Express

before installing express, you must install the node.

Node.js คือ JavaScript runtime สำหรับฝั่ง Server และเป็น Open Source ซึ่งเขียนด้วยภาษา JavaScript สรุป NodeJS ก็คือ Platform ตัวหนึ่งที่เขียนด้วย JavaScript สำหรับเป็น Web Server ซึ่งจะแตกต่างจากการเขียน JavaScript ที่ฝั่ง Client เช่น ไม่มี document.getElementById() หรือ window.alert() แต่จะมีสิ่งที่เรียกว่า Module เข้ามาแทน ซึ่งเป็นคล้ายๆ built-in library

- Node.js เป็น Non-blocking I/O นั่นคือ มันจะทำงานโดยที่ไม่ต้องรอกัน สามารถทำงานที่สองได้เลย โดยที่ไม่ต้องรอให้งานแรกเสร็จก่อน
- ถ้าเป็นแบบ Blocking คือ เมื่อคำสั่งแรกทำงาน ก็ต้องรอจนจบ task ถึงจะเริ่มทำคำสั่งที่สองได้

install node

การติดตั้ง Node.js ทำได้ง่ายแค่เข้าเว็บ Node.js จากนั้นเลือก OS ที่เราใช้งาน และทำการติดตั้งโดยแนะนำให้เลือกเวอร์ชั่นที่เป็น LTS (Long Term Support)

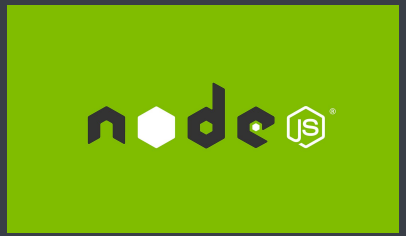
<u>คำสั่ง</u>

- node -v

  เพื่อตรวจสอบเวอร์ชั่นของ node
- npm -v

  เพื่อตรวจสอบเวอร์ชั่นของ npm
- npm init

  เพื่อสร้างไฟล์ package.json สำหรับแอปพลิเคชันของคุณ



https://nodejs.org/en/

package.json

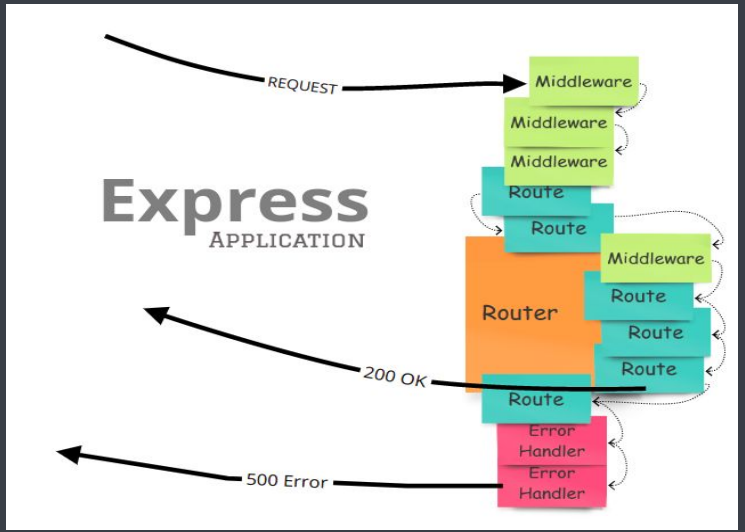
initial project

repository : git@gitlab.thinknet.co.th:bootcamp/expressjs.git

branch : EP0_initial_project

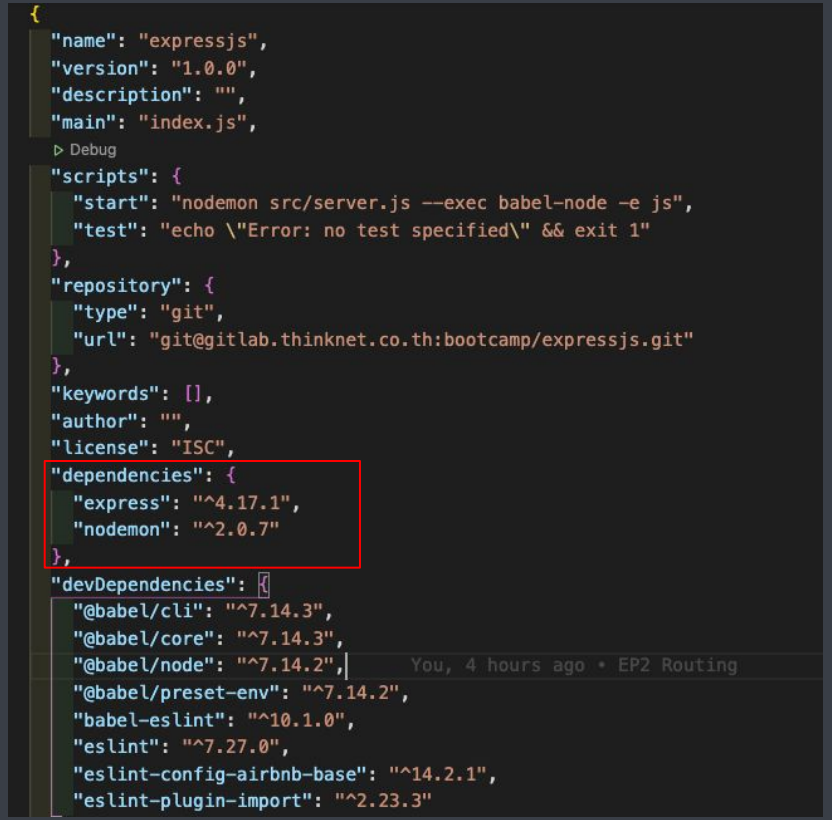
express

Express เป็น Web Framework ของ Node.js ที่ได้รับความนิยมตัวหนึ่ง ด้วยความที่มันใช้งานง่าย และมีความยืดหยุ่นสูง สามารถทำได้ทั้งเป็น API หรือนำมาเป็นเว็บ Server

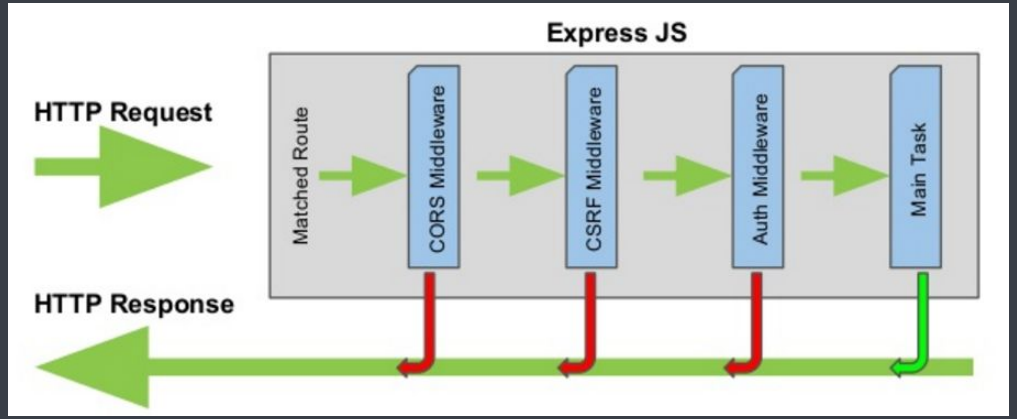


install express

- npm install express



structure


Express JS
HTTP Request
Matched Route
CORS Middleware
CSRF Middleware
Auth Middleware
Main Task
HTTP Response

start express server

Example 01

file src/server.js

```
import express from 'express'

const app = express()

const PORT = 3000

app.listen(PORT, (err) => {
  if (err) console.log('Error in server setup')
  console.log('Server listening on Port', PORT)
})
```

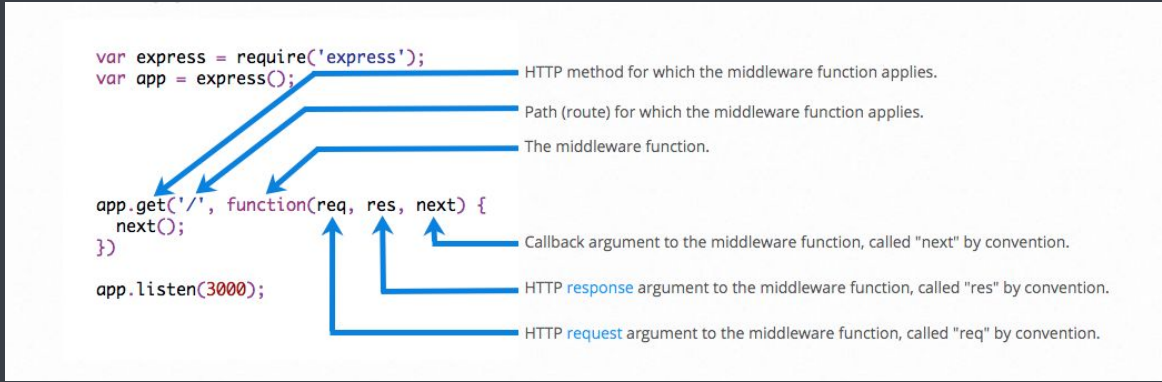
Routing

routing

คือ การกำหนด Path ที่ใช้ในการเรียกข้อมูลผ่าน HTTP request เช่น GET POST PUT หรือ DELETE แล้วให้ทำการ response ข้อมูล หรือ การทำงานใดๆ ให้สอดคล้องกับ URL ใดๆ ที่ระบุเข้ามา ซึ่งสามารถกำหนดได้ว่าจะให้ทำคำสั่งเดียวหรือหลายคำสั่ง

รูปแบบการกำหนด route

app.METHOD(PATH, HANDLER)



routing method

- GET
- POST
- PUT
- DELETE

โดยเมื่อมี HTTP request ขึ้น ก็จะทำให้เกิด request object และ response object โดย object ทั้งสอง เราสามารถนำเข้าไปใช้งานในฟังก์ชั่น เพื่อกำหนด หรือทำงานต่างๆ ต่อไปได้

routing paths

ตัวอย่างการกำหนด Routing Paths

/product จะสามารถเข้าถึงได้โดยผ่าน “localhost:3000/product”

/database.json จะสามารถเข้าถึงได้โดยผ่าน “localhost:3000/database.json”

ในตัวอย่างนี้จำลองโดยเข้าถึงผ่าน localhost:3000

** route path สามารถกำหนดรูปแบบ **Regular Expression** ได้ด้วย สามารถดูเพิ่มเติมได้ที่นี้

routing parameters

Route parameter หรือ เรียกอีกอย่างหนึ่งว่า URL segments เป็นส่วนที่ใช้สำหรับเก็บค่าไว้ในตำแหน่งต่างๆ ของ URL โดยค่าเหล่านี้ จะถูกเรียกใช้งานผ่าน req.params โดยชื่อของ parameter จะกำหนดเป็นชื่อ key

ตัวอย่างการกำหนด Routing Parameters

/product/:ID เรียกใช้งานด้วย /product/1

/product/:ID/color/:color เรียกใช้งานด้วย /product/1/color/bule

routing parameters

นอกจากการส่งค่าที่ URL segments ยังมีรูปแบบอื่นๆ ในการส่งข้อมูลได้เช่นกัน

- query string สามารถสังเกตุจาก URL เวลาใช้งานบน browser เช่น “https://www.google.com/search?q=ค้นหา” สามารถเรียกใช้งานด้วย req.query
- body สามารถเรียกใช้งานด้วย req.body
- header สามารถเรียกใช้งานด้วย req.header

routing handlers

Route handler หรือก็คือฟังก์ชั่นส่วนที่ทำงานเมื่อเข้าเงื่อนไข หรือ url ที่ผู้ใช้งานเรียกเข้ามา ตรงกับรูปแบบที่เรากำหนด เราสามารถกำหนดให้ ทำฟังก์ชั่นได้มากกว่า 1 ฟังก์ชั่น โดยใช้ middleware function เข้ามาช่วย โดยมีคำสั่ง next() เพื่อส่งไปทำงานฟังก์ชั่นที่เหลือ

ตัวอย่าง

```
router.get('/products', (req, res, next) => {
        const result = 1+2
        console.log(result)
})
```

request methods

ทำหน้าที่ดึงข้อมูลจาก HTTP Request ที่ส่งเข้ามาเพื่อใช้งาน

request method common

- req.body()
- req.cookies()
- req.params()
- req.query()

response methods

ทำหน้าที่ส่งกลับข้อมูลมายังผู้ใช้งาน และสิ้นสุดการทำงาน

response method common

- res.end()
- res.json()
- res.redirect()
- res.render()
- res.send()
- res.sendStatus()

** ซึ่งถ้าไม่มีการใช้งาน method ในส่วนนี้ การ request จากฝั่งผู้ใช้ อาจจะมีอาการค้างหรือหยุดการทำงานไป

best practices for REST API design

- รับส่งข้อมูลในรูปแบบ JSON
- ใช้คำนามแทนกริยาในการกำหนดชื่อ path
- มีการรับมือกับ errors ที่จะเกิดและส่งกลับมาเป็น error codes ให้ผู้ใช้เข้าใจ
- เปิดให้ใช้งาน filtering, sorting, and pagination
- มีการดูแลในเรื่องความปลอดภัยอยู่เสมอ
- ทำ Cache data เพื่อให้ performance ดียิ่งขึ้น
- ทำ Versioning our APIs

Best practices for REST API design

HTTP status codes

1xx: Informational - Request received, continuing process
2xx: Success - The action was successfully received, understood, and accepted
3xx: Redirection - Further action must be taken in order to complete the request
4xx: Client Error - The request contains bad syntax or cannot be fulfilled
5xx: Server Error - The server failed to fulfill an apparently valid request

HTTP Status Codes Common

200 : OK
201 : Created
400 : Bad Request
401 : Unauthorized
403 : Forbidden
404 : Not Found
500 : Internal Server Error
502 : Bad Gateway

https://www.ietf.org/assignments/http-status-codes/http-status-codes.xml

start routing

Example 02

file src/routes.js

```
import express from 'express'
import product from './resources/product.json'

const router = express.Router()

router
  .get('/products', (req, res, next) => res.status(200).json(product))
  .get('/products/:ID', (req, res) => {
    const { ID } = req.params
    const result = product.find((item) => item.id === ID)
    res.status(200).json(result)
  })

export default router
```