 URL을 기본으로 사용하게 합니다.



WebClient 클래스는 Spring Reactive Web 라이브러리에 있습니다.

## 3.5. 요청 컨트롤러

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 3절. 자바 스프링 부트 프로젝트

```
15     @Autowired
16     private WebClient webClient;
17
18     @PostMapping("/java_service")
19     public String serviceRequest(@RequestParam("file")
20     MultipartFile file, @RequestParam("message") String message) {
21         MultipartBodyBuilder bodyBuilder = new MultipartBodyBuilder();
22         bodyBuilder.part("message", message);
23         bodyBuilder.part("file", file.getResource());
24
25         String result = webClient.post()
26             .uri("/detect")
27             .contentType(MediaType.MULTIPART_FORM_DATA)
28             .body(BodyInserters.fromMultipartData(bodyBuilder.build()))
29             .retrieve()
30             .bodyToMono(String.class)
31             .block();
32         return result;
33     }
```



Spring Boot에서 RestController를 사용하여 파일과 데이터를 멀티파트 폼 데이터 형식으로 전송하는 REST API 엔드포인트를 구현하세요.

- 클래스명: RestReqController.java
- @RestController 어노테이션을 사용
- 엔드포인트: `/java\_service`

# 3.6. 비동기 요청을 위한 HTML 페이지

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 3절. 자바 스프링 부트 프로젝트

index.html

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <meta charset="UTF-8">
5 <title>Welcome</title>
6 </head>
7 <body>
8 <h1>Welcome Java.</h1>
9 <form method="post" enctype="multipart/form-data" id="fileUploadForm">
10 데이터 : <input type="text" name="message" value="test hello"><p>
11 파일 : <input type="file" name="file"><p>
12 <input type="button" value=" 비동기 요청 ">
13 </form>
14 <div id="result">여기에 요청 결과가 출력되어야 합니다.</div>
15
16 <script type="text/javascript">
17 var button = document.querySelector("input[type=button]");
18
19 button.addEventListener("click", function() {
20     var form = document.getElementById("fileUploadForm");
21     var form_data = new FormData(form);
22     button.disabled = true;
23
24     var xhr = new XMLHttpRequest();
25
26     xhr.open("POST", "http://localhost:8080/java_service", true);
27
```

```
28     xhr.onload = function() {
29         if (xhr.status >= 200 && xhr.status < 300) {
30             var response = JSON.parse(xhr.responseText);
31             var resultDiv = document.getElementById("result");
32             resultDiv.innerHTML = response.message + "<br>";
33             var img_src = "data:image/png;base64," + response.image;
34             var img = document.createElement("img");
35             img.src = img_src;
36             resultDiv.appendChild(img);
37             button.disabled = false;
38         } else {
39             console.error("ERROR: ", xhr.statusText);
40             alert("fail" + xhr.statusText);
41             button.disabled = false;
42         }
43     };
44
45     xhr.onerror = function() {
46         console.error("ERROR: ", xhr.statusText);
47         alert("fail" + xhr.statusText);
48         button.disabled = false;
49     };
50
51     xhr.send(form_data);
52 });
53 </script>
54 </body>
55 </html>
```



이 코드는 비동기 요청을 실행하고 결과를 반환 받아 화면에 출력할 HTML입니다.

- 이 코드는 자바스크립트를 이용해서 비동기 요청을 구현했습니다.



## 3.7. 요청 테스트

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 3절. 자바 스프링 부트 프로젝트



파이썬 FastAPI 서버가 실행중이어야 합니다.



프로젝트를 실행했다면,  
http://localhost:8080으로 접속한 후 파일을  
선택하고 [비동기 요청] 버튼으로 테스트하세요.  
선택한 이미지에 탐지한 객체의 바운딩 박스  
정보가 표시되어 있어야 합니다.

## 3.8. 자주 발생하는 오류

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 3절. 자바 스프링 부트 프로젝트

- `WebClientResponseException$NotFound: 404 Not Found from POST http://localhost:8000/detect1` 오류는 자바에서 파이썬의 주소 또는 엔드포인트 URL 잘 못 입력한 경우입니다.
- `'422 UNPROCESSABLE_ENTITY'` 오류는 클라이언트가 보낸 요청이 서버에서 처리할 수 없는 형식이거나 필요한 데이터가 부족할 때 발생합니다. 이 오류는 주로 요청 데이터가 서버의 예상 형식과 일치하지 않을 때 발생합니다.
- `java.lang.IllegalArgumentException: 'part' must not be null` 오류는 주로 클라이언트에서 서버로 POST 요청을 보낼 때, 폼 데이터나 파일이 제대로 전송되지 않아서 발생합니다. 이 경우, 클라이언트에서 보내는 요청이 서버가 기대하는 형식과 맞지 않아서 발생할 수 있습니다.
- 파이썬의 실행 로그와 자바의 실행 로그에 아무것도 출력되지 않지만 브라우저에는 'Fail' 경고창일 뜰 경우는 크롬 브라우저의 개발자도구(F12)에서 오류를 확인해 보세요. 브라우저에서 자바 서버에 접속하는 주소를 127.0.0.1로 했지만, 비동기 요청하는 주소가 localhost로 요청했다면 아래와 같은 오류를 볼 수 있습니다. 127.0.0.1/:1 Access to XMLHttpRequest at 'http://localhost:8080/java\_service' from origin 'http://127.0.0.1:8080' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

## 3.8. 자주 발생하는 오류

2. 자바 스프링 부트와 파이썬 FastAPI로 배우는 객체 탐지: AI 모델 연동 및 이미지 처리 / 3절. 자바 스프링 부트 프로젝트

- 아래 그림처럼 객체가 너무 많이 탐지될 경우는 윈도우에서 CPU를 사용하는 경우 PyTorch와 관련된 버그입니다. 다음 명령으로 ultralytics 버전을 다운그레이드하고 사용하세요. 제가 테스트한 ultralytics 버전은 PyTorch2.4.0+cpu 에서 8.2.58 및 8.2.60버전입니다.
- `pip install ultralytics==8.2.58`



# 3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신

자바 스프링 부트 프로젝트와 파이썬 AI 프로젝트 연결하기



## 실시간 영상 스트리밍 서비스 개요

- ▶ 아키텍처와 개발환경
- ▶ MQTT



## AI 서버에서 실시간 객체 탐지 후 영상 스트리밍

- ▶ 객체 탐지 후 MQTT 브로커에 영상 스트리밍하기
- ▶ 객체 탐지 서비스 실행



## MQTT 브로커에서 이미지 수신하기

- ▶ 뷰-컨트롤러
- ▶ 자바 웹에서 이미지 수신하기

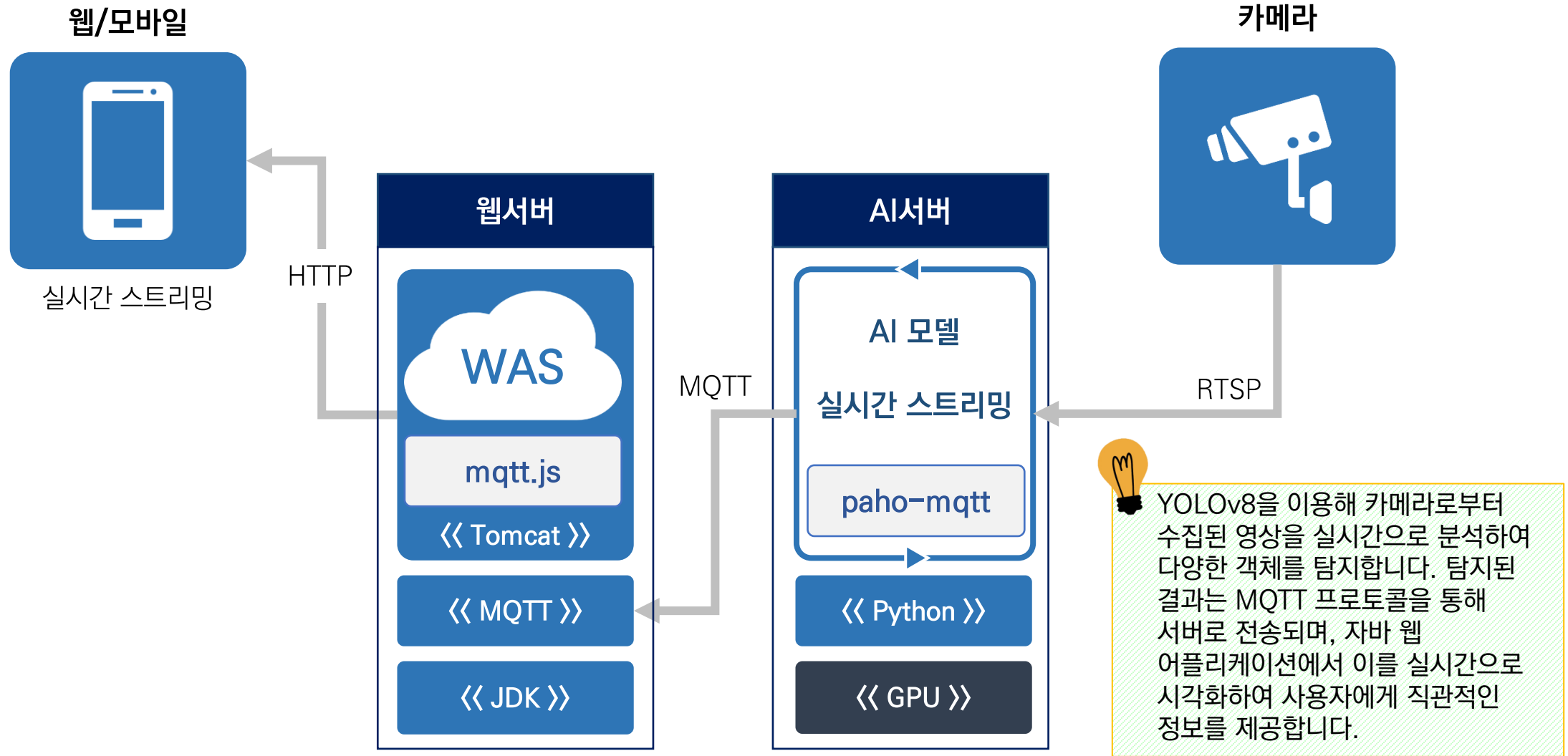
# 1절. 실시간 영상 스트리밍 서비스 개요

## 3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신



# 1.1. 시스템 아키텍처

## 3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 1절. 실시간 영상 스트리밍 서비스 개요



## 1.2. 개발 환경

### 3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 1절. 실시간 영상 스트리밍 서비스 개요



#### 자바

도구 및 라이브러리	버전	다운로드
JDK	21 LTS	<a href="https://adoptium.net/">https://adoptium.net/</a>
Spring Tools 4 for Eclipse	4.24.0	<a href="https://spring.io/tools">https://spring.io/tools</a>
Spring Boot	3.3.2	스프링 부트 버전은 2.5 이상이어야 함.
MQTT.js	5.10.0	<a href="https://github.com/mqttjs/MQTT.js">https://github.com/mqttjs/MQTT.js</a>

MQTT는 내려받지 않아도 됩니다.  
코드에서는 아래의 CDN 주소를 사용합니다.  
<https://unpkg.com/mqtt/dist/mqtt.min.js>  
<https://unpkg.com/mqtt@5.10.0/dist/mqtt.min.js>



#### 파이썬

도구 및 라이브러리	버전
파이썬	3.12
PyCharm community edition	2024.1.6
opencv-python	4.10.0
ultralytics	8.2.58
paho-mqtt	1.6.1

```
pip install opencv-python paho-mqtt ultralytics
```

# 1.3. MQTT

## 3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 1절. 실시간 영상 스트리밍 서비스 개요

### ✓ MQTT

- MQTT는 경량의 메시지 브로커 기반의 프로토콜
- 낮은 대역폭과 제한된 환경에서도 신뢰성 있게 통신할 수 있도록 설계되었음

### ✓ MQTT 서버

- <https://mosquitto.org/download/>
- mosquitto-2.0.18

mosquitto/mosquitto.conf

```
1 # MQTT 기본 리스너 설정
2 listener 1883
3 protocol mqtt
4
5 # WebSocket 리스너 설정
6 listener 9001
7 protocol websockets
8
9 allow_anonymous true
```

### ✓ MQTT 클라이언트(파이썬)

- MQTT 브로커에 연결하여 메시지를 게시(publish)하거나 구독(subscribe)하는 장치나 애플리케이션

```
pip install paho-mqtt
```

### ✓ MQTT 클라이언트(JS)

- Node.js 환경에서 MQTT 프로토콜을 구현한 자바스크립트 클라이언트 라이브러리
- <https://github.com/mqttjs/MQTT.js>

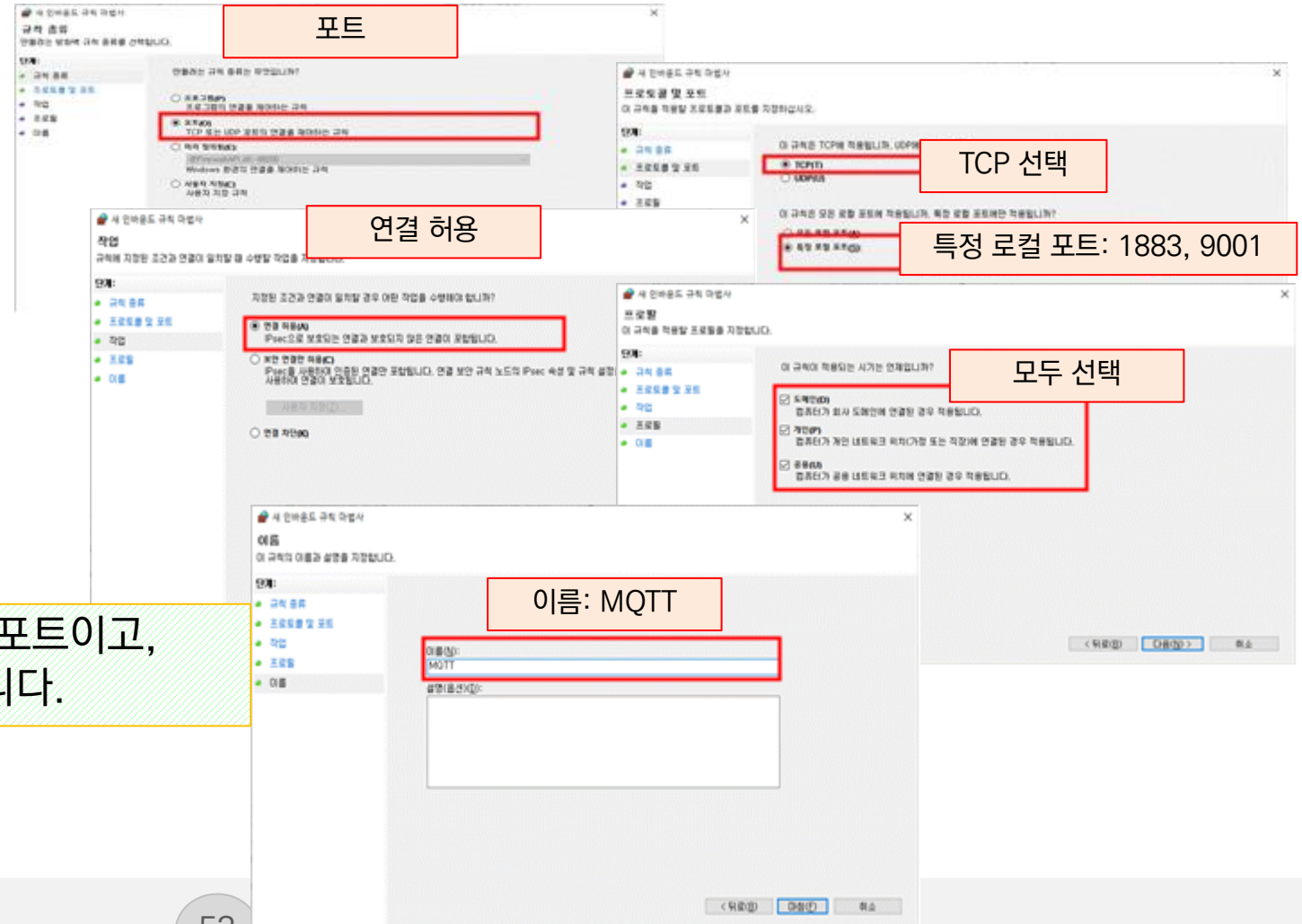
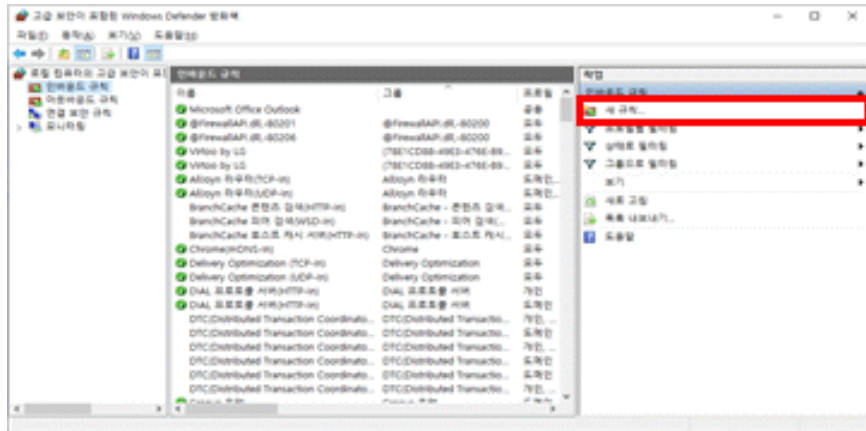


# 1.3. MQTT – 방화벽 설정

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 1절. 실시간 영상 스트리밍 서비스 개요

Windows Defender 방화벽 -> 고급 설정 -> 인바운드 규칙 -> [새 규칙...]을 클릭해서 인바운드 방화벽 규칙을 추가

‘규칙 종류 -> 포트 -> 작업 -> 프로필 -> 이름’ 순서로 설정



1883은 MQTT 기본 리스너의 포트이고, 9001은 웹소켓 리스너 포트입니다.

## 2절. AI 서버에서 실시간 객체 탐지 후 영상 스트리밍

### 3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신



## 2.1. 객체 탐지 후 MQTT 브로커에 영상 스트리밍하기

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 2절. AI 서버에서 실시간 객체 탐지 후 영상 스트리밍



### 1) 라이브러리 импорт

camera.py

```
1 import base64
2 import io
3 from PIL import Image
4 import cv2
5 import numpy as np
6 import paho.mqtt.client as mqtt
7 import json
8 from ultralytics import YOLO
9
```



### 2) YOLO 모델 로드

camera.py

```
10 # YOLO 모델 로드
11 model = YOLO('yolov8n.pt')
12
```

## 2.1. 객체 탐지 후 MQTT 브로커에 영상 스트리밍하기

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 2절. AI 서버에서 실시간 객체 탐지 후 영상 스트리밍

### ✓ 3) MQTT 클라이언트 설정 및 브로커 연결

camera.py

```
13  # MQTT 설정
14  broker = 'localhost'
15  port = 1883
16  topic = '/camera/objects'
17
18  # MQTT 클라이언트 설정
19  client = mqtt.Client()
20
21
22  def on_connect(client, userdata, flags, rc):
23      print(f"Connected with result code {rc}")
24
25
26  client.on_connect = on_connect
27  client.connect(broker, port)
28
29
```

### ✓ 4) 클래스 라벨별 색상 설정 함수 정의

camera.py

```
30  # 클래스 라벨별 색상 설정 함수
31  def get_colors(num_colors):
32      np.random.seed(0)
33      colors = [tuple(np.random.randint(0, 255, 3).tolist()) for _ in
34                range(num_colors)]
35      return colors
36
37  # 클래스 라벨 및 색상 설정
38  class_names = model.names
39  num_classes = len(class_names)
40  colors = get_colors(num_classes)
41
42
```

## 2.1. 객체 탐지 후 MQTT 브로커에 영상 스트리밍하기

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 2절. AI 서버에서 실시간 객체 탐지 후 영상 스트리밍



### 5) 객체 탐지 함수

camera.py

```
43 def detect_objects(image: np.array):
44     results = model(image, verbose=False) # 객체 탐지
45     class_names = model.names # 클래스 이름 저장
46
47     # 결과를 바운딩 박스와 정확도로 이미지에 표시
48     for result in results:
49         boxes = result.bboxes.xyxy # 바운딩 박스
50         confidences = result.bboxes.conf # 신뢰도
51         class_ids = result.bboxes.cls # 클래스
52         for box, confidence, class_id in zip(boxes, confidences, class_ids):
53             x1, y1, x2, y2 = map(int, box) # 좌표를 정수로 변환
54             label = class_names[int(class_id)] # 클래스 이름
55             cv2.rectangle(image, (x1,y1), (x2,y2), colors[int(class_id)], 2)
56             cv2.putText(image, f'{label} {confidence:.2f}', (x1,y1),
57                 cv2.FONT_HERSHEY_SIMPLEX, 0.9, colors[int(class_id)], 2)
58     return image
59
60
```

## 2.1. 객체 탐지 후 MQTT 브로커에 영상 스트리밍하기

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 2절. AI 서버에서 실시간 객체 탐지 후 영상 스트리밍

### ✓ 6) 카메라에서 프레임 캡처 및 객체 탐지 루프

camera.py

```
61  # 카메라에서 프레임 캡처
62  cap = cv2.VideoCapture(0)
63
64  while cap.isOpened():
65      ret, frame = cap.read()
66      if not ret:
67          break
68
69      result_image = detect_objects(frame)
70
71      # 이미지 결과를base64로 인코딩
72      _, buffer = cv2.imencode('.jpg', result_image)
73      jpg_as_text = base64.b64encode(buffer).decode('utf-8')
74
75      # 객체 탐지 이미지를 전송
76      payload = json.dumps({'image': jpg_as_text})
77      client.publish(topic, payload)
78
79      # 프레임을 화면에 표시
80      cv2.imshow('Frame', np.array(result_image))
81
82      # 'q' 키를 누르면 종료
83      if cv2.waitKey(1) & 0xFF == ord('q'):
84          break
85
86      # 리소스 해제
87      cap.release()
88      cv2.destroyAllWindows()
89      client.disconnect()
```

## 2.2. 전체 코드

### 3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 2절. AI 서버에서 실시간 객체 탐지 후 영상 스트리밍

camera.py

```
1 import base64
2 import io
3 from PIL import Image
4 import cv2
5 import numpy as np
6 import paho.mqtt.client as mqtt
7 import json
8 from ultralytics import YOLO
9
10 # YOLO 모델 로드
11 model = YOLO('yolov8n.pt')
12
13 # MQTT 설정
14 broker = 'localhost'
15 port = 1883
16 topic = '/camera/objects'
17
18 # MQTT 클라이언트 설정
19 client = mqtt.Client()
20
21
22 def on_connect(client, userdata, flags, rc):
23     print(f"Connected with result code {rc}")
24
25
26 client.on_connect = on_connect
27 client.connect(broker, port)
28
29
```

```
30 # 클래스 라벨별 색상 설정 함수
31 def get_colors(num_colors):
32     np.random.seed(0)
33     colors = [tuple(np.random.randint(0, 255, 3).tolist()) for _ in range(num_colors)]
34     return colors
35
36
37 # 클래스 라벨 및 색상 설정
38 class_names = model.names
39 num_classes = len(class_names)
40 colors = get_colors(num_classes)
41
42
43 def detect_objects(image: np.array):
44     results = model(image, verbose=False) # 객체 탐지
45     class_names = model.names # 클래스 이름 저장
46
47     # 결과를 바운딩 박스와 정확도로 이미지에 표시
48     for result in results:
49         boxes = result.boxes.xyxy # 바운딩 박스
50         confidences = result.boxes.conf # 신뢰도
51         class_ids = result.boxes.cls # 클래스
52         for box, confidence, class_id in zip(boxes, confidences, class_ids):
53             x1, y1, x2, y2 = map(int, box) # 좌표를 정수로 변환
54             label = class_names[int(class_id)] # 클래스 이름
55             cv2.rectangle(image, (x1,y1), (x2,y2), colors[int(class_id)], 2)
56             cv2.putText(image, f'{label} {confidence:.2f}', (x1,y1),
57                         cv2.FONT_HERSHEY_SIMPLEX, 0.9, colors[int(class_id)], 2)
58
59     return image
```



전체 코드는 아래의 깃허브 주소에서 볼 수 있습니다.

- <https://github.com/hjk7902/java2ai>  
-> camera.py

```
60
61 # 카메라에서 프레임 캡처
62 cap = cv2.VideoCapture(0)
63
64 while cap.isOpened():
65     ret, frame = cap.read()
66     if not ret:
67         break
68
69     result_image = detect_objects(frame)
70
71     # 이미지 결과를base64로 인코딩
72     _, buffer = cv2.imencode('.jpg', result_image)
73     jpg_as_text = base64.b64encode(buffer).decode('utf-8')
74
75     # 객체 탐지 이미지를 전송
76     payload = json.dumps({'image': jpg_as_text})
77     client.publish(topic, payload)
78
79     # 프레임을 화면에 표시
80     cv2.imshow('Frame', np.array(result_image))
81
82     # 'q' 키를 누르면 종료
83     if cv2.waitKey(1) & 0xFF == ord('q'):
84         break
85
86 # 리소스 해제
87 cap.release()
88 cv2.destroyAllWindows()
89 client.disconnect()
```

## 2.3. 객체 탐지 서비스 실행

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 2절. AI 서버에서 실시간 객체 탐지 후 영상 스트리밍



```
python camera.py
```



python 명령으로 프로그램을 실행하면  
카메라가 연결되고, 카메라의 프레임에 객체  
탐지 결과가 반영되어 있어야 합니다.



## 3절. MQTT 브로커에서 이미지 수신하기

### 3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신



# 3.1. 뷰 컨트롤러

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 3절. MQTT 브로커에서 이미지 수신하기

## WebConfig.java

```
1  package com.example.myapp.config;
2
3  import org.springframework.context.annotation.Configuration;
4  import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
5  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
6
7  @Configuration
8  public class WebConfig implements WebMvcConfigurer {
9
10     @Override
11     public void addViewControllers(ViewControllerRegistry registry) {
12         registry.addViewController("/ai").setViewName("ai");
13     }
14 }
```



이 설정대로라면 요청 URL이  
'http://localhost:8080/ai'이면 ai.html  
파일이 실행됩니다.

## 3.2. 자바 웹에서 이미지 수신하기

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 3절. MQTT 브로커에서 이미지 수신하기

### ✓ 1) HTML 기본 구조

ai.html

```
1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <title>MQTT Client Example</title>
5      <meta charset="utf-8">
6      <meta name="viewport" content="height=device-height">
7      <script src="https://unpkg.com/mqtt/dist/mqtt.min.js"></script>
8      <style>
9          div {
10              width: 100%;
11              height: 100%;
12          }
13          img#cameraView {
14              max-width: 100%;
15              max-height: 100%;
16              bottom: 0;
17              left: 0;
18              margin: auto;
19              overflow: auto;
20              position: fixed;
21              right: 0;
22              top: 0;
23          }
24      </style>
25 </head>
26 <body>
27     <h1>MQTT Client Example</h1>
28     <div align="center">
29         <img id="cameraView" width="100%" height="100%"/>
30     </div>
... 생략 ...
72 </body>
73 </html>
```

## 3.2. 자바 웹에서 이미지 수신하기

### 3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 3절. MQTT 브로커에서 이미지 수신하기



#### 2) JavaScript MQTT 클라이언트

ai.html

```
31 <script type="text/javascript">
32   // WebSocket을 통한 MQTT 연결
33   const broker = 'ws://localhost:9001';
34   const topic = '/camera/objects';
35
36   // MQTT 클라이언트 생성 및 연결
37   const client = mqtt.connect(broker);
38
39   client.on('connect', () => {
40     console.log('Connected to broker');
41     client.subscribe(topic, (err) => {
42       if (!err) {
43         console.log(`Subscribed to topic: ${topic}`);
44       }
45     });
46   });
47
```

```
48   // 메시지 수신 시 처리
49   client.on('message', (topic, message) => {
50     try {
51       // 메시지의 payload를 JSON으로 파싱
52       const payload = JSON.parse(message.toString());
53
54       // base64 이미지 추출
55       const base64Image = payload.image;
56
57       // img 태그의 src 속성에 base64 이미지 설정
58       document.getElementById("cameraView").src =
59         `data:image/jpg;base64,${base64Image}`; # `는 역따옴표입니다.
60     } catch (e) {
61       console.error('Failed to parse message:', e);
62     }
63   });
64
65   client.on('error', (error) => {
66     console.error('Connection error:', error);
67   });
68
69   client.on('close', () => {
70     console.log('Disconnected from broker');
71   });
72 </script>
```

## 3.3. 전체 코드

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 3절. MQTT 브로커에서 이미지 수신하기

ai.html

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4   <title>MQTT Client Example</title>
5   <meta charset="utf-8">
6   <meta name="viewport" content="height=device-height">
7   <script src="https://unpkg.com/mqtt/dist/mqtt.min.js"></script>
8   <style>
9     div {
10       width: 100%;
11       height: 100%;
12     }
13     img#cameraView {
14       max-width: 100%;
15       max-height: 100%;
16       bottom: 0;
17       left: 0;
18       margin: auto;
19       overflow: auto;
20       position: fixed;
21       right: 0;
22       top: 0;
23     }
24   </style>
25 </head>
```

```
25 </head>
26 <body>
27   <h1>MQTT Client Example</h1>
28   <div align="center">
29     <img id="cameraView" width="100%" height="100%">
30   </div>
31   <script type="text/javascript">
32     // WebSocket을 이용한 MQTT 연결
33     const broker = 'ws://localhost:9001';
34     const topic = '/camera/objects';
35
36     // MQTT 클라이언트 생성 및 연결
37     const client = mqtt.connect(broker);
38
39     client.on('connect', () => {
40       console.log('Connected to broker');
41       client.subscribe(topic, (err) => {
42         if (!err) {
43           console.log(`Subscribed to topic: ${topic}`);
44         }
45       });
46     });
47
```

```
48   // 메시지 수신 시 처리
49   client.on('message', (topic, message) => {
50     try {
51       // 메시지의 payload를 JSON으로 파싱
52       const payload = JSON.parse(message.toString());
53
54       // base64 이미지 추출
55       const base64Image = payload.image;
56
57       // img 태그의 src 속성에 base64 이미지 설정
58       document.getElementById("cameraView").src =
59         `data:image/jpeg;base64,${base64Image}`; // `는 역따옴표입니다.
60     } catch (e) {
61       console.error('Failed to parse message:', e);
62     }
63
64     client.on('error', (error) => {
65       console.error('Connection error:', error);
66     });
67
68     client.on('close', () => {
69       console.log('Disconnected from broker');
70     });
71   </script>
72 </body>
73 </html>
```

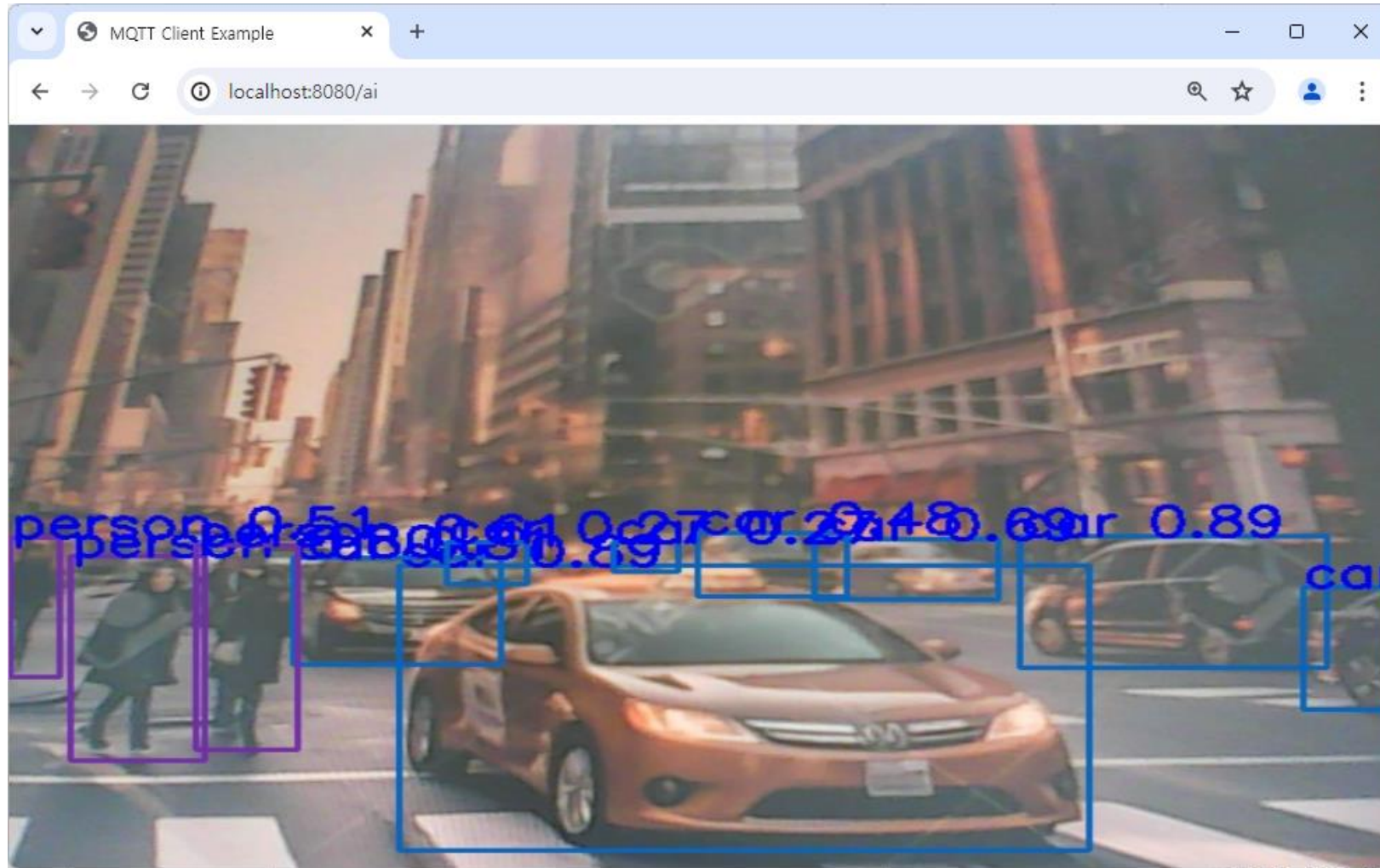


전체 코드는 아래의 깃허브 주소에서 볼 수 있습니다.

- <https://github.com/hjk7902/java2ai> -> ai.html

## 3.4. 실행

3. MQTT를 이용한 실시간 객체 탐지 영상 전송과 수신 / 3절. MQTT 브로커에서 이미지 수신하기



스프링 부트 프로젝트를 실행하고 카메라의 프레임이 브라우저 화면에도 동일하게 표시되는지 확인하세요.



파이썬 애플리케이션이 실행되고 있어야 합니다. 자바 프로젝트와 파이썬 프로젝트의 실행 순서는 무관합니다.