Modul 2: Arsitektur Modular (Clean Layering & BFF)

Mata Kuliah: Rekayasa Perangkat Lunak Lintas Platform / Integrasi Sistem

Tujuan:

1. Mahasiswa mampu merancang dan mengimplementasikan struktur proyek yang modular menggunakan prinsip *Clean Architecture*.
2. Mahasiswa mampu memahami dan menerapkan pola *Backend for Frontend* (BFF) untuk efisiensi komunikasi data lintas platform.
3. Mahasiswa mampu membedakan kebutuhan format data antara platform Web dan Mobile melalui logika *Services*.

1. Pendahuluan

Dalam pengembangan sistem terintegrasi yang melibatkan Web, Mobile, dan IoT, penggunaan arsitektur yang buruk seringkali menyebabkan kode menjadi sulit dikelola atau "spageti". Modul ini memperkenalkan dua konsep utama:

- **Clean Architecture:** Metode pengorganisasian kode di mana logika bisnis inti berada di pusat dan tidak bergantung pada alat eksternal seperti framework atau database.
- **Backend for Frontend (BFF):** Pola penyediaan lapisan API khusus yang menyesuaikan kebutuhan data perangkat tertentu. Misalnya, mengirimkan data yang lebih ringkas untuk perangkat mobile guna menghemat baterai dan bandwidth.

2. Persiapan Lingkungan Kerja

Pastikan perangkat lunak berikut telah terpasang di OS Anda (Windows atau Linux):

- **Runtime:** Node.js (Versi LTS).
- **IDE:** Visual Studio Code.
- **Framework:** NestJS CLI (untuk mempermudah manajemen modul).
- **Skill Dasar:** Pemahaman TypeScript dan Pemrograman Berorientasi Objek (OOP).

3. Implementasi Kode & Struktur Modular

A. Inisialisasi Proyek

Gunakan terminal untuk membuat boilerplate proyek baru:

Bash
Install CLI secara global

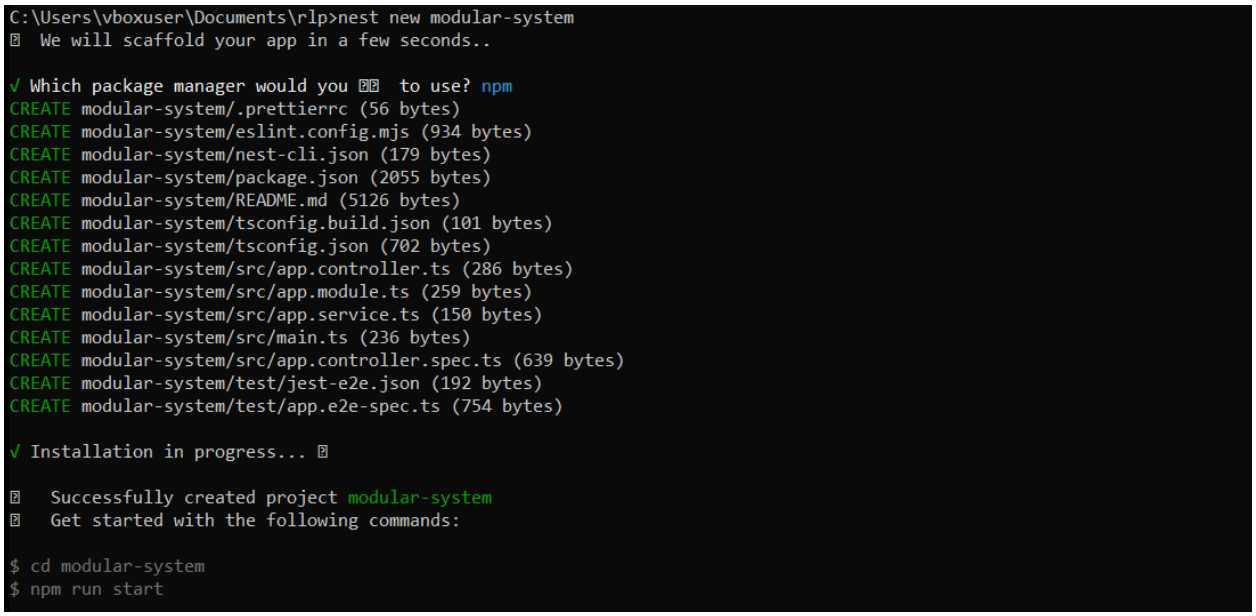
```
npm i -g @nestjs/cli
```

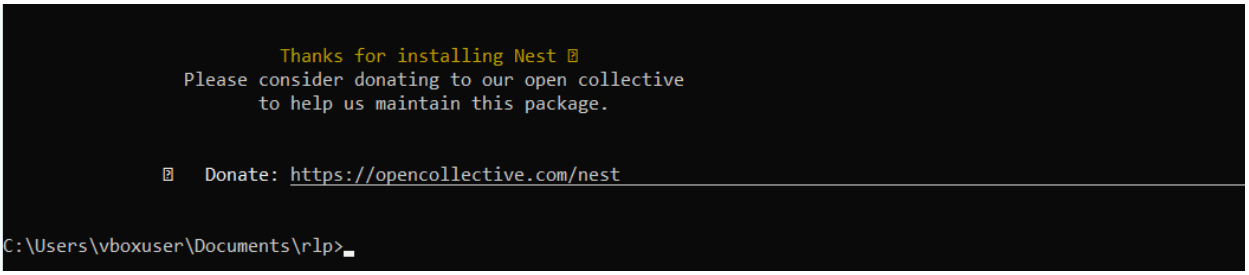
Buat proyek baru

```
nest new modular-system
```

masuk ke direktori proyek baru

```
cd modular-system
```





B. Pembuatan Fitur dengan Clean Layering

Kita akan membuat fitur "Sensor" dengan struktur folder yang memisahkan tanggung jawab (Separation of Concerns):

1. **Domain Layer:** Berisi *Entities* (Aturan bisnis inti). Bebas dari dependensi framework.
2. **Application Layer:** Berisi *Services* (Logika aplikasi/BFF).
3. **Infrastructure Layer:** Berisi *Controllers*, *Modules*, dan integrasi eksternal (Database/API)

C. Implementasi Logika BFF

Gunakan perintah-perintah ini di dalam terminal proyek NestJS Anda:

1. Men-generate Modul Utama

Langkah pertama adalah membuat modul pembungkus agar semua komponen terorganisir dalam satu domain.

Bash

```
nest generate module sensor
```

2. Men-generate Controller (Infrastructure Layer)

Gunakan *flag* `--no-spec` jika Anda tidak ingin membuat file pengujian saat ini. Kita tempatkan di dalam folder `infrastructure`.

Bash

```
nest generate controller sensor/infrastructure/sensor --no-spec
```

3. Men-generate Service (Application Layer)

Service ini akan menampung logika BFF. Secara default, Nest akan menempatkannya di folder utama modul, yang berfungsi sebagai *Application Layer*.

Bash

```
nest generate service sensor/sensor --no-spec
```

4. Membuat Folder dan File Domain (Manual)

Nest CLI tidak memiliki perintah khusus untuk *Entity* atau *Domain Class* karena ini adalah TypeScript murni tanpa decorator framework. Anda perlu membuatnya secara manual:

Di Linux/macOS:

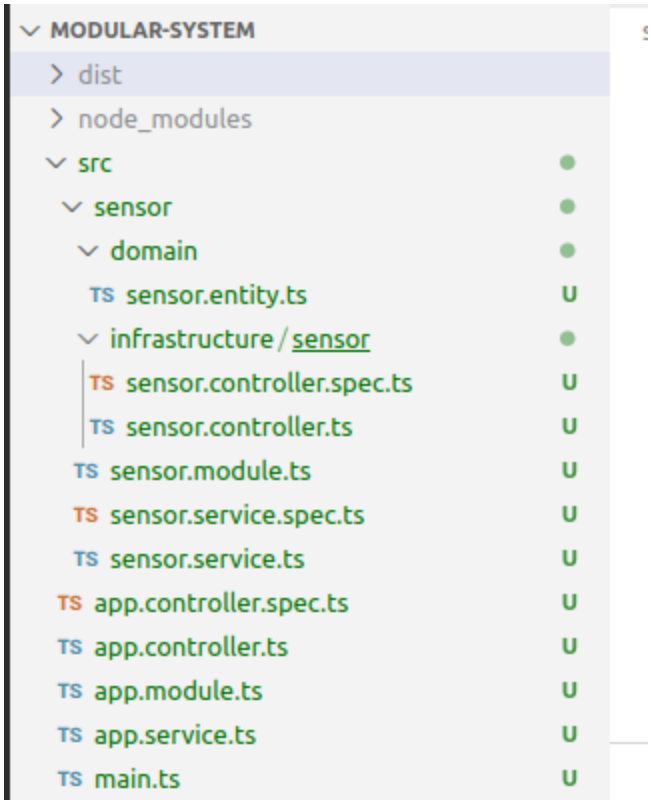
Bash

```
mkdir -p src/sensor/domain && touch src/sensor/domain/sensor.entity.ts
```

Di Windows (Command Prompt):

DOS

```
mkdir src\sensor\domain && type nul > src\sensor\domain\sensor.entity.ts
```



D. Implementasi Kode BFF

1. Domain Layer (The Heart of the System)

Kita harus memastikan Entity menggunakan TypeScript murni tanpa decorator framework jika ingin benar-benar "Clean".

TypeScript

```
// src/sensor/domain/sensor.entity.ts
export class SensorData {
  constructor(
    public readonly id: string,
    public value: number,
    public unit: string,
    public deviceName: string,
    public batteryLevel: number,
    public timestamp: Date,
  ) {}

  // Business Logic: Aturan validasi data
```

```
  isNormal(): boolean {
    return this.value >= 10 && this.value <= 40; // Contoh ambang batas suhu
  }

  isBatteryCritical(): boolean {
    return this.batteryLevel < 20;
  }
}
```

2. Application Layer (Logic BFF)

Di sini kita membuat fungsi agar lebih deskriptif dan menangani transformasi data secara eksplisit.

TypeScript

```
// 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());

  // BFF: Transformasi data khusus untuk Mobile
  // Fokus: Hemat Bandwidth & Kecepatan Baca
  getSummaryForMobile() {
    return {
      v: this.mockData.value,
      u: this.mockData.unit,
      st: this.mockData.isNormal() ? 'OK' : 'WARN', // Menggunakan logika domain
      batt: this.mockData.isBatteryCritical() ? 'LOW' : 'GOOD',
    };
  }

  // BFF: Transformasi data khusus untuk Web Dashboard
  // Fokus: Analitik Lengkap & Transparansi Data
  getFullDetailsForWeb() {
```

```
    return {
      id: this.mockData.id,
      deviceName: this.mockData.deviceName,
      reading: {
        current: this.mockData.value,
        unit: this.mockData.unit,
        isOptimal: this.mockData.isNormal(),
      },
      status: {
        battery: `${this.mockData.batteryLevel}%`,
        lastSeen: this.mockData.timestamp.toISOString(),
      },
    };
  }
}
```

3. Infrastructure Layer (The External Interface)

Pada bagian ini, kita mengimplementasikan modul dan *controller* yang akan mengekspos logika BFF yang telah dibuat di layer aplikasi ke dalam *endpoint* API yang dapat diakses oleh Web dan Mobile.

3.1. Controller (Adaptor untuk HTTP)

Controller berfungsi sebagai gerbang masuk permintaan. Ia tidak memiliki logika bisnis, melainkan hanya bertugas memanggil *service* yang sesuai.

TypeScript

```
// src/sensor/infrastructure/sensor.controller.ts
import { Controller, Get } from '@nestjs/common';
import { SensorService } from '../sensor.service'; // Import dari Application Layer

@Controller('sensor')
export class SensorController {
  constructor(private readonly sensorService: SensorService) {}

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

```
  }

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

3.2. Module (The Orchestrator)

Module di NestJS berfungsi untuk mengatur dependensi dan membungkus seluruh layer (Domain, Application, Infrastructure) menjadi satu kesatuan fungsional.

TypeScript

```
// src/sensor/sensor.module.ts
import { Module } from '@nestjs/common';
import { SensorController } from './infrastructure/sensor.controller';
import { SensorService } from './sensor.service';

@Module({
  controllers: [SensorController],
  providers: [SensorService],
})
export class SensorModule {}
```

4. Instruksi Pengujian Praktikum

Setelah seluruh kode diimplementasikan, mahasiswa wajib melakukan serangkaian pengujian berikut untuk memvalidasi fungsionalitas sistem:

A. Verifikasi Struktur Proyek

Sebelum menjalankan server, pastikan struktur folder telah mengikuti prinsip *Clean Layering*. Gunakan perintah `tree` di terminal atau cek *Side Bar* VS Code untuk memastikan susunan berikut:

- `src/sensor/domain/sensor.entity.ts` (Logic Bisnis)

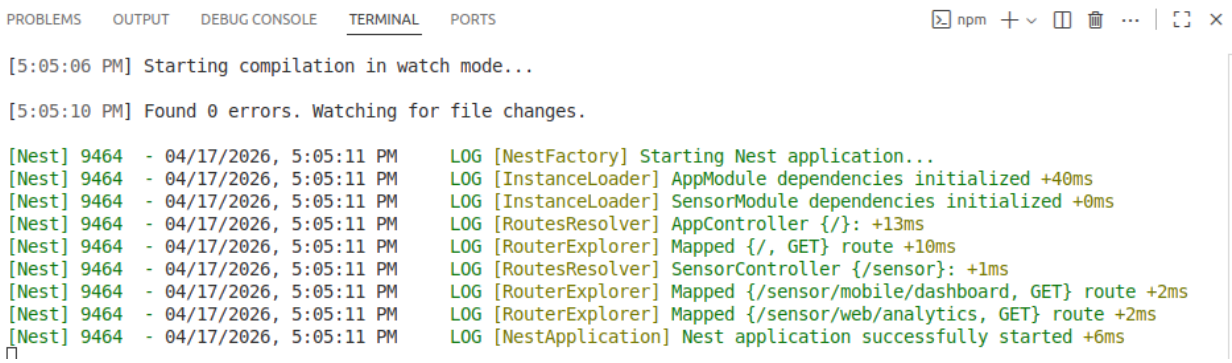
- `src/sensor/sensor.service.ts` (Application Logic/BFF)
- `src/sensor/infrastructure/sensor.controller.ts` (Infrastructure/Endpoint)

B. Menjalankan Server Development

Jalankan perintah berikut pada terminal di dalam direktori proyek:

Bash

```
npm run start:dev
```



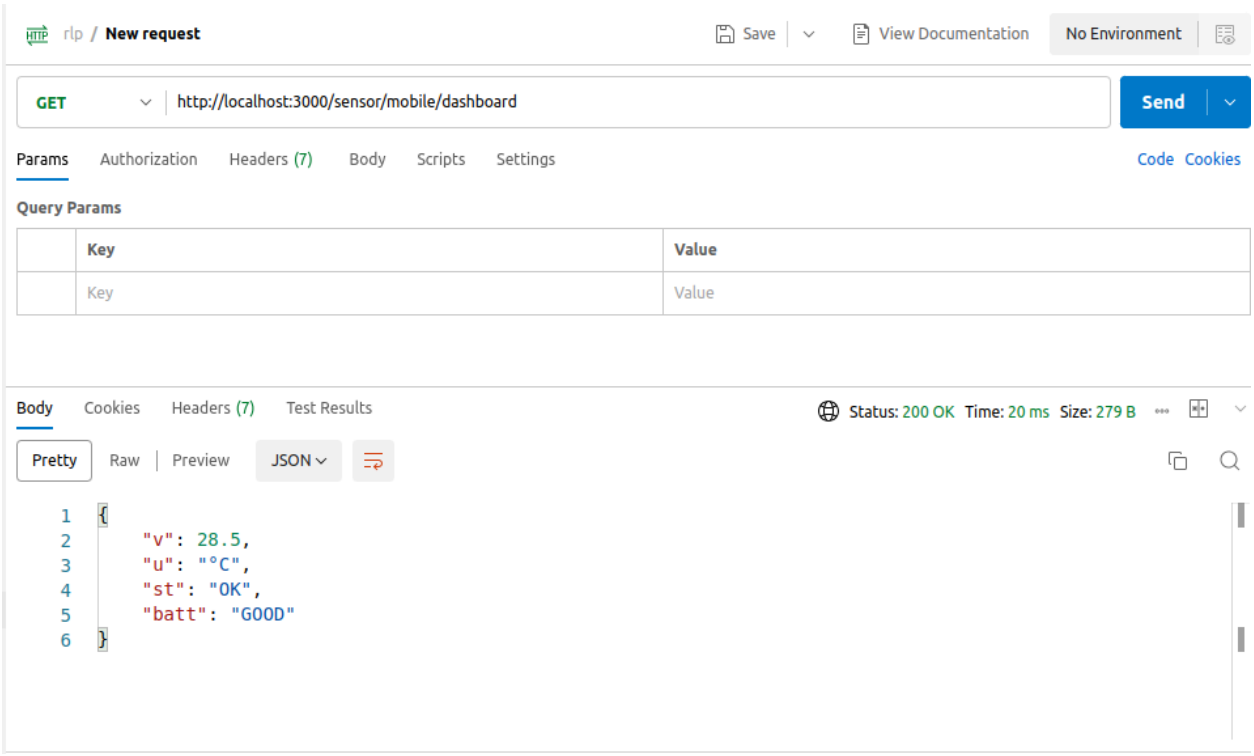
Pastikan log di terminal menunjukkan bahwa `SensorModule`, `SensorController`, dan `SensorService` telah terinisialisasi dengan sukses (*Mapped {/sensor, GET} route*).

C. Pengujian Endpoint dengan Postman/Insomnia

Lakukan pengujian pada dua *endpoint* berbeda untuk melihat penerapan pola BFF:

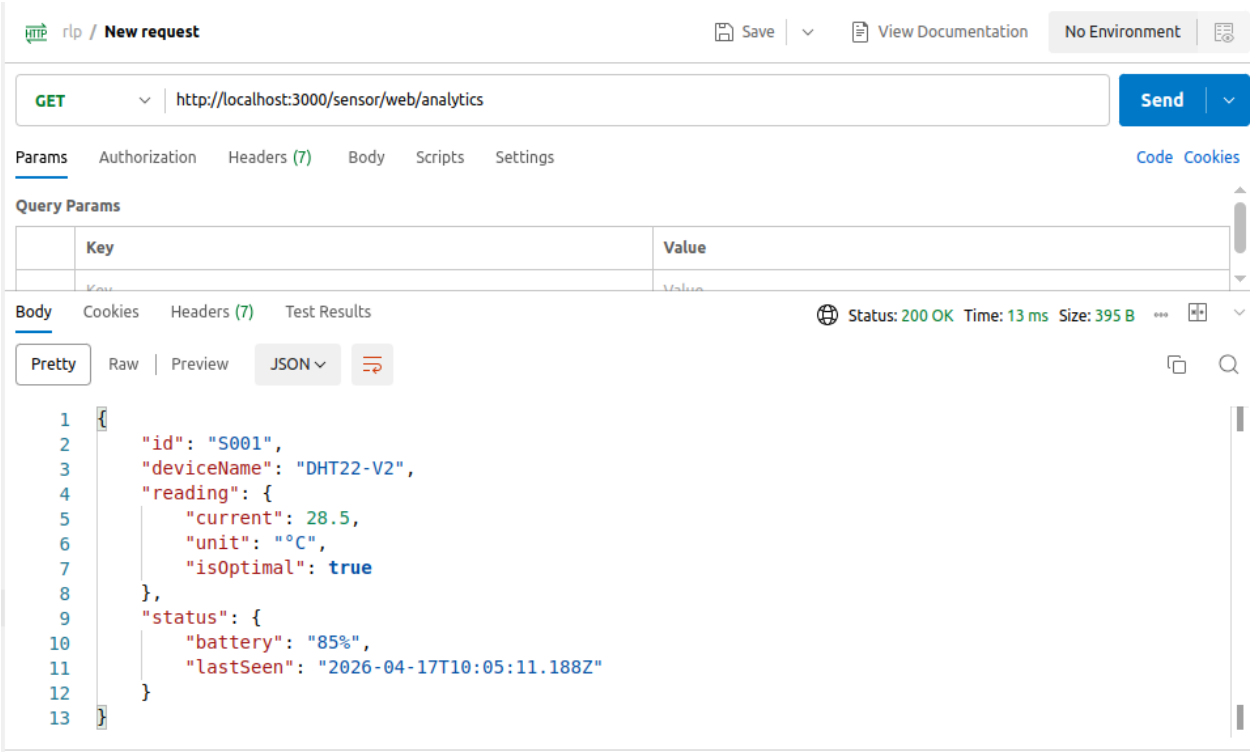
1. Uji Endpoint Mobile (Ringkas):

- **Method:** `GET`
- **URL:** `http://localhost:3000/sensor/mobile/dashboard`
- **Ekspektasi Hasil:** Anda harus menerima objek JSON dengan *key* yang disingkat (misal: `v`, `u`, `st`). Periksa apakah logika `st` (status) berubah menjadi `WARN` jika Anda mengubah nilai suhu pada *mock data* di Service menjadi di luar batas normal.



2. Uji Endpoint Web (Detail):

- **Method:** `GET`
- **URL:** `http://localhost:3000/sensor/web/analytics`
- **Ekspektasi Hasil:** Anda harus menerima objek JSON yang lengkap dengan rincian nama perangkat, ID, baterai, dan *timestamp* dalam format ISO.



D. Uji Independensi Domain (Simulasi Perubahan)

1. Buka file `sensor.entity.ts`.
2. Ubah ambang batas suhu pada fungsi `isNormal()` (misal: dari 40 derajat menjadi 30 derajat).
3. Tanpa mengubah satu baris kode pun di *Controller* maupun *Service*, akses kembali kedua *endpoint* tadi.
4. **Validasi:** Jika kedua *endpoint* mencerminkan perubahan status tersebut secara otomatis, maka Anda telah berhasil mengimplementasikan **Clean Architecture** di mana logika bisnis terpusat di satu tempat (*Single Source of Truth*).

5. Tugas Praktikum: "Smart Home Controller"

Skenario: Anda sedang membangun sistem *Smart Home* yang mengontrol lampu rumah melalui aplikasi mobile dan dashboard web.

Instruksi:

1. **Generate Module:** Buat modul baru bernama light-control menggunakan NestJS CLI.

2. **Domain Layer:** Buat *Class* LightStatus dengan atribut isOn (boolean) dan brightness (number).
3. **Implementasi BFF:** Di dalam light-control.service.ts, buat dua fungsi:
    - getMobileStatus(): Hanya mengembalikan status isOn.
    - getWebStatus(): Mengembalikan isOn dan brightness.
4. **Uji Coba:** Jalankan server dengan npm run start:dev dan akses endpoint tersebut melalui browser atau Postman.

Instruksi Tambahan Tugas:

1. **Folder Mapping:** Mahasiswa wajib memisahkan file ke dalam struktur folder berikut:
    - `src/light-control/domain/` (untuk Entity)
    - `src/light-control/application/` (untuk Service/BFF)
    - `src/light-control/infrastructure/` (untuk Controller)
2. **Validasi Endpoint:** Gunakan aplikasi **Postman** atau **Insomnia** untuk menguji perbedaan hasil antara:
    - `GET http://localhost:3000/light-control/mobile`
    - `GET http://localhost:3000/light-control/web`

Hasil yang Harus Dikumpulkan:

- Screenshot struktur folder di VS Code yang menunjukkan penerapan *layering* (domain, application, infrastructure).
- Screenshot hasil respons JSON dari endpoint mobile dan web yang menunjukkan perbedaan struktur data.