**เฉลยใบงานวิชา Web Programming (ฉบับวิเคราะห์)**

**ข้อ 1: การวิเคราะห์โครงสร้างเว็บ (HTML)**

**คำตอบ:** ควรปรับปรุงแท็ก div ต่างๆ ให้เป็น Semantic HTML5 ดังนี้:

* <div class="card"> ควรเปลี่ยนเป็น <article> เนื่องจากเป็นส่วนของเนื้อหาที่สมบูรณ์ในตัวเองและสามารถนำไปใช้ที่อื่นได้
* <div class="card-header"> ควรเปลี่ยนเป็น <header> เพื่อระบุว่าเป็นส่วนหัวของ <article>
* <div class="card-body"> สามารถเอา div ออกได้ เพื่อให้เนื้อหาอยู่ภายใต้ <article> โดยตรง ทำให้โครงสร้างเรียบง่ายขึ้น
* <div class="card-footer"> ควรเปลี่ยนเป็น <footer> เพื่อระบุว่าเป็นส่วนท้ายของ <article>

**โค้ดที่ปรับปรุงแล้ว:**

รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

**คำอธิบายอย่างละเอียด:** การใช้ Semantic Tags มีความสำคัญอย่างยิ่งด้วยเหตุผล 3 ประการหลัก:

1. **การเข้าถึง (Accessibility):** โปรแกรมช่วยอ่านหน้าจอ (Screen Readers) สำหรับผู้พิการทางสายตา จะเข้าใจโครงสร้างของหน้าเว็บได้ดีขึ้น สามารถบอกผู้ใช้ได้ว่าส่วนนี้คือหัวเรื่อง (header), เนื้อหาหลัก, หรือส่วนท้าย (footer) ทำให้ผู้ใช้เข้าใจบริบทของข้อมูลได้ง่ายขึ้น
2. **SEO (Search Engine Optimization):** Search Engines เช่น Google จะเข้าใจความหมายและความสำคัญของเนื้อหาในแต่ละส่วนได้ดีขึ้น เช่น เนื้อหาในแท็ก <article> หรือ <h1> จะถูกให้น้ำหนักความสำคัญมากกว่าเนื้อหาใน div ทั่วไป ซึ่งส่งผลดีต่ออันดับในการค้นหา
3. **ความสามารถในการบำรุงรักษา (Maintainability):** สำหรับนักพัฒนา การอ่านโค้ดที่มีความหมายในตัวจะทำให้เข้าใจโครงสร้างได้รวดเร็วกว่าการอ่าน div ซ้อนกันหลายๆ ชั้น ช่วยให้การแก้ไขและพัฒนาต่อทำได้ง่ายขึ้น

**ข้อ 2: การวิเคราะห์ผลลัพธ์จาก CSS (CSS)**

**คำตอบ:** <p class="highlight-text"> จะแสดงผลเป็นข้อความ "โปรโมชั่นพิเศษ!" ที่มีลักษณะดังนี้:

* **สีพื้นหลัง:** สีเหลือง (yellow)
* **สีตัวอักษร:** สีแดง (red)
* **ขนาดตัวอักษร:** 16px
* **ขอบ:** เส้นประ (dashed) หนา 2px สีแดง (red)

**คำอธิบายอย่างละเอียด:** ผลลัพธ์นี้เกิดจากหลักการของ **CSS Specificity (ความจำเพาะเจาะจง)** และ **Cascading (การไล่ลำดับ)**

1. เบราว์เซอร์จะเริ่มจาก Style พื้นฐานของแท็ก p ซึ่งกำหนด color: blue และ font-size: 16px
2. จากนั้น เบราว์เซอร์จะพิจารณา Class Selector (.highlight-text) ซึ่งมีความจำเพาะเจาะจง **สูงกว่า** Element Selector (p)
3. ดังนั้น Style Rules ที่อยู่ใน .highlight-text จะถูกนำมาใช้ **ทับ** (override) Style ที่ขัดแย้งกันใน p นั่นคือ color: red จะทับ color: blue
4. ส่วน Style ที่ไม่ขัดแย้งกัน เช่น font-size: 16px จาก p จะยังคงถูกนำมาใช้งานตามปกติ
5. Style อื่นๆ ที่มีเฉพาะใน .highlight-text เช่น background-color และ border ก็จะถูกนำมาใช้กับ element นี้โดยตรง

**ข้อ 3: การวิเคราะห์การทำงานของ JavaScript (JavaScript)**

**คำตอบ:** หลังจากผู้ใช้คลิกปุ่ม "ยืนยันข้อมูล" จะเกิดเหตุการณ์ต่อไปนี้ตามลำดับ:

1. ข้อความใน <p id="status-message"> จะเปลี่ยนจาก "กรุณารอสักครู่..." เป็น "บันทึกข้อมูลสำเร็จ!"
2. สีของข้อความนั้นจะเปลี่ยนเป็นสีเขียว
3. รูปแบบของข้อความนั้นจะเปลี่ยนเป็นตัวหนา
4. ข้อความบนปุ่ม <button id="confirm-btn"> จะเปลี่ยนจาก "ยืนยันข้อมูล" เป็น "ดำเนินการเรียบร้อย"
5. ปุ่มนั้นจะถูกปิดการใช้งาน (disabled) ทำให้ไม่สามารถคลิกซ้ำได้อีก

**ผลลัพธ์สุดท้ายที่ผู้ใช้เห็น:** ผู้ใช้จะเห็นข้อความ "บันทึกข้อมูลสำเร็จ!" ที่เป็นตัวหนาสีเขียว และปุ่มที่เขียนว่า "ดำเนินการเรียบร้อย" ซึ่งเป็นสีเทาจางๆ และไม่สามารถกดได้

**คำอธิบายอย่างละเอียด:** โค้ดนี้ทำงานโดยการผูก event listener เข้ากับเหตุการณ์ click ของปุ่ม เมื่อการคลิกเกิดขึ้น ฟังก์ชันลูกศร (arrow function) ที่เป็น callback จะถูกเรียกใช้งาน ซึ่งภายในฟังก์ชันจะทำการจัดการ DOM (DOM Manipulation) โดยตรง:

* message.textContent = ...: เปลี่ยนแปลงเนื้อหาข้อความ
* message.style.color = ... และ message.style.fontWeight = ...: เพิ่ม inline style เพื่อเปลี่ยนสีและรูปแบบตัวอักษร
* button.textContent = ...: เปลี่ยนข้อความบนปุ่ม
* button.disabled = true;: เปลี่ยน attribute disabled ของปุ่มให้เป็น true ซึ่งเป็นกลไกป้องกันการส่งข้อมูลซ้ำซ้อน

**ข้อ 4: การวิเคราะห์ความสอดคล้องของข้อมูล (TypeScript)**

**คำตอบ:** object ชื่อ newUser **ไม่สอดคล้อง** กับ interface User มีจุดที่ผิดพลาด 2 จุดคือ:

1. userId: ใน interface กำหนดให้เป็น number แต่ใน object เป็น string ('U001')
2. isActive: ใน interface กำหนดให้เป็น boolean แต่ใน object เป็น string ('true')

**ต้องแก้ไขดังนี้จึงจะถูกต้อง:**

รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

**คำอธิบายอย่างละเอียด:** TypeScript เป็นระบบ Static Type Checking ซึ่งหมายความว่ามันจะตรวจสอบชนิดข้อมูล ณ เวลาคอมไพล์ (compile-time) ก่อนที่โค้ดจะถูกรัน interface ทำหน้าที่เหมือน "พิมพ์เขียว" หรือ "สัญญา" ที่กำหนดว่า object ที่จะใช้งานร่วมกับ type นี้ต้องมีรูปร่างหน้าตาและชนิดข้อมูลของแต่ละ property ตรงตามที่กำหนดไว้

* readonly userId: number;: หมายความว่า property userId ต้องเป็นตัวเลขและเมื่อกำหนดค่าครั้งแรกแล้วจะเปลี่ยนแปลงไม่ได้อีก
* isActive: boolean;: หมายความว่า property isActive ต้องมีค่าเป็น true หรือ false เท่านั้น
* lastLogin?: Date;: เครื่องหมาย ? หมายถึง property นี้เป็น optional คือจะมีหรือไม่มีก็ได้ใน object

**ข้อ 5: การออกแบบสถาปัตยกรรมคอมโพเนนต์ (React)**

**คำตอบ:** หน้า "ตะกร้าสินค้า" สามารถแบ่งออกเป็นคอมโพเนนต์ย่อยๆ ได้ดังนี้:

* **ShoppingCartPage (คอมโพเนนต์หลัก/หน้า):**
  + **หน้าที่:** เป็นคอมโพเนนต์ที่ใหญ่ที่สุด ทำหน้าที่เป็นคอนเทนเนอร์สำหรับคอมโพเนนต์ย่อยอื่นๆ ทั้งหมด จัดการ State หลักของตะกร้าสินค้า (เช่น รายการสินค้าทั้งหมด) และอาจจะดึงข้อมูลตะกร้าสินค้ามาจาก API
* **CartItemList (รายการสินค้าในตะกร้า):**
  + **หน้าที่:** รับข้อมูลรายการสินค้าทั้งหมด (array of items) มาจาก ShoppingCartPage ผ่าน props แล้วทำการวนลูป (map) เพื่อแสดงผล CartItem แต่ละชิ้น ไม่จัดการ state ของตัวเอง แต่ส่งต่อฟังก์ชันสำหรับลบหรือแก้ไขสินค้าไปยัง CartItem
* **CartItem (สินค้าแต่ละชิ้นในตะกร้า):**
  + **หน้าที่:** แสดงข้อมูลของสินค้าเพียง **ชิ้นเดียว** เช่น รูปภาพ, ชื่อ, ราคา, จำนวน และปุ่มสำหรับเพิ่ม/ลดจำนวน หรือลบสินค้าออกจากตะกร้า รับข้อมูลและฟังก์ชันต่างๆ มาจาก CartItemList ผ่าน props
* **OrderSummary (สรุปยอดคำสั่งซื้อ):**
  + **หน้าที่:** แสดงข้อมูลสรุปผลการคำนวณ เช่น ราคารวม, ส่วนลด, ค่าจัดส่ง และยอดสุทธิที่ต้องชำระ รับข้อมูลราคารวมมาจาก ShoppingCartPage ผ่าน props และมีปุ่ม "ดำเนินการชำระเงิน"

**คำอธิบายอย่างละเอียด:** การออกแบบสถาปัตยกรรมแบบนี้เป็นไปตามหลักการ **Single Responsibility Principle** คือแต่ละคอมโพเนนต์มีหน้าที่รับผิดชอบเพียงอย่างเดียว ทำให้โค้ดเป็นสัดส่วน, นำกลับมาใช้ใหม่ได้ง่าย (Reusable), ทดสอบง่าย (Testable), และบำรุงรักษาง่าย (Maintainable) ข้อมูลจะไหลจากบนลงล่าง (Top-down data flow) จาก ShoppingCartPage ไปยังคอมโพเนนต์ย่อยๆ ผ่านทาง props

**ข้อ 6: การวิเคราะห์การทำงานของเซิร์ฟเวอร์ (Node.js)**

**คำตอบ:**

* **http://localhost:8080/**:
  + **Status Code:** 200 (OK)
  + **Response:** <h1>Homepage</h1> (ข้อความ HTML)
* **http://localhost:8080/about**:
  + **Status Code:** 200 (OK)
  + **Response:** About Us page (ข้อความธรรมดา - text/plain)
* **http://localhost:8080/products**:
  + **Status Code:** 404 (Not Found)
  + **Response:** <h2>404 - Not Found</h2> (ข้อความ HTML)

**คำอธิบายอย่างละเอียด:** โค้ดนี้สร้างเซิร์ฟเวอร์พื้นฐานด้วย http module ของ Node.js ทุกครั้งที่มี request เข้ามา, callback function (req, res) => { ... } จะทำงานโดยตรวจสอบ req.url (path ของ URL ที่ร้องขอ)

* เมื่อ req.url ตรงกับ '/'เงื่อนไข if แรกจะเป็นจริง เซิร์ฟเวอร์จะตอบกลับด้วย status 200 และเนื้อหา HTML
* เมื่อ req.url ตรงกับ '/about'เงื่อนไข else if จะเป็นจริง เซิร์ฟเวอร์จะตอบกลับด้วย status 200 และเนื้อหาแบบ text ธรรมดา
* เมื่อ req.url เป็นค่าอื่น (เช่น '/products') ซึ่งไม่ตรงกับสองเงื่อนไขแรก โปรแกรมจะทำงานในบล็อก else สุดท้าย และตอบกลับด้วย status 404 (Not Found)

**ข้อ 7: การวิเคราะห์เส้นทาง API (Express.js)**

**คำตอบ:**

**การจับคู่ Route กับหลักการ CRUD:**

* **Create:** Route 2 - app.post('/api/users', ...)
* **Read:**
  + Route 1 - app.get('/api/users', ...) (อ่านข้อมูลผู้ใช้ทั้งหมด)
  + Route 3 - app.get('/api/users/:userId', ...) (อ่านข้อมูลผู้ใช้คนเดียว)
* **Update:** Route 5 - app.put('/api/users/:userId', ...)
* **Delete:** Route 4 - app.delete('/api/users/:userId', ...)

**การใช้งาน req.params.userId และ req.body:**

* **req.params.userId** ถูกใช้ใน **Route 3, 4, และ 5**
  + **วัตถุประสงค์:** เพื่อระบุตัวตน (Identifier) ของผู้ใช้ที่ต้องการจะกระทำด้วย เช่น ดึงข้อมูล, ลบ, หรืออัปเดตผู้ใช้ที่มี ID ตรงกับค่าที่ส่งมาใน URL
* **req.body** ถูกใช้ใน **Route 2 และ 5**
  + **วัตถุประสงค์:** เพื่อรับข้อมูล (Payload) ที่ถูกส่งมาจาก Client ในส่วน body ของ request
    - ใน POST (Route 2) จะใช้ req.body เพื่อรับข้อมูลของผู้ใช้ใหม่ที่จะสร้าง
    - ใน PUT (Route 5) จะใช้ req.body เพื่อรับข้อมูลใหม่ที่จะนำไปอัปเดตผู้ใช้เดิม

**ข้อ 8: แนวคิดเกี่ยวกับฐานข้อมูล (Database Concepts)**

**คำตอบ:**

|  |  |  |
| --- | --- | --- |
| คุณสมบัติ | SQL (เช่น MySQL) | NoSQL (เช่น MongoDB) |
| โครงสร้างข้อมูล (Structure) | แบบตาราง (Table) ที่มีแถว (Row) และคอลัมน์ (Column) | แบบเอกสาร (Document) คล้าย JSON ที่มี Key-Value Pairs |
| ความยืดหยุ่น (Flexibility) | ต่ำ (Strict Schema) ต้องกำหนดโครงสร้างตารางตายตัวก่อนบันทึกข้อมูล | สูง (Flexible/Dynamic Schema) แต่ละเอกสารใน Collection ไม่จำเป็นต้องมีโครงสร้างเหมือนกัน |
| การขยายระบบ (Scalability) | ขยายแบบแนวตั้ง (Vertical Scaling) คือการเพิ่มทรัพยากรให้เซิร์ฟเวอร์เครื่องเดิม (เช่น เพิ่ม RAM, CPU) | ขยายแบบแนวนอน (Horizontal Scaling) คือการเพิ่มจำนวนเซิร์ฟเวอร์เข้ามาในระบบ (Clustering) |

**คำอธิบายอย่างละเอียด:**

* **SQL** เหมาะสำหรับระบบที่ต้องการความถูกต้องของข้อมูลสูง (High Integrity) และข้อมูลมีความสัมพันธ์ที่ซับซ้อนและชัดเจน เช่น ระบบธนาคาร, ระบบลงทะเบียนเรียน
* **NoSQL** เหมาะสำหรับระบบที่ต้องการความยืดหยุ่นสูง, จัดการข้อมูลที่ไม่มีโครงสร้างแน่นอน, และต้องการรองรับข้อมูลปริมาณมหาศาลและการขยายตัวอย่างรวดเร็ว เช่น Social Media, ระบบ IoT

**ข้อ 9: การวิเคราะห์โครงสร้างข้อมูล (Mongoose)**

**คำตอบ:** Schema นี้จะสร้าง Document สำหรับ Product ใน MongoDB ที่มีลักษณะดังนี้:

* **productName**: เป็นฟิลด์ที่เก็บข้อมูลประเภท String, **ต้องมีข้อมูลเสมอ** (required: true), และค่าในฟิลด์นี้ของแต่ละ Document **ห้ามซ้ำกัน** ใน Collection (unique: true)
* **price**: เป็นฟิลด์ที่เก็บข้อมูลประเภท Number, **ต้องมีข้อมูลเสมอ** (required: true), แต่ถ้าหากไม่มีการส่งค่านี้มาตอนสร้างข้อมูล ค่าเริ่มต้นจะเป็น 0 (default: 0)
* **tags**: เป็นฟิลด์ที่เก็บข้อมูลประเภท Array โดยที่สมาชิกแต่ละตัวใน Array ต้องเป็น String เท่านั้น
* **onSale**: เป็นฟิลด์ที่เก็บข้อมูลประเภท Boolean (true หรือ false) และถ้าหากไม่มีการส่งค่านี้มาตอนสร้างข้อมูล ค่าเริ่มต้นจะเป็น false (default: false)

**ตัวอย่าง Document ที่ถูกต้องตาม Schema นี้:**

รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร

เนื้อหาที่สร้างโดย AI อาจไม่ถูกต้อง

*หมายเหตุ: Mongoose จะเพิ่มฟิลด์ \_id, createdAt, และ updatedAt ให้โดยอัตโนมัติ (หากไม่ได้ปิดการใช้งาน)*

**ข้อ 10: การเชื่อมต่อระบบ (Full-Stack Integration)**

**คำตอบ:** ขั้นตอนการเชื่อมต่อเพื่อดึงข้อมูลหนังสือทั้งหมดมาแสดงผล มีดังนี้:

**ฝั่ง Frontend (React):**

1. **การร้องขอข้อมูล:** เมื่อคอมโพเนนต์ที่ต้องการแสดงรายการหนังสือ (เช่น BookList) ถูก render ครั้งแรก, จะใช้ useEffect Hook เพื่อทำการยิง API request เพียงครั้งเดียว
2. **การส่ง Request:** ใช้ fetch API หรือไลบรารีอย่าง axios เพื่อส่ง GET request ไปยัง Endpoint ของ Backend (เช่น http://localhost:5000/api/books)
3. **การจัดการ State:** สร้าง State ด้วย useState Hook เพื่อเก็บข้อมูลหนังสือที่ได้รับกลับมา (เช่น const [books, setBooks] = useState([])) รวมถึง State สำหรับสถานะการโหลด (loading) และข้อผิดพลาด (error)
4. **การรับและอัปเดต State:** เมื่อได้รับ Response จาก Backend, ทำการแปลงข้อมูล JSON และเรียกใช้ setBooks() เพื่ออัปเดต State ซึ่งจะทำให้ React ทำการ re-render คอมโพเนนต์
5. **การแสดงผล:** ในส่วน JSX ของคอมโพเนนต์, ใช้เมธอด .map() กับ State books เพื่อวนลูปสร้างคอมโพเนนต์ย่อย (เช่น <BookCard />) สำหรับหนังสือแต่ละเล่ม

**ฝั่ง Backend (Express.js):**

6. **การรับ Request:** Express server ที่ทำงานอยู่ จะรับ GET request ที่เข้ามาที่ Route /api/books

7. **การประมวลผล:** Route Handler ที่ผูกไว้กับเส้นทางนี้จะถูกเรียกใช้งาน

8. **การติดต่อฐานข้อมูล:** ภายใน Handler, จะใช้ Mongoose Model (เช่น Book.find({})) เพื่อสั่งให้ Mongoose ไปค้นหาข้อมูลหนังสือ **ทั้งหมด** จาก Collection ในฐานข้อมูล MongoDB

9. **การส่ง Response:** เมื่อ Mongoose ได้รับข้อมูลจากฐานข้อมูลกลับมาแล้ว, Handler จะทำการส่งข้อมูลนั้นกลับไปยัง Frontend ในรูปแบบ JSON พร้อมกับ HTTP Status Code 200 OK