

RECIPE BOOK

A JAVA-SPRING BOOT PROJECT

PRESENTED BY:

S.V.S.SAI APARNA (BBPBV008)

S.SAI SINDHU (BBPBV009)

A.S.L.YASASWINI (BBPBV011)

SHAIK AFSANA (BBPBV012)

CONTENTS

- Problem Statement
- Introduction
- Software Requirements
- Hardware Requirements
- Architecture
- Modules involved in the project
- Output
- Conclusion



PROBLEM STATEMENT

Challenges of Traditional Recipe Management are,

- >> **Manual Process:** Recipes are often managed on paper or in physical cookbooks, making them difficult to organize and search.
- >> **Limited Access:** Recipes may not be easily accessible to family members or friends.
- >> **Lack of Personalization:** Users cannot easily customize recipes based on dietary preferences or available ingredients.
- >> **Difficulty in Scaling:** For professional chefs or cooking enthusiasts, managing a large number of recipes can be overwhelming. etc...

SOLUTION: Digitalizing the Recipe Management

INTRODUCTION

The Recipe Book Web Application is a Java Spring Boot project that provides a user-friendly platform for managing recipes. You can expand upon the basic features like create, update, get and delete the recipes.

> It allows users to perform the following actions:

1. **Recipe Search:** Users can search for recipes based on ids, ingredients, or categories, making it easy to find specific recipes.
2. **Recipe Categories:** Recipes can be categorized into various types. Users can browse recipes by category.
3. **Recipe Ingredients:** A recipe can have many ingredients. Users can easily search the ingredients of a recipe by using recipe ids.

We can even consider adding features like user reviews, meal planning and integration with external APIs to enhance the applications functionality.

“ FOOD IS EVERYTHING EVERYWHERE ALL AT ONCE ”

The potential **applications** of the Recipe Book System include

- Recipe Sharing Platform
- Meal planning and Grocery Lists
- Personal Recipe Organizer
- Cooking Education and tutorials
- Nutrition and Dietary Tracking
- Restaurant and Menu Recommendation
- Integration with Smart Appliances
- Cookbook Publications
- Event and Party Planning
- Restaurant Reviews and Recommendations
- Localization and Cuisine Exploration
- Food Blogging and Content Creation
- Cooking Competitions and Challenges etc...

SOFTWARE REQUIREMENTS

- **DEVELOPMENT ENVIRONMENT:** Java Development Kit (JDK)
- **FRAMEWORK :** Spring Boot
- **DATABASE:** MySQL
- **INTEGRATED DEVELOPMENT ENVIRONMENT(IDE):** Eclipse
- **BUILD TOOL:** Apache Maven
- **WEB SERVER:** Apache Tomcat
- **API TESTING TOOL:** Postman
- **DEPENDENCY MANAGEMENT:** Apache Maven

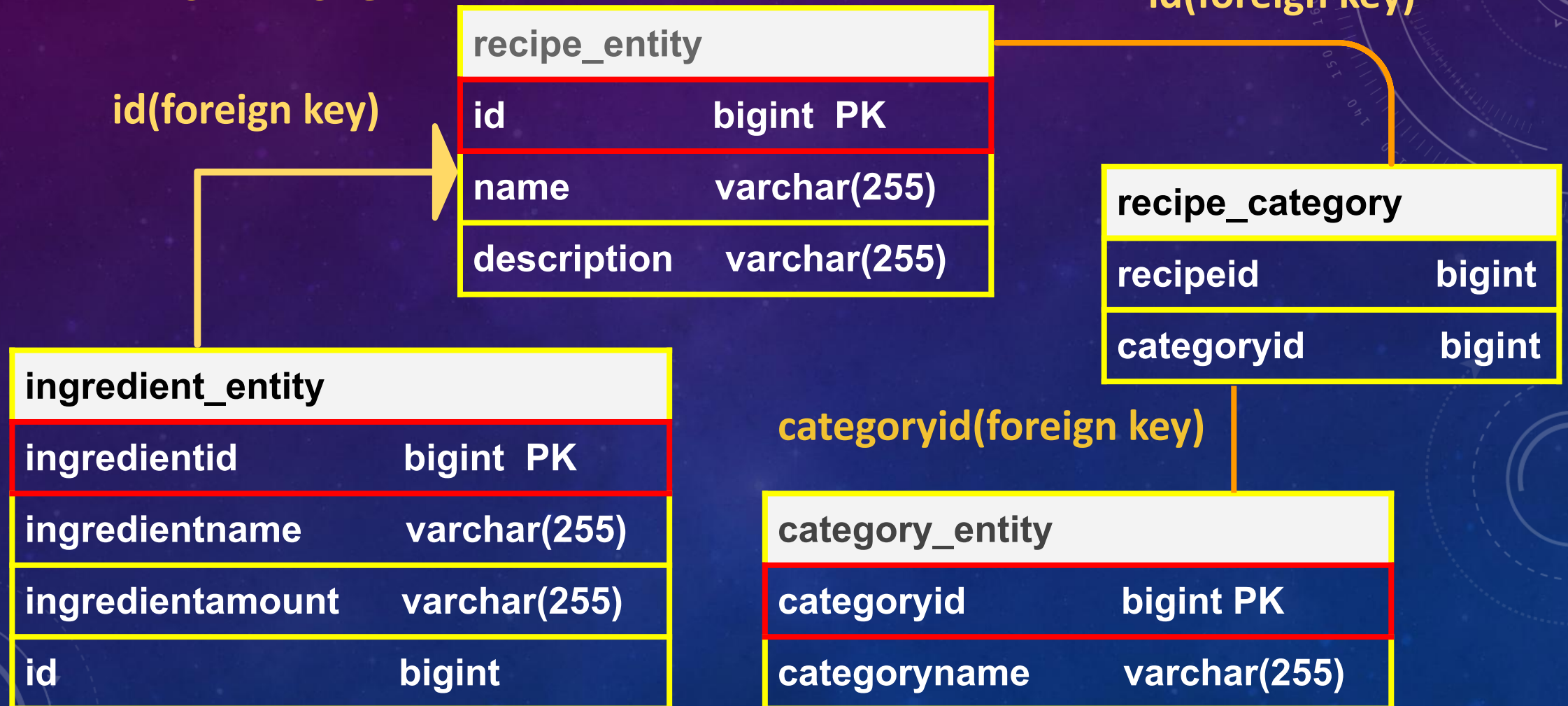
HARDWARE REQUIREMENTS:

Laptop 4Ghz minimum, multicore-core processor,
memory(RAM)-4GB or high, Hard disk space 1TB.



ARCHITECTURE

DATABASE DESIGN



PROJECT STRUCTURE



Recipe Book

→src/main/java

→com.bbproject

> RecipeBookApplication.java

→ com.bbproject.controller

> CategoryController.java

> IngredientController.java

> RecipeController.java

→ com.bbproject.entity

> CategoryEntity.java

> IngredientEntity.java

> RecipeEntity.java

→ com.bbproject.repository

> CategoryRepository.java

> IngredientRepository.java

> RecipeRepository.java

→com.bbproject.service

> CategoryService.java

> IngredientService.java

> RecipeService.java

→src/main/resources

> application.properties

→pom.xml

MODULES INVOLVED

1 RECIPE MANAGEMENT MODULE:

Description: The core module for adding, editing, and organizing recipes.

Key Features:

- Recipe creation with name, description, ingredients, and categories.
- Editing and updating existing recipes.
- Categorization and tagging of recipes.
- Recipe_id acts as primary key to uniquely identify recipes.
- This module maintains one to many relation with ingredient module.
- This module maintains a many to many relation with category module.

Importance: Recipe management is at the heart of the application, allowing users to store and organize their culinary creations.

2 INGREDIENT MANAGEMENT MODULE

Description: This module deals with managing ingredients used in recipes.

Key Features:

- Adding new ingredients.
- Viewing ingredient details and measurements.
- Associating ingredients with recipes.
- Ingredient_id acts as primary key to uniquely identify the ingredient.
- This module maintains a many to one relation with recipe module.

Importance: Ingredient management ensures accurate recipe descriptions and measurements.

3 CATEGORY MANAGEMENT MODULE

Description: Handles the organization of recipes into categories or types.

Key Features:

- Creating recipe categories (e.g., appetizers, desserts, main courses).
- Assigning recipes to categories.
- Browsing recipes by category.
- Category_id acts as primary key to uniquely identify the category.
- This module maintains a many to many relation with recipe module.

Importance: Category management helps users find recipes quickly by grouping them into relevant sections.

OUTPUT

RECIPE TABLE

The screenshot displays a database management interface. On the left, a tree view shows the database structure under 'sys', including tables like 'category_entity', 'ingredient_entity', 'recipe_category', 'recipe_entity', and 'sys_config'. The 'recipe_entity' table is selected, and its details are shown in the 'Table: recipe_entity' section, including columns 'id', 'description', and 'name' with their respective data types.

The main area shows the 'Result Grid' for the 'recipe_entity' table. It contains a toolbar with options like 'Filter Rows', 'Edit', 'Export/Import', and 'Wrap Cell Content'. The grid displays the following data:

	id	description	name
▶	1	Add flour, salt, cheese. Mix it well. Keep it in ov...	PIZZA
	2	Add maida flour, sugar, coco powder, eggs. Mix...	CAKE
	4	Add tomatoes, onions, chatpowder. Mix it well. ...	CHAT
*	NULL	NULL	NULL

Below the result grid, the 'Output' section shows a list of actions and their results:

#	Time	Action	Message
✓ 26	11:56:36	SELECT * FROM sys.recipe_category LIMIT 0, 1000	6 row(s) returned
✓ 27	12:04:07	SELECT * FROM sys.ingredient_entity LIMIT 0, 1000	11 row(s) returned
✓ 28	12:04:14	SELECT * FROM sys.recipe_entity LIMIT 0, 1000	4 row(s) returned
✓ 29	12:06:16	SELECT * FROM sys.ingredient_entity LIMIT 0, 1000	12 row(s) returned
✓ 30	12:08:18	SELECT * FROM sys.category_entity LIMIT 0, 1000	5 row(s) returned
✓ 31	12:10:43	SELECT * FROM sys.recipe_entity LIMIT 0, 1000	3 row(s) returned

INGREDIENT TABLE

The screenshot displays a database management interface. On the left, a tree view shows the database structure with 'ingredient_entity' selected under the 'Tables' folder. Below this, the 'Table: ingredient_entity' is detailed with its columns: 'ingredientid' (bigint), 'ingredientamount' (varchar(2)), 'ingredientname' (varchar(2)), and 'recipe_id' (bigint). The main area shows a 'Result Grid' with 12 rows of data. The bottom panel shows the 'Output' tab with a list of SQL queries and their results.

	ingredientid	ingredientamount	ingredientname	recipe_id
▶	1	1 Kilogram	Flour	1
	2	2 Table Spoon	Salt	1
	3	200 Grams	Cheese	1
	4	500 Grams	Maid Flour	2
	5	500 Grams	Sugar	2
	6	Half Dozen	Eggs	2
	7	200 Grams	Coco Powder	2
	13	5 Tomatoes	Tomatoes	4
	14	5 Onions	Onions	4
	15	100 Grams	Chat Powder	4
•	NULL	NULL	NULL	NULL

#	Time	Action	Message
✓ 29	12:06:16	SELECT * FROM sys.ingredient_entity LIMIT 0, 1000	12 row(s) returned
✓ 30	12:08:18	SELECT * FROM sys.category_entity LIMIT 0, 1000	5 row(s) returned
✓ 31	12:10:43	SELECT * FROM sys.recipe_entity LIMIT 0, 1000	3 row(s) returned
✓ 32	12:11:29	SELECT * FROM sys.ingredient_entity LIMIT 0, 1000	7 row(s) returned
✓ 33	12:13:17	SELECT * FROM sys.recipe_entity LIMIT 0, 1000	3 row(s) returned
✓ 34	12:13:23	SELECT * FROM sys.ingredient_entity LIMIT 0, 1000	7 row(s) returned

CATEGORY TABLE

The screenshot displays a database management interface. On the left, a tree view shows the database structure under 'sys', including tables like 'category_entity', 'ingredient_entity', 'recipe_category', 'recipe_entity', and 'sys_config'. The 'category_entity' table is selected, and its details are shown in the 'Administration' tab. The table has two columns: 'categoryid' (bigint AI PK) and 'categoryname' (varchar(255)).

The 'Result Grid' shows the data for the 'category_entity' table:

categoryid	categoryname
1	Junk Food
2	Heavy Food
3	Light Food
4	Pastry/Bakery Items
NULL	NULL

The 'Output' tab shows the results of a series of SQL queries. The queries are all 'SELECT * FROM sys.category_entity LIMIT 0, 1000', and the results show 5 row(s) returned for each query.

#	Time	Action	Message
30	12:08:18	SELECT * FROM sys.category_entity LIMIT 0, 1000	5 row(s) returned
31	12:10:43	SELECT * FROM sys.recipe_entity LIMIT 0, 1000	3 row(s) returned
32	12:11:29	SELECT * FROM sys.ingredient_entity LIMIT 0, 1000	7 row(s) returned
33	12:13:17	SELECT * FROM sys.recipe_entity LIMIT 0, 1000	3 row(s) returned
34	12:13:23	SELECT * FROM sys.ingredient_entity LIMIT 0, 1000	7 row(s) returned
35	12:15:33	SELECT * FROM sys.category_entity LIMIT 0, 1000	4 row(s) returned

RECIPE_CATEGORY- table which contain two fields recipe_id and category_id to maintain many to many relations between recipe_entity and category_entity.

The screenshot displays a database management interface. On the left, a tree view shows the database structure under 'sys', including tables like category_entity, ingredient_entity, recipe_category, recipe_entity, and sys_config. The 'recipe_category' table is selected, and its schema is shown in the bottom left: recipe_id (bigint) and category_id (bigint).

The main area shows a 'Result Grid' with the following data:

	recipe_id	category_id
▶	2	4
	2	1
	1	1
	1	2
	4	3

Below the result grid, the 'Output' tab shows a list of actions and their results:

#	Time	Action	Message
✓ 34	12:13:23	SELECT * FROM sys.ingredient_entity LIMIT 0, 1000	7 row(s) returned
✓ 35	12:15:33	SELECT * FROM sys.category_entity LIMIT 0, 1000	4 row(s) returned
✓ 36	12:15:41	SELECT * FROM sys.recipe_category LIMIT 0, 1000	4 row(s) returned
✓ 37	12:16:22	SELECT * FROM sys.category_entity LIMIT 0, 1000	4 row(s) returned
✓ 38	12:16:28	SELECT * FROM sys.recipe_entity LIMIT 0, 1000	3 row(s) returned
✓ 39	12:17:37	SELECT * FROM sys.recipe_category LIMIT 0, 1000	5 row(s) returned

GETTING INGREDIENTS BY RECIPE_ID

The screenshot shows the Postman API client interface. The top navigation bar includes 'Home', 'Workspaces', and 'Explore'. A search bar is present with the text 'Search Postman'. A notification banner states: 'You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.' The left sidebar shows a 'History' panel with a list of recent requests. The main panel displays a GET request to 'http://localhost:1974/ingredients/byRecipe/2'. The response is a JSON array of ingredients for recipe 2, including Maid Flour, Sugar, Eggs, and Coco Powder. The status is 200 OK, and the response size is 476 B.

History

- GET http://localhost:1974/ingredients/byRecipe/2
- GET http://localhost:1974/ingredients/3
- GET http://localhost:1974/ingredients
- GET http://localhost:1974/categories
- POST http://localhost:1974/categories
- POST http://localhost:1974/categories
- POST http://localhost:1974/categories
- POST http://localhost:1974/recipes/3/ingredients
- POST http://localhost:1974/recipes/3/ingredients
- POST http://localhost:1974/recipes/3/ingredients
- POST http://localhost:1974/recipes/2/ingredients
- POST http://localhost:1974/recipes/2/ingredients
- POST http://localhost:1974/recipes/2/ingredients
- POST http://localhost:1974/recipes/2/ingredients
- POST http://localhost:1974/recipes/1/ingredients
- POST http://localhost:1974/recipes/1/ingredients
- POST http://localhost:1974/recipes/1/ingredients
- GET http://localhost:1974/recipes/2
- GET http://localhost:1974/recipes

GET http://localhost:1974/ingredients/byRecipe/2

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

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 157 ms Size: 476 B Save Response

```
[{"ingredientid": 4, "ingredientname": "Maid Flour", "ingredientamount": "500 Grams"}, {"ingredientid": 5, "ingredientname": "Sugar", "ingredientamount": "500 Grams"}, {"ingredientid": 6, "ingredientname": "Eggs", "ingredientamount": "Half Dozen"}, {"ingredientid": 7, "ingredientname": "Coco Powder", "ingredientamount": "200 Grams"}]
```

GETTING CATEGORIES BY RECIPE_ID

The screenshot displays the Postman interface with a GET request to `http://localhost:1974/categories/byRecipe/1`. The response is a JSON array with two objects, each containing `categoryid` and `categoryname`.

Request Details:

- Method: GET
- URL: `http://localhost:1974/categories/byRecipe/1`

Response Details:

- Status: 200 OK
- Time: 48 ms
- Size: 254 B

Response Body (JSON):

```
[
  {
    "categoryid": 1,
    "categoryname": "Junk Food"
  },
  {
    "categoryid": 2,
    "categoryname": "Heavy Food"
  }
]
```

History List:

- GET `http://localhost:1974/categories/byRecipe/1`
- GET `http://localhost:1974/categories/3`
- GET `http://localhost:1974/categories`
- GET `http://localhost:1974/ingredients/byRecipe/2`
- GET `http://localhost:1974/ingredients/3`
- GET `http://localhost:1974/ingredients`
- GET `http://localhost:1974/recipes/byCategory/1`
- GET `http://localhost:1974/recipes/byCategory/4`
- GET `http://localhost:1974/recipes/byIngredient/5`
- GET `http://localhost:1974/recipes`
- GET `http://localhost:1974/recipes/3`
- GET `http://localhost:1974/recipes`
- POST `http://localhost:1974/recipes`
- POST `http://localhost:1974/recipes/3/categories`
- POST `http://localhost:1974/recipes/1/categories`
- POST `http://localhost:1974/recipes/3/categories`
- POST `http://localhost:1974/recipes/2/categories`
- POST `http://localhost:1974/recipes/1/categories`
- POST `http://localhost:1974/recipes/2/categories`
- GET `http://localhost:1974/ingredients/byRecipe/2`
- GET `http://localhost:1974/ingredients/3`

GETTING RECIPES BY CATEGORY_ID

The screenshot displays the Postman API client interface. At the top, there's a navigation bar with 'Home', 'Workspaces', and 'Explore' options, along with a search bar and user account links. A notification banner indicates the user is using the Lightweight API Client. The left sidebar shows a 'History' panel with a list of recent requests. The main workspace is configured for a GET request to `http://localhost:1974/recipes/byCategory/1`. The 'Body' tab is selected, showing a JSON response with three recipe entries. The status bar at the bottom indicates a successful 200 OK response.

History

- Today
 - GET `http://localhost:1974/recipes/byCategory/1`
 - GET `http://localhost:1974/recipes/byCategory/4`
 - GET `http://localhost:1974/recipes/byIngredient/5`
 - GET `http://localhost:1974/recipes`
 - GET `http://localhost:1974/recipes/3`
 - GET `http://localhost:1974/recipes`
 - POST `http://localhost:1974/recipes`
 - POST `http://localhost:1974/recipes/3/categories`
 - POST `http://localhost:1974/recipes/1/categories`
 - POST `http://localhost:1974/recipes/3/categories`
 - POST `http://localhost:1974/recipes/2/categories`
 - POST `http://localhost:1974/recipes/1/categories`
 - POST `http://localhost:1974/recipes/2/categories`
 - GET `http://localhost:1974/ingredients/byRecipe/2`
 - GET `http://localhost:1974/ingredients/3`
 - GET `http://localhost:1974/ingredients`
 - GET `http://localhost:1974/categories`
 - POST `http://localhost:1974/categories`
 - POST `http://localhost:1974/categories`
 - POST `http://localhost:1974/categories`
 - POST `http://localhost:1974/categories`

GET `http://localhost:1974/recipes/byCategory/1`

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

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

1

Body Cookies Headers (5) Test Results Status: 200 OK Time: 34 ms Size: 528 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 2,
4     "name": "CAKE",
5     "description": "Add maida flour, sugar, coco powder, eggs. Mix it well. Keep it in oven. Decorate it with cream"
6   },
7   {
8     "id": 1,
9     "name": "PIZZA",
10    "description": "Add flour, salt, cheese. Mix it well. Keep it in oven. Decorate it with vegies"
11  },
12  {
13    "id": 3,
14    "name": "CHAT",
15    "description": "Add tomato, onion, puffrice, chat powder. Mix it well. Serve and eat."
16  }
17 ]
```


GETTING RECIPE BY INGREDIENT_ID

The screenshot displays the Postman interface with a history of requests on the left and a detailed view of a GET request on the right.

History:

- GET http://localhost:1974/recipes/byIngredient/5
- POST http://localhost:1974/recipes/4/categories
- POST http://localhost:1974/recipes/4/catgeories
- POST http://localhost:1974/recipes/4/ingredients
- POST http://localhost:1974/recipes/4/ingredients
- POST http://localhost:1974/recipes/4/ingredients
- PUT http://localhost:1974/recipes/4
- DEL http://localhost:1974/categories/5
- DEL http://localhost:1974/categories/6
- POST http://localhost:1974/categories/6
- POST http://localhost:1974/categories
- POST http://localhost:1974/categories
- DEL http://localhost:1974/ingredients/12
- DEL http://localhost:1974/recipes/3
- PUT http://localhost:1974/categories/4
- PUT http://localhost:1974/ingredients/12
- POST http://localhost:1974/ingredients/12
- POST http://localhost:1974/ingredients
- PUT http://localhost:1974/recipes/4
- PUT http://localhost:1974/recipes/3
- GET http://localhost:1974/categories/byRecipe/1

Active Request: GET http://localhost:1974/recipes/byIngredient/5

Response: Status: 200 OK, Time: 348 ms, Size: 300 B

```
1 {
2   {
3     "id": 2,
4     "name": "CAKE",
5     "description": "Add maida flour, sugar, coco powder, eggs. Mix it well. Keep it in oven. Decorate it with cream"
6   }
7 }
```

CONCLUSION

The main purpose of this project is to create the recipes, upload them in the database, update, delete them. All these recipes are categorized. The main entities in recipe book are recipes, ingredients and categories.

Java Spring Boot is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities: Autoconfiguration, an opinionated approach to configuration, the ability to create standalone applications. Spring Boot helps developers create applications that *just run*. Specifically, it lets you create standalone applications that run on their own, without relying on an external web server, by embedding a web server such as Tomcat or Netty into your app during the initialization process. As a result, you can launch your application on any platform by simply hitting the Run command.

In conclusion, the recipe book is created by using java spring boot because, there is no external server required and it also helps with auto configuration.

“ GOOD FOOD ENDS WITH GOOD TALK “

- Jules Renard

THANK YOU

