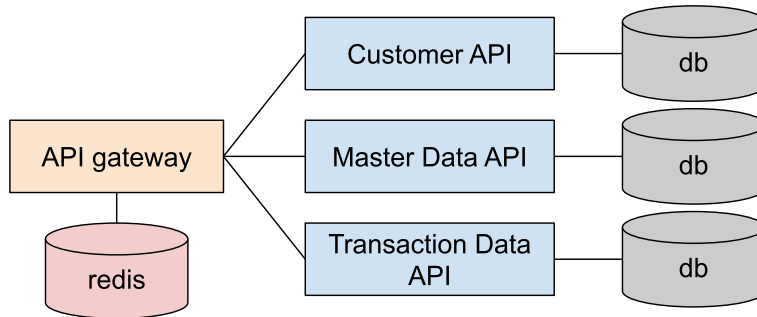


Back-end Questions

1. Assuming the system currently has three microservices: Customer API, Master Data API, and Transaction Data API, there is a new feature that requires data from all three microservices to be displayed in near real-time. The current technology stack includes REST APIs and an RDBMS database. How would you design a new API for this feature?

Answer

1. สร้าง API gateway เป็นตัวกลางในการเรียก data จาก API ทั้ง 3 เส้น โดยการ mapping data



2. โดยในการยิง API 3 เส้น สามารถใช้ Promise ช่วยได้ โดยการทำงานแบบ concurrently จะทำให้ได้ response เร็วขึ้น เช่น

```
const [customer, masterData, transaction] = await Promise.all([
  getCustomerAPI(),
  getMasterDataAPI(),
  getTransactionDataAPI()
]);
```

3. กรณีมีบาง service down หากต้องการ return data ของ service เฉพาะเท่าที่มี สามารถเปลี่ยนเป็น Promise.allSettled() แล้ว filter เฉพาะ Promise ที่ fulfilled

```
const data = await Promise.allSettled([
  getCustomerAPI(),
  getMasterDataAPI(),
  getTransactionDataAPI()
]).filter(
  promise => promise.status === 'fulfilled',
)
```

ก็จะสามารถ return response เท่าที่มีได้

```
{
  "customer": {...},
  "masterData": {...},
  "transaction": null,
}
```

4. หาก data ที่จะต้อง return ของ API gateway หรือ service ย่อยๆ เป็น static data หรือไม่ได้มีการเปลี่ยนแปลงบ่อย อาจเก็บ cache ใน redis เพื่อเพิ่ม performance ก็ได้

2. Assuming the team has started planning a new project, the project manager asks you for a performance test strategy plan for this release. How would you recommend proceeding to the project manager?

Answer

1. การวางแผน performance test ควรเริ่มจากการกำหนดจุดประสงค์ของการ test ก่อนว่าทำไปทำไม เช่น test เพื่อหา resource ที่เพียงพอต่อการใช้งาน, test ว่าระบบ scale ได้ไม่มีปัญหา, test ว่าระบบรับข้อมูลปริมาณมากๆ ได้, หา maximum data size ที่ระบบยังทำงานได้
2. จากนั้นพิจารณา usecase ของระบบ เช่น
 - 2.1. จำนวน user และ role ที่เข้ามาใช้ระบบ เช่น customer 1000 คน, admin 20 คน
 - 2.2. ช่วงเวลาที่ใช้งาน เช่น ระบบใช้งาน 24 ชั่วโมง โดย peak ช่วง 8.00 - 16.00
 - 2.3. use case ที่ user ใช้งานบ่อยๆ เช่น customer จอกรับในระบบ, ดูข้อมูลของตัวเอง, admin จัดการเอกสาร upload รูป, file ต่างๆ
3. กำหนด environment ให้ใกล้เคียง production ที่สุด ทั้ง resource CPU, memory, ขนาด database รวมถึง data ที่จะใช้ test ทั้งขนาดและจำนวน
4. กำหนด criteria ของการ test เช่น API response time P95 < 1 second
5. แนะนำให้ใช้ K6 เป็น tools ในการ test เพราะสามารถเขียน automated test เป็น JavaScript ได้ และสามารถจำลอง scenario ได้หลายแบบ เช่น
 - 5.1. Load Test - test ว่ารองรับปริมาณ traffic ที่กำหนดได้, scale ได้ทัน
 - 5.2. Stress Test - test ว่ารองรับปริมาณ traffic ที่สูงขึ้นเรื่อยๆ ได้
 - 5.3. Spike Test - test ว่ารองรับปริมาณ traffic ที่สูงขึ้นแบบทันทีใด ในช่วงเวลาสั้นๆ ได้
 - 5.4. Breakpoint Test - test เพิ่ม traffic ไปเรื่อยๆ เพื่อหาจุดสูงสุดที่ระบบรับได้

3. Design and develop two APIs using NestJS and Postgres with the following specifications:
1. Create a Multilingual Product API: Develop an API that allows for the creation of products, each with attributes for name and description that support multiple languages.

Answer

สร้าง API POST /products โดยรับ body เป็น array ของ productInfos มี language (required), name (required), description (optional)

The screenshot shows a REST client interface with a POST request to `localhost:8000/products`. The request body is a JSON array of product information objects in Thai, English, and Japanese. The response is a 201 Created status with a JSON object containing a success message and the created products with assigned IDs.

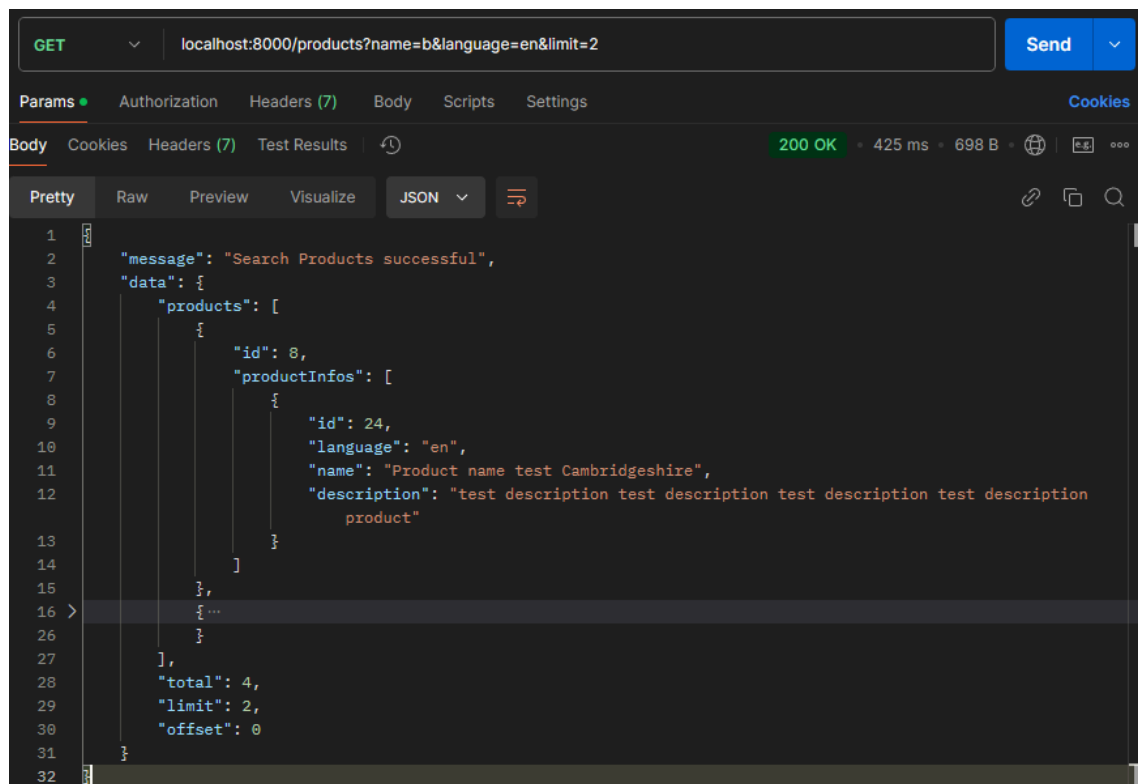
```
1 {
2   "productInfos": [
3     {
4       "language": " th",
5       "name": "ทดสอบชื่อสินค้า 1 ",
6       "description": "ทดสอบคำอธิบาย
7       ทดสอบคำอธิบาย ทดสอบคำอธิบาย
8       ทดสอบคำอธิบาย 1"
9     },
10    {
11      "language": " EN ",
12      "name": " Product name test 1",
13      "description": " test description
14      test description test
15      description test description 1"
16    },
17    {
18      "language": "jp",
19      "name": " おはようございます 1",
20      "description": "おはようございます おはよ
21      うございます おはようございます おはよ
22      うございます 1"
23    }
24  ]
25 }
26 }
```

```
1 {
2   "message": "Create Products successful",
3   "data": {
4     "productInfos": [
5       {
6         "language": "th",
7         "name": "ทดสอบชื่อสินค้า 1",
8         "description": "ทดสอบคำอธิบาย
9         ทดสอบคำอธิบาย ทดสอบคำอธิบาย
10        ทดสอบคำอธิบาย 1",
11        "id": 16
12      },
13      {
14        "language": "en",
15        "name": "Product name test 1",
16        "description": "test description
17        test description test
18        description test description
19        1",
20        "id": 17
21      },
22      {
23        "language": "jp",
24        "name": "おはようございます 1",
25        "description": "おはようございます
26        おはようございます おはようござい
27        ます おはようございます 1",
28        "id": 18
29      }
30    ],
31     "id": 6
32   }
33 }
```

2. Multilingual Product Search API: Implement an API that enables searching for products by name in any language and returns results in a paginated format.

Answer

สร้าง API GET /products โดยรับ query เป็น language, name, description (partial search), limit, offset โดยมีค่า default limit = 10, offset = 0



```
GET localhost:8000/products?name=b&language=en&limit=2
Send

Params Authorization Headers (7) Body Scripts Settings Cookies
Body Cookies Headers (7) Test Results 200 OK • 425 ms • 698 B
Pretty Raw Preview Visualize JSON

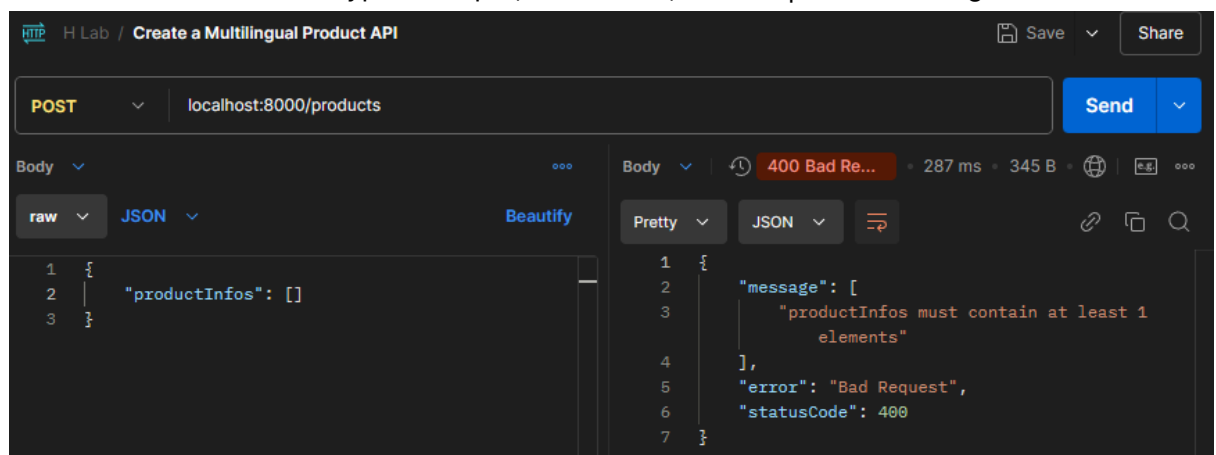
1  {
2    "message": "Search Products successful",
3    "data": {
4      "products": [
5        {
6          "id": 8,
7          "productInfos": [
8            {
9              "id": 24,
10             "language": "en",
11             "name": "Product name test Cambridgeshire",
12             "description": "test description test description test description test description product"
13           }
14         ]
15       },
16       { ... }
17     ],
18     "total": 4,
19     "limit": 2,
20     "offset": 0
21   }
22 }
```

Additional Requirements:

1. Validation: Outline how you will validate data inputs in both APIs to ensure data integrity.

Answer

ใช้ class-validator validate type ของ input, กำหนด max, min ต่างๆ และ trim string ก่อน

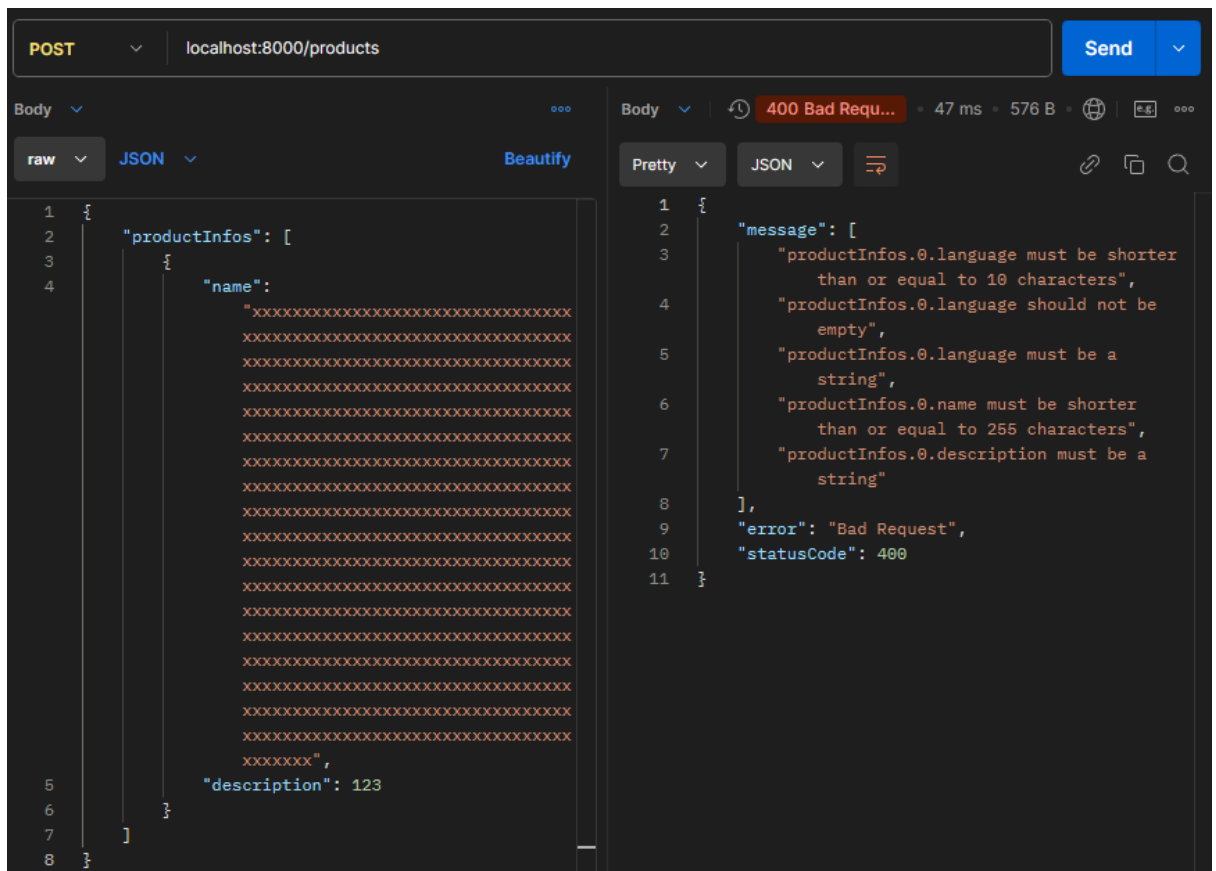


```
POST localhost:8000/products
Send

Body raw JSON Beautify
1 {
2   "productInfos": []
3 }

Body 400 Bad Re... • 287 ms • 345 B
Pretty JSON

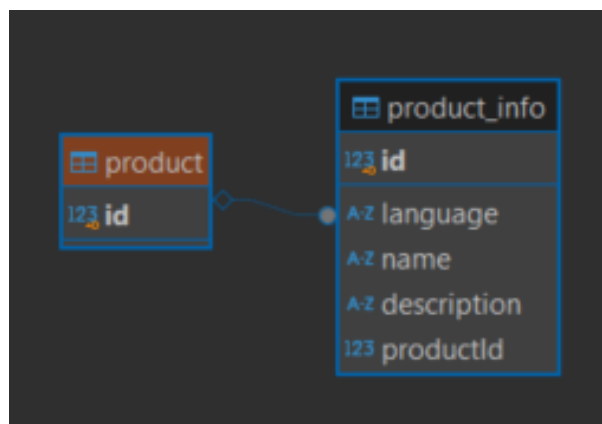
1 {
2   "message": [
3     "productInfos must contain at least 1 elements"
4   ],
5   "error": "Bad Request",
6   "statusCode": 400
7 }
```



2. Database Design: Describe the database schema and the approach you will use to handle multilingual support for product information.

Answer

สร้าง 2 table product, product_info relate กันแบบ 1-many โดยแต่ละ product_info คือ ข้อมูลของ product ในแต่ละภาษา



3. Testing Strategy: Explain your strategy for testing these APIs, including how you will handle unit tests, integration tests, and any end-to-end testing considerations.

Answer

เขียน unit test ของ Controller, Service โดยใช้ lib testing ของ nestjs เขียนทั้ง happy case และ unhappy case

```
PASS src/modules/products/products.service.spec.ts (6.599 s)
PASS src/modules/products/products.controller.spec.ts (8.811 s)

Test Suites: 2 passed, 2 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        9.4 s, estimated 10 s
Ran all test suites.
```

React Questions

1. useCallback ใช้ทำอะไร

Answer

useCallback ใช้เพื่อป้องกันการสร้าง function ใหม่ทุกครั้งที่มีการ rerender เนื่อง state ของ component เปลี่ยนแปลง เช่น เมื่อ App ถูกสร้าง function handleSetCount() จะถูกสร้างเพียงครั้งเดียว โดยที่ count=0 เมื่อกดปุ่ม "+" state ของ count จะถูกอัปเดต โดย function handleSetCount() จะยังคงเหมือนเดิม

useCallback คล้ายกับ useMemo แต่เป็นการเก็บ cache ของ function แทน data ซึ่งจะใช้ในกรณีที่ส่ง function ผ่าน props ไปให้ component ลูก

```
export function App(props) {
  const [count, setCount] = useState(0);
  const handleSetCount = useCallback(() => {
    console.log(count)
    setCount((prev) => prev + 1)
  }, [])

  return (
    <div>
      <div>
        count: {count}
      </div>
      <div>
        <button onClick={handleSetCount}>+</button>
      </div>
    </div>
  );
}
```

2. Write a unit test for the UserProfile React component using Jest and React Testing Library.

Answer

```
import React from "react";
import { render, screen } from "@testing-library/react";
import "@testing-library/jest-dom";
import UserProfile from "../UserProfile";

global.fetch = jest.fn();

describe("UserProfile", () => {
  afterEach(() => {
    jest.clearAllMocks();
  });

  test("case loading", () => {
    render(<UserProfile userId={123} />);
    expect(screen.getByText("Loading...")).toBeInTheDocument();
  });

  test("case fetch success", async () => {
    const mockUserData = { name: "test name", email: "test email" };
    fetch.mockResolvedValueOnce({
      ok: true,
      json: async () => mockUserData,
    });

    render(<UserProfile userId={123} />);

    expect(await screen.findByText("test name")).toBeInTheDocument();
    expect(await screen.findByText("Email: test email")).toBeInTheDocument();
  });

  test("case error", async () => {
    fetch.mockResolvedValueOnce({
      ok: false,
    });

    render(<UserProfile userId={123} />);

    expect(
      await screen.findByText("Error: Failed to fetch user data")
    ).toBeInTheDocument();
  });
});
```


PASS src/component/**UserProfile.test.js**

UserProfile

- ✓ case loading (129 ms)
- ✓ case fetch success (18 ms)
- ✓ case error (14 ms)

Test Suites: 1 **passed**, 1 total

Tests: 3 **passed**, 3 total

Snapshots: 0 total

Time: 3.312 s

Ran all test suites related to changed files.