



Node JS

Gold - Chapter 4 - Topic 3

Selamat datang di **Chapter 4 Topic 3** online course
Full-stack Web dari Binar Academy!





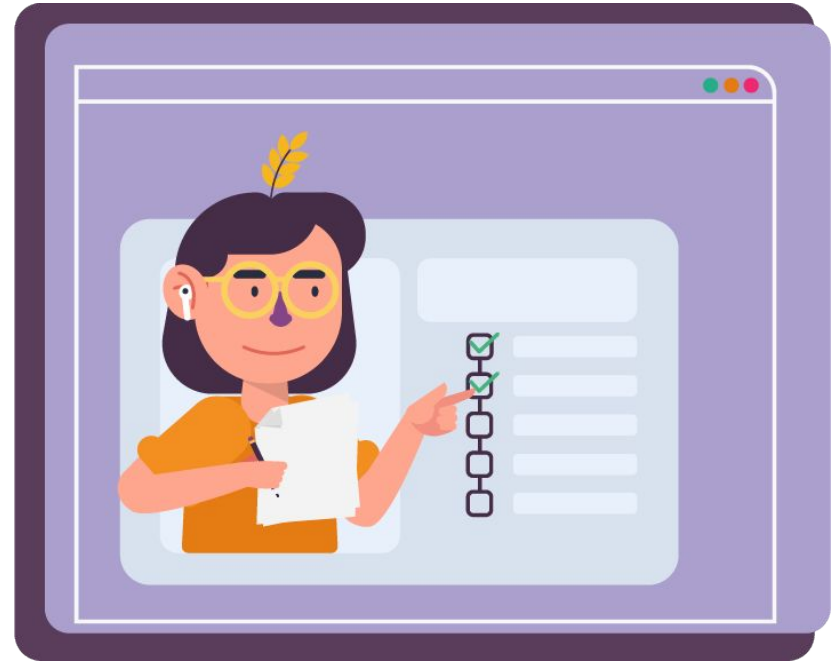
Nah, pada topik ini kita bakal bahas tuntas tentang **runtime environment Node.JS** serta **bagaimana cara menggunakannya**. Cus langsung aja~





Detailnya, kita bakal bahas hal-hal berikut ini:

- Pemahaman runtime environment
- Module pada Node.js
- Menginisiasi project menggunakan Package Manager
- Menggunakan module untuk read & write file
- Perbedaan menjalankan code di Node.js dan browser





Di chapter sebelumnya, kita sudah membahas yang namanya Javascript.

Dari kemarin kita baru menjalankan kode-kode Javascript di dalam browser.

Nah, kalau sekarang kita belajar **Node.Js**, maka selanjutnya kita dapat menjalankan Javascript langsung dari terminal. Menarik kan? Yukkk~





Kamu tahu nggak sih, sebelum tahun 2009, Javascript hanya dapat dijalankan melalui browser saja, karena browser memiliki engine khusus untuk menjalankan Javascript.

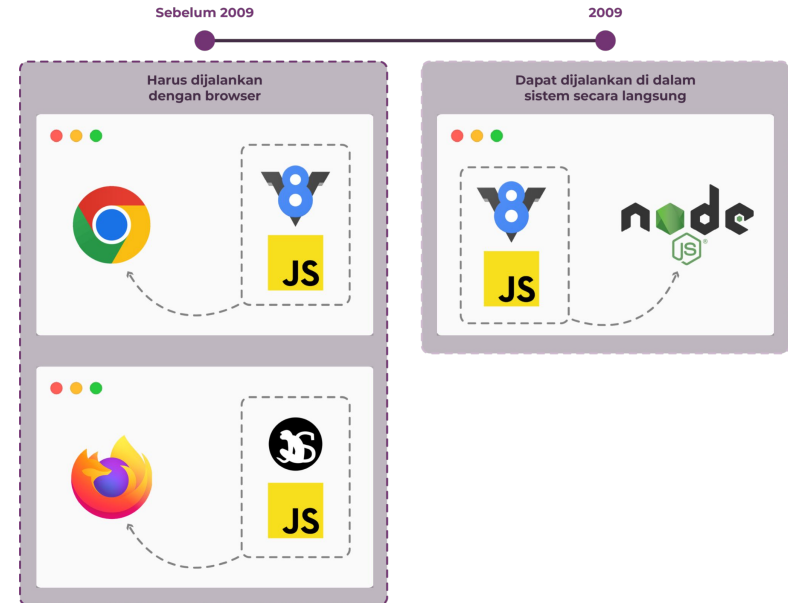
Berikut beberapa engine yang terkenal:

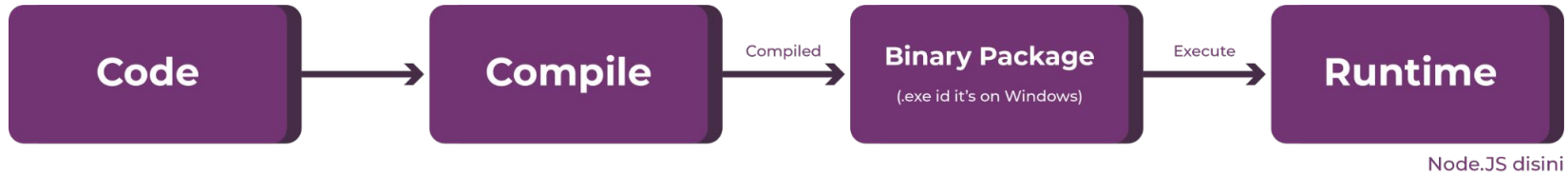
- **V8 (Chrome)**
- **SpiderMonkey (Firefox)**

Nah, **Ryan Dahl** pada tahun 2009 membuat runtime environment untuk Javascript di dalam sistem secara langsung.

Runtime tersebut memakai V8 Engine dari Chrome yang dimodifikasi sedemikian rupa agar dapat berjalan di dalam sistem secara langsung.

Runtime itulah yang kita sebut **Node.js**~





Jadi, sebenarnya program itu memiliki beberapa fase di dalam komputer ketika dia akan di deliver dan dijalankan. Runtime merupakan saat dimana aplikasi itu berjalan. Dan Node.js memiliki peran penting dalam fase ini.

Javascript adalah interpreted language, dimana program tidak perlu melakukan kompilasi. Sehingga, semua informasi **kode Javascript akan dieksekusi langsung di dalam runtime environment.**

Di bahasa lain bisa saja berbeda. Sebagai contoh Java. Ketika kita ingin menjalankan aplikasi Java, kita harus melakukan compile kode terlebih dahulu menjadi Binary Package (.exe kalau di windows). Baru setelah dieksekusi aplikasinya, aplikasi tersebut akan berada pada runtime environment.



Biar makin paham kita bakal coba mengeksekusi file Javascript menggunakan Node Js. Ikutin langkahnya ya!

1. Buatlah sebuah directory dan di dalam directory tersebut buat sebuah file javascript dengan nama **'index.js'**
2. Buat program sederhana di dalam file index.js tadi
3. Perhatikan contoh kode di samping!
4. Dari terminal, masih di dalam directory tadi, ketik perintah **'node index.js'**
5. Outputnya akan menghasilkan tulisan 'hello world'

Nah, kira-kira begitulah cara mengeksekusi file Javascript menggunakan Node Js~

```
// index.js  
console.log("hello world")
```




Kalo tadi Node.Js merupakan tools untuk menjalankan code, sekarang, kita akan belajar **module apa saja yang bisa kita gunakan di Node.Js~**



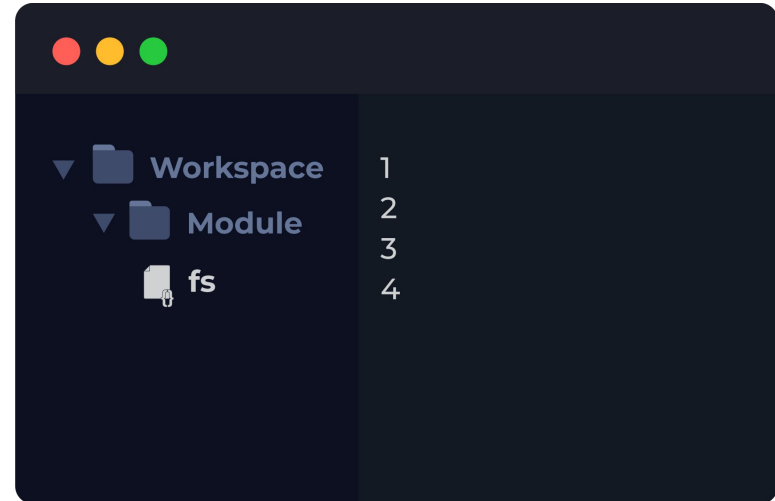


Module

Module adalah **seperangkat kode yang dapat kita pakai berdasarkan fungsi modul tersebut.**

Contohnya, Node.JS memiliki module bernama fs, module ini berguna untuk melakukan hal-hal yang terkait dengan file sistem. Sebagai contoh, Read File, atau Write File. Jadi, kita tidak perlu menulis kode itu dari nol.

Nah~ module ternyata punya beberapa jenis lho. Kita cek satu persatu, yuk!





Setelah tau definisinya, lanjut kita bahas tiga tipe module di Node.Js~

- **Core Module**

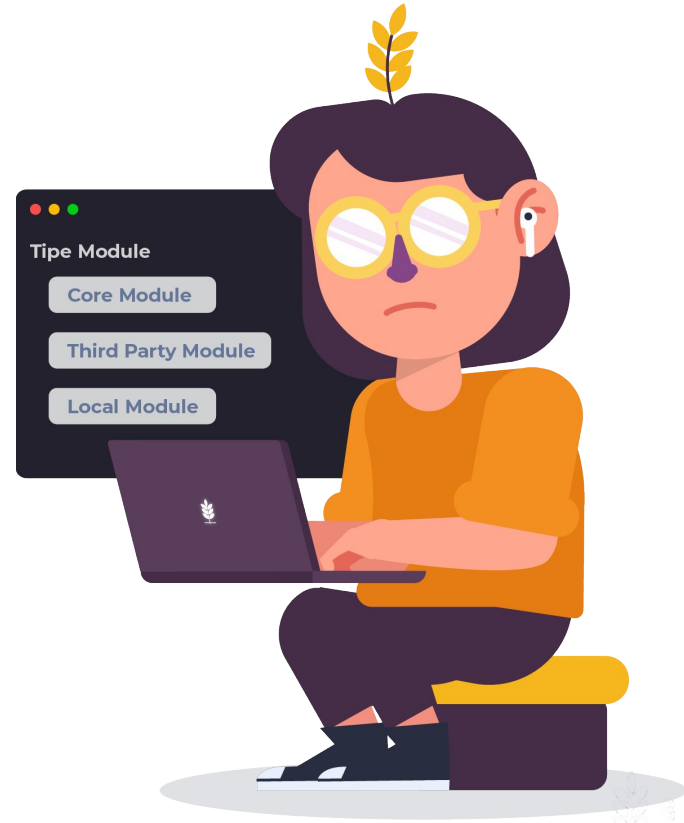
Module yang **sudah ada dan dapat dipakai langsung** setelah instalasi Node.JS. Module ini dibuat langsung oleh developer Node.JS.

- **Third Party Module**

Module yang **dibuat oleh pihak ketiga diluar developer Node.JS**. Untuk memakainya kita harus melakukan instalasi terlebih dahulu.

- **Local Module**

Module yang **kita buat sendiri**.





Biar makin ngerti, habis ini kita bakal belajar sambil praktik dikit-dikit lewat **inisialisasi project Node.Js**, ya!



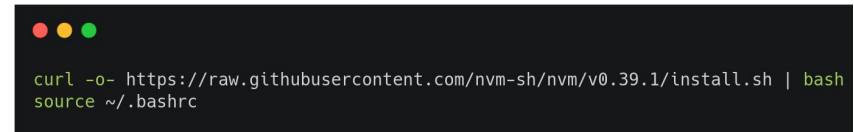


Instalasi Node.Js

Sebelum kita eksplorasi lebih jauh, tentunya kita harus punya Node.Js. Nah, untuk instalasi Node Js kita bisa menggunakan 3 cara yaitu :

1. Dengan cara **download package nya di** <https://nodejs.org>. Jangan lupa untuk pilih versi yang stable yah!
2. Cara yang kedua melalui terminal. Biasanya user linux atau mac menggunakan cara ini, **tinggal tulis perintahnya di terminal Linux**, berikut perintahnya "*sudo apt-get install node*", cek gambar di samping yaaa~
3. **Recommended**, Menggunakan [NVM](#) (MacOS, Ubuntu, & Ubuntu WSL Only) Lihat gambar disamping.

Ketika kita melakukan instalasi Node.Js, maka secara otomatis akan terinstall juga **package managernya** atau **NPM**. Nah, terus NPM itu fungsinya apa ya?



```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
source ~/.bashrc
```



Instalasi Package Manager atau NPM

Package manager adalah sebuah **aplikasi yang digunakan untuk mengatur (menambah ataupun menghapus) third party module dalam proyek yang kita kerjakan.**

Selain NPM, Yarn merupakan salah satu alternatif package manager untuk Node Js. Yarn adalah **package manager unofficial dari Node.JS namun sangat banyak yang menggunakan**, dikarenakan lebih minimalis dan cepat dibanding dengan NPM.

Untuk cara instalasi yarn, kamu bisa mengunjungi link ini:

[Download Yarn](#)





Setelah menginstall **Node.Js** dan **package manager**, sekarang kamu sudah bisa membuat proyek Node.Js pertama kamu.

Coba ikutin langkah ini ya :

1. Buat directory baru, kemudian masuk kedalam directory tersebut
2. Ketikan perintah `npm init` atau `yarn init`, (tergantung package manager yang kamu gunakan) di terminal
3. Akan muncul beberapa pertanyaan, seperti nama proyek, deskripsi proyek, author, dll. Seperti gambar disamping!
4. Kamu boleh mengisi atau mengosongkan pertanyaan tersebut.

```
user@ubuntu: ~$ mkdir my-app && cd my-app
# inisialisasi proyek dengan npm
user@ubuntu: ~$ npm init
package name: (my-app)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
```



Selamat! Kamu udah berhasil bikin project pertama kamu. Selanjutnya, kita akan mengkonfigurasi package manager menggunakan **package.json**~





Package.json

Sekarang, liat isi dari proyek yang baru kamu buat! Saat membuat proyek menggunakan package manager, maka secara otomatis akan membuat sebuah file yang bernama '**package.json**'.

File package.json ini berisi informasi dari proyek Node Js nya. Perhatikan isi file package.json di samping deh!


```
{  
  "name": "my-app",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "test"  
  },  
  "author": "",  
  "license": "ISC"  
}
```



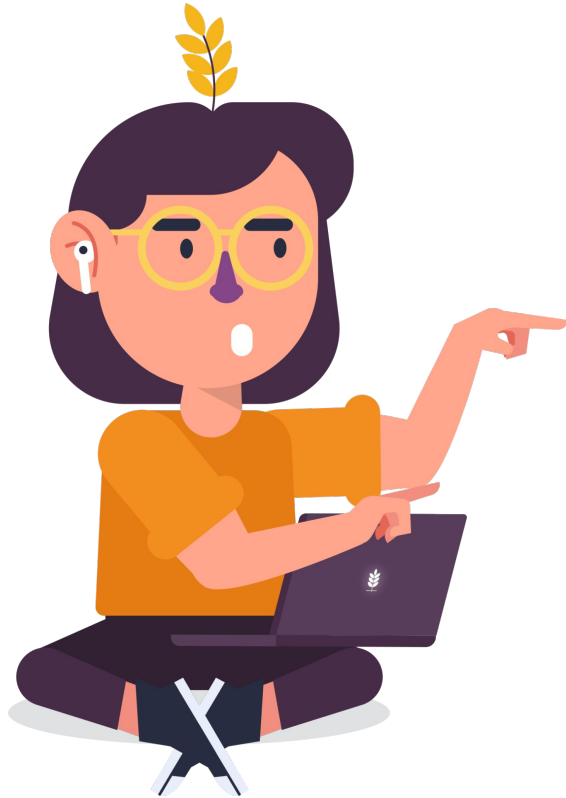
Script

Di dalam file package.json ada sebuah properti yang bernama scripts, yang mana **properti ini berbentuk data string yang berisikan perintah-perintah terminal.**

Script ini berguna untuk mempersingkat dan mendefinisikan perintah-perintah yang terkait terhadap project tersebut. Nanti, **script yang kita buat nanti akan kita panggil menggunakan package manager (NPM).**



```
// index.js  
console.log("hello world")
```



Untuk memahami tentang fungsi dari script ini, kita akan membuat latihan. Di dalam directory proyek tadi coba buat sebuah file Javascript dengan nama `index.js`, di dalam file `js` buat sebuah program untuk menampilkan string “hello world” ke layar.

Setelah itu kita akan menambahkan scripts ke file `package.json`.



Setelah tau definisinya, yuk lanjut menambah sebuah script dengan nama "start". **Script 'start' ini berfungsi untuk mengeksekusi file index.js tadi.**

Nah, sekarang dari terminal di dalam directory proyek coba jalan perintah '**npm run start**'. Harusnya, hasilnya adalah tulisan "hello world" akan di cetak di layar.

```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "test",
    "start": "node index.js"
  },
  "author": "",
  "license": "ISC"
}
```



Third Party Module

Selanjutnya kita akan mencoba **menambah 2 third party module** ke dalam projek kita, yaitu module chalk dan nodemon. Perhatikan kode di samping ya!

Sekarang coba liat lagi file package.json.



```
# menginstall dependencies
user@ubuntu: ~$ npm install chalk --save
# menginstall dev dependencies
user@ubuntu: ~$ node install nodemon --save-dev
```



Setelah kita menambah 2 module third party tadi, maka akan muncul update di file package.json. Ada **2 properti baru yaitu 'dependencies' dan 'devDependencies'**

Properti ini akan **memberitahu package manager, module apa saja yang dipakai dalam project**. Sehingga akan mempermudah kita ketika kita ingin mendeliver project ini ke server ataupun ke komputer lain. Kita tidak perlu mengingat-ingat module apa saja yang perlu diinstal.

Lalu apa perbedaan antara 'dependencies' dan 'devDependencies' ?

```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "test"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "chalk": "^4.1.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.4"
  }
}
```



Apasih bedanya Dependencies dan devDependencies?

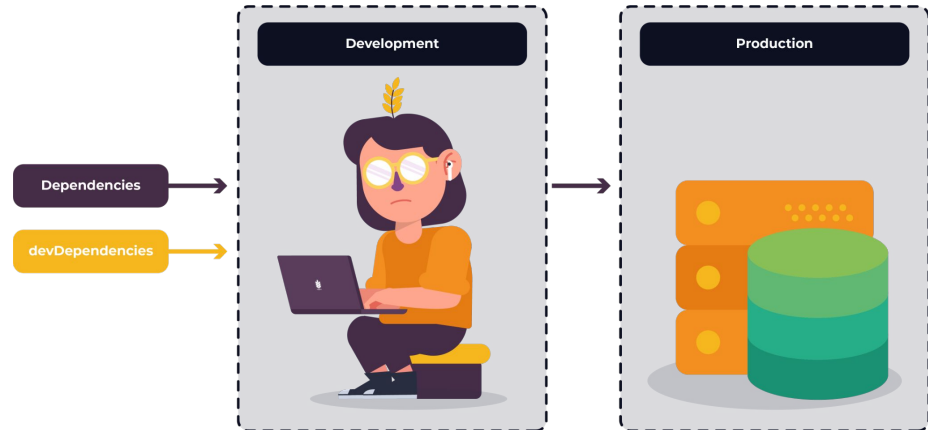
Dependencies

berisi daftar module yang akan kita gunakan sampai tahap **production**. Artinya modulnya akan ikut di install ke server nantinya.

devDependencies

berisi daftar module yang hanya digunakan saat **development** atau pengerjaan proyek dan tidak akan ikut di install di server.

Saat menginstall 2 module tadi, selain update di package.json, secara otomatis akan di tambah juga directory bernama **"node_modules"**. Nah, di dalam folder ini lah third party module disimpan.





Setelah membuat projek Node js pertama kita, sekarang kita akan bermain-main menggunakan module.

Untuk point ini, kita kan mencoba memakai Core Module yang bernama os.Module ini kita pakai untuk mengetahui berapa memory yang tersedia di system yang menjalankan aplikasi ini.

Caranya, buka file index.js yang kita buat tadi, buat script seperti gambar di samping. Gampang kan?



```
// index.js
// Ini akan mengimport module bernama `os`
const os = require('os')

console.log('Free Memory:', os.freemem())
```




Export and Import module

Export module

Statement yang dipakai untuk **mengekspos suatu data dari suatu file ke file manapun** yang mengimpor dia. Sebagai contoh, disini kita akan membuat file baru bernama segitiga.js.

Kemudian di dalam file tersebut buat sebuah function bernama luasSegitiga. Selanjutnya luasSegitiga inilah yang kita sebut sebagai local module, yang nantinya akan bisa kita export. Cek gambar di samping yaa~

Untuk mengeksport sesuatu di dalam file, kita menggunakan module objek, lalu kita assign property bernama exports dengan apapun yang ingin kita export.

```
// ini file segitiga.js
function luasSegitiga(alas, tinggi) {
  return alas * tinggi / 2
}

module.exports = luasSegitiga
```



Import Module

Import adalah suatu **statement yang dipakai untuk mengambil, atau memakai suatu data dari file lain, atau module lain.**

Sebagai contoh, kita bisa import file yang memiliki function untuk menghitung luas segitiga tadi. Perhatikan kode di samping ya.

```
//index.js
// Ini akan mengimport module bernama `os`
const os = require('os')
const luasSegitiga = require('./segitiga.js')

console.log('Free Memory:', os.freemem())

console.log(luasSegitiga(3, 4))
```



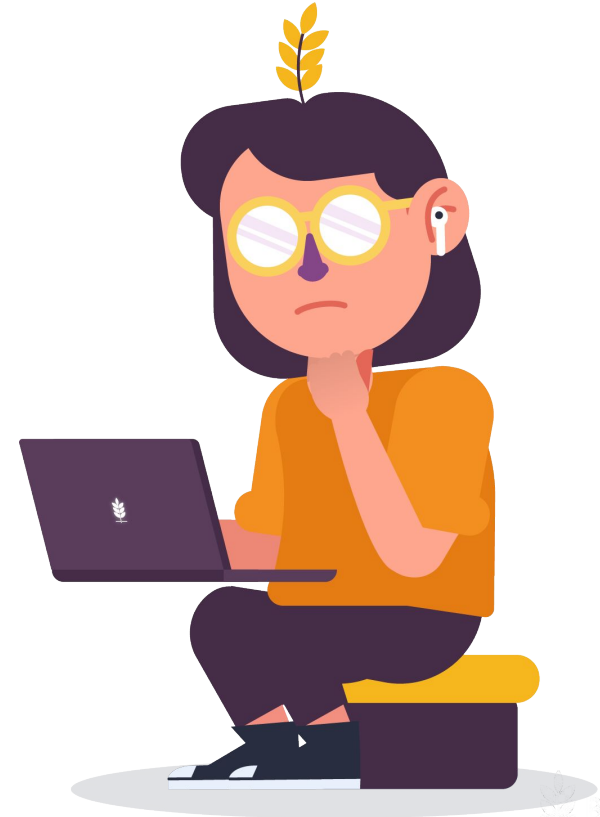
Sebelumnya, kita sudah belajar bagaimana menggunakan module di Node.js. Kita juga sudah belajar bagaimana cara export dan import modulnya.

Biar tambah paham, kita akan latihan lagi, nih. Kali ini kita akan latihan menggunakan **fs module**. Kuyy~



Seperti yang kita tahu Node.js berjalan langsung di dalam system, maka dari itu kita bisa mengakses file system. Fs Modul menyediakan banyak fungsi yang sangat berguna untuk mengakses dan berinteraksi dengan file system.

Karena kita bisa mengakses file system, maka kita bisa melakukan **read** and **write** langsung disitu Hmmm... apa ya pengertiannya?





Read and Write File

Read

Read artinya **kita akan membaca isi dari sebuah file**. Disini, buat sebuah file bernama 'text.txt', di dalam file tersebut tuliskan nama kamu.

Selanjutnya kita akan membaca file tersebut dari file index.js menggunakan fs module. Perhatikan kode di samping ya!

- Parameter pertama dari fs.readFileSync ini adalah **path relative terhadap process.cwd()** atau lokasi dari kita menjalankan node namafile.
- **Parameter kedua adalah text-encoding**. Jika kita tidak menambahkan parameter kedua, maka hasil dari function tersebut berupa Buffer

```
// index.js
// Ini akan mengimport module bernama `fs`
const fs = require('fs')

const isi = fs.readFileSync('./text.txt','utf-8')

console.log(isi)

// ouputnya Nama kamu
```



Write

Ketika kita ingin menulis suatu file, maka kita bisa memakai method bernama `writeFileSync`. Method ini kita pakai **untuk menyimpan string ke dalam suatu file**. Perhatikan contoh kode di samping!

Ketika file tersebut dijalankan, maka akan terbuat file baru bernama `test.txt` yang berisi "I love Binar".

```
// index.js
// Ini akan mengimport module bernama `fs`
const fs = require('fs')

fs.writeFileSync('./test.txt', "I love Binar")
```



Selanjutnya kita akan membuat sebuah data dan menyimpan data tersebut dalam bentuk json.

Pertama buat sebuah file bernama create.js. Selanjutnya tuliskan kodenya seperti gambar di samping yaa~

```
const createPerson = function (person) {  
  fs.writeFileSync('./person.json', JSON.stringify(person))  
  return person;  
}  
  
const Sabrina = createPerson({  
  name: 'Sabrina',  
  age: 22,  
  address: 'BSD'  
})
```



Terakhir tinggal menampilkan data dari Sabrina tadi.

Di file index.js import file person.json yang telah kita buat sebelumnya.

```
// index.js
// Ini akan mengimport module bernama `fs`
const fs = require('fs')
const sabrina = require('./person.json')

console.log(sabrina)
```




Nah, sekarang kamu udah paham kan kalo Node.js berjalan langsung di dalam system dan Fs Modul menyediakan banyak fungsi yang berinteraksi dengan file system.

Selanjutnya kita akan melihat perbedaan **Node.js** dan **Browser**, cekidottt~



Apa sih bedanya Node.js dan Browser?

| | Node.js | Browser |
|-------------------------|--|---|
| Menulis kode | Dapat mengakses system secara langsung | Hanya bisa mengakses web API saja, seperti DOM dan fetch |
| Menjalankan kode | Melalui terminal yang sudah ditulis dengan file .js | Melalui file HTML yang mengimpor Javascript |
| Complexity | Project yang menggunakan salah satu library / framework Javascript baik saat masa development ataupun pada saat deployment | Tidak ada sistem modul di dalamnya, semua fungsi yang diimpor melalui link tinggal pake aja. |
| Kegunaan | Untuk mengembangkan project web ataupun mobile yang bersifat hybrid | Merender halaman secara dimanis |

Saatnya kita
Quiz!





1. Pemahaman yang tepat mengenai Node.Js adalah...

- A. Runtime Javascript
- B. Framework Javascript
- C. Bahasa Pemrograman



1. Pemahaman yang tepat mengenai Node.Js adalah...

- A. Runtime Javascript
- B. Framework Javascript
- C. Bahasa Pemrograman

Node Js bukan sebuah bahasa pemrograman atau framework. Node js adalah runtime javascript.



2. Ketika mengerjakan project A Sabrina membuat module sendiri khusus untuk projectnya tersebut. Disebut apakah module yang dibuat oleh Sabrina?

- A. Core module
- B. Local module
- C. Third party module



2. Ketika mengerjakan project A Sabrina membuat module sendiri khusus untuk projectnya tersebut. Disebut apakah module yang dibuat oleh Sabrina?

- A. Core module
- B. Local module**
- C. Third party module

Core Module adalah module yang dibuat langsung oleh developer Node.JS.
Third Party Module adalah module yang dibuat oleh pihak ketiga diluar developer Node.JS
Local Module adalah module yang kita buat sendiri.



3. Menurut kamu, Statement yang digunakan untuk mengambil data dari suatu file atau module adalah...

- A. import
- B. export



3. Menurut kamu, Statement yang digunakan untuk mengambil data dari suatu file atau module adalah...

- A. import**
- B. export

Import adalah statement yang di gunakan untuk mengambil data dari suatu file sedangkan Export adalah statement yang di gunakan untuk mengirim data dari suatu file.



4. Manakah pernyataan dibawah ini yang salah

- A. NPM dapat digunakan untuk mengatur (menambah ataupun menghapus) third party module dalam proyek Node.js
- B. Yarn dapat digunakan untuk mengatur (menambah ataupun menghapus) third party module dalam proyek Node.js
- C. PIP dapat digunakan untuk mengatur (menambah ataupun menghapus) third party module dalam proyek Node.js



4. Manakah pernyataan dibawah ini yang salah

- A. NPM dapat digunakan untuk mengatur (menambah ataupun menghapus) third party module dalam proyek Node.Js
- B. Yarn dapat digunakan untuk mengatur (menambah ataupun menghapus) third party module dalam proyek Node.Js
- C. **PIP dapat digunakan untuk mengatur (menambah ataupun menghapus) third party module dalam proyek Node.Js**

PIP adalah package manager untuk Python. Package manager untuk node js adalah NPM dan Yarn.



5. Di dalam package.json terdapat property “dependencies” yang berisi module-module yang di gunakan dalam projek. Jenis module yang di tampung dalam property dependencies adalah ...

- A. Third party module
- B. Local module
- C. Core module



5. Di dalam package.json terdapat property “dependencies” yang berisi module-module yang di gunakan dalam projek. Jenis module yang di tampung dalam property dependencies adalah ...

- A. Third party module**
- B. Local module
- C. Core module

Dependencies berisi berisis informasi third party module yang kita gunakan dalam projek yang kita kerjakan.



Referensi dan bacaan lebih lanjut~

1.





Nah, selesai sudah pembahasan kita di Chapter 4 Topic 3 ini.

Selanjutnya, kita bakal bahas **aturan-aturan dalam menuliskan koding**.

Penasaran kayak gimana? Cus langsung ke topik selanjutnya~



Terima Kasih!



Next Topic

loading...