261207 Computer Engineering Lab

Lab 08 : Express (basic API) (20 pts)

ให้นักศึกษาสร้าง API Todo อย่างง่าย โดยใช้ความรู้เรื่อง Express + TypeScript โดยที่รายการของ Todo (Task) แต่ละรายการ จะมีโครงสร้างดังนี้

Key	Type	Detail
id	number	ตัวเลขที่ใช้อ้างอิงรายการ โดยที่ id ของแต่ละรายการจะมีค่าไม่เท่ากัน
name	string	ชื่อของรายการ
complete	boolean	สถานะขีดฆ่าของรายการ (true กับ false)

โดย API ที่ต้องการให้สร้าง จะมีความต้องการดังนี้

1. GET /me (1 pts)

เป็น route ที่ใช้แสดงข้อมูลของน.ศ.ออกมา โดยจะแสดงชื่อ-นามสกุล และรหัสนักศึกษาตามตัวอย่าง โดยที่ทั้ง name และ code จะมีค่าเป็น string

Method	Route	Status code	Response

GET	/me	200 OK	{
			“name”: “Chayanin Suatap”,
			“code”: “610631100”
			}

2. POST /todo (4 pts)

ใช้เพิ่ม task ใหม่เข้าไปในระบบ โดยจะมีรูปแบบของ request body ตามในตาราง และจะมีการส่ง **รายการ task ทั้งหมดที่อัปเดตแล้ว**กลับมาด้วย

2.1. กรณีทำงานได้ถูกต้อง (2 pts)

ใน request body จะมี name เป็น string และ complete เป็น boolean โดยทุกครั้งที่มีการเพิ่ม task ใหม่ ๆ ค่า id ของ task จะมีค่าเพิ่มขึ้นเสมอ

Method	Route	Body	Response

POST	/todo	{
		“name”:
		“Have breakfast”,
		“complete” : true
		}
			Status code : 200 OK
			Response body :
			{
			“status”: “success”,
			“tasks”: [
			{
			“id”: 1,
			“name”: “Have breakfast”,
			“complete”: true}
			]
			}
			(ยกตัวอย่าง เมื่อเริ่มต้นยังไม่มี task ใด ๆ ในระบบเลย)

2.2. <u>กรณีทำงานไม่ถูกต้อง (2 pts)</u>

จะเกิดขึ้นเมื่อมีการแนบ request body ไม่ถูกต้อง ซึ่งมี 3 กรณี ได้แก่

1. name ไม่ใช่ string
2. name เป็น string ที่ว่างเปล่า (คือ “”)
3. complete ไม่ใช่ boolean

Method	Route	Body	Response

POST	/todo	{
		“name”: false,
		“complete” : 1234
		}
			Status code : 400 Bad request
			Response body :
			{
			“status”: “failed”,
			“message”: “Invalid input data”
			}

Hints: น.ศ.อาจใช้ฟังก์ชัน typeof ในการเช็ค type ของตัวแปรได้ เช่น typeof(x) === “string”

3. GET /todo (5 pts)

Route นี้ใช้สำหรับแสดงข้อมูล task ที่อยู่ในระบบทั้งหมด

3.1. กรณีไม่มี query parameters (2 pts)

ให้แสดง task ทั้งหมดตามลำดับจาก task ที่ถูกเพิ่มเข้ามาก่อนไปจนถึง task ที่ถูกเพิ่มเข้าไปเป็นรายการสุดท้าย

Method	Route	Response

GET	/todo	**Status code : 200 OK**
		Response body :
		{
		“status”: “success”,
		“tasks”: [
		{“id”: 1, “name”: “Have breakfast”, “complete”: false},
		{“id”: 2, “name”: “Take a shower”, “complete”: false},
		{“id”: 3, “name”: “Read a book”, “complete”: true}
		]
		}

3.2. กรณีมี query parameters (3 pts)

Query parameters ที่เป็นไปได้ ได้แก่ order=asc (ascending) และ order=desc (descending) โดยให้แสดง task ทั้งหมดเรียงจากตัวอักษร A – Z หากใช้ order=asc แต่ให้แสดง task ทั้งหมดเรียงลำดับจากตัวอักษร Z – A หากใช้ order=desc

Method	Route	Response

GET	/todo?order=asc	**Status code : 200 OK**
		Response body :
		{
		“status”: “success”,
		“tasks”: [
		{“id”: 1, “name”: “Have breakfast”, “complete”: false},
		{“id”: 3, “name”: “Read a book”, “complete”: true},
		{“id”: 2, “name”: “Take a shower”, “complete”: false}
		]
		}

	/todo?order=desc	**Status code : 200 OK**
		Response body :
		{
		“status”: “success”,
		“tasks”: [
		{“id”: 2, “name”: “Take a shower”, “complete”: false},
		{“id”: 3, “name”: “Read a book”, “complete”: true},
		{“id”: 1, “name”: “Have breakfast”, “complete”: false}
		]
		}

Hints :

1. ศึกษาการรับ request แบบ query parameter ได้ใน repo ของ lab
2. Array ของ JavaScript มีฟังก์ชัน sort ให้เรียกใช้ ซึ่งสามารถใช้จัดเรียงข้อมูลได้

   ศึกษาได้ที่นี่ : Sort array by firstname (alphabetically) in Javascript - Stack Overflow

4. PUT /todo/:id (4 pts)

ใช้สลับสถานะ complete ของ task ที่มีอยู่แล้ว (จาก true เป็น false หรือจาก false เป็น true) โดยจะอ้างอิงด้วย id ซึ่งแนบมากับ route URL

4.1. กรณีที่ทำงานถูกต้อง (2 pts)

จะส่ง task ที่อัปเดตแล้วกลับมาใน response body ด้วย

Method	Route	Response

PUT	/todo/1	**Status code : 200 OK**
		Response body :
		{
		“status”: “success”,
		“task”: {
		“id”: 1,
		“name”: “Have breakfast”, “complete”: true}
		}

4.2. กรณีที่ทำงานไม่ถูกต้อง (หา id ไม่เจอ) (2 pts)

Method	Route	Response

PUT	/todo/999	**Status code : 404 Not Found**
		Response body :
		{
		“status”: “failed”,
		“message”: “Id is not found”
		}

5. <u>DELETE /todo/:id (2 pts)</u>

ใช้ลบ task โดยอ้างอิงจาก id ที่แนบมาใน route URL

5.1. กรณีที่ทำงานถูกต้อง (1 pts)

แสดงรายการ task ทั้งหมดที่อัปเดตแล้วใน response หลังการจากลบ

Method	Route	Response
DELETE	/todo/1	**Status code : 200 OK** **Response body :** {“status”: “success”, “tasks”: [{“id”: 2, “name”: “Take a shower”, “complete”: false}, {“id”: 3, “name”: “Read a book”, “complete”: true}] }

5.2. กรณีที่ทำงานไม่ถูกต้อง (หา id ไม่เจอ) (1 pts)

Method	Route	Response

DELETE	/todo/999	**Status code : 404 Not Found**
		Response body :
		{
		“status”: “failed”,
		“message”: “Id is not found”
		}

6. การ Deploy (4 pts)

6.1. เมื่อเขียนโปรแกรมเสร็จแล้วให้ push code ตามปกติ

6.2. Deploy โดยใช้บริการของ Heroku ให้ตั้งชื่อ app เป็น **lab08-รหัสนักศึกษา** ดูวิธีการทำได้ที่นี่ Deploying Express App

6.3. กอปปี้ URL ที่ deploy ได้แปะลงในไฟล์ readme.md ของ git repository (4 pts)

หมายเหตุ :

- มี app ที่เสร็จสมบูรณ์แล้ว Deploy อยู่ที่ https://lab-08-finished-261207.herokuapp.com/ น.ศ. ทดลองส่ง request ไปตาม route ต่าง ๆ ได้เพื่อทดสอบผลลัพธ์
- การตั้งชื่อ route หากตั้งชื่อซ้ำกัน แต่ว่ามี HTTP method แตกต่างกัน ก็สามารถตั้งชื่อให้ซ้ำกันได้ เช่น route “/todo” ต่าง ๆ ในแลบนี้ (ซึ่งเป็นการตั้งชื่อตามมาตรฐาน ผู้ที่เรียกใช้ API ก็จะคาดเดาการทำงานของ route ต่าง ๆ จาก HTTP method เช่น GET /todo ก็จะใช้สำหรับเรียกดูข้อมูล todo ทั้งหมด หรือ POST /todo ก็จะใช้สำหรับเพิ่มข้อมูล todo ใหม่ เป็นต้น)
- การตรวจงาน จะใช้โปรแกรมตรวจอัตโนมัติ เพราะฉะนั้น น.ศ.ควรกำหนด response ของ API ต่าง ๆ ให้ถูกต้อง เช่น “status” : “success” ก็ควรใช้ตามที่โจทย์กำหนด ไม่ควรใช้ “status” : “Alright” เป็นต้น