Work on project. Stage 3/5: Store a recipe

Project: Recipes

Store a recipe

Hard 3 hours **475** users solved this stage. Latest completion was **about 19 hours ago**.

Description

In the previous stage, we have improved our service, so it can handle a lot of recipes. But when we close our program, it deletes all recipes. In this stage, you'll implement one of the main features of the service – connect the service to a database and store the recipes there. No more lost recipes!

You will also need a new endpoint that will allow deleting a recipe by the recipe `id`. Make sure that the service accepts only valid recipes – recipes without directions or ingredients are frustrating. We won't change the recipe structure in this stage.

38 / 38 Prerequisites

- Anonymous classes
- Default methods
- Lambda expressions 2
- Method references
- Functional interfaces

Show all

Join a study group for the project Recipes

Discuss your current project with fellow learners and help each other.

Objectives

First of all, include all necessary dependencies and configurations in the `build.gradle` and `application.properties` files.

For testing reasons, the `application.properties` file should contain the following line with the database name:

```
spring.datasource.url=jdbc:h2:file:../recipes_db
```

The service should support the same endpoints as in the previous stage:

- `POST /api/recipe/new` receives a recipe as a JSON object and returns a JSON object with one `id` field;
- `GET /api/recipe/{id}` returns a recipe with a specified `id` as a JSON object.

To complete the stage you need to add the following functionality:

- Store all recipes permanently in a database: after a server restart, all added recipes should be available to a user;
- Implement a new `DELETE /api/recipe/{id}` endpoint. It deletes a recipe with a specified `{id}`. The server should respond with the `204 (No Content)` status code. If a recipe with a specified id does not exist, the server should return `404 (Not found)`;
- The service should accept only valid recipes – all fields are obligatory, `name` and `description` shouldn't be blank, and JSON arrays should contain at least one item. If a recipe doesn't meet these requirements, the service should respond with the `400 (Bad Request)` status code.

Examples

Example 1: `POST /api/recipe/new` request

```
{
   "name": "Warming Ginger Tea",
   "description": "Ginger tea is a warming drink for cool weather, ...",
   "ingredients": ["1 inch ginger root, minced", "1/2 lemon, juiced", "1/2 teaspoon manuka honey"],
   "directions": ["Place all ingredients in a mug and fill with warm water (not too hot so you keep the beneficial h
10 minutes", "Drink and enjoy"]
}
```

Response:

```
{
   "id": 1
}
```

Example 2: Response for the `GET /api/recipe/1` request

```
{
   "name": "Warming Ginger Tea",
   "description": "Ginger tea is a warming drink for cool weather, ...",
   "ingredients": ["1 inch ginger root, minced", "1/2 lemon, juiced", "1/2 teaspoon manuka honey"],
   "directions": ["Place all ingredients in a mug and fill with warm water (not too hot so you keep the beneficial h
10 minutes", "Drink and enjoy"]
}
```

Example 3:

`DELETE /api/recipe/1` request

Status code: `204 (No Content)`

`DELETE /api/recipe/1` request

Status code: `404 (Not found)`

Example 4:

`GET /api/recipe/1` request

Status code: `404 (Not found)`

Report a typo

See hint

Write a program

Code Editor IDE

CONNECTION STATUS

IDE / Checking the plugin's status

Solve in IDE

Look up solution (100)

Comments (56) Hints (14) Useful links (12) Solutions (124) Show discussion