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

Usa Sammapun, Kasetsart University

---

ในแลปนี้ เราจะสร้าง web API ที่มีการเก็บข้อมูลในฐานข้อมูล

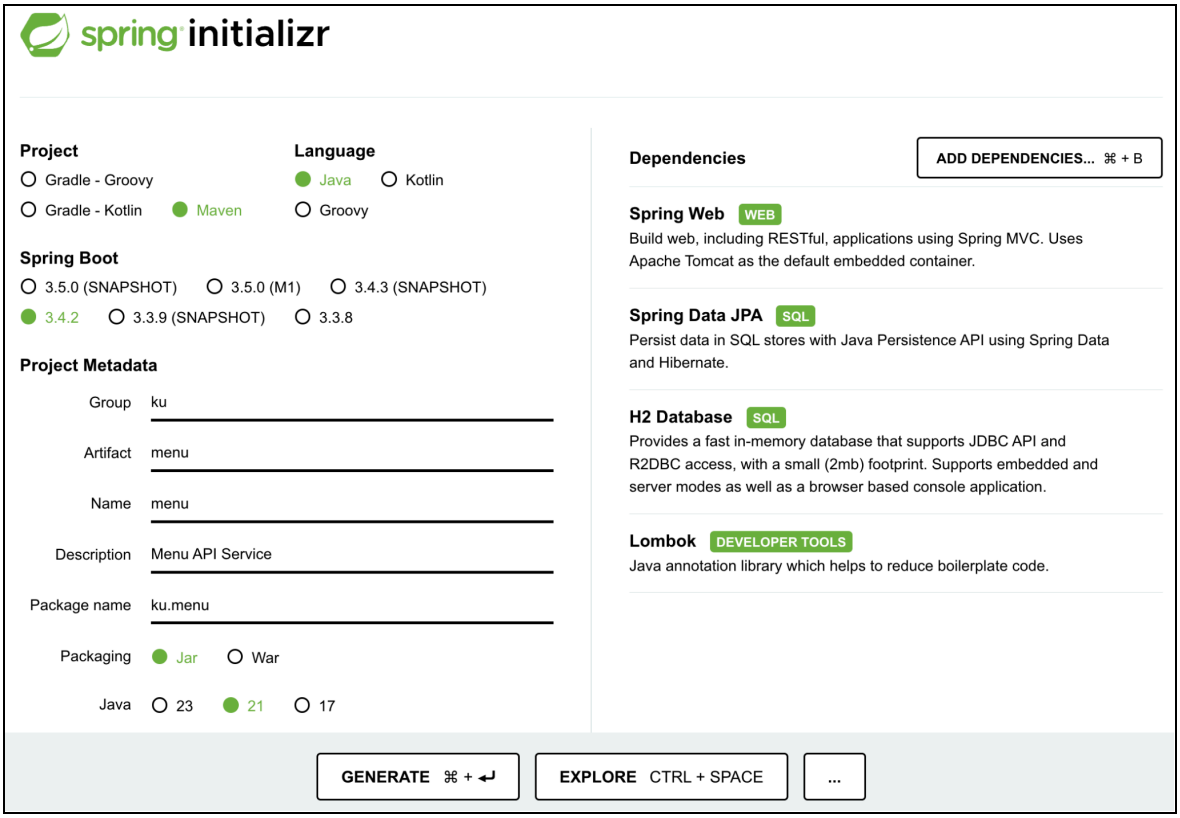
- เพื่อให้เข้าใจ API และการคืนค่าเป็น JSON
- เพื่อให้สามารถใช้ Spring Data แบบ JPA ซึ่งลดการใช้ SQL statement ทำให้การเชื่อมต่อฐานข้อมูลง่ายขึ้นมาก

ในแลปนี้ เราจะใช้ H2 ซึ่งเป็นฐานข้อมูลแบบ in-memory ก่อน

1. **เชื่อมต่อกับฐานข้อมูล H2** ซึ่งเป็นฐานข้อมูลแบบ in-memory ช่วยในการพัฒนาได้เร็ว
    - **H2 Database** เป็น database ที่ใช้งานง่าย มาพร้อมกับ Spring Boot ทำให้ไม่ต้อง install แยก แต่เก็บข้อมูลไว้ในหน่วยความจำ เมื่อปิดและเปิดโปรแกรมใหม่ ข้อมูลจะหายไป มีข้อจำกัดมากกว่า database เช่น MySQL, PostgreSQL, Oracle เป็นต้น
2. **เชื่อมต่อกับฐานข้อมูล MySQL** เมื่อปิดและเปิดโปรแกรมใหม่ ข้อมูลจะ**ไม่หาย**ไป

# I. Spring Boot starter

1. ไปที่ https://start.spring.io/
    - ใส่ข้อมูล project ตามต้องการ นิสิตสามารถใช้ตามอาจารย์ได้เลย
    - เลือก Java version ให้ตรงตามเครื่องคอมพิวเตอร์ของนิสิต
2. พิมพ์และเลือก dependencies ตามรูป แล้วกดปุ่ม GENERATE



3. จะได้เป็นไฟล์ menu.zip ให้ unzip ไฟล์
4. เปิดโปรแกรม IntelliJ แล้วเลือก “Open”
5. เลือกโฟลเดอร์ menu ที่เพิ่ง unzip
6. จะใช้เวลานานในการโหลดโค้ดครั้งแรก

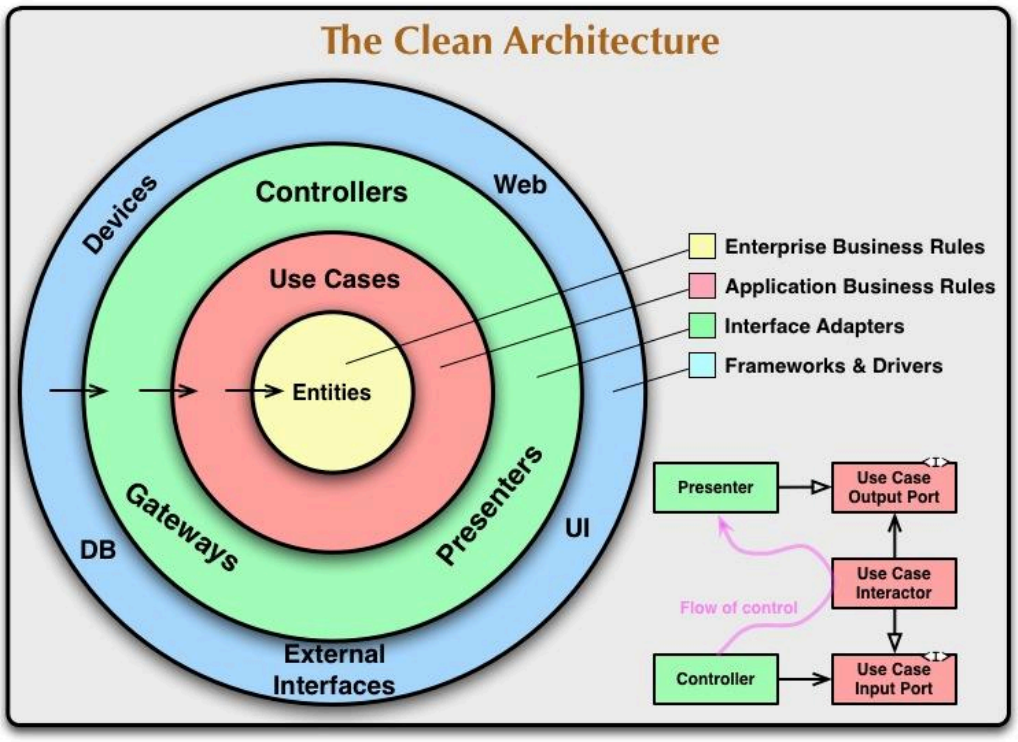
## เชื่อมต่อกับ database

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

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

```
server.port = 8090

# Enabling H2 Console
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console/

# 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
```

## วาง Clean Architecture ในแบบ Java Spring Boot




https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

8. เราแยก layer เพื่อแยกหน้าที่ (ตรงกับหลักการ Single Responsibility Principle (SRP) และ separation of concerns ด้วย)
    - หน้าที่ controller : รับผิดชอบการ handle user request
    - หน้าที่ use case หรือ service : รับผิดชอบประมวลผลและจัดการข้อมูลระดับ application
    - หน้าที่ entity และ repository : รับผิดชอบประมวลผลข้อมูลระดับ domain และเชื่อมต่อฐานข้อมูล

9. package entity และ repository
    - Package entity จะมีคลาส Menu ในการประมวลผลข้อมูลระดับ domain
    - Package repository จะมี interface `MenuRepository` ช่วยต่อ database ในการ select all / select / insert หรือ update / delete โดยไม่ต้องเขียน implementation เอง และไม่ต้องใช้ SQL

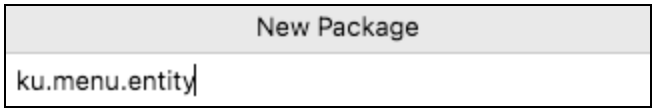
10. package service
    - เป็น application business rules ใช้ประมวลผลต่าง ๆ และจัดการข้อมูลระดับ application
    - จะมีคลาส MenuService ในการเชื่อมต่อกับ MenuRepository

11. package controller
    - เป็น interface adapters ที่เชื่อมกับ user และรับ request จาก user

- จะมีคลาส MenuController

## Layer entity (ใช้ Spring Data JPA)

12. สร้าง package entity
    - เราจะจัดระเบียบโค้ดไปพร้อมกับการเขียนโค้ด




13. สร้างคลาส Menu
    - สังเกต annotation `@Data` เป็นส่วนหนึ่งของ lombok tool ที่ช่วยสร้างเมธอด getter และ setter ในการอ่านและกำหนดค่าตัวแปรในคลาส
    - สังเกต annotation `@Entity` บอก JPA ว่าเป็นตารางในฐานข้อมูล (JPA จะสร้างตารางชื่อ menu ให้โดยอัตโนมัติ) และให้ map ข้อมูลในตารางมาเป็น object ของคลาสนี้
    - สังเกต annotation `@Id` บอก JPA ว่าเป็น primary key
    - สังเกต annotation `@GeneratedValue` บอก JPA ว่า generate id ให้โดยอัตโนมัติ
        i. เราใช้ UUID เป็น type ของ id เพื่อให้ JPA generate id แบบ random ให้อัตโนมัติ เพื่อความปลอดภัย

```java
package ku.menu.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.Data;

import java.util.UUID;

@Entity
@Data
public class Menu {

   @Id
   @GeneratedValue
   private UUID id;

   private String name;
   private double price;
   private String category;
}
```

14. สร้าง package repository
15. สร้าง interface `MenuRepository` ซึ่ง extends interface ที่ชื่อว่า `JpaRepository<type-of-data , type-of-primary-key>` โดยจะมี implementation ของเมทอด `findAll()`, `findById(UUID id)`, `save(Menu menu)`, `deleteById(UUID id)` ให้มาอัตโนมัติ ช่วยต่อ database ในการ select all / select / insert หรือ update / delete โดยไม่ต้องเขียน implementation เอง และไม่ต้องใช้ SQL

```
package ku.menu.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import ku.menu.entity.Menu;

import java.util.UUID;

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

## Layer Use Case (Service) และ Controller

16. สร้าง package service
17. สร้างคลาส MenuService ใน package service
    - สังเกต annotation `@Service` ซึ่ง Spring จะสร้าง object นี้ให้อัตโนมัติ และมีลักษณะเป็น service ที่จะทำงานตลอดเวลาเพื่อให้บริการ
    - สังเกต annotation `@Autowired` Spring จะทำ dependency injection ให้ โดยส่ง object ของ MenuRepository มาให้ MenuService โดยอัตโนมัติ

```
package ku.menu.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ku.menu.entity.Menu;
import 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. สร้าง package controller
19. สร้างคลาส MenuController ใน package controller
    - สังเกต annotation `@RestController` จะคล้ายกับ @Service แต่จะทำหน้าที่รับและส่ง user request โดยใช้ protocol REST โดย Spring จะสร้าง object นี้ให้อัตโนมัติ

```
package ku.menu.controller;

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

import java.util.List;

@RestController
public class MenuController {

    @Autowired
    private MenuService service;

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

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

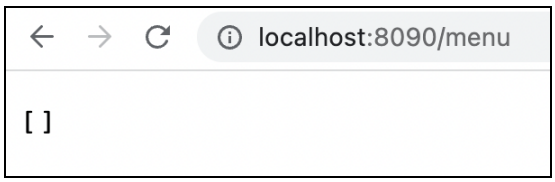
20. Run โปรแกรม (คลิกรันที่ MenuApplication)

21. ถ้าใช้ Visual Studio Code ให้ download Apache Maven
    https://maven.apache.org/download.cgi
    - ให้รันด้วยคำสั่งต่อไปนี้

```
mvn spring-boot:run
```

22. ไปที่ลิงก์นี้ http://localhost:8090/h2-console เพื่อเข้าถึง H2 database
    - ใส่ JDBC URL และ username/password ให้ตรงกับใน application.properties
    - แล้วดูว่า มีตาราง menu หรือไม่

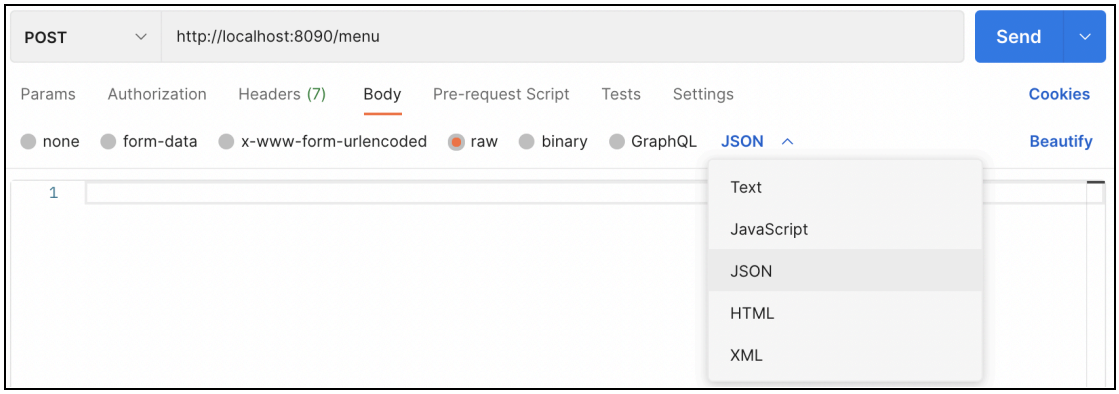
23. ไปที่ http://localhost:8090/menu
    - จะได้ลิสต์ว่าง เพราะยังไม่ได้เพิ่มข้อมูล



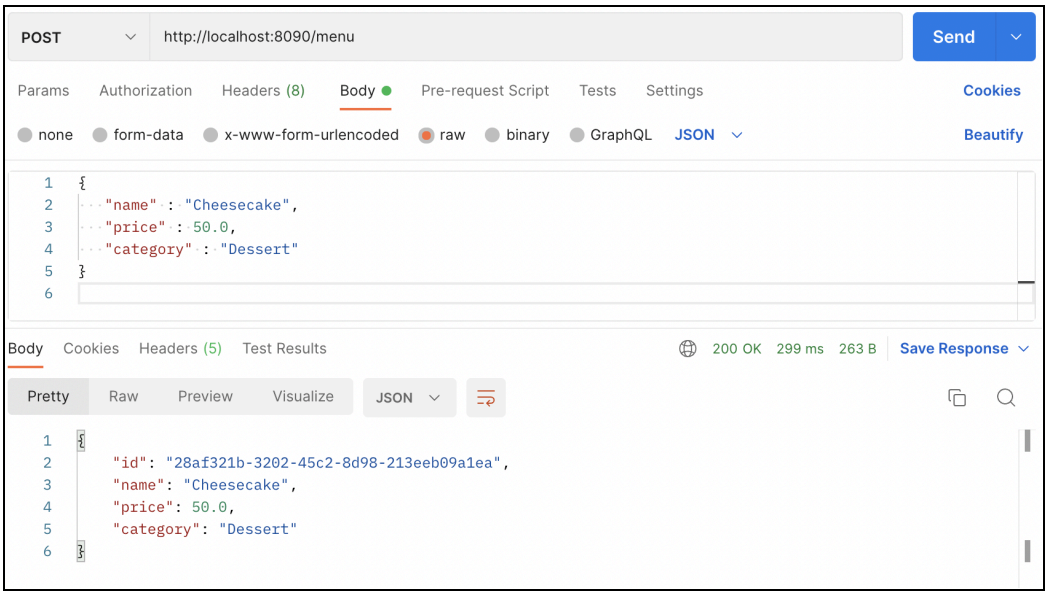

## เพิ่มข้อมูลด้วยฟังก์ชัน POST

24. Download โปรแกรม PostMan เพื่อทดสอบการทำงานของ Post
    - https://www.postman.com/downloads/

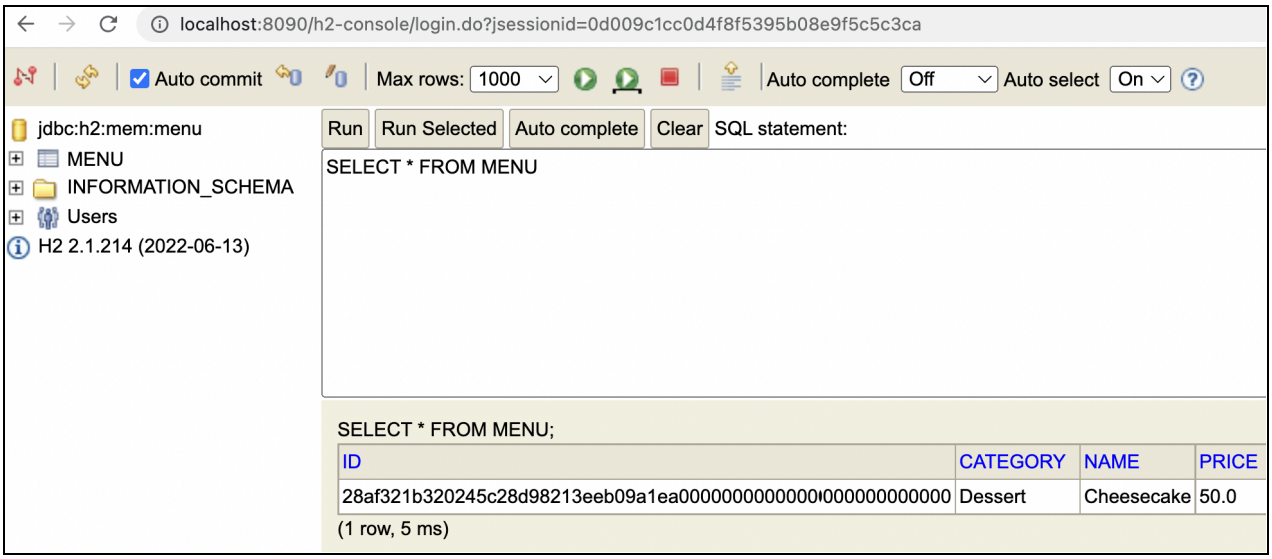
25. เปิดโปรแกรม PostMan
    - เลือกคำสั่ง POST และใส่ url http://localhost:8090/menu
    - เลือก tab ที่เป็น Body เลือกแบบ raw และ JSON
    - กดที่ “Beautify” เพื่อแสดง JSON ในรูปแบบที่อ่านง่าย
    - ใส่ข้อมูลรายการอาหารที่ต้องการ
    - แล้วกด Send



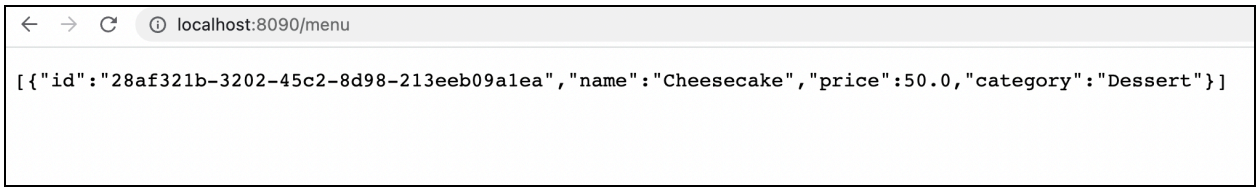

```
{
  "name" : "Cheesecake",
  "price" : 50.0,
  "category" : "Dessert"
}
```




26. ไปที่ลิงก์นี้ http://localhost:8090/h2-console เพื่อเข้าถึง H2 database
    - แล้วดูว่า มีข้อมูลที่เพิ่งใส่ไปหรือไม่




27. ไปที่ http://localhost:8090/menu




28. ทดลอง POST ข้อมูลรายการอาหารเพิ่มเติม

## เพิ่มฟังก์ชันและ endpoint อื่น ๆ

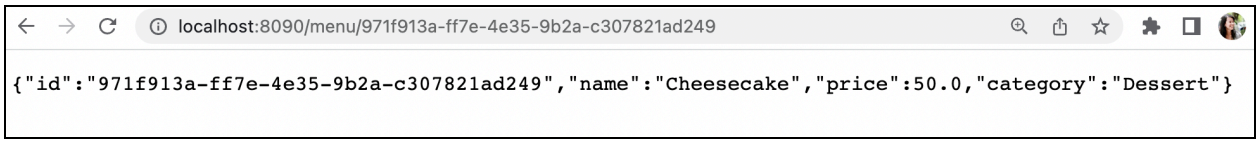
29. เพิ่มการ GET ด้วย id
    - เพิ่มการค้นด้วย id ใน service

```
@Service
public class MenuService {

   // ...

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

}
```

    - เพิ่ม get mapping ที่รับ path variable เป็น name

```
@RestController
public class MenuController {

   // ..

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

    - Rerun และ Post ข้อมูล
    - จากนั้นให้ลอง GET ไปที่ http://localhost:8090/menu/UUID โดยให้ใส่ UUID ของข้อมูลนิสิต
    - (เราใช้ GET ใน Postman ก็ได้)



30. เพิ่มการ PUT

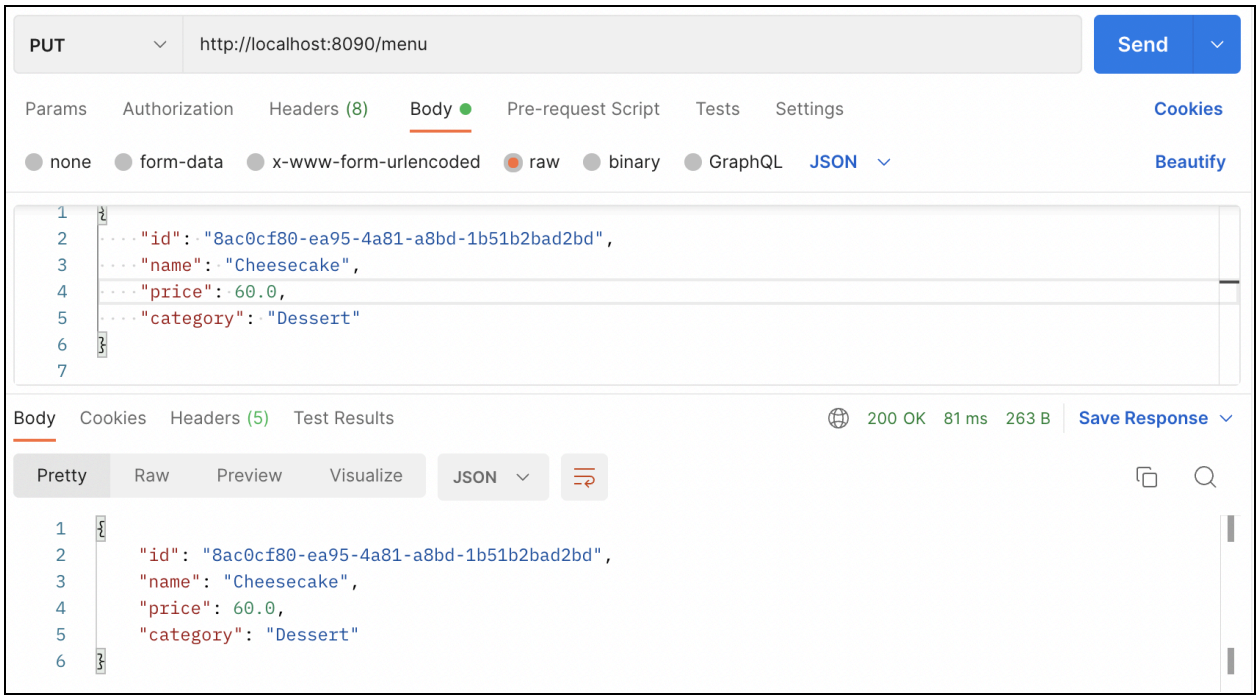
- เพิ่มการ update ใน MenuService class

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

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



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

    - Rerun และ post ข้อมูล
    - ใช้ Postman เรียก DELETE function


## เพิ่มการค้นหาข้อมูลด้วยชื่อหรือประเภท

32. 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);
}
```

33. เพิ่มการค้นด้วย name และ category ใน service

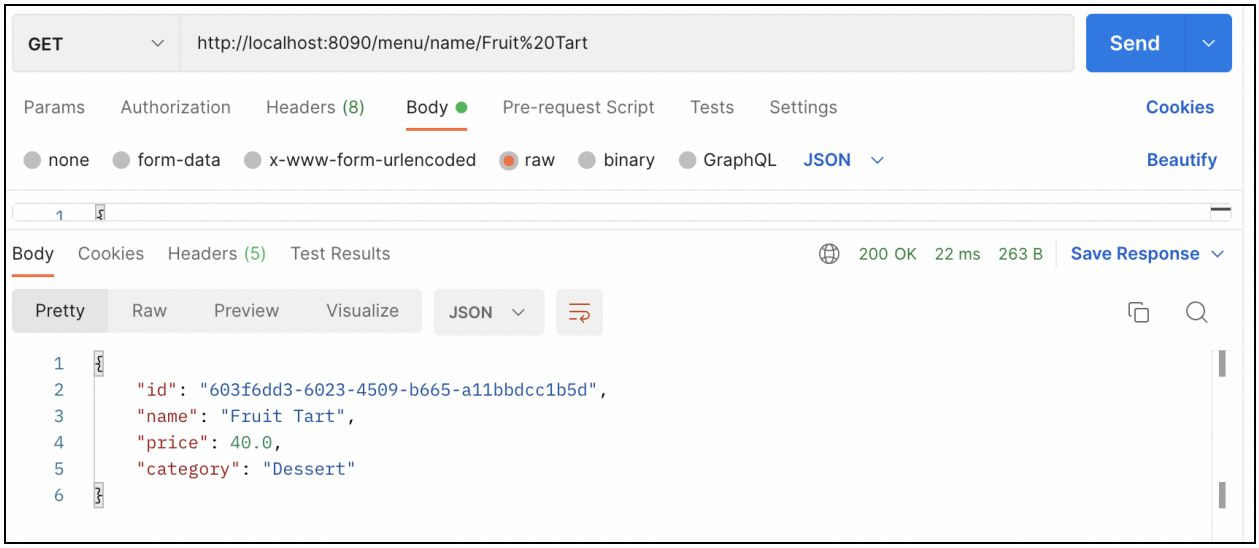
```
@Service
public class MenuService {

   // ...

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

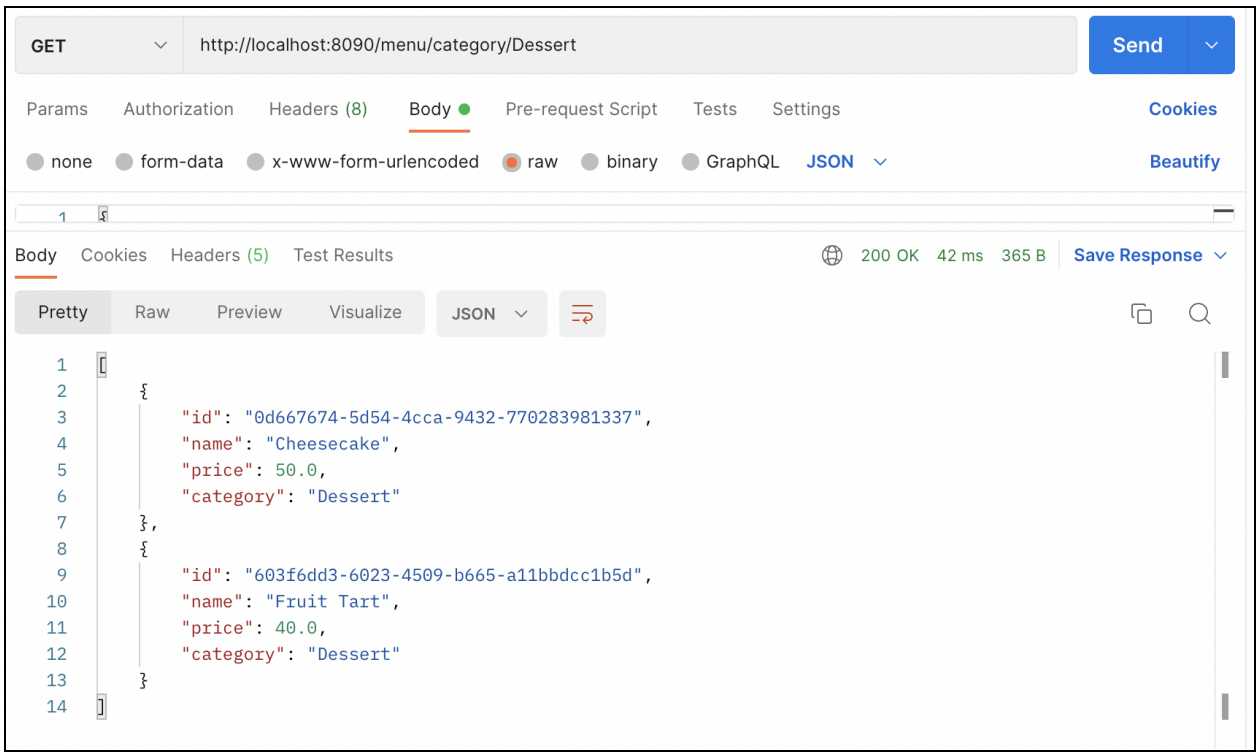
```
    public List<Menu> getMenuByCategory(String category) {
        return menuRepository.findByCategory(category);
    }
}
```

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

```
@RestController
public class MenuController {

    // ..

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

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

35. 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/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
Body
Cookies
Headers (5)
Test Results
200 OK
22 ms
263 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"id": "603f6dd3-6023-4509-b665-a11bbdcc1b5d",
"name": "Fruit Tart",
"price": 40.0,
"category": "Dessert"
}


GET
http://localhost:8090/menu/category/Dessert
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
Body
Cookies
Headers (5)
Test Results
200 OK
42 ms
365 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"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"
}
]