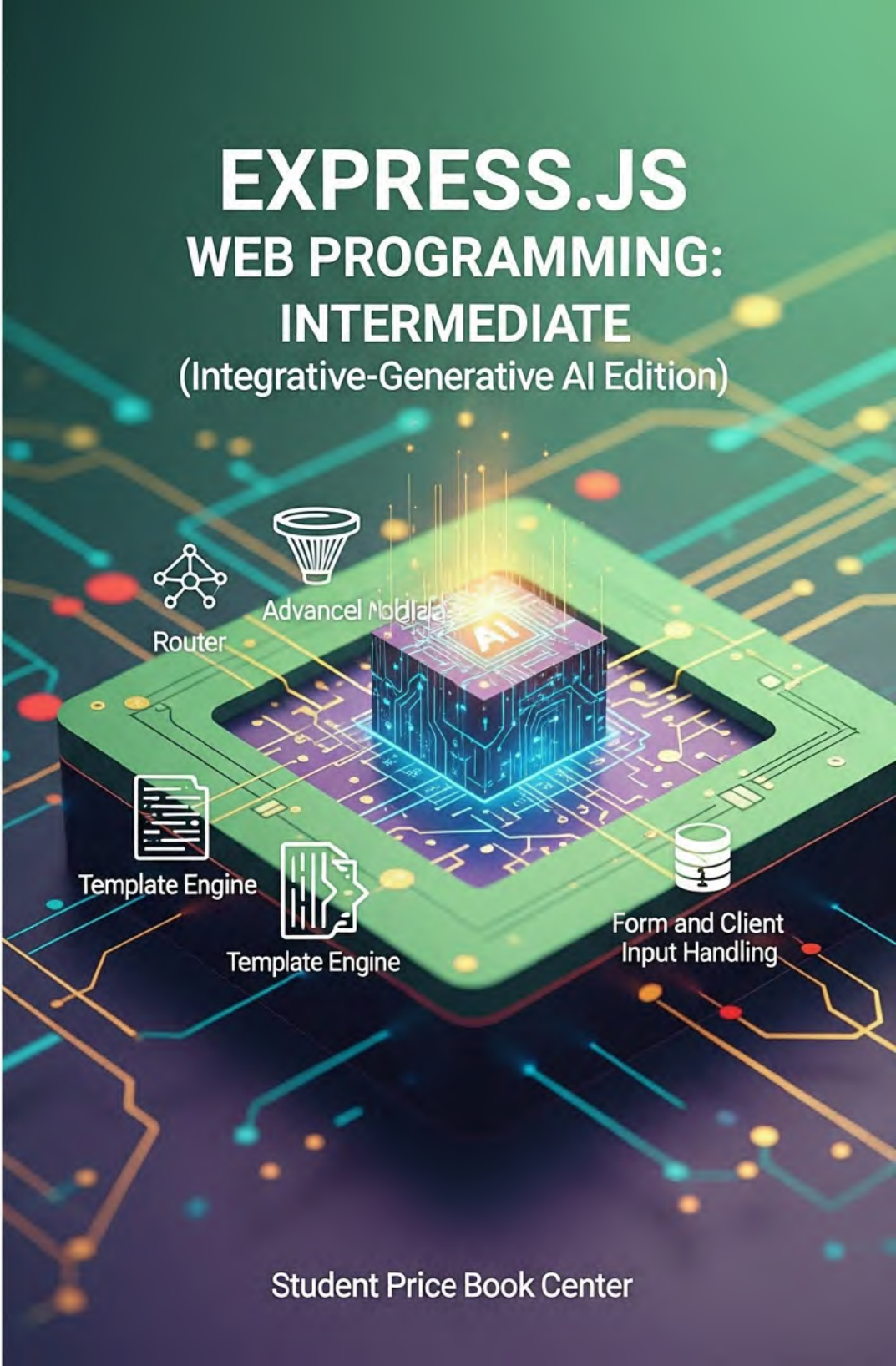
EXPRESS.JS
WEB PROGRAMMING: INTERMEDIATE
(Integrative-Generative AI Edition)



Student Price Book Center

คำนำ

หนังสือ *Express.js Web Programming: Intermediate* เล่มนี้จัดทำขึ้นเพื่อเสริมสร้างความรู้และทักษะให้กับนักพัฒนาเว็บที่มีพื้นฐาน Express.js อยู่แล้ว และต้องการก้าวไปสู่ระดับกลางถึงขั้นสูง โดยเน้นการใช้งานจริงและการประยุกต์ใช้องค์ประกอบสำคัญของ Express.js ในการพัฒนาเว็บแอปพลิเคชันที่มีโครงสร้างชัดเจน มีประสิทธิภาพ และสามารถขยายได้ในระดับองค์กร

ผู้เขียนตระหนักดีว่า แม้ Express.js จะมีลักษณะเรียบง่ายและยืดหยุ่นสูง แต่การพัฒนาแอปพลิเคชันที่ซับซ้อนโดยไม่จัดโครงสร้างให้ดีตั้งแต่ต้น อาจนำไปสู่โค้ดที่ดูแลรักษายากและเกิดข้อผิดพลาดได้ง่าย หนังสือเล่มนี้จึงออกแบบให้ผู้อ่านได้เรียนรู้การปรับปรุงโค้ดให้มีความเป็นระบบ ผ่านเทคนิคการแยกโมดูล การสร้าง middleware ที่มีประสิทธิภาพ การเชื่อมโยงกับ template engine และการจัดการข้อมูลจากฟอร์มของผู้ใช้

เริ่มต้นจาก **บทที่ 5: Router Module และการแยกไฟล์** ผู้อ่านจะได้เรียนรู้การใช้ express.Router() เพื่อแยกเส้นทาง (routes) ออกเป็นโมดูลย่อย ทำให้โค้ดมีความเป็นระเบียบและจัดการง่าย พร้อมตัวอย่างการจัดการ route hierarchy อย่างเป็นระบบ ซึ่งเป็นพื้นฐานสำคัญสำหรับการพัฒนาแอปพลิเคชันขนาดกลางถึงขนาดใหญ่

ถัดมา **บทที่ 6: Middleware ขั้นสูง** จะเน้นไปที่การใช้งาน middleware แบบเจาะลึก ทั้งจาก third-party เช่น body-parser และ cookie-parser และการสร้าง middleware ขึ้นมาเองเพื่อรองรับกระบวนการเฉพาะ รวมถึงการจัดการข้อผิดพลาดด้วย error-handling middleware อย่างเป็นระบบ ซึ่งมีความสำคัญในการเพิ่มความเสถียรให้กับแอปพลิเคชัน

บทที่ 7: Template Engine จะพาผู้อ่านเข้าสู่โลกของการแสดงผล HTML แบบไดนามิก โดยใช้ template engine ยอดนิยมอย่าง Pug และ EJS ผู้อ่านจะได้เรียนรู้การติดตั้ง การส่งข้อมูลจาก server ไปยัง template การสร้าง layout และ partials เพื่อใช้โครงสร้าง HTML ร่วมกันหลายหน้า ทำให้การออกแบบหน้าจอเว็บง่ายขึ้นและสามารถปรับแต่งได้อย่างยืดหยุ่น

สุดท้าย **บทที่ 8: การจัดการ Form และการรับข้อมูลจาก Client** จะสอนวิธีรับข้อมูลจากผู้ใช้อย่างปลอดภัยและมีประสิทธิภาพ ไม่ว่าจะเป็นการอ่านข้อมูลจากแบบฟอร์ม POST การจัดการ multipart/form-data สำหรับการแนบไฟล์ และการใช้งาน multer เพื่ออัปโหลดไฟล์ไปยังเซิร์ฟเวอร์ พร้อมตัวอย่างการใช้งานในสถานการณ์จริงที่สามารถนำไปปรับใช้กับโปรเจกต์ของผู้อ่านได้ทันที

หนังสือเล่มนี้เหมาะสำหรับนักพัฒนาที่ต้องการต่อยอดจากพื้นฐาน Express.js ไปสู่การพัฒนาแอปพลิเคชันระดับมืออาชีพ ผู้เรียนจะได้ฝึกคิดเชิงโครงสร้าง รู้จักการจัดการระบบที่มีหลายองค์ประกอบ และสามารถสร้างระบบที่พร้อมใช้งานในโลกจริงได้อย่างมั่นใจ ขอให้หนังสือเล่มนี้เป็นเครื่องมือสำคัญที่จะช่วยให้คุณพัฒนาแอปพลิเคชันเว็บได้อย่างมีประสิทธิภาพและยั่งยืนในระยะยาว.

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 5

Router Module และการแยกไฟล์

(Router Module and Modularization)

เนื้อหา

บทนำบทที่ 5: Router Module และการแยกไฟล์

ในกระบวนการพัฒนาแอปพลิเคชันด้วย Express.js เมื่อระบบเริ่มมีความซับซ้อนมากขึ้น การจัดการ routing ภายในไฟล์เดียวจะกลายเป็นเรื่องที่ยุ่งยากและไม่ยืดหยุ่น การจัดระเบียบโครงสร้างของ route ให้เป็นระบบจึงเป็นสิ่งจำเป็น เพื่อให้โค้ดสามารถดูแลรักษาได้ง่าย มีความชัดเจน และสามารถขยายได้ในอนาคต บทที่ 5 นี้จึงมุ่งเน้นที่แนวทางการแยก routing ออกเป็นโมดูลย่อยด้วยการใช้งาน express.Router() ซึ่งเป็นฟีเจอร์สำคัญของ Express.js

การใช้งาน express.Router() ช่วยให้นักพัฒนาสามารถสร้าง routing เฉพาะส่วนหรือเฉพาะโมดูลได้อย่างเป็นอิสระ โดยไม่ต้องกระจุกอยู่ในไฟล์หลัก การแยก route แต่ละชุดออกเป็นไฟล์ย่อยที่รับผิดชอบเฉพาะด้าน ช่วยให้สามารถควบคุมและปรับปรุงโค้ดในระดับโมดูลได้อย่างมีประสิทธิภาพ ซึ่งสอดคล้องกับแนวคิด modularization ที่นิยมใช้ในการพัฒนาระบบขนาดกลางถึงขนาดใหญ่

นอกจากนี้ บทนี้ยังอธิบายแนวทางการจัดการโครงสร้าง route hierarchy อย่างมีระบบ เช่น การสร้าง route ซ้อนกัน (nested routing) และการกำหนด prefix เพื่อให้สามารถกำหนด path หลักของ route ย่อยได้อย่างเหมาะสม การวางโครงสร้าง route hierarchy อย่างถูกต้อง จะช่วยเพิ่มความเข้าใจของทีมพัฒนาและลดความสับสนระหว่างเส้นทางที่มีลักษณะใกล้เคียงกัน

เพื่อให้ผู้อ่านสามารถนำไปประยุกต์ใช้งานได้จริง บทนี้จึงมีตัวอย่างการสร้างไฟล์ routing สำหรับโมดูลต่าง ๆ เช่น /users, /products, หรือ /orders พร้อมแนวทางการนำเข้ามาใช้งานในไฟล์หลักของแอปพลิเคชัน (เช่น app.js หรือ server.js) ตัวอย่างเหล่านี้จะช่วยให้ผู้อ่านเห็นภาพการออกแบบ route แบบแยกไฟล์ที่ชัดเจนยิ่งขึ้น

เมื่อจบบทนี้ ผู้อ่านจะมีความเข้าใจในหลักการแยก routing ออกเป็นโมดูลย่อย และสามารถนำแนวคิด modular routing ไปใช้จัดระเบียบโปรเจกต์ได้อย่างมีประสิทธิภาพ ทั้งนี้จะเป็นพื้นฐานสำคัญสำหรับการพัฒนาแอปพลิเคชันขนาดใหญ่ด้วย Express.js ที่สามารถดูแลรักษาและขยายได้ในระยะยาว.

Router Module และการแยกไฟล์

---

1. การใช้งาน express.Router()

- express.Router() คือฟีเจอร์ของ Express ที่ช่วยให้เราสามารถสร้างกลุ่มของ route ที่แยกจากกันได้อย่างอิสระ
- Router ช่วยทำให้โค้ดสะอาดและดูแลง่ายขึ้น โดยเฉพาะโปรเจกต์ใหญ่ที่มีหลาย route
- Router คือ mini-application ที่สามารถใช้ middleware, routes เหมือนกับแอปหลักได้

ตัวอย่างใช้งาน

```
const express = require('express');
const app = express();
const router = express.Router();

router.get('/hello', (req, res) => {
  res.send('Hello from router!');
});

app.use('/api', router);

app.listen(3000);
```

ในตัวอย่างนี้ เมื่อ client เรียก /api/hello จะได้ข้อความ "Hello from router!"

---

2. การแยก route ออกเป็นไฟล์ย่อย

- แทนที่จะเขียน route ทั้งหมดในไฟล์หลัก (เช่น index.js) เราสามารถแยกแต่ละกลุ่ม route ไปเป็นไฟล์ย่อยได้
- แต่ละไฟล์จะ export ตัว router ออกมา และในไฟล์หลักจะ import และ mount router เหล่านั้น

ตัวอย่างโครงสร้างไฟล์

```
project/
├── index.js
└── routes/
```

```
├── users.js
└── products.js
```

ตัวอย่างไฟล์ routes/users.js

```
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.send('User list');
});

router.get('/:id', (req, res) => {
  res.send(`User ID: ${req.params.id}`);
});

module.exports = router;
```

ตัวอย่างไฟล์ index.js

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

const usersRouter = require('./routes/users');

app.use('/users', usersRouter);

app.listen(3000, () => {
  console.log('Server started on port 3000');
});
```

---

3. การจัดการ route hierarchy และ modularization

- การแยก router ช่วยให้เราสามารถสร้าง route hierarchy ได้ง่าย เช่น /users, /users/:id, /products
- สามารถใช้ middleware ใน router ย่อยแต่ละตัวได้
- เพิ่มความยืดหยุ่นและง่ายต่อการดูแลรักษาโปรเจกต์ใหญ่

ตัวอย่างเพิ่ม middleware ใน router

```
// routes/users.js
```

```
router.use((req, res, next) => {
  console.log('Users router middleware');
  next();
});
```

สรุป

หัวข้อ	รายละเอียด
express.Router()	สร้าง router ย่อยที่เป็น mini-app
แยกไฟล์ router	แยก route ตามฟีเจอร์หรือโมดูล
route hierarchy	จัดลำดับ route และใช้ middleware ได้
modularization	เพิ่มความสะดวกในการจัดการและขยายระบบ

Router Module และการแยกไฟล์ — รายละเอียดเชิงลึก

---

1. การใช้งาน express.Router()

ความหมายและหน้าที่

- express.Router() คืออ็อบเจกต์ที่ให้เราสร้าง **กลุ่มของเส้นทาง (routes) และ middleware** แยกกันได้
- Router ทำงานเหมือน mini Express app ที่จัดการเฉพาะ route หรือโมดูลหนึ่ง ๆ
- ช่วยให้โค้ดมีความเป็นโมดูล (modular) และดูแลง่าย

วิธีสร้างและใช้งาน

```
const express = require('express');
const router = express.Router();

// สร้าง route ภายใน router
router.get('/test', (req, res) => {
  res.send('Hello from Router!');
});

module.exports = router;
```

- นำ router นี้ไปใช้งานในแอปหลัก เช่น

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

```
const myRouter = require('./myRouter');

app.use('/api', myRouter);
```

- เส้นทาง /api/test จะเรียก router ข้างต้น

ข้อดีของการใช้ Router

- แยกความรับผิดชอบของแต่ละ route ชัดเจน
- สามารถใช้ middleware ใน router ได้เฉพาะเจาะจง
- รองรับการขยายระบบเมื่อโปรเจกต์ใหญ่ขึ้น

---

2. การแยก route ออกเป็นไฟล์ย่อย (Modularization)

เหตุผลที่ต้องแยกไฟล์

- ถ้าเขียน route ทั้งหมดในไฟล์เดียวจะทำให้โค้ดยาวและดูแลยาก
- การแยกไฟล์ช่วยให้แบ่งแยกฟีเจอร์แต่ละส่วน (เช่น users, products, orders) ได้ชัดเจน
- ทีมพัฒนาสามารถทำงานแยกกันได้

โครงสร้างโปรเจกต์แนะนำ

```
project/
├── index.js        // แอปหลัก
└── routes/
    ├── users.js    // route สำหรับผู้ใช้
    ├── products.js // route สำหรับสินค้า
    └── orders.js   // route สำหรับคำสั่งซื้อ
```

ตัวอย่างไฟล์ routes/users.js

```
const express = require('express');
const router = express.Router();

// middleware ใน router users
router.use((req, res, next) => {
  console.log('Users router middleware');
  next();
});

// route แสดงรายชื่อผู้ใช้
router.get('/', (req, res) => {
  res.json([{ id: 1, name: 'Alice' }, { id: 2, name: 'Bob' }]);
```

```
});

// route แสดงรายละเอียดผู้ใช้ตาม id
router.get('/:id', (req, res) => {
  const id = parseInt(req.params.id);
  res.json({ id, name: `User${id}` });
});

module.exports = router;
```

การนำไปใช้ในไฟล์หลัก (index.js)

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

const usersRouter = require('./routes/users');
const productsRouter = require('./routes/products');

app.use('/users', usersRouter);
app.use('/products', productsRouter);

app.listen(3000, () => console.log('Server running at http://localhost:3000'));
```

---

3. การจัดการ Route Hierarchy และ Modularization

Hierarchy ของ route

- สามารถจัดลำดับ route ด้วย prefix ที่เหมาะสม เช่น /users, /users/:id
- แต่ละ router มีเส้นทางแยกกันตามโมดูล และ mount ไว้ในแอปหลัก

การใช้ Middleware ใน router

- สามารถกำหนด middleware เฉพาะ router เพื่อจัดการ request ก่อนเข้าถึง route ต่าง ๆ
- ตัวอย่าง:

```
// middleware ใน router
router.use((req, res, next) => {
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.originalUrl}`);
  next();
});
```

Modularization เต็มรูปแบบ

- แยกแต่ละฟีเจอร์ (users, products, orders) เป็น router ต่างหาก
- แต่ละ router อาจมี middleware และ logic ของตัวเอง
- ง่ายต่อการ maintain, ทดสอบ, และขยายระบบ

ตัวอย่างการ mount router หลายระดับ

```
// ใน index.js
app.use('/api/users', usersRouter);
app.use('/api/products', productsRouter);
```

ในแต่ละ router เช่น

```
// routes/users.js
router.get('/:id/orders', (req, res) => {
  // ดึง order ของ user นั้น ๆ
});
```

สรุปข้อดีของการใช้ Router Module และแยกไฟล์

ข้อดี	คำอธิบาย
Modularization	แยกโค้ดเป็นโมดูลย่อย ดูแลง่าย
Reusability	นำ router ย่อยไปใช้ซ้ำในหลายที่ได้
Maintainability	จัดการง่ายเมื่อระบบใหญ่
Middleware Scope Control	กำหนด middleware เฉพาะ router ได้ ไม่กระทบ router อื่น
Cleaner Main App File	ไฟล์หลักไม่รก อ่านง่ายขึ้น

การใช้งาน express.Router()

---

1. ความหมายของ express.Router()

- เป็นอ็อบเจ็กต์ตัวช่วยใน Express ที่ทำหน้าที่ **จัดกลุ่ม route และ middleware** ที่เกี่ยวข้องกัน
- ทำให้เราสามารถแยก route เป็นโมดูลย่อยๆ ได้สะดวก
- Router เหมือน mini app ที่มีฟังก์ชันครบ เช่น รับ request, ใช้ middleware, ส่ง response ฯลฯ

---

2. วิธีสร้าง Router

```
const express = require('express');
const router = express.Router();
```

- สร้าง instance ของ router ด้วย express.Router()

---

3. การเพิ่มเส้นทาง (Routes) ใน Router

```
router.get('/hello', (req, res) => {
  res.send('Hello from Router!');
});

router.post('/submit', (req, res) => {
  res.send('Form submitted!');
});
```

- กำหนด route และ method ตามปกติ แต่ใน router นี้

---

4. การใช้ Middleware ใน Router

```
router.use((req, res, next) => {
  console.log('Router-level middleware');
  next();
});
```

- สามารถใช้ middleware เฉพาะ router นี้ได้ เช่น ตรวจสอบ authentication, logging ฯลฯ

---

5. การเชื่อมต่อ Router กับแอปหลัก (Mounting Router)

```
const express = require('express');
const app = express();
const myRouter = require('./myRouter');  // ไฟล์ที่ export router

app.use('/api', myRouter);
```

- เส้นทางทั้งหมดใน myRouter จะถูกต่อด้วย prefix /api
- เช่น myRouter มี route /hello เวลาเรียกจริงจะเป็น /api/hello

---

6. ตัวอย่างโค้ดครบ

```
// myRouter.js
const express = require('express');
const router = express.Router();

router.use((req, res, next) => {
```

```
  console.log(`Request to ${req.originalUrl}`);
  next();
});

router.get('/hello', (req, res) => {
  res.send('Hello from Router!');
});

module.exports = router;

// index.js
const express = require('express');
const app = express();
const myRouter = require('./myRouter');

app.use('/api', myRouter);

app.listen(3000, () => {
  console.log('Server started on port 3000');
});
```

---

7. สรุปข้อดีของ express.Router()

- แบ่ง route เป็นโมดูลแยกตามฟีเจอร์
- สามารถใช้ middleware แยกเฉพาะ router ได้
- ทำให้โค้ดดูแลง่ายและยืดหยุ่นในโปรเจกต์ใหญ่

นี่คือตัวอย่างโปรแกรม Express.js แบบเต็มไฟล์ จำนวน 3 โปรแกรมพื้นฐาน และ 3 โปรแกรมแนวประยุกต์ ที่เน้นการใช้งาน express.Router() แยกไฟล์ พร้อมโครงสร้างและคำอธิบายโค้ด รวมผลการรัน

ตัวอย่างพื้นฐาน 3 โปรแกรม

ตัวอย่าง 1: Router แยกไฟล์ สำหรับ Users

โครงสร้างโปรเจกต์

```
basic-app-1/
├── index.js
└── routes/
    └── users.js
```

โค้ด routes/users.js

```
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.send('User List');
});

router.get('/:id', (req, res) => {
  res.send(`User ID: ${req.params.id}`);
});

module.exports = router;
```

โค้ด index.js

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

const usersRouter = require('./routes/users');

app.use('/users', usersRouter);

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

คำอธิบาย

- แยก route /users และ /users/:id ไว้ในไฟล์ routes/users.js
- นำ router มาติดตั้งที่ path /users ใน index.js
- เมื่อเรียก http://localhost:3000/users จะได้ข้อความ “User List”
- เรียก http://localhost:3000/users/123 จะได้ “User ID: 123”

ผลการรัน (curl)

```
curl http://localhost:3000/users
# User List

curl http://localhost:3000/users/123
# User ID: 123
```

ตัวอย่าง 2: Router แยกไฟล์ สำหรับ Products พร้อม Middleware

โครงสร้างโปรเจกต์

```
basic-app-2/
├── index.js
└── routes/
    └── products.js
```

โค้ด routes/products.js

```
const express = require('express');
const router = express.Router();

// Middleware logger
router.use((req, res, next) => {
  console.log(`Products router: ${req.method} ${req.originalUrl}`);
  next();
});

router.get('/', (req, res) => {
  res.json([{ id: 1, name: 'Laptop' }, { id: 2, name: 'Phone' }]);
});

router.get('/:id', (req, res) => {
  res.json({ id: req.params.id, name: `Product ${req.params.id}` });
});

module.exports = router;
```

โค้ด index.js

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

const productsRouter = require('./routes/products');

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

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

คำอธิบาย

- เพิ่ม middleware logger ภายใน router เพื่อแสดง HTTP method และ URL
- ส่ง response เป็น JSON รายการสินค้าและสินค้าแต่ละชิ้น
- เรียก /products ได้รายการสินค้าทั้งหมด
- เรียก /products/10 ได้ข้อมูลสินค้า id=10

ผลการรัน (curl)

```
curl http://localhost:3000/products
# [{"id":1,"name":"Laptop"},{"id":2,"name":"Phone"}]

curl http://localhost:3000/products/10
# {"id":"10","name":"Product 10"}
```

ตัวอย่าง 3: Router สำหรับ Orders พร้อม Route Nested

โครงสร้างโปรเจกต์

```
basic-app-3/
├── index.js
└── routes/
    └── orders.js
```

โค้ด routes/orders.js

```
const express = require('express');
const router = express.Router();
```

```
// รายการออร์เดอร์ทั้งหมด
router.get('/', (req, res) => {
  res.send('Orders list');
});

// รายละเอียดออร์เดอร์ตาม id
router.get('/:orderId', (req, res) => {
  res.send(`Order ID: ${req.params.orderId}`);
});

// สินค้าในออร์เดอร์
router.get('/:orderId/items', (req, res) => {
  res.send(`Items in Order ID: ${req.params.orderId}`);
});

module.exports = router;
```

โค้ด index.js

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

const ordersRouter = require('./routes/orders');

app.use('/orders', ordersRouter);

app.listen(3000, () => {
  console.log('Server running at http://localhost:3000');
});
```

คำอธิบาย

- มี route หลัก /orders สำหรับรายการออร์เดอร์
- route nested /orders/:orderId/items สำหรับสินค้าภายในออร์เดอร์นั้น
- แสดงการใช้ route hierarchy และ parameters

ผลการรัน (curl)

```
curl http://localhost:3000/orders
# Orders list

curl http://localhost:3000/orders/45
# Order ID: 45

curl http://localhost:3000/orders/45/items
# Items in Order ID: 45
```

ตัวอย่างแนวประยุกต์ 3 โปรแกรม

ตัวอย่าง 4: User API พร้อม CRUD Operation

โครงสร้างโปรเจกต์

```
app-advanced-1/
├── index.js
└── routes/
    └── users.js
```

โค้ด routes/users.js

```
const express = require('express');
const router = express.Router();

let users = [
  { id: 1, name: 'Alice' },
  { id: 2, name: 'Bob' },
];

// อ่านผู้ใช้ทั้งหมด
router.get('/', (req, res) => {
  res.json(users);
});

// อ่านผู้ใช้ตาม id
router.get('/:id', (req, res) => {
  const id = parseInt(req.params.id);
```

```
  const user = users.find(u => u.id === id);
  if (!user) return res.status(404).json({ error: 'User not found' });
  res.json(user);
});

// สร้างผู้ใช้ใหม่
router.post('/', express.json(), (req, res) => {
  const { name } = req.body;
  if (!name) return res.status(400).json({ error: 'Name is required' });
  const newUser = { id: users.length + 1, name };
  users.push(newUser);
  res.status(201).json(newUser);
});

// แก้ไขผู้ใช้
router.put('/:id', express.json(), (req, res) => {
  const id = parseInt(req.params.id);
  const { name } = req.body;
  const userIndex = users.findIndex(u => u.id === id);
  if (userIndex === -1) return res.status(404).json({ error: 'User not found' });
  users[userIndex].name = name || users[userIndex].name;
  res.json(users[userIndex]);
});

// ลบผู้ใช้
router.delete('/:id', (req, res) => {
  const id = parseInt(req.params.id);
  users = users.filter(u => u.id !== id);
  res.status(204).send();
});

module.exports = router;
```

โค้ด index.js

```
const express = require('express');
```

```
const app = express();

const usersRouter = require('./routes/users');

app.use('/users', usersRouter);

app.listen(3000, () => {
  console.log('User API running on port 3000');
});
```

คำอธิบาย

- ใช้ router แยกไฟล์จัดการ User API แบบ CRUD
- ใช้ express.json() middleware สำหรับอ่าน JSON body ใน POST, PUT
- ตรวจสอบ error และส่ง status code ตามมาตรฐาน HTTP

ผลการรัน (curl)

```
curl -i http://localhost:3000/users
# 200 OK [{"id":1,"name":"Alice"},{"id":2,"name":"Bob"}]

curl -i http://localhost:3000/users/1
# 200 OK {"id":1,"name":"Alice"}

curl -i -X POST http://localhost:3000/users -H "Content-Type: application/json" -d
'{"name":"Charlie"}'
# 201 Created {"id":3,"name":"Charlie"}

curl -i -X PUT http://localhost:3000/users/3 -H "Content-Type: application/json" -d
'{"name":"Charles"}'
# 200 OK {"id":3,"name":"Charles"}

curl -i -X DELETE http://localhost:3000/users/3
# 204 No Content
```

ตัวอย่าง 5: Blog API แยก router พร้อม Middleware ตรวจสอบ Authentication (ง่าย ๆ)

โครงสร้างโปรเจกต์

```
app-advanced-2/
├── index.js
└── routes/
    └── posts.js
```

โค้ด routes/posts.js

```
const express = require('express');
const router = express.Router();

let posts = [
  { id: 1, title: 'First Post', content: 'Hello World' },
];

// middleware ตรวจสอบ token ง่ายๆ
router.use((req, res, next) => {
  const token = req.headers['x-auth-token'];
  if (!token || token !== 'secret-token') {
    return res.status(401).json({ error: 'Unauthorized' });
  }
  next();
});

router.get('/', (req, res) => {
  res.json(posts);
});

router.post('/', express.json(), (req, res) => {
  const { title, content } = req.body;
  if (!title || !content) {
    return res.status(400).json({ error: 'Title and content required' });
  }
  const newPost = { id: posts.length + 1, title, content };
  posts.push(newPost);
  res.status(201).json(newPost);
```