

움직임 감지 웹 서비스

실시간 움직임 감지, 웹으로 만나다

20192494 김범준

20202446 김재영

20215287 이우민

20222534 신서림

목차

발표 순서

01 프로젝트 개요

05 개발 과정

02 프로젝트 목표 및 기대 효과

06 문제 해결

03 사용 기술 및 주요 부품

07 결과 및 시연

04 시스템 아키텍처

08 향후 개선 방향

09 Q & A

HC-SR501 & ESP-32를 활용한 움직임 감지 웹 서비스

실시간 움직임 감지, 웹으로 만나다

프로젝트 동기

일상 속 보안과 자동화에 대한 관심에서 출발하여 저비용 센서와 아두이노를 이용해 물리적 이벤트를 웹사이트와 연결하는 경험을 해보고 싶었다.

프로젝트 개요

ESP32에서 감지된 움직임 데이터를 서버로 전송하고, 웹 인터페이스를 통해 실시간으로 모니터링할 수 있는 시스템을 개발

프로젝트 목표 & 기대 효과

◎ 구체적인 목표

- ✓ **하드웨어 제어:** HC-SR501 센서 값 수신 및 ESP-32를 통한 Wi-Fi 통신 구현
- ✓ **안정적인 서버 구축:** Node.js와 Express.js 기반의 데이터 수신 및 처리를 위한 API 서버 개발
- ✓ **실시간 통신 구현:** HTTPP 통신을 활용하여 서버와 클라이언트 간의 데이터 전송
- ✓ **직관적 웹 인터페이스:** 감지되면 웹 UI에 변화를 주어 움직임 확인

⚡ 기대 효과

- ⚡ **보안성 강화:** 특정 구역의 무단 침입을 실시간으로 감지하고 즉각적인 알림 제공
- ⚡ **높은 확장성:** 온습도, 조도 등 다양한 센서를 추가하여 스마트 홈 시스템으로 발전시킬 수 있는 기반 구축
- ⚡ **기술 역량 향상:** IoT 시스템 전반에 대한 통합적인 개발 능력 확보

사용 기술 및 주요 부품

⚙️ Hardware



HC-SR501 (PIR 센서)

인체의 적외선을 감지하여 움직임을 포착하는 핵심 센서



ESP-32

Wi-Fi/Bluetooth가 내장된 고성능 마이크로컨트롤러



점퍼 케이블

센서, 보드 등 각 부품 간의 회로 연결



EVA 폼보드

프로토타입의 외관을 보호하고 고정하기 위한 케이스 제작

💻 Software & Platform



Node.js & Express.js

ESP-32로부터 데이터를 수신하는 백엔드 API 서버 구축



MongoDB

감지된 움직임 데이터를 저장하는 NoSQL 데이터베이스



HTML & Tailwind CSS

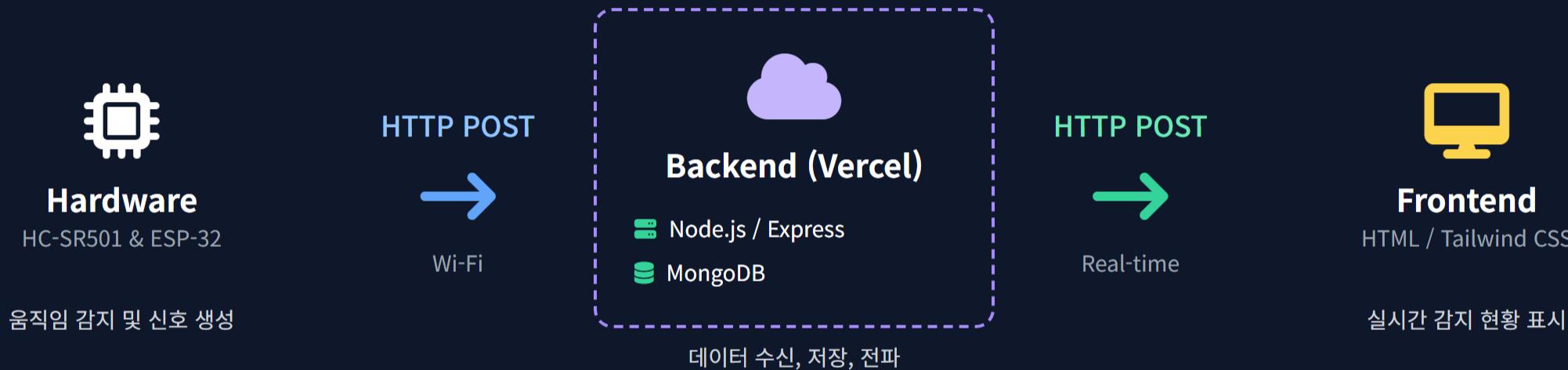
실시간 감지 현황을 보여주는 프론트엔드 웹 페이지 개발



Vercel

개발된 웹 서비스를 배포하고 호스팅하는 클라우드 플랫폼

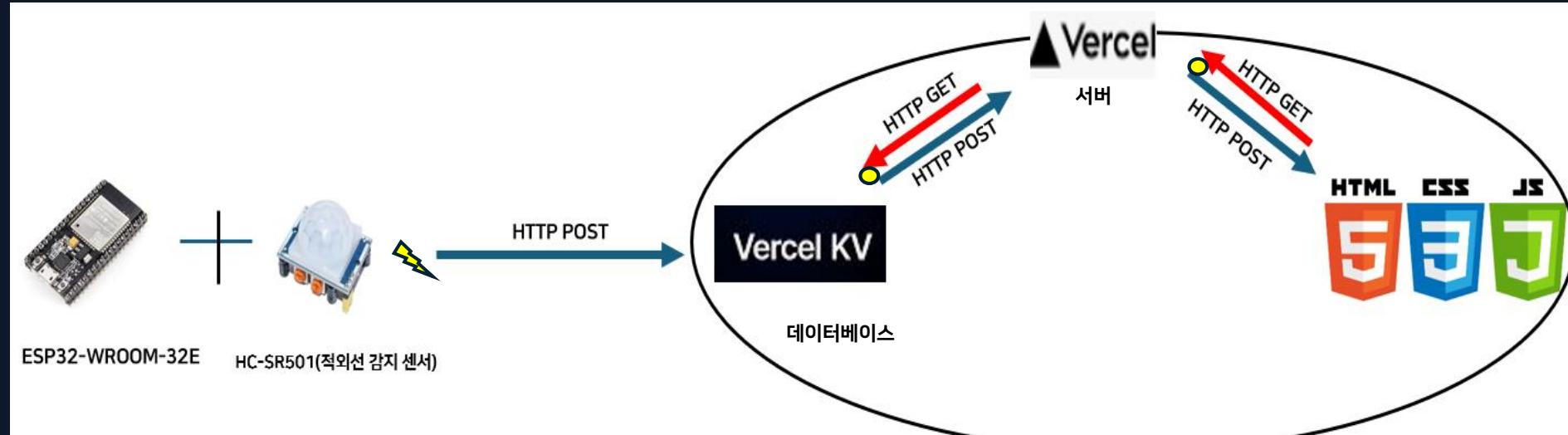
시스템 아키텍처 & 데이터 흐름



데이터 흐름 요약

1. **감지**: HC-SR501이 움직임을 감지하여 ESP-32에 신호 전달
2. **전송**: ESP-32가 Wi-Fi를 통해 Vercel 서버로 HTTP POST 요청 전송
3. **처리 및 저장**: Node.js 서버가 요청을 받아 MongoDB에 로그 저장
4. **실시간 전파**: 서버는 연결된 클라이언트에게 HTTP로 송수신

데이터 흐름 시작화

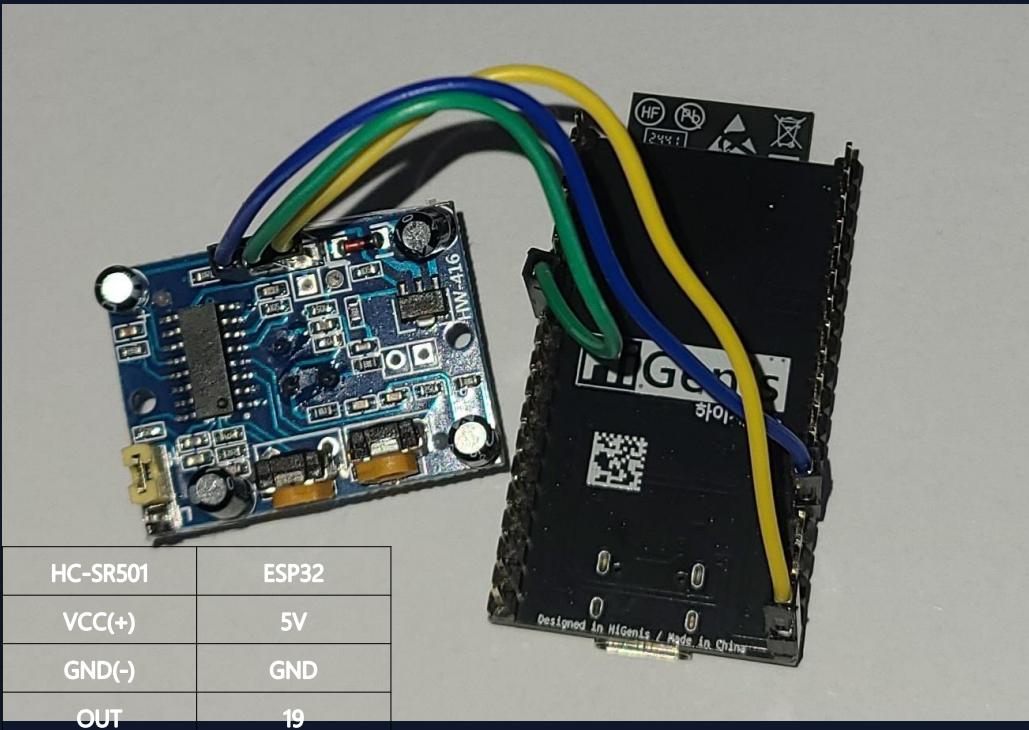


개발 과정

하드웨어 구성 및 펌웨어 개발

▣ 하드웨어 연결

HC-SR501 센서와 ESP-32 보드를 점퍼 케이블로 연결하여 물리적 회로를 구성



▣ ESP-32 펌웨어

Arduino IDE를 사용하여 센서 값을 읽고, Wi-Fi에 연결한 뒤 움직임이 감지되면 서버로 HTTP POST 요청

```
const char* ssid = "wifi"; // WIFI 이름
const char* password = "wifii234"; // WIFI 비밀번호

// Vercel 설정
const char* vercelURL = "https://kiilikiii.vercel.app/api/motion-detected"; // Vercel API URL
const int PIR_PIN = 0;
int lastPirState = LOW;
unsigned long lastMotionTime = 0;
const unsigned long DEBOUNCE_TIME = 2000; // 2초 디바운스

void setup() {
    Serial.begin(115200);
    pinMode(PIR_PIN, INPUT);
    // WiFi 연결
    WiFi.begin(ssid, password);
    Serial.print("WiFi 연결 중...");

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println();
    Serial.println("WiFi 연결 성공!");
    Serial.print("IP 주소: ");
    Serial.println(WiFi.localIP());
}

// 센서 연결
Serial.println("센서 연결화 중... (30초)");
delay(30000);
Serial.println("시스템 준비 완료!");

void loop() {
    int pirValue = digitalRead(PIR_PIN);
    unsigned long currentTime = millis();

    if (pirValue == HIGH && lastPirState == LOW && (currentTime - lastMotionTime) > DEBOUNCE_TIME) {
        Serial.println("움직임 감지! Vercel로 데이터 전송 중...");

        if (sendMotionData()) {
            Serial.println("데이터 전송 성공");
        } else {
            Serial.println("X 데이터 전송 실패");
        }
        lastMotionTime = currentTime;
    }

    lastPirState = pirValue;
    delay(100);
}

bool sendMotionData() {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("WiFi 연결 실패");
        return false;
    }

    const String jsonData = "{\"motion\":true,\"timestamp\":\"" + String(millis()) + "\",\"device_id\":\"ESP32_PIR_01\"}";
    String httpData = "Content-Type: application/json\r\n\r\n" + jsonData;

    HttpClient http;
    http.begin(vercelURL);
    http.addHeader("Content-Type", "application/json");

    // JSON 문자열 직접 생성
    String jsonData = "[";
    jsonData += "\"motion\":true,";
    jsonData += "\"timestamp\":\"" + String(millis()) + "\",";
    jsonData += "\"device_id\":\"ESP32_PIR_01\"";
    jsonData += "]";

    Serial.print("전송 데이터: ");
    Serial.println(jsonData);

    // POST 요청
    int httpResponseCode = http.POST(jsonData);

    if (httpResponseCode > 0) {
        String response = http.getString();
        Serial.print("응답 코드: ");
        Serial.println(httpResponseCode);
        Serial.print("응답 내용: ");
        Serial.println(response);

        http.end();
        return (httpResponseCode == 200);
    } else {
        Serial.print("HTTP 오류: ");
        Serial.println(httpResponseCode);
        http.end();
        return false;
    }
}
```

개발 과정

백엔드 개발



데이터베이스, 서버 운영 방법 변경

초기 MongoDB 사용 계획이었지만 24시간 서버 운영의 현실적 어려움으로 인해 Vercel의 서비스 기능을 사용하며 데이터베이스도 변경

```
// api/motion-detected.js
const { createClient } = require('@vercel/kv');

const kv = createClient({
  url: process.env.KV_REST_API_URL,
  token: process.env.KV_REST_API_TOKEN,
});

module.exports = async (req, res) => {
  console.log('ESP32로부터 움직임 감지 알림을 받았습니다!');

  try {
    await kv.set('motion_status', 'detected', { ex: 5 });
    console.log('Vercel KV에 상태 업데이트 완료: detected');

    res.status(200).json({ status: 'success', message: 'Status Updated to Detected in KV' });

  } catch (error) {
    console.error('Vercel KV 업데이트 오류:', error);

    res.status(500).json({ status: 'error', message: 'Error updating status in KV', error: error.message });
  }
};
```

움직임 감지 데이터 요청

```
// api/get-motion-status.js
const { createClient } = require('@vercel/kv');

const kv = createClient({
  url: process.env.KV_REST_API_URL,
  token: process.env.KV_REST_API_TOKEN,
});

module.exports = async (req, res) => {
  console.log('웹페이지로부터 상태 요청을 받았습니다.');

  try {
    const motionStatus = await kv.get('motion_status');
    const currentStatus = motionStatus || 'no_motion';
    console.log('Vercel KV에서 읽은 상태:', currentStatus);

    res.status(200).json({ status: currentStatus });

  } catch (error) {
    console.error('Vercel KV 읽기 오류:', error);

    res.status(500).json({ status: 'error', message: 'Error reading status from KV', error: error.message });
  }
};
```

데이터베이스 통신

개발 과정

백엔드 개발

The screenshot shows the Vercel KV interface. At the top, there are buttons for "Open in Upstash" and "Connect Project". Below that is a table for managing databases, showing one entry for "Upstash For Redis". The table includes columns for Status, Created, Plan, Current Period, and Period Total. A "Quickstart" section at the bottom provides snippets for various languages (env.local, TypeScript, Python, redis-cli, curl) and includes a "Copy Snippet" button.

```
1 KV_URL="redis://default:ASdHAAIjcdFMzhjljVnZDY3NjAOYjEyOTZiZD1kNTBjODQxYzBiOHaxMA@still-raptor-10055.upstash.io:6379"
2 KV_REST_API_URL="https://still-raptor-10055.upstash.io"
3 KV_REST_API_TOKEN="ASdHAAIjcdFMzhjljVnZDY3NjAOYjEyOTZiZD1kNTBjODQxYzBiOHaxMA"
4 KV_REST_API_READ_ONLY_TOKEN="AidHAAIjcdFM4nP9hxOcnhQznnuK11Nb9RfgyuuBeat77x0fo0g9ycg"
5 REDIS_URL="redis://default:ASdHAAIjcdFMzhjljVnZDY3NjAOYjEyOTZiZD1kNTBjODQxYzBiOHaxMA@still-raptor-10055.upstash.io:6379"
```

Vercel KV

The screenshot shows the kiiikiii website deployment dashboard. It features a "Production Deployment" card for a deployment named "kiiikiii-d01rw4jx-jiijyoungs-projects.vercel.app". The card displays the status as "Ready" (2m ago by JJJJYoung), the source branch as "main", and the commit hash as "98a0cbc final". Below this is a "Deployment Configuration" section with tabs for "Fluid Compute", "Deployment Protection", and "Skew Protection". A note says "To update your Production Deployment, push to the main branch." At the bottom are three cards: "Firewall" (24h), "Observability" (6h), and "Analytics".

웹사이트 클라우드 호스팅

The screenshot shows a terminal window titled "MINGW64/c/server" with the command \$ git status. The output indicates the branch is up to date with 'origin/main'. It then shows changes not staged for commit, specifically a modified file "public/index.html". The next command \$ git add . is run. Following this, a git commit is made with the message "[main 98a0cbc] final" and a note about 1 file changed, 1 insertion(+), and 1 deletion(-). Finally, the command \$ git push is run, showing the process of compressing objects, writing objects, and resolving deltas on the remote GitHub repository.

```
KJ@DESKTOP-CF7SNAT MINGW64 /c/server (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   public/index.html

no changes added to commit (use "git add" and/or "git commit -a")

KJ@DESKTOP-CF7SNAT MINGW64 /c/server (main)
$ git add .

KJ@DESKTOP-CF7SNAT MINGW64 /c/server (main)
$ git commit -m final
[main 98a0cbc] final
 1 file changed, 1 insertion(+), 1 deletion(-)

KJ@DESKTOP-CF7SNAT MINGW64 /c/server (main)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 12 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 367 bytes | 367.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/JJJJYoung/kiiikiii.git
 a60026d..98a0cbc main -> main
```

Github 연동 배포

개발 과정

프론트엔드 개발

```
.body {class="bg-teal-50 h-screen flex items-center justify-center p-4">
  <div class="absolute w-full h-full rounded-full bg-purple-200 flex flex-col items-center justify-center status-transition shadow-lg">
    <span class="pulse-ring inactive" style="animation-delay: 0s;"></span>
    <span class="pulse-ring inactive" style="animation-delay: 0.5s;"></span>
    <span class="pulse-ring inactive" style="animation-delay: 1s;"></span>
    <span class="pulse-ring inactive" style="animation-delay: 1.5s;"></span>
  </div>
  <div id="centralcircle" class="absolute w-full h-full rounded-full bg-purple-200 flex flex-col items-center justify-center status-transition shadow-l1">
    <div class="text-center space-y-4">
      <div class="text-2xl text-gray-600 font-bold">● 움직임 감지 상태 ●</div>
      <div class="text-3xl font-bold text-gray-800 status-transition">● 움직임 감지 중...</div>
    </div>
    <div class="text-xs text-gray-600 text-center mt-4 px-4">
      ESP-32 + HC-SR501 모듈  

      Team Kiiikii
    </div>
    <br/>
    <br/>
    <br/>
    <div id="statusicon" class="text-4xl status-transition">●</div>
  </div>
</div>

<!-- 연결 상태 표시 -->
<div id="connectionStatus" class="fixed top-4 right-4 px-3 py-1 rounded-full text-sm font-medium bg-gray-200 text-gray-600">
  연결 중...
</div>

<script>
let isMotionDetected = false;
let isConnected = false;

async function loadData() {
  const statusDiv = document.getElementById('status');
  const connection = document.getElementById('connectionStatus');
  const centralCircle = document.getElementById('centralcircle');
  const connectionStatus = document.getElementById('connectionstatus');
  const pulseElements = document.querySelectorAll('.pulse-ring');

  try {
    // API 호출
    const response = await fetch('/api/get-motion-status');
    const data = await response.json();

    isConnected = true;
    connectionStatus.textContent = '';
    connectionStatus.className = 'fixed top-4 right-4 px-3 py-1 rounded-full text-sm font-medium bg-green-200 text-green-800';

    if (data.status == "detected") {
      // 움직임 감지
      if (!isMotionDetected) {
        isMotionDetected = true;

        statusDiv.textContent = "▲ 움직임 감지!";
        statusDiv.className = 'text-3xl font-bold text-red-500 status-transition';
        statusIcon.textContent = '🔴';

        centralCircle.className = 'absolute w-full h-full rounded-full bg-red-200 flex flex-col items-center justify-center status-transition shadow';

        // 월드 애플리케이션 시작
        pulseElements.forEach(element => {
          element.classList.remove('inactive');
          element.classList.add('active');
        });
      }
    } else {
      // 움직임 감지
      if (!isMotionDetected) {
        isMotionDetected = false;

        statusDiv.textContent = "▣ 완전";
        statusDiv.className = 'text-3xl font-bold text-green-500 status-transition';
        statusIcon.textContent = '🟢';

        centralCircle.className = 'absolute w-full h-full rounded-full bg-purple-200 flex flex-col items-center justify-center status-transition shadow';

        // 월드 애플리케이션 정지
        pulseElements.forEach(element => {
          element.classList.remove('active');
          element.classList.add('inactive');
        });
      }
    }
  } catch (error) {
    // 오류 처리
    isConnected = false;
    connectionStatus.textContent = '연결 실패';
    connectionStatus.className = 'fixed top-4 right-4 px-3 py-1 rounded-full text-sm font-medium bg-red-200 text-red-800';

    statusDiv.textContent = "▲ 연결 오류";
    statusDiv.className = 'text-3xl font-bold text-gray-500 status-transition';
    statusIcon.textContent = '✗';

    centralCircle.className = 'absolute w-full h-full rounded-full bg-gray-200 flex flex-col items-center justify-center status-transition shadow-lg';

    // 오류 API 호출
    pulseElements.forEach(element => {
      element.classList.remove('active');
      element.classList.add('inactive');
    });
  }
}
loadData();
</script>
```

별도의 CSS 파일 없이 빠르고 일관된 UI를 구축

개발 과정

프레임 제작



문제 해결

⚡ 문제점 : 센서 오작동 (전압 부족)

HC-SR501 센서의 최소 동작 전압은 4.5V이지만, 초기 사용 보드의 출력 전압이 3.3V에 그쳐 센서가 불안정하게 동작하거나 값을 읽지 못하는 문제가 발생했습니다.

💻 해결: 하드웨어 교체 (ESP-32)

안정적인 5V 출력이 가능한 ESP-32 보드로 교체하여 문제를 해결했습니다. 이를 통해 센서에 안정적인 전원을 공급하여 오작동을 근본적으로 방지할 수 있었습니다.

결과 & 시연

주요 구현 기능

실시간 움직임 감지 및 전송

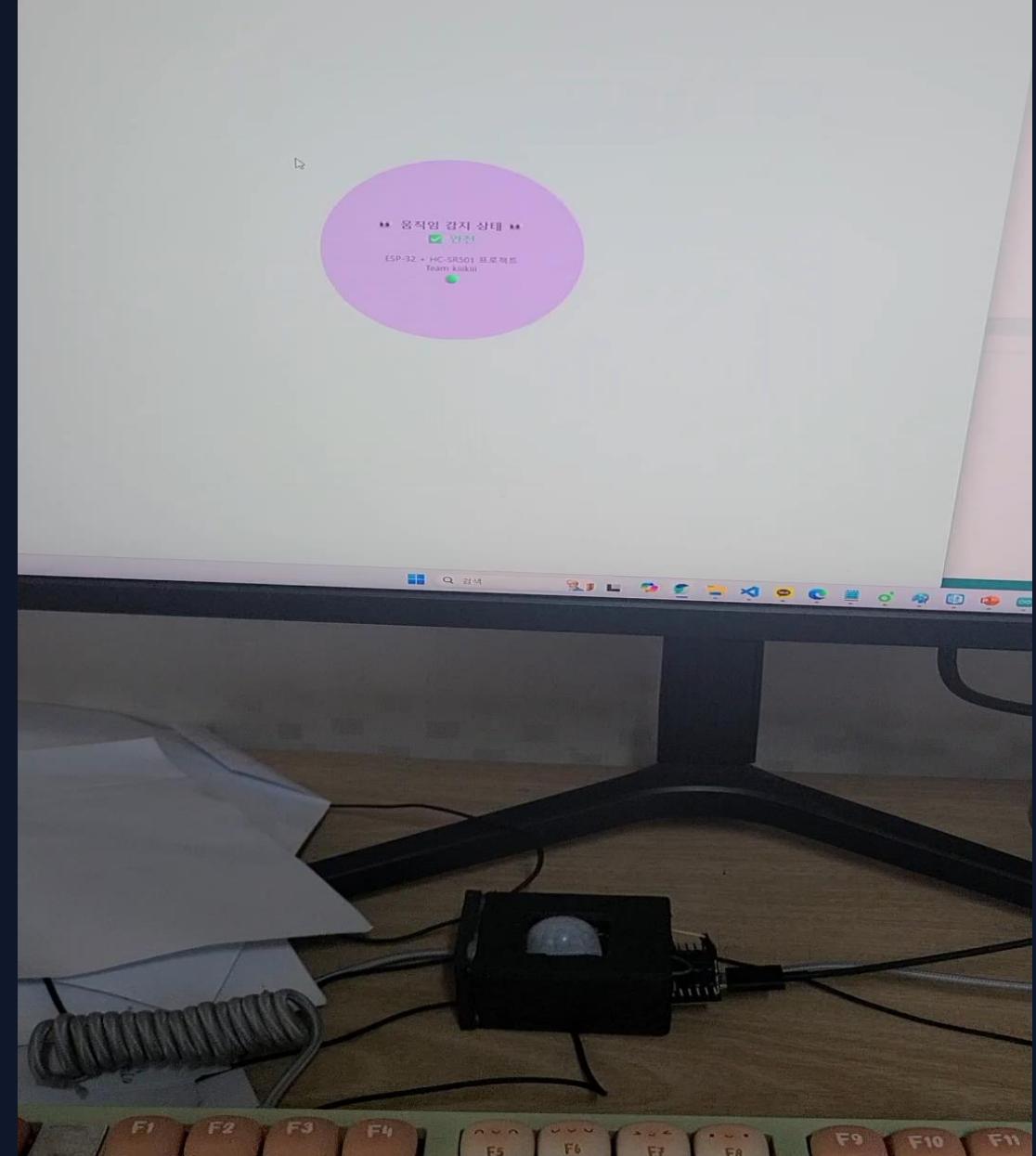
센서가 움직임을 감지하면 즉시 ESP-32가 서버로 데이터를 전송합니다.

반응형 웹디자인

새로운 움직임이 감지될 때마다 UI가 자동 업데이트됩니다.

클라우드 기반 서비스

Vercel을 통해 안정적으로 배포되어 어디서든 접속 및 확인이 가능합니다.



향후 개선 방향

❖ 향후 개선 방향

- ▣ **카메라 모듈 연동:** 움직임 감지 시 스냅샷을 촬영하여 웹 대시보드에 함께 전송하는 기능 추가
- 🔔 **푸시 알림 서비스:** 모바일 기기나 데스크탑에 실시간 푸시 알림을 보내 즉각적인 대응이 가능하도록 개선
- ⌚ **데이터 시각화:** 축적된 감지 로그를 바탕으로 시간대별/요일별 움직임 빈도 등 통계 데이터 시각화
- ₩ **다양한 센서 확장:** 온습도, 조도, 소리 센서 등을 추가하여 종합적인 스마트홈 모니터링 시스템으로 발전

Q & A
