



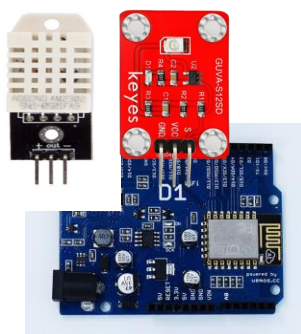
곰팡이 탐지기(IoT)

ESP8266 목표 (시스템 개요)

- 이 프로젝트는 DHT22와 자외선 센서를 활용하여 온도, 습도, 자외선(UV)을 실시간으로 측정하고, 곰팡이 발생 가능성을 예측하여 사용자에게 알림을 제공하는 IoT 기반 환경 모니터링 시스템입니다

핵심 기능

- 🌡️ 온도·습도 실시간 수집 및 시각화
- ☀️ 자외선(UV) 지수 측정
- 👤 사용자별 데이터 저장 및 관리
- 🚨 곰팡이 위험 상태 자동 감지 및 알림
- 📱 웹 대시보드를 통한 실시간 모니터링



ESP8266



시작하기

Step by Step Setup

-  1. 준비물
 - ESP8266, DHT22, GUV A-S12SD, 브레드보드
 - Wi-Fi, Firebase 계정, PC (VSCode/Arduino IDE)
-  2. 하드웨어 연결
 - DHT22: VCC→5V, DAT→D5, GND→GND
 - GUV A-S12SD: +→5V, S→A0, -→GND
-  3. 펌웨어 업로드
 - Arduino IDE 보드 매니저로 "ESP8266" 설치
 - 라이브러리: DHT, FirebaseESP8266
 - 시리얼 모니터(115200)로 센서 값 확인
-  4. Firebase 설정 및 연동
 - 프로젝트 생성 → Realtime Database 활성화
 - Authentication → Email/Password 사용 설정
 - 웹 앱 추가 → apiKey, databaseURL 등 복사

Step by Step use

1- 회원가입 (Sign Up)

- 첫 화면에서 회원가입(Register) 버튼을 누릅니다.
- 이름, 이메일, 비밀번호를 입력하고 계정을 생성합니다.
- Firebase Authentication을 통해 자동으로 UID가 생성됩니다.
👉 *회원가입 완료 후 로그인 화면으로 돌아갑니다.*

2- 로그인 (Login)

- 가입한 이메일과 비밀번호로 로그인합니다.
- 로그인 시 개인 전용 폴더(usuarios/{uid}/)가 생성됩니다.
- 잘못 입력하면 “로그인 실패” 메시지가 표시됩니다.
👉 *성공 시 대시보드 화면으로 자동 이동합니다.*

3- 디바이스 등록 및 연결 (Add & Connect Device)

-1단계: 웹 앱에서 디바이스 등록

- Settings(설정) 페이지로 이동합니다.
- Add Device(디바이스 추가) 버튼을 누릅니다.
- 사용할 디바이스 이름을 입력합니다. 예: “LivingRoom”, “Kitchen”, “BedRoom”.
- 저장(Save) 을 누르면 Firebase Database에 다음 경로가 자동으로 생성됩니다: usuarios/{uid}/devices/{deviceId}/sensores
- 디바이스 리스트에 새 항목이 나타나면 등록 완료입니다.
💡 *이 단계는 사용자 계정에 “빈 디바이스 슬롯” 을 만드는 과정입니다.*

-2단계: 휴대폰으로 디바이스 연결

- 스마트폰의 핫스팟 또는 Wi-Fi에 ESP8266을 연결합니다.
(전원 공급 시 자동으로 연결됨)
- ESP8266 펌웨어에는 Firebase 정보(apiKey, Database URL, UID)가 이미 포함되어 있습니다.
- 기기가 Wi-Fi에 연결되면 자동으로 Firebase로 데이터를 전송합니다.
- 웹 앱의 Dashboard에서 온도·습도·자외선(UV) 값이 표시되면
✅ 연결 성공입니다.
- 이후부터는 휴대폰을 꺼도, 센서가 Wi-Fi에만 연결되어 있으면 실시간으로 데이터가 계속 전송됩니다.

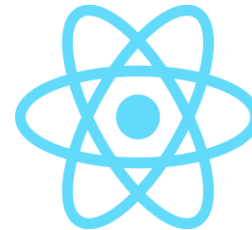
사용 기술

하드웨어: ESP8266 + DHT22 + GUVVA-S12SD

백엔드 / 클라우드: Firebase Realtime Database / Firebase Authentication

프론트엔드: React.js (Vite) + CSS

통신: HTTP / Firebase SDK (Realtime updates)



시스템 구조 (System Architecture & Data Flow)

- 동작 흐름

- 🔧 센서 (DHT22, GUV-A-S12SD)

- 온도, 습도, 자외선(UV) 데이터를 측정

- 🖥️ 마이크로컨트롤러 (ESP8266)

- Wi-Fi를 통해 측정 데이터를 Firebase Realtime Database로 전송

- ☁️ 클라우드 (Firebase)

- 실시간 데이터 저장 및 사용자별 관리

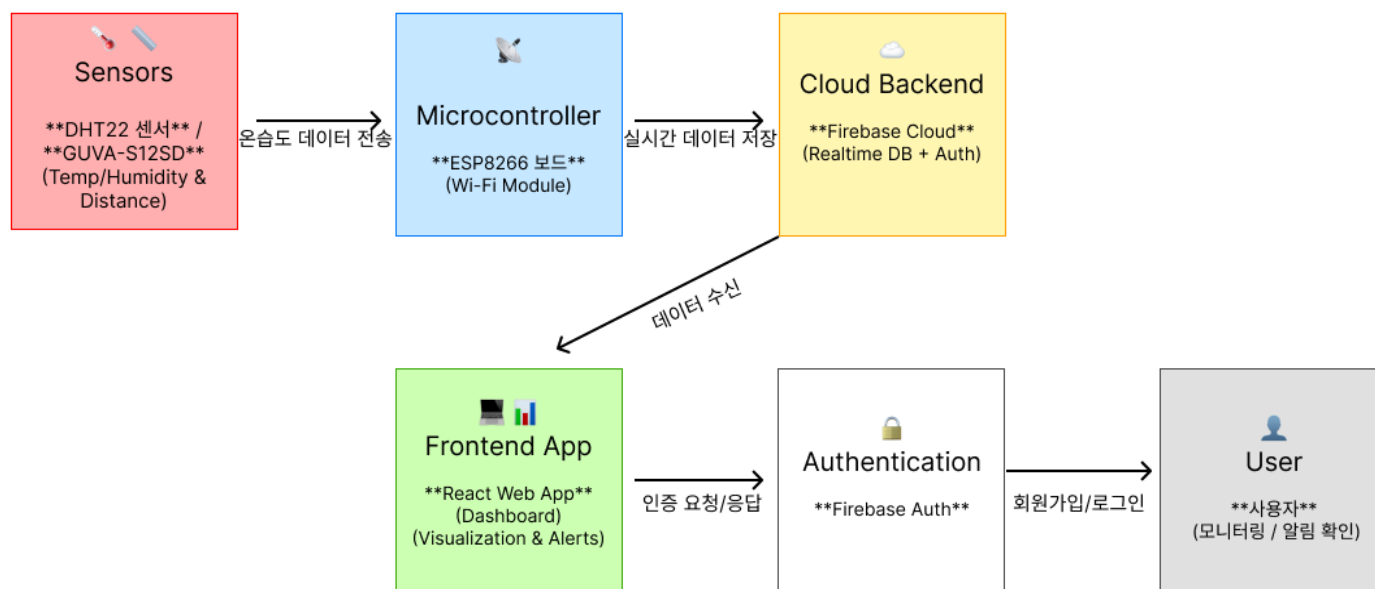
- Firebase Authentication을 통해 사용자 로그인/회원가입 처리

- 💻 프론트엔드 (React 웹 앱)

- 저장된 데이터를 실시간으로 불러와 대시보드 시각화 및 경고 표시

- 사용자는 로그인 후 자신의 데이터를 확인하고 분석 가능

시스템 구조

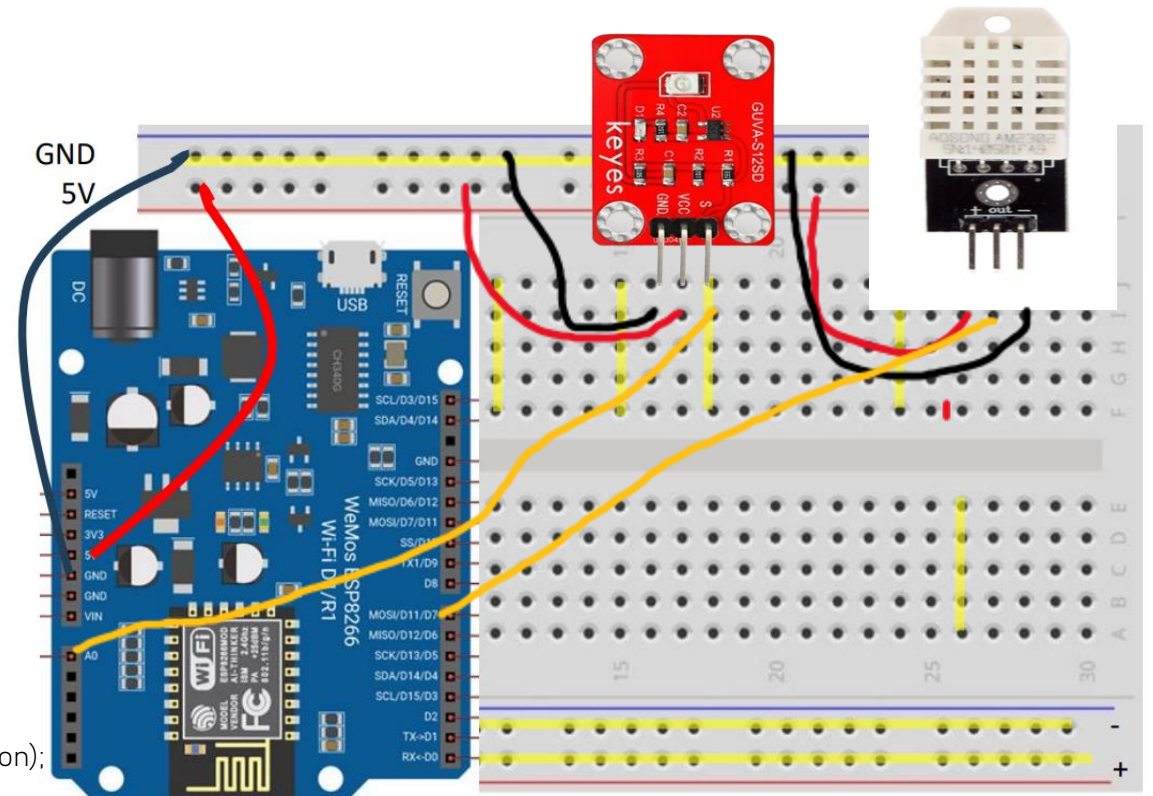


센서 데이터 수집

```
// ✅ 센서 데이터 읽기
float h = dht.readHumidity(); // 습도
float t = dht.readTemperature(); // 온도
int rawValue = analogRead(A0); // 자외선 센서 값
float voltage = rawValue * (3.3 / 1023.0);
float uvIndex = voltage / 0.1; // 0.1V ≈ 1 UV
if (uvIndex > 11) uvIndex = 11; // UV 최대값 제한
```

```
// ✅ Firebase로 전송할 JSON 생성
FirebaseJson json;
json.set("temperature", t);
json.set("humidity", h);
json.set("uv", uvIndex);
json.set("timestamp", millis());
```

```
// ✅ Firebase 업로드
Firebase.pushJSON(fbdo, "/usuarios/" + userID + "/devices/" + deviceId + "/sensores", json);
```

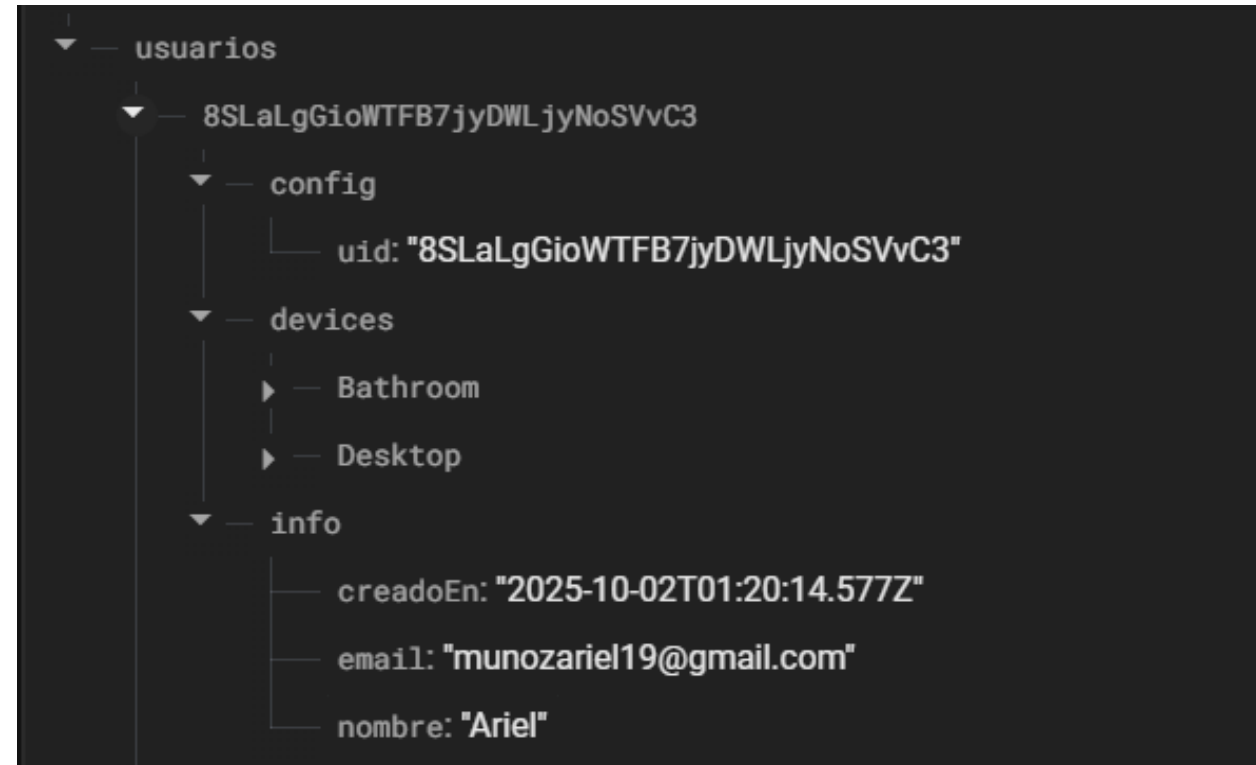


실시간 데이터베이스 구조

Firebase Realtime Database는 사용자별로 구성되어 있으며 각 사용자는 여러 장치를 등록하고 관리할 수 있습니다.

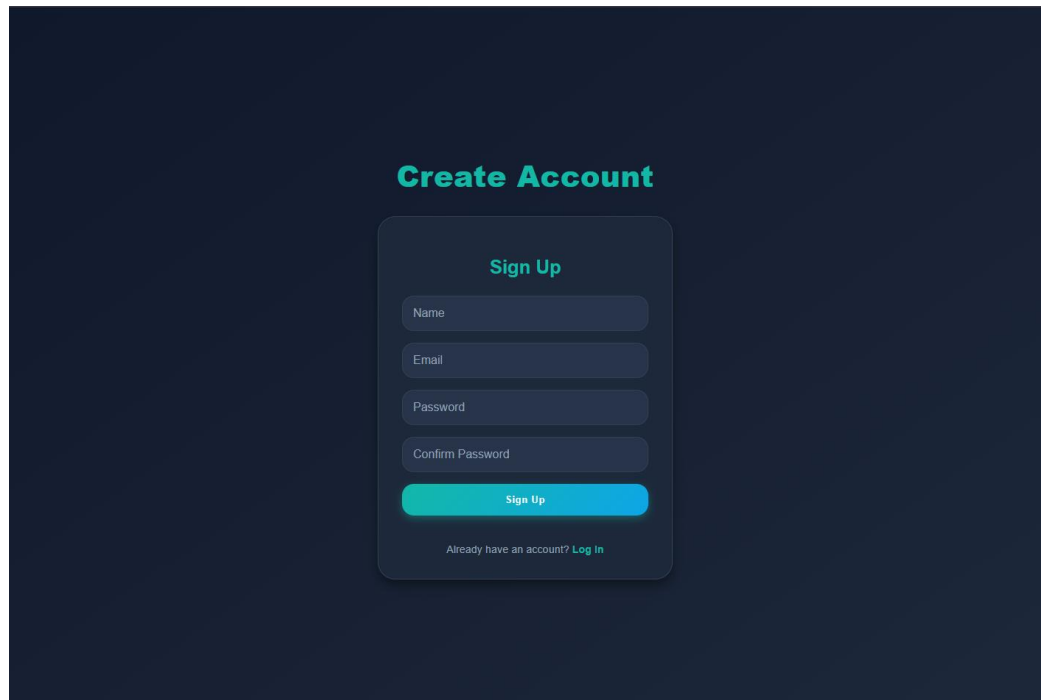
각 장치에는 센서 데이터가 실시간으로 저장됩니다.

- 🌡 온도, 💧 습도, ☀ 자외선(UV) 값 측정
- 매 1분마다 JSON 데이터 전송
- 📡 수집된 데이터는 React 대시보드에 실시간으로 반영됩니다.

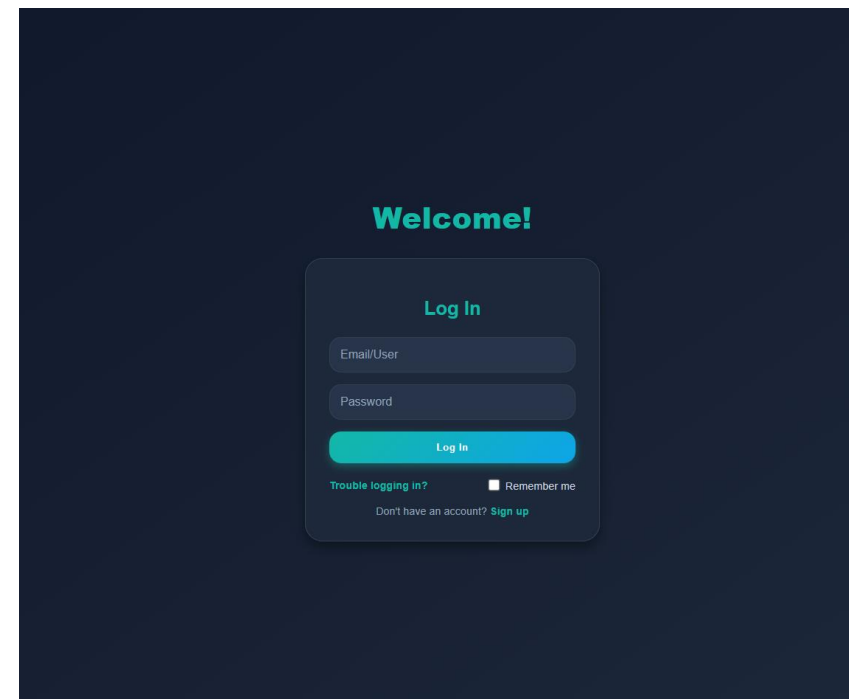


회원가입 및 로그인

사용자는 Firebase Authentication을 통해 계정을 생성하고 로그인할 수 있습니다.
각 사용자는 고유한 UID를 가지며, 데이터베이스 내에 개인 폴더가 자동으로 생성됩니다.
폴더에는 사용자 정보(info), 설정(config), 그리고 장치 목록(devices)이 포함됩니다.



The image shows a 'Create Account' screen with a dark blue background. At the top, the text 'Create Account' is displayed in a light blue font. Below it, there is a rounded rectangle containing the 'Sign Up' section. The 'Sign Up' section has a title 'Sign Up' in light blue. It contains four input fields: 'Name', 'Email', 'Password', and 'Confirm Password', each with a light blue placeholder text. Below these fields is a large, rounded, light blue button labeled 'Sign Up'. At the bottom of the rounded rectangle, there is a link 'Already have an account? Log In' in a small, light blue font.




The image shows a 'Welcome!' screen with a dark blue background. At the top, the text 'Welcome!' is displayed in a light blue font. Below it, there is a rounded rectangle containing the 'Log In' section. The 'Log In' section has a title 'Log In' in light blue. It contains two input fields: 'Email/User' and 'Password', each with a light blue placeholder text. Below these fields is a large, rounded, light blue button labeled 'Log In'. At the bottom of the rounded rectangle, there is a link 'Trouble logging in?' in a small, light blue font, followed by a checkbox labeled 'Remember me' and a link 'Don't have an account? Sign up' in a small, light blue font.

회원가입 및 로그인 코

//  회원가입 처리 함수 (Firebase Auth + DB 초기화)

```
const handleRegister = async () => {
  try {
    // 사용자 등록 (Firebase Authentication)
    const userCredential = await createUserWithEmailAndPassword(auth, email,
password);
    const user = userCredential.user;

    //  DB 구조 생성: config, info, devices
    await set(ref(database, `usuarios/${user.uid}`), {
      config: { uid: user.uid },
      info: {
        nombre: name,
        email: user.email,
        creadoEn: new Date().toISOString(),
      },
      devices: {}, // 장치 목록 (초기 비어 있음)
    });

    alert(t("register.account_created"));
    navigate("/");
  } catch (error) {
    console.error("❌ 회원가입 실패:", error.message);
  }
};
```

//  로그인 처리 함수 (Firebase Auth + UID 검증)

```
const handleLogin = async () => {
  if (!email || !password) {
    alert(t("login.fill_all_fields", "Please fill all fields"));
    return;
  }
  try {
    //  이메일과 비밀번호로 Firebase Auth 인증
    const userCredential = await signInWithEmailAndPassword(auth, email,
password);
    const user = userCredential.user;

    console.log("로그인된 사용자:", user.uid);

    //  UID 확인 및 DB에 기록 (예외적 복구용)
    await set(ref(database, `usuarios/${user.uid}/config/uid`), user.uid);

    console.log("UID 저장 완료:", user.uid);

    //  로그인 성공 시 홈 화면으로 이동
    alert(t("login.login_success"));

    navigate("/home");
  } catch (error) {
    console.error("❌ 로그인 실패:", error);
    alert("❌ " + t("login.login_error") + ": " + error.message);
  }
};
```

대시보드

실시간 센서 데이터를 표시하고, 최근 데이터를 그래프로 시각화하는 메인 화면입니다.

주요 기능

- 📡 sensors 노드의 데이터를 실시간으로 받아와 사용자 고유 UID에 맞게 복사
- 🌡️ 현재 온도 (°C/°F 변환 가능)
- 💧 현재 습도 (%)
- ☀️ 현재 자외선(UV) 지수
- 📈 최근 3시간 이내 데이터를 5분 간격으로 필터링해 그래프로 표시
- 🚨 온도, 습도, 자외선이 임계치를 초과하면 Alert 상태 표시



대시보드 코드

```
// ✅ Firebase에서 사용자별 디바이스 목록 불러오기
useEffect(() => {
  const user = auth.currentUser;
  if (!user) return;

  const devicesRef = ref(database, `usuarios/${user.uid}/devices`);
  onValue(devicesRef, (snapshot) => {
    const data = snapshot.val() || {};
    const list = Object.entries(data).map(([id, value]) => ({
      id,
      name: value.name || id,
    }));
    setDevices(list);
    if (list.length > 0 && !selectedDevice) {
      setSelectedDevice(list[0].id); // 첫 디바이스 자동 선택
    }
  });
}, []);
```

```
// ✅ 선택된 디바이스의 센서 데이터 실시간 수신
useEffect(() => {
  if (!selectedDevice) return;
  const user = auth.currentUser;
  if (!user) return;

  const deviceRef = ref(
    database,
    `usuarios/${user.uid}/devices/${selectedDevice}/sensores`
  );

  const unsubscribe = onValue(deviceRef, (snapshot) => {
    const data = snapshot.val();
    if (!data) {
      setLatestData(null);
      setChartData([]);
      setLoading(false);
      return;
    }
  });

  // 📦 데이터 배열로 변환 및 3시간 내 필터링
  const entries = Object.entries(data).map(([id, v]) => ({
    id,
    ...v,
    time: new Date(v.timestamp).toLocaleTimeString([], {
      hour: "2-digit",
      minute: "2-digit",
    }),
  }));
}, []);
```

```
const threeHoursAgo = Date.now() - 3 * 60 * 60 * 1000;
const last3hData = entries.filter((e) => e.timestamp >= threeHoursAgo);
```

```
// 🌸 5분 간격으로 그래프용 데이터 구성
const filteredEvery5Min = [];
let lastTime = 0;
const interval = 5 * 60 * 1000;
```

```
for (const entry of last3hData) {
  if (entry.timestamp - lastTime >= interval) {
    filteredEvery5Min.push(entry);
    lastTime = entry.timestamp;
  }
}
```

```
setLatestData(filteredEvery5Min.at(-1)); // 최근 데이터
setChartData(filteredEvery5Min); // 그래프 데이터
setLoading(false);
});
```

```
return () => unsubscribe();
}, [selectedDevice]);
```

```
// 🚨 임계값 초과 시 Alert 상태 표시
```

```
const status =
  latestData &&
  (latestData.temperature > tempAlertLimit ||
   latestData.humidity > humidAlertLimit ||
   latestData.uv > 8)
  ? t("home.alert")
  : t("home.normal");
```

센서 기록 상세

센서 데이터를 날짜별로 필터링해 자세히 볼 수 있는 페이지입니다.

사용자가 선택한 디바이스별로 기록을 조회할 수 있으며 모든 데이터는 Firebase Realtime Database에서 실시간으로 불러옵니다.

- 🔍 Firebase 경로: usuarios/{uid}/devices/{device}/sensores
각 사용자별, 디바이스별 센서 데이터 실시간 조회
- 📱 디바이스 선택 기능: 드롭다운으로 여러 센서 장치 중 하나 선택 가능
- 📅 날짜별 필터 기능 (<input type="date" />)
- 📄 테이블 표시 항목: 데이터 번호, 날짜, 시간
- 온도, 습도, 자외선 지수
- 📱 반응형 UI로 모바일에서도 보기 편함
- ⬅️ Home 버튼으로 메인 대시보드로 이동

#	Date	Time	Temperature (°C)	Humidity (%)	UV Index
1	2025.10.09	08:02 p.m.	25.2	57.7	0.194
2	2025.10.09	08:13 p.m.	23.7	60.2	0.226
3	2025.10.09	08:14 p.m.	24.1	61.1	0.129
4	2025.10.09	08:15 p.m.	24.1	61.1	0.097
5	2025.10.09	08:16 p.m.	24.3	59.2	0.226
6	2025.10.09	08:17 p.m.	24.2	58.7	0.194

센서 기록 상세 코드

```
// ✅ Firebase에서 디바이스 목록 불러오기
useEffect(() => {
  const user = auth.currentUser;
  if (!user) return; // 사용자 인증 확인

  const uid = user.uid;
  const devicesRef = ref(database, `usuarios/${uid}/devices`);

  // 📁 Firebase에서 디바이스 목록 실시간 수신
  const unsubscribeDevices = onValue(devicesRef, (snapshot) => {
    const data = snapshot.val();
    if (!data) {
      setDevices([]); // 데이터 없을 시 초기화
      return;
    }

    const deviceNames = Object.keys(data); // 💡 디바이스 이름 목록
    // 생성
    setDevices(deviceNames);

    // ⚙️ 디바이스 선택이 없을 경우 첫 번째 자동 선택
    if (!selectedDevice && deviceNames.length > 0) {
      setSelectedDevice(deviceNames[0]);
    }
  });

  return () => unsubscribeDevices();
}, []);
```

```
// ✅ 날짜별 필터링 로직
const filteredData = useMemo(() => {
  if (!selectedDate) return historicalData; // 날짜 선택 안 하면 전체
  // 표시
  return historicalData.filter((d) => {
    // 🕒 timestamp를 YYYY-MM-DD 형식으로 변환 후 비교
    const date = new Date(d.timestamp).toISOString().split("T")[0];
    return date === selectedDate;
  });
}, [historicalData, selectedDate]);
```

```
// ✅ 선택된 디바이스의 센서 데이터 실시간 수신
useEffect(() => {
  const user = auth.currentUser;
  if (!user || !selectedDevice) return;

  const uid = user.uid;
  const sensoresPath = `usuarios/${uid}/devices/${selectedDevice}/sensores`;
  console.log("📡 Leyendo datos desde:", sensoresPath);

  const sensoresRef = ref(database, sensoresPath);
  const unsubscribeData = onValue(sensoresRef, (snapshot) => {
    const data = snapshot.val();
    if (!data) {
      setHistoricalData([]); // 데이터 없을 시 초기화
      return;
    }

    // 📌 Firebase 데이터를 배열로 변환 및 timestamp 기준 정렬
    const entries = Object.entries(data).map(([id, value]) => ({
      id,
      ...value,
    }));
    entries.sort((a, b) => a.timestamp - b.timestamp);
    setHistoricalData(entries);
  });

  return () => unsubscribeData();
}, [selectedDevice]);
```

```
// ✅ 필터링된 데이터 테이블 렌더링

  {filteredData.map((d, index) => (
    <tr key={index}>
      <td>{index + 1}</td> { /* 💎 데이터 번호 */}
      <td>{new Date(d.timestamp).toLocaleDateString("ko-KR")}</td> { /* 📅 날짜 */}
      <td>
        {new Date(d.timestamp).toLocaleTimeString([], {
          hour: "2-digit",
          minute: "2-digit",
        })}
      </td> { /* 🕒 시간 */}
      <td>{d.temperature?.toFixed(1)}</td> { /* 🌡 온도 */}
      <td>{d.humidity?.toFixed(1)}</td> { /* 💧 습도 */}
      <td>{d.uv?.toFixed(3)}</td> { /* ☀ 자외선 */}
    </tr>
  ))}
</tbody>
```

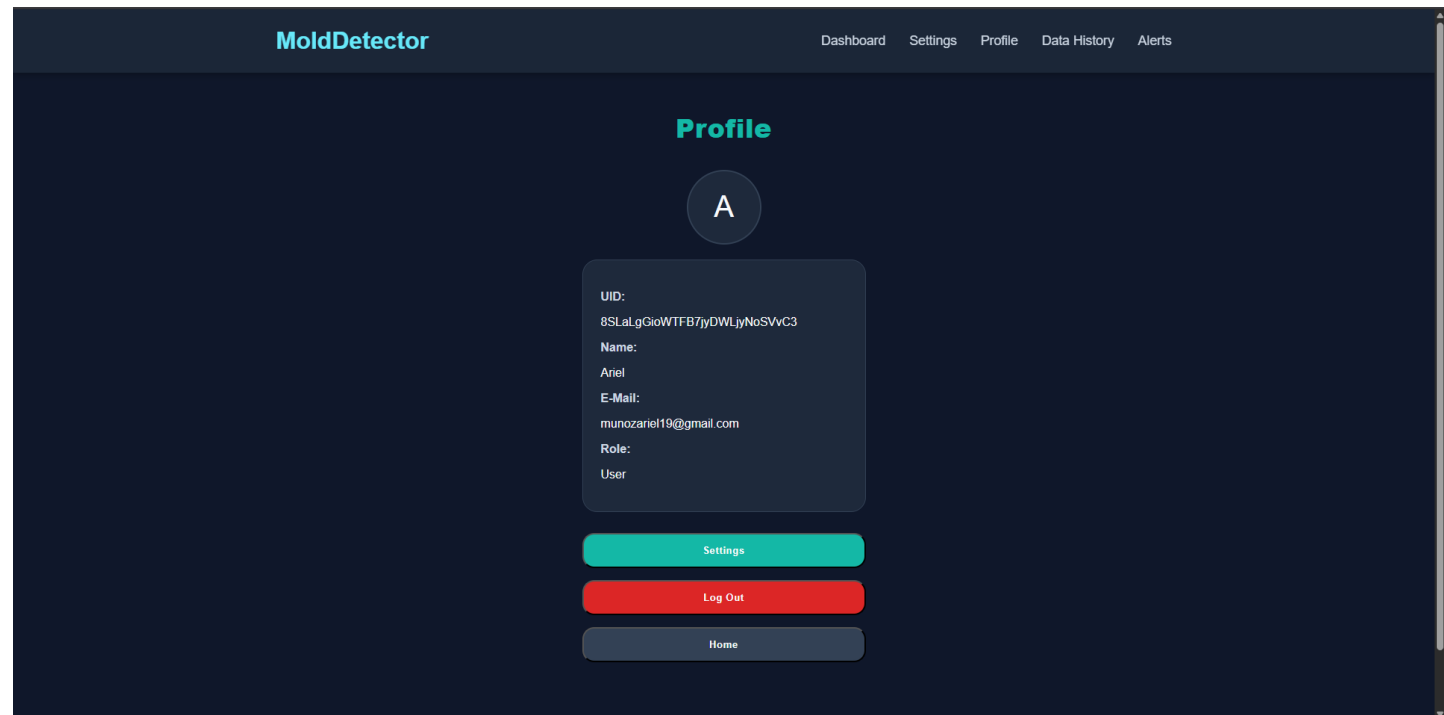
사용자 프로필 페이지

로그인한 사용자의 정보를 보여주는
페이지입니다.
표시 항목:

- UID
- 이름
- 이메일
- 역할(Role) (기본값: "User")

버튼:

- ⚙ Settings → 설정 페이지 이동
- 🚪 Log Out → Firebase signOut()으로
로그아웃 후 /login으로 이동
- 🏠 Home → 대시보드 이동



프로필 페이지 코드

```
// ✅ Firebase에서 사용자 정보 가져오기
useEffect(() => {
  const auth = getAuth(); // 🔑 Firebase Auth 인스턴스 가져오기
  const user = auth.currentUser; // 현재 로그인된 사용자 확인

  if (user) {
    const uid = user.uid; // 사용자 UID
    // 💡 Firebase Realtime DB에서 사용자 정보 조회
    get(ref(database, `usuarios/${uid}/info`)).then((snapshot) => {
      if (snapshot.exists()) {
        setUserInfo({ uid, ...snapshot.val() }); // ✅ 정보 존재 시 상태 저장
      } else {
        setUserInfo({ uid, email: user.email }); // ⚠️ 정보 없을 시 이메일만 표시
      }
    });
  }
}, []);
```

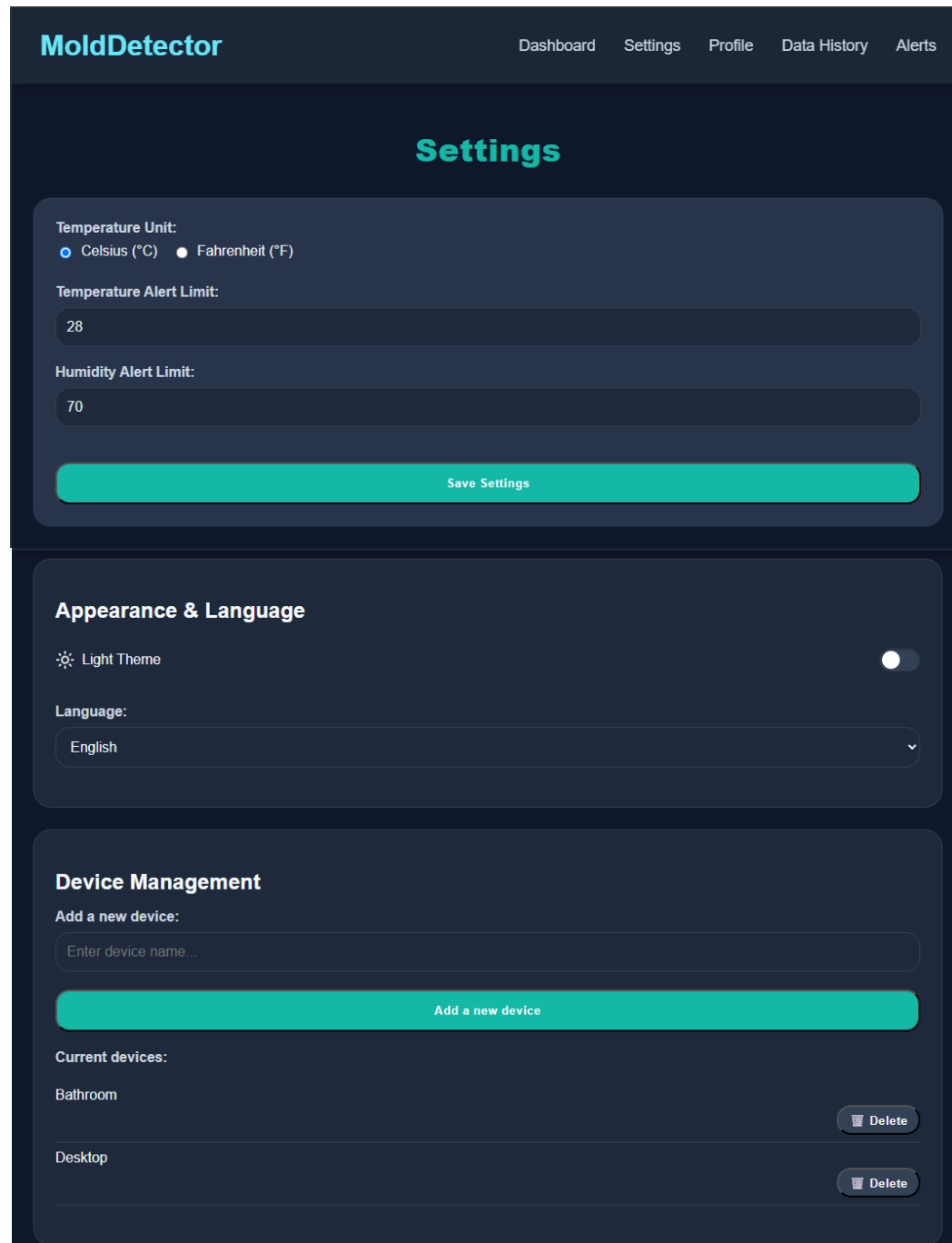
```
// ✅ 로그아웃 기능
const handleLogout = async () => {
  const auth = getAuth();
  try {
    await signOut(auth); // 🔑 Firebase 인증 세션 종료
    console.log("✅ Session closed successfully");
    navigate("/login"); // ➡ 로그인 페이지로 이동
  } catch (error) {
    console.error("❌ Logout error:", error.message); // 에러 로그
  }
};
```

설정 페이지

센서 경고 임계값과 앱 환경(테마/언어), 디바이스 관리를 변경할 수 있는 페이지입니다.

주요 기능

- 🌡 온도/습도 임계값 변경 → Firebase에 저장
- °C ↔ °F 단위 전환 (isCelsius)
- 🌙 Theme: Light / Dark 토글
- 🌐 Language: 한국어/스페인어/영어 선택 (i18n)
- 🛠 Device Management: 디바이스 추가/목록/삭제



설정 페이지 코드

//  Firebase에서 사용자 설정 불러오기

```
useEffect(() => {
  const uid = auth.currentUser.uid;
  onValue(ref(database, `usuarios/${uid}/config`), (snap) => {
    const cfg = snap.val() || {};
    setIsCelsius(cfg.isCelsius ?? true);
    setTempAlertLimit(cfg.tempAlertLimit ?? 28);
    setHumidAlertLimit(cfg.humidAlertLimit ?? 70);
    i18n.changeLanguage(cfg.language ?? "en");
  });
}, []);
```

//  설정 저장 (임계값, 단위, 언어)

```
const saveSettings = async () => {
  const uid = auth.currentUser.uid;
  await set(ref(database, `usuarios/${uid}/config`), {
    isCelsius, tempAlertLimit, humidAlertLimit,
    language: i18n.language,
  });
  alert(t("settings.saved"));
};
```

//  다국어(i18n) 언어 변경

```
const handleLanguage = (lang) => {
  i18n.changeLanguage(lang);
  set(ref(database, `usuarios/${uid}/config/language`), lang);
};
```

//  Theme Toggle (light ↔ dark)

```
const toggleTheme = () => setTheme(t === "dark" ? "light" : "dark");
useEffect(() => {
  document.documentElement.dataset.theme = theme;
}, [theme]);
```

// 장치 추가

```
const addDevice = async () => {
  await push(ref(database, `usuarios/${uid}/devices`), { name:
  deviceName });
};
```

//  장치 삭제

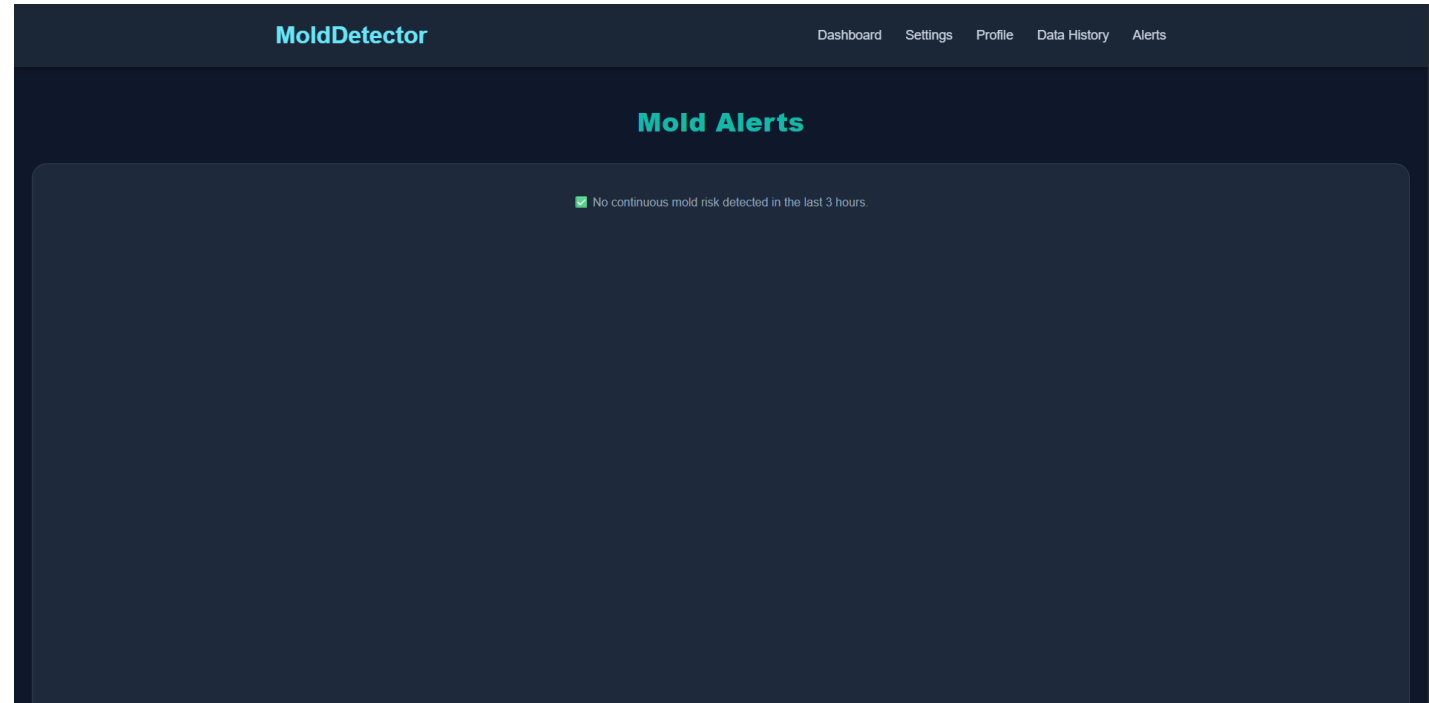
```
const deleteDevice = async (id) => {
  await remove(ref(database, `usuarios/${uid}/devices/${id}`));
};
```

알림 페이지

센서 데이터의 최근 3시간 동안 곰팡이 발생 조건을 감지하여 사용자에게 알림을 표시하는 페이지입니다.

주요 기능:

- 📁 Firebase 경로:
usuarios/{uid}/devices/{device}/sensores
- 🕒 최근 3시간 데이터만 분석
- 🌡️ 임계값 조건: 온도 > 설정값 / 습도 > 설정값 / 자외선 < 특정 수준
- ⚠️ 위험 감지 시 “곰팡이 위험 경고” 표시
- ✅ 조건 미충족 시 “정상 상태” 메시지 표시
- 🔄 실시간 데이터 반영 (onValue 사용)



알림 페이지 코드

//  Firebase에서 최근 데이터 불러오기

```
useEffect(() => {  
  const user = auth.currentUser;  
  if (!user) return;  
  
  const deviceRef = ref(database, `usuarios/${user.uid}/devices/${selected}/sensores`);  
  onValue(deviceRef, (snapshot) => {  
    const data = snapshot.val() || {};  
    const entries = Object.values(data).sort((a, b) => b.timestamp - a.timestamp);  
    setSensorData(entries.slice(0, 50)); // 최근 50개만  
  });  
}, [selected]);
```

//  곰팡이 위험 감지 로직

```
useEffect(() => {  
  if (!sensorData.length) return;  
  
  const now = Date.now();  
  const last3h = sensorData.filter(d => now - d.timestamp <= 3 * 60 * 60 * 1000);  
  
  const risk = last3h.some(d =>  
    d.temperature > tempAlertLimit &&  
    d.humidity > humidAlertLimit &&  
    d.uv < 0.2  
  );  
  
  setIsRisk(risk);  
}, [sensorData]);
```