Modul 7: Autentikasi, Otorisasi, & Keamanan Dasar

Mata Kuliah: Rekayasa Perangkat Lunak Lintas Platform / Integrasi Sistem

Tujuan:

1. Mahasiswa mampu mengimplementasikan sistem login yang aman menggunakan **JSON Web Token (JWT)**.
2. Mahasiswa mampu menerapkan pengamanan *endpoint* API (**Otorisasi**) di sisi backend.
3. Mahasiswa mampu mengelola penyimpanan kredensial secara aman di sisi klien (Web & Mobile).

1. Pendahuluan

Keamanan dalam sistem lintas platform (IoT, Web, Mobile) bukan sekadar opsi, melainkan kebutuhan. Tanpa pengamanan, data sensor atau kontrol perangkat dapat dieksploitasi. Modul ini memperkenalkan **JWT (JSON Web Token)** sebagai standar autentikasi *stateless*.

Logika Utama:

- **Authentication:** Proses verifikasi identitas (Siapa Anda?).
- **Authorization:** Proses pengecekan hak akses (Apa yang boleh Anda lakukan?).
- **Statelessness:** Server tidak menyimpan sesi; klien harus mengirimkan token pada setiap permintaan data.

2. Persiapan Lingkungan

A. Backend: NestJS (Modul 2) - Install: `npm install @nestjs/jwt @nestjs/passport passport passport-jwt`.

Pada terminal proyek `modular-system` Anda, instal paket-paket standar industri untuk autentikasi menggunakan Passport.js dan JWT:

- **Instalasi Dependency:**

Bash

```
npm install @nestjs/jwt @nestjs/passport passport passport-jwt
```

- **Type Definitions:** (Untuk mendukung pengembangan dengan TypeScript):

Bash

```
npm install @types/passport-jwt -D
```

- **Verifikasi:** Pastikan baris tersebut muncul di berkas `package.json` Anda di bawah bagian `dependencies`.

```
"dependencies": {
  "@nestjs/common": "^11.0.1",
  "@nestjs/core": "^11.0.1",
  "@nestjs/jwt": "^11.0.2",
  "@nestjs/passport": "^11.0.5",
  "@nestjs/platform-express": "^11.0.1",
  "passport": "^0.7.0",
  "passport-jwt": "^4.0.1",
  "reflect-metadata": "^0.2.2",
  "rxjs": "^7.8.1"
},
```

B. Frontend (Web): React (Modul 3).

Pastikan proyek React Anda sudah siap untuk menangani pengiriman *header* HTTP secara global.

- **Pustaka HTTP:** Jika belum dilakukan di Modul 6, pastikan Axios sudah terpasang:

Bash

```
npm install axios
```

- **Kebutuhan UI:** Siapkan folder `src/context/` atau `src/store/` untuk menyimpan status login (misal: `AuthContext.jsx`) agar token dapat diakses oleh semua komponen dashboard.

C. Frontend (Mobile): Flutter (Modul 5) - Tambahkan: `flutter_secure_storage`.

Penyimpanan biasa di mobile (*Shared Preferences*) tidak cukup aman untuk menyimpan kunci akses (token). Kita akan menggunakan penyimpanan terenkripsi.

- **Tambah Dependensi:** Buka terminal di proyek Flutter Anda dan jalankan:

Bash

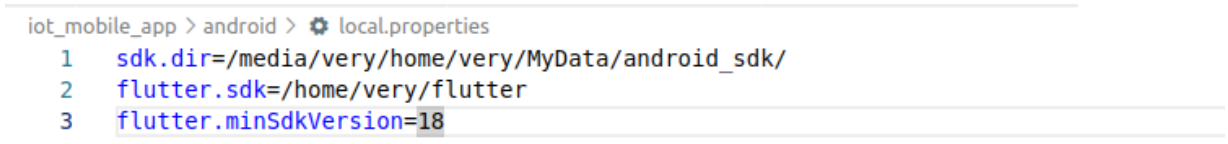
```
flutter pub add flutter_secure_storage
```

- **Konfigurasi Platform (Khusus Android):** Pastikan `minSdkVersion` di file `android/app/build.gradle` adalah **18** atau lebih tinggi agar fitur enkripsi berjalan lancar.

Periksa local.properties

Pada proyek Flutter modern (versi 2.8 ke atas), minSdkVersion biasanya dikelola sebagai sebuah variabel.

- Buka folder utama (root) proyek Anda.
- Buka file android/local.properties.
- Cari baris yang dimulai dengan flutter.minSdkVersion. Contoh: flutter.minSdkVersion=21.
- Jika variabel tidak ada: Anda bisa langsung menambahkan baris baru di bagian paling bawah file tersebut, misalnya: flutter.minSdkVersion=21 (sesuaikan angka dengan versi minimum API Android yang Anda butuhkan).



- **Update Pubspec:** Jalankan `flutter pub get` untuk memastikan pustaka terdaftar secara permanen di `pubspec.yaml`.

```
(base) very@very-Inspiron-3443:~/rlp/iot_mobile_app$ flutter pub get
Resolving dependencies...
Downloading packages...
  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)
  vector_math 2.2.0 (2.3.0 available)
Got dependencies!
4 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
```

- **Alat Bantu:** Postman untuk simulasi login.

3. Implementasi Kode Secara Rinci

A. Sisi Backend (NestJS - Auth Service)

Kita akan membuat modul autentikasi yang mengeluarkan token jika kredensial valid.

- Jwt Strategy

Buat file baru jwt.strategy.ts di dalam folder autentikasi `src/auth/`. Kelas strategi standar harus memperluas fungsi PassportStrategy dan diberi dekorasi @Injectable().

Buka file tersebut dan masukkan kode berikut.

```
// src/auth/jwt.strategy.ts
import { Injectable } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: process.env.JWT_SECRET ?? 'development-secret',
    });
  }

  validate(payload: { sub: string; username: string; role: string }) {
    return {
      userId: payload.sub,
```

```
      username: payload.username,
      role: payload.role,
    };
  }
}
```

- Service Auth

Buka terminal di direktori proyek `modular-system` dan jalankan perintah:

Bash

```
nest generate service auth
```

Perintah ini akan secara otomatis membuat file `auth.service.ts` pada folder `src/auth/`



Buka file tersebut dan masukkan kode berikut. Kode ini berfungsi sebagai penyedia layanan untuk memvalidasi identitas pengguna dan menerbitkan token JWT.

```
// src/auth/auth.service.ts
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';

@Injectable()
export class AuthService {
  constructor(private jwtService: JwtService) {}

  login(username: string, pass: string) {
    // Simulasi pengecekan database
    if (username === 'admin' && pass === 'password123') {
      const payload = { username: username, sub: 'user-001', role: 'ADMIN' };
      return {
        access_token: this.jwtService.sign(payload),
      };
    }
```

```
    throw new UnauthorizedException('Username atau Password salah');
  }
}
```

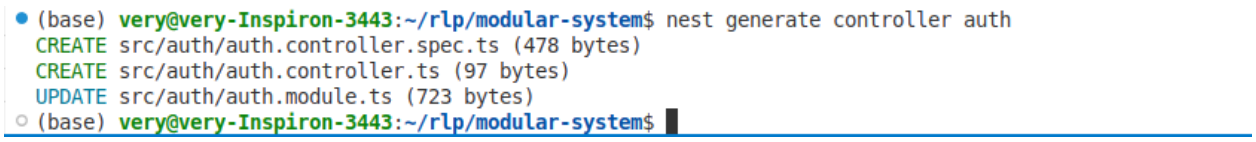
- Controller Auth

Buka terminal di direktori proyek `modular-system` dan jalankan perintah:

Bash

```
nest generate controller auth
```

Perintah ini akan secara otomatis membuat file `auth.controller.ts` pada folder `src/auth/`



Buka file tersebut dan masukkan kode berikut.

```
// src/auth/auth.controller.ts
import { Body, Controller, Post } from '@nestjs/common';
import { AuthService } from './auth.service';

type LoginBody = {
  username: string;
  password: string;
};

@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('login')
  login(@Body() body: LoginBody) {
    return this.authService.login(body.username, body.password);
  }
}
```

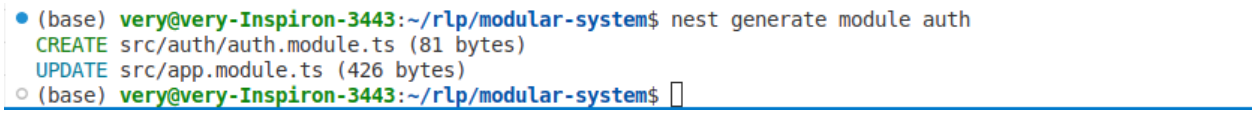
- Module Auth

Buka terminal di direktori proyek `modular-system` dan jalankan perintah:

Bash

```
nest generate module auth
```

Perintah ini akan secara otomatis membuat file `auth.module.ts` pada folder `src/auth/`



Buka file tersebut dan masukkan kode berikut.

```
// src/auth/auth.module.ts
import { Module } from '@nestjs/common';
import { JwtModule } from '@nestjs/jwt';
import { PassportModule } from '@nestjs/passport';
import { JwtStrategy } from './jwt.strategy';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';

@Module({
  imports: [
    PassportModule.register({ defaultStrategy: 'jwt' }),
    JwtModule.register({
      secret: process.env.JWT_SECRET ?? 'development-secret',
      signOptions: { expiresIn: '1h' },
    }),
  ],
  controllers: [AuthController],
  providers: [AuthService, JwtStrategy],
  exports: [AuthService, PassportModule],
})
export class AuthModule {}
```

B. Mengamankan Endpoint (Auth Guard))

Menerapkan "Satpam" pada *controller* Sensor agar data hanya bisa diakses oleh pengguna yang memiliki token.

Langkah-langkah Penyesuaian:

1. Buka kembali file src/sensor/infrastructure/sensor.controller.ts yang Anda buat pada Modul 2.
2. Tambahkan dekorator @UseGuards(AuthGuard('jwt')) di atas kelas SensorController agar seluruh *endpoint* di dalamnya terlindungi secara otomatis.

```
// src/sensor/infrastructure/sensor.controller.ts
import { Controller, Get, UseGuards } from '@nestjs/common'; // Tambahkan
UseGuards
import { AuthGuard } from '@nestjs/passport'; // Tambahkan AuthGuard
import { SensorService } from '../../sensor.service';

@Controller('sensor')
@UseGuards(AuthGuard('jwt')) // Menerapkan keamanan untuk seluruh endpoint di
kelas ini
export class SensorController {
  constructor(private readonly sensorService: SensorService) {}

  // Endpoint khusus untuk Mobile (Kini Terproteksi)
  @Get('mobile/dashboard')
  getMobileView() {
    return this.sensorService.getSummaryForMobile();
  }

  // Endpoint khusus untuk Web (Kini Terproteksi)
  @Get('web/analytics')
  getWebView() {
    return this.sensorService.getFullDetailsForWeb();
  }
}
```

C. Verifikasi Keamanan Backend dengan Postman:

Mahasiswa wajib melakukan pengujian berikut untuk memastikan sinkronisasi kode berhasil:

Jalankan server backend terlebih dahulu dengan perintah berikut pada terminal di dalam direktori proyek:

Bash

```
npm run start:dev
```

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

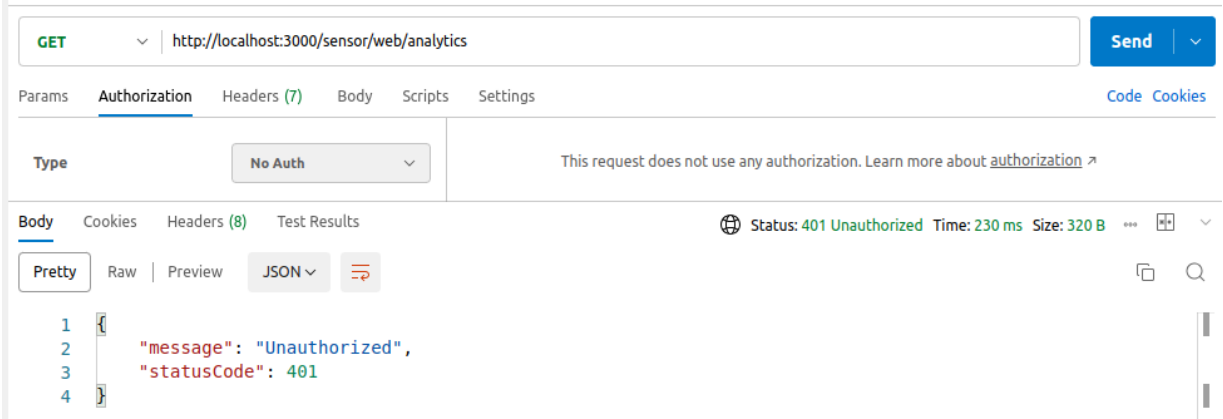
Verifikasi Sinkronisasi & Keamanan dengan Postman

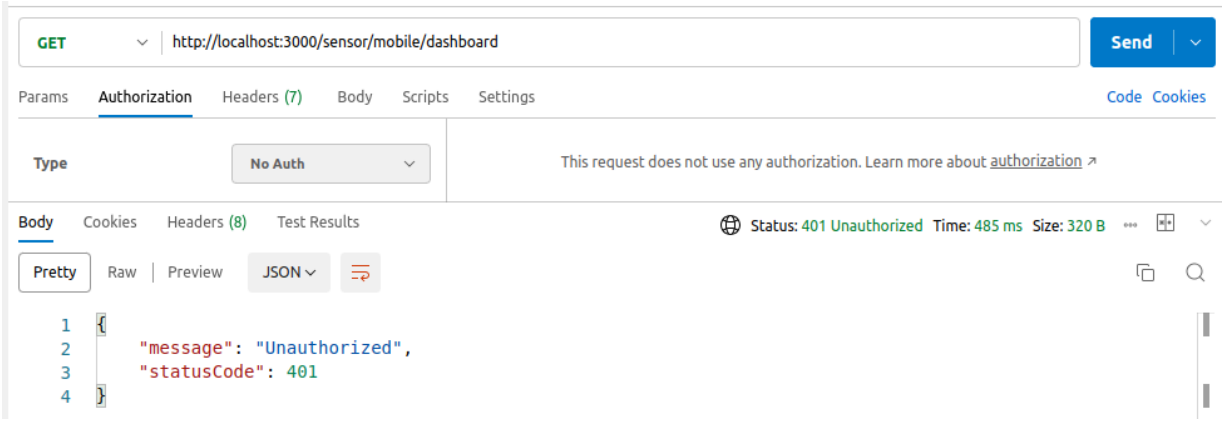
Setelah mengimplementasikan *Auth Guard* pada SensorController , mahasiswa wajib melakukan pengujian berikut untuk memastikan bahwa "Satpam" digital telah bekerja dengan benar dan hanya mengizinkan akses bagi pengguna yang terautentikasi.

Skenario 1: Pengujian Akses Tanpa Token (Unauthorized)

Tujuan: Memastikan *endpoint* terproteksi menolak permintaan ilegal.

1. Buka aplikasi **Postman**.
2. Buat *Request* baru dengan metode **GET**.
3. Masukkan URL: http://localhost:3000/sensor/web/analytics.
4. Pastikan pada tab *Authorization*, tipenya adalah **No Auth**.
5. Klik **Send**.
6. **Hasil yang Diharapkan:** Anda harus menerima status **401 Unauthorized** dengan pesan error dari NestJS. Ini membuktikan bahwa *Guard* berhasil mencegat akses publik.





Skenario 2: Proses Login & Mendapatkan Kredensial

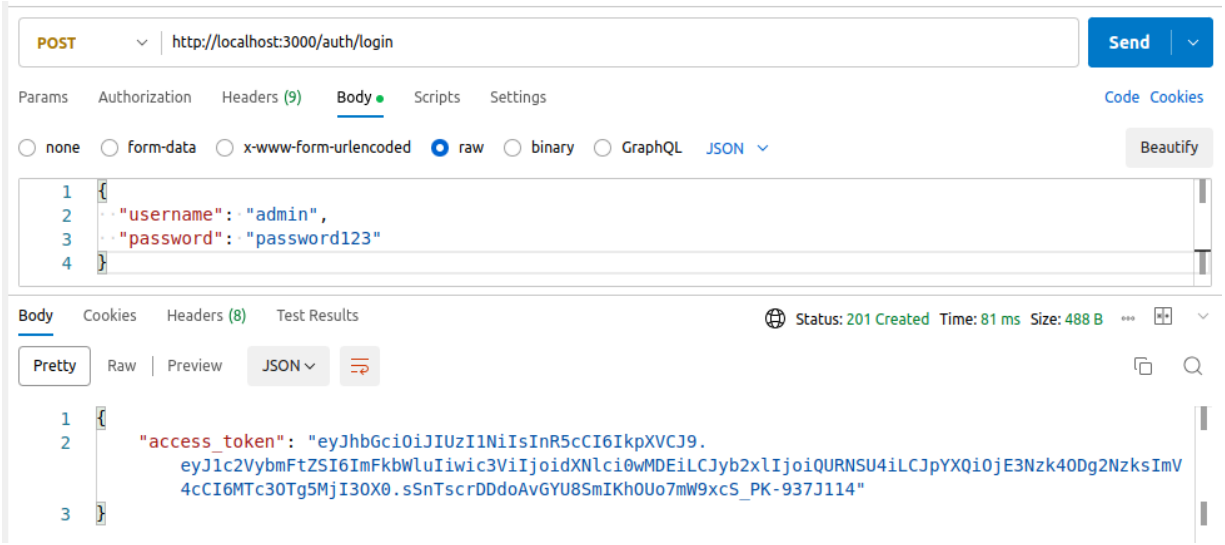
Tujuan: Mendapatkan tiket digital (JWT) melalui *endpoint* autentikasi.

1. Buat *Request* baru dengan metode **POST**.
2. Masukkan URL: http://localhost:3000/auth/login.
3. Pilih tab **Body**, pilih **raw**, dan ubah format menjadi **JSON**. Masukkan kredensial berikut:

JSON

```
{
  "username": "admin",
  "password": "password123"
}
```

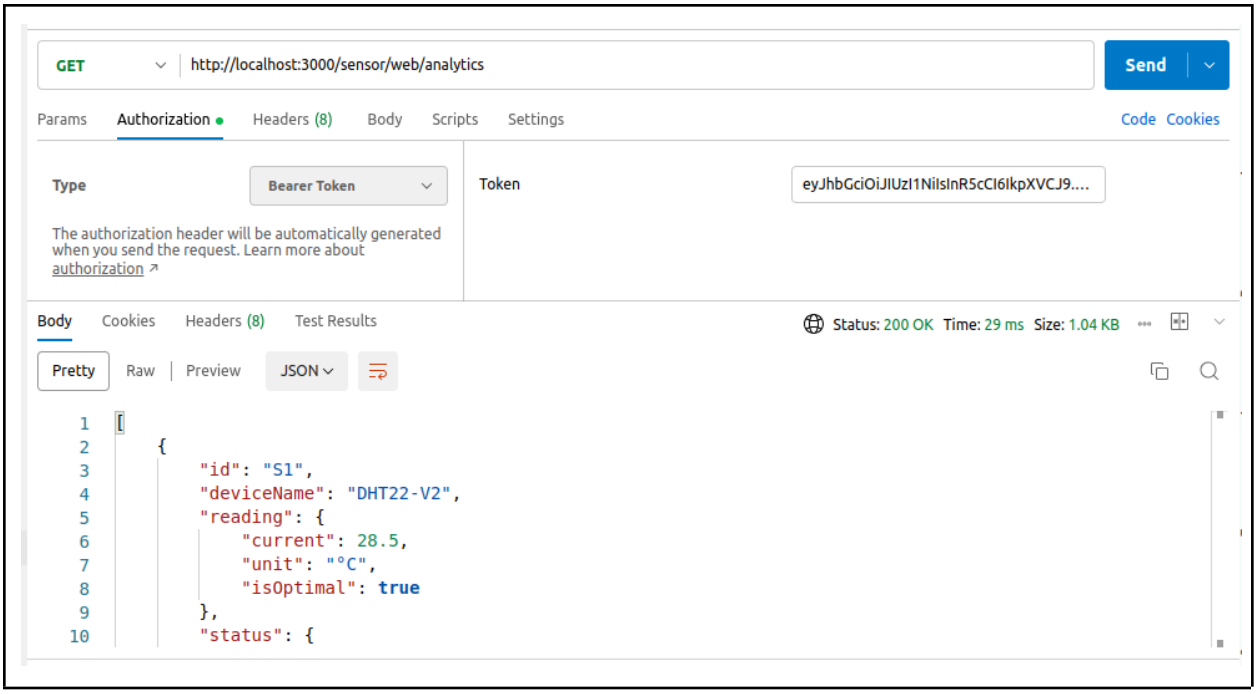
4. Klik **Send**.
5. **Hasil yang Diharapkan:** Server merespons dengan status **201 Created** dan memberikan objek access_token.
6. **Salin** seluruh kode token yang muncul (tanpa tanda kutip).



Skenario 3: Pengujian Akses Dengan Token (Authorized)

Tujuan: Memverifikasi bahwa token yang valid dapat membuka akses ke data sensor.

1. Kembali ke tab *Request* **GET** http://localhost:3000/sensor/web/analytics (dari Skenario 1).
2. Buka tab **Authorization**.
3. Pada kolom *Type*, pilih **Bearer Token**.
4. **Tempelkan (Paste)** kode token yang telah Anda salin sebelumnya ke kolom **Token**.
5. Klik **Send**.
6. **Hasil yang Diharapkan:** Anda harus menerima status **200 OK** dan data sensor (seperti deviceName, currentReading, dsb.) akan muncul secara detail.



D. Sisi Klien (Mobile - Secure Storage)

Berbeda dengan Web, pada Mobile, penyimpanan data sensitif seperti JWT tidak boleh dilakukan menggunakan Shared Preferences biasa karena data tersebut disimpan dalam bentuk teks polos yang mudah dicuri. Kita akan menggunakan Flutter Secure Storage yang memanfaatkan Keystore (Android) dan Keychain (iOS) untuk mengenkripsi token di tingkat perangkat keras.

1. Penyiapan Layanan Autentikasi (AuthService)

Buat file baru lib/services/auth_service.dart untuk mengelola proses simpan, ambil, dan hapus token.

Dart

```
// lib/services/auth_service.dart
import 'package:flutter_secure_storage/flutter_secure_storage.dart';

class AuthService {
  // Inisialisasi penyimpanan aman
  final _storage = const FlutterSecureStorage();

  // Menyimpan Token setelah login berhasil
```

```
  Future<void> saveToken(String token) async {
    await _storage.write(key: 'jwt_token', value: token);
  }

  // Mengambil Token untuk disertakan dalam Header API
  Future<String?> getToken() async {
    return await _storage.read(key: 'jwt_token');
  }

  // Menghapus Token saat Logout
  Future<void> logout() async {
    await _storage.delete(key: 'jwt_token');
  }
}
```

2. Integrasi Token pada ApiService

Modifikasi file lib/services/api_service.dart (dari Modul 6) agar setiap permintaan data sensor menyertakan token di dalam *Header Authorization*.

Dart

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

class ApiService {
  final AuthService _authService = AuthService();

  // 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";
    }
  }

  // Fungsi Login dipindahkan ke AuthService atau tetap di sini sebagai aksi POST
  Future<void> login(String username, String password) async {
```

```
  final response = await http.post(
    Uri.parse('$baseUrl/auth/login'),
    headers: {'Content-Type': 'application/json'},
    body: json.encode({
      'username': 'admin',
      'password': 'password123',
    }),
  );

  if (response.statusCode == 201 || response.statusCode == 200) {
    final decoded = json.decode(response.body);
    String token = decoded['access_token'];
    // Simpan ke Secure Storage sesuai standar Modul 7
    await _authService.saveToken(token);
  } else {
    throw Exception('Username atau Password salah');
  }
}

// Perbaikan Fetch Data dengan Header Authorization
Future<List<Map<String, dynamic>>> fetchMobileDashboard() async {
  // 1. Ambil token dari Secure Storage
  String? token = await _authService.getToken();

  // 2. Kirim request dengan Header Bearer Token
  final response = await http.get(
    Uri.parse('$baseUrl/sensor/mobile/dashboard'),
    headers: {
      'Content-Type': 'application/json',
      'Authorization': 'Bearer $token',
    },
  );

  if (response.statusCode == 200) {
    final decoded = json.decode(response.body);

    if (decoded is List) {
      return decoded.cast<Map<String, dynamic>>();
    }

    if (decoded is Map<String, dynamic>) {
      return [decoded];
    }

    throw Exception('Format data BFF tidak sesuai');
  } else if (response.statusCode == 401) {
```

```
      // Jika 401 (Unauthorized), paksa logout dan hapus token
      await _authService.logout();
      throw Exception('Sesi berakhir. Silakan login kembali.');
    } else {
      throw Exception('Gagal memuat data dari BFF NestJS: ${response.statusCode}');
    }
  }
}
```

3. Penyesuaian pada SensorProvider

Tambahkan method `login()` di `SensorProvider` sebagai wrapper ke `ApiService.login(...)`.

Di file sensor_provider.dart, di dalam class `SensorProvider`, tambahkan ini:

Dart

```
Future<void> login() async {
  await _apiService.login('admin', 'password123');
}
```

Sehingga kode lengkap jadi seperti ini:

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,
```

```
  });

  factory Sensor.fromJson(Map<String, dynamic> json) {
    final id = json['id']?.toString() ?? '-';
    final rawState = json['st']?.toString() ?? 'UNKNOWN';

    return Sensor(
      id: id,
      name: json['name']?.toString() ?? id,
      value: (json['v'] as num?)?.toDouble() ?? 0,
      unit: json['u']?.toString() ?? '',
      state: rawState == 'WARN' ? 'WARNING' : rawState,
    );
  }
}

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> login() async {
    await _apiService.login('admin', 'password123');
  }

  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.fromJson).toList();
    } catch (error) {
      _errorMessage = error.toString();
    } finally {
      _isLoading = false;
      notifyListeners();
```

```
    }
  }
}
```

4. Penyesuaian pada Main

Lalu di file main.dart, ubah bagian `initState()` menjadi login dulu, baru fetch data:

Dart

```
@override
void initState() {
  super.initState();
  WidgetsBinding.instance.addPostFrameCallback((_) async {
    final provider = context.read<SensorProvider>();

    await provider.login();
    await provider.fetchAndSyncData();
  });
}
```

Sehingga kode lengkap jadi seperti ini:
Dart

```
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();
    });
    */
    WidgetsBinding.instance.addPostFrameCallback((_) async {
      final provider = context.read<SensorProvider>();
      await provider.login();
      await provider.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"),
        )
      ],
    ),
  );
 }
}
```

Alurnya menjadi:

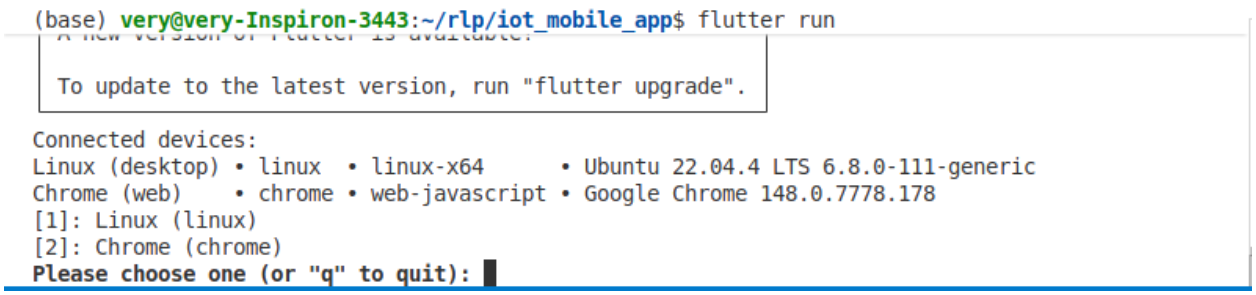
1. Flutter web dibuka.

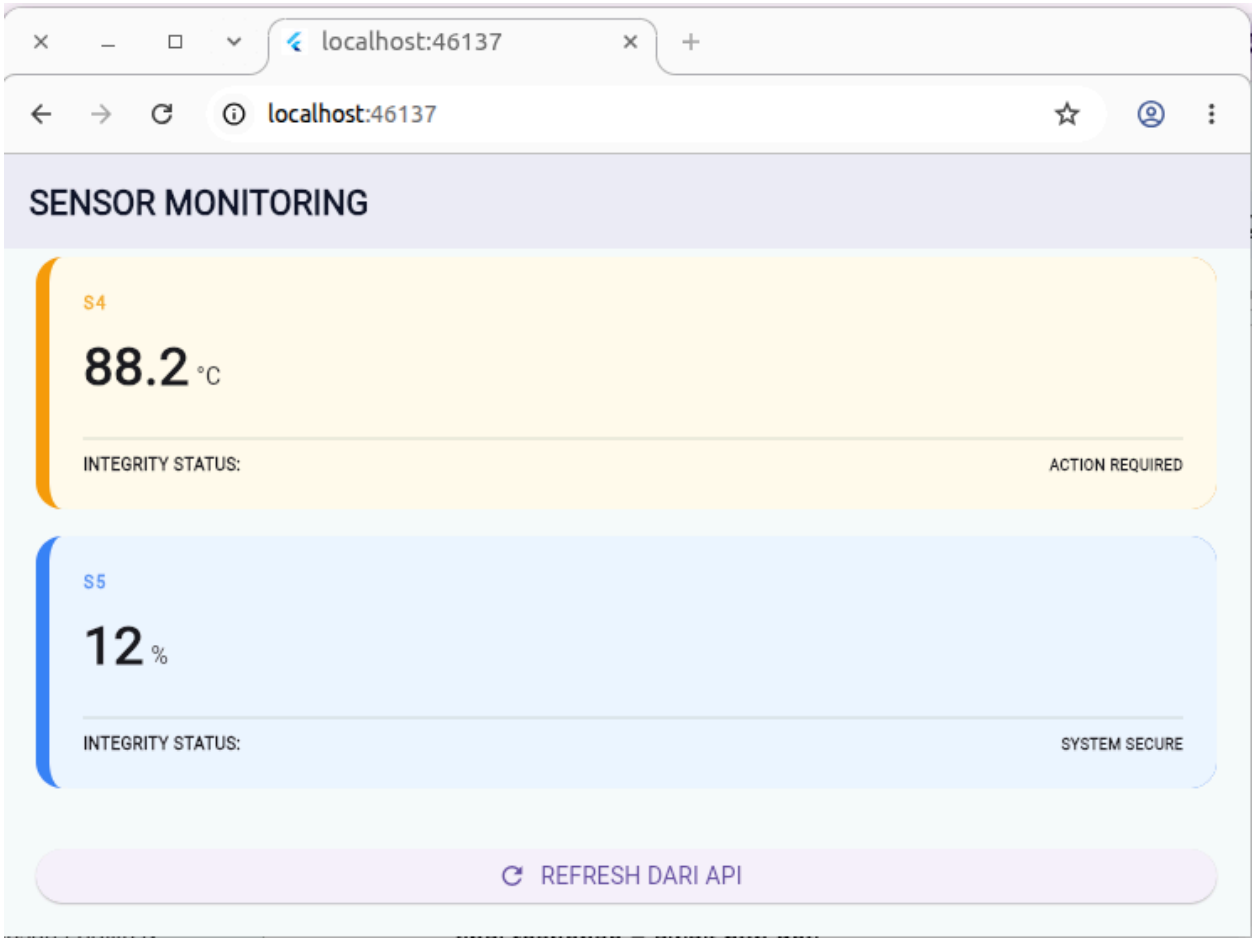
2. App login otomatis ke `/auth/login`.
3. Token JWT disimpan lewat `AuthService.saveToken()`.
4. Dashboard memanggil `/sensor/mobile/dashboard`.
5. Header `Authorization: Bearer <token>` valid.
6. Pesan `Sesi berakhir. Silakan login kembali.` tidak muncul lagi, selama backend berjalan dan JWT valid.

Jalankan Aplikasi Klien

Gunakan perintah standar untuk menjalankan aplikasi dalam mode **Debug,** pilih device yang akan digunakan:

```
flutter run
```





4. Penjelasan Logika Keamanan

1. **JWT Structure:** Terdiri dari Header, Payload, dan Signature. Mahasiswa harus paham bahwa Payload dapat dibaca siapa saja, sehingga **jangan pernah menyimpan password di dalam JWT**.
2. **Bearer Token:** Klien mengirimkan token melalui HTTP Header: `Authorization: Bearer <token>`.
3. **Storage Logic:** * **Web:** Gunakan `LocalStorage` atau `HttpOnly Cookies`.
    - **Mobile:** Gunakan `Secure Storage` (Keychain di iOS / SharedPrefs terenkripsi di Android) untuk mencegah pencurian data jika perangkat di-*root*.
4. `@UseGuards(AuthGuard('jwt'))`: Dekorator ini berfungsi untuk mengecek apakah permintaan (*request*) menyertakan **Bearer Token** yang valid di bagian *Header*.
   DOCX

5. **Otorisasi Otomatis**: Jika token tidak disertakan atau sudah kedaluwarsa, NestJS secara otomatis akan menolak permintaan dan mengirimkan pesan error **401 Unauthorized** sebelum fungsi di dalam *controller* dijalankan.
   DOCX
6. **Skalabilitas**: Dengan meletakkan *guard* di tingkat kelas (`@Controller`), Anda tidak perlu menulis ulang kode keamanan pada setiap fungsi `getMobileView` atau `getWebView` satu per satu.
   DOCX

5. Tugas Praktikum (Latihan)

Skenario: Implementasikan sistem login pada aplikasi Smart Home Anda.

1. **Backend:** Buatlah endpoint `POST /auth/login`.
2. **Otorisasi:** Proteksi endpoint `GET /sensor/web/dashboard` (Modul 2) sehingga hanya bisa diakses jika menyertakan JWT.
3. **Analisis:** Gunakan Postman untuk:
   - Login dan dapatkan token.
   - Akses data sensor tanpa token (Pastikan muncul error `401 Unauthorized`).
   - Akses data sensor dengan token di Header.
4. **Deep Dive:** Buka jwt.io dan tempelkan token Anda. Identifikasi isi *Payload*-nya.

Hasil yang Harus Dikumpulkan:

- Screenshot respons Postman (Sukses & Error 401).
- Penjelasan singkat mengapa `Secret Key` JWT harus disimpan di `.env` (Environment Variable).