**เฉลยวิชา: Web Programming (ฉบับวิเคราะห์)**

**ข้อที่ 1: React Components & Props (วิเคราะห์โค้ด)**

**คำถาม 1:** Component App ส่งข้อมูลอะไรบ้างไปยัง UserCard อันแรก? และข้อมูลเหล่านั้นถูกส่งไปในรูปแบบใด?

**คำตอบ:** Component App ส่งข้อมูลทั้งหมดที่อยู่ในตัวแปร userA ไปยัง UserCard อันแรก ซึ่งประกอบด้วย:

* name: 'Somsri'
* role: 'Admin'
* isActive: true

ข้อมูลเหล่านี้ถูกรวมกันเป็น **Object** และถูกส่งผ่าน **Prop** ที่ชื่อว่า user ไปยัง Component UserCard ครับ

**คำถาม 2:** UserCard อันที่สองจะแสดงผลข้อความว่า "Status: Offline" หรือไม่? เพราะเหตุใด?

**คำตอบ:** **ใช่ครับ**, UserCard อันที่สองจะแสดงผลว่า "Status: Offline"

**เหตุผล:** ใน Component App, UserCard อันที่สองได้รับ Prop user ที่มีค่า isActive เป็น false ({..., isActive: false })

เมื่อ UserCard ได้รับข้อมูลนี้ ในส่วนของโค้ด JSX จะมีการใช้ Ternary Operator เพื่อตรวจสอบเงื่อนไข: {isActive ? 'Online' : 'Offline'}

เนื่องจาก isActive เป็น false เงื่อนไขจึงเป็นเท็จ ทำให้โค้ดเลือกแสดงผลค่าที่อยู่หลังเครื่องหมายโคลอน (:) ซึ่งก็คือข้อความ **"Offline"** นั่นเอง

**ข้อที่ 2: Node.js & Non-Blocking I/O (คิดวิเคราะห์)**

**คำตอบ:**

ผู้ใช้คนที่สอง**ไม่จำเป็นต้องรอ**ให้การอ่านไฟล์ของผู้ใช้คนแรกเสร็จสิ้นก่อน และจะได้รับการตอบกลับ (Response) เกือบจะในทันที นี่คือเหตุผลตามหลักการทำงานของ Event Loop และ Non-Blocking I/O ของ Node.js:

1. **Request ที่ 1 (อ่านไฟล์):** เมื่อ Request แรกเข้ามา, Event Loop ซึ่งทำงานบน Main Thread จะรับงานนี้ มันเห็นว่าเป็นงาน I/O (Input/Output) ที่ใช้เวลานาน แทนที่จะรอให้ไฟล์อ่านเสร็จ (ซึ่งจะ "Block" การทำงานทั้งหมด) Node.js จะ**มอบหมาย (Delegate)** งานอ่านไฟล์นี้ไปให้ **Worker Thread** ที่อยู่ใน Thread Pool (ซึ่งจัดการโดย OS) พร้อมกับลงทะเบียน **Callback Function** ว่า "เมื่องานนี้เสร็จแล้ว ให้เอางานนี้กลับมาต่อคิว"
2. **Main Thread ว่างทันที:** หลังจากมอบหมายงานอ่านไฟล์ไปแล้ว Main Thread ของ Node.js ก็จะว่างทันทีและพร้อมรับงานใหม่
3. **Request ที่ 2 (คำนวณ):** เมื่อ Request ที่สองเข้ามา Event Loop จะรับงานนี้ไปทำทันที เนื่องจากเป็นการคำนวณทางคณิตศาสตร์ง่ายๆ ซึ่งเป็นงานที่ทำบน CPU และเสร็จเร็วมาก (10ms) Main Thread จึงประมวลผลงานนี้จนเสร็จและส่ง Response กลับไปให้ผู้ใช้คนที่สองได้เลย
4. **สิ้นสุด Request ที่ 1:** หลังจากผ่านไป 5 วินาที Worker Thread อ่านไฟล์เสร็จเรียบร้อย มันจะนำผลลัพธ์ที่ได้และ Callback Function ที่ลงทะเบียนไว้ตอนแรก ไปใส่ไว้ใน **Event Queue (คิวของงานที่เสร็จแล้ว)**
5. **Event Loop ทำงานต่อ:** เมื่อ Event Loop ทำงานครบรอบและพบว่า Main Thread ว่าง มันจะไปดึงงานจาก Event Queue มาทำ ซึ่งก็คือ Callback Function ของ Request แรกนั่นเอง จากนั้นจึงประมวลผลและส่ง Response กลับไปให้ผู้ใช้คนแรก

**สรุป:** ด้วยสถาปัตยกรรม Non-Blocking I/O ทำให้ Node.js สามารถจัดการ Request จำนวนมากพร้อมกันได้อย่างมีประสิทธิภาพโดยใช้เพียง Single Thread เพราะมันไม่เคยรอให้งาน I/O ที่ช้าเสร็จ แต่จะใช้เวลานั้นไปทำงานอื่นที่รออยู่ก่อน ทำให้ผู้ใช้คนที่สองได้รับประสบการณ์การใช้งานที่รวดเร็ว โดยไม่ถูกกระทบจาก Request ที่ช้าของผู้ใช้คนอื่น

**ข้อที่ 3: RESTful API & HTTP Methods (วิเคราะห์หลักการ)**

**คำตอบ:**

Endpoints ทั้ง 3 ข้อขัดกับหลักการออกแบบ RESTful API ดังนี้:

1. **POST /getAllProducts**
   * **ข้อผิดพลาด:**
     + **ใช้ HTTP Method ผิด:** การ "ดึงข้อมูล" (Get) ควรใช้ GET ไม่ใช่ POST
     + **URL มีคำกริยา:** URL ควรเป็น "คำนาม" ที่สื่อถึงทรัพยากร (products) ไม่ใช่คำกริยาที่บอกการกระทำ (getAllProducts)
   * **วิธีแก้ไขที่ถูกต้อง:** GET /products
2. **GET /product/delete?id=123**
   * **ข้อผิดพลาด:**
     + **ใช้ HTTP Method ผิด:** GET เป็น Method ที่ปลอดภัย (Safe Method) หมายความว่าไม่ควรทำให้เกิดการเปลี่ยนแปลงข้อมูลบนเซิร์ฟเวอร์ การ "ลบข้อมูล" ควรใช้ DELETE
     + **ระบุ ID ใน Query Parameter:** การระบุทรัพยากรที่ต้องการจัดการโดยตรง ควรทำผ่าน Path ของ URL ไม่ใช่ Query Parameter
     + **ใช้ชื่อทรัพยากรเป็นเอกพจน์:** ชื่อทรัพยากรควรเป็นพหูพจน์ (products) เพื่อสื่อถึงคอลเลกชัน
   * **วิธีแก้ไขที่ถูกต้อง:** DELETE /products/123
3. **POST /updateProduct**
   * **ข้อผิดพลาด:**
     + **ใช้ HTTP Method ผิด:** การ "อัปเดต" ข้อมูลที่มีอยู่แล้ว ควรใช้ PUT (แก้ไขทั้งหมด) หรือ PATCH (แก้ไขบางส่วน) ไม่ใช่ POST ซึ่งใช้สำหรับ "สร้าง" ข้อมูลใหม่
     + **URL มีคำกริยา:** URL ไม่ควรมีคำกริยา (updateProduct)
     + **ไม่ได้ระบุ ID ใน URL:** ไม่สามารถรู้ได้ว่าจะอัปเดตสินค้าชิ้นไหนหากไม่ระบุ ID ใน Path
   * **วิธีแก้ไขที่ถูกต้อง:** PUT /products/123 (โดยส่งข้อมูลแก้ไขไปใน Body)

**ข้อที่ 4: Database & Mongoose (คิดวิเคราะห์)**

**คำตอบ:**

**Schema** และ **Model** เป็นสองแนวคิดหลักใน Mongoose ที่ทำงานร่วมกันเพื่อเชื่อมต่อโค้ดของเรากับฐานข้อมูล MongoDB แต่มีหน้าที่แตกต่างกันอย่างชัดเจน

* **Schema (พิมพ์เขียว):**
  + **หน้าที่:** Schema คือ **"โครงสร้าง" หรือ "พิมพ์เขียว"** ของข้อมูลใน Collection มันทำหน้าที่กำหนดว่า Document แต่ละอันควรมีหน้าตาเป็นอย่างไร โดยจะระบุถึงชื่อฟิลด์ (Field), ชนิดข้อมูล (Data Type เช่น String, Number), ค่าเริ่มต้น (Default Value), และกฎเกณฑ์ต่างๆ (Validation Rules เช่น required: true)
  + **การทำงาน:** Schema **ไม่ได้ติดต่อกับฐานข้อมูลโดยตรง** มันเป็นเพียง Object ในฝั่ง JavaScript ที่ใช้อธิบายโครงสร้างข้อมูลเท่านั้น
* **Model (ตัวแทนดำเนินการ):**
  + **หน้าที่:** Model คือ **"ตัวกลาง" หรือ "อินเทอร์เฟซ"** ที่เราใช้ในการโต้ตอบกับ Collection ในฐานข้อมูลจริงๆ มันถูกสร้างขึ้นมาจาก Schema และมอบฟังก์ชันต่างๆ สำหรับการดำเนินการกับข้อมูล (CRUD Operations) เช่น การสร้าง, ค้นหา, อัปเดต, และลบข้อมูล
  + **การทำงาน:** เวลาเราต้องการบันทึกหรือค้นหาข้อมูล เราจะสั่งงานผ่าน Model เช่น Product.find() หรือ newProduct.save() แล้ว Model จะนำคำสั่งนั้นไปแปลงเป็นภาษาที่ MongoDB เข้าใจและส่งไปทำงานที่ฐานข้อมูล

**ความสำคัญของการกำหนด Schema ที่ดี:** การกำหนด Schema ที่ดีมีความสำคัญอย่างยิ่งเพราะเป็นปราการด่านแรกที่ช่วยรับประกัน **ความถูกต้องและความน่าเชื่อถือของข้อมูล (Data Integrity)**

1. **สร้างความสอดคล้องกัน (Consistency):** ทำให้มั่นใจได้ว่าข้อมูลทั้งหมดที่ถูกบันทึกลงใน Collection จะมีโครงสร้างพื้นฐานเหมือนกัน ป้องกันการบันทึกข้อมูลที่สะเปะสะปะ
2. **การตรวจสอบความถูกต้อง (Validation):** สามารถบังคับใช้กฎเกณฑ์ต่างๆ เช่น ฟิลด์นี้ห้ามว่าง (required), อีเมลต้องอยู่ในรูปแบบที่ถูกต้อง ซึ่งช่วยกรองข้อมูลที่ผิดพลาดออกไปตั้งแต่ระดับแอปพลิเคชัน ก่อนที่ข้อมูลจะไปถึงฐานข้อมูล
3. **ทำให้โค้ดคาดเดาได้ง่าย (Predictability):** นักพัฒนาที่มาทำงานต่อจะเข้าใจได้ทันทีว่าข้อมูลใน Collection นี้มีหน้าตาเป็นอย่างไร ทำให้การเขียนโค้ดเพื่อจัดการข้อมูลนั้นง่ายขึ้นและลดข้อผิดพลาด

**ข้อที่ 5: Full-Stack Data Flow (คิดวิเคราะห์)**

**คำตอบ:**

การเดินทางของข้อมูลตั้งแต่ผู้ใช้คลิกปุ่ม "บันทึก" ใน React จนถึงข้อมูลถูกจัดเก็บใน MongoDB มีลำดับขั้นตอนดังนี้:

**1. Frontend (React) - การโต้ตอบและส่งคำขอ:**

* **Event Trigger:** ผู้ใช้กรอกข้อมูลในฟอร์ม แล้วคลิกปุ่ม "บันทึก" การคลิกนี้จะไปกระตุ้นฟังก์ชัน handleSubmit ที่ผูกไว้กับฟอร์ม
* **Data Packaging:** ฟังก์ชัน handleSubmit รวบรวมข้อมูลจาก State ของ React component (เช่น productName, price) มาสร้างเป็น JavaScript object
* **API Call:** ใช้เครื่องมือเช่น fetch หรือ axios เพื่อส่ง HTTP Request แบบ POST ไปยัง Backend API Endpoint ที่กำหนดไว้ (เช่น http://localhost:5000/api/products) โดยแนบ object ข้อมูลสินค้าไปใน **Body** ของ Request ในรูปแบบ JSON
* **UI Update:** React อาจจะแสดงสถานะกำลังโหลด (Loading Spinner) เพื่อรอการตอบกลับจากเซิร์ฟเวอร์

**2. Backend (Express) - การรับและประมวลผลคำขอ:**

* **Routing:** Express Server รับ Request ที่เข้ามา และระบบ Router จะตรวจสอบเส้นทาง (/api/products) และ Method (POST) แล้วส่งต่อไปยัง Controller ที่รับผิดชอบ
* **Middleware:** Middleware express.json() ทำการแปลงข้อมูล JSON ใน Body ของ Request ให้กลายเป็น JavaScript object ที่สามารถเข้าถึงได้ผ่าน req.body
* **Controller Logic:** Controller ดึงข้อมูลสินค้าจาก req.body มาใช้งาน

**3. Backend (Mongoose) & Database (MongoDB) - การบันทึกข้อมูล:**

* **Model Interaction:** Controller ใช้ Mongoose Model ที่สร้างไว้ (เช่น Product) เพื่อสร้าง Instance ของเอกสารใหม่: const newProduct = new Product({ name: req.body.name, price: req.body.price });
* **Database Command:** Controller เรียกใช้เมธอด .save() บน Instance นั้น (await newProduct.save();)
* **Data Persistence:** Mongoose จะแปลงคำสั่ง .save() เป็นภาษาที่ MongoDB เข้าใจ แล้วส่งไปที่เซิร์ฟเวอร์ MongoDB ซึ่งจะทำการบันทึกเอกสารใหม่นี้ลงใน Collection products พร้อมทั้งสร้าง \_id ที่ไม่ซ้ำกันให้โดยอัตโนมัติ

**4. การส่งข้อมูลกลับ (Response Flow):**

* **Database to Backend:** เมื่อบันทึกสำเร็จ MongoDB จะส่งข้อมูลของเอกสารที่เพิ่งสร้าง (รวมถึง \_id ใหม่) กลับมาให้ Mongoose
* **Backend to Frontend:** Controller ของ Express ได้รับข้อมูลยืนยันจาก Mongoose แล้วจึงส่ง HTTP Response กลับไปให้ React โดยมักจะมี **Status Code 201 Created** และแนบข้อมูลของสินค้าที่เพิ่งสร้างเสร็จสมบูรณ์ไปใน Body ของ Response ด้วย
* **Frontend UI Update:** React ได้รับ Response สำเร็จ ก็จะนำข้อมูลสินค้าใหม่ที่ได้กลับมาไปอัปเดต State ของแอปพลิเคชัน อาจจะแสดงข้อความ "บันทึกสำเร็จ", ล้างข้อมูลในฟอร์ม, และเพิ่มสินค้าใหม่ลงในรายการที่แสดงบนหน้าจอ โดยที่ผู้ใช้ไม่ต้องรีเฟรชหน้าเว็บเลย