**การสร้าง CRUD Operations** **ด้วย Node.js**

สรุปสั้น ๆ ก่อนเริ่มอ่าน

สำหรับใครที่ได้เริ่มหัดเขียน Back-end สำหรับ Web application อยู่ แต่ยังติดปัญหาไม่แน่ใจว่าควรจะเริ่มจากตรงไหน อะไรยังไงดี ในบทความนี้ ผมจะมาเสนอวิธีสร้าง Basic Operations สำหรับการเริ่มต้นทำงานพื้นฐานต่าง ๆ ของฝั่ง Back-end ด้วย Node.js ทั้ง Create, Read, Update และ Delete หรือเรียกย่อ ๆ ว่า CRUD พร้อมตัวอย่างการใช้งานจริงของทั้ง 4 Operations นี้ หากใครที่ยังไม่แน่ใจว่าจะเริ่มต้นยังไงดี บทความนี้ ก็จะเป็นอีกหนึ่งตัวช่วย ทีทำให้สามารถเริ่มสร้าง Back-end Operations ในระดับต้นได้อย่างแน่นอน !!

ติดตั้งสิ่งที่จำเป็น

อย่างแรกที่เราจำเป็นต้องติดตั้งเลย นั่นก็คือ Node.js สามารถติดตั้งได้ที่ นี่ เลย

จากนั้น ให้เราสร้าง Folder เปล่า ๆ มาชิ้นนึง และมาเริ่มสร้าง Package ของ Node.js ด้วยคำสั่งผ่าน TERMINAL

npm init -y

เราก็จะได้ Package เริ่มต้นของ Node.js มาแล้ว

จากนั้น ให้ทำการติดตั้ง Framework ที่ชื่อว่า Express ด้วยคำสั่ง

npm install express

โดย Express จะถูกนำมาใช้สำหรับการสร้าง Web application และ API บน Node.js นั่นเอง

แล้วเราก็จะมาติดตั้งของเล่นเพิ่มเติม ให้ชีวิตง่ายขึ้น ด้วย

npm install debug

npm install morgan

npm install nodemon

npm install bodyParser

โดย Debug และ Morqan จะเป็นตัวช่วยให้เราดูการสถานะต่าง ๆ ของระบบเราได้ง่ายขึ้น

ส่วน Nodemon จะทำให้เราไม่จำเป็นต้อง restart server ของเราทุกครั้งที่มีการแก้โค้ด

และ bodyParser จะทำให้เราสามารถทำงานร่วมกันกับข้อมูลที่เป็น json ได้ง่ายขึ้นนั่นเอง

สุดท้าย ให้เราเข้าไปที่ File package.json และแก้ไขส่วนต่าง ๆ ดังนี้

  "scripts": {

    "start": "set DEBUG=app & nodemon app.js",

    "debug": "set DEBUG=\* & nodemon app.js",

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "nodemonConfig": {

    "restartable": "rs",

    "delay": 2000,

    "env": {

      "PORT": 3000

    }

  },

  "type": "module"

เริ่มสร้าง Server แรกกัน

เริ่มต้นด้วยการสร้าง File หลักในการทำงานของเรา ชื่อว่า app.js และ import และกำหนดการทำงานต่าง ๆ ดังนี้

import express from "express";

import debug from "debug";

import morgan from "morgan";  
import bodyParser from "body-parser";

const app = express();  
const PORT = process.env.PORT;

app.use(morgan("combined"));

app.use(bodyParser.json());

app.get("/", (req, res) => {

  res.send("Welcome to Product management system");

});

app.listen(PORT, () => {

  debug(`Server running at <http://localhost:${PORT}>`);

});

จากนั้น ลองใช้คำสั่ง

npm start

และลองเข้าไปที่ <http://localhost:3000> ก็จะเห็นข้อความว่า Welcome to Product management system ซึ่งหมายถึงว่า สามารถเข้าใช้งานได้อยากปกติแล้วนั่นเอง

มาเริ่มสร้าง CRUD Operations กัน

จากที่ได้เกริ่นนำคร่าว ๆ ไปแล้วข้างต้นบนส่วน สรุปสั้น ๆ ก่อนเริ่มอ่าน ว่า CRUD ย่อมาจาก Create, Read, Update และ Delete แต่ก่อนที่เราจะเริ่มสร้างแต่ละตัวนั้น มาทำความเข้าใจคร่าว ๆ เกี่ยวกับ Routing กันก่อน

Routing คือ การกำหนดวิธีการจัดการกับ HTTP requests ที่มี path และ HTTP verb ที่ระบุ เช่น

POST คือการสร้างข้อมูลใหม่ เข้าสู่ระบบ ในที่นี้จะอยู่ในตัว C: Create

GET คือการเรียกข้อมูลต่าง ๆ จากระบบออกมาดู หรือ R: Read

PUT คือการแก้ไขข้อมูลใด ๆ ในระบบ หรือ U: Update

DELETE คือการลบข้อมูลใด ๆ ออกจากระบบ หรือ D: Delete

สำหรับใน Express.js เราสามารถใช้ app.get(), app.post(), app.put(), app.delete() และอื่น ๆ ในการจัดการ route ได้

ถ้าพอเข้าใจแล้ว เราก็จะมาเริ่มสร้างกันเลย

ขั้นแรก สร้าง Folder ที่ชื่อว่า src จากนั้นก็สร้าง Folder ที่ว่า router ด้านใน src อีกที เพื่อไว้ใช้รวบรวม File router ต่าง ๆ ของระบบเรา

ต่อมา ก็สร้าง File ที่ชื่อว่า productRouter.js และเริ่มเขียนกันได้เลย

import express from "express";

let products = [];

let lastestId;

const productsRouter = express.Router();

export default productsRouter;

ให้ import express มาใช้งานในนี้ด้วย แล้วกำหนดตัวแปร product เพื่อใช้เก็บข้อมูลต่าง ๆ และตัวแปร lastestId เพื่อป้องกันไม่ให้ Id ในระบบซ้ำกัน

กลับมาที่ File app.js ให้เพิ่มการเข้าถึง File productRouter.js โดย

import productsRouter from "./src/router/productRouter.js";

และเรียกใช้โดย

app.use("/products", productsRouter);

แล้วเรามาเริ่มสร้าง CRUD กันได้เลยที่ File productRouter.js

มาเริ่มที่ POST กัน

// POST request

productsRouter.route("/").post((req, res) => {

  if (

    !req.body.name ||

    !req.body.price ||

    isNaN(req.body.price) ||

    isNaN(req.body.stock)

  )

    // if นี้ สำหรับป้องกันไม่ให้ค่าที่จำเป็นถูกบันทึกเป็นค่าว่าง และส่วนที่เป็นตัวเลข ก็ต้องเป็นตัวเลขอย่างถูกต้อง

    return res.send("invalid body").status(204); // หากไม่เป็นไปตามกำหนด จะส่งข้อความว่า "invalid body" พร้อม status 204

  lastestId = lastestId ? lastestId : 0; // เช็คว่าเคยมีข้อมูลในระบบหรือไม่ ก่อนนำไปรวมเป็นก้อนใหม่

  // นำข้อมูลมารวมเป็นก้อนใหม่ ก่อนนำไปเก็บไว้ที่ตัวแปร product

  const newProduct = {

    id: lastestId + 1,

    name: String(req.body.name),

    category: String(req.body.category),

    price: Number(req.body.price),

    stock: Number(req.body.stock),

  };

  lastestId = newProduct.id;

  products.push(newProduct);

  res.json(newProduct); // ทำการส่ง Response กลับไป เมื่อจบการทำงาน

});

จากนั้นก็ GET

// GET request

productsRouter.route("/").get((req, res) => {

  if (products.length === 0)

    return res.status(404).send("There are no products yet."); // เช็คว่า มีข้อมูลในระบบหรือไม่ ถ้าไม่ก็ส่งข้อความ "There are no products yet." กลับ พร้อม status 404

  res.json(products); // หรือหากมีอยู่ ก็ให้แสดงออกไปทั้งหมด

});

productsRouter.route("/:id").get((req, res) => {

  // มีการรับ Params ชื่อ id เข้ามาด้วย

  const product = products.find((p) => p.id === parseInt(req.params.id)); // และนำมาเช็คว่า มีข้อมูลนั้น ๆ ไหม

  if (!product) return res.status(404).send("Product not found."); // ถ้าไม่มี ก็ส่งข้อความ "Product not found." กลับ พร้อม status 404

  res.json(product); // หรือหากมีอยู่ ก็ให้แสดงออกไปทั้งหมด

});

และก็ PUT

// PUT request

productsRouter.route("/:id").put((req, res) => {

  // มีการรับ Params ชื่อ id เข้ามาด้วย

  const product = products.find((p) => p.id === parseInt(req.params.id)); // และนำมาเช็คว่า มีข้อมูลนั้น ๆ ไหม

  if (!product) return res.status(404).send("Product not found."); // ถ้าไม่มี ก็ส่งข้อความ "Product not found." กลับ พร้อม status 404

  if (

    !req.body.name ||

    !req.body.price ||

    isNaN(req.body.price) ||

    isNaN(req.body.stock)

  )

    // if นี้ สำหรับป้องกันไม่ให้ค่าที่จำเป็นถูกบันทึกเป็นค่าว่าง และส่วนที่เป็นตัวเลข ก็ต้องเป็นตัวเลขอย่างถูกต้อง

    return res.send("invalid body").status(204); // หากไม่เป็นไปตามกำหนด จะส่งข้อความว่า "invalid body" พร้อม status 204

  product.name = String(req.body.name);

  product.category = String(req.body.category);

  product.price = Number(req.body.price);

  product.stock = Number(req.body.stock);

  // กำหนดค่าต่าง ๆ เป็นค่าใหม่

  res.json(product); // ทำการส่ง Response กลับไป เมื่อจบการทำงาน

});

และสุดท้ายก็ Delete

// DELETE request

productsRouter.route("/:id").delete((req, res) => {

  // มีการรับ Params ชื่อ id เข้ามาด้วยเช่นเคย

  const productIndex = products.findIndex(

    (p) => p.id === parseInt(req.params.id)

  ); // และนำมาหา index ของข้อมูลนั้น ๆ

  if (productIndex === -1) return res.status(404).send("Product not found."); // ถ้าไม่มี ก็ส่งข้อความ "Product not found." กลับ พร้อม status 404

  const deletedProduct = products.splice(productIndex, 1); // หากพบ ก็จะทำการ splice ข้อมูลนั้น ๆ ออกไปจากระบบ

  res.json(deletedProduct);  // ทำการส่ง Response กลับไป เมื่อจบการทำงาน

});

หลังจากการสร้างมาอย่างยาวนาน เรามาทดสอบกันดูดีกว่า ด้วย Program Postman

สำหรับเส้น POST

A screenshot of a computer

Description automatically generated

จากนั้นก็เส้น GET

A screenshot of a computer

Description automatically generated

และก็เส้น PUT

A screenshot of a computer

Description automatically generated

พร้อมกลับมาเช็คผลการแก้ไขที่เส้น GET อีกครั้ง

A screenshot of a computer

Description automatically generated

และสุดท้ายก็เส้น DELETE

A screenshot of a video

Description automatically generated

พร้อมกลับมาเช็คผลการลบที่เส้น GET อีกครั้ง

A screenshot of a computer

Description automatically generated

จะเห็นว่า ในทุก ๆ เส้นของเรา ทำงานได้อย่างถูกต้องแล้วนั่นเอง

สุดท้ายก่อนจากกัน

เท่านี้ เราก็สามารถสร้าง CRUD Operation พื้นฐานได้สำเร็จ พร้อมนำไปประยุกต์ใช้งานได้อย่างหลากหลายต่อไปแล้ว สำหรับ Code ทั้งหมดของบทความนี้ ผมก็ได้รวบรวมไว้ที่ [https://github.com/thitipongr/Product-Management-System เรียบร้อย](https://github.com/thitipongr/Product-Management-System%20เรียบร้อย)แล้ว สามารถเข้าไปศึกษาเพิ่มเติมกันได้เลยครับ

อ้างอิง

1. DevInit #2 (Back-end Developer) สืบค้นเมื่อ 26 มีนาคม 2567  
   จาก: <https://school.borntodev.com/course/devinit-2-back-end-developer>