

Lab 5 : Spring Web API with Spring Data (JPA)

Usa Sammapun, Kasetsart University

ในแลปนี้ เราจะสร้าง web API ที่มีการเก็บข้อมูลในฐานข้อมูล

- เพื่อให้เข้าใจ API และการคืนค่าเป็น JSON
- เพื่อให้สามารถใช้ Spring Data แบบ JPA ซึ่งจะลดการใช้ SQL statement ทำให้การเชื่อมต่อฐานข้อมูลง่ายขึ้นมาก

ในแลปนี้ เราจะใช้ H2 ซึ่งเป็นฐานข้อมูลแบบ in-memory ก่อน

1. **เชื่อมต่อกับฐานข้อมูล H2** ซึ่งเป็นฐานข้อมูลแบบ in-memory ช่วยในการพัฒนาได้เร็ว
 - **H2 Database** เป็น database ตัวหนึ่งที่ใช้งานง่าย มาพร้อมกับ Spring Boot เลย ทำให้ไม่ต้องมีการ install แยก แต่เป็น database ที่เก็บข้อมูลไว้ในหน่วยความจำ เมื่อปิดและเปิดโปรแกรมใหม่ ข้อมูลจะหายไป และมีข้อจำกัดมากกว่า database เช่น MySQL, PostgreSQL, Oracle เป็นต้น
2. **เชื่อมต่อกับฐานข้อมูล MySQL** เมื่อปิดและเปิดโปรแกรมใหม่ ข้อมูลจะ**ไม่หายไป**

0. Start Sprint 1 ใน Jira

1. สร้าง story การเพิ่มเมนู และลากเข้า Sprint 1
2. Start Sprint 1

I. Spring Boot starter

3. ไปที่ <https://start.spring.io/>
 - ใส่ข้อมูล project ตามต้องการ นิสิตสามารถใช้ตามอาจารย์ได้เลย
 - **ต้องเลือก Spring Boot version ให้เป็น 2.6.10**
 - เลือก Java version ให้ตรงตามเครื่องคอมพิวเตอร์ของนิสิต
4. พิมพ์และเลือก dependencies ตามรูป แล้วกดปุ่ม GENERATE
5. จะได้เป็นไฟล์ menu.zip ให้ unzip ไฟล์

6. เปิดโปรแกรม IntelliJ แล้วเลือก “Open”
7. เลือกไฟล์เดอร์ menu ที่เพิ่ง unzip
8. จะใช้เวลาในการโหลดโค้ดครั้งแรก

เชื่อมต่อกับ database

ในการเชื่อมต่อกับ database เราจะต้อง configure เพื่อระบุรายละเอียดของ database ที่ใช้ เช่น driver, url, username, password โดยจะระบุในไฟล์ `/src/main/resources/application.properties`

9. กำหนดค่าการเชื่อมต่อ H2 ที่ไฟล์ `/src/main/resources/application.properties`
 - `spring.jpa.hibernate.ddl-auto=update` config นี้จะบอกให้ JPA สร้างตารางให้โดยอัตโนมัติ ถ้าฐานข้อมูลยังไม่มีตารางนั้น ๆ แต่ถ้ามีตารางแล้ว จะอัปเดตให้ถ้ามีการเพิ่มคอลัมน์ เหมาะกับการ dev
 - ถ้าใช้เป็น `create` JPA จะสร้างตารางให้ใหม่ทุกครั้ง เหมาะกับการ test
 - ถ้าใน `production` ที่มีตารางพร้อมข้อมูลแล้ว ไม่ควรใส่ config นี้เลย

```
server.port = 8090

# Enabling H2 Console
```

```
spring.h2.console.settings.web-allow-others=true
spring.h2.console.enabled=true

# Datasource
spring.datasource.url=jdbc:h2:mem:menu
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=test
spring.datasource.password=test

# JPA
spring.jpa.show-sql=true
Spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
```

ใช้ Spring Data JPA

10. สร้าง package model

- เราจะจัดระเบียบโค้ดไปพร้อมกับการเขียนโค้ด

New Package
th.ac.ku.menu.model

11. สร้างคลาส Menu

- สังกัด annotation `@Entity` บอก JPA ว่าเป็นตารางในฐานข้อมูล (JPA จะสร้างตารางชื่อ menu ให้โดยอัตโนมัติ) และให้ map ข้อมูลในตารางมาเป็น object ของคลาสนี้
- สังกัด annotation `@Id` บอก JPA ว่าเป็น primary key
- สังกัด annotation `@GeneratedValue` บอก JPA ว่า generate id ให้โดยอัตโนมัติ
 - i. เราใช้ UUID เป็น type ของ id เพื่อให้ JPA generate id แบบ random ให้อัตโนมัติ เพื่อความปลอดภัย

```
package th.ac.ku.menu.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import java.util.UUID;

@Entity
public class Menu {

    @Id
    @GeneratedValue
    private UUID id;

    private String name;
    private double price;
    private String category;

    // ให้ Generate..
    // - Getters และ Setters ทั้งหมด
}
```

12. สร้าง package repository

13. สร้าง interface MenuRepository ซึ่ง extends interface ที่ชื่อว่า

`JpaRepository<type-of-data , type-of-primary-key>` โดยจะมี implementation ของเมทอด `findAll()`, `findById(int id)`, `save(Menu menu)`, `deleteById(int id)` ให้มาอัตโนมัติ ช่วยต่อ database ในการ select all / select / insert หรือ update / delete โดยไม่ต้องเขียน implementation เอง และไม่ต้องใช้ SQL

```
package th.ac.ku.menu.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import th.ac.ku.menu.model.Menu;

import java.util.UUID;

@Repository
public interface MenuRepository extends JpaRepository<Menu, UUID> {
}
```

สร้าง Service และ Controller สำหรับ Restaurant API

14. สร้าง package service

- จะมีคลาส MenuService ในการเชื่อมต่อกับ MenuRepository

15. สร้าง package controller

- จะมีคลาส MenuController เพื่อรับ request จาก user

16. แยก layer เพื่อแยกหน้าที่ (ตรงกับหลักการออกแบบ Single Responsibility Principle (SRP))

- หน้าที่ controller : รับผิดชอบการ handle user request
- หน้าที่ service : รับผิดชอบการประมวลผลและจัดการข้อมูล
- หน้าที่ repository : รับผิดชอบการเชื่อมต่อฐานข้อมูล

17. สร้างคลาส MenuService ใน package service

- สังเกต annotation `@Service` จะคล้ายกับ `@Component` ซึ่ง Spring จะสร้าง object นี้ให้อัตโนมัติ และมีลักษณะเป็น service ที่จะทำงานตลอดเวลาเพื่อให้บริการ
- สังเกต annotation `@Autowired` Spring จะทำ dependency injection ให้ โดยส่ง object ของ MenuRepository มาให้ MenuService โดยอัตโนมัติ

```
package th.ac.ku.menu.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import th.ac.ku.menu.model.Menu;
import th.ac.ku.menu.repository.MenuRepository;

import java.util.List;

@Service
public class MenuService {

    @Autowired
    private MenuRepository menuRepository;

    public List<Menu> getAll() {
        return menuRepository.findAll();
    }

    public Menu create(Menu menu) {
        Menu record = menuRepository.save(menu);
        return record;
    }
}
```

18. สร้างคลาส MenuController ใน package controller

- สังเกต annotation `@RestController` จะคล้ายกับ `@Service` แต่จะทำหน้าที่รับและส่ง user request โดยใช้ protocol REST โดย Spring จะสร้าง object นี้ให้อัตโนมัติ

```
package th.ac.ku.menu.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import th.ac.ku.menu.model.Menu;
import th.ac.ku.menu.service.MenuService;

import java.util.List;

@RestController
@RequestMapping("/menu")
public class MenuController {

    @Autowired
    private MenuService service;

    @GetMapping
    public List<Menu> getAll() {
        return service.getAll();
    }

    @PostMapping
    public Menu create(@RequestBody Menu menu) {
        return service.create(menu);
    }
}
```

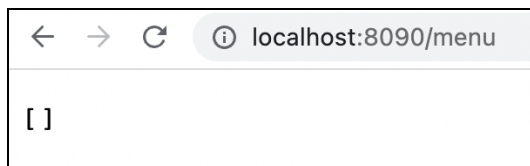
19. Run โปรแกรม (คลิกรันที่ MenuApplication)

20. ไปที่ลิงก์นี้ <http://localhost:8090/h2-console> เพื่อเข้าถึง H2 database

- ใส่ JDBC URL และ username/password ให้ตรงกับใน application.properties
- แล้วดูว่า มีตาราง menu หรือไม่

21. ไปที่ <http://localhost:8090/menu>

- จะได้ลิสต์ว่าง เพราะยังไม่ได้เพิ่มข้อมูล



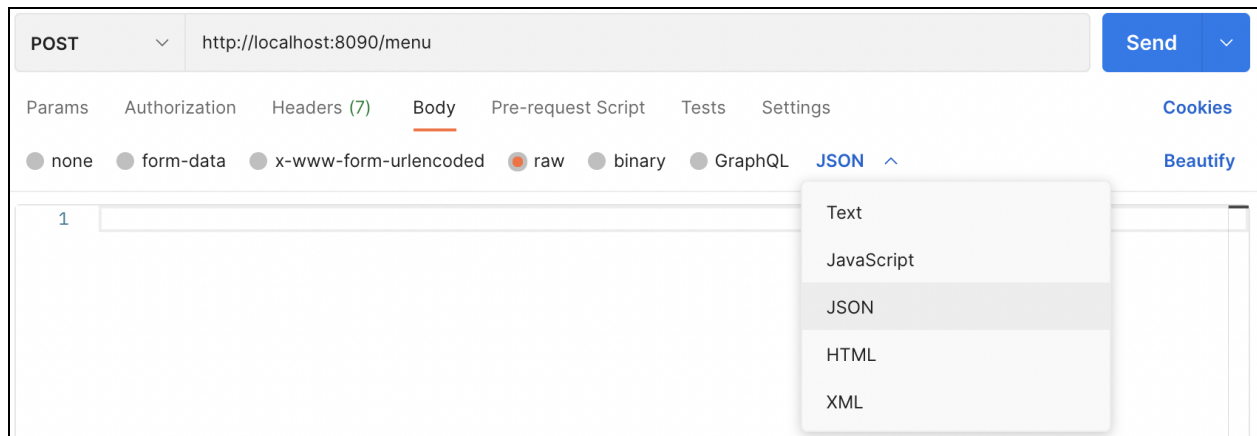
เพิ่มข้อมูลด้วยฟังก์ชัน POST

22. Download โปรแกรม PostMan เพื่อทดสอบการทำงานของ Post

- <https://www.postman.com/downloads/>

23. เปิดโปรแกรม PostMan

- เลือกคำสั่ง POST และใส่ url <http://localhost:8090/menu>
- เลือก tab ที่เป็น Body เลือกแบบ raw และ JSON
- กดที่ “Beautify” เพื่อแสดง JSON ในรูปแบบที่อ่านง่าย
- ใส่ข้อมูลรายการอาหารที่ต้องการ
- แล้วกด Send



```
{  
  "name" : "Cheesecake",  
  "price" : 50.0,  
  "category" : "Dessert"  
}
```

POST

http://localhost:8090/menu

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Beautify

```
1 {
2   "name": "Cheesecake",
3   "price": 50.0,
4   "category": "Dessert"
5 }
6
```

Body

Cookies

Headers (5)

Test Results

200 OK

299 ms

263 B

Save Response

Pretty

Raw

Preview

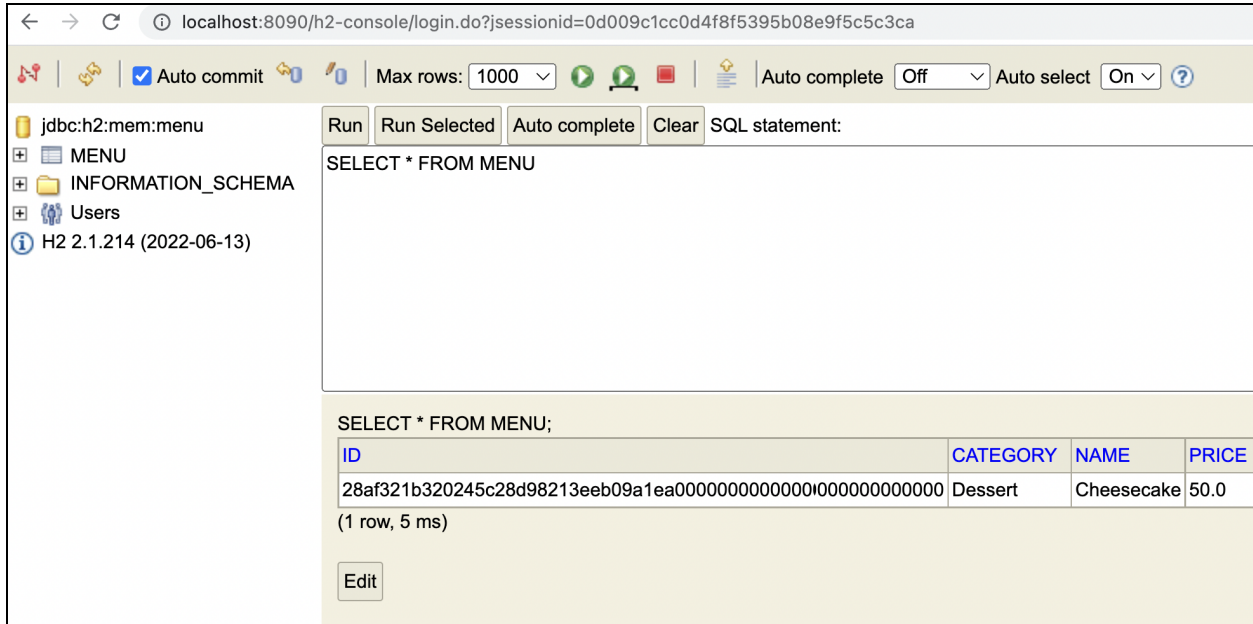
Visualize

JSON

```
1 {
2   "id": "28af321b-3202-45c2-8d98-213eeb09a1ea",
3   "name": "Cheesecake",
4   "price": 50.0,
5   "category": "Dessert"
6 }
```


24. ไปที่ลิงก์นี้ <http://localhost:8090/h2-console> เพื่อเข้าถึง H2 database

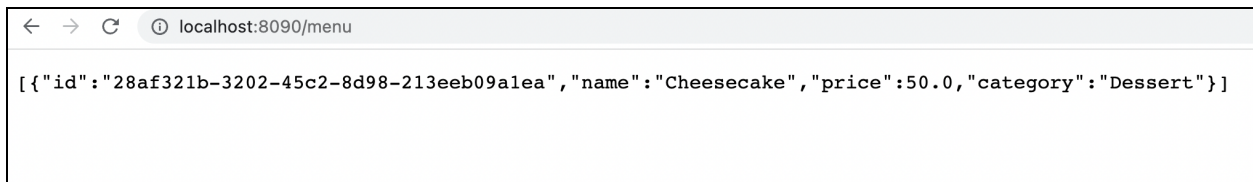
- แล้วดูว่า มีข้อมูลที่เพิ่งใส่ไปหรือไม่



The screenshot shows the H2 database console interface. The left sidebar displays the database structure: jdbc:h2:mem:menu, MENU, INFORMATION_SCHEMA, Users, and H2 2.1.214 (2022-06-13). The main area shows the SQL statement 'SELECT * FROM MENU' and its execution results. The results are displayed in a table with columns ID, CATEGORY, NAME, and PRICE. The table contains one row with the ID '28af321b320245c28d98213eeb09a1ea00000000000000000000000000000000', CATEGORY 'Dessert', NAME 'Cheesecake', and PRICE 50.0. The execution time is 5 ms.

ID	CATEGORY	NAME	PRICE
28af321b320245c28d98213eeb09a1ea00000000000000000000000000000000	Dessert	Cheesecake	50.0

25. ไปที่ <http://localhost:8090/menu>



The screenshot shows the REST client interface with the URL 'localhost:8090/menu'. The response is a JSON array containing one object representing a menu item.

```
[{"id": "28af321b-3202-45c2-8d98-213eeb09a1ea", "name": "Cheesecake", "price": 50.0, "category": "Dessert"}]
```

26. ทดลอง POST ข้อมูลรายการอาหารเพิ่มเติม

เพิ่มฟังก์ชันและ endpoint อื่น ๆ

27. เพิ่มการ GET ด้วย id

- เพิ่มการค้นด้วย id ใน service

```
@Service
public class MenuService {

    // ...

    public Menu getMenuById(UUID id) {
        return menuRepository.findById(id).get();
    }

}
```

- เพิ่ม get mapping ที่รับ path variable เป็น name

```
@RestController
@RequestMapping("/menu")
public class MenuController {

    // ..

    @GetMapping("/{id}")
    public Menu getMenuById(@PathVariable UUID id) {
        return service.getMenuById(id);
    }
}
```

- Rerun และ Post ข้อมูล
- จากนั้นให้ลอง GET ไปที่ <http://localhost:8090/menu/971f913a-ff7e-4e35-9b2a-c307821ad249> โดยใส่ UUID ของข้อมูลนี้
- (เราใช้ GET ใน Postman ก็ได้)



28. เพิ่มการ PUT

- เพิ่มการ update ใน MenuService class

```
public Menu update(Menu requestBody) {
    UUID id = requestBody.getId();
    Menu record = menuRepository.findById(id).get();
    record.setName(requestBody.getName());
    record.setPrice(requestBody.getPrice());
    record.setCategory(requestBody.getCategory());

    record = menuRepository.save(record);
    return record;
}
```

- เพิ่ม put mapping ที่รับ request body ใน MenuController

```
@PutMapping
public Menu update(@RequestBody Menu menu) {
    return service.update(menu);
}
```

- Rerun และ post ข้อมูลเดิม
- ใช้ Postman เรียก PUT function โดยเปลี่ยนข้อมูล เช่น price

PUT http://localhost:8090/menu Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```

1  {
2    "id": "8ac0cf80-ea95-4a81-a8bd-1b51b2bad2bd",
3    "name": "Cheesecake",
4    "price": 60.0,
5    "category": "Dessert"
6  }
7

```

Body Cookies Headers (5) Test Results 200 OK 81 ms 263 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "id": "8ac0cf80-ea95-4a81-a8bd-1b51b2bad2bd",
3    "name": "Cheesecake",
4    "price": 60.0,
5    "category": "Dessert"
6  }

```

29. เพิ่มการ DELETE

- เพิ่มการ delete ใน MenuService

```

public Menu delete(UUID id) {
    Menu record = menuRepository.findById(id).get();
    menuRepository.deleteById(id);
    return record;
}

```

- เพิ่ม delete mapping ที่รับ path variable เป็น id ใน MenuController

```

@DeleteMapping("/{id}")
public Menu delete(@PathVariable UUID id) {
    return service.delete(id);
}

```

- Rerun และ post ข้อมูล
- ใช้ Postman เรียก DELETE function

DELETE http://localhost:8090/menu/e7194c2c-27cf-474b-8cec-f4e241050b9f Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

เพิ่มการค้นหาข้อมูลด้วยชื่อหรือประเภท

30. เพิ่มการ query ด้วย name หรือ category ใน Repository

- (ตั้งสมมติฐานว่า ชื่อเมนูอาหาร นั้น unique)
- ถ้า query ด้วย attribute ใน class เราแค่ประกาศ findByAttribute() ก็พอแล้ว Spring จะ implement ให้อัตโนมัติ

```
@Repository
public interface MenuRepository extends JpaRepository<Menu, UUID> {
    Menu findByName(String name);
    List<Menu> findByCategory(String category);
}
```

31. เพิ่มการค้นหาด้วย name และ category ใน service

```
@Service
public class MenuService {

    // ...

    public Menu getMenuByName(String name) {
        return menuRepository.findByName(name);
    }

    public List<Menu> getMenuByCategory(String category) {
        return menuRepository.findByCategory(category);
    }
}
```

32. เพิ่ม get mapping ที่รับ path variable เป็น name และ category

```
@RestController
@RequestMapping("/menu")
public class MenuController {

    // ..

    @GetMapping("/name/{name}")
    public Menu getMenuByName(@PathVariable String name) {
        return service.getMenuByName(name);
    }

    @GetMapping("/category/{category}")
    public List<Menu> getMenuByCategory(@PathVariable String category) {
        return service.getMenuByCategory(category);
    }
}
```

33. Rerun และ post หลาย ๆ ข้อมูล

- ใช้ Postman เรียกเพื่อหาตามชื่อและประเภท
- ถ้ามีการเว้นวรรคในชื่อหรือประเภท ให้ใช้ %20 แทนการเว้นวรรค เช่น
- <http://localhost:8090/menu/name/Fruit%20Tart>

GET ▼ http://localhost:8090/menu/name/หมูบึ่ง Send ▼

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▼ Beautify

1 5

Body Cookies Headers (5) Test Results 200 OK 104 ms 276 B Save Response ▼

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "id": "189602a2-6978-44b3-ad9b-1b437a987ac1",
3   "name": "หมูบึ่ง",
4   "price": 20.0,
5   "category": "Appetizer"
6 }
```

GET ▼ http://localhost:8090/menu/name/Fruit%20Tart Send ▼

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL JSON ▼ Beautify

1 5

Body Cookies Headers (5) Test Results 200 OK 22 ms 263 B Save Response ▼

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {
2   "id": "603f6dd3-6023-4509-b665-a11bbdcc1b5d",
3   "name": "Fruit Tart",
4   "price": 40.0,
5   "category": "Dessert"
6 }
```

GET

http://localhost:8090/menu/category/Dessert

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

Beautify

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

5

Body

Cookies

Headers (5)

Test Results

200 OK

42 ms

365 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

```
{
  "id": "0d667674-5d54-4cca-9432-770283981337",
  "name": "Cheesecake",
  "price": 50.0,
  "category": "Dessert"
},
{
  "id": "603f6dd3-6023-4509-b665-a11bbdcc1b5d",
  "name": "Fruit Tart",
  "price": 40.0,
  "category": "Dessert"
}
```