Modul 6: Integrasi API & Keamanan Lintas Platform

Mata Kuliah: Rekayasa Perangkat Lunak Lintas Platform / Integrasi Sistem

Tujuan Praktikum:

1. Mahasiswa mampu menghubungkan aplikasi klien (Web & Mobile) dengan layanan *backend* (BFF) menggunakan protokol HTTP secara asinkron.
2. Mahasiswa mampu mengatasi masalah kebijakan keamanan lintas asal (*Cross-Origin Resource Sharing* - CORS) pada sisi server.
3. Mahasiswa mampu melakukan serialisasi dan rekonstruksi objek data berbasis JSON (*JSON Parsing*) lintas bahasa pemrograman.

1. Pendahuluan

Integrasi sistem adalah tahapan di mana komponen-komponen terpisah (Backend, Web, Mobile) yang telah dirancang sebelumnya mulai disatukan untuk berkomunikasi secara dinamis. Dalam ekosistem lintas platform, tantangan utamanya terletak pada cara penanganan jaringan dan kebijakan keamanan pada masing-masing sistem operasi.

Pada modul ini, kita akan menghubungkan dashboard Web React (Modul 4) dan aplikasi Mobile Flutter (Modul 5) agar mengonsumsi data langsung dari BFF NestJS (Modul 2) secara *real-time*.

2. Persiapan Lingkungan Kerja

- **Backend (modular-system):** Proyek NestJS dari Modul 2.
- **Web Client (ui-system):** Proyek React dari Modul 4 (Jalankan perintah: `npm install axios` di terminal proyek web Anda).

• Jalankan perintah di terminal Web Client: npm install axios

```
(base) very@very-Inspiron-3443:~/rlp/ui-system$ npm install axios

added 22 packages, and audited 237 packages in 3s

57 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 11.12.1 -> 11.14.1
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.14.1
npm notice To update run: npm install -g npm@11.14.1
npm notice
(base) very@very-Inspiron-3443:~/rlp/ui-system$
```

- **Mobile Client:** Proyek Flutter dari Modul 5 (Tambahkan dependensi `http` pada berkas `pubspec.yaml`).

1. Buka terminal di VS Code pada proyek Flutter Anda (`iot_mobile_app`).
2. Jalankan perintah berikut:
   flutter pub add http
3. Perintah ini akan otomatis mengunduh package dan mendaftarkannya ke dalam berkas `pubspec.yaml`.

```
(base) very@very-Inspiron-3443:~/rlp/iot_mobile_app$ flutter pub add http
Resolving dependencies...
Downloading packages...
+ http 1.6.0
+ http_parser 4.1.2
  matcher 0.12.19 (0.12.20 available)
  meta 1.17.0 (1.18.2 available)
  test_api 0.7.10 (0.7.12 available)
+ typed_data 1.4.0
  vector_math 2.2.0 (2.3.0 available)
+ web 1.1.1
Changed 4 dependencies!
4 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
(base) very@very-Inspiron-3443:~/rlp/iot mobile app$
```

- **Alat Pengujian:** Postman atau Insomnia untuk memvalidasi *request*.

3. Implementasi Kode Secara Rinci

A. Mengubah Sensor Data menjadi List

Update file **src/sensor/sensor.servic.ts** pada project **Server Backend (NestJS) → modular-system**

```
// src/sensor/sensor.service.ts
import { Injectable } from '@nestjs/common';
import { SensorData } from './domain/sensor.entity';

@Injectable()
export class SensorService {
  // Simulasi pengambilan data dari database/IoT
  //private mockData = new SensorData('S001', 28.5, '°C', 'DHT22-V2', 85, new
Date());

  private mockData: SensorData[] = [
    new SensorData('S1', 28.5, '°C', 'DHT22-V2', 85, new Date()),
    new SensorData('S2', 24.5, '°C', 'Ambient Temp', 85, new Date()),
    new SensorData('S3', 1012, 'hPa', 'Pressure', 85, new Date()),
    new SensorData('S4', 88.2, '°C', 'Core Heat', 85, new Date()),
    new SensorData('S5', 12, '%', 'Battery Level', 85, new Date()),
  ];

  // BFF: Transformasi data khusus untuk Mobile
  // Fokus: Hemat Bandwidth & Kecepatan Baca
  getSummaryForMobile() {
    return this.mockData.map((sensor) => ({
      id: sensor.id, // Ditambahkan agar aplikasi mobile tahu ini data dari sensor mana
      v: sensor.value,
      u: sensor.unit,
      st: sensor.isNormal() ? 'OK' : 'WARN',
      batt: sensor.isBatteryCritical() ? 'LOW' : 'GOOD',
    }));
  }

  // BFF: Transformasi data khusus untuk Web Dashboard
  // Fokus: Analitik Lengkap & Transparansi Data
  getFullDetailsForWeb() {
    return this.mockData.map((sensor) => ({
      id: sensor.id,
      deviceName: sensor.deviceName,
      reading: {
        current: sensor.value,
        unit: sensor.unit,
        isOptimal: sensor.isNormal(),
      },
      status: {
        battery: `${sensor.batteryLevel}%`,
        lastSeen: sensor.timestamp.toISOString(),
      },
    }));
```

```
  }
}
```

B. Konfigurasi Keamanan di Backend (NestJS) → modular-system

Secara default, browser akan memblokir request dari asal (*origin*) yang berbeda demi keamanan. Kita perlu mengaktifkan CORS pada berkas utama NestJS agar aplikasi React dapat membaca data API.

Update file **src/main.ts** pada project **Server Backend (NestJS) → modular-system**

```
// backend/src/main.ts
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  // Logika Keamanan: Mengaktifkan CORS untuk menerima request luar
  app.enableCors({
    origin: '*', // Pada tahap produksi, batasi ke domain web spesifik saja
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    allowedHeaders: 'Content-Type, Authorization',
  });

  await app.listen(3000);
}
void bootstrap();
```

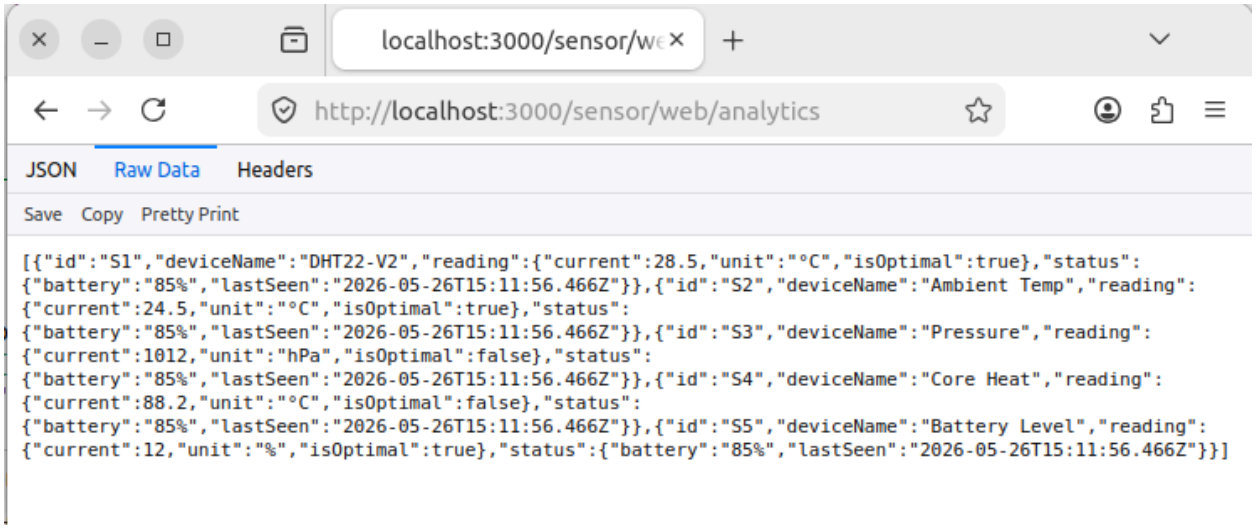
Jalankan **Server Backend (NestJS) → modular-system** dengan perintah berikut pada terminal proyek backend:

```
~/rlp/modular-system$ npm run start:dev
```

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

[10:11:47 PM] Starting compilation in watch mode...

[10:11:55 PM] Found 0 errors. Watching for file changes.

[Nest] 19160  - 05/26/2026, 10:11:56 PM     LOG [NestFactory] Starting Nest application...
[Nest] 19160  - 05/26/2026, 10:11:56 PM     LOG [InstanceLoader] AppModule dependencies initialized +30ms
[Nest] 19160  - 05/26/2026, 10:11:56 PM     LOG [InstanceLoader] SensorModule dependencies initialized +1ms
[Nest] 19160  - 05/26/2026, 10:11:56 PM     LOG [RoutesResolver] AppController {/}: +6ms
[Nest] 19160  - 05/26/2026, 10:11:56 PM     LOG [RouterExplorer] Mapped {/, GET} route +11ms
[Nest] 19160  - 05/26/2026, 10:11:56 PM     LOG [RoutesResolver] SensorController {/sensor}: +1ms
[Nest] 19160  - 05/26/2026, 10:11:56 PM     LOG [RouterExplorer] Mapped {/sensor/mobile/dashboard, GET} route +2ms
[Nest] 19160  - 05/26/2026, 10:11:56 PM     LOG [RouterExplorer] Mapped {/sensor/web/analytics, GET} route +1ms
[Nest] 19160  - 05/26/2026, 10:11:56 PM     LOG [NestApplication] Nest application successfully started +6ms
```

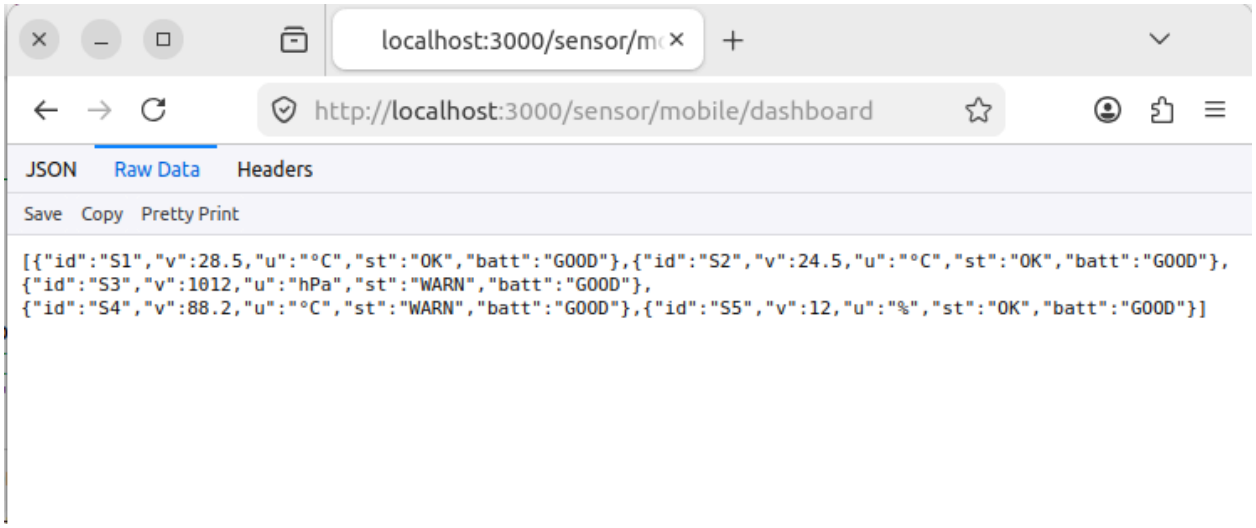
Akses API **/sensor/web/analytics** menggunakan browser atau aplikasi Postman pada alamat:

http://localhost:3000/sensor/web/analytics



Akses API **/sensor/mobile/dashboard** menggunakan browser atau aplikasi Postman pada alamat:

http://localhost:3000/sensor/mobile/dashboard



C. Integrasi API pada Web Client (React + Axios) → ui-system

Ubah data simulasi statis di berkas `src/App.jsx` pada proyek React (Modul 4) dengan mengambil data dinamis menggunakan library **Axios**.

```
// src/App.jsx
import { useState, useEffect } from 'react';
import axios from 'axios';
import StatusCard from './components/StatusCard';
import './index.css';

function App() {
  const [isOnline, setIsOnline] = useState(navigator.onLine);
  const [sensor, setSensor] = useState(null);
  const [loading, setLoading] = useState(true);

  // 1. Detektor Status Jaringan Offline/Online
  useEffect(() => {
    const handleOnline = () => setIsOnline(true);
    const handleOffline = () => setIsOnline(false);
    window.addEventListener('online', handleOnline);
    window.addEventListener('offline', handleOffline);
    return () => {
      window.removeEventListener('online', handleOnline);
      window.removeEventListener('offline', handleOffline);
    };
```

```
  }, []);

  // 2. Fetch Data dari BFF NestJS (Modul 2)
  useEffect(() => {
    const fetchApiData = async () => {
      try {
        const response = await axios.get('http://localhost:3000/sensor/web/analytics');
        setSensor(response.data);
      } catch (error) {
        console.error("Gagal terhubung ke BFF NestJS:", error);
      } finally {
        setLoading(false);
      }
    };
    fetchApiData();
  }, []);

  if (loading) {
    return (
            <div  className="min-h-screen  bg-tech-bg  flex  items-center  justify-center
font-bold">
        Menghubungkan ke BFF Node.js...
      </div>
    );
  }

  return (
    <div className="min-h-screen bg-tech-bg p-safe-area font-sans text-slate-900">
      {/* Notifikasi Offline */}
      {!isOnline && (
          <div className="fixed top-0 left-0 right-0 bg-red-600 text-white p-2 text-center
z-50 animate-bounce">
          Anda sedang offline. Menampilkan data cadangan lokal.
        </div>
      )}

      {/* Header */}
      <header className="max-w-5xl mx-auto mb-8">
        <div className="flex items-center gap-3">
          <div className="relative flex h-3 w-3">
```

```
          <span className="animate-ping absolute inline-flex h-full w-full rounded-full bg-green-400 opacity-75"></span>
                        <span className="relative inline-flex rounded-full h-3 w-3 bg-green-500"></span>
        </div>
        <div>
          <h1 className="text-xl font-black uppercase tracking-tighter">
            SensorNodes <span className="text-tech-primary">Controller</span>
          </h1>
                    <p className="text-slate-500 text-[10px] font-bold tracking-widest uppercase">
            BFF Connected & Active
          </p>
        </div>
      </div>
    </header>

    {/* Grid Dashboard Menggunakan Data API */}
            <main className="max-w-5xl mx-auto grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
      {Array.isArray(sensors) && sensors.map((sensor) => (
        <StatusCard
          key={sensor.id} // Wajib di React untuk optimasi performa render list
          label={sensor.deviceName}
          value={sensor.reading.current}
          unit={sensor.reading.unit}
          status={sensor.reading.isOptimal ? 'OK' : 'DANGER'}
          isCompact={false}
        />
      ))}
    </main>

    {/* Footer / Status Bar untuk Mobile */}
    <footer className="max-w-5xl mx-auto mt-12 opacity-50 text-center"><br/>
      <p className="text-[10px] font-medium italic">
        BFF Node.js Connected • System Version 2.0.4
      </p>
    </footer>
  </div>
);
```

```
}

export default App;
```

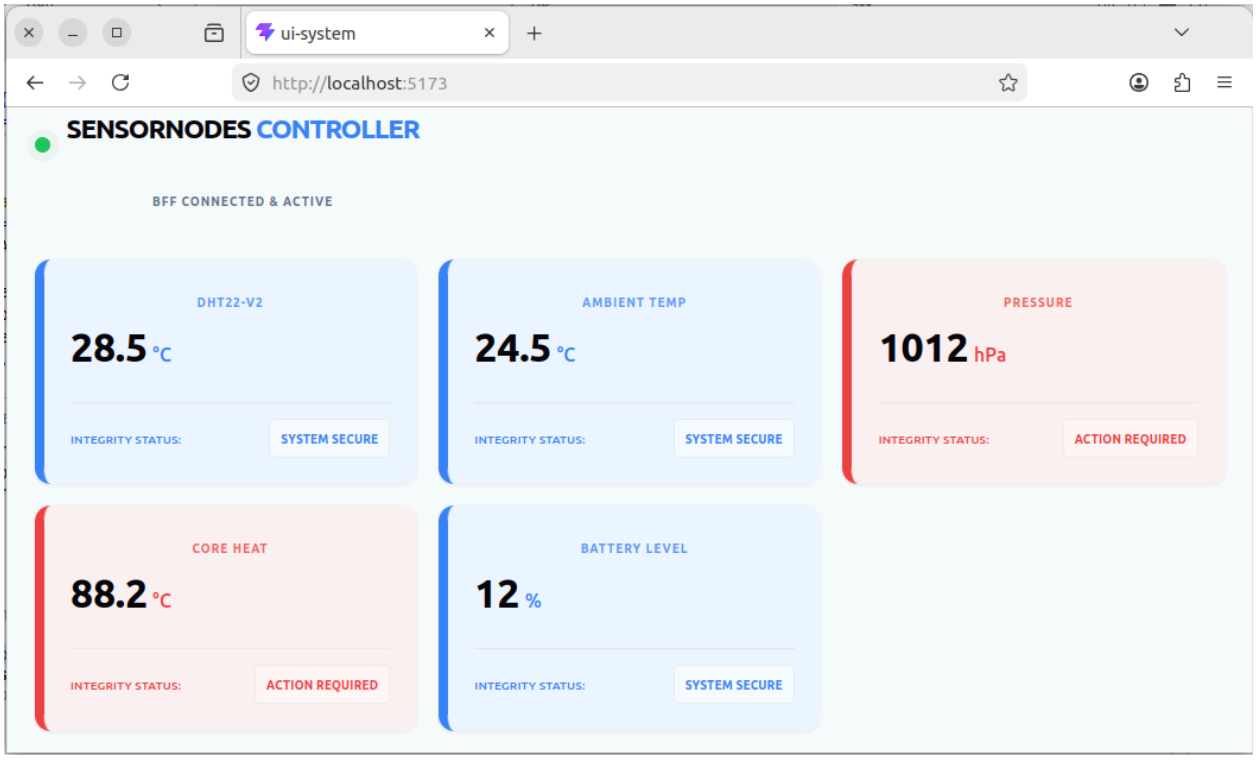
Jalankan **Server Web Client (React + Axios)** → **ui-system** dengan perintah berikut pada terminal proyek web client:

```
~/rlp/ui-system$ npm run dev
```

```
(base) very@very-Inspiron-3443:~/rlp/ui-system$ npm run dev

> ui-system@0.0.0 dev
> vite


  VITE v8.0.8  ready in 741 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Akses web client menggunakan browser pada alamat:

```
http://localhost:5173/
```



D. Integrasi API pada Mobile Client (Flutter + Http) → iot_mobile_app

Ubah logika pada `SensorProvider` di proyek Flutter (Modul 5) untuk melakukan panggilan HTTP asinkron. Gunakan alamat IP **`10.0.2.2`** agar Emulator Android dapat menjembatani koneksi ke port `localhost:3000` pada PC host Anda.

Buat file **api_service.dart** berikut pada folder **lib/services**

```
// lib/services/api_service.dart
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'package:flutter/foundation.dart'; // Untuk kCheckGui

class ApiService {
  // Penentuan URL secara dinamis agar berjalan di Emulator maupun Web
  String get baseUrl {
    if (kIsWeb) {
      return "http://localhost:3000"; // Host untuk browser web
    } else {
      // 10.0.2.2 adalah jembatan emulator Android ke localhost PC host
      return "http://10.0.2.2:3000";
    }
  }

  Future<List<Map<String, dynamic>>> fetchMobileDashboard() async {
    final response = await http.get(Uri.parse('$baseUrl/mobile/dashboard'));

    if (response.statusCode == 200) {
      final decoded = json.decode(response.body);
      if (decoded is List) {
        return decoded.cast<Map<String, dynamic>>();
      }
      throw Exception('Format data BFF tidak sesuai');
    } else {
      throw Exception('Gagal memuat data dari BFF NestJS');
    }
  }
}
```

Perbarui berkas `lib/providers/sensor_provider.dart` agar terhubung dengan layanan `ApiService` di atas:

```
// lib/providers/sensor_provider.dart
import 'package:flutter/material.dart';
import '../services/api_service.dart';

class Sensor {
  final String id;
  final String name;
  final double value;
  final String unit;
  final String state; // OK, WARNING, DANGER

  Sensor({
    required this.id,
    required this.name,
    required this.value,
    required this.unit,
    required this.state,
  });
}

class SensorProvider with ChangeNotifier {
  final ApiService _apiService = ApiService();
  List<Sensor> _sensors = [];
  bool _isLoading = false;
  String? _errorMessage;

  List<Sensor> get sensors => _sensors;
  bool get isLoading => _isLoading;
  String? get errorMessage => _errorMessage;

  Future<void> fetchAndSyncData() async {
    _isLoading = true;
    _errorMessage = null;
    notifyListeners();

    try {
      final data = await _apiService.fetchMobileDashboard();

      // Mengubah JSON BFF Mobile ([{ "v": 28.5, "u": "°C", "st": "OK" }]) menjadi objek
Dart.
```

```
      _sensors = data.map((sensor) {
        return Sensor(
          id: sensor['id'] as String,
          name: sensor['id'] as String,
          value: (sensor['v'] as num).toDouble(),
          unit: sensor['u'] as String,
          state: sensor['st'] == 'WARN' ? 'WARNING' : sensor['st'] as String,
        );
      }).toList();
    } catch (error) {
      _errorMessage = error.toString();
    } finally {
      _isLoading = false;
      notifyListeners();
    }
  }
}
```

Ubah tombol pemicu simulasi pada berkas `lib/main.dart` menjadi tombol **"REFRESH DATA"** untuk menarik data riil dari backend secara asinkron:

Potongan Aksi Button

```
// lib/main.dart (Potongan Aksi Button)
ElevatedButton.icon(
  onPressed: () {
    context.read<SensorProvider>().fetchAndSyncData();
  },
  icon: const Icon(Icons.refresh),
  label: const Text("REFRESH DARI API"),
)
```

Kode lib/main.dart lengkap

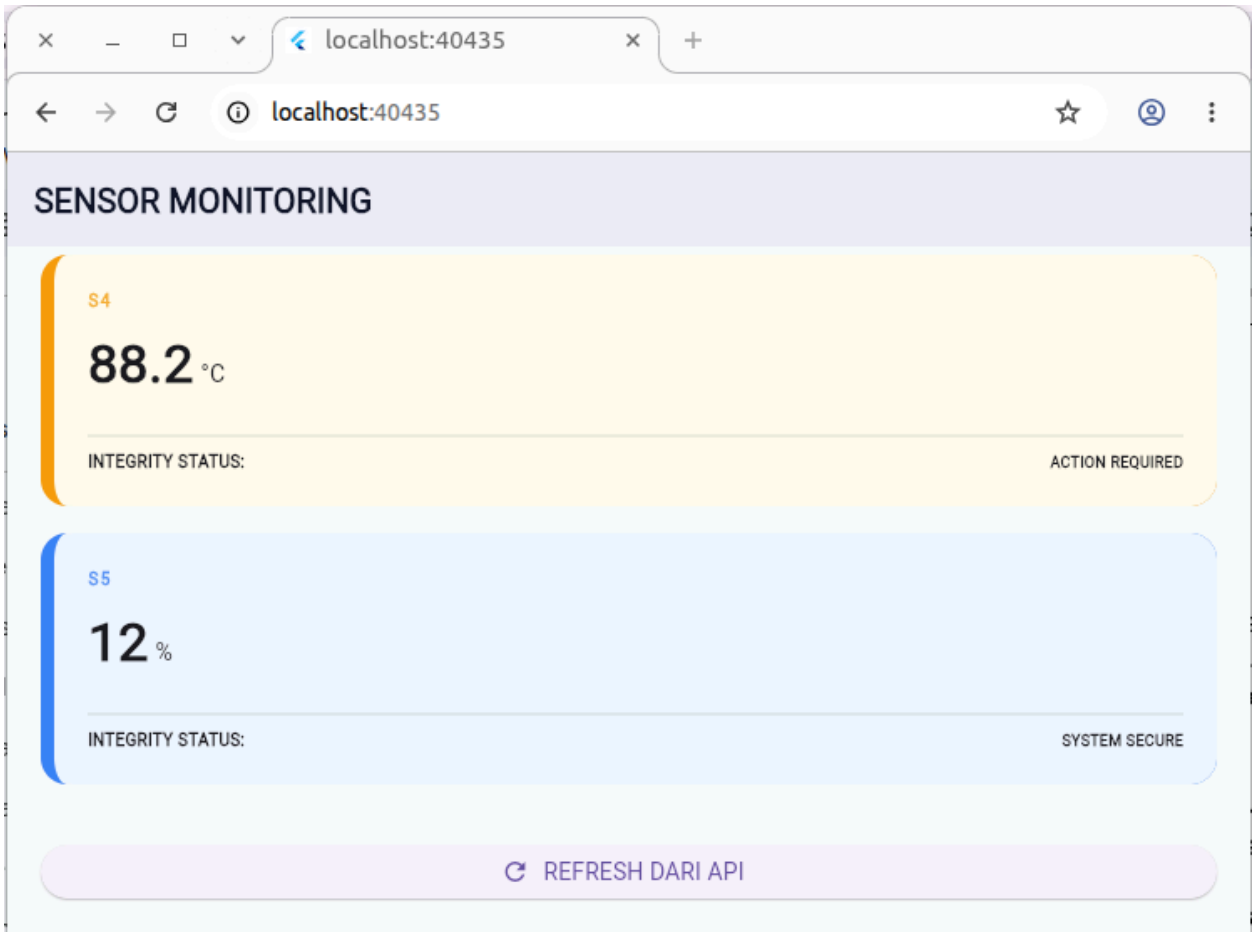
```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'providers/sensor_provider.dart';
import 'shared/theme.dart';
```

```
import 'widgets/status_card.dart';

void main() {
  runApp(
    ChangeNotifierProvider(
      create: (context) => SensorProvider(),
      child: const MyApp(),
    ),
  );
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: AppTheme.lightTheme,
      home: const DashboardScreen(),
    );
  }
}

class DashboardScreen extends StatefulWidget {
  const DashboardScreen({super.key});

  @override
  State<DashboardScreen> createState() => _DashboardScreenState();
}

class _DashboardScreenState extends State<DashboardScreen> {
  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      context.read<SensorProvider>().fetchAndSyncData();
    });
  }

  @override
  Widget build(BuildContext context) {
    final sensorProvider = context.watch<SensorProvider>();

    return Scaffold(
      appBar: AppBar(
```

```
    title: const Text("SENSOR MONITORING"),
  ),
  body: ListView(
    padding: const EdgeInsets.all(20),
    children: [
      if (sensorProvider.isLoading)
        const Padding(
          padding: EdgeInsets.only(bottom: 16),
          child: LinearProgressIndicator(),
        ),
      if (sensorProvider.errorMessage != null)
        Padding(
          padding: const EdgeInsets.only(bottom: 16),
          child: Text(
            sensorProvider.errorMessage!,
            style: const TextStyle(
              color: Colors.red,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ...sensorProvider.sensors.map((s) => Padding(
        padding: const EdgeInsets.only(bottom: 16),
        child: StatusCard(
          label: s.name,
          value: s.value,
          unit: s.unit,
          status: s.state,
        ),
      )),
      const SizedBox(height: 20),
      /*
      ElevatedButton(
        onPressed: () {
          context.read<SensorProvider>().updateTemperature('S1', 45.0);
        },
        child: const Text("SIMULASI OVERHEAT (S1)"),
      )
    */
      ElevatedButton.icon(
        onPressed: sensorProvider.isLoading
            ? null
            : () {
                context.read<SensorProvider>().fetchAndSyncData();
              },
        icon: const Icon(Icons.refresh),
```

```
            label: const Text("REFRESH DARI API"),
          )
        ],
      ),
    );
  }
}
```

Tampilan Flutter - Chrome



4. Penjelasan Logika & Keamanan

1. **Aturan CORS (Cross-Origin Resource Sharing):** Mekanisme keamanan bawaan browser yang membatasi request HTTP lintas domain. Dengan

memanggil `enableCors()` di NestJS, kita memberi tahu server untuk memercayai klien luar yang mengakses rute API.
2. **Pola Pemanggilan Asinkron:** Penggunaan `async/await` di React dan Dart menjamin bahwa aplikasi tetap responsif (*non-blocking*). Saat klien menunggu respons jaringan dari BFF, UI tetap dapat berinteraksi (seperti animasi memutar atau memuat halaman).
3. **Emulator Networking Bridge:** Emulator Android berjalan pada mesin virtual terisolasi dengan jaringan virtualnya sendiri. Menggunakan `localhost` atau `127.0.0.1` dari dalam emulator akan merujuk pada emulator itu sendiri, bukan PC Anda. Alamat `10.0.2.2` adalah alamat IP khusus yang dipetakan untuk mengakses `127.0.0.1` komputer host.

5. Tugas Praktikum: "Integrasi Kendali Lampu Pintar"

Skenario: Selesaikan integrasi penuh fitur kendali lampu (*Smart Home Controller*) dari ujung ke ujung.

Instruksi:

1. **Backend:** Pastikan rute `GET http://localhost:3000/light-control/web` and `GET http://localhost:3000/light-control/mobile` (Modul 2) berjalan normal.
2. **Web Client:** Gunakan Axios untuk membaca status `isOn` dan `brightness` dari server, lalu ubah tampilan dashboard React secara dinamis.
3. **Mobile Client:** Tambahkan fungsi panggilan HTTP pada berkas `ApiService` mobile untuk menarik status lampu, simpan ke dalam *State*, dan sinkronisasikan ke dalam komponen UI *Switch* di Flutter.

Hasil yang Harus Dikumpulkan:

1. Tangkapan layar (*screenshot*) terminal NestJS yang menampilkan catatan riwayat (*network logs*) saat menerima panggilan API dari aplikasi Web dan Mobile.
2. Tangkapan layar yang menunjukkan bahwa aplikasi Web dan Mobile Anda memuat dan menyajikan data dari server secara bersamaan dengan nilai yang sinkron.