# MODUL PERTEMUAN 4.

# FULLSTACK DEVELOPMENT

Materi

**MEMBUAT API SERVER DENGAN JSON DUMMY**

SEKOLAH TINGGI ILMU KOMPUTER PGRI BANYUWANGI

STIKOM PGRI BANYUWANGI

2024

**MATERI PEMBELAJARAN**

## A. Api Server

Untuk dapat membuat sebuah api server maka kita harus terlebih dahulu mengetahui apa itu api. Untuk itu bacalah materi berikut :

### 1. API (Application Programming Interface)

#### a. Pengertian API

API adalah sekumpulan aturan dan definisi yang memungkinkan dua aplikasi perangkat lunak untuk berkomunikasi satu sama lain. API mendefinisikan cara komponen perangkat lunak harus berinteraksi dan memungkinkan pengembang untuk menggunakan fungsi yang sudah ada tanpa harus menulis kode dari awal.

#### b. Kegunaan API

1) **Integrasi Layanan**: Memungkinkan aplikasi berbeda untuk saling berkomunikasi dan bertukar data.
2) **Akses ke Fungsi**: Memberikan akses ke fungsi atau data tertentu dari suatu aplikasi tanpa mengungkapkan seluruh kode sumber.
3) **Efisiensi Pengembangan**: Menghemat waktu dan usaha dalam pengembangan perangkat lunak dengan menggunakan kembali komponen yang sudah ada.
4) **Konektivitas Lintas Platform**: Memungkinkan interoperabilitas antara sistem yang berbeda, misalnya aplikasi web dan aplikasi mobile.

#### c. Contoh Penggunaan API

1) Integrasi pembayaran online (misalnya, PayPal, Stripe).
2) Aplikasi media sosial yang menampilkan feed dari berbagai platform (misalnya, Facebook, Twitter).
3) Layanan cuaca yang menyediakan data cuaca kepada aplikasi lain.

### 2. REST API (Representational State Transfer API)

#### a. Pengertian REST API

REST API adalah jenis khusus dari API yang mengikuti prinsip-prinsip arsitektur REST. REST adalah gaya arsitektur yang menggunakan protokol HTTP untuk komunikasi dan biasanya beroperasi di atas HTTP/HTTPS. REST API mendefinisikan serangkaian aturan untuk bagaimana klien dan server harus berinteraksi, seringkali menggunakan format data seperti JSON atau XML.

b. Karakteristik REST API

1. **Stateless**: Setiap permintaan dari klien ke server harus berisi semua informasi yang diperlukan untuk memahami dan memproses permintaan. Server tidak menyimpan informasi status klien antar permintaan.
2. **Client-Server**: Pemisahan tanggung jawab antara klien (front-end) dan server (back-end).
3. **Cacheable**: Data respons dari server dapat di-cache untuk meningkatkan kinerja.

4. **Layered System**: Arsitektur bisa memiliki beberapa lapisan yang berinteraksi satu sama lain tanpa klien mengetahui secara langsung interaksi tersebut.
5. **Uniform Interface**: Definisi antarmuka yang seragam untuk memfasilitasi interaksi sistem.

**c. Contoh Penggunaan REST API**

1) API publik seperti Google Maps API, yang menyediakan data peta dan layanan geolokasi.
2) API media sosial seperti Facebook Graph API atau Twitter API, yang memungkinkan pengembang mengakses data pengguna dan posting.
3) API layanan web seperti Amazon S3, yang menyediakan layanan penyimpanan awan.

**3. Perbedaan API dan REST API**

**Perbedaan Utama**

**a. Protokol dan Standar:**

1) API: Bisa menggunakan berbagai protokol dan standar, termasuk SOAP, XML-RPC, dan lainnya.
2) REST API: Khusus menggunakan protokol HTTP/HTTPS dan mengikuti prinsip arsitektur REST.

**b. Format Data:**

1) API: Format data dapat beragam, seperti JSON, XML, YAML, dll.
2) REST API: Biasanya menggunakan JSON atau XML, dengan JSON lebih umum digunakan karena kemudahannya

**c. Kondisi Status:**

1) API: Bisa stateless atau stateful tergantung implementasinya.
2) REST API: Harus stateless sesuai prinsip REST.

**Persamaan**

a. **Tujuan**: Keduanya digunakan untuk memungkinkan komunikasi dan integrasi antar aplikasi.
b. **Keamanan**: Keduanya dapat diimplementasikan dengan berbagai mekanisme keamanan seperti OAuth, API keys, dan token-based authentication.
c. **Efisiensi Pengembangan**: Keduanya dirancang untuk meningkatkan efisiensi pengembangan perangkat lunak dengan memungkinkan penggunaan kembali fungsi atau data yang sudah ada.

**4. Kegunaan REST API**

a. **Pengembangan Web**: Banyak digunakan dalam pengembangan aplikasi web untuk memisahkan front-end dan back-end.
b. **Integrasi Aplikasi**: Memfasilitasi integrasi antara berbagai aplikasi dan layanan, baik internal maupun eksternal.
c. **Layanan Microservices**: Memungkinkan pengembangan aplikasi dengan arsitektur microservices, di mana berbagai layanan kecil berkomunikasi melalui REST API.

d. **Akses Data Publik**: Menyediakan akses ke data publik atau layanan dari penyedia API seperti penyedia data cuaca, layanan berita, dan lainnya.

## B. JSON (JavaScript Object Notation)

### 1. Pengertian JSON

JSON (JavaScript Object Notation) adalah format data yang digunakan untuk pertukaran data antara server dan aplikasi web. JSON adalah format teks yang ringan, mudah dibaca, dan ditulis oleh manusia, serta mudah diurai dan dihasilkan oleh mesin. JSON didasarkan pada subset bahasa pemrograman JavaScript, tetapi format ini independen dari bahasa dan dapat digunakan dengan berbagai bahasa pemrograman seperti Python, Java, C++, dan lainnya.

### 2. Struktur JSON

JSON terdiri dari dua struktur utama:

a. **Objek (Object)**: Dibatasi oleh kurung kurawal `{}` . Objek berisi pasangan kunci-nilai (key-value pairs).

b. **Array (Array)**: Dibatasi oleh tanda kurung siku `[]` . Array adalah kumpulan dari nilai yang diurutkan.

### 3. Contoh Objek JSON

**Json Object**

```
{
  "nama": "John Doe",
   "usia": 30,
   "alamat": {
              "jalan": "Jalan Merdeka No. 1",
              "kota": "Jakarta",
              "negara": "Indonesia"
   },
  "hobi": ["membaca", "bersepeda", "berenang"]
}
```

### 4. Contoh Array JSON

**Json Array**

```
[
  {
    "nama": "John Doe",
     "usia": 30
   },
   {
      "nama": "Jane Doe",
       "usia": 25
  }
]
```

5. **Elemen Dasar JSON**
   a. Nilai String: Harus berada dalam tanda kutip ganda.

| **json** |
| --- |
| "nama": "John Doe" |

   b. Nilai Number: Tidak menggunakan tanda kutip.

| **json** |
| --- |
| "usia": 30 |

   c. Nilai Boolean: `true` atau `false`.

| **json** |
| --- |
| "menikah": true |

   d. Null: Menunjukkan nilai kosong.

| **json** |
| --- |
| "anak": null |

   e. Objek: Dibatasi oleh kurung kurawal `{}`.

| json |
| --- |
| "alamat": {<br>"jalan": "Jalan Merdeka No. 1",<br>"kota": "Jakarta"<br>} |

   f. Array: Dibatasi oleh tanda kurung siku `[]`.

| json |
| --- |
| "hobi": ["membaca", "bersepeda", "berenang"] |

6. **Kegunaan JSON**
   a. Pertukaran Data: Digunakan secara luas untuk pertukaran data antara klien dan server dalam aplikasi web.
   b. Penyimpanan Data: Banyak digunakan dalam basis data NoSQL seperti MongoDB untuk menyimpan dan mengelola data.
   c. Konfigurasi: Digunakan untuk menyimpan file konfigurasi aplikasi.
   d. APIs: JSON adalah format yang paling umum digunakan dalam REST API untuk mengirim dan menerima data.

7. **Kelebihan JSON**
   a. Ringan dan Efisien: JSON adalah format teks yang ringan, sehingga lebih cepat untuk diunduh dan diunggah.

b. Mudah Dibaca dan Ditulis: Sintaks JSON sederhana dan mudah dipahami oleh manusia.
c. Bahasa Independen: JSON dapat digunakan dengan berbagai bahasa pemrograman.
d. Mudah Diurai dan Dibuat: Kebanyakan bahasa pemrograman memiliki pustaka atau metode bawaan untuk mengurai dan membuat JSON.

## C. Express js

### 1. Pengertian Express.js

Express.js adalah kerangka kerja (framework) web yang minimalis dan fleksibel untuk Node.js, dirancang untuk membangun aplikasi web dan API. Express menyediakan sekumpulan fitur yang kuat untuk aplikasi web dan mobile, serta mempermudah pengembangan server-side dengan menyediakan antarmuka yang sederhana namun kaya fitur.

### 2. Fitur Utama Express.js

a. Routing: Sistem routing yang kuat dan fleksibel, memungkinkan penanganan permintaan HTTP dengan mudah.
b. Middleware: Mendukung middleware untuk menangani berbagai tugas seperti pengolahan permintaan, autentikasi, logging, dan lainnya.
c. Template Engines: Mendukung berbagai mesin template untuk rendering halaman web dinamis.
d. Kompatibilitas: Kompatibel dengan berbagai pustaka dan modul Node.js lainnya, memudahkan integrasi.
e. Error Handling: Fasilitas penanganan kesalahan yang terintegrasi.
f. Performance: Ringan dan cepat, cocok untuk aplikasi dengan performa tinggi.

### 3. Kegunaan Express.js

a. Aplikasi Web: Digunakan untuk membuat berbagai jenis aplikasi web, dari yang sederhana hingga yang kompleks.
b. API: Memfasilitasi pembuatan API RESTful yang efisien dan terstruktur.
c. Middleware: Penggunaan middleware untuk menangani tugas-tugas spesifik dalam aplikasi, seperti validasi, parsing, dan autentikasi.

### 4. Middleware dalam Express.js

Middleware adalah fungsi yang memiliki akses ke objek permintaan (`req`), objek respons (`res`), dan fungsi middleware berikutnya dalam siklus permintaan-respons aplikasi. Middleware dapat melakukan tugas-tugas seperti:

a. Melakukan Perubahan pada Objek Permintaan dan Respons: Misalnya, menambahkan data tambahan.
b. Mengakhiri Siklus Permintaan-Respons: Misalnya, mengirim respons ke klien.
c. Memanggil Middleware Berikutnya dalam Rantai: Dengan memanggil fungsi `next()`.

5. **Keuntungan Menggunakan Express.js**
    a. Kemudahan Penggunaan: Sintaks sederhana dan mudah dipelajari.
    b. Ekosistem yang Kuat: Banyak modul dan pustaka yang bisa diintegrasikan dengan mudah.
    c. Fleksibilitas: Dapat digunakan untuk berbagai jenis aplikasi, dari API sederhana hingga aplikasi web yang kompleks.
    d. Kinerja Tinggi: Dirancang untuk efisiensi dan kinerja yang optimal.

Express.js telah menjadi pilihan populer bagi pengembang yang ingin membangun aplikasi web dan API dengan cepat dan efisien menggunakan Node.js, berkat kesederhanaannya, fleksibilitas, dan ekosistem yang kaya.

**MATERI PRAKTIKUM**

**A. Persiapan**

- Buatlah sebuah folder baru
- Kemudian dengan dengan cmd masuk kedalam folder
- Ketikkan "***npm init –y*** " untuk
- Ketikkan "***npm install express***"
- Kemudian bualah sebuah file baru dengan extensi .js contoh server.js
- Buat pula file data.json

B. Masukkan Code berikut :

```
const express = require('express');
const fs = require('fs').promises;

const app = express();
const PORT = process.env.PORT || 3001;

app.use(express.json());

// Path menuju file JSON
const dataFilePath = 'data.json';

// Mendapatkan data dari file JSON
async function getData() {
  try {
    const data = await fs.readFile(dataFilePath, 'utf8');
    return JSON.parse(data);
  } catch (err) {
    console.error('Error reading data:', err);
    return [];
  }
}
```

```
// Menyimpan data ke dalam file JSON
async function saveData(data) {
  try {
    await fs.writeFile(dataFilePath, JSON.stringify(data, null, 2));
    console.log('Data saved successfully.');
  } catch (err) {
    console.error('Error saving data:', err);
  }
}

// Menyambut permintaan ke root URL
app.get('/', (req, res) => {
  res.send('Welcome to the REST API!');
});

// Mendapatkan semua item
app.get('/items', async (req, res) => {
  const data = await getData();
  res.json(data);
});
```

```
// Menambahkan item baru
app.post('/items', async (req, res) => {
  const newItem = req.body;
  const data = await getData();
  data.push(newItem);
  await saveData(data);
  res.json(newItem);
});

// Menampilkan detail item berdasarkan ID
app.get('/items/:id', async (req, res) => {
  const { id } = req.params;
  const data = await getData();
  const item = data.find(item => item.id === parseInt(id));
  if (item) {
    res.json(item);
  } else {
    res.status(404).send('Item not found.');
  }
});
```

```
// Menyambut permintaan PUT untuk memperbarui item berdasarkan ID
app.put('/items/:id', async (req, res) => {
  const { id } = req.params;
  const newData = req.body;
  let data = await getData();
  const index = data.findIndex(item => item.id === parseInt(id));
  if (index !== -1) {
    data[index] = { ...data[index], ...newData };
    await saveData(data);
    res.json(data[index]);
  } else {
    res.status(404).send('Item not found.');
  }
});


// Menghapus item berdasarkan ID
app.delete('/items/:id', async (req, res) => {
  const { id } = req.params;
  let data = await getData();
  data = data.filter(item => item.id !== parseInt(id));
  await saveData(data);
  res.send('Item deleted successfully.');
});

// Menjalankan server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

## C. Buat file data.json

Buatlah file data.json yang isinya adalah sebagai berikut :

| Data.json |
| --- |
| [] |

## D. Jalankan Aplikasi:

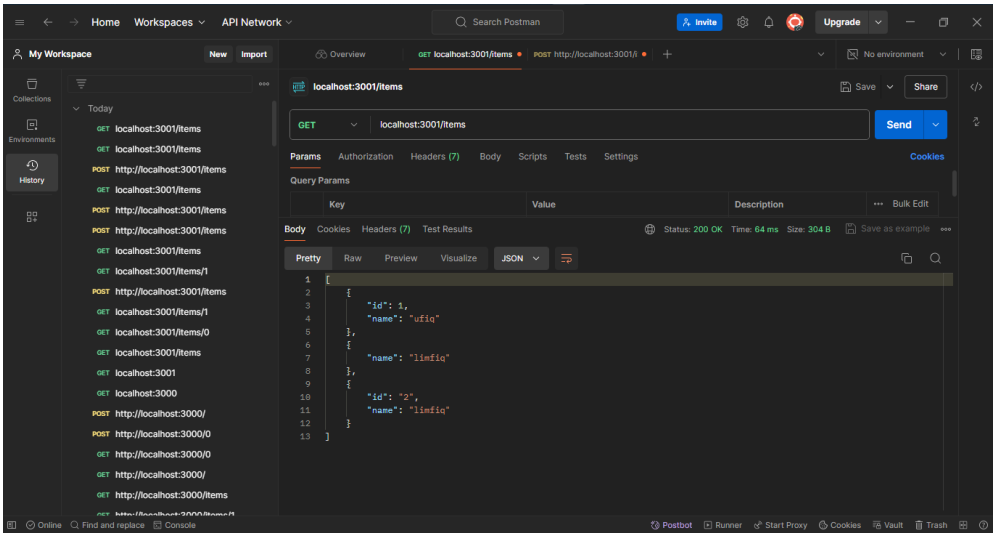
1. Jalankan perintah "***npx json-server namafile.json***" tunggu proses hingga selesai sampai seperti dalam gambar berikut :

```
PS F:\fullstack\fullstack\pertemuan4> npx json-server data.json
JSON Server started on PORT :3000
Press CTRL-C to stop
Watching data.json...

ᕙ(⇀‸↼‶)ᕗ

Index:
http://localhost:3000/

Static files:
Serving ./public directory if it exists

Endpoints:
http://localhost:3000/0
```

2. Buka cmd baru ketikkan perintah "***node server.json***"

```
PS F:\fullstack\fullstack\pertemuan4> nodemon server.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
```

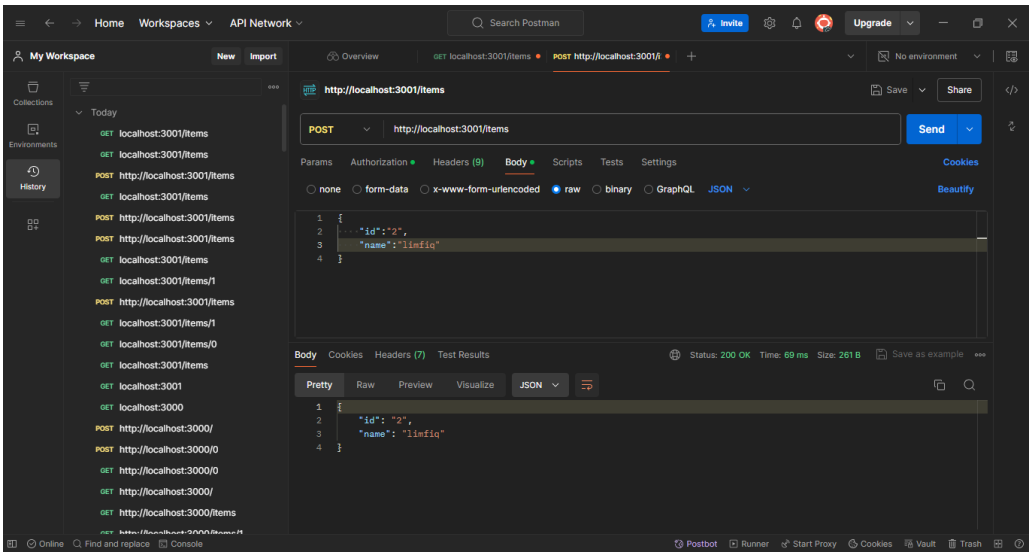
3. Kemudian bukalah postman anda dengan metode get dan address http://localhost:3001/items



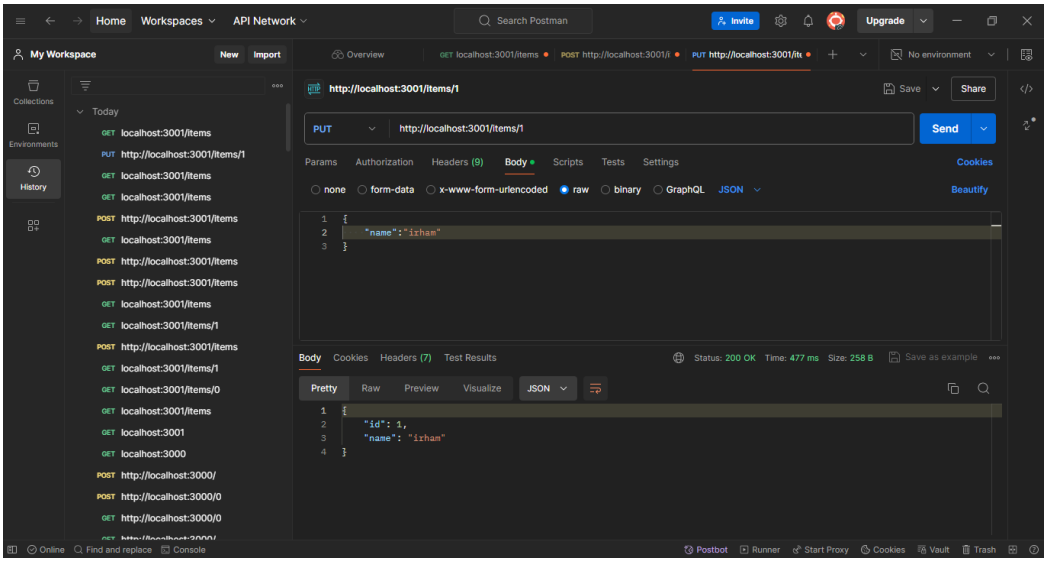
4. Untuk Menambahkan data gunakan metode post dan address : http://localhost:3001/items, masukk ke body➔raw ➔json dan masukkan kode

```
{
  “id”:”isi dengan id”,
  “name”:”isi dengan nama”
}
```

Kemudian klik send




5. Untuk mengedit data silahkan pilih metode put dan address : http://localhost:3001/items/id kemudian masuk ke body ➔ raw ➔json dan tambahkan json yang akan dijadikan perubahan




6. Sedangkan untuk menghapus silahkan gunakan methode delete dan address : http://localhost:3001/items/id kemudian klik send

**TUGAS**

1. Buatlah program diatas dan cobalah dengan data dengan bentuk lain.
2. Ambillah screenshot untuk keperluan laporan
3. Buatlah laporan untuk modul ke empat
4. Upload ke elearming pada pertemuan 4