AI와 열화상 데이터 분석을 활용한 IoT 기반 서버 쿨링 시스템

팀원 : 홍수민, 길기훈, 오민석, 지원근
담당 교수 : 이상금 교수님
팀명 : 5인분 같은 4인분

목차

프로젝트 개요

프로젝트명

AI와 열화상 데이터 분석을 활용한 IoT 기반 서버 쿨링 시스템

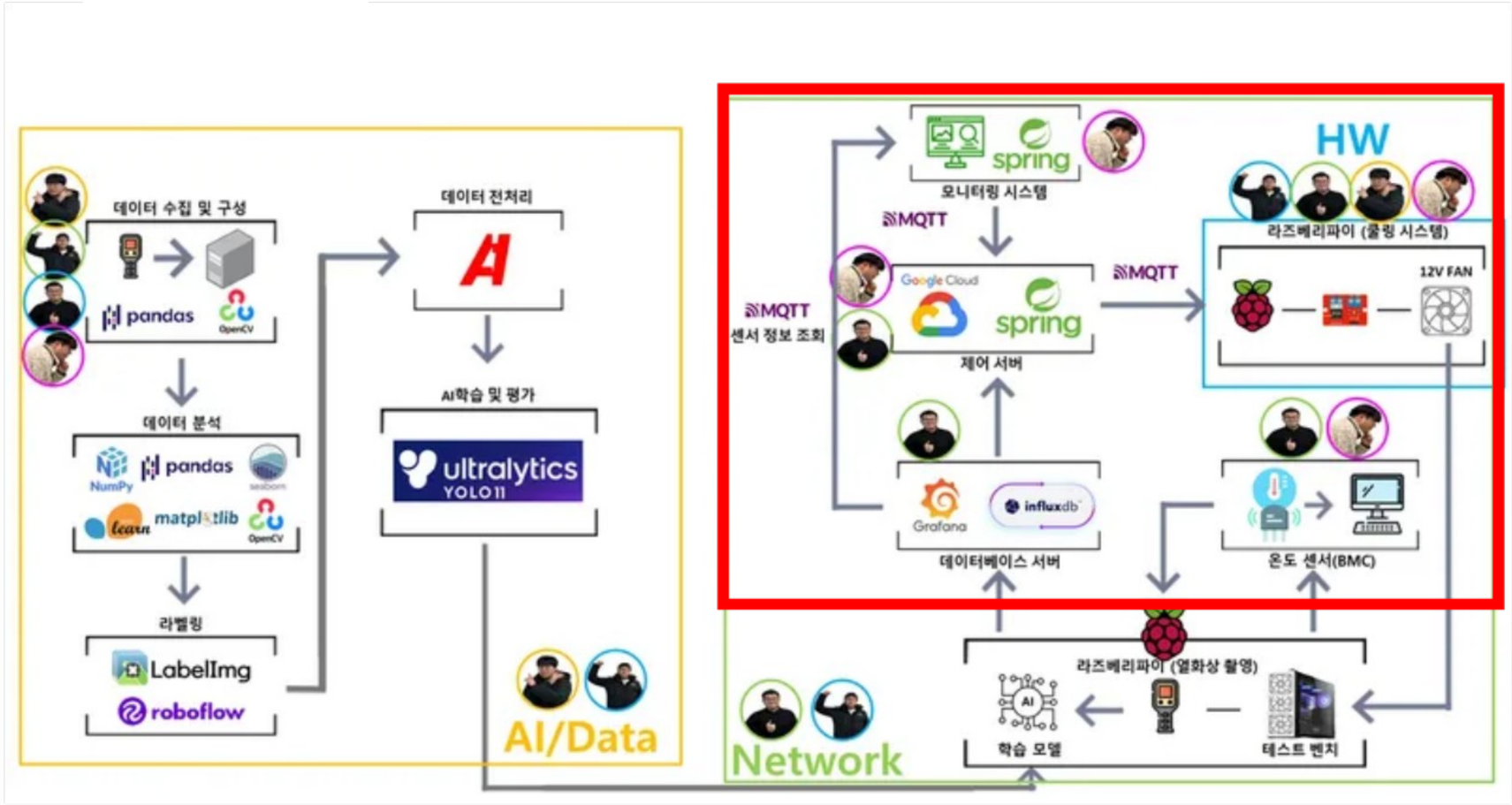
프로젝트 목표

서버의 온도를 효율적으로 제어하여 과열로 인한 장애를 사전에 방지하는 것

프로젝트 배경

고성능 하드웨어의 발열 증가로 인해 GPU 등의 주변에서 고온 현상발생

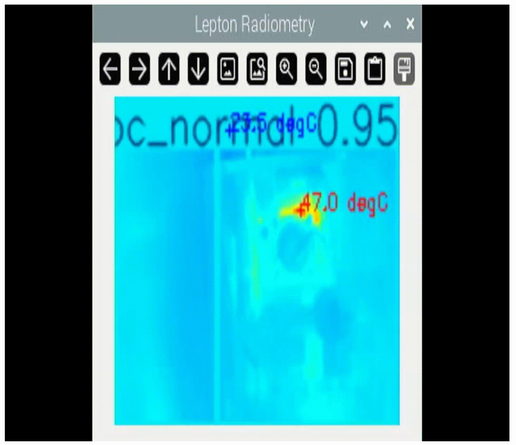
지난 단계 요약



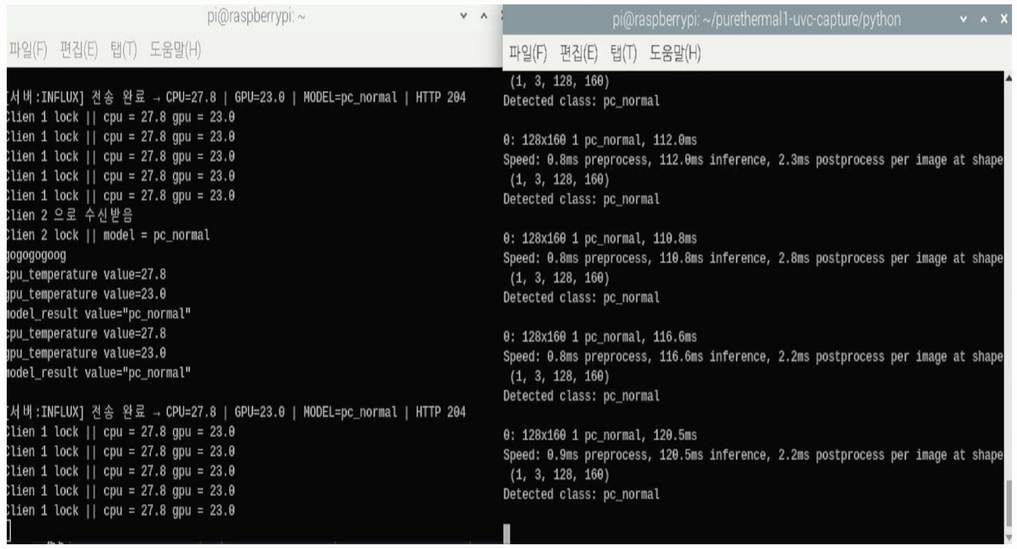
프로젝트 구성 및 목표

1. 라즈베리파이와 열화상 카메라 연동
2. 열화상 이미지 데이터 수집
3. 데이터 분석 및 YOLO 모델 학습
4. DB 구축
5. 서버 구축
6. 하드웨어 구성
7. 모니터링 시스템 구현

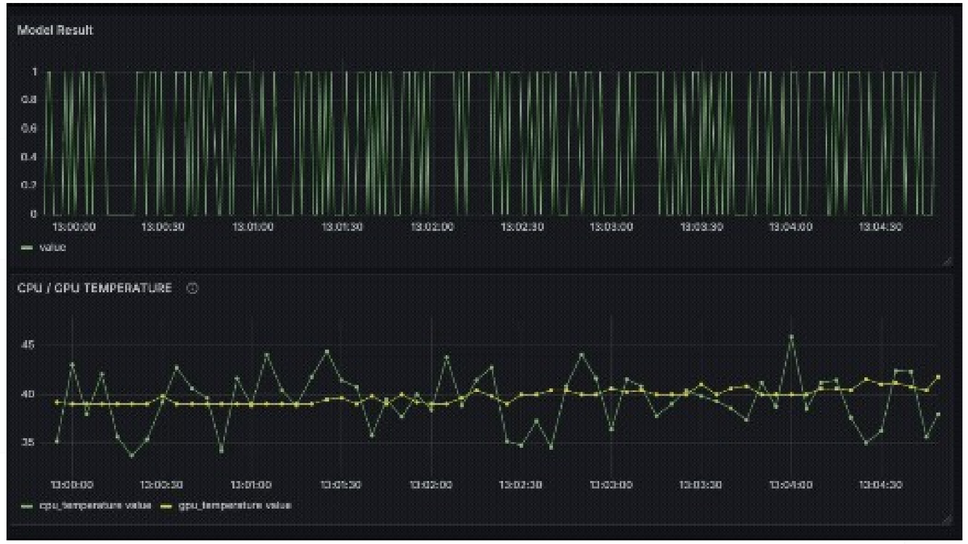
지난 단계 요약



<YOLO11n 모델 판별>



<라즈베리파이 전송(측정되는 컴퓨터 온도값) 및 DB 전송>



<Grafana 구동>

진행 방향

서버 구성

- GCP를 이용하여 서버 구성
- DB 서버
- 제어 서버

하드웨어 구성

- 맞춤형 케이스 제작
- 팬 제어 회로
- 열화상 카메라 부착

모니터링 시스템

- Spring 웹 서버 구축
- Grafana 구축

진행 상황(모니터링)

```java
package com.example.demo.model;

import java.time.LocalDateTime;

/**
 * 사용자 모델 클래스
 * - 사용자 정보를 표현하는 도메인 모델입니다.
 */
public class User {
    private Long id;
    private String username;
    private String password;
    private String name;
    private String email;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    // 기본 생성자
    public User() {
    }

    // 모든 필드를 포함한 생성자
    public User(Long id, String username, String password, String name, String email,
                LocalDateTime createdAt, LocalDateTime updatedAt) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.name = name;
        this.email = email;
        this.createdAt = createdAt;
        this.updatedAt = updatedAt;
    }

}
```

<코드 일부분>

- 아키텍처
  - 도메인 중심의 계층형 구조(Controller - Service - Repository)를 채택
- 도메인
  - 사용자 계정 정보를 관리하는 User 모델을 중심으로 개발 (사용자명, 비밀번호, 이메일 포함)
- 데이터베이스
  - 로그인 기능이 구현 향후 MySQL로 전환 예정. User 모델은 JPA 엔티티로 확장
- 입력검증 및 보안
  - DTO와 Bean Validation을 사용하여 Controller에서 입력 데이터를 검증
  - 비밀번호 해시(BCrypt)와 JWT 또는 세션 기반

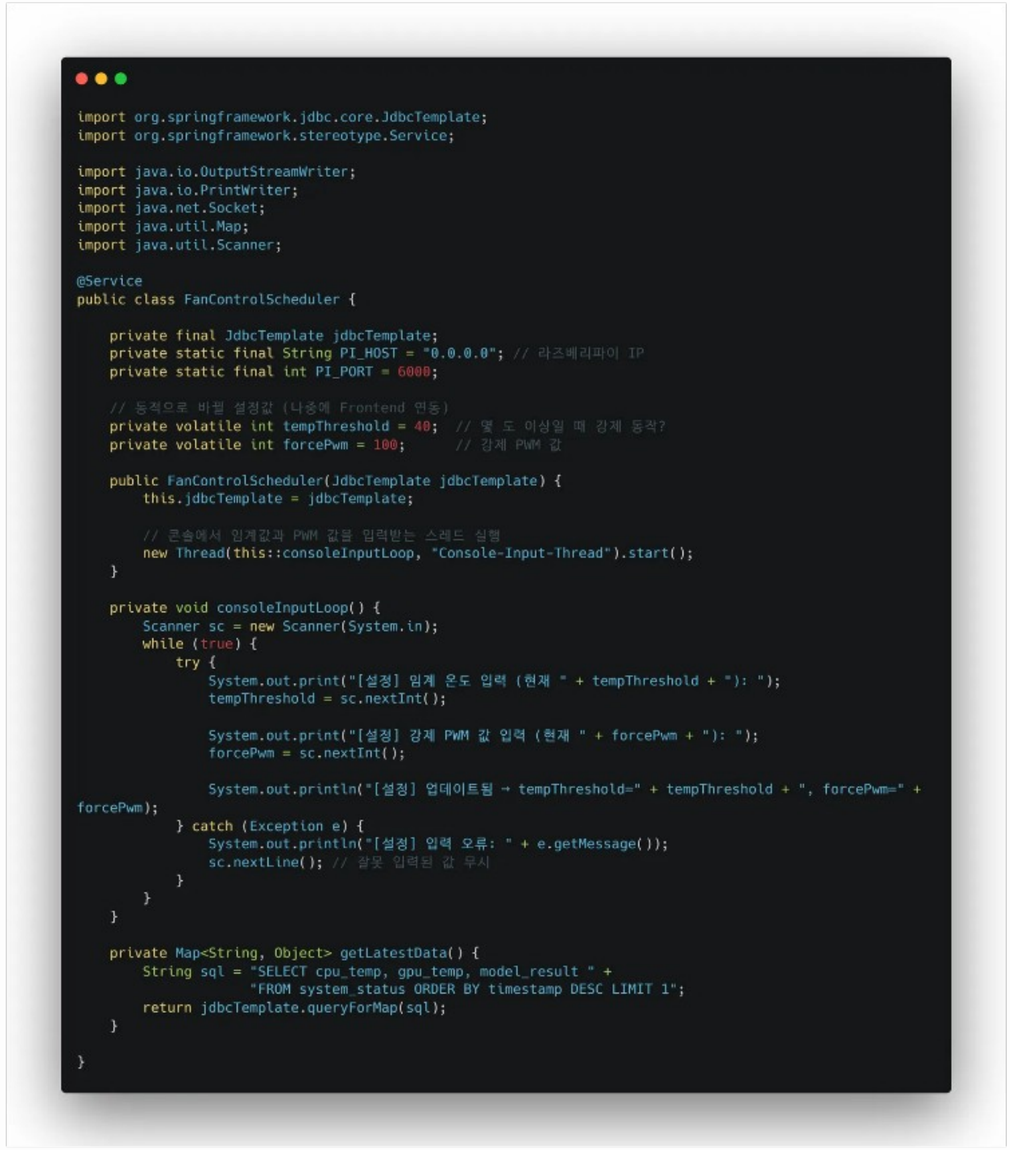
진행 상황(모니터링)

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api")
public class DashboardController {

    private final TemperatureService temperatureService;

    @Autowired
    public DashboardController(TemperatureService temperatureService) {
        this.temperatureService = temperatureService;
    }

    /**
     * (Task 4) 관리자 대시보드용 통합 API
     * 대시보드 초기 로딩 시 현재 시스템 상태를 가져옵니다.
     */
    @GetMapping("/dashboard/status")
    public ResponseEntity<SystemStatusDto> getDashboardStatus() {
        return ResponseEntity.ok(temperatureService.getCurrentStatus());
    }

    /**
     * (외부 연동용) 열화상 카메라 시스템이 온도를 전송할 엔드포인트
     * @param temperatureData 온도 데이터 (예: {"temperature": 75.5})
     */
    @PostMapping("/temperature/ingest")
    public ResponseEntity<Void> ingestTemperature(@RequestBody TemperatureData temperatureData)
{
        temperatureService.processNewTemperature(temperatureData.getTemperature());
        return ResponseEntity.ok().build();
    }

    // POST 요청 본문을 받기 위한 간단한 클래스
    static class TemperatureData {
        private double temperature;
        public double getTemperature() { return temperature; }
        public void setTemperature(double temperature) { this.temperature = temperature; }
    }
}
```

<코드 일부분>

- RESTful API를 통해 현재 온도/쿨러 상태 동기적 제공
- 모든 실시간 정보는 WebSocket 기반 STOMP 프로토콜을 사용하여 비동기적 푸시
- 백엔드 서비스 계층에서는 새로운 온도 데이터가 수신될 때마다 정의된 임계치를 감시하는 로직이 동작
- 임계치 초과 시 자동으로 쿨러를 제어하고 상태 변화와 경고 메시지를 포함한 통합 데이터를 모니터링 시스템에 전송

진행 상황(임베디드)



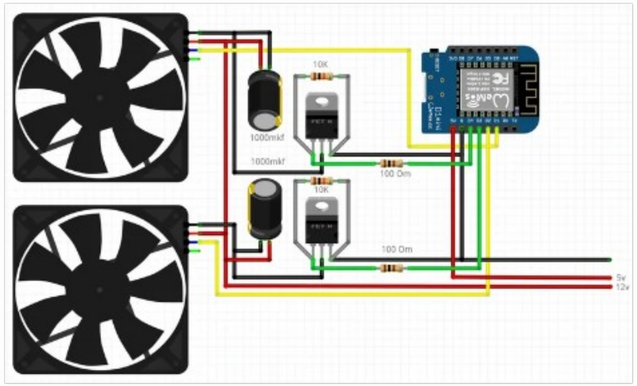
```java
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Service;

import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Map;
import java.util.Scanner;

@Service
public class FanControlScheduler {

    private final JdbcTemplate jdbcTemplate;
    private static final String PI_HOST = "0.0.0.0"; // 라즈베리파이 IP
    private static final int PI_PORT = 6000;

    // 동적으로 바뀔 설정값 (나중에 Frontend 연동)
    private volatile int tempThreshold = 40;  // 몇 도 이상일 때 강제 동작?
    private volatile int forcePwm = 100;      // 강제 PWM 값

    public FanControlScheduler(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;

        // 콘솔에서 임계값과 PWM 값을 입력받는 스레드 실행
        new Thread(this::consoleInputLoop, "Console-Input-Thread").start();
    }

    private void consoleInputLoop() {
        Scanner sc = new Scanner(System.in);
        while (true) {
            try {
                System.out.print("[설정] 임계 온도 입력 (현재 " + tempThreshold + "): ");
                tempThreshold = sc.nextInt();

                System.out.print("[설정] 강제 PWM 값 입력 (현재 " + forcePwm + "): ");
                forcePwm = sc.nextInt();

                System.out.println("[설정] 업데이트됨 → tempThreshold=" + tempThreshold + ", forcePwm=" +
forcePwm);
            } catch (Exception e) {
                System.out.println("[설정] 입력 오류: " + e.getMessage());
                sc.nextLine(); // 잘못 입력된 값 무시
            }
        }
    }

    private Map<String, Object> getLatestData() {
        String sql = "SELECT cpu_temp, gpu_temp, model_result " +
                     "FROM system_status ORDER BY timestamp DESC LIMIT 1";
        return jdbcTemplate.queryForMap(sql);
    }

}
```

<코드 일부분>

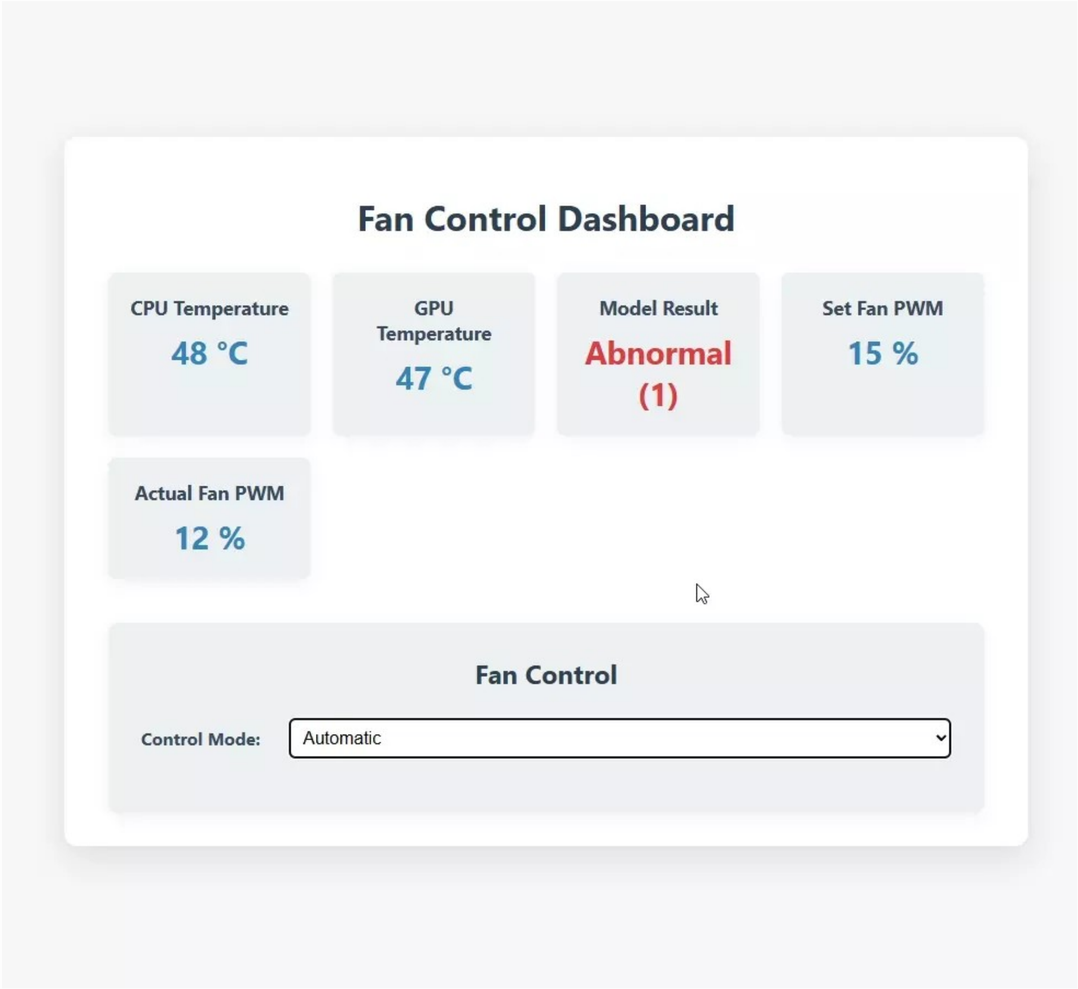
- Fan 제어용 라즈베리파이와 제어 서버간 네트워크를 설계·구축
- 두 장치는 VPN 망을 통해 연결되며, 제어 서버에서 전달되는 PWM 제어 신호를 라즈베리파이가 수신하여 해당 신호에 따라 팬 속도를 조절
- 팬이 실제로 제어 신호에 맞게 동작하고 있는지 검증하기 위해, 라즈베리파이는 PWM 동작 피드백 데이터를 수집하여 InfluxDB에 저장

진행 예정

<아크릴 케이스 (예시)>



<12V FAN 구동 MOSFET 회로 >



<모니터링 대시보드 (WEB UI)>