Belajar Coding Node.js

Ditulis oleh: RAKIFSUL

Pada tanggal: 2024-12-05

Daftar Isi

Belajar Node JS Cara Membuat Project

Source Code Project Ini

contoh_nodejs_create

Pendahuluan

Node.js adalah sebuah platform runtime environment yang bisa digunakan untuk banyak hal...

Salah satu manfaat dari Node.js adalah membuat backend dari aplikasi web.

Tapi, sebenarnya tidak hanya itu.

Secara keseluruhan, Node.js juga bisa digunakan untuk membuat aplikasi command line, bahkan menjadi bagian dari Electron js yang digunakan untuk aplikasi desktop GUI.

Saat Node.js diinstall, biasanya NPM juga ikut terinstall.

NPM itu sendiri adalah package manager yang fungsinya untuk memanajemen package-package yang digunakan dalam sebuah project Node.js, bahkan selain Node.js sekalipun.

Jadi tidak aneh jika React juga bisa menggunakan NPM untuk memanajemen package-package yang digunakan di dalamnya.

Dalam tutorial ini, kita akan membuat project Node.js dari nol dengan menggunakan npm.

Lebih Lanjut tentang Node.js

Node.js adalah lingkungan runtime JavaScript yang dibangun di atas mesin JavaScript V8 dari Google Chrome.

Berikut adalah beberapa poin penting tentang Node.js:

- JavaScript di Sisi Server: Node.js memungkinkan untuk menjalankan kode JavaScript di sisi server, yang sebelumnya terbatas pada browser. Ini memungkinkan pengembang untuk menggunakan JavaScript di kedua sisi, baik di sisi klien (browser) maupun di sisi server.
- Non-blocking dan Event-driven: Node.js menggunakan model pemrograman non-blocking dan event-driven yang memungkinkan untuk menangani banyak koneksi secara bersamaan dengan cepat dan efisien. Ini berguna dalam membangun aplikasi yang skala besar atau real-time seperti aplikasi web, streaming, dan permainan.
- Asynchronous I/O: Node.js memungkinkan untuk melakukan operasi I/O secara asinkron, yang berarti operasi-operasi seperti membaca atau menulis ke berkas, mengambil data dari database, atau memanggil API eksternal dapat dilakukan tanpa menghentikan eksekusi program. Hal ini mengoptimalkan penggunaan CPU dan mempercepat respons aplikasi.
- NPM (Node Package Manager): Node.js dilengkapi dengan NPM, manajer paket yang kuat untuk mengelola dependensi dan paket-paket JavaScript. NPM memungkinkan untuk dengan mudah menginstal, mengelola, dan membagikan kode JavaScript dalam proyek-proyek Node.js.
- Komunitas yang Besar: Node.js memiliki komunitas pengembang yang besar dan aktif yang terus berkontribusi dalam pengembangan, dokumentasi, dan sumber daya belajar. Ini membuat Node.js menjadi salah satu ekosistem pengembangan yang paling populer dan berkembang pesat di dunia.

- Fleksibilitas dan Skalabilitas: Node.js cocok digunakan untuk berbagai jenis aplikasi, mulai dari aplikasi web, aplikasi backend, server API, hingga aplikasi real-time seperti chat atau game. Dengan model non-blocking dan event-driven, Node.js juga dapat dengan mudah di-skala secara horizontal untuk menangani beban yang lebih besar.
- Cross-platform: Node.js dapat dijalankan di berbagai platform, termasuk Windows, macOS, dan Linux. Ini membuatnya menjadi pilihan yang fleksibel untuk pengembangan aplikasi yang berjalan di berbagai lingkungan.

Secara keseluruhan, Node.js adalah lingkungan runtime yang kuat dan fleksibel untuk menjalankan kode JavaScript di sisi server.

Dengan model pemrograman non-blocking dan event-driven, serta dukungan dari NPM dan komunitas yang besar, Node.js menjadi pilihan yang populer untuk pengembangan aplikasi modern.

Lebih Lanjut tentang NPM

NPM (Node Package Manager) adalah manajer paket untuk ekosistem Node.js yang memungkinkan untuk mencari, menginstal, dan mengelola paket-paket JavaScript dalam proyek-proyek Node.js.

Berikut adalah beberapa poin penting tentang NPM:

- Manajer Paket: NPM adalah manajer paket yang kuat yang digunakan untuk mengelola dependensi dan paket-paket JavaScript dalam proyek-proyek Node.js. Ini memungkinkan pengembang untuk mengatur dan menyimpan kode JavaScript dari berbagai sumber dengan mudah.
- Repository Publik: NPM memiliki repositori publik yang luas yang berisi ribuan paket-paket JavaScript yang tersedia untuk digunakan. Pengembang dapat mencari dan menemukan paket yang sesuai dengan kebutuhan proyek mereka di repositori publik NPM.
- Instalasi Paket: NPM memungkinkan untuk menginstal paket-paket JavaScript dalam proyek dengan mudah menggunakan perintah npm install. NPM akan mengunduh dan menginstal paket beserta dependensinya ke dalam direktori proyek secara otomatis.
- Manajemen Versi: NPM memungkinkan untuk mengelola versi paket dengan mudah. Pengembang dapat menginstal versi spesifik dari paket, memperbarui paket ke versi terbaru, atau mengunci versi paket untuk memastikan konsistensi dalam proyek.
- Scripting: NPM memungkinkan untuk menambahkan skrip-sript kustom ke dalam berkas package.json yang mempermudah dalam menjalankan tugas-tugas tertentu, seperti kompilasi kode, menjalankan uji coba, atau memulai server pengembangan.
- Paket Lokal dan Global: NPM memungkinkan untuk menginstal paket-paket secara lokal dalam proyek atau secara global di seluruh sistem. Pengembang dapat memilih untuk menggunakan paket dalam ruang lingkup proyek atau membuatnya tersedia secara global di seluruh sistem.
- Kemudahan Kolaborasi: NPM memfasilitasi kolaborasi dan berbagi kode antar pengembang. Pengembang dapat dengan mudah berbagi proyek dan dependensinya dengan orang lain menggunakan file package.json dan repositori publik NPM.

Secara keseluruhan, NPM adalah alat yang sangat penting dalam ekosistem Node.js yang memungkinkan untuk mencari, menginstal, dan mengelola paket-paket JavaScript dengan mudah dalam proyek-proyek Node.js.

Ini membantu meningkatkan produktivitas dan memperluas fungsionalitas proyek dengan menggunakan kode dari komunitas yang luas dan beragam.

Tujuan

Berikut ini adalah tujuan dari tutorial ini:

- Pembaca mampu membuat project Node.js dari nol.
- Pembaca mengenal Node.js dan NPM.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.
- Bisa meng-akses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, pastikan Anda telah meng-install Node.js dan NPM.

Selanjutnya, buatlah folder bernama "contoh_nodejs_create".

Sebenarnya, nama foldernya bisa apapun, tapi dalam tutorial ini, gunakan saja nama tersebut agar tutorial ini lebih mudah dipahami.

Sekarang, buka folder "contoh_nodejs_create" dengan command line.

Kemudian, jalankan:

```
npm init -y
```

Nanti, akan muncul file bernama "package.json" yang isinya seperti ini:

```
{
  "name": "contoh_nodejs_create",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Selanjutnya, Anda bisa membuat file baru bernama "index.js" di dalam folder tersebut.

Alternatifnya, Anda bisa ganti "index.js" dengan "yanglain.js" di file "package.json" dan membuat file "yanglain.js" bukannya "index.js" di dalam folder "tutorial-0".

Dengan kata lain, bebas-bebas saja...

Sekarang, buka file "index.js" dengan teks editor, kemudian isi dengan kode ini:

```
// mem-print teks "hello world".
console.log("hello world");
```

Selanjutnya, jalankan:

```
node index.js
```

Nanti akan muncul output:

```
hello world
```

Sampai di sini langkah-langkahnya selesai.

Pembahasan

Pertama-tama, kita telah selesai membuat project dengan perintah:

```
npm init -y
```

Perintah tersebut akan membuatkan "package.json" dengan isi default.

Jika tanpa "-y":

```
npm init
```

Nanti Anda akan melihat prompt untuk melakukan perubahan isi dari "package.json".

Namun, jika Anda ingin mengubah isi dari "package.json", sebenarnya Anda juga bisa menggunakan teks editor setelah "package.json" dibuat.

Jadi, bebas-bebas sajalah...

Sekarang, kita bahas isi dari file "index.js".

Script "index.js" yang berisi:

```
// mem-print teks "hello world".
```

```
console.log("hello world");
```

Tidak mem-print:

```
// mem-print teks "hello world".
```

Tapi hanya:

```
hello world
```

Saja. Itu karena karakter "//" gunanya adalah untuk menandai komentar dan setiap komentar tidak dieksekusi.

Jadi, tanda "//" berfungsi ganda: sebagai komentar dan untuk menonaktifkan kode.

Setelah project dibuat, Anda bisa dengan aman meng-install package-package dari NPM.

Karena, daftarnya akan dimasukkan ke file "package.json".

Tepatnya di property "dependencies" jika menggunakan perintah:

```
npm install <nama-package>
```

Dan di property "devDependencies" jika menggunakan perintah:

```
npm install <nama-package> --save-dev
```

Tapi ingat, perintah tadi harus dijalankan dalam folder project, yakni yang ada file "package.json"-nya.

Penutup

Sekarang, seharusnya Anda telah mengenal Node.js, NPM, dan cara menggunakannya.

Membuat project Node.js seharusnya tidak masalah sekarang.

Belajar Node JS Menggunakan Nodemon

Source Code Project Ini

contoh_nodejs_nodemon

Pendahuluan

Dalam membangun aplikasi Node.js, bukan tidak mungkin akan ada perubahan dalam script Node.js yang kita buat.

Akan tetapi, perubahan kode tersebut hanya bisa dilihat jika aplikasi Node.js-nya di-restart.

Mungkin tidak masalah jika aplikasi yang kita buat berjalan selewat saja, seperti misalnya aplikasi command line.

Namun, dalam aplikasi web, biasanya akan ada infinite loop untuk melakukan listen port tertentu.

Artinya, jika aplikasi semacam tadi mengalami perubahan kode, kita harus menghentikan aplikasi tersebut, kemudian menjalankannya lagi.

Jika itu dilakukan manual dan secara sering, tentunya akan merepotkan.

Di sinilah nodemon dapat membantu kita.

nodemon adalah salah satu package NPM yang fungsinya mengawasi script-script dalam project Node.js.

Setelah nodemon dijalankan, maka jika ada script yang diawasinya berubah, kemudian di-save, maka nodemon akan secara otomatis me-restart aplikasi.

Lebih Lanjut tentang Nodemon

Nodemon adalah alat pengembangan (development tool) yang berguna untuk memonitor perubahan pada berkas dalam proyek Node.js dan secara otomatis memulai ulang server atau aplikasi Node.js ketika ada perubahan.

Berikut adalah beberapa poin penting tentang Nodemon:

- Memantau Perubahan Berkas: Nodemon secara terus-menerus memantau berkas dalam proyek Node.js, seperti berkas JavaScript atau JSON. Ini dilakukan dengan memeriksa waktu modifikasi (timestamp) pada berkas-berkas tersebut.
- Pemulihan Otomatis: Ketika Nodemon mendeteksi perubahan pada berkas, seperti penyimpanan baru atau penyuntingan, ia akan secara otomatis memulai ulang server Node.js atau aplikasi yang sedang berjalan. Ini memungkinkan untuk melihat perubahan secara langsung tanpa perlu memulai ulang server secara manual setiap kali ada perubahan pada kode.
- Mempercepat Proses Pengembangan: Nodemon membantu meningkatkan produktivitas pengembangan dengan mempercepat siklus pengembangan. Pengembang tidak perlu lagi menghentikan dan memulai ulang server secara manual setiap kali ada perubahan pada kode, yang dapat menghemat waktu dan mengurangi gangguan.
- Konfigurasi Mudah: Nodemon dapat dikonfigurasi dengan mudah menggunakan berkas konfigurasi atau opsi baris perintah. Ini memungkinkan untuk menyesuaikan perilaku Nodemon sesuai dengan kebutuhan proyek, seperti menambahkan pengecualian untuk berkas tertentu atau menyesuaikan interval pemantauan.
- Dukungan untuk Berbagai Jenis Proyek: Nodemon dapat digunakan dengan berbagai jenis proyek Node.js, termasuk proyek aplikasi web, server API, atau aplikasi terpisah lainnya. Ini membuatnya menjadi alat yang serbaguna untuk pengembangan Node.js.

Secara keseluruhan, Nodemon adalah alat yang berguna dan populer dalam ekosistem pengembangan Node.js karena kemampuannya untuk memantau perubahan kode secara otomatis dan memulai ulang server dengan cepat.

Ini membantu meningkatkan produktivitas pengembangan dan memudahkan proses pengembangan aplikasi Node.js.

Tujuan

Berikut ini adalah tujuan dari tutorial ini:

- Pembaca mampu menggunakan nodemon.
- Pembaca mengenal nodemon.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.
- Bisa meng-akses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, buatlah project Node.js bernama "contoh_nodejs_nodemon" dengan cara yang telah dijelaskan di "Belajar Node JS Cara Membuat Project".

Selanjutnya, ubah file "package.json" menjadi seperti ini:

```
{
  "name": "contoh_nodejs_nodemon",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon -e js,html -w ./ index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Selanjutnya, isi file "index.js" dengan kode ini:

```
const express = require("express");
const app = express();

app.get("/", (req, res, next) => {
    res.sendFile(__dirname + "/views/" + "index.html");
});

app.listen(3000, () => {
```

```
    console.log("Server berjalan di port 3000");
});
```

Selanjutnya buat folder "views" yang di dalamnya ada file "index.html".

Isi file "index.html" adalah seperti ini:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Belajar Nodemon</title>
</head>

<body>
    <h1>Belajar Nodemon</h1>
</body>

</html>
```

Selanjutnya, install express:

```
npm install express
```

Dan install nodemon sebagai dev dependencies:

```
npm install nodemon --save-dev
```

Nanti, di "package.json" akan ada tambahan entry:

```
"dependencies": {
    "express": "^4.17.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.7"
  }
```

Catat bahwa versinya mungkin berbeda dengan "package.json" milik saya.

Sekarang, jalankan aplikasi ini:

```
npm run dev
```

Selanjutnya, ubah file "index.js", kemudian save, kemudian perhatikan command line, akan terlihat bahwa aplikasi ini di-restart

Itu juga akan terjadi jika kita mengubah dan men-save "index.html".

Pembahasan

Pertama-tama, kita membuat project Node.js dan mengisinya dengan beberapa script.

Karena kita telah menginstall nodemon, maka perintah node diganti dengan nodemon, sesuai dengan yang ada di "package.json":

```
"dev": "nodemon -e js,html -w ./ index.js"
```

Menurut kode itu, saat perintah npm run dev dijalankan, semua file berekstensi ".js" dan ".html" yang ada di folder project dan subfoldernya akan diawasi.

Jika terjadi perubahan, maka aplikasi akan di-restart.

Dengan kata lain, parameter "-e" menentukan ekstensi apa yang diawasi dan parameter "-w" menentukan folder mana yang diawasi.

Penutup

Sekarang, seharusnya Anda telah mengenal nodemon dan cara menggunakannya.

Belajar Node JS Hello World Tanpa Framework

Source Code Project Ini

contoh_nodejs_hello_world_tanpa_framework

Pendahuluan

Pada umumnya, saat kita akan membuat aplikasi web dengan Node.js, maka setidaknya kita akan menyiapkan package-package 3rd party dari NPM.

Misalnya saja, Express.

Namun, dengan menggunakan Express, sebenarnya kita telah banyak menyembunyikan cara kerja Node.js di belakang layar.

Walaupun sebenarnya aplikasi hello world bisa dibuat dengan menggunakan Express, ada baiknya juga jika kita tahu caranya tanpa menggunakan framework apapun.

Pada aplikasi ini, kita akan belajar cara meng-import modul, membuat server http, merespons http request dan mengakhirinya, serta membuat server tadi untuk berjalan di port tertentu.

Manfaat Membuat Hello World tanpa Framework di Node.js

Membuat program "Hello, World!" tanpa menggunakan framework di Node.js memiliki beberapa manfaat:

- Pemahaman Dasar Node.js: Membuat program sederhana tanpa menggunakan framework memungkinkan untuk memahami dasar-dasar Node.js dengan lebih baik. Ini termasuk memahami bagaimana membuat, menjalankan, dan mengelola proses Node.js secara langsung tanpa adanya lapisan abstraksi yang ditambahkan oleh framework.
- Kontrol Penuh: Tanpa menggunakan framework, pengembang memiliki kontrol penuh atas kode yang ditulis. Ini memungkinkan untuk menyesuaikan kode sesuai dengan kebutuhan proyek secara spesifik, tanpa terikat oleh keputusan desain atau struktur yang diberlakukan oleh framework tertentu.
- Keterampilan Pemrograman Umum: Membuat program sederhana tanpa menggunakan framework memungkinkan untuk fokus pada keterampilan pemrograman umum, seperti manipulasi string, manipulasi array, logika pengaturan aliran, dan lain-lain. Ini merupakan fondasi yang kuat untuk memahami dan mengembangkan keterampilan pemrograman Node.js secara lebih luas.
- Pengenalan Proses Asinkron: Node.js terkenal karena model pemrograman asinkronnya yang kuat. Dengan membuat program "Hello, World!" tanpa framework, pengembang dapat memahami dasar-dasar pemrograman asinkron di Node.js, termasuk penggunaan callback, promise, atau async/await secara langsung.
- Ringan dan Efisien: Tanpa menggunakan framework, program "Hello, World!" cenderung lebih ringan dan efisien karena tidak ada lapisan abstraksi tambahan yang perlu dimuat atau dieksekusi. Ini membuat program lebih cepat untuk dijalankan dan membutuhkan sumber daya yang lebih sedikit.

Meskipun membuat program "Hello, World!" tanpa framework mungkin terlihat sederhana, namun itu memberikan kesempatan yang berharga untuk memahami dasar-dasar Node.js secara lebih mendalam dan membangun dasar yang kuat untuk pengembangan aplikasi yang lebih kompleks di masa depan.

Tujuan

Berikut ini adalah tujuan dari tutorial ini:

- Pembaca mampu membuat aplikasi hello world tanpa menggunakan framework.
- Pembaca mengenal modul http milik Node.js.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.
- Bisa meng-akses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, buatlah project Node.js bernama "contoh_nodejs_hello_world_tanpa_framework" dengan cara yang telah dijelaskan di "Belajar Node JS Cara Membuat Project".

Selanjutnya, ubah file "package.json" menjadi seperti ini:

```
{
  "name": "contoh_nodejs_hello_world_tanpa_framework",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "node index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Selanjutnya, isi file "index.js" dengan kode ini:

```
// import module http
const http = require("http");

// buat server. dan apapun request nya...
const server = http.createServer((req, res) => {
    // response dengan teks hello world. kemudian akhiri
    res.write("Hello World!");
    res.end();
});

// jalankan server di port 3000
server.listen(3000, () => {
    console.log("Server berjalan di port 3000.");
});
```

Sekarang, jalankan aplikasi ini:

```
npm run dev
```

Buka web browser Anda di:

http://localhost:3000

Nanti akan muncul output:

```
Hello World!
```

Di browser Anda.

Pembahasan

Pertama-tama, kita membuat project Node.js dan mengisinya dengan beberapa script.

Pada "index.js", kita mulai dari mengimpor modul http:

```
// import module http
const http = require("http");
```

Pada kode di atas, kita mengimpor modul http dan menyimpannya dalam const http.

Node.js menyediakan function require untuk meng-import modul.

Jika modul yang di-import adalah modul bawaan, maka kita tidak perlu meng-install-nya terlebih dahulu dengan npm.

Selain itu, jika modul yang di-import bukan modul buatan sendiri, kita tidak perlu menggunakan path pada require, cukup nama modulnya saja

Kemudian, kita membuat http server dengan cara ini:

```
// buat server. dan apapun request nya...
const server = http.createServer((req, res) => {
    // response dengan teks hello world. kemudian akhiri
    res.write("Hello World!");
    res.end();
});
```

req merupakan parameter untuk request.

res merupakan parameter untuk response.

Berhubung response-nya adalah sama untuk request apapun, maka, di sini saya tidak melakukan filtering terhadap request yang bisa dilakukan menggunakan parameter req.

Selanjutnya, kita menjalankan server tersebut dengan cara melakukan listen di port 3000:

```
// jalankan server di port 3000
server.listen(3000, () => {
    console.log("Server berjalan di port 3000.");
});
```

Dalam keadaan ini, port 3000 sedang digunakan oleh aplikasi ini.

Jadi, jika Anda menjalankan aplikasi lain yang harus menggunakan port 3000, maka kemungkinan aplikasi lain tersebut akan tidak berjalan.

Hal sebaliknya juga berlaku.

Jika sebelum aplikasi ini dijalankan port 3000 sedang digunakan, maka aplikasi ini juga tidak akan berjalan.

Lalu, kenapa kita mengakses aplikasi ini di:

http://localhost:3000

?

Itu karena, port 3000 adalah port yang kita gunakan untuk server http di aplikasi ini.

Adapun domain dari komputer lokal kita adalah localhost.

Selain itu, kita juga menggunakan protokol http dalam aplikasi ini.

Maka, yang perlu diakses browser untuk melihat output aplikasi ini adalah:

http://localhost:3000

Penutup

Sekarang, seharusnya Anda telah mengenal modul http dan cara menggunakannya.

Belajar Node JS Hello World Dengan Express

Source Code Project Ini

contoh_nodejs_hello_world_dengan_express

Pendahuluan

Membuat aplikasi server dengan Node.js tanpa framework itu memungkinkan.

Seperti apa yang dijelaskan pada "Belajar Node JS Hello World Tanpa Framework".

Namun, dengan menggunakan Express, sebenarnya kita telah banyak menghemat waktu penulisan kode.

Itu karena dengan menggunakan Express, banyak hal yang kompleks dilakukan di balik layar.

Pada tutorial sebelumnya, aplikasi yang kita buat masih sederhana.

Jadi, mungkin saja kode yang ditulis hanya sedikit.

Lalu, bagaimana jika aplikasi yang kita buat cukup besar?

Tanpa menggunakan framework tentunya akan merepotkan.

Oleh karena itu, mempelajari framework Node.js seperti Express menurut saya cukup bermanfaat.

Pada tutorial ini, saya akan membahas cara membuat aplikasi hello world dengan Node.js dan Express.

Tujuan

Berikut ini adalah tujuan dari tutorial ini:

- Pembaca mampu membuat aplikasi hello world menggunakan Express.

- Pembaca mengenal Express.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.
- Bisa meng-akses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, buatlah project Node.js bernama "contoh_nodejs_hello_world_dengan_express" dengan cara yang telah dijelaskan di "Belajar Node JS Cara Membuat Project".

Selanjutnya, ubah file "package.json" menjadi seperti ini:

```
{
    "name": "contoh_nodejs_hello_world_dengan_express",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "dev": "node index.js"
    },
    "keywords": [],
    "author": "",
    "license": "ISC",
    "dependencies": {
        "express": "^4.17.1"
    }
}
```

Selanjutnya, isi file "index.js" dengan kode ini:

```
// import module express
const express = require("express");

// inisialisasi express
const server = express();

// handle request "/"
server.get("/", (req, res, next) => {
    // response dengan teks "Hello World!", kemudian akhiri
    res.write("Hello World!");
    res.end();
});

// jalankan server di port 3000
server.listen(3000, () => {
```

```
    console.log("Server berjalan di port 3000.");
});
```

Selanjutnya, jalankan:

```
npm install
```

Sekarang, jalankan aplikasi ini:

```
npm run dev
```

Buka web browser Anda di:

http://localhost:3000/

Nanti akan muncul output:

```
Hello World!
```

Di browser Anda.

Pembahasan

Pertama-tama, kita membuat project Node.js dan mengisinya dengan beberapa script.

Selanjutnya, kita menjalankan perintah npm install tanpa diikuti nama package.

Itu maksudnya adalah untuk meng-install seluruh dependencies yang tertulis di "package.json", dalam hal ini hanya Express.

Pada "index.js", kita mulai dari mengimpor modul express:

```
// import module express
const express = require("express");
```

Pada kode di atas, kita mengimpor modul express dan menyimpannya dalam const express.

Node.js menyediakan function require untuk meng-import modul.

Jika modul yang di-import adalah modul pihak ke-3, maka kita perlu meng-install-nya terlebih dahulu dengan npm. Kita telah melakukannya dengan npm install tadi.

Selain itu, jika modul yang di-import bukan modul buatan sendiri, kita tidak perlu menggunakan path pada require, cukup nama modulnya saja

Pada baris kode selanjutnya, kita menginisialisasi dan membuat objek express:

```
// inisialisasi express
const server = express();
```

Objek tersebut disimpan pada const server.

Selanjutnya, server melakukan filter terhadap request GET ke path "/" atau root:

```
// handle request "/"
server.get("/", (req, res, next) => {
    // response dengan teks "Hello World!", kemudian akhiri
    res.write("Hello World!");
    res.end();
});
```

Request tadi direspon dengan teks "Hello World!", kemudian diakhiri dengan res.end().

Yang perlu Anda ingat di kode tadi adalah bahwa kode tadi bersifat pasif.

Dengan kata lain, kita tidak memerintahkan server untuk membuka browser client dan menampilkan teks "Hello World!", melainkan menunggu ada request dari browser client, dan jika request itu datang, maka tangani dengan callback di parameternya, sehingga teks "Hello World!" akan tampil.

Selanjutnya, kita menjalankan server tersebut dengan cara melakukan listen di port 3000:

```
// jalankan server di port 3000
server.listen(3000, () => {
    console.log("Server berjalan di port 3000.");
});
```

Dalam keadaan ini, port 3000 sedang digunakan oleh aplikasi ini.

Jadi, jika Anda menjalankan aplikasi lain yang harus menggunakan port 3000, maka kemungkinan aplikasi lain tersebut akan tidak berjalan.

Hal sebaliknya juga berlaku.

Jika sebelum aplikasi ini dijalankan port 3000 sedang digunakan, maka aplikasi ini juga tidak akan berjalan.

Lalu, kenapa kita mengakses aplikasi ini di:

http://localhost:3000/

?

Itu karena, port 3000 adalah port yang kita gunakan untuk server express di aplikasi ini.

Adapun domain dari komputer lokal kita adalah localhost.

Selain itu, kita juga menggunakan protokol http dalam aplikasi ini.

Adapun penambahan "/" di akhir URL berarti browser melakukan request terhadap path "/" di server express.

Maka, yang perlu diakses browser untuk melihat output aplikasi ini adalah:

http://localhost:3000/

Penutup

Sekarang, seharusnya Anda telah mengenal Express dan cara menggunakannya.

Belajar Node JS Menggunakan Express Middleware

Source Code Project Ini

contoh_nodejs_express_middleware

Pendahuluan

Dalam menangani request dan response, Express memberi kita kesempatan untuk melakukan sesuatu di antara keduanya.

Caranya adalah dengan menggunakan middleware.

Middleware merupakan perantara antara request yang masuk dan response yang keluar.

Pada prakteknya, middleware bisa digunakan untuk autentikasi, logging, validasi data, dan lain-lain.

Secara umum ada dua jenis middleware di Express.

- Middleware global
- Middleware lokal

Middleware global berlaku untuk seluruh request handler, sedangkan middleware lokal hanya berlaku pada request handler yang mendaftarkannya.

Dalam tutorial ini saya akan membahas keduanya.

Lebih Lanjut tentang Express Middleware

Express.js, sebagai kerangka kerja web yang populer untuk Node.js, memperkenalkan konsep yang kuat yang dikenal sebagai middleware.

Middleware adalah serangkaian fungsi yang berurutan yang memiliki akses ke objek permintaan (request) dan objek tanggapan (response) dalam siklus hidup permintaan HTTP.

Dalam bagian ini, kita akan fokus pada konsep, jenis-jenis middleware, dan bagaimana mereka berinteraksi dengan proses pengelolaan permintaan.

Pengertian Middleware

Middleware, dalam konteks Express.js, bertindak sebagai perantara antara permintaan yang diterima oleh server dan tanggapan yang dikirimkan oleh server. Middleware dapat digunakan untuk melakukan berbagai tugas, seperti memodifikasi objek permintaan atau objek tanggapan, menjalankan fungsi tertentu sebelum atau setelah pengolahan permintaan, atau bahkan menghentikan proses pengolahan permintaan.

Urutan Eksekusi Middleware

Eksekusi middleware dalam Express.js mengikuti urutan tertentu.

Ketika server menerima permintaan, middleware dijalankan berdasarkan urutan yang didefinisikan dalam kode.

Setiap middleware dapat memutuskan untuk melanjutkan eksekusi ke middleware berikutnya atau menghentikan proses dan mengirimkan tanggapan.

Fungsi Utama Middleware

Mengubah Objek Permintaan dan Tanggapan

Middleware dapat memodifikasi objek permintaan (request) dan objek tanggapan (response) sebelum atau setelah pengolahan permintaan utama. Contohnya, middleware dapat menambahkan informasi tambahan ke objek permintaan atau menetapkan header khusus di objek tanggapan.

Melakukan Tugas Tertentu

Middleware dapat menjalankan tugas tertentu sebelum atau setelah pengolahan permintaan utama. Ini bisa termasuk verifikasi keamanan, autentikasi pengguna, atau logging. Contohnya, middleware dapat memeriksa apakah pengguna telah mengautentikasi sebelum membiarkan mereka mengakses rute tertentu.

Manipulasi Aliran Kontrol

Middleware dapat memanipulasi aliran kontrol dalam proses pengolahan permintaan. Middleware dapat memutuskan apakah permintaan harus melanjutkan ke middleware berikutnya atau dihentikan. Contohnya, middleware dapat menentukan apakah pengguna memiliki hak akses untuk mengakses sumber daya tertentu dan memutuskan apakah permintaan harus diizinkan atau ditolak.

Penanganan Error

Middleware dapat menangani kesalahan yang terjadi selama proses pengolahan permintaan. Middleware khusus yang disebut middleware penanganan kesalahan dapat menangkap dan mengelola kesalahan yang tidak tertangani. Contohnya, middleware penanganan kesalahan dapat mengirim tanggapan kesalahan yang sesuai kepada klien.

Jenis-jenis Middleware dalam Express.js

Middleware Tingkat Aplikasi (Application-level Middleware)

Middleware ini beroperasi pada seluruh aplikasi dan didefinisikan menggunakan app.use(). Contoh: Middleware logger yang mencatat setiap permintaan ke konsol.

Middleware Tingkat Router (Router-level Middleware)

Middleware ini beroperasi pada tingkat router dan didefinisikan menggunakan router.use(). Contoh: Middleware otentikasi yang memeriksa keaslian pengguna sebelum mengizinkan akses ke rute tertentu.

Middleware Tanggapan (Error-handling Middleware)

Middleware ini menangani kesalahan yang mungkin terjadi selama proses pengolahan permintaan. Didefinisikan dengan fungsi yang menerima parameter kesalahan, permintaan, tanggapan, dan fungsi berikutnya (next).

Middleware Built-in (Built-in Middleware)

Express.js menyertakan beberapa middleware bawaan yang dapat digunakan tanpa instalasi tambahan. Contoh: Middleware express.static untuk menyajikan berkas statis seperti gambar atau file CSS.

Middleware Pihak Ketiga (Third-party Middleware)

Middleware yang dibuat oleh pihak ketiga dan dapat diintegrasikan dengan Express.js. Contoh: body-parser untuk menangani data yang dikirimkan melalui formulir HTML.

Implementasi Umum Middleware dalam Express.js

Logger Middleware

Middleware logger dapat digunakan untuk mencatat informasi seperti metode HTTP, URL yang diminta, dan waktu permintaan. Berguna untuk pemantauan dan debugging.

Middleware Otentikasi

Middleware otentikasi dapat memeriksa apakah pengguna telah diotentikasi sebelum membiarkan mereka mengakses rute tertentu. Menawarkan perlindungan terhadap akses yang tidak sah.

Middleware Pengecekan Izin

Middleware ini dapat memeriksa izin pengguna sebelum mengizinkan mereka mengakses sumber daya tertentu. Memastikan bahwa pengguna hanya dapat mengakses bagian dari aplikasi yang sesuai dengan izin mereka.

Middleware Penanganan Kesalahan

Middleware penanganan kesalahan dapat menangkap dan mengelola kesalahan yang terjadi selama proses pengolahan permintaan. Memberikan tanggapan kesalahan yang baik kepada klien.

Langkah-langkah Implementasi Middleware

Pengenalan Middleware

Identifikasi tujuan dan fungsionalitas middleware yang akan diimplementasikan. Tentukan apakah middleware akan beroperasi pada tingkat aplikasi, tingkat router, atau sebagai penangan kesalahan.

Pendaftaran Middleware

Jika menggunakan middleware pihak ketiga, instal dan daftarkan middleware tersebut menggunakan npm atau cara instalasi yang sesuai. Jika membuat middleware sendiri, definisikan fungsi middleware dan siapkan kode untuk pendaftaran middleware.

Pengaturan Middleware

Terapkan pengaturan middleware dengan menggunakan app.use() untuk middleware tingkat aplikasi atau router.use() untuk middleware tingkat router. Tentukan urutan eksekusi jika relevan, karena urutan dapat memengaruhi hasil akhir.

Integrasi Middleware dengan Rute

Terapkan middleware pada rute tertentu dengan menggunakan app.use() atau router.use() pada tingkat yang sesuai. Pastikan middleware diatur setelah endpoint tetapi sebelum penanganan rute utama jika diterapkan pada tingkat rute.

Pengujian dan Debugging

Jalankan aplikasi dan uji fungsionalitas middleware. Gunakan alat debugging seperti logger atau pesan konsol untuk memastikan middleware beroperasi sesuai yang diharapkan.

Keuntungan Penggunaan Middleware

Modularitas

Middleware memungkinkan pengembangan aplikasi yang modular dan dapat dikelola dengan memisahkan tugas dan tanggung jawab.

Reusabilitas

Middleware dapat digunakan kembali di seluruh aplikasi untuk melakukan tugas yang serupa, meningkatkan efisiensi pengembangan.

Ekstensibilitas

Ekosistem middleware yang luas dan beragam memungkinkan pengembang untuk memilih dan mengintegrasikan fungsionalitas tambahan sesuai kebutuhan.

Pemisahan Tanggung Jawab

Middleware memungkinkan pemisahan tanggung jawab dalam pengelolaan permintaan, mempermudah pemeliharaan dan debugging.

Penanganan Error yang Lebih Baik

Middleware penanganan kesalahan memastikan bahwa kesalahan yang tidak tertangani diatasi dengan baik dan memberikan tanggapan yang sesuai. Middleware dalam Express.js adalah komponen kritis dalam pengelolaan permintaan dan tanggapan dalam aplikasi web.

Dengan memberikan cara untuk memodifikasi objek permintaan dan tanggapan, menjalankan tugas tertentu, dan mengelola aliran kontrol, middleware memberikan fleksibilitas dan kekuatan dalam pengembangan aplikasi.

Dengan memahami konsep, jenis-jenis, dan langkah-langkah implementasinya, pengembang dapat mengoptimalkan penggunaan middleware untuk meningkatkan modularitas, reusabilitas, dan ekstensibilitas aplikasi mereka.

Melalui integrasi middleware yang bijak, pengembang dapat membangun aplikasi yang tangguh, aman, dan mudah dikelola.

Tujuan

Berikut ini adalah tujuan dari tutorial ini:

- Pembaca mengenal middleware dalam Express.
- Pembaca mampu menggunakan middleware dengan Express.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.
- Bisa meng-akses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, buatlah project Node.js bernama "contoh_nodejs_express_middleware" dengan cara yang telah dijelaskan di "Belajar Node JS Cara Membuat Project".

Selanjutnya, ubah file "package.json" menjadi seperti ini:

```
{
    "name": "contoh_nodejs_express_middleware",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "dev": "node index.js"
    },
    "keywords": [],
    "author": "",
    "license": "ISC",
    "dependencies": {
        "express": "^4.17.1"
```

```
    }
}
```

Selanjutnya, isi file "index.js" dengan kode ini:

```
// import modules
const express = require("express");

// inisialisasi express
const app = express();

// middleware ini berjalan secara global
app.use((req, res, next) => {
    console.log("MIDDLEWARE 1");
    next();
});

// request get untuk "/" atau dengan kata lain halaman index
// tanpa middleware lokal
app.get("/", (req, res) => {
    console.log("INDEX");
    res.send("INDEX");
});

// request get untuk "/about" atau dengan kata lain halaman about
// dengan middleware lokal
// perhatikan bahwa parameter ke-2 diisi function. itu middleware nya
app.get(
    "/about",
    (req, res, next) => {
        console.log("MIDDLEWARE 2");
        next();
    },
    (req, res) => {
        console.log("ABOUT");
        res.send("ABOUT");
    }
);

// jalankan server di port 3000
app.listen(3000, () => {
    console.log("Server berjalan di port 3000");
});
```

Selanjutnya, jalankan:

```
npm install
```

Sekarang, jalankan aplikasi ini:

```
npm run dev
```

Buka web browser Anda di:

http://localhost:3000/

Nanti akan muncul output di command line:

```
MIDDLEWARE 1
INDEX
MIDDLEWARE 1
```

Kenapa ada 2 "MIDDLEWARE 1"? Nanti akan saya bahas di bagian pembahasan.

Jika Anda mengakses:

http://localhost:3000/about

Nanti akan muncul output di command line:

```
MIDDLEWARE 1
MIDDLEWARE 2
ABOUT
```

Pembahasan

Pertama-tama, kita membuat project Node.js dan mengisinya dengan beberapa script.

Selanjutnya, kita menjalankan perintah npm install tanpa diikuti nama package.

Itu maksudnya adalah untuk meng-install seluruh dependencies yang tertulis di "package.json", dalam hal ini hanya Express.

Pada "index.js", kita mulai dari mengimpor modul express:

```
// import modules
const express = require("express");
```

Pada kode di atas, kita mengimpor modul express dan menyimpannya dalam const express.

Node.js menyediakan function require untuk meng-import modul.

Jika modul yang di-import adalah modul pihak ke-3, maka kita perlu meng-install-nya terlebih dahulu dengan npm. Kita telah melakukannya dengan npm install tadi.

Selain itu, jika modul yang di-import bukan modul buatan sendiri, kita tidak perlu menggunakan path pada require, cukup nama modulnya saja

Pada baris kode selanjutnya, kita menginisialisasi dan membuat objek express:

```
// inisialisasi express
const app = express();
```

Objek tersebut disimpan pada const app.

Selanjutnya, kita mendaftarkan middleware global:

```
// middleware ini berjalan secara global
app.use((req, res, next) => {
    console.log("MIDDLEWARE 1");
    next();
});
```

Dengan adanya middleware global tersebut, setiap kali ada request apapun yang masuk ke server, maka console.log akan memprint "MIDDLEWARE 1".

Selanjutnya, kode ini mendaftarkan halaman INDEX:

```
// request get untuk "/" atau dengan kata lain halaman index
// tanpa middleware lokal
app.get("/", (req, res) => {
    console.log("INDEX");
    res.send("INDEX");
});
```

Dalam contoh sebelumnya, "MIDDLEWARE 1" muncul dua kali saat mengakses INDEX.

Itu karena, jika kita mengakses server via web browser, maka biasanya web browser juga akan me-request favicon, jadi ada 2 request.

Adapun URL dari favicon beda dengan INDEX, sehingga teks "INDEX" hanya muncul satu kali.

Selanjutnya, kita mendaftarkan halaman ABOUT:

```
// request get untuk "/about" atau dengan kata lain halaman about
// dengan middleware lokal
// perhatikan bahwa parameter ke-2 diisi function. itu middleware nya
app.get(
    "/about",
    (req, res, next) => {
        console.log("MIDDLEWARE 2");
```

```
        next();
    },
    (req, res) => {
        console.log("ABOUT");
        res.send("ABOUT");
    }
);
```

Tampak bahwa ada "MIDDLEWARE 2" yang terpasang pada request handler.

Dengan demikian, outputnya, jika Anda belum membersihkan cache browser Anda adalah:

```
MIDDLEWARE 1
MIDDLEWARE 2
ABOUT
```

Kenapa "MIDDLEWARE 1" hanya ada satu?

Itu karena, favicon sudah di-request sebelumnya, sehingga browser tidak me-request-nya lagi.

Kecuali mungkin jika Anda telah menghapus cache browser Anda.

Selanjutnya, kita menjalankan server tersebut dengan cara melakukan listen di port 3000:

```
// jalankan server di port 3000
app.listen(3000, () => {
    console.log("Server berjalan di port 3000.");
});
```

Dalam keadaan ini, port 3000 sedang digunakan oleh aplikasi ini.

Jadi, jika Anda menjalankan aplikasi lain yang harus menggunakan port 3000, maka kemungkinan aplikasi lain tersebut akan tidak berjalan.

Hal sebaliknya juga berlaku.

Jika sebelum aplikasi ini dijalankan port 3000 sedang digunakan, maka aplikasi ini juga tidak akan berjalan.

Penutup

Sekarang, seharusnya Anda telah mengenal dan mampu menggunakan middleware dalam Express.

Belajar Node JS Menggunakan Express Router

Source Code Project Ini

contoh_nodejs_express_router

CATATAN

Perhatikan bahwa awalan "/" dalam tutorial ini berarti root folder. Ini akan diikuti dengan nama file, baik untuk di root folder maupun sub folder-nya.

Pendahuluan

Menangani request dan memberi response dari user merupakan hal yang sangat sering dilakukan dengan Express.

Sebenarnya, tanpa router sekalipun, kita tetap bisa membuat server dan meng-handle request dengan Express.

Namun, ketika aplikasi semakin besar dan semakin banyak request handler-nya, tentunya satu file script akan semakin panjang karena request handler tadi.

Agar routes dalam aplikasi yang menggunakan Express lebih rapi, maka digunakan Router.

Dengan router, request handler bisa dipisahkan ke file-file script terpisah sehingga lebih rapi.

Lebih Lanjut tentang Express Router

Express.js, sebagai salah satu kerangka kerja web paling populer untuk Node.js, memperkenalkan konsep router untuk membantu dalam pengelolaan dan pengorganisasian rute-rute aplikasi.

Router memungkinkan pengembang untuk mengelompokkan dan menangani rute-rute tertentu secara terorganisir.

Dalam bagian ini, kita akan menjelaskan router dalam Express.js tanpa menggunakan kode, memfokuskan pada pengertian dasar, fungsionalitas, dan manfaat yang ditawarkan oleh router.

Pengertian Router

Router dalam Express.js adalah modul yang digunakan untuk mengelola rute-rute atau endpoint-endpoint dalam aplikasi web. Rute-rute ini mencakup URL atau URI yang dapat diakses oleh pengguna melalui permintaan HTTP. Dengan menggunakan router, pengembang dapat membagi logika aplikasi menjadi bagian-bagian terpisah, membuat kode lebih terstruktur, dan meningkatkan skalabilitas.

Fungsionalitas Utama Router

Pemisahan Logika Aplikasi

Router memungkinkan pemisahan logika aplikasi berdasarkan rute. Ini mempermudah pengembangan dan pemeliharaan kode dengan mengorganisirnya ke dalam modul-modul terpisah.

Manajemen Endpoint

Router mengelola endpoint-endpoint atau rute-rute tertentu dalam aplikasi. Setiap router dapat menangani satu atau lebih rute, dan setiap rute dapat diarahkan ke logika pengendali yang berbeda.

Skema Rute yang Jelas

Router membantu dalam menyusun skema rute yang jelas dan terstruktur. Ini membuat mudah untuk memahami dan mengelola semua rute yang tersedia dalam aplikasi.

Kemampuan Pengelompokan

Router memungkinkan pengelompokan rute-rute terkait. Ini berguna ketika ada beberapa rute yang memerlukan logika atau penanganan yang serupa.

Middleware Khusus Router

Setiap router dapat memiliki middleware khusus yang hanya berlaku untuk rute-rute dalam router tersebut. Hal ini memberikan fleksibilitas tambahan dalam menangani permintaan yang terkait dengan router tertentu.

Pemetaan Rute ke Fungsi Pengendali

Router memetakan setiap rute ke fungsi pengendali (handler) yang bertanggung jawab untuk menangani permintaan yang sesuai. Ini memungkinkan logika bisnis terpisah dari definisi rute.

Manfaat Router dalam Express.js

Pemeliharaan Kode yang Lebih Mudah

Dengan menggunakan router, pengembang dapat memisahkan logika aplikasi ke dalam modul-modul terpisah berdasarkan fungsionalitas atau fitur tertentu. Ini membuat pemeliharaan kode lebih mudah karena setiap modul dapat fokus pada satu aspek tertentu dari aplikasi.

Organisasi Terstruktur

Router membantu dalam menyusun struktur yang terorganisir untuk rute-rute dalam aplikasi. Struktur yang jelas memudahkan pemahaman dan navigasi melalui kode.

Skalabilitas yang Lebih Baik

Saat aplikasi tumbuh, router memungkinkan pengembang untuk menangani kompleksitas dengan cara yang terstruktur. Rute-rute dapat diorganisir ke dalam router-ruter terpisah, memfasilitasi pengelolaan proyek yang besar.

Pengelompokan Middleware

Router memungkinkan pengelompokan middleware tertentu untuk digunakan hanya dalam konteks router tersebut. Hal ini memungkinkan penanganan permintaan yang terkait dengan router tertentu memiliki middleware yang khusus dan terpisah dari middleware lainnya.

Penggunaan Middleware Global dan Lokal

Router memungkinkan pengembang untuk menggunakan middleware secara global, yang berlaku untuk seluruh aplikasi, dan juga middleware lokal yang hanya berlaku untuk rute-rute tertentu dalam router.

Struktur Dasar Penggunaan Router

Pendefinisian Router

Pertama, router perlu didefinisikan menggunakan metode express.Router(). Ini menciptakan instance router yang dapat digunakan untuk menangani rute-rute tertentu.

Definisi Rute pada Router

Rute-rute ditambahkan ke router menggunakan metode seperti router.get(), router.post(), atau metode lainnya sesuai dengan metode HTTP yang diinginkan.

Pengelompokan dan Penggunaan Router

Router dapat dikelompokkan untuk menangani rute-rute tertentu. Setiap router kemudian dapat digunakan sebagai middleware dalam aplikasi utama dengan menggunakan app.use().

Pemetaan Rute ke Fungsi Pengendali

Setiap rute pada router dapat diarahkan ke fungsi pengendali yang akan menangani permintaan yang sesuai dengan rute tersebut.

Penanganan Error pada Router

Router dapat memiliki middleware penanganan kesalahan sendiri untuk menangani kesalahan yang terjadi dalam konteks router tersebut. Router dalam Express.js adalah alat yang kuat untuk mengelola dan mengorganisir rute-rute dalam aplikasi web.

Dengan memungkinkan pemisahan logika, pengelompokan, dan definisi rute yang terstruktur, router membantu dalam mengelola kompleksitas pengembangan aplikasi.

Keuntungan pemeliharaan kode yang lebih mudah, struktur yang terorganisir, dan kemampuan pengelompokan membuat router menjadi komponen kunci dalam pembangunan aplikasi web yang skalabel dan mudah dikelola.

Dengan pemahaman yang baik tentang penggunaan router, pengembang dapat meningkatkan efisiensi dan keterbacaan kode mereka, mendukung pengelolaan proyek yang besar, dan membangun aplikasi yang tangguh dan mudah dipelihara.

Tujuan

Berikut ini adalah tujuan dari tutorial ini:

- Pembaca mengenal router dalam Express.
- Pembaca mampu menggunakan router dengan Express.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.

- Bisa meng-akses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, buatlah project Node.js bernama "contoh_nodejs_express_router" dengan cara yang telah dijelaskan di "Belajar Node JS Cara Membuat Project".

Selanjutnya, ubah file "/package.json" menjadi seperti ini:

```
{
    "name": "contoh_nodejs_express_router",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "dev": "node index.js"
    },
    "keywords": [],
    "author": "",
    "license": "ISC",
    "dependencies": {
        "express": "^4.17.1"
    }
}
```

Selanjutnya, isi file "/index.js" dengan kode ini:

```
// import module express
const express = require("express");

// inisialisasi express
const app = express();

// begin: import routers
const indexRouter = require("./routes/index.js");
const aboutRouter = require("./routes/about.js");
// end: import routers

// begin: gunakan sebagai routers. parameter pertama adalah path prefix nya
app.use("/", indexRouter);
app.use("/about", aboutRouter);
// end: gunakan sebagai routers. parameter pertama adalah path prefix nya

// jalankan server di port 3000
app.listen(3000, () => {
    console.log("Server berjalan di port 3000");
});
```

Selanjutnya, buat file "/routes/index.js", kemudian isi dengan kode ini:

```
// import express
const express = require("express");

// buat routernya
const router = express.Router();

// handle request "/"
router.get("/", (req, res) => {
    // kirim teks bertuliskan "INDEX"
    res.send("INDEX");
});

// export modul ini
module.exports = router;
```

Selanjutnya, buat file "/routes/about.js", kemudian isi dengan kode ini:

```
// import express
const express = require("express");

// buat routernya
const router = express.Router();

// handle request /about.
// di parameternya tertulis "/" karena saat
// di-use di root index.js sudah ada path "/about"
router.get("/", (req, res) => {
    // kirim teks bertuliskan "ABOUT"
    res.send("ABOUT");
});

// export module ini
module.exports = router;
```

Selanjutnya, jalankan:

```
npm install
```

Sekarang, jalankan aplikasi ini:

```
npm run dev
```

Buka web browser Anda di:

http://localhost:3000/

Nanti akan muncul output di browser:

```
INDEX
```

Buka web browser Anda di:

http://localhost:3000/about

Nanti akan muncul output di browser:

```
ABOUT
```

Pembahasan

Pertama-tama, kita membuat project Node.js dan mengisinya dengan beberapa script.

Selanjutnya, kita menjalankan perintah npm install tanpa diikuti nama package.

Itu maksudnya adalah untuk meng-install seluruh dependencies yang tertulis di "/package.json", dalam hal ini hanya Express.

Pada "/index.js", kita mulai dari mengimpor modul express:

```
// import modules
const express = require("express");
```

Pada kode di atas, kita mengimpor modul express dan menyimpannya dalam const express.

Node.js menyediakan function require untuk meng-import modul.

Jika modul yang di-import adalah modul pihak ke-3, maka kita perlu meng-install-nya terlebih dahulu dengan npm. Kita telah melakukannya dengan npm install tadi.

Selain itu, jika modul yang di-import bukan modul buatan sendiri, kita tidak perlu menggunakan path pada require, cukup nama modulnya saja

Pada baris kode selanjutnya, kita menginisialisasi dan membuat objek express:

```
// inisialisasi express
const app = express();
```

Objek tersebut disimpan pada const app.

Selanjutnya, kita mengimpor routers yang ada di folder "/routes", yakni "/routes/index.js" dan "/routes/about.js":

```
// begin: import routers
const indexRouter = require("./routes/index.js");
const aboutRouter = require("./routes/about.js");
// end: import routers
```

Mengimpor router saja tidak cukup.

Harus di-use juga.

Kode ini mendaftarkan kedua routers tadi:

```
// begin: gunakan sebagai routers. parameter pertama adalah path prefix nya
app.use("/", indexRouter);
app.use("/about", aboutRouter);
// end: gunakan sebagai routers. parameter pertama adalah path prefix nya
```

Perhatikan bahwa parameter pertama adalah path untuk prefix dari masing-masing request di masing-masing router.

Karena router index prefix-nya "/" maka tidak perlu ada tambahan path saat mengaksesnya.

http://localhost:3000/ akan mengakses "/"-nya index.

Adapun router about prefix-nya "/about", maka untuk mengakses "/" di about harus ditambahkan "/about".

http://localhost:3000/about akan mengakses "/"-nya about.

Ini adalah isi dari script "/routes/index.js":

```
// import express
const express = require("express");

// buat routernya
const router = express.Router();

// handle request "/"
router.get("/", (req, res) => {
    // kirim teks bertuliskan "INDEX"
    res.send("INDEX");
});

// export modul ini
module.exports = router;
```

Dan ini adalah isi dari script "/routes/about.js":

```
// import express
const express = require("express");

// buat routernya
const router = express.Router();

// handle request /about.
// di parameternya tertulis "/" karena saat
// di-use di root index.js sudah ada path "/about"
router.get("/", (req, res) => {
    // kirim teks bertuliskan "ABOUT"
    res.send("ABOUT");
});

// export module ini
module.exports = router;
```

Tidak ada hal yang aneh pada kedua script di atas kecuali:

```
// export module ini
module.exports = router;
```

Maksud dari kode tersebut adalah untuk meng-export script sebagai modul, sehingga dapat di-require seperti yang telah kita lakukan di "/index.js".

Penutup

Sekarang, seharusnya Anda telah mengenal dan mampu menggunakan Express router.

Belajar Node JS Membaca Dan Menulis File

Source Code Project Ini

contoh_nodejs_membaca_dan_menulis_file

Pendahuluan

Kali ini, kita akan belajar membaca dan menulis file di Node.js.

Pada dasarnya, untuk melakukan itu ada beberapa metode:

- Dengan cara asynchronous dan dengan callback.
- Dengan cara asynchronous dan async await.
- Dengan cara menggunakan versi synchronous dari API-nya.

Di bawah ini ada 4 script:

- readfile.js dan readfilesync.js menunjukkan cara untuk membaca file.
- writefile.js dan writefilesync.js menunjukkan cara untuk menulis file.

Yang memiliki akhiran “sync” menggunakan metode synchronous, sedangkan yang tidak menggunakan metode asynchronous.

Lebih Lanjut tentang File Reading dan File Writing

File reading dan file writing adalah dua konsep penting dalam pengelolaan berkas (file) di Node.js:

File Reading (Membaca Berkas):

Tujuan Utama:

- File reading atau pembacaan berkas adalah proses membaca data dari berkas yang ada di sistem file.

Proses:

- Node.js menyediakan metode dan fungsi untuk membaca berkas secara asinkron atau sinkron.
- Dalam operasi asinkron, Node.js akan membaca berkas secara non-blokir, yang berarti kode akan terus berjalan tanpa menunggu pembacaan berkas selesai. Callback akan dipanggil ketika pembacaan selesai.
- Dalam operasi sinkron, Node.js akan membaca berkas secara blokir, yang berarti kode akan menunggu pembacaan berkas selesai sebelum melanjutkan eksekusi kode berikutnya.

Penggunaan:

- File reading digunakan ketika aplikasi perlu membaca konfigurasi, data, atau teks dari berkas untuk diproses lebih lanjut, seperti dalam aplikasi web untuk membaca template HTML atau dalam aplikasi server untuk membaca file konfigurasi.

File Writing (Menulis Berkas):

Tujuan Utama:

- File writing atau penulisan berkas adalah proses menulis data ke dalam berkas yang ada di sistem file.

Proses:

- Node.js menyediakan metode dan fungsi untuk menulis data ke berkas secara asinkron atau sinkron.
- Seperti file reading, dalam operasi asinkron, Node.js akan menulis berkas secara non-blokir, sementara dalam operasi sinkron, Node.js akan menulis berkas secara blokir.

Penggunaan:

- File writing digunakan ketika aplikasi perlu menyimpan data hasil proses atau hasil operasi ke dalam berkas, seperti log aplikasi, file konfigurasi yang diperbarui, atau menyimpan data yang diperlukan untuk aplikasi berikutnya.

Pentingnya Asinkronisitas:

- Dalam penggunaan file reading dan file writing, operasi asinkron lebih umum digunakan karena memungkinkan aplikasi untuk tetap responsif dan efisien. Dengan demikian, aplikasi dapat melanjutkan eksekusi operasi lain tanpa menunggu pembacaan atau penulisan berkas selesai. File reading dan file writing merupakan bagian integral dari pengelolaan berkas di Node.js dan memungkinkan

pengembang untuk mengakses, membaca, dan menulis data dari dan ke dalam berkas dengan mudah dan efisien dalam aplikasi mereka.

Langkah-Langkah

Buat folder bernama "contoh_nodejs_membaca_dan_menulis_file", kemudian masuk ke dalamnya.

Selanjutnya buat file "readfile.js" dan isi dengan kode ini:

```
// file: readfile.js

// begin: import modules
const fs = require('fs');
const util = require('util');
// end: import modules

// baca file "hello.txt" secara asynchronous
function readHello() {
    fs.readFile('./hello.txt', 'utf8', function (err, data) {
        // tampilkan isinya
        console.log(data);
    });
}

// baca file "hello.txt" secara asynchronous dengan async await
async function readHelloAsyncAwait() {
    // ubah dulu jadi promise
    const readFileAsyncAwait = util.promisify(fs.readFile);

    // sekarang baru bisa menggunakan async await
    const hello = await readFileAsyncAwait('./hello.txt', 'utf8');

    // tampilkan isinya
    console.log(hello);
}

// jalankan
readHello();
readHelloAsyncAwait();
```

Selanjutnya buat file "readfilesync.js" dan isi dengan kode ini:

```
// file: readfilesync.js

// import module fs
const fs = require('fs');

// baca file "hello.txt"
function readHello() {
    // baca secara synchronous. sudah tersedia fungsinya secara default
```

```
    const hello = fs.readFileSync('./hello.txt', 'utf8');

    // tampilkan isinya
    console.log(hello);
}

readHello();
```

Selanjutnya buat file "writefile.js" dan isi dengan kode ini:

```
// file: writefile.js

// begin: import modules
const fs = require('fs');
const util = require('util');
// end: import modules

// tulis file secara asyncrhonous
function writeHello() {
    fs.writeFile('./hello-write-file.txt', "hello world writeFile", function () {
        console.log('file sudah ditulis.');
    })
}

// tulis file secara asyncrhonous dengan async await
async function writeHelloAsyncAwait() {
    // ubah dulu jadi promise
    const writeFileAsyncAwait = util.promisify(fs.writeFile);

    // sekarang bisa menggunakan async await
    await writeFileAsyncAwait('./hello-write-file-async-await.txt', "hello world
writeFileAsyncAwait");
    console.log('file sudah ditulis.');
}

// jalankan
// writeHello();

writeHelloAsyncAwait();
```

Selanjutnya buat file "writefilesync.js" dan isi dengan kode ini:

```
// file: writefilesync.js

// import module fs
const fs = require('fs');

// tulis file secara synchronous. fungsinya sudah tersedia secara default
function writeHello() {
```

```
    fs.writeFileSync('./hello-write-file-sync.txt', "hello world writeFileSync");
}

// jalankan
writeHello();
```

Selanjutnya, buat file bernama "hello.txt" dan isi dengan:

```
hello world
```

Untuk menjalankan kode-kode tadi, buka folder projectnya via terminal, kemudian jalankan perintah:

```
node <nama_script>
```

Pembahasan

Pembahasan "readfile.js"

Kode ini adalah sebuah program JavaScript yang bertujuan untuk membaca isi dari file "hello.txt" secara asynchronous menggunakan Node.js.

Berikut adalah penjelasan baris per baris:

```
const fs = require('fs');
const util = require('util');
COPY
Baris ini mengimpor dua modul dari Node.js, yaitu fs (File System) untuk melakukan
operasi file dan util untuk menggunakan utilitas bawaan Node.js.

 function readHello() {
       fs.readFile('./hello.txt', 'utf8', function (err, data) {
           // tampilkan isinya
           console.log(data);
       });
   }
```

Fungsi readHello() digunakan untuk membaca file "hello.txt" secara asynchronous menggunakan fs.readFile().

Callback function yang diberikan akan dipanggil setelah file selesai dibaca.

Jika terjadi kesalahan, err akan berisi informasi tentang kesalahan, jika tidak, data akan berisi isi dari file yang dibaca.

```
async function readHelloAsyncAwait() {
       // ubah dulu jadi promise
```

```
        const readFileAsyncAwait = util.promisify(fs.readFile);

        // sekarang baru bisa menggunakan async await
        const hello = await readFileAsyncAwait('./hello.txt', 'utf8');

        // tampilkan isinya
        console.log(hello);
    }
```

Fungsi readHelloAsyncAwait() juga membaca file "hello.txt" secara asynchronous, tetapi menggunakan util.promisify() untuk mengubah fungsi fs.readFile() menjadi promise-based sehingga dapat digunakan dengan async/await.

Ini membuat kodenya terlihat lebih bersih dan mudah dipahami.

```
readHello();
readHelloAsyncAwait();
```

Terakhir, kedua fungsi tersebut dipanggil untuk dieksekusi. readHello() akan menampilkan isi file "hello.txt" menggunakan callback, sementara readHelloAsyncAwait() akan melakukannya menggunakan async/await.

Karena operasi baca file dilakukan secara asynchronous, pemanggilan kedua fungsi tidak akan saling menunggu satu sama lain, sehingga keduanya akan dieksekusi secara bersamaan.

Pembahasan "readfilesync.js"

Kode ini merupakan program JavaScript yang bertujuan untuk membaca isi dari file "hello.txt", namun kali ini menggunakan metode synchronous (blocking) daripada asynchronous.

Berikut penjelasan baris per baris:

```
const fs = require('fs');
```

Baris ini mengimpor modul fs (File System) dari Node.js, yang digunakan untuk melakukan operasi file.

```
function readHello() {
    const hello = fs.readFileSync('./hello.txt', 'utf8');
    console.log(hello);
}
```

Fungsi readHello() membaca file "hello.txt" secara synchronous menggunakan fs.readFileSync().

Ini berarti eksekusi program akan terhenti sementara file dibaca, dan program tidak akan melanjutkan eksekusi selama proses baca file belum selesai.

Setelah file dibaca, isi file akan disimpan dalam variabel hello, dan kemudian ditampilkan di konsol menggunakan console.log().

```
readHello();
```

Terakhir, fungsi readHello() dipanggil untuk dieksekusi.

Karena metode fs.readFileSync() bersifat synchronous, pemanggilan fungsi ini akan menghentikan eksekusi program sampai proses pembacaan file selesai.

Ketika menggunakan readFileSync, proses eksekusi program akan dihentikan sementara sampai operasi file selesai.

Ini berarti bahwa jika ada operasi lain yang perlu dilakukan, seperti menanggapi permintaan HTTP atau input pengguna, maka proses tersebut akan ditunda sampai pembacaan file selesai.

Hal ini dapat menghambat kinerja dalam aplikasi yang membutuhkan respon cepat terhadap berbagai input.

Sebaliknya, metode asynchronous, seperti yang digunakan dalam kode sebelumnya, memungkinkan program untuk melanjutkan eksekusi tanpa harus menunggu selesainya operasi file.

Pembahasan "writefile.js"

Kode ini bertujuan untuk menulis ke file menggunakan Node.js, baik secara synchronous maupun asynchronous dengan menggunakan async/await.

Berikut penjelasan baris per baris:

```
const fs = require('fs');
const util = require('util');
```

Baris ini mengimpor modul fs (File System) dan util dari Node.js.

fs digunakan untuk operasi file, sedangkan util akan digunakan untuk mengubah fungsi fs.writeFile() menjadi promise-based agar bisa digunakan dengan async/await.

```
function writeHello() {
    fs.writeFile('./hello-write-file.txt', "hello world writeFile", function () {
        console.log('file sudah ditulis.');
    })
}
```

Fungsi writeHello() menulis ke file "hello-write-file.txt" secara asynchronous menggunakan fs.writeFile().

Setelah file selesai ditulis, callback function akan dipanggil dan pesan "file sudah ditulis." akan ditampilkan di konsol.

```
async function writeHelloAsyncAwait() {
    const writeFileAsyncAwait = util.promisify(fs.writeFile);

    await writeFileAsyncAwait('./hello-write-file-async-await.txt', "hello world
writeFileAsyncAwait");
    console.log('file sudah ditulis.');
}
```

Fungsi writeHelloAsyncAwait() melakukan hal yang sama seperti writeHello(), tetapi menggunakan async/await.

Pertama, fungsi util.promisify() digunakan untuk mengubah fungsi fs.writeFile() menjadi promise-based.

Kemudian, await digunakan untuk menulis ke file "hello-write-file-async-await.txt". Setelah selesai, pesan "file sudah ditulis." akan ditampilkan di konsol.

```
writeHelloAsyncAwait();
```

Terakhir, fungsi writeHelloAsyncAwait() dipanggil untuk menulis ke file secara asynchronous.

Kode ini menunjukkan cara menulis ke file baik secara synchronous maupun asynchronous dengan menggunakan Node.js.

Metode asynchronous lebih disukai karena tidak menghentikan eksekusi program saat menulis file, sehingga tidak menghalangi proses lain yang sedang berjalan.

Pembahasan "writefilesync.js"

Kode ini adalah sebuah program sederhana yang bertujuan untuk menulis ke file secara synchronous (blok) menggunakan Node.js.

Berikut adalah penjelasan baris per baris:

```
const fs = require('fs');
```

Baris ini mengimpor modul fs (File System) dari Node.js, yang digunakan untuk melakukan operasi file.

```
function writeHello() {
    fs.writeFileSync('./hello-write-file-sync.txt', "hello world writeFileSync");
}
```

Fungsi writeHello() menulis teks "hello world writeFileSync" ke dalam file bernama "hello-write-file-sync.txt" secara synchronous menggunakan fs.writeFileSync().

Ini berarti eksekusi program akan terhenti sementara sampai proses penulisan file selesai.

```
writeHello();
```

Pemanggilan fungsi writeHello() dijalankan secara langsung.

Ini akan menyebabkan proses penulisan ke file berlangsung pada saat program dijalankan.

Kode ini berguna untuk membuat atau memperbarui file dengan konten yang diberikan secara langsung dan dalam mode blocking.

Namun, karena metode fs.writeFileSync() bersifat synchronous, hal ini dapat menyebabkan program berhenti menanggapi input atau permintaan lainnya sampai proses penulisan selesai.

Oleh karena itu, penggunaan metode asynchronous lebih disarankan dalam kebanyakan kasus agar program dapat terus berjalan dan merespons input atau permintaan yang datang.

Penutup

Sekian dan terima kasih.

Belajar Node JS Menampilkan Machine ID

Source Code Project Ini

contoh_nodejs_menampilkan_machine_id

Pendahuluan

Kali ini, kita akan belajar cara menampilkan machine ID di Node.js

Ketika kita membuat sebuah aplikasi, terutama aplikasi desktop, mungkin saja kita ingin membatasi penginstallan aplikasi tersebut pada satu komputer saja.

Dengan kata lain kita mungkin menginginkan sebuah sistem lisensi dalam aplikasi kita.

Agar hal tersebut dapat terwujud diperlukan suatu cara untuk mendapatkan identitas unik dari OS di komputer tersebut.

Node Machine ID (https://www.npmjs.com/package/node-machine-id) bisa membantu kita untuk mendapatkan unique id dari OS yang menggunakannya.

Pada dasarnya Node Machine ID menggunakan registry untuk mendapatkan unique id di Windows.

Tepatnya di “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography”.

Lebih Lanjut tentang node-machine-id

node-machine-id adalah sebuah paket npm yang menyediakan cara untuk mendapatkan ID mesin (machine ID) dari sistem tempat aplikasi Node.js berjalan.

ID mesin ini biasanya berupa nilai unik yang berhubungan dengan perangkat keras atau konfigurasi sistem, dan sering digunakan untuk tujuan lisensi, identifikasi, atau keamanan dalam aplikasi.

Beberapa poin penting tentang node-machine-id:

- Mendapatkan ID Mesin Secara Konsisten: Paket ini memungkinkan aplikasi Node.js untuk mendapatkan ID mesin yang konsisten dari sistem yang berjalan di berbagai platform, seperti Windows, macOS, dan Linux. Ini membantu dalam menjaga keunikan dan ketahanan aplikasi dalam situasi berbagai lingkungan sistem.
- Menggunakan Metode Penyedia ID Yang Berbeda: Node-machine-id menyediakan metode untuk mendapatkan ID mesin menggunakan berbagai penyedia, seperti UUID, serial disk, atau metode kustom. Ini memberikan fleksibilitas dalam memilih metode yang paling sesuai dengan kebutuhan aplikasi.
- Keselarasan dengan Standar Platform: Paket ini dirancang untuk mematuhi standar platform terkait pengambilan ID mesin. Misalnya, di Windows, itu akan mencoba mendapatkan nomor serial disk, sementara di macOS, itu akan menggunakan UUID.
- Penggunaan yang Sederhana: Node-machine-id menyediakan antarmuka yang sederhana dan mudah digunakan untuk mendapatkan ID mesin dari sistem. Ini memungkinkan pengembang untuk dengan cepat mengintegrasikan fitur ini ke dalam aplikasi mereka tanpa banyak kode tambahan.
- Kompabilitas dengan Lingkungan Node.js: Paket ini didesain untuk berjalan dengan baik di lingkungan Node.js, dan dapat diinstal dan digunakan melalui npm, manajer paket standar untuk proyek Node.js.

node-machine-id sangat berguna dalam aplikasi yang memerlukan identifikasi unik dari mesin atau sistem tempat aplikasi dijalankan.

Ini dapat digunakan untuk mengimplementasikan fitur lisensi, kontrol akses, atau identifikasi perangkat keras dalam aplikasi Node.js dengan cara yang mudah dan andal.

Langkah-Langkah

Buat folder bernama "contoh_nodejs_menampilkan_machine_id", kemudian masuk ke dalamnya.

Selanjutnya buat file "tampilkan-machine-id-asynchronous.js" dan isi dengan kode ini:

```
// file: tampilkan-machine-id-asynchronous.js

// import modul node-machine-id
const machineId = require("node-machine-id").machineId;

// ambil machine id nya
async function dapatkanMachineID(original) {
    let mcid = await machineId({ original: original });
    return mcid;
}

// bungkus dalam sebuah function
async function run() {
    let mcid = await dapatkanMachineID(true);
    console.log(mcid);
}
```

```
// jalankan
run();
```

Selanjutnya buat file "tampilkan-machine-id-synchronous.js" dan isi dengan kode ini:

```
// file: tampilkan-machine-id-synchronous.js

// import modul node-machine-id
const machineIdSync = require("node-machine-id").machineIdSync;

// ambil machine id nya
let mcid = machineIdSync({ original: true });
console.log(mcid);
```

Untuk menjalankan kode-kode tadi, buka folder projectnya via terminal, kemudian jalankan perintah:

```
node <nama_script>
```

Pembahasan

Pembahasan "tampilkan-machine-id-asynchronous.js"

Kode ini adalah contoh penerapan pemrograman asinkron di JavaScript menggunakan fungsi async/await.

Mari kita bahas langkah-langkahnya:

- Import Modul node-machine-id: Pada baris awal, modul node-machine-id diimpor menggunakan pernyataan require. Modul ini digunakan untuk mendapatkan ID mesin dari sistem yang sedang berjalan.
- Fungsi dapatkanMachineID: Ini adalah fungsi asinkron yang bertugas untuk mendapatkan ID mesin. Fungsi ini menggunakan kata kunci async, yang menunjukkan bahwa fungsi tersebut akan mengembalikan janji (promise) secara implisit. Dalam tubuh fungsi, ID mesin diperoleh menggunakan fungsi machineId dari modul node-machine-id. Dengan menggunakan kata kunci await, proses ini ditunda hingga hasilnya tersedia.
- Fungsi run: Fungsi ini juga dideklarasikan sebagai async. Ini digunakan untuk menjalankan logika aplikasi utama. Di dalamnya, dapatkanMachineID dipanggil dengan parameter true, yang kemudian menunggu hasilnya menggunakan await. Setelah hasil diterima, nilai ID mesin dicetak ke konsol.
- Jalankan: Terakhir, fungsi run dipanggil untuk memulai eksekusi program.

Jadi, keseluruhan kode ini bertujuan untuk mendapatkan ID mesin secara asinkron dan mencetaknya ke konsol.

Dengan menggunakan async/await, kode ini lebih mudah dipahami dan dijalankan secara bersamaan dengan operasi lainnya tanpa menghalangi eksekusi program.

Pembahasan "tampilkan-machine-id-synchronous.js"

Kode ini adalah contoh penerapan pemrograman secara sinkron di JavaScript untuk mendapatkan ID mesin menggunakan modul node-machine-id.

Berikut adalah penjelasan singkatnya:

- Import Modul node-machine-id: Modul node-machine-id diimpor menggunakan pernyataan require.
- Fungsi machineIdSync: Ini adalah fungsi sinkron yang digunakan untuk mendapatkan ID mesin secara langsung. Karena fungsi ini bersifat sinkron, tidak perlu menunggu hasilnya, sehingga ID mesin langsung diperoleh saat pemanggilan fungsi tersebut.
- Mendapatkan ID Mesin: Fungsi machineIdSync dipanggil dengan parameter { original: true } untuk mendapatkan ID mesin. Hasilnya disimpan dalam variabel mcid.
- Cetak ID Mesin: Nilai ID mesin yang diperoleh kemudian dicetak ke konsol menggunakan console.log.

Dalam kode ini, proses mendapatkan ID mesin dilakukan secara sinkron, yang berarti eksekusi program akan berhenti dan menunggu sampai hasilnya diperoleh sebelum melanjutkan eksekusi program selanjutnya.

Ini berbeda dengan pendekatan asinkron di mana program dapat melanjutkan eksekusi dan melakukan tugas-tugas lain sambil menunggu hasil operasi asinkron.

Penutup

Sekian dan terima kasih.

Belajar Node JS Mengenal bcryptjs

Source Code Project Ini

contoh_nodejs_bcryptjs

Pendahuluan

Dalam aplikasi web yang dibuat dengan Node.js, sangat wajar jika terdapat fitur login.

Dengan fitur login, maka aplikasi web dapat memfilter siapa yang berhak atas konten apa.

Biasanya fitur login tersebut disandingkan dengan fitur register atau yang semacamnya.

Saat register, biasanya user dihadapkan dengan pengisian password-nya.

Di backend, data password tersebut biasanya di-hash dengan algoritma tertentu.

Nanti, data password yang tersimpan di database adalah versi ter-hash-nya.

Saat login, input password dengan password yang ter-hash tadi akan dibandingkan.

Jika cocok, maka user akan lolos untuk login.

Sederhananya kira-kira begitu.

Untuk melakukan hash itu sendiri saya biasanya menggunakan sebuah package bernama bcryptjs.

Di sini saya akan memperlihatkan sebagian cara penggunaannya.

Lebih Lanjut tentang bcryptjs dan bcrypt

bcrypt dan bcryptjs adalah kedua library yang digunakan untuk hashing password di berbagai bahasa pemrograman, termasuk JavaScript.

Keduanya berfungsi untuk mengamankan password dengan menghasilkan hash yang sulit untuk dipecahkan kembali menjadi password aslinya.

Namun, ada beberapa perbedaan antara keduanya:

bcrypt

Asal

bcrypt adalah library native yang tersedia di banyak bahasa pemrograman, termasuk Node.js. Ini adalah implementasi dari algoritma bcrypt yang ditulis dalam bahasa C.

Instalasi

Untuk menggunakan bcrypt, seringkali diperlukan untuk menginstal modul tambahan yang tergantung pada versi platform yang digunakan. Misalnya, di Node.js, Anda perlu menginstal modul bcrypt dengan npm.

Kinerja

Karena bcrypt ditulis dalam bahasa C, ini dapat memiliki kinerja yang lebih baik daripada bcryptjs, terutama dalam kasus volume besar hashing.

bcryptjs

Asal

bcryptjs adalah alternatif pure JavaScript untuk bcrypt. Ini ditulis sepenuhnya dalam JavaScript dan tidak bergantung pada modul tambahan.

Instalasi

Karena tidak bergantung pada modul tambahan, bcryptjs dapat diinstal langsung dengan npm tanpa kebutuhan untuk membangun atau menginstal dependensi tambahan.

Portabilitas

Karena tidak bergantung pada bahasa C atau modul tambahan, bcryptjs dapat digunakan di berbagai platform tanpa masalah kompatibilitas.

Kinerja

bcryptjs mungkin memiliki kinerja yang sedikit lebih lambat daripada bcrypt, terutama dalam kasus hashing besar volume, karena berjalan di atas mesin virtual JavaScript.

Dengan kata lain

- bcrypt adalah library native yang efisien dan sering digunakan dalam lingkungan produksi.
- bcryptjs adalah alternatif pure JavaScript yang mudah digunakan dan portabel di berbagai platform.
- Pemilihan antara keduanya sering tergantung pada kebutuhan spesifik aplikasi, preferensi pengembang, dan kebijakan keamanan. Jika kinerja dan efisiensi sangat penting, bcrypt mungkin menjadi pilihan yang lebih baik. Namun, jika portabilitas dan kemudahan penggunaan lebih penting, bcryptjs dapat menjadi pilihan yang baik.

Tujuan

Berikut ini adalah tujuan dari tutorial ini:

- Pembaca mampu membuat hash dengan bcryptjs.
- Pembaca mampu membandingkan password yang diinputkan dengan yang ter-hash.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.
- Bisa meng-akses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, buatlah project Node.js bernama "contoh_nodejs_bcryptjs" dengan cara yang telah dijelaskan di "Belajar Node JS Cara Membuat Project".

Selanjutnya, ubah file "package.json" menjadi seperti ini:

```
{
  "name": "contoh_nodejs_bcryptjs",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3"
  }
}
```

Selanjutnya, isi file "index.js" dengan kode ini:

```
const bcrypt = require("bcryptjs");
const password = "password";

const theHash = bcrypt.hashSync("password");
console.log(theHash);

let isMatch = bcrypt.compareSync(password, theHash);
console.log(isMatch);

isMatch = bcrypt.compareSync(password + "123", theHash);
console.log(isMatch);
```

Selanjutnya, jalankan:

```
npm install
```

Sekarang, jalankan aplikasi ini:

```
node index.js
```

Nanti akan muncul output di command line:

```
$2a$10$yS8MyiaY0guoKlQbh16kPO4yRlkCzvi2MzHb66FHOuCkCK4Amc4KW
true
false
```

Pembahasan

Pertama-tama, kita membuat project Node.js dan mengisinya dengan beberapa script.

Selanjutnya, kita menjalankan perintah npm install tanpa diikuti nama package.

Itu maksudnya adalah untuk meng-install seluruh dependencies yang tertulis di "package.json", dalam hal ini hanya bcryptjs.

Pada "index.js", kita mulai dari mengimpor modul bcryptjs:

```
const bcrypt = require("bcryptjs");
```

Pada kode di atas, kita mengimpor modul bcryptjs dan menyimpannya dalam const bcrypt.

Node.js menyediakan function require untuk meng-import modul.

Jika modul yang di-import adalah modul pihak ke-3, maka kita perlu meng-install-nya terlebih dahulu dengan npm. Kita telah melakukannya dengan npm install tadi.

Selain itu, jika modul yang di-import bukan modul buatan sendiri, kita tidak perlu menggunakan path pada require, cukup nama modulnya saja

Langkah selanjutnya, kita membuat konstanta dari password, yakni "password":

```
const password = "password";
```

Kemudian, kita membuat hash dari konstanta tersebut, lalu tampilkan:

```
const theHash = bcrypt.hashSync("password");
console.log(theHash);
```

Di komputer saya, console.log tadi menghasilkan ini:

```
$2a$10$yS8MyiaY0guoKlQbh16kPO4yRlkCzvi2MzHb66FHOuCkCK4Amc4KW
```

Kemudian, kita membandingkan antara input dengan yang di-hash tadi:

```
let isMatch = bcrypt.compareSync(password, theHash);
console.log(isMatch);
```

Di situ hasilnya true, karena memang cocok.

Jika kita ubah dengan menambahkan "123":

```
isMatch = bcrypt.compareSync(password + "123", theHash);
console.log(isMatch);
```

Maka hasilnya akan false, karena tidak cocok.

Penutup

Sekarang, seharusnya tujuan tutorial ini telah tercapai.

Belajar Node JS Mengenal dotenv

Source Code Project Ini

contoh_nodejs_dotenv

Pendahuluan

Aplikasi yang dibuat dengan Node.js ada kalanya perlu konfigurasi.

Misalnya saja dalam aplikasi web diperlukan konfigurasi seperti:

- Nama website
- Base URL
- Cookie secret
- Dan lain-lain

Untuk mengubah konfigurasi tersebut, bisa saja dilakukan dengan mengubah kode, tapi itu kurang efisien.

Akan lebih baik jika konfigurasi tersebut disimpan dalam sebuah file terpisah agar saat kita ingin mengubahnya, kita tidak perlu mengubah script kode kita berulang kali.

Solusinya, kita bisa menggunakan package dotenv yang ada di NPM.

Mari kita coba...

Lebih Lanjut tentang dotenv

dotenv adalah alat yang digunakan dalam pengembangan perangkat lunak untuk mengelola konfigurasi aplikasi dengan lebih efisien.

Dalam pengembangan perangkat lunak modern, penting untuk memisahkan konfigurasi dari kode sumber aplikasi untuk meningkatkan portabilitas, keamanan, dan fleksibilitas.

Dalam bagian ini, saya akan menjelaskan secara mendalam tentang dotenv, termasuk definisi, tujuan, manfaat, dan praktik terbaik.

Definisi dotenv:

Konsep Dasar:

dotenv adalah alat yang memungkinkan pengembang menyimpan konfigurasi aplikasi seperti kunci API, URL database, atau pengaturan lingkungan lainnya dalam file teks terpisah. Nama "dotenv" berasal dari praktik menyimpan variabel lingkungan dalam file bernama ".env".

Implementasi:

Implementasi dotenv dapat berbeda-beda tergantung pada bahasa pemrograman atau lingkungan pengembangan yang digunakan. Biasanya, dotenv terdiri dari file teks sederhana yang berisi pasangan kunci-nilai yang dipisahkan oleh tanda sama dengan (=), dan aplikasi akan membaca file ini untuk mengambil konfigurasi saat memulai.

Tujuan dotenv:

Memisahkan Konfigurasi dari Kode Sumber:

Salah satu tujuan utama dotenv adalah memisahkan konfigurasi dari kode sumber aplikasi. Dengan menggunakan file konfigurasi terpisah, pengembang dapat menghindari menyematkan konfigurasi langsung ke dalam kode, yang membuatnya lebih mudah dikelola dan diperbarui.

Keamanan:

Dengan menyimpan konfigurasi sensitif seperti kunci API atau sandi dalam file terpisah, dotenv membantu meningkatkan keamanan aplikasi. Ini mencegah informasi sensitif dari tersimpan di repositori kode sumber yang dapat diakses publik.

Portabilitas:

Menggunakan dotenv membuat aplikasi lebih mudah dipindahkan dari satu lingkungan ke lingkungan lainnya tanpa perlu mengubah kode sumber. Pengembang dapat dengan mudah mengonfigurasi aplikasi untuk lingkungan pengembangan, uji, dan produksi dengan file .env yang berbeda.

Fleksibilitas:

Dengan menyimpan konfigurasi dalam file .env terpisah, pengembang dapat dengan mudah mengubah konfigurasi tanpa perlu memodifikasi kode sumber. Ini memungkinkan penyesuaian cepat dan fleksibel terhadap perubahan lingkungan atau persyaratan aplikasi.

Manfaat Penggunaan dotenv:

Kejelasan dan Organisasi Kode:

Dengan menggunakan dotenv, konfigurasi aplikasi dipisahkan dengan jelas dari kode sumber, meningkatkan kejelasan dan organisasi kode.

Keamanan Konfigurasi:

Dengan menyimpan konfigurasi sensitif di luar kode sumber, dotenv membantu meningkatkan keamanan aplikasi dengan mencegah paparan informasi sensitif.

Fleksibilitas dan Scalability:

dotenv memungkinkan aplikasi untuk dengan mudah beradaptasi dengan perubahan lingkungan atau persyaratan dengan memungkinkan pengaturan konfigurasi yang fleksibel dan skalabel.

Portabilitas dan Reproduktibilitas:

Dengan menyimpan konfigurasi dalam file .env terpisah, aplikasi dapat dipindahkan dari satu lingkungan ke lingkungan lainnya tanpa perlu memodifikasi kode sumber, meningkatkan portabilitas dan reproduktibilitas.

Pengujian yang Mudah:

Dengan menggunakan konfigurasi yang dipisahkan, dotenv membuat pengujian aplikasi menjadi lebih mudah karena pengembang dapat dengan mudah menyediakan konfigurasi yang berbeda untuk lingkungan

pengembangan dan pengujian.

Praktik Terbaik dalam Menggunakan dotenv:

Jangan Simpan Informasi Rahasia di Repositori Kode Sumber:

Pastikan untuk tidak menyimpan informasi rahasia seperti kunci API atau sandi di dalam repositori kode sumber. Gunakan file .env untuk menyimpan informasi sensitif tersebut.

Gunakan Nama Variabel yang Deskriptif:

Gunakan nama variabel yang deskriptif untuk memudahkan pengelolaan dan pemahaman konfigurasi aplikasi. Ini membantu meningkatkan kejelasan kode.

Gunakan .env.example sebagai Template:

Sertakan file .env.example di repositori kode sumber sebagai template untuk konfigurasi aplikasi. Ini membantu pengembang baru memahami variabel yang dibutuhkan.

Gunakan Penanganan Error dengan Bijak:

Pastikan untuk menangani error dengan bijaksana saat membaca atau menggunakan konfigurasi dari dotenv. Ini membantu mencegah aplikasi gagal karena konfigurasi yang tidak valid atau hilang.

Perbarui dan Monitor Konfigurasi secara Teratur:

Perbarui dan monitor file .env secara teratur untuk memastikan bahwa konfigurasi aplikasi tetap up-to-date dan sesuai dengan kebutuhan proyek. dotenv adalah alat yang bermanfaat dalam pengembangan perangkat lunak modern untuk mengelola konfigurasi aplikasi dengan lebih efisien.

Dengan memisahkan konfigurasi dari kode sumber, dotenv membantu meningkatkan keamanan, fleksibilitas, dan portabilitas aplikasi.

Dengan mengikuti praktik terbaik, pengembang dapat memanfaatkan potensi penuh dotenv untuk mengelola konfigurasi aplikasi dengan lebih baik.

Tujuan

Berikut ini adalah tujuan dari tutorial ini:

- Pembaca mengenal dotenv dan mampu menggunakannya.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.
- Bisa meng-akses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, buatlah project Node.js bernama "contoh_nodejs_dotenv" dengan cara yang telah dijelaskan di "Belajar Node JS Cara Membuat Project".

Selanjutnya, ubah file "package.json" menjadi seperti ini:

```
{
    "name": "contoh_nodejs_dotenv",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "keywords": [],
    "author": "",
    "license": "ISC",
    "dependencies": {
        "dotenv": "^16.3.2"
    }
}
```

Selanjutnya, isi file "index.js" dengan kode ini:

```
const dotEnv = require("dotenv");
dotEnv.config();

console.log(process.env.APP_NAME, typeof process.env.APP_NAME);
console.log(process.env.BASE_URL, typeof process.env.BASE_URL);
console.log(process.env.NUM_OF_CORES, typeof process.env.BASE_URL);
```

Selanjutnya, buat file bernama ".env" pada root folder project ini, kemudian isi dengan:

```
APP_NAME=TUTORIAL_DOTENV
BASE_URL=http://localhost:3000
NUM_OF_CORES=100
```

Selanjutnya, jalankan:

```
npm install
```

Sekarang, jalankan aplikasi ini:

```
node index.js
```

Nanti akan muncul output di command line:

```
TUTORIAL_DOTENV string
http://localhost:3000 string
100 string
```

Pembahasan

Pertama-tama, kita membuat project Node.js dan mengisinya dengan beberapa script.

Selanjutnya, kita menjalankan perintah npm install tanpa diikuti nama package.

Itu maksudnya adalah untuk meng-install seluruh dependencies yang tertulis di "package.json", dalam hal ini hanya dotenv.

Pada "index.js", kita mulai dari mengimpor modul dotenv:

```
const dotEnv = require("dotenv");
```

Kemudian kita menginisialisasinya;

```
dotEnv.config();
```

Mulai dari sini, seluruh isi file ".env" akan masuk ke memori.

Sehingga:

```
console.log(process.env.APP_NAME, typeof process.env.APP_NAME);
console.log(process.env.BASE_URL, typeof process.env.BASE_URL);
console.log(process.env.NUM_OF_CORES, typeof process.env.BASE_URL);
```

Kita bisa mengaksesnya dengan process.env diikuti dengan key yang ada di file ".env":

```
APP_NAME=TUTORIAL_DOTENV
BASE_URL=http://localhost:3000
NUM_OF_CORES=100
```

Dengan kata lain:

- Untuk mengakses APP_NAME di ".env", process.env.APP_NAME
- Untuk mengakses BASE_URL di ".env", process.env.BASE_URL
- Untuk mengakses NUM_OF_CORES di ".env", process.env.NUM_OF_CORES

Semua tipe data yang tadi diimpor adalah string, sehingga jika perlu tipe lain mungkin perlu dikonversi dulu.

Kira-kira seperti itu...

Penutup

Sekarang, seharusnya Anda telah mengenal dotenv dan mampu menggunakannya.

Belajar Node JS Mengenal Axios

Source Code Project Ini

contoh_nodejs_axios

Pendahuluan

Pada aplikasi tertentu, ada saat di mana aplikasi tersebut perlu mengakses suatu URL dengan sebuah request.

Misalnya, saat aplikasi tersebut akan mengonsumsi REST API target.

Untuk melakukannya, bisa tanpa library eksternal. Misalnya, dengan modul https bawaan Node.js.

Tapi, menurut saya, karena Axios cukup popular, saya lebih suka untuk menggunakannya.

Jadi, tidak ada salahnya mempelajari library tersebut.

Lebih Lanjut tentang Axios

Axios adalah sebuah library JavaScript yang populer untuk melakukan HTTP requests dari browser atau Node.js. Ini adalah alat yang kuat dan mudah digunakan untuk berinteraksi dengan API dan mengambil data dari server.

Beberapa fitur dan keunggulan dari Axios termasuk:

- Sintaks yang Mudah Dipahami: Axios menggunakan promise-based API yang membuatnya mudah untuk membuat HTTP requests dan menangani respons.
- Dukungan untuk Browser dan Node.js: Axios dapat digunakan baik di browser maupun di server-side JavaScript (Node.js).
- Intersepsi Request dan Response: Anda dapat dengan mudah mengintersep request dan response untuk melakukan transformasi atau menambahkan header secara dinamis.
- Dukungan untuk Pembatalan Request: Axios memungkinkan Anda untuk membatalkan request yang sedang berlangsung, yang dapat berguna untuk meningkatkan kinerja dan menghindari request yang tidak perlu.
- Dukungan untuk XSRF Protection: Axios menyediakan mekanisme bawaan untuk melindungi aplikasi Anda dari serangan XSRF (Cross-Site Request Forgery) dengan menggunakan token XSRF.
- Dukungan untuk Berbagai Jenis Request: Axios mendukung berbagai jenis HTTP requests seperti GET, POST, PUT, DELETE, dan sebagainya.

- Dukungan untuk Mengirim Data dalam Bentuk JSON: Axios secara otomatis mengubah data yang dikirim dalam bentuk objek JavaScript menjadi format JSON.

Karena keunggulan-keunggulannya ini, Axios menjadi pilihan yang populer bagi para pengembang untuk melakukan komunikasi dengan server dalam aplikasi web mereka.

Untuk mengetahui lebih lanjut tentang Axios, kunjungi:

https://axios-http.com/docs/intro

Tujuan

Tujuan dari artikel ini adalah:

- Pembaca mengenal Axios.
- Pembaca mampu mencoba menggunakan Axios.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.
- Bisa mengakses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, buatlah project Node.js bernama "contoh_nodejs_axios" dengan cara yang telah dijelaskan di "Belajar Node JS Cara Membuat Project".

Selanjutnya, ubah file "package.json" menjadi seperti ini:

```
{
  "name": "contoh_nodejs_axios",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "axios": "^1.6.8"
  }
}
```

Selanjutnya, isi file "index.js" dengan kode ini:

```
// import modul
const axios = require("axios");

// buat method async yang otomatis langsung dijalankan
(async () => {
    // gunakan axios dengan method get.
    // di sini bisa pakai await karena sudah dimasukkan kedalam method async.
    const ret = await axios.get("https://quotes.toscrape.com");

    // tampilkan hasilnya.
    console.log(ret);
})();
```

Selanjutnya, jalankan:

```
npm install
```

Sekarang, jalankan aplikasi ini:

```
node index.js
```

Nanti akan muncul output di command line:

```
{
  status: 200,
  statusText: 'OK',
  headers: Object [AxiosHeaders] {
    date: 'Tue, 07 May 2024 16:10:58 GMT',
    'content-type': 'text/html; charset=utf-8',
    'transfer-encoding': 'chunked',
    connection: 'keep-alive',
    'strict-transport-security': 'max-age=0; includeSubDomains; preload'
  },
... dan selanjutnya
```

Pembahasan

Kode Node.js di bagian "Langkah-Langkah" adalah contoh sederhana penggunaan Axios untuk melakukan HTTP GET request ke URL https://quotes.toscrape.com dan menampilkan responsnya.

Mari kita jelaskan baris per baris:

- `const axios = require("axios");`: Baris ini mengimpor modul Axios ke dalam kode. Ini memungkinkan kita untuk menggunakan semua fitur dan fungsi yang disediakan oleh Axios.

- `(async () => { ... })();`: Ini adalah sebuah fungsi anonim yang didefinisikan dan langsung dieksekusi (self-invoking). Fungsi ini menggunakan async/await untuk mengelola asinkronitas. Dengan demikian, kita dapat menggunakan kata kunci await di dalamnya untuk menunggu hasil dari operasi asinkron seperti HTTP request.
- `const ret = await axios.get("https://quotes.toscrape.com");`: Di dalam fungsi async, ini adalah langkah utama. Kami menggunakan metode `axios.get()` untuk membuat permintaan GET ke URL https://quotes.toscrape.com . Kata kunci await memungkinkan kode untuk menunggu sampai permintaan selesai dan hasilnya dikembalikan sebelum melanjutkan eksekusi.
- `console.log(ret);`: Setelah permintaan selesai dan hasilnya diterima, responsnya disimpan dalam variabel `ret`. Kemudian, kita mencetak respons tersebut menggunakan `console.log()` untuk melihat apa yang telah diterima dari server.

Jadi, secara keseluruhan, kode ini menggunakan Axios untuk membuat permintaan GET ke https://quotes.toscrape.com secara asinkron dan menampilkan responsnya dalam konsol.

Penutup

Begitulah contoh sederhana penggunaan Axios dalam project Node.js.

Anda boleh coba method-method lainnya seperti POST, PUT, DELETE, dan lain-lain.

Belajar Node JS Mengenal Puppeteer

Source Code Project Ini

contoh_nodejs_puppeteer

Pendahuluan

Setelah Anda belajar tentang Axios pada "Belajar Node JS Mengenal Axios", mungkin Anda akan menyadari sesuatu.

Yakni, isi HTML yang diperoleh melalui Axios tadi masih bisa diolah.

Dengan library tertentu, mengekstrak data berharga dari HTML yang dikembalikan bisa dilakukan dengan mudah.

TAPI... itu dengan asumsi bahwa data berharga tersebut ada di HTML nya, bukan tersembunyi dalam JavaScript.

Lalu, bagaimana dengan halaman web yang kontennya dinamis, dalam artian hanya tampil setelah diolah dengan JavaScript?

Jawabannya ada pada Puppeteer.

Puppeteer itu sendiri mengandalkan web browser untuk melakukan perolehan data.

Analoginya:

- Data yang dikembalikan dari Axios menyerupai browser view source.
- Data yang dikembalikan dari Puppeteer menyerupai browser inspect element.

Di artikel ini, saya akan memberikan contoh sederhana untuk menggunakan Puppeteer.

Lebih Lanjut tentang Puppeteer

Puppeteer adalah sebuah library Node.js yang dikembangkan oleh tim Google Chrome.

Ini memungkinkan otomatisasi browser headless (tanpa UI) melalui antarmuka yang ramah pengembang.

Berikut adalah beberapa poin penting tentang Puppeteer:

- Automasi Browser: Puppeteer memungkinkan Anda mengendalikan browser Chrome atau Chromium secara programatik melalui JavaScript. Anda dapat mengotomatiskan interaksi seperti klik, isi formulir, navigasi, dan lainnya.
- Headless Mode: Salah satu fitur utama dari Puppeteer adalah kemampuannya untuk menjalankan browser dalam mode tanpa kepala (headless). Ini berarti browser tidak akan memiliki antarmuka pengguna grafis (GUI), yang membuatnya lebih efisien untuk menjalankan skrip otomatisasi di latar belakang.
- Testing Otomatis: Puppeteer sangat berguna untuk pengujian otomatis di situs web. Anda dapat mengotomatiskan serangkaian aksi pengguna untuk menguji fungsi dan kinerja situs web Anda.
- Web Scraping: Dengan menggunakan Puppeteer, Anda dapat melakukan web scraping, yaitu mengekstrak data dari halaman web secara otomatis. Ini dapat berguna untuk mengumpulkan informasi dari situs web yang tidak menyediakan API.
- Rendering dan Screenshot: Puppeteer dapat digunakan untuk merender halaman web dan mengambil tangkapan layar (screenshot). Ini berguna jika Anda perlu membuat laporan visual atau melakukan pemantauan visual terhadap perubahan pada situs web.
- Deteksi dan Bypass CAPTCHA: Dengan menggunakan Puppeteer, Anda dapat memprogram agar melakukan navigasi ke halaman web yang membutuhkan CAPTCHA, dan dalam beberapa kasus, Anda bahkan dapat membuat skrip yang secara otomatis memecahkan CAPTCHA.
- Dukungan untuk Chrome DevTools Protocol: Puppeteer dibangun di atas protokol DevTools Chrome, yang memberikan kontrol penuh terhadap perilaku browser. Ini memungkinkan Anda untuk melakukan debugging, analisis, dan optimisasi performa.

Keseluruhan, Puppeteer adalah alat yang sangat kuat dan fleksibel untuk mengotomatisasi interaksi dengan browser, baik untuk pengujian otomatis, web scraping, atau tujuan lainnya.

Tujuan

Tujuan dari artikel ini adalah:

- Pembaca mengenal Puppeteer.
- Pembaca mampu mencoba menggunakan Puppeteer.

Prasyarat

Berikut ini adalah prasyarat dari tutorial ini:

- Menggunakan sistem operasi Windows 10 ke atas.
- Men-download dan meng-install Node.js dan NPM.
- Bisa mengakses Node.js dan NPM dari PowerShell di folder manapun.

Langkah-Langkah

Pertama, buatlah project Node.js bernama "contoh_nodejs_puppeteer" dengan cara yang telah dijelaskan di "Belajar Node JS Cara Membuat Project".

Selanjutnya, ubah file "package.json" menjadi seperti ini:

```
{
  "name": "contoh_nodejs_puppeteer",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "puppeteer": "^22.8.0"
  }
}
```

Selanjutnya, ubah file "index.js" menjadi seperti ini:

```
// import module
const puppeteer = require("puppeteer");

(async () => {
    // jalankan browser dan buka halaman kosong
    const browser = await puppeteer.launch();
    const page = await browser.newPage();

    // navigasi ke URL ini
    // sebagai referensi, kunjungi https://quotes.toscrape.com dan lihat menunya.
    // di sana ada menu bertuliskan "reading".
    // disclaimer: mungkin halaman tersebut telah berubah saat Anda
mengunjunginya.
    await page.goto("https://quotes.toscrape.com");

    // terapkan ukuran layar
    await page.setViewport({ width: 1024, height: 768 });

    // tunggu hingga elemen dengan selector ini muncul
    await page.waitForSelector("body > div > div:nth-child(2) > div.col-md-4.tags-
box > span:nth-child(7) > a");

    // baca teks-nya
    let result = await page.evaluate(() => {
        let elements = Array.from(document.querySelectorAll("body > div > div:nth-
```

```
child(2) > div.col-md-4.tags-box > span:nth-child(7) > a"));
        let textContent = elements.map((element) => {
            return element.textContent;
        });
        return textContent;
    });

    // tampilkan hasilnya
    console.log(result);

    // tutup puppeteer browser
    await browser.close();
})();
```

Selanjutnya, jalankan:

```
npm install
```

Sekarang, jalankan aplikasi ini:

```
node index.js
```

Nanti akan muncul output di command line:

```
[ 'reading' ]
```

Pembahasan

Kode ini menggunakan Puppeteer untuk mengotomatisasi browser Chrome atau Chromium dalam menjalankan serangkaian tindakan pada halaman web tertentu.

Mari kita jelaskan baris per baris:

- `const puppeteer = require("puppeteer");`: Ini mengimpor modul Puppeteer ke dalam skrip Node.js Anda. Ini memungkinkan Anda menggunakan semua fungsi dan fitur yang disediakan oleh Puppeteer.
- `(async () => { ... })();`: Ini adalah fungsi anonim yang didefinisikan dan segera dieksekusi (self-invoking). Ini menggunakan async/await untuk mengelola asinkronitas dalam JavaScript.
- `const browser = await puppeteer.launch();`: Baris ini memulai browser menggunakan metode `puppeteer.launch()`. Ini akan membuat instance baru dari browser Chrome atau Chromium yang akan digunakan untuk menjalankan tindakan-tindakan berikutnya.
- `const page = await browser.newPage();`: Ini membuat halaman baru di dalam browser yang telah dibuka sebelumnya. Instance halaman baru ini akan digunakan untuk melakukan interaksi selanjutnya, seperti navigasi ke URL tertentu dan mengevaluasi elemen-elemen di dalamnya.

- `await page.goto("https://quotes.toscrape.com");`: Ini adalah langkah penting, di mana skrip melakukan navigasi ke URL "https://quotes.toscrape.com". Ini membuka halaman web yang ingin kita ambil informasinya.
- `await page.setViewport({ width: 1024, height: 768 });`: Baris ini menetapkan ukuran viewport (area yang terlihat oleh pengguna) pada halaman web. Dalam contoh ini, viewport ditetapkan dengan lebar 1024 piksel dan tinggi 768 piksel.
- `await page.waitForSelector("body > div > div:nth-child(2) > div.col-md-4.tags-box > span:nth-child(7) > a");`: Ini adalah langkah yang memastikan bahwa elemen dengan selector yang ditentukan telah muncul di halaman. Skrip akan menunggu hingga elemen tersebut tersedia sebelum melanjutkan.
- `let result = await page.evaluate(() => { ... });`: Ini menggunakan metode `page.evaluate()` untuk mengeksekusi fungsi evaluasi di dalam lingkungan halaman web. Dalam kasus ini, itu akan mengeksekusi fungsi yang mencari elemen dengan selector tertentu dan mengambil teksnya.
- `console.log(result);`: Hasil yang diperoleh dari evaluasi sebelumnya kemudian dicetak ke konsol. Ini akan menampilkan teks yang diambil dari elemen yang diidentifikasi sebelumnya.
- `await browser.close();`: Langkah terakhir adalah menutup browser menggunakan browser.close() setelah selesai melakukan tindakan-tindakan yang diinginkan.

Ingat bahwa pada `let result = await page.evaluate(() => { ... });`, kode dalam blok fungsi yang menjadi parameter page.evaluate, scope-nya bukan di Node.js, melainkan di browser, jadi, itu JavaScript-nya browser.

Keseluruhan, skrip ini mengotomatisasi browser untuk membuka halaman "https://quotes.toscrape.com", menunggu hingga elemen tertentu muncul, dan kemudian mengambil teks dari elemen tersebut untuk kemudian dicetak ke console.

Penutup

Begitulah contoh sederhana penggunaan puppeteer.

Jika ingin belajar lebih lanjut, kunjungi dokumentasinya.