**เชื่อมต่อ CRUD Operations กับ MySQL Database**

สรุปสั้น ๆ ก่อนเริ่มอ่าน

สำหรับใครที่ได้เขียน Back-end สำหรับ Web application มาบ้างแล้ว แต่ยังไม่เคยลองเชื่อมต่อกับ Database (ฐานข้อมูล) หรือเคยใช้เพียงข้อมูลภายในเมื่อ run server เหมือนในบทความก่อนก่อนหน้านี้ของผม https://www.borntodev.com/2024/04/05/การสร้าง-crud-operations-ด้วย-node-js/ ทำให้อาจจะมีปัญหาคือ เมื่อมีการ Restart server ข้อมูลทั้งหมดที่ถูกบันทึก ก็จะหายไปทันทีนั่นเอง ในบทความนี้ ผมก็จะมาเสนอวิธีสร้างการเชื่อมต่อกับ Database พร้อมตัวอย่างการใช้งาน จาก CRUD Operations ที่สร้างขึ้นจากทบความก่อนหน้านี้ของผมเอง

ติดตั้งสิ่งที่จำเป็น และ เริ่มสร้าง Server แรก

สามารถเข้าไปทำตามขั้นตอนจากบทความก่อนหน้านี้ได้เลยครับ <https://www.borntodev.com/2024/04/05/การสร้าง-crud-operations-ด้วย-node-js/>

มาเริ่มสร้าง CRUD Operations พร้อมต่อกับ ฐานข้อมูล กัน

สำหรับฐานข้อมูลที่เราจะใช้กันในบทความนี้คือ MySQL ที่เป็นฐานข้อมูลประเภท SQL ที่มีความเหมาะสบกับข้อมูลที่มีความซับซ้อนนั่นเอง

ขั้นแรก เรามาทำการสร้าง MySQL Database กันก่อน ด้วยคำสั่ง SQL ดังนี้

CREATE TABLE products (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL,

category VARCHAR(255),

price FLOAT,

stock INT

);

จากนั้น ให้ทำการติดตั้ง Library สำหรับติดต่อกับ ฐานข้อมูล ของ Node.js กันก่อน ด้วยคำสั่ง

npm install mysql

จากนั้น ให้สร้าง 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";

import mysql from "mysql";

import db\_config from "./../../db\_config.json" assert { type: "json" };

const productsRouter = express.Router();

export default productsRouter;

const con = mysql.createConnection({

  host: db\_config.host,

  port: db\_config.port,

  user: db\_config.user,

  password: db\_config.password,

  database: db\_config.database,

}); // เชื่อมต่อฐานข้อมูลด้วยข้อมูลที่เตรียมไว้ที่ File db\_config

con.connect((err) => {

  if (err) throw err;

  console.log("Database Connected!");

}); // ตรวจสอบการเชื่อมต่อฐานข้อมูล

ให้ import express และ mysql มาใช้งาน แล้วกำหนดตัวแปร con สำหรับเชื่อมต่อ ฐานข้อมูล ด้วยข้อมูลที่เตรียมไว้ พร้อมตรวจสอบการเชื่อมต่อด้วย function connect

จากนั้น ให้ เตรียม SQL สำหรับเรียกรับข้อมูล id ของ product และเรียกใช้เพื่อนำข้อมูล id ล่าสุด มาเก็บไว้ที่ ตัวแปร lastestId เพื่อป้องกันไม่ให้ Id ในระบบซ้ำกัน

let lastestId;

const getLastId\_sql = "SELECT id FROM products ORDER BY id DESC LIMIT 1"; // เตรียม SQL สำหรับเรียกรับข้อมูล id ของ product

con.query(getLastId\_sql, (err, result) => {

  if (err) throw err;

  lastestId = result[0]?.id ? result[0].id : 0;

}); // เรียกขอ และนำข้อมูล id ของ product ล่าสุดมาเก็บเบื่องต้นไว้ที่ lastestId

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

  // 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),

  };

  const insertNewProduct\_sql = `INSERT INTO products (id, name, category, price, stock) VALUES (${newProduct.id}, "${newProduct.name}", "${newProduct.category}",${newProduct.price},${newProduct.stock})`; // เตรียม SQL สำหรับการเพิ่มสินค้าใหม่ จากข้อมูลที่เตรียมไว้

  con.query(insertNewProduct\_sql, (err, result) => {

    if (err) throw res.send(err).status(204);

  }); // เรียกใช้ SQL สำหรับการเพิ่มสินค้าใหม่

  lastestId = newProduct.id; // นำข้อมูล id ล่าสุดที่ถูกบันทึก มาเก็บไว้ใช้ในการเพิ่มข้อมูลรายการต่อไป

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

});

จากนั้นก็ GET

// GET request

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

  const { page = 1, limit = 10 } = req.query; // รายการข้อมูล page และ limit ไว้ โดยกำหนดค่าเริ่มต้นที่ 1 และ 10 ตามลำดับ

  const findWithPage\_sql = `SELECT \* FROM products LIMIT ${limit} OFFSET ${

    page \* limit - limit

  }`; // เตรียม SQL สำหรับการเรียกดูกลุ่มข้อมูลสินค้าตาม page และ limit ที่กำหนดไว้

  con.query(findWithPage\_sql, (err, result) => {

    if (err) throw err;

    if (result.length === 0)

      return res.send("There are no products here.").status(204); // หากไม่มี ก็แสดงข้อความแจ้ง

    res.json(result);

  }); // เรียกใช้ SQL สำหรับการเรียกดูกลุ่มข้อมูลสินค้า

});

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

  const { id } = req.params;

  const fineById = `SELECT \* FROM products WHERE id = ${id}`; // เตรียม SQL สำหรับการเรียกดูข้อมูลสินค้าตาม id ที่กำหนดไว้

  con.query(fineById, (err, result) => {

    if (err) throw err;

    if (result.length === 0) return res.send("Product not found.").status(204); // หากไม่มี ก็แสดงข้อความแจ้ง

    res.json(result);

  });

});

และก็ PUT

// PUT request

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

  const { id } = req.params;

  const fineById = `SELECT \* FROM products WHERE id = ${id}`; // เตรียม SQL สำหรับการเรียกดูข้อมูลสินค้าตาม id ที่กำหนดไว้

  con.query(fineById, (err, findIdResult) => {

    if (err) throw err;

    if (findIdResult.length === 0)

      return res.send("Product not found.").status(204); // หากไม่มี ก็แสดงข้อความแจ้ง

    const oldData = findIdResult[0];

    const packData = `name = "${req.body.name || oldData.name}", category = "${

      req.body.category || oldData.category

    }", price = ${req.body.price || oldData.price}, stock = ${

      req.body.stock || oldData.stock

    }`; // กำหนดให้ หากนำเข้าข้อมูลว่างเปล่า ให้บันทึกเป็นข้อมูลเดิม

    const updateProduct\_sql = `UPDATE products SET ${packData} WHERE id = "${oldData.id}"`; // เตรียม SQL สำหรับการแก้ไขข้อมูลตามที่กำหนด

    con.query(updateProduct\_sql, (err) => {

      if (err) throw err;

      res.json({

        name: req.body.name || oldData.name,

        category: req.body.category || oldData.category,

        price: req.body.price || oldData.price,

        stock: req.body.stock || oldData.stock,

      });

    });

  });

});

และสุดท้ายก็ Delete

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

  const { id } = req.params;

  const fineById = `SELECT \* FROM products WHERE id = ${id}`; // เตรียม SQL สำหรับการเรียกดูข้อมูลสินค้าตาม id ที่กำหนดไว้

  con.query(fineById, (err, findIdResult) => {

    if (err) throw err;

    if (findIdResult.length === 0)

      return res.send("Product not found.").status(204); // หากไม่มี ก็แสดงข้อความแจ้ง

    const deletedProduct = findIdResult[0];

    const deleteProduct\_sql = `DELETE FROM products WHERE id = ${id}`; // เตรียม SQL สำหรับการลบข้อมูลตาม id ที่กำหนด

    con.query(deleteProduct\_sql, (err) => {

      if (err) throw err;

      res.json(deletedProduct);

    });

  });

});

หลังจากการสร้างมาอย่างยาวนาน เรามาทดสอบกันดูดีกว่า ด้วย Program Postman

สำหรับเส้น POST

A screenshot of a computer program

Description automatically generated

จากนั้นก็เส้น GET

A screenshot of a computer

Description automatically generated

และก็เส้น PUT

A screenshot of a computer program

Description automatically generated

พร้อมกลับมาเช็คผลการแก้ไขที่เส้น GET อีกครั้ง

A screenshot of a computer

Description automatically generated

และสุดท้ายก็เส้น DELETE

A screenshot of a computer

Description automatically generated

พร้อมกลับมาเช็คผลการลบที่เส้น GET อีกครั้ง

A screenshot of a computer

Description automatically generated

จะเห็นว่า ในทุก ๆ เส้นของเรา ทำงานได้อย่างถูกต้องแล้วนั่นเอง

สุดท้ายก่อนจากกัน

เท่านี้ เราก็สามารถสร้าง CRUD Operation พื้นฐาน พร้อมเชื่อมต่อ และใช้งานฐานข้อมูล MySQL ได้สำเร็จ พร้อมนำไปประยุกต์ใช้งานได้อย่างหลากหลายต่อไปแล้ว สำหรับ Code ทั้งหมดของบทความนี้ ผมก็ได้รวบรวมไว้ที่ <https://github.com/thitipongr/Product-Management-System-v2> เรียบร้อยแล้ว สามารถเข้าไปศึกษาเพิ่มเติมกันได้เลยครับ

อ้างอิง

1. DevInit #2 (Back-end Developer) สืบค้นเมื่อ 4 เมษายน 2567  
   จาก: <https://school.borntodev.com/course/devinit-2-back-end-developer>