

Brief of Assignment 3

| | |
|--------------------------|---|
| Academic Year | 2/2567 |
| Assignment title | Simple API with Scala (Individual Work) |
| Assessor(s) | Dr. Issarapong Khuankrue and/or Teaching Assistant(s) |
| Issue Date | March 4, 2025 |
| Submission Date | March 11, 2025 ก่อน 23.59 |
| Score | 10 Points |
| Course Learning Outcomes | CLO5 : สามารถเลือกใช้การเขียนโปรแกรมเชิงฟังก์ชันตามโจทย์ที่กำหนดได้ |
| Student ID/ Name | 66070286 ปณัสยา บุญประกอบ |

Assignment Brief and Guidance:

Submission Format:

- ส่งไฟล์นี้ โดยตั้งชื่อเป็น student_id.docx
- ส่งภาพที่ export จาก miro เป็น task1_student_id.pdf
- ส่ง source code เป็น task2_student_id.zip
- ส่ง ไฟล์ data.txt เพื่อประกอบการทดสอบ code

Task 1: ออกแบบ API โดยใช้ข้อมูลใกล้ตัวนักศึกษา (ทำเป็นกลุ่มได้) โดยแสดงผลลัพธ์ที่นี้ (5คะแนน)

โดยต้องแสดง

- บริบท (Context) ของข้อมูล
- Methods
- Resources
- Requests / Responses

เกณฑ์ประเมินสำหรับ Task 1

| | >80% | 60 – 40 % | 20 - 40% | < 20% |
|--------------------|--|---|---|-------------------------|
| Design API | สามารถออกแบบตามโจทย์ที่กำหนดได้ครบถ้วน เหมาะสม | สามารถออกแบบตามโจทย์ที่กำหนดได้ แต่ยังขาดความสมบูรณ์ตามที่โจทย์ | สามารถออกแบบตามโจทย์ที่กำหนดได้ บางส่วน | ไม่สามารถออกแบบ API ได้ |
| ได้รับคะแนน | 5 | 3-4 | 1-2 | 0 |

วางแผนผลลัพธ์ที่

| API Bounded context | | | |
|---|--|--|--|
| Context: ข้อมูลเสื้อผ้ามงคล | | | |
| Resources ข้อมูลเสื้อผ้ามงคล https://www.wongnai.com/articles/lucky-color-shirt { id INT, day TEXT, colors TEXT, description TEXT } | Parameters: GET: /days/:id และ /days POST:/days PUT: /days/:id DELETE: /days/:id | Requests GET: /days GET: /days/:id POST: /days body: { id, color, description } DELETE: /days/:id | Responses: GET: /days/:id { "id": 1, "day": "monday", "color": "gold", "description": "lucky" } GET: /days { "id": 1, "day": "monday", "color": "yellow", "description": "money" } |
| Methods GET POST PUT DELETE | Notes: | | |

อินทรา รัชประภัสร์ 66070228
ชนิสรา จันทร์คำจาร 66070248
ปณิสยา บุญประกอบ 66070286

Task 2: สร้าง API โดยพิจารณาตัวอย่างดังต่อไปนี้ (5คะแนน)

Example : AkkaHttp

1. สร้างโปรเจกต์ใหม่ เลือก template

```
> sbt new akka/akka-http-quickstart-scala.g8
```

2. ตั้งชื่อโปรเจกต์ใหม่ แต่ไม่จำเป็นต้องกำหนด version ต่างๆ ของ library

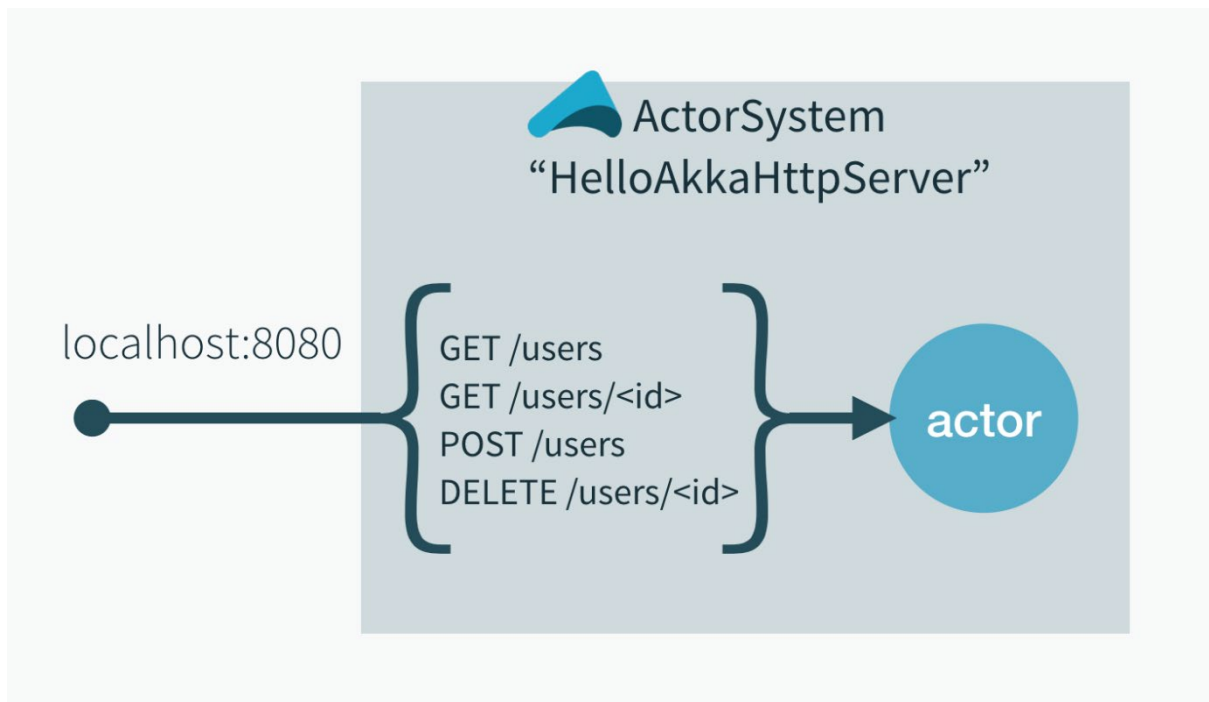
3. เรียกคำสั่ง

```
> sbt.bat
```

4. ภายใต sbt prompt เรียกคำสั่ง

```
> reStart  
// note เรียก reStop เมื่อต้องการหยุด
```

ตัวอย่างนี้จะแสดง



ใช้ postman เรียกดู localhost:3000/users

และ ลอง POST ที่ละบรรทัดข้อมูลดังนี้

```
{ "name": "MrX", "age": 31, "countryOfResidence": "Canada" }
{ "name": "Anonymous", "age": 55, "countryOfResidence": "Iceland" }
{ "name": "Bill", "age": 67, "countryOfResidence": "USA" }
```

เรียก GET โดยใช้ name เป็น id

The screenshot shows the Postman interface. At the top, the URL bar is set to "localhost:3000/users/Bill" with the method "GET". The "Body" tab is selected, showing a JSON payload: `{ "name": "Bill", "age": 67, "countryOfResidence": "USA" }`. The bottom section displays the response in "Pretty" format, which is a JSON object: `{ "age": 67, "countryOfResidence": "USA", "name": "Bill" }`. The status bar at the bottom indicates a "200 OK" response with a 7 ms latency and 185 B of data.

พิจารณาตัวอย่าง code ในไฟล์ akka-example.zip

นักศึกษา สร้างโปรเจคเพื่อรองรับ API ตามที่ได้ออกแบบไว้ใน Task 1 ให้ครบถ้วน

เกณฑ์ประเมินสำหรับ Task 2

| | >80% | 60 – 40 % | 20 - 40% | < 20% |
|---|---|--|---|--------------------------------|
| Select the appropriate code to solve the problems | สามารถเลือกใช้การเขียนโปรแกรมเชิงฟังก์ชันตามโจทย์ที่กำหนดได้ถูกต้อง และสามารถอธิบายความเหมาะสมได้ชัดเจน | สามารถเลือกใช้การเขียนโปรแกรมเชิงฟังก์ชันตามโจทย์ที่กำหนดได้ | สามารถเลือกใช้การเขียนโปรแกรมเชิงฟังก์ชันตามโจทย์ที่กำหนดได้บางส่วน ต้องการคำแนะนำเพื่อการปรับปรุงแก้ไข | ไม่สามารถเลือกใช้เครื่องมือได้ |
| ได้รับคะแนน | 5 | 3-4 | 1-2 | 0 |

วางแผนตัวอย่างผลลัพธ์ ด้วย postman ดังนี้

The screenshot shows a Postman interface for a POST request to `localhost:3000/days`. The request body is a JSON object:

```

1  {
2    "id": 1,
3    "day": "Monday",
4    "colors": "Red",
5    "description": "Unlucky🐼🐼🐼🐼🐼"
6  }

```

The response is shown in the 'Test Results' tab, indicating a successful status (201 Created) and the response body:

```

1  {
2    "description": "data added successfully"
3  }

```

The screenshot displays a REST client interface with two tabs. The top tab is for the endpoint `localhost:3000/days`. It shows a `GET` request with a `JSON` body. The body content is:

```
1 {
2   "id": 1,
3   "day": "Monday",
4   "colors": "Red",
5   "description": "Unlucky🐞🐞🐞🐞🐞"
6 }
```

The response is shown in the bottom panel, indicating a `200 OK` status with a response time of `19 ms` and a size of `227 B`. The response body is a JSON object:

```
1 {
2   "days": [
3     {
4       "colors": "Red",
5       "day": "Monday",
6       "description": "Unlucky🐞🐞🐞🐞🐞",
7       "id": 1
8     }
9   ]
10 }
```

The bottom tab is for the endpoint `localhost:3000/days/1`. It shows a `GET` request with a `JSON` body. The body content is:

```
1 {
2   "id": 1,
3   "day": "Monday",
4   "colors": "Red",
5   "description": "Unlucky🐞🐞🐞🐞🐞"
6 }
```

The response is shown in the bottom panel, indicating a `200 OK` status with a response time of `28 ms` and a size of `216 B`. The response body is a JSON object:

```
1 {
2   "colors": "Red",
3   "day": "Monday",
4   "description": "Unlucky🐞🐞🐞🐞🐞",
5   "id": 1
6 }
```

The image shows a REST client interface with two tabs. The top tab is for a DELETE request to `localhost:3000/days/1`. The request body is a JSON object: `{ "id": 1, "day": "Monday", "colors": "Red", "description": "Unlucky" }`. The response is a 200 OK status with a response time of 10 ms and a body of 175 B. The response body is `{ "description": "data delete successfully" }`.

The bottom tab is for a PUT request to `localhost:3000/days/1`. The request body is a JSON object: `{ "id": 1, "day": "Monday", "colors": "000000Range", "description": "Unlucky" }`. The response is a 200 OK status with a response time of 12 ms and a body of 176 B. The response body is `{ "description": "data update successfully" }`.

Both requests are configured with the following settings: Method (DELETE/PUT), URL (`localhost:3000/days/1`), Headers (8), Body (JSON), Pre-request Script, Tests, Settings, Cookies, and Beautify.