

Pengenalan Web Server & NodeJs

Sio Jurnal Pipin, S.Kom.



Hosting

- Tempat penyimpanan data atau file computer
- Memiliki bentuk fisik yang biasa disebut dengan server. Seperti gambar, video, audio dan sebagainya.
- File-file tersebut akan disimpan dalam sebuah tempat yang dinamakan server hosting.
- Server hosting merupakan tempat penyimpanan data, dimana nantinya server akan menampilkan data-data tersebut pada web client (browser) seperti Chrome, Mozilla, Safari, dan Opera.

Jenis Server Hosting

- **Share Hosting**

Dapat digunakan secara bersama - sama atau disebut juga di sharing.

Dalam server ini biasanya terdapat beberapa nama domain dengan kepemilikan yang berbeda.

- **Virtual Private Server**

Merupakan sebuah server yang dibagi - bagi menjadi sebuah virtual mesin, jadi dalam satu computer server terdapat beberapa system operasi server, tentunya secara kepemilikan antara system operasi satu dengan yang lainnya berbeda.

Jenis Server Hosting

- **Dedicated Server**

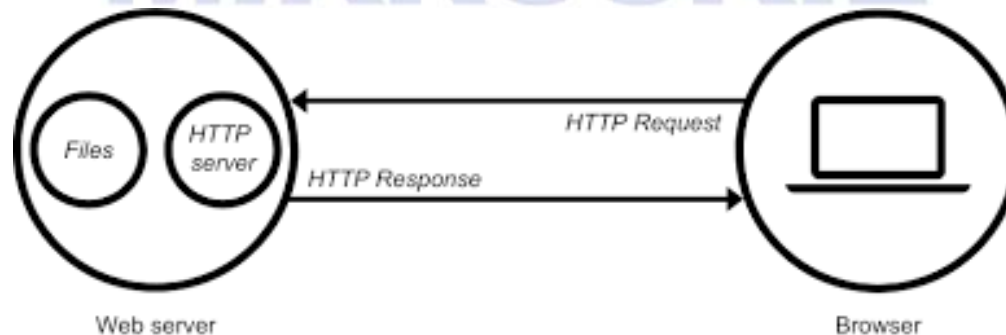
Server yang satu ini biasanya digunakan untuk penyimpanan aplikasi yang lebih besar dari pada Share hosting maupun VPS, jadi pengguna menyewa secara keseluruhan dari computer server yang disediakan oleh webhost.

- **Colocation Server**

Sebuah server yang digunakan untuk keperluan web hosting

Web Server

- Software yang memberikan layanan data yang mempunyai fungsi untuk menerima permintaan HTTP atau HTTPS yang dikirim oleh klien melalui web browser dan mengirimkan Kembali hasilnya dalam bentuk halaman web (HTML / JS).
- Berguna sebagai tempat aplikasi web dan sebagai penerima request dari client.



Node.js

- Sebuah platform software yang dipakai untuk membangun aplikasi – aplikasi serverside yang fleksibel di sebuah jaringan aplikasi.
- Menggunakan javascript sebagai basis bahasa pemrogramannya.
- Memiliki skalabilitas yang sangat tinggi, berbasis **event** (event driven programming), dan memiliki konsep **asynchronous** yang sangat bagus sehingga membuatnya ringan dan efisien.

Node.js

- Platform perangkat lunak yang berjalan pada sisi-server dan aplikasi jaringan.
- Platform ini terdiri dari 2 hal, yaitu **runtime environment** dan **script library**.
- Memiliki pustaka **server HTTP** sendiri sehingga memungkinkan untuk menjalankan webserver tanpa menggunakan program webserver seperti apache atau lighttpd.
- Menggunakan Engine Javascript dari Google bernama V8

Install Node.js

- Download Node.js versi LTS di <https://nodejs.org/en/download/>
- Install dengan klik “**Next**” hingga proses **instalasi** dimulai.
- Verifikasi Instalasi menggunakan command prompt atau PowerShell dengan perintah:
node -v dan **npm -v**

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19042.844]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\SIO>node -v
v14.15.3

C:\Users\SIO>npm -v
6.14.9
```


Module HTTP Node.js

- Module yang digunakan untuk **membuat server**
- Modul HTTP membuat server HTTP yang dengan port server dan memberikan respons kembali ke klien.
- Dapat digunakan untuk mengambil **URL routing**, dan **Query String**.
- **Dokumentasi:** <https://nodejs.org/api/http.html>

Membuat Web Server dengan HTTP

- Buat file **index.js** pada folder M01-HttpModule
- Edit file **index.js** seperti gambar di bawah
- Pada folder M01-HttpModule run terminal / CMD dengan perintah **node index.js**

```
SIQ@DESKTOP-U2DMK91 MINGW64 ~/OneDrive - mikroskil.ac.id/Mikroskil/Mata Kuliah/6. Pemrograman Web Lanjutan/Webserver/M01-HttpModule
$ node index.js
```

```
index.js  X
M01-HttpModule > index.js > ...
1  const http = require("http");
2
3  //buat object server dengan listen port 3000
4  http
5  .createServer(function (req, res){
6    ....//Respon ke browser / client
7    ....res.write("Module HTTP Mobile and Web");
8
9    ....//Respon diakhiri
10   ....res.end();
11   ..})
12   ..listen(3000);
13
```

Buka browser dengan alamat
Localhost:3000

← → ↻ ⓘ localhost:3000

Module HTTP Mobile and Web

HTTP Header – Jenis Konten

- Fungsi `writeHead(status_code, server_response)` memiliki 2 parameter, yakni:
 1. HTTP Status code, bila nilainya 200 itu berarti OK. Nilai status code lainnya adalah 400 yang berarti Bad Request, dll | Baca lainnya <https://developer.mozilla.org/id/docs/Web/HTTP/Status>.
 2. Server Response/Response header yang berisi sebuah nilai keterangan seperti **'Content-Type': 'text/html'**. | baca lainnya disini <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

Jenis lainnya seperti:

```
response.writeHead(200, {'Content-Type': 'application/json'}); // untuk JSON
```

```
response.writeHead(200, {'Content-Type': 'application/pdf'}); // untuk PDF
```

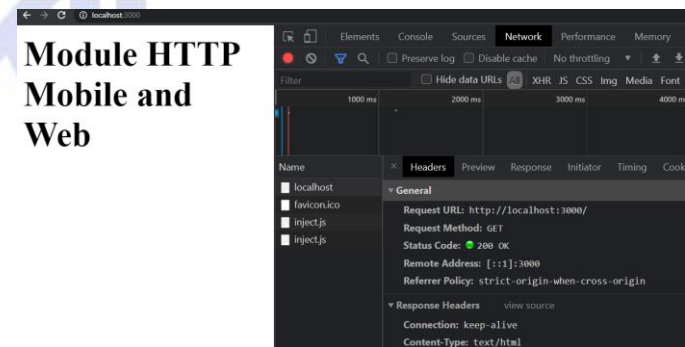
```
response.writeHead(200, {'Content-Type': 'application/xml'}); // untuk XML
```

HTTP Header - WriteHead

- Tambahkan **baris 7** dan ubah **baris 11** dengan HTML **H1** seperti pada gambar.
- Stop server di CMD dengan **CTRL + C**, lalu jalankan ulang server dengan perintah **node index.js**

```
index.js x
M01-HttpModule > .\index.js > ...
1  const http = require("http");
2
3  //buat object server dengan listen port 3000
4  http
5  .createServer(function (req, res) {
6    //http Header
7    res.writeHead(200, { "Content-Type": "text/html" });
8
9    //Respon ke browser / client
10   res.write("<H1>Module HTTP Mobile and Web</H1>");
11
12   //Respon diakhiri
13   res.end();
14 }
15 .listen(3000);
16
```

Cek dengan inspek element, pada **Bagian Network**, pilih **localhost**. Terdapat status code **200** dan **Content-Type: Text/html**



Latihan

- Install NodeJs pada perangkat kamu
- Buatlah web server menggunakan dengan listen post 3400 untuk menampilkan pesan dengan formatting HTML.



Module Fs dan HTTP, Async dan Sync, Request dan Response

Sio Jurnal Pipin, S.Kom.



**STMIK - STIE
MIKROSKIL**

PRODI. TEKNIK INFORMATIKA (S-1)

File System (fs Module)

- Bagian dari Node.js dengan memanggil
const fs = require('fs')
- Berguna untuk mengakses dan berinteraksi dengan sistem file.
- **Module fs** menggunakan metode **asynchronous** secara default, tetapi juga dapat bekerja secara **sinkron** dengan menambahkan **Sync**.

File System (fs Module)

Method	Fungsi
fs.appendFile()	Menambahkan data ke file. Jika file tidak ada, maka file akan dibuat.
fs.chmod()	Mengubah izin file yang ditentukan oleh nama file
fs.chown()	Mengubah pemilik dan grup file yang ditentukan oleh nama file yang dikirimkan
fs.close()	Menutup file
fs.copyFile()	Menyalin file
fs.createReadStream()	Membuat aliran file yang dapat dibaca
fs.createWriteStream()	Membuat aliran file yang dapat ditulis
fs.mkdir()	Membuat folder baru
fs.mkdtemp()	Buat direktori sementara
fs.open()	atur mode file / buka file
fs.readdir()	Membaca isi direktori
fs.access()	Memeriksa apakah file tersebut ada dan NodeJs dapat mengaksesnya dengan izin

File System (fs Module)

Method	Fungsi
fs.readFile() / fs.read()	Membaca isi file
fs.rename()	Ganti nama file atau folder
fs.rmdir()	Hapus folder
fs.unwatchFile()	Berhenti mengawasi perubahan pada file
fs.watchFile()	Mulai perhatikan perubahan pada file
fs.writeFile() / fs.write()	menulis data ke file

MIKROSKIL

File System – readFile()

- Membaca isi file
- **Implementasi** pada module http
 1. Terima request dari user
 2. **Baca file HTML** pada server
 3. Kirim respon ke client

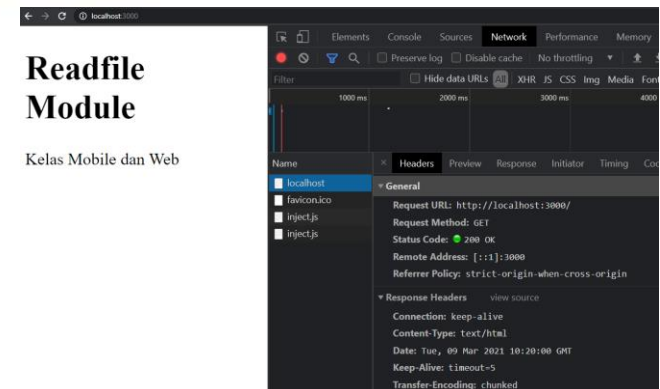
MIKROSKIL

File System – readFile()

- Buat dalam **folder M02** sebuah file **index.js** dan **index.html**; file html silahkan buat dengan menerapkan **tag h1** dan **p** isi index.js seperti pada gambar berikut:

```
index.js x index.html
M02-fsModuleAsync > . index.js > ...
1  const http = require("http");
2  const fs = require("fs");
3
4  //buat object server dengan listen port 3000
5  http
6  .createServer(function (req, res) {
7    //menggunakan modul readFile
8    fs.readFile("index.html", (err, data) => {
9      //mengembalikan pesan error ketika gagal baca file
10     if (err) throw err;
11
12     //http header
13     res.writeHead(200, { "Content-Type": "text/html" });
14
15     //Respon ke browser / client berupa data dari file index.html
16     res.write(data);
17
18     //Respon diakhiri
19     res.end();
20   });
21 }
22 .listen(3000);
23
```

- Run server pada terminal:
node index.js dalam folder M02.
- Akses pada localhost:3000



File System – Membuat file

Method yang dapat digunakan untuk membuat file:

- **fs.appendFile()** untuk membuat dan mengisi file;
- **fs.open()** untuk membuat, membuka, dan menulis file;
- **fs.writeFile()** untuk membuat dan menulis file.



MIKROSKIL

File System – appendFile

Digunakan untuk membuat dan mengisi file.

- Buat file **appendFile.js**
- Run dengan **node appendFile.js**
- File **mw.txt** akan bertambah pada folder **M02**

```
appendFile.js X
M02-fsModuleAsync > node appendFile.js > ...
1  const fs = require("fs");
2
3  //buat file mw.txt dan mengisi dengan data:
4  fs.appendFile("mw.txt", "Kelas Mobile dan Web!", function (err) {
5    if (err) throw err;
6    console.log("Berhasil disimpan!");
7  });
8
9  //run dengan: node appendFile.js
10
```

```

M02-fsModuleAsync
├── appendFile.js
├── index.html
├── index.js
└── mw.txt
```

File System – fs.open()

- Digunakan untuk membuka dan menulis file.

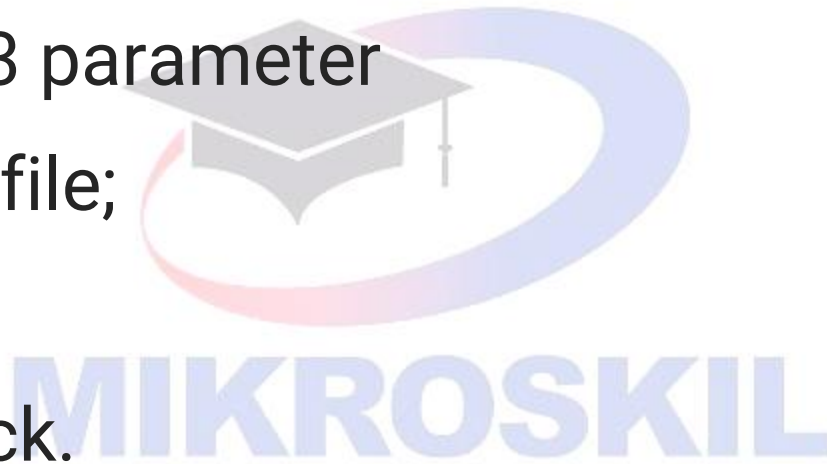
fs.open(filename, flags, callback)

- Terdapat 3 parameter

- a. Nama file;

- b. Flag;

- c. callback.



File System – fs.open()

Flag	Description
r	To open file to read and throws exception if file doesn't exists.
r+	Open file to read and write. Throws exception if file doesn't exists.
rs+	Open file in synchronous mode to read and write.
w	Open file for writing. File is created if it doesn't exists.
wx	It is same as 'w' but fails if path exists.
w+	Open file to read and write. File is created if it doesn't exists.
wx+	It is same as 'w+' but fails if path exists.
a	Open file to append. File is created if it doesn't exists.
ax	It is same as 'a' but fails if path exists.
a+	Open file for reading and appending. File is created if it doesn't exists.
ax+	It is same as 'a+' but fails if path exists.

File System – fs.open()

- Buat file openwriteFile.js, isi seperti gambar di bawah
- Run dari terminal dengan node openwriteFile.js

```
openwriteFile.js X
M02-fsModuleAsync > openwriteFile.js > ...
1  var fs = require("fs");
2
3  fs.open("datamw.txt", "w+", function (err, file) {
4    if (err) throw err;
5
6    // data yang akan kita tulis ke file
7    let data = "Kelas Mobile dan Web!";
8
9    // tulis konten ke file
10   fs.writeFile(file, data, (err) => {
11     if (err) throw err;
12     console.log("Tersimpan!");
13   });
14
15   // baca file
16   fs.readFile(file, (err, data) => {
17     if (err) throw err;
18     console.log(data.toString("utf8"));
19   });
20 });
21
```

Pada fungsi `fs.readFile()`, digunakan fungsi `toString('utf8')` untuk mengubah **buffer** menjadi teks dengan **encode UTF-8**.

Async dan Sync

- File System defaultnya adalah asynchrone namun dapat bekerja secara sinkron dengan menambahkan Sync
 - `fs.rename()`
 - **`fs.renameSync()`**
 - `fs.write()`
 - **`fs.writeSync()`**
- Synchronous adalah sebuah operasi yang akan dijalankan setelah operasi sebelumnya selesai dijalankan atau **berurutan**.
- Asynchronous sebaliknya, tidak perlu menunggu operasi sebelumnya selesai untuk mengeksekusi operasi setelahnya.

Async dan Sync

- Perubahan terjadi pada bentuk sync yaitu dapat code dibungkus pada blok try/catch.
- Buat file **mwsore.json**
- Buat dan isi file **asyncRename.js**
- Run dengan terminal: **node asyncRename.js**

```
JS asyncRename.js X
M02-fsModuleAsync > JS asyncRename.js > ...
1  const fs = require("fs");
2
3  fs.rename("mwpagi.json", "mwsore.json", (err) => {
4    if (err) {
5      return console.error(err);
6    }
7
8    //done
9    console.log("Berhasil mengganti nama!");
10 });
11 |
```

Async dan Sync

- Buat dan isi file **syncRename.js**
- Run dengan terminal: **node syncRename.js**

A screenshot of a code editor window titled 'syncRename.js'. The editor shows the following JavaScript code:

```
1  const fs = require("fs");
2
3  try {
4    fs.renameSync("mwsore.json", "mwpagi.json");
5    //done
6    console.log("Berhasil mengganti nama!");
7  } catch (err) {
8    console.error(err);
9  }
10
```

- Perbedaan utama di sini adalah eksekusi skrip akan diblok, hingga operasi file berhasil.

Request dan Response

- Pada http server, bentuk req dan respon dapat dilihat seperti ini
`http.createServer(function (req, res) {})`
- Pada aplikasi ExpressJs bentuknya ada
`app.get('/', function (req, res) {})`
- **Request** merupakan permintaan HTTP berisi permintaan permintaan parameter string, konten, atribut HTTP header.
- **Response** merupakan respon HTTP, yaitu ketika menerima permintaan dikirim ke data respon klien.

HTTP Client

- Digunakan dalam pembuatan HTTP Request dan pembacaan HTTP Response.
- Mengambil data dari server lain untuk diproses lebih lanjut dari sisi server
- Http client pada NodeJS yaitu **http.request** dengan parameter options, dan callback.
- Property dalam option yaitu:
 - **host**, nama domain atau IP server tujuan. Nilai standar adalah "localhost".
 - **hostname**, sama seperti host. Gunakan nilai ini untuk mendukung url.parse.
 - **port**, port dari server yang dituju. Nilai standar adalah 80.
 - **localAddress**, antarmuka lokal yang ingin diikatkan pada koneksi lokal.
 - **socketPath**, Unix Domain Socket (gunakan salah satu dari host:port atau socketPath).
 - **method**, method dari HTTP Request. Nilai standar adalah GET.
 - **path**, path dari request. Nilai standar adalah /. Jika ada, query string juga diikutkan, misalnya: index.html?data=123. Jika terdapat karakter ilegal sebuah Exception akan dilemparkan. Untuk sekarang, karakter ilegal hanya spasi.
 - **headers**, sebuah objek yang berisi header HTTP.
 - **auth**, informasi autentikasi dengan HTTP Basic Authentication (user:pass) jika server memerlukan informasi ini.
 - **agent**, digunakan untuk mengatur HTTP Agent. Pembahasan HTTP Agent akan dilakukan pada bagian lain.
 - **keepAlive**, sebuah boolean untuk menentukan apakah koneksi tetap disimpan untuk request selanjutnya. Nilai standar adalah false.
 - **keepAliveMsecs**, menentukan waktu koneksi akan tetap dibuka, dalam ukuran milisecond. Nilai standar adalah 1000. Property ini hanya relevan jika keepAlive bernilai true.

HTTP Client

- Buat file httpClient.js pada folder M02 lalu run terminal dengan **node httpClient.js**.

```
httpClient.js X
M02-fsModuleAsync > .js httpClient.js > ...
1  var http = require("http");
2
3  //menentukan property options yang akan digunakan
4  var options = {
5    hostname: "www.google.com",
6    port: 80,
7    path: "/",
8    method: "GET",
9
10   //header digunakan untuk menentukan tipe header yang digunakan seperti tipe
    konten dll.
11   headers: {
12     "Content-Type": "application/json",
13   },
14 };
15
16 var req = http.request(options, function(response) {
17   console.log(response.statusCode);
18   console.log(response.statusMessage);
19   console.log(response.headers);
20 });
21
22 //untuk handle error
23 req.on("error", function(e) {
24   console.log("Error: " + e.message);
25 });
26
27 //tutup koneksi diakhir.
28 req.end();
29
30 //Jalankan dengan node httpClient.js
31
```

NPM, ExpressJs dan Serve Static

Sio Jurnalists Pipin, S.Kom.



NPM (Node Package Manager)

- NPM menyediakan dua fungsionalitas utama: **repositori online untuk Paket / Modul Node.js** yang dapat dicari di **NodeJS.org**. dan, **Utilitas perintah** untuk Instal Paket Node.js, manajemen Versi dan Manajemen *dependency* module Node.js (lihat **package.json**)
- **NPM** digunakan untuk:
 1. **Membuat Project Baru;**
 2. **Menginstal modul atau library;**
 3. **Menjalankan skrip command line;**

Lihat: <https://docs.npmjs.com/about-npm>

Membuat Proyek dengan NPM

```
SIO@DESKTOP-U2DMK91 MINGW64 ~/OneDrive - mikroskil.ac.id/Mikroskil/Mata Kuliah/6. Pemrograman Web Lanjutan/Webserver/M03-npm (main)
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (m03-npm)
version: (1.0.0)
description: Minggu3
entry point: (index.js)
test command:
git repository:
keywords:
author: Sio
license: (ISC)
About to write to C:\Users\SIO\OneDrive - mikroskil.ac.id\Mikroskil\Mata Kuliah\6. Pemrograman Web Lanjutan\Webserver\M03-npm\package.json:

{
  "name": "m03-npm",
  "version": "1.0.0",
  "description": "Minggu3",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Sio",
  "license": "ISC"
}

Is this OK? (yes) █
```

Catatan:

- Pastikan telah menginstall Node.JS
- Buat folder baru: **M03-NPM**
- Pada folder tersebut, buka terminal dan ketikkan perintah: **npm init**
- **Enter** untuk Ok.

package.json pada NPM

- Untuk menggunakan package NPM, proyek harus memiliki file **package.json** yang terdapat metadata tertentu pada proyek.
- **Metadata** tersebut menunjukkan beberapa aspek dari proyek dengan urutan sebagai berikut:
 - Nama proyek
 - Versi
 - Deskripsi
 - Entry point
 - Test command
 - Repositori git
 - Keyword
 - Lisensi
 - Dependensi
 - devDependency
- Metadata dapat mempermudah identifikasi proyek dan berlaku juga sebagai dasar untuk user dalam mendapatkan informasi proyek.



Package.json pada NPM

Informasi project berisi **nama**, **versi**, dan **deskripsi**. Lalu di bagian script, berisi script bash atau command line untuk dieksekusi dengan NPM.

Selain properti-properti di atas, masih ada lagi properti lain:

- **dependencies** berisi keterangan modul atau library yang dibutuhkan aplikasi;
- **devDependencies** berisi keterangan modul atau library yang dibutuhkan untuk pengembangan aplikasi.



```
1  {
2    "name": "m03-npm",
3    "version": "1.0.0",
4    "description": "Minggu3",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Sio",
10   "license": "ISC"
11 }
```

Install Module dengan NPM

NPM memungkinkan user untuk menginstall modul atau library dengan perintah:

npm install <nama modul>

Cari nama package di: <https://www.npmjs.com/>

The screenshot shows the npmjs.com website with the package 'express' selected. The page includes a search bar with 'express' entered, a navigation bar with links like 'Products', 'Pricing', and 'Docun', and a main content area for the 'express' package. The package page displays the version '4.17.1', a 'Public' status, and a 'Published 2 years ago' timestamp. It also shows '30 Dependencies', '51.785 Dependents', and '264 Versions'. A large 'express' logo is prominently displayed. Below the logo, a description reads 'Fast, unopinionated, minimalist web framework for node.' and a series of badges indicate 'npm v4.17.1', 'downloads 74M/month', 'linux passing', 'windows passing', and 'coverage 100%'. A code block shows a snippet of JavaScript code for setting up an Express app. On the right side, there is an 'Install' section with the command '> npm i express', a 'Weekly Downloads' graph showing 16,806,950 downloads, and a table with package details.

Version	License
4.17.1	MIT

Unpacked Size	Total Files
208 kB	16

Issues	Pull Requests
107	59

Install Module dengan NPM

NPM memungkinkan user untuk menginstall modul atau library dengan perintah:

npm install <nama modul>

Cari nama package di: <https://www.npmjs.com/>

The screenshot shows the npmjs.com website for the 'express' package. At the top, there's a navigation bar with 'Products', 'Pricing', and 'Docun'. Below that, a search bar contains 'express' and a 'Search' button. A banner below the search bar says 'Wondering what's next for npm? [Check out our public roadmap!](#)'. The main content area features the 'express' package name with a 'DT' tag, version '4.17.1', and 'Public' status. It also shows 'Published 2 years ago'. Below this, there are links for 'Readme', 'Explore', '30 Dependencies', '51.785 Dependents', and '264 Versions'. The 'express' logo is prominently displayed. A description states 'Fast, unopinionated, minimalist web framework for node.' Below the description, there are badges for 'npm v4.17.1', 'downloads 74M/month', 'linux passing', 'windows passing', and 'coverage 100%'. A code block shows a snippet of JavaScript code for setting up an Express app. On the right side, there's an 'Install' section with the command '> npm i express'. Below that, a 'Weekly Downloads' chart shows a value of '16.806.950'. A table lists package details: Version (4.17.1), License (MIT), Unpacked Size (208 kB), Total Files (16), Issues (107), and Pull Requests (59).

npmjs.com/package/express

No Punches Made

Products Pricing Docun

npm

express

Search

Wondering what's next for npm? [Check out our public roadmap!](#)

express DT

4.17.1 • Public • Published 2 years ago

Readme Explore BETA 30 Dependencies 51.785 Dependents 264 Versions

express

Fast, unopinionated, minimalist web framework for **node**.

npm v4.17.1 downloads 74M/month linux passing windows passing coverage 100%

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

Install

> npm i express

Weekly Downloads

16.806.950

Version	License
4.17.1	MIT
Unpacked Size	Total Files
208 kB	16
Issues	Pull Requests
107	59

Install ExpressJs dengan NPM | Cont.

Untuk menginstall package ExpressJs dapat dilakukan dengan melihat instruksi perintah pada situs npmjs.com

Install

```
> npm i express
```

Weekly Downloads

16.806.950

Pada folder M03-npm, install express dengan perintah:

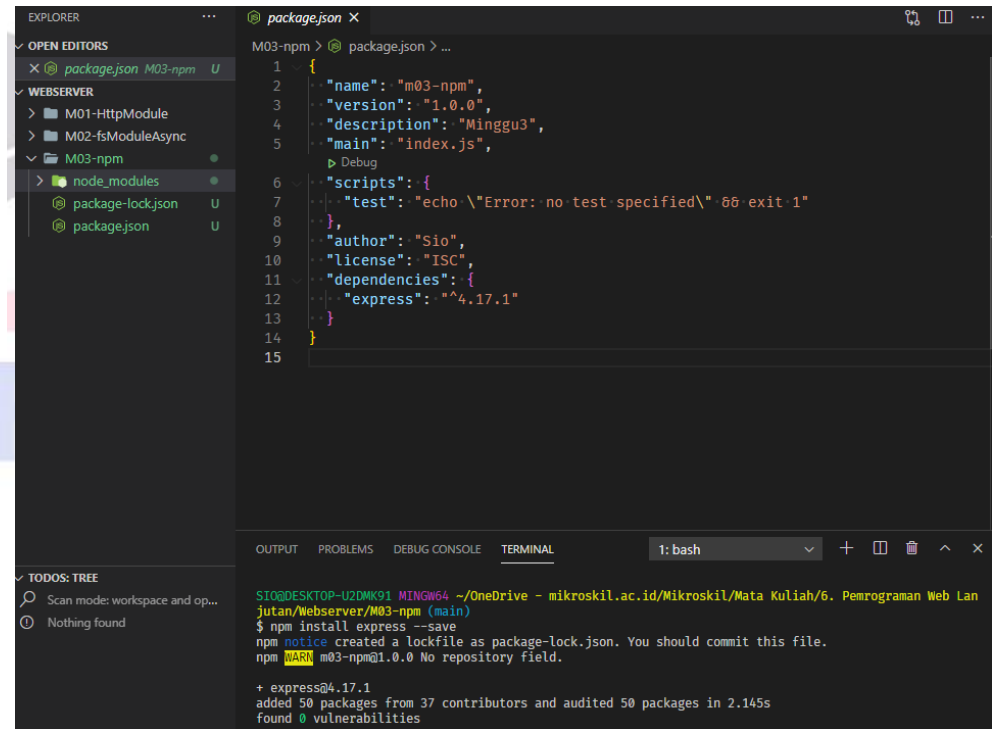
npm install express --save

Catatan: **--save**, berfungsi untuk menyimpan express dalam **dependencies list**.

Setelah install, pada package.json akan bertambah express beserta versinya. Juga folder **node_module**.

Baca:

<https://expressjs.com/en/starter/installing.html>



```
package.json
1 {
2   "name": "m03-npm",
3   "version": "1.0.0",
4   "description": "Minggu3",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "Sio",
10  "license": "ISC",
11  "dependencies": {
12    "express": "^4.17.1"
13  }
14 }
15
```

```
npm WARN m03-npm@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 2.145s
found 0 vulnerabilities
```

Cari tahu

1. **Bagaimana cara module / package untuk Dev dan Global**
2. **Bagaimana cara menghapus Package yang telah diinstall sebelumnya?**
3. **Bagaimana cara tambah scripts di package.json**
4. **Bagaimana cara run / mengeksekusi server http melalui NPM?**



ExpressJs

- **ExpressJS** merupakan **framework** minimal yang sangat fleksibel. Dapat digunakan untuk **membuat web server HTML, server file statik**, aplikasi chat, search engine, sosial media, layanan web dengan akses melalui REST API atau aplikasi hybrid yaitu selain pengguna mempunyai akses melalui REST API juga mempunyai akses ke HTML page.
- Untuk memudahkan pembuatan aplikasi web anda bisa menggunakan framework ExpressJS daripada harus menggunakan module http bawaan Node.js.
- Framework ini menawarkan beberapa **fitur seperti routing, rendering view** dan **mendukung middleware** dengan kata lain anda akan banyak menghemat waktu dalam pengembangan aplikasi Node.Js.

Source: <https://expressjs.com/en/starter/installing.html>



Server File - ExpressJs

Struktur file project ExpressJs adalah sebagai berikut:

- **node_modules**, di dalamnya terdapat berbagai folder library - library Node.js yang dipasang melalui node package manager atau NPM
- **publik**, kita dapat menaruh berbagai file HTML, CSS, Javascript, atau gambar di dalam folder ini.
- **routes**, di dalamnya terdapat berbagai file yang berisi action yang diterima oleh routing yang kita definisikan. Dapat menerima request berupa GET, POST, PUT, DELETE, OPTION, dan HEAD
- **views**, di dalamnya terdapat berbagai file jade atau html yang digunakan oleh routes untuk menampilkan halaman
- **app.js**, file utama Express.js yang berisi penggunaan package utama dan konfigurasi utama
- **npm-debug.log**, file yang berisi hasil debug atau error yang dicatat oleh Express.js selama running
- **package.json**, berisi struktur JSON yang mendefinisikan profil proyek dan dependensi paket apa saja yang dibutuhkan aplikasi yang kita kembangkan. NPM akan melacak dependensi dengan melihat file ini

```
server-file-statis
├─ package.json
├─ app.js
├─ node_modules
└─ publik
    └─ index.html
```

Server static dengan ExpressJs

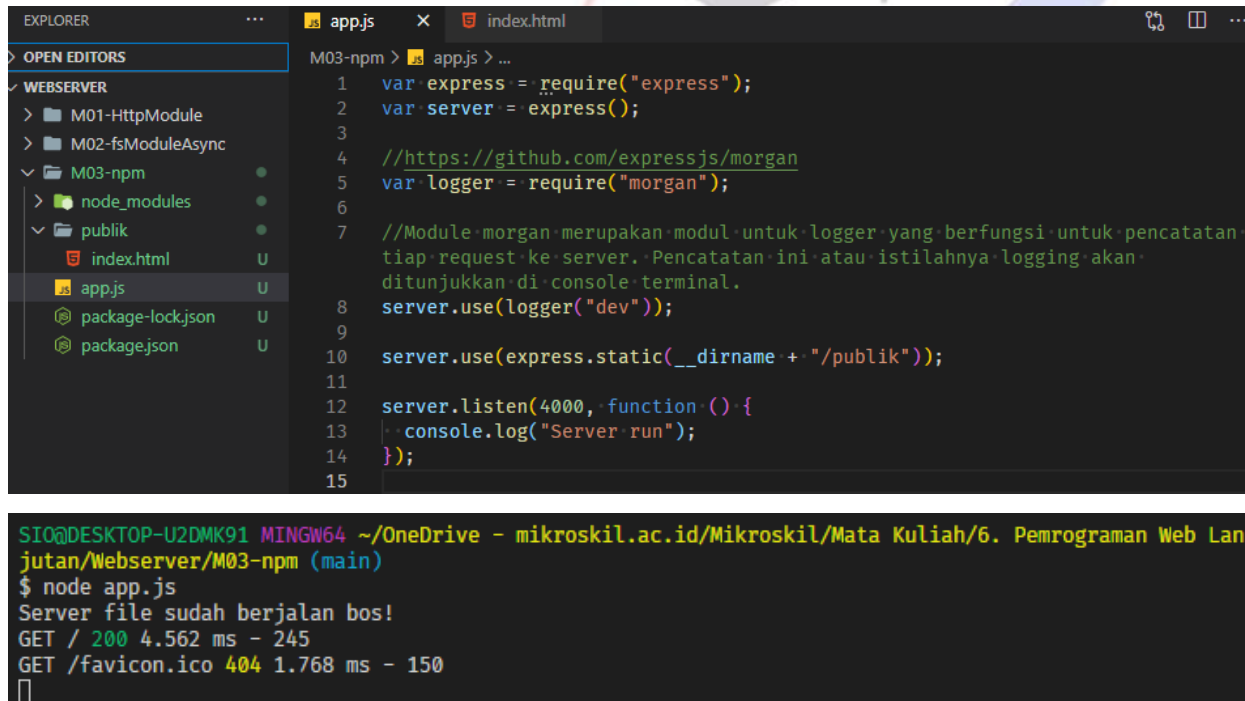
Server static ini akan menampilkan halaman index dari folder publik yaitu index.html

Pada server di bawah, digunakan **Module morgan** merupakan modul untuk logger yang berfungsi untuk **pencatatan tiap request ke server**.

Pencatatan ini atau istilahnya logging akan ditunjukkan di console terminal.

Install dengan: ***npm install morgan --save***

Run server: ***node app.js***



The screenshot shows a VS Code editor with a file explorer on the left and a code editor in the center. The file explorer shows a project structure with folders 'M01-HttpModule', 'M02-fsModuleAsync', and 'M03-npm'. The 'M03-npm' folder is expanded, showing 'node_modules', 'publik', 'index.html', 'app.js', 'package-lock.json', and 'package.json'. The 'app.js' file is selected. The code editor shows the following code:

```
1 var express = require("express");
2 var server = express();
3
4 //https://github.com/expressjs/morgan
5 var logger = require("morgan");
6
7 //Module morgan merupakan modul untuk logger yang berfungsi untuk pencatatan
8 //tiap request ke server. Pencatatan ini atau istilahnya logging akan
9 //ditunjukkan di console terminal.
10 server.use(logger("dev"));
11
12 server.use(express.static(__dirname + "/publik"));
13
14 server.listen(4000, function () {
15   console.log("Server run");
16 });
```

Below the code editor, a terminal window shows the output of running the server:

```
SIO@DESKTOP-U2DMK91 MINGW64 ~/OneDrive - mikroskil.ac.id/Mikroskil/Mata Kuliah/6. Pemrograman Web Lan
jutan/Webserver/M03-npm (main)
$ node app.js
Server file sudah berjalan bos!
GET / 200 4.562 ms - 245
GET /favicon.ico 404 1.768 ms - 150
```

Dengan menggunakan **Morgan** setiap aksi oleh user akan dicatat terminal.

Cari tahu

1. Bagaimana cara menjalankan server dengan npm? Contoh: `npm run dev`
2. Server REST dengan express js



ExpressJS: Routing, Basic Method Post, Content-Type Body Parser

Sio Jurnalís Pipin, S.Kom.



**STMIK - STIE
MIKROSKIL**

PRODI. TEKNIK INFORMATIKA (S-1)

Routing

- Routing mengatur bagaimana endpoints (URL) menanggapi permintaan klien, atau termasuk bagian-bagian dari aplikasi yang harus menangani permintaan spesifik.

Contoh:

```
app.get('/', (req, res) => { /* */ })
```

Penjelasan:

Router ini dapat di akses dari url root '/' menggunakan method get dan merespon dengan apa yang kita harapkan.

Route Parameters

- Parameter pada routing Express.js berfungsi untuk menangani permintaan spesifik dari klien melalui melalui segment URL untuk menerima **value** yang telah ditentukan posisi dalam URL.
- Nilai yang diambil diisi dalam objek **req.params**, dengan nama parameter rute yang ditentukan dalam jalur masing-masing.

```
Route path: /users/:userId/books/:bookId  
Request URL: http://localhost:3000/users/34/books/8989  
req.params: { "userId": "34", "bookId": "8989" }
```

Router ini dapat didefinisikan sebagai berikut:

```
app.get('/users/:userId/books/:bookId', function (req, res) {  
  res.send(req.params)  
})
```

Method Post dan Body parser

- Memungkinkan untuk mengirim data dalam jumlah besar
- Dikirim dalam **body** data.
- Lebih aman karena data disisipkan dalam **body** data request tidak melalui parameter url.
- Karena kita mengirim data melalui body, maka dibutuhkan package **body parse**
(<https://www.npmjs.com/package/body-parser>)
- body-parser merupakan sebuah **middleware** yang membaca input formulir dan menyimpannya sebagai **objek** javascript yang dapat diakses melalui **req.body**

Menggunakan Method Post

- Install body parse pada folder project: `npm i body-parser --save`

```
$ npm i body-parser --save
npm WARN m04-route@1.0.0 No repository field.

+ body-parser@1.19.0
updated 1 package and audited 55 packages in 1.727s
found 0 vulnerabilities
```

- Pada file index.js tambahkan method post

```
18 //route dengan method post
19 var data = bodyParser.urlencoded({ extended: false });
20 app.post("/api/datamhs", data, function (req, res) {
21   res.send(req.body);
22 });
```

- Pada file index.html buat sebuah form menuju url **datamhs**

```
<body>
<form action="http://127.0.0.1:4000/api/datamhs" method="POST">
  NIM: <input type="text" name="nim"><br>
  Nama: <input type="text" name="nama"><br>
  Kelas: <input type="text" name="kelas"><br>
  HP: <input type="text" name="hp">
  <input type="submit" value="Submit">
</form>
</body>
```

```
$ node index.js
Server run
POST /api/datamhs 200 26.477 ms - 72
```


ExpressJs Routing Query dan Params

Sio Jurnalis Pipin, S.Kom.



Routing: Query Parameter

- Ketika melakukan permintaan (request) terkadang bisa jadi **meminta**, atau juga **meminta sekaligus mengirim value** ke server yaitu melalui url, atau form (html).
- Berfungsi untuk mengirim dan mengambil value yang dikirim melalui request
- Merupakan Query String yang bagian dari URL (**Uniform Resource Locator**) setelah tanda tanya (?) atau garis miring (/). Ini bertujuan untuk mengirim sejumlah kecil informasi ke server melalui url. Informasi ini biasanya digunakan sebagai parameter untuk mengkueri database, atau mungkin untuk memfilter hasil.

Request Query

- Request query dalam Expressjs mengidentifikasi query (key) pada url, contoh:

<http://localhost:3000/search?nama=budi&nim=19111>

- Implementasi pada ExpressJs

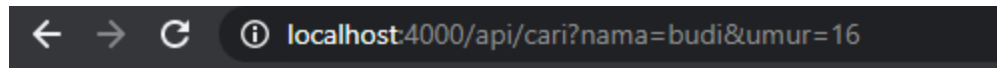
Pada index.js buat route:

```
21 app.get("/api/cari", function (req, res, next) {  
22   var nama = req.query.nama;  
23   console.log(`nama : ${nama}`);  
24   var umur = req.query.umur;  
25   console.log(`umur : ${umur}`);  
26   res.send(umur);  
27 });
```

```
SIO@DESKTOP-U2DMK91 MINGW64 ~/OneDrive - mikroskil.ac.id  
$ node index.js  
Server run  
nama : budi  
umur : 16  
GET /api/cari?nama=budi&umur=16 200 7.512 ms - 2
```

- Ini dapat di akses pada :

<http://localhost:4000/api/cari?nama=budi&umur=16>



Request Params

- Berfungsi untuk menangkap nilai yang dikirimkan melalui url yang mengirimkan nilai secara langsung tanpa key, contoh:

<http://localhost:3000/budi/19111>

- Implementasi pada ExpressJs

Pada index.js buat route:

```
//route dengan method get
app.get("/api/:nim/:nama", function (req, res) {
  res.statusCode = 200;
  //content-type pada expressjs
  res.setHeader("Content-Type", "text/plain");
  res.send(req.params);
});
```



- Ini dapat di akses pada :
- <http://localhost:4000/api/budi/16>

Middleware dan Request, Response, Next dan Error

Sio Jurnal Pipin, S.Kom.



Middleware

- Middleware merupakan fungsi yang mengaitkan ke dalam sebuah **routing process**, dan beberapa operasi di beberapa titik, tergantung pada apa yang ingin dilakukan.
- Biasanya digunakan untuk verifikasi permintaan atau respons, atau mengakhiri permintaan sebelum mencapai route tujuan / fungsi di **controller**.
- Verifikasi disini dapat berupa pengecekan cookies, token user, validasi url dst.
- Biasanya untuk kasus ini kita menggunakan
 - Morgan : <https://github.com/expressjs/morgan>
 - cookie-parser :
<http://expressjs.com/en/resources/middleware/cookie-parser.html>

Middleware | lanjutan

- Package Middleware tersebut Ini biasanya akan ditambahkan pada server yang digunakan dengan penulisan:

```
app.use((req, res, next) => { /* */ })
```

- Contoh:

```
M05-queryparams > . index.js > ...
1  var express = require("express");
2  var app = express();
3  var bodyParser = require("body-parser");
4
5  //https://github.com/expressjs/morgan
6  var logger = require("morgan");
7
```

```
SIO@DESKTOP-U2DMK91 MINGW64 ~/OneDrive - mikroskil.ac.id
$ node index.js
Server run
GET /api/budi/1222 200 4.761 ms - 28
GET /favicon.ico 404 1.269 ms - 150
```

- Saat ada user yang mengakses router maka middleware morgan yang berfungsi sebagai logger akan menampilkan informasi path url yang diakses oleh user beserta status code, dan informasi lainnya.

Middleware | lanjutan

- Selain itu, middleware dapat berupa fungsi yang sering digunakan untuk menjadi perantara, sekaligus verifikasi aktivitas sebelum menuju router.
- Middleware function dapat digunakan untuk router secara spesifik. Misal validasi role admin hanya pada router yang biasanya diakses oleh admin saja. Format:

```
const myMiddleware = (req, res, next) => {  
  /* ... */  
  next()  
}  
  
app.get('/', myMiddleware, (req, res) => res.send('Hello World!'))
```

Contoh: sebuah fungsi middleware untuk cek jwt token. Jika valid, maka fungsi **next()** akan mengizinkan lanjut pada router tujuan.

```
3 module.exports = (req, res, next) => {  
4   try {  
5     const token = req.headers.authorization.split(" ")[1];  
6     const decoded = jwt.verify(token, process.env.JWT_KEY);  
7     req.userData = decoded;  
8     next();  
9   } catch (error) {  
10    return res.status(401).json({ message: 'Auth failed' });  
11  }  
12 }
```


Req dan Res dalam Komunikasi Server

- ExpressJs memungkinkan untuk menangani Request dan Reponse secara manual.
- Proses Komunikasi server dengan ExpressJs yaitu:
 1. Memanggil *web server* yang telah disediakan NodeJS.
 2. Membaca URL yang ingin diakses pengguna, dan memanggil fungsi untuk memproses URL tersebut.
 3. Memproses HTTP Request dari *client*.
 4. Menghasilkan HTTP Response untuk *client*.
- **request**, yang merupakan instan dari ***http.IncomingMessage***, mewakili HTTP Request yang dikirimkan pengguna. Sebuah koneksi bisa saja menghasilkan beberapa HTTP Request. Kita akan membahas bagian ini lebih jauh nantinya.
- **response**, instan dari ***http.ServerResponse***, mewakili HTTP Response yang akan diberikan kepada pengguna.

Req, Res dan Next ()dalam Komunikasi Server | Lanjutan

```
app.use((req, res, next) => { /* */ })
```

- **Request** merupakan object HTTP request yang dikirim dari client ke server
- **Result** merupakan hasil result yang dikirim dari server ke client, ini adalah object yang kita gunakan untuk mengirim data ke client baik berupa data json, text biasa maupun html
- **Next** merupakan callback untuk melanjutkan proses life-cycle ke middleware berikutnya
- **Callback next()** merupakan callback untuk meneruskan proses atau data ke middleware berikutnya secara berurutan. Jika kita passing data / response ke error parameter maka proses tersebut akan terhenti di middleware tersebut dan mengirim data ke error handler yang dipasang di file utama, misal seperti app.js atau index.js.

Contoh Middleware

- Pada index.js buatlah sebuah middleware

```
12 //Middleware cek nim
13 const myMiddleware = (req, res, next) => {
14   if (req.params.nim === "123") {
15     console.log("Nim terverifikasi");
16     next();
17   } else {
18     const err = {
19       status: "error",
20       data: {
21         nim: req.params.nim,
22       },
23     };
24     next(err);
25   }
26 };
```

- Kemudian route dengan method get

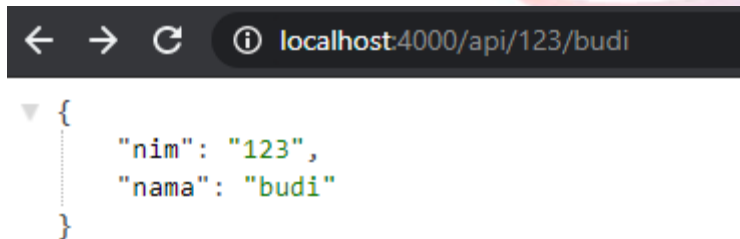
```
28 //route dengan method get
29 app.get("/api/:nim/:nama", myMiddleware, function (req, res) {
30   res.statusCode = 200;
31   //content-type pada expressjs
32   res.setHeader("Content-Type", "text/plain");
33   res.send(req.params);
34 });
```

- Dan fungsi untuk handel error sederhana

```
36 //ErrorHandling
37 app.use((error, req, res, next) => {
38   res.send(error);
39 });
```

Contoh Middleware

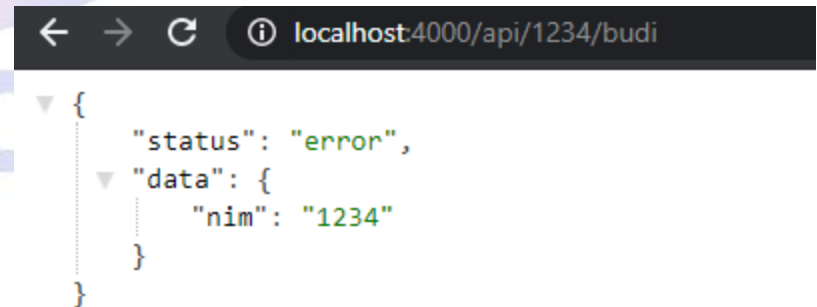
- Fungsi middleware yang telah dibuat sebelumnya akan mengecek nim yang dikirim oleh user melalui url. Jika nim sama dengan verifikasi pada middleware maka akan lanjut pada fungsi tujuannya.
- Jika middleware mengecek itu tidak valid maka akan dilempar keluar (hentikan proses) ke fungsi error handling yang telah kita buat.



A screenshot of a web browser's developer console showing a successful JSON response. The address bar displays `localhost:4000/api/123/budi`. The response body is a JSON object with the following structure:

```
{
  "nim": "123",
  "nama": "budi"
}
```

Requist Valid



A screenshot of a web browser's developer console showing an error JSON response. The address bar displays `localhost:4000/api/1234/budi`. The response body is a JSON object with the following structure:

```
{
  "status": "error",
  "data": {
    "nim": "1234"
  }
}
```

Tidak diproses ke fungsi tujuan