Express.js

Web Programming: Beginner

(Integrative-Generative AI Edition)

Student Price Book Center

Introduction to Express.js and Installation
Basic Routing Basic Middleware
Request and Response Management
Bibliography

คำนำ

ในยุคที่โลกอินเทอร์เน็ตขับเคลื่อนด้วยข้อมูลและการเชื่อมต่อแบบเรียลไทม์ การพัฒนาแอปพลิเคชันฝั่งเซิร์ฟเวอร์ที่มีประสิทธิภาพ รวดเร็ว และยืดหยุ่น จึงกลายเป็นปัจจัยสำคัญของการสร้างบริการออนไลน์ Express.js หนึ่งในเฟรมเวิร์กยอดนิยมของ Node.js ได้รับความสนใจจากนักพัฒนาทั่วโลกด้วยจุดเด่นด้านความเรียบง่าย ใช้งานไม่ซับซ้อน แต่สามารถนำไปประยุกต์ใช้ได้กับแอปพลิเคชันทั้งขนาดเล็กและใหญ่ได้อย่างทรงพลัง หนังสือเล่มนี้จึงถูกจัดทำขึ้นมาเพื่อเป็นแหล่งเรียนรู้ที่เป็นระบบ ครอบคลุมตั้งแต่พื้นฐาน จนสามารถต่อยอดสู่การพัฒนาเว็บเซอร์วิสแบบมืออาชีพได้อย่างมั่นใจ

Express.js Web Programming: Beginner เขียนขึ้นโดยมีวัตถุประสงค์หลักเพื่อปูพื้นฐานให้ผู้อ่านเข้าใจการทำงานของ Express.js ตั้งแต่ระดับเริ่มต้น โดยไม่จำเป็นต้องมีประสบการณ์มาก่อนในด้านการพัฒนา Node.js หรือการเขียน REST API มาก่อน โดยเนื้อหาภายในถูกเรียบเรียงอย่างเป็นลำดับขั้น ให้ผู้อ่านสามารถฝึกฝนได้ด้วยตนเองผ่านการเขียนโค้ดจริงในแต่ละบท พร้อมทั้งมีคำอธิบายที่ชัดเจน แยกหัวข้อย่อยอย่างเหมาะสมสำหรับผู้อ่านที่ต้องการเรียนรู้เฉพาะจุด

ในบทที่ 1 ผู้อ่านจะได้รับการแนะนำให้รู้จักกับ Express.js ตั้งแต่ความหมาย จุดประสงค์ของการใช้งาน รวมถึงประวัติความเป็นมาของเฟรมเวิร์กนี้ เพื่อสร้างความเข้าใจเบื้องต้น ก่อนเข้าสู่การติดตั้ง Node.js และ npm ซึ่งเป็นเครื่องมือสำคัญสำหรับการพัฒนา JavaScript ฝั่งเซิร์ฟเวอร์ จากนั้นจะมีการแนะนำการสร้างโปรเจกต์ Express.js แรก พร้อมอธิบายโครงสร้างพื้นฐานของโปรเจกต์และการเขียนเซิร์ฟเวอร์เบื้องต้นด้วย Express

ในบทที่ 2 จะเป็นการเจาะลึกเรื่องการ Routing ซึ่งเป็นหัวใจสำคัญของเว็บเซิร์ฟเวอร์ โดยอธิบายตั้งแต่รูปแบบ HTTP เมธอด (GET, POST, PUT, DELETE) การใช้ Route Parameters และ Query Parameters การติดตั้ง curl เพื่อทดสอบ API การจัดการคำตอบที่ส่งกลับ (Response) และเทคนิคการจัดการเส้นทางแบบละเอียด โดยมีตัวอย่างที่ช่วยให้เข้าใจและสามารถทดลองใช้งานได้จริง

บทที่ 3 พาผู้อ่านเข้าสู่โลกของ **Middleware** ซึ่งเป็นองค์ประกอบสำคัญที่อยู่เบื้องหลังความสามารถในการควบคุมการไหลของคำขอและคำตอบใน Express.js โดยอธิบายแนวคิดของ Middleware การใช้งานแบบง่าย รวมถึง Middleware พื้นฐานอย่าง Logger และ Static File Handler ที่ใช้ให้บริการ HTML, CSS และรูปภาพ ผู้อ่านจะได้เห็นภาพการประยุกต์ Middleware เพื่อจัดการการไหลของข้อมูลภายในระบบให้เป็นระบบระเบียบ

สำหรับบทที่ 4 ผู้อ่านจะได้เรียนรู้การจัดการกับ Request และ Response อย่างละเอียด โดยเน้นการดึงข้อมูลจากคำขอ (Request) ในรูปแบบต่าง ๆ ได้แก่ Body, Query, และ Params พร้อมทั้งการสร้างคำตอบที่เหมาะสมผ่านเมธอดของ Express เช่น res.send(), res.json(), และ res.status() โดยอธิบายความแตกต่างและสถานการณ์ที่ควรใช้อย่างชัดเจน เพื่อให้สามารถสร้าง API ที่ถูกต้องตามมาตรฐาน และใช้งานร่วมกับ Frontend หรือ Client ภายนอกได้อย่างราบรื่น

เนื้อหาทั้งหมดในเล่มนี้ไม่เพียงแต่มุ่งเน้นที่ความรู้เชิงเทคนิคเท่านั้น แต่ยังมุ่งหวังให้ผู้อ่านเข้าใจบริบทในการนำ Express.js ไปใช้จริง ทั้งในด้านการออกแบบระบบ การทดสอบ และการบำรุงรักษาใน

ระยะยาว โดยจัดวางเนื้อหาให้ง่ายต่อการเรียนรู้ และเหมาะสำหรับทั้งนักศึกษา ผู้เริ่มต้นเขียนโค้ด และนักพัฒนาที่ต้องการเสริมความเข้าใจด้านเซิร์ฟเวอร์ด้วย JavaScript

หวังเป็นอย่างยิ่งว่าหนังสือ *Express.js Web Programming: Beginner* เล่มนี้ จะช่วยเสริมสร้างความรู้ พัฒนาทักษะ และเป็นแหล่งอ้างอิงสำคัญสำหรับผู้อ่านทุกคนที่ต้องการเริ่มต้นอย่างมั่นคงกับ Express.js และต่อยอดสู่การสร้างแอปพลิเคชันที่แข็งแกร่งในอนาคต.

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1 แนะนำ Express.js และการติดตั้ง (Introduction to Express.js and Installation)..... 1

- แนะนำ Express.js และการติดตั้ง
- Express.js คืออะไร?
- ประวัติการพัฒนา Express.js
- การติดตั้ง Node.js และ npm (Node Package Manager)
- สร้างโปรเจกต์ Express.js แรก
- โครงสร้างโปรเจกต์ Express.js พื้นฐาน
- สร้าง Server ตัวแรกด้วย Express.js

บทที่ 2 Routing พื้นฐาน (Basic Routing).....25

- Routing พื้นฐานใน Express.js
- Routing พื้นฐานใน Express.js เชิงลึก
- การกำหนดเส้นทาง HTTP (GET, POST, PUT, DELETE) ใน Express.js
- การติดตั้ง curl แบบละเอียด
- คำสั่งติดตั้ง Dependencies
- การใช้งาน Route Parameters และ Query Parameters ใน Express.js
- การตอบกลับ (Response) แบบพื้นฐานใน Express.js

บทที่ 3 Middleware เบื้องต้น (Basic Middleware).....78

- Middleware เบื้องต้นใน Express.js
- รายละเอียดเชิงลึก: Middleware ใน Express.js
- Middleware คืออะไร? (ในบริบทของ Express.js)
- การใช้ Middleware ใน Express
- การจัดการ Static Files (HTML, CSS, รูปภาพ) ใน Express.js
- โปรแกรมบูรณาการ Express.js

บทที่ 4 การจัดการ Request และ Response (Request and Response Management)...125

- การจัดการ Request และ Response
- การจัดการ Request และ Response (เชิงลึก)

- การอ่านข้อมูลจาก Request (Body, Query, Params) ใน Express.js
- การส่ง Response (status codes, headers, JSON)
- การใช้งาน res.send(), res.json(), และ res.status() ใน Express.js

บรรณานุกรม 168

บทที่ 1

แนะนำ Express.js และการติดตั้ง

(Introduction to Express.js and Installation)

เนื้อหา

- แนะนำ Express.js และการติดตั้ง
- Express.js คืออะไร?
- ประวัติการพัฒนา Express.js
- การติดตั้ง Node.js และ npm (Node Package Manager)
- สร้างโปรเจกต์ Express.js แรก
- โครงสร้างโปรเจกต์ Express.js พื้นฐาน
- สร้าง Server ตัวแรกด้วย Express.js

บทนำบทที่ 1: แนะนำ Express.js และการติดตั้ง

ในยุคที่แอปพลิเคชันเว็บกลายเป็นหัวใจสำคัญของธุรกิจและการให้บริการออนไลน์ การพัฒนาเซิร์ฟเวอร์ฝั่งแบ็กเอนด์อย่างมีประสิทธิภาพจึงเป็นสิ่งจำเป็น Express.js จึงถือกำเนิดขึ้นในฐานะเฟรมเวิร์กที่เรียบง่ายแต่ทรงพลังสำหรับ Node.js ที่ช่วยให้นักพัฒนาสามารถสร้างเว็บเซิร์ฟเวอร์และ RESTful API ได้อย่างรวดเร็ว โดยไม่ต้องเขียนโค้ดที่ซับซ้อนหรือจัดการกับรายละเอียดระดับต่ำมากเกินไป

Express.js คือเฟรมเวิร์กแบบ Minimal และ Flexible สำหรับ Node.js ที่ออกแบบมาเพื่อช่วยให้การพัฒนาเว็บแอปพลิเคชันและ API เป็นเรื่องง่ายขึ้น ด้วยรูปแบบการใช้งานที่ชัดเจนและมีระบบ middleware ที่ยืดหยุ่น ทำให้ Express เป็นที่นิยมในหมู่นักพัฒนา JavaScript ทั่วโลก และกลายเป็นแกนหลักของเทคโนโลยีแบ็กเอนด์ในหลายโปรเจกต์ขนาดเล็กไปจนถึงระดับองค์กร

ก่อนที่จะเริ่มต้นใช้งาน Express.js จำเป็นต้องติดตั้ง Node.js และ npm (Node Package Manager) ให้พร้อม เนื่องจาก Express ทำงานอยู่บนแพลตฟอร์ม Node.js ซึ่งเป็นรันไทม์ที่ช่วยให้สามารถเขียน JavaScript บนฝั่งเซิร์ฟเวอร์ได้ การติดตั้ง Node.js และ npm เป็นขั้นตอนแรกที่สำคัญ ซึ่งจะเปิดประตูไปสู่การจัดการแพ็กเกจและการพัฒนาเว็บแอปพลิเคชันอย่างเต็มรูปแบบ

เมื่อเครื่องมือพื้นฐานพร้อมแล้ว ผู้อ่านจะได้เริ่มต้นสร้างโปรเจกต์ Express.js แรกด้วยตนเอง โดยสามารถเลือกใช้คำสั่ง npm init สำหรับสร้างโครงการด้วยตนเองทีละขั้นตอน หรือใช้ express-generator สำหรับสร้างโครงสร้างโปรเจกต์อย่างรวดเร็ว ซึ่งจะช่วยลดเวลาการเตรียมโครงสร้างและเพิ่มเวลาในการพัฒนาโค้ดที่สำคัญแทน

ภายหลังจากสร้างโปรเจกต์แล้ว ผู้อ่านจะได้เรียนรู้เกี่ยวกับโครงสร้างของโปรเจกต์ Express เบื้องต้น เช่น ไฟล์ app.js หรือ index.js ที่เป็นจุดเริ่มต้นของแอปพลิเคชัน โฟลเดอร์สำหรับ route, view, public และการแยกโค้ดให้เป็นระบบ ซึ่งเป็นแนวทางปฏิบัติที่ดีสำหรับการพัฒนาโค้ดที่สามารถดูแลและขยายได้ในระยะยาว

จากนั้นผู้อ่านจะได้ลงมือเขียน Server ตัวแรกด้วย Express ด้วยการกำหนดพอร์ต เซต route แรก และลองรันแอปพลิเคชันในเบราว์เซอร์ ซึ่งจะทำให้เข้าใจกลไกการทำงานของ Express อย่างเป็นรูปธรรม ทั้งการรับคำขอ (request) และการส่งกลับข้อมูล (response) สู่ผู้ใช้

บทแรกนี้จึงเป็นจุดเริ่มต้นที่สำคัญของการทำความเข้าใจ Express.js ตั้งแต่พื้นฐาน โดยให้ทั้งแนวคิด ภาพรวม และลงมือปฏิบัติจริง ผู้อ่านจะสามารถติดตั้งเครื่องมือที่จำเป็น สร้างเซิร์ฟเวอร์เบื้องต้น และเตรียมความพร้อมสำหรับการเรียนรู้ฟีเจอร์ขั้นสูงในบทถัด ๆ ไปอย่างมั่นใจ.

แนะนำ Express.js และการติดตั้ง

---

1. Express.js คืออะไร? ทำไมต้องใช้?

Express.js คือเว็บเฟรมเวิร์กของ Node.js ที่ช่วยให้การสร้างเว็บเซิร์ฟเวอร์หรือ API ง่ายและเร็วขึ้นมาก โดย Express.js จะช่วยจัดการเรื่อง routing, middleware, request/response handling ให้เราเรียบง่ายและยืดหยุ่น

ทำไมต้องใช้ Express.js?

- **เบาและเร็ว** — ไม่ได้บังคับโครงสร้างหรือฟีเจอร์มากเกินไป
- **ยืดหยุ่น** — เราสามารถเลือกใช้ middleware หรือฟีเจอร์เสริมต่าง ๆ ได้ตามต้องการ
- **ชุมชนขนาดใหญ่** — มีแพ็กเกจและเครื่องมือเสริมมากมาย
- **เหมาะกับการสร้าง RESTful API และเว็บแอป**
- **ทำงานร่วมกับ Node.js ได้ดี** — ใช้ JavaScript ทั้งฝั่งเซิร์ฟเวอร์และไคลเอนต์

---

2. การติดตั้ง Node.js และ npm

Express.js รันบน Node.js ดังนั้นคุณต้องติดตั้ง Node.js ก่อน

วิธีติดตั้ง Node.js

1. เข้าเว็บ https://nodejs.org
2. ดาวน์โหลดตัวติดตั้ง (LTS version แนะนำสำหรับผู้เริ่มต้น)
3. ติดตั้งตามขั้นตอน (ติดตั้ง npm มาให้ด้วยอัตโนมัติ)

ตรวจสอบการติดตั้ง

เปิด Terminal หรือ Command Prompt แล้วพิมพ์คำสั่ง

```
node -v
```

ถ้าแสดงเวอร์ชัน เช่น v18.15.0 แปลว่าติดตั้งสำเร็จ

npm -v

แสดงเวอร์ชัน npm (ตัวจัดการแพ็กเกจของ Node.js)

---

3. สร้างโปรเจกต์ Express.js แรก

ขั้นตอนสร้างโปรเจกต์

1. สร้างโฟลเดอร์โปรเจกต์ใหม่
2. mkdir my-express-app
3. cd my-express-app
4. สั่งสร้างไฟล์ package.json เพื่อจัดการ dependencies
5. npm init -y
   จะได้ไฟล์ package.json แบบพื้นฐานที่บอกชื่อโปรเจกต์และรายละเอียดอื่น ๆ
6. ติดตั้ง Express
7. npm install express

---

4. โครงสร้างโปรเจกต์ Express พื้นฐาน

หลังติดตั้งและสร้างโปรเจกต์แล้ว โครงสร้างจะประมาณนี้

my-express-app/

```
├── node_modules/       # โฟลเดอร์เก็บแพ็กเกจ npm ที่ติดตั้ง
├── package.json        # ไฟล์กำหนดรายละเอียดโปรเจกต์และ dependencies
├── package-lock.json   # ไฟล์ล็อคเวอร์ชันของแพ็กเกจ
└── index.js            # ไฟล์เริ่มต้นเขียนโค้ด Express Server
```

- node_modules จะถูกสร้างเมื่อเรารัน npm install
- index.js คือไฟล์ที่เราจะเขียนโค้ดเซิร์ฟเวอร์ (ชื่อไฟล์ตั้งเองได้)
- package.json เป็น metadata และ dependency list ของโปรเจกต์

---

5. สร้าง Server ตัวแรกด้วย Express

เปิดไฟล์ index.js แล้วเขียนโค้ดดังนี้

```
// นำเข้า Express module
const express = require('express');

// สร้างแอป Express
const app = express();
```

```
// กำหนดพอร์ตที่เซิร์ฟเวอร์จะฟัง (ใช้ 3000 เป็นตัวอย่าง)
const PORT = 3000;

// กำหนด route พื้นฐาน (root route)
app.get('/', (req, res) => {
  res.send('Hello, Express!');
});

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

อธิบายโค้ด

- require('express') คือการนำ Express เข้ามาใช้งาน
- express() สร้างตัวแอป Express
- app.get('/', ...) กำหนด route HTTP GET ที่เส้นทาง /
- res.send() ส่งข้อความตอบกลับ client
- app.listen(PORT, ...) สั่งให้แอปเริ่มฟังพอร์ต 3000 และแสดงข้อความเมื่อพร้อม

วิธีรัน Server

ใน Terminal ให้รันคำสั่ง

node index.js

ถ้าทุกอย่างถูกต้อง จะเห็นข้อความ

Server is running at http://localhost:3000

เปิดเว็บเบราว์เซอร์แล้วไปที่ URL: http://localhost:3000

จะเห็นข้อความ

Hello, Express!

แปลว่าเราได้สร้าง Express Server ตัวแรกสำเร็จแล้ว!

แนะนำ Express.js และการติดตั้ง (เชิงลึก)

---

1. Express.js คืออะไร? ทำไมต้องใช้?

พื้นฐานของ Express.js

- Express.js คือ **Web Framework** ที่สร้างบน **Node.js** ซึ่งช่วยจัดการการรับ-ส่ง HTTP requests, การกำหนด routing, middleware และการตอบสนอง (response) ให้เป็นเรื่องง่าย
- Node.js เป็น environment สำหรับรัน JavaScript ฝั่ง server แบบ event-driven และ non-blocking I/O ซึ่งทำให้ Express.js สามารถรองรับการทำงานพร้อมกันได้จำนวนมากโดยประสิทธิภาพสูง

เหตุผลที่ Express.js โดดเด่น

- **Minimalist & Unopinionated:** Express ไม่บังคับโครงสร้างหรือ pattern ใด ๆ เราจึงเลือกวิธีการจัดการแอปเองได้ตามต้องการ
- **Middleware-based architecture:** ฟีเจอร์ส่วนใหญ่ของ Express เกิดจาก middleware ที่เราเลือกเพิ่มเข้าไป ซึ่งทำให้ปรับแต่งได้ยืดหยุ่น
- **Routing system:** จัดการเส้นทาง URL ได้ง่ายและซับซ้อนในระดับสูง
- **Extensive ecosystem:** มี middleware และ plugins จากชุมชนมากมาย เช่น body-parser, cookie-parser, multer, passport
- **JavaScript Full Stack:** ใช้ภาษาเดียวกันทั้ง client และ server ทำให้ทีมพัฒนาสามารถประสานงานง่าย

การเปรียบเทียบกับ Framework อื่น

- เมื่อเทียบกับเฟรมเวิร์กแบบ monolithic เช่น Ruby on Rails หรือ Django, Express จะเบาและยืดหยุ่นกว่า แต่ต้องเขียนโค้ดจัดการเองมากกว่า
- เหมาะกับ microservices และ API backend ที่ต้องการความรวดเร็วและควบคุมได้สูง

---

2. การติดตั้ง Node.js และ npm (เชิงลึก)

Node.js คืออะไร?

- Node.js คือ runtime ที่รัน JavaScript บน server โดยใช้ V8 engine ของ Chrome
- เน้นการประมวลผลแบบ non-blocking event-driven ซึ่งเหมาะกับ I/O-intensive application เช่นเว็บเซิร์ฟเวอร์

npm คืออะไร?

- npm ย่อมาจาก Node Package Manager
- เป็นเครื่องมือสำหรับจัดการ package (library/module) ของ Node.js
- ใช้ติดตั้ง, อัปเดต, ลบ dependencies ในโปรเจกต์

แนะนำการติดตั้ง

- ควรเลือกเวอร์ชัน LTS (Long Term Support) สำหรับความเสถียร
- ติดตั้งเสร็จแล้ว แนะนำให้ทดสอบ node -v และ npm -v ว่าระบบรับรู้คำสั่งเหล่านี้

การตั้งค่า npm

- บางครั้งต้องตั้ง proxy หรือ registry ตามเครือข่ายที่ใช้งาน

- คำสั่งพื้นฐานเช่น npm install <package>, npm uninstall <package>, npm update

---

3. สร้างโปรเจกต์ Express.js แรก (เชิงลึก)

npm init ทำอะไร?

- คำสั่ง npm init จะสร้างไฟล์ package.json ซึ่งเป็น metadata ของโปรเจกต์
- ไฟล์นี้เก็บข้อมูลชื่อโปรเจกต์, เวอร์ชัน, รายการ dependencies, สคริปต์ที่รันได้
- ใช้ -y เพื่อข้ามการตอบคำถาม และสร้างไฟล์ด้วยค่าเริ่มต้น

การติดตั้ง Express

- npm install express จะเพิ่ม express ลงใน node_modules
- บันทึกเวอร์ชันลงใน package.json โดยอัตโนมัติ (under dependencies)
- package-lock.json จะล็อคเวอร์ชัน dependencies ให้แน่นอน

ทำไมต้องใช้ package.json และ node_modules

- ช่วยให้จัดการ dependencies ได้เป็นระบบ
- ทีมพัฒนาสามารถแชร์โค้ดและติดตั้ง dependencies เดียวกันได้ง่ายแค่รัน npm install

---

4. โครงสร้างโปรเจกต์ Express พื้นฐาน (เชิงลึก)

ไฟล์สำคัญและโฟลเดอร์

- node_modules/ — โฟลเดอร์ที่เก็บแพ็กเกจทั้งหมดที่ติดตั้ง (ไม่ควรใส่ใน Git)
- package.json — เป็นหัวใจของโปรเจกต์ที่ระบุข้อมูลและ dependencies
- package-lock.json — ล็อคเวอร์ชันของ dependencies ย่อยทั้งหมด เพื่อความแม่นยำในการติดตั้งซ้ำ
- index.js — จุดเริ่มต้นของโปรเจกต์ (ตั้งชื่อเองได้)

คำแนะนำการจัดโครงสร้างโปรเจกต์

- สำหรับโปรเจกต์เล็กใช้ไฟล์เดียวก็ได้ แต่เมื่อโปรเจกต์โตควรแยกโฟลเดอร์ เช่น
    - /routes สำหรับ route handlers
    - /controllers สำหรับ logic
    - /models สำหรับ data models
    - /middlewares สำหรับ middleware

---

5. สร้าง Server ตัวแรกด้วย Express (เชิงลึก)

วิธีการทำงานของโค้ด

```
const express = require('express');
const app = express();
const PORT = 3000;
```

- require('express') โหลดโมดูล express จาก node_modules
- express() สร้าง instance ของแอปพลิเคชัน Express ซึ่งเป็นฟังก์ชันหลักสำหรับจัดการ request และ response

```
app.get('/', (req, res) => {
  res.send('Hello, Express!');
});
```

- app.get() กำหนด HTTP method GET และเส้นทาง /
- เมื่อผู้ใช้เข้าหน้าเว็บ root (http://localhost:3000/) ฟังก์ชัน callback จะถูกเรียก
- req คือ Request object ที่เก็บข้อมูล request เช่น headers, params, body
- res คือ Response object ที่ใช้ส่งข้อมูลกลับ client
- res.send() ส่งข้อความ plain text กลับไป

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

- app.listen() เปิดพอร์ตให้เซิร์ฟเวอร์รับคำขอได้
- callback ที่ส่งเข้าไปเรียกเมื่อเซิร์ฟเวอร์เริ่มทำงาน

เพิ่มเติม

- Express ใช้ระบบ **Event-driven** และ **Asynchronous**
- เมื่อมี HTTP request เข้ามา Express จะจับ route ที่ตรงและเรียก middleware/handler ตามลำดับ
- res.send() จะส่ง response พร้อม header และปิด connection

Bonus: สิ่งที่ควรรู้ในขั้นแรก

- **พอร์ต 3000** เป็นพอร์ตมาตรฐานสำหรับการพัฒนา แต่สามารถเปลี่ยนได้ตามต้องการ
- การใช้ nodemon (ติดตั้งด้วย npm install -g nodemon) จะช่วย restart server อัตโนมัติเมื่อโค้ดเปลี่ยน
- Express รองรับ HTTP method หลายแบบ เช่น app.post(), app.put(), app.delete()
- Express รองรับการกำหนดหลาย route และ chain middleware ได้

Express.js คืออะไร?

Express.js คือเว็บเฟรมเวิร์ก (Web Framework) สำหรับ Node.js ที่ช่วยให้นักพัฒนาสร้างเว็บเซิร์ฟเวอร์และ API ได้อย่างรวดเร็วและง่ายดาย
มันเป็นไลบรารีที่ให้ฟีเจอร์พื้นฐานในการจัดการ HTTP request/response, routing, middleware และการจัดการโครงสร้างแอปที่ยืดหยุ่น

จุดเด่นของ Express.js

- **Minimal & Unopinionated**

  Express ออกแบบให้มีขนาดเล็กและไม่บังคับวิธีการพัฒนา (unopinionated) ทำให้คุณเลือกออกแบบสถาปัตยกรรมของแอปเองได้
- **Middleware-based architecture**

  ทุกอย่างใน Express คือ “Middleware” ซึ่งเป็นฟังก์ชันที่ประมวลผล HTTP request/response โดยเรียงลำดับกันไป (pipeline)
- **Routing system**

  Express มีระบบ routing ที่ง่ายและยืดหยุ่น ช่วยให้กำหนดเส้นทาง URL เพื่อจัดการคำขอต่าง ๆ ได้สะดวก
- **Extensive ecosystem**

  มีแพ็กเกจเสริม (middleware) มากมายจากชุมชน เช่น body-parser, cookie-parser, multer, passport
- **ใช้ภาษา JavaScript ทั้งฝั่ง Server และ Client**

  ทำให้ทีมพัฒนาสามารถใช้ภาษาเดียวกัน ลดความซับซ้อนในการพัฒนาและเรียนรู้

ทำไมต้องใช้ Express.js?

1. ลดความซับซ้อนของ Node.js HTTP Module

- Node.js มีโมดูล http ให้ใช้งานโดยตรง แต่ค่อนข้างต่ำระดับ (low-level) ต้องเขียนโค้ดจัดการ routing, parsing request, ส่ง response ด้วยตัวเองทั้งหมด
- Express ช่วยห่อหุ้มและเพิ่ม API ที่ใช้งานง่ายกว่า เช่น app.get(), app.post() เพื่อจัดการ route ได้รวดเร็ว

2. ยืดหยุ่นและขยายได้ง่าย

- Express ให้เราสามารถใส่ middleware เพื่อประมวลผล request/response ก่อนส่งถึง handler ได้ เช่น ตรวจสอบสิทธิ์, logging, แปลงข้อมูล
- คุณสามารถเลือกเพิ่มฟีเจอร์ที่ต้องการเท่านั้น ทำให้แอปไม่หนักและง่ายต่อการดูแลรักษา

3. รวดเร็วและประสิทธิภาพดี

- Express รันบน Node.js ซึ่งใช้ event-driven, non-blocking I/O ทำให้สามารถจัดการคำขอจำนวนมากพร้อมกันได้ดี
- เหมาะกับการสร้าง RESTful API, เว็บแอปแบบ real-time, หรือ microservices

4. ชุมชนและเอกสารดี

- Express เป็นหนึ่งในเว็บเฟรมเวิร์กยอดนิยมสำหรับ Node.js มีเอกสารดีและชุมชนใหญ่
- มีตัวอย่าง, บทความ, และ middleware เสริมมากมายให้เลือกใช้ ช่วยลดเวลาพัฒนา

5. สร้างสถาปัตยกรรมโมเดิร์นได้ง่าย

- รองรับการทำงานแบบ MVC, microservices, และ Server-Side Rendering
- ใช้งานร่วมกับ template engine เช่น Pug, EJS ได้อย่างราบรื่น

สรุปสั้น ๆ

ข้อดีของ Express.js	คำอธิบาย
ง่ายต่อการเริ่มต้น	API เข้าใจง่าย ใช้โค้ดไม่เยอะ
ยืดหยุ่นสูง	เลือกใช้ middleware และโครงสร้างเองได้
ประสิทธิภาพสูง	ใช้ event-driven architecture ของ Node.js
ชุมชนขนาดใหญ่	มี middleware และเครื่องมือมากมาย
ใช้ JavaScript ทั้งระบบ	ทำให้พัฒนาและบำรุงรักษาง่าย

ประวัติการพัฒนา Express.js

เริ่มต้นและการกำเนิด (2010-2011)

- **ปี 2010**: Express.js ถูกพัฒนาโดย **TJ Holowaychuk** นักพัฒนาชื่อดังในวงการ Node.js
- จุดประสงค์คือสร้างเว็บเฟรมเวิร์กที่มีขนาดเล็ก, ยืดหยุ่น, และใช้งานง่ายสำหรับ Node.js
- เปิดตัวครั้งแรกบน GitHub ในปี 2010 และได้รับความสนใจอย่างรวดเร็วในวงการ Node.js
- Express.js เป็นส่วนหนึ่งของ **"Connect"** ซึ่งเป็น middleware framework สำหรับ Node.js

การเติบโตและการพัฒนา (2012-2015)

- Express.js ได้กลายเป็นเว็บเฟรมเวิร์กยอดนิยมอันดับต้น ๆ ของ Node.js
- เริ่มมีการเพิ่มฟีเจอร์ routing, middleware management, และ error handling ที่สมบูรณ์มากขึ้น
- มีการนำไปใช้ในโปรเจกต์โอเพ่นซอร์สและแอปจริงหลายตัว
- ชุมชนผู้ใช้งานและผู้พัฒนาเติบโตขึ้นอย่างรวดเร็ว

การเปลี่ยนมือสู่ Node.js Foundation / OpenJS Foundation (2015-ปัจจุบัน)

- ปี 2015: Express.js ได้กลายเป็นส่วนหนึ่งของ **Node.js Foundation** เพื่อการดูแลรักษาและพัฒนาอย่างต่อเนื่อง
- ปัจจุบัน Express.js อยู่ภายใต้การดูแลของ **OpenJS Foundation** ซึ่งดูแลโปรเจกต์โอเพ่นซอร์สของ JavaScript หลายตัว
- มีการปรับปรุงเวอร์ชันอย่างต่อเนื่อง เช่น Express 4.x ที่เป็นเวอร์ชันหลัก และ Express 5.x ที่กำลังอยู่ในขั้นพัฒนา (alpha/beta)
- เน้นการปรับปรุงเรื่อง Promise support, async/await, และ modularity

---

แนวโน้มในอนาคตของ Express.js

- **รองรับ ES6+ และ async/await**: ปรับให้เข้ากับมาตรฐาน JavaScript ใหม่ ๆ ทำให้เขียนโค้ดง่ายและชัดเจนขึ้น
- **โมดูลาร์มากขึ้น**: Express จะยืดหยุ่นและแยกส่วนได้มากขึ้น เพื่อรองรับ microservices และ serverless architectures
- **การปรับปรุงประสิทธิภาพและความปลอดภัย**: เน้นการทำงานที่รวดเร็วและปลอดภัยตามมาตรฐานสมัยใหม่
- **บูรณาการกับเทคโนโลยีใหม่**: เช่น GraphQL, WebSockets, และ Cloud-native services
- **ชุมชนและ ecosystem เติบโตอย่างต่อเนื่อง**: มีเครื่องมือและ middleware ใหม่ ๆ เกิดขึ้นเสมอ

องค์กรที่ใช้งาน Express.js

Express.js เป็นเฟรมเวิร์กยอดนิยมที่ถูกใช้โดยองค์กรและบริษัทชั้นนำมากมายทั่วโลก เช่น

องค์กร / บริษัท	การใช้งาน
IBM	ใช้ในการพัฒนาแอปพลิเคชันและ API หลายตัว
Uber	ใช้สำหรับบางส่วนของ backend API
Accenture	พัฒนาโซลูชันและระบบสำหรับลูกค้าด้วย Express
PayPal	ใช้สำหรับระบบ backend บางส่วน
Netflix	ใช้ Express ในการจัดการ API บางตัว
LinkedIn	ใช้ Node.js และ Express ในระบบ backend บางส่วน
Walmart	ใช้สำหรับระบบ backend และ API
Mozilla	ใช้สำหรับบริการบางตัวและ API

นอกจากนี้ ยังมีสตาร์ทอัพและบริษัทเทคโนโลยีขนาดกลางและเล็กอีกมากมายที่เลือกใช้ Express.js เพราะความง่ายและประสิทธิภาพ

การติดตั้ง Node.js และ npm (Node Package Manager)

---

1. Node.js คืออะไร?

- Node.js คือ **JavaScript runtime** ที่ใช้รัน JavaScript บนฝั่ง Server
- พัฒนาด้วย V8 JavaScript engine ของ Google Chrome

- ช่วยให้คุณสร้างแอปพลิเคชัน server-side ที่มีประสิทธิภาพสูง รองรับการประมวลผลแบบ asynchronous และ event-driven

---

2. npm คืออะไร?

- npm ย่อมาจาก **Node Package Manager**
- เป็นเครื่องมือจัดการแพ็กเกจ (package manager) ของ Node.js
- ใช้ติดตั้ง, อัปเดต, ลบ แพ็กเกจหรือไลบรารีต่าง ๆ ที่ใช้ในโปรเจกต์
- npm จะถูกติดตั้งมาอัตโนมัติเมื่อคุณติดตั้ง Node.js

---

3. วิธีติดตั้ง Node.js และ npm

ขั้นตอนการติดตั้ง

A. สำหรับ Windows และ macOS

1. ไปที่เว็บไซต์ https://nodejs.org/
2. ดาวน์โหลดตัวติดตั้ง Node.js
    - แนะนำดาวน์โหลด **LTS (Long Term Support)** เพราะเสถียรและเหมาะกับการใช้งานในโปรดักชัน
    - มีเวอร์ชันสำหรับ Windows, macOS, Linux ให้เลือก
3. รันไฟล์ติดตั้งและทำตามขั้นตอน
    - เลือก “Next” ไปเรื่อย ๆ
    - ติ๊กเลือกให้ติดตั้ง npm และตั้งค่า PATH ให้เรียบร้อย
4. ติดตั้งเสร็จแล้วเปิด Command Prompt (Windows) หรือ Terminal (macOS/Linux)

B. สำหรับ Linux

- ใช้คำสั่งติดตั้งผ่าน package manager ของระบบ เช่น apt, yum หรือใช้ Node Version Manager (nvm) ซึ่งแนะนำให้ใช้มากกว่า

---

4. ตรวจสอบการติดตั้ง

เปิด Terminal หรือ Command Prompt แล้วพิมพ์คำสั่งตรวจสอบเวอร์ชัน:

node -v

ถ้าติดตั้งสำเร็จ จะเห็นเวอร์ชัน Node.js เช่น v18.15.0

npm -v

จะแสดงเวอร์ชัน npm เช่น 9.5.1

---

5. แนะนำการใช้งาน npm เบื้องต้น

- สร้างโปรเจกต์ใหม่และสร้างไฟล์ package.json

npm init

ระบบจะถามข้อมูลโปรเจกต์ เช่นชื่อ, เวอร์ชัน, entry file ฯลฯ

ถ้าต้องการใช้ค่าเริ่มต้นทั้งหมดให้รัน

npm init -y

- ติดตั้งแพ็กเกจ เช่น Express

npm install express

คำสั่งนี้จะติดตั้ง Express และบันทึกเวอร์ชันลงใน package.json อัตโนมัติ

---

6. เคล็ดลับและข้อควรระวัง

- ควรใช้ **LTS Version** ของ Node.js ในงาน production เพราะเสถียรกว่าเวอร์ชัน Current
- การติดตั้งแบบ global (เช่น npm install -g nodemon) จะทำให้เรียกใช้งานได้ทั่วเครื่อง
- แนะนำใช้ **Node Version Manager (nvm)** สำหรับจัดการเวอร์ชัน Node.js หลายเวอร์ชันในเครื่องเดียวกัน (เหมาะกับนักพัฒนาที่ต้องสลับโปรเจกต์)
- อย่าใส่โฟลเดอร์ node_modules ลงระบบ version control (Git) ให้เพิ่มใน .gitignore เสมอ

สร้างโปรเจกต์ Express.js แรก

---

1. สร้างโฟลเดอร์โปรเจกต์ใหม่

เปิด Terminal หรือ Command Prompt แล้วรันคำสั่ง:

mkdir my-express-app

cd my-express-app

คำอธิบาย

- mkdir สร้างโฟลเดอร์ใหม่ชื่อ my-express-app
- cd เข้าไปยังโฟลเดอร์นั้น

---

2. สร้างไฟล์ package.json

เพื่อจัดการ metadata และ dependencies ของโปรเจกต์ ให้รันคำสั่ง:

npm init -y

คำอธิบาย

- คำสั่งนี้จะสร้างไฟล์ package.json ด้วยค่าตั้งต้น (default)
- ไฟล์นี้จะบอกชื่อโปรเจกต์, เวอร์ชัน, entry point (เช่น index.js), และบันทึกรายการแพ็กเกจที่ติดตั้งในอนาคต

---

3. ติดตั้ง Express