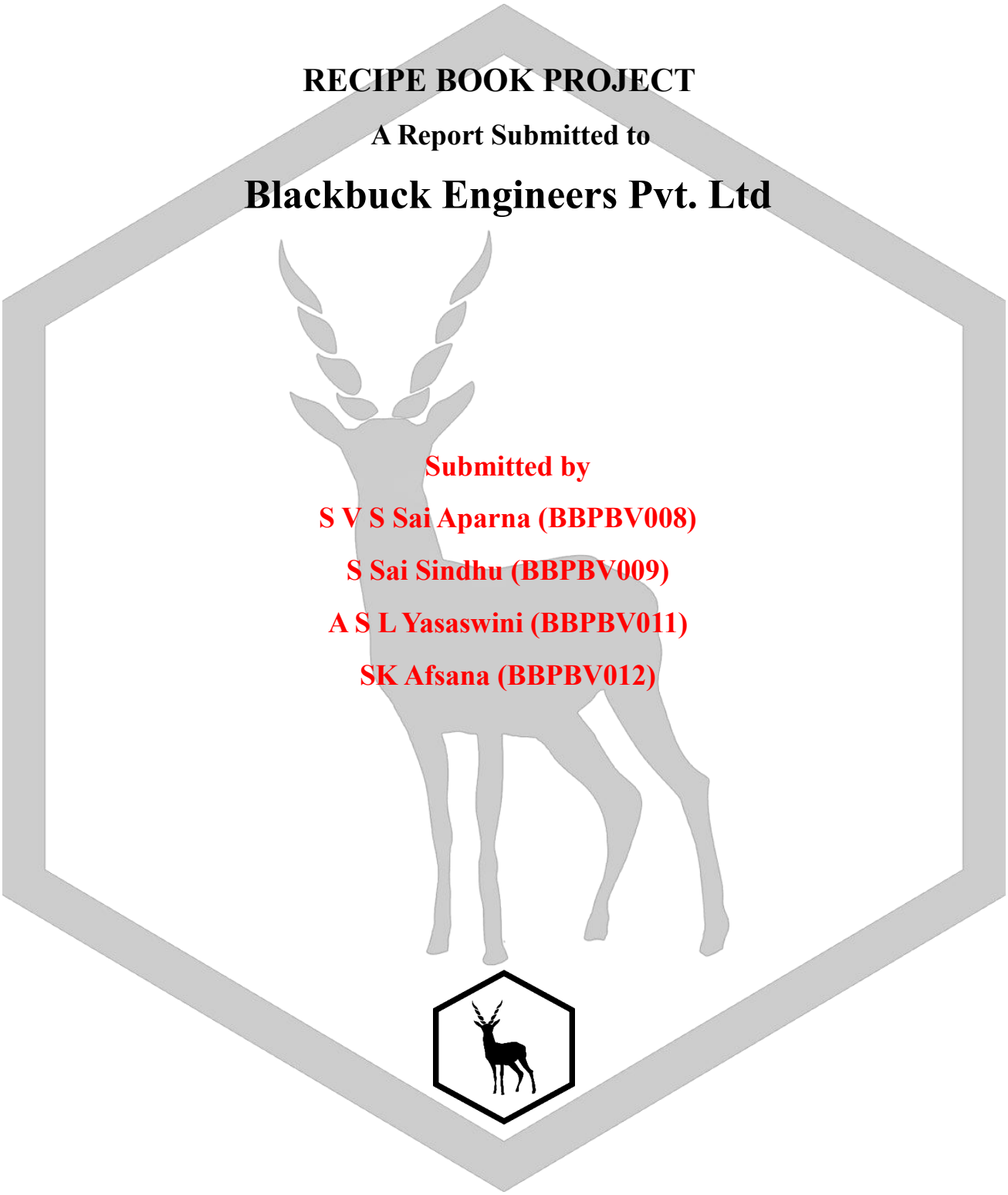
AN INTERNSHIP REPORT ON

RECIPE BOOK PROJECT

A Report Submitted to

Blackbuck Engineers Pvt. Ltd



Submitted by

S V S Sai Aparna (BBPBV008)

S Sai Sindhu (BBPBV009)

A S L Yasaswini (BBPBV011)

SK Afsana (BBPBV012)

Blackbuck Engineers Pvt. Ltd

Road no 36, Jubilee Hills, Hyderabad

CERTIFICATE

TABLE OF CONTENTS

ACKNOWLEDGEMENT

We would like to acknowledge all those without whom this project would not have been successful. Firstly, we would like to thank our Internet and Java & FSD Programmer Mr. Ramakrishna Sir who guided us throughout the project and gave his immense support. He made us understand how to complete this project and without his presence, this project would not have been complete. We also got to know a lot about parallelization and its benefits. This Project has been a source to learn and bring our theoretical knowledge to the real-life world. Once again, thanks to everyone for making this project successful.

Place :

Date :

ABSTRACT

The Recipe Book System is a Java-based web application developed using Spring Boot. It offers users a user-friendly platform to create, manage, and discover recipes. With features like recipe management, search capabilities, user profiles, and scalability, it simplifies cooking while fostering a sense of community among cooking enthusiasts.It empowers users to streamline their cooking adventures, explore new flavors, and connect with a vibrant community of like-minded food enthusiasts.

Welcome to the future of cooking.The Recipe Book System is a versatile and user-friendly web application that caters to the needs of both novice and experienced cooks. With its focus on recipe management, discovery, and community engagement, it aims to simplify the cooking experience and promote culinary exploration in the digital age.

1. INTRODUCTION

1.1.SYSTEM OVERVIEW

The Recipe Book Management System is a robust and user-centric web application developed on the Spring Boot framework, leveraging Java's power and versatility. . It simplifies the culinary journey by providing an elegant interface for recipe management and an intelligent search engine for exploring an extensive collection of culinary delights. Users can create personalized profiles to save their favorite recipes, share their own creations, and engage with a vibrant community of food enthusiasts. Designed with scalability in mind, the system adapts to the evolving needs of both novice and experienced cooks, making it the ultimate digital companion for culinary exploration and innovation.

Hence,the main objective of this intuitive platform is to empower users to effortlessly create, organize, and discover recipes.

1.2.OBJECTIVE

The primary objective of the Recipe Book System is to offer a comprehensive and user-friendly platform for culinary enthusiasts. This project aims to simplify the process of recipe creation, organization, and discovery while fostering a vibrant community of food lovers. Our goal is to provide users, whether they are novice cooks or seasoned chefs, with a seamless and enjoyable experience in the world of cooking.

Through intuitive recipe management, robust search capabilities, and personalized user profiles, we aim to empower users to streamline their culinary adventures, explore new flavors, and connect with like-minded individuals. Additionally, the project is designed with scalability in mind, ensuring its adaptability to accommodate a growing user base and future enhancements. Ultimately, our objective is to redefine the cooking experience in the digital age, making it more accessible, engaging, and enjoyable for all.

1.3.PROBLEM IDENTIFICATION

There are some potential problems that can be faced or identified in the Recipe Book System:

- Usability Challenges
- Performance Issues
- Data Management
- Security Concerns
- Content Quality
- Scalability
- Community Management
- Mobile Responsiveness
- Integration Challenges

- User Engagement
- Content Curation
- Feedback Handling

1.4.APPLICATIONS

The potential applications of the Recipe Book System

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

1.5.LITERATURE SURVEY

1).Introduction to Recipe Management and Culinary Applications**

- Provide an overview of the importance of recipe management in the culinary domain.
- Explain the relevance of a Recipe Book System in simplifying cooking and fostering culinary exploration.
- Set the context for the literature survey.

2). Existing Recipe Management Solutions

- Review and analyze popular existing recipe management software, both web-based and desktop applications.
- Highlight key features, strengths, and weaknesses of these solutions.
- Discuss how they address the needs of culinary enthusiasts.

3). Culinary Communities and User Engagement Platforms

- Explore culinary communities and platforms where users share recipes, cooking tips, and engage in discussions.
- Discuss the role of community engagement and user-generated content in culinary applications.

4). Mobile Applications for Culinary Enthusiasts

- Investigate mobile applications related to cooking, recipe discovery, and meal planning.
- Examine trends in mobile app development for culinary enthusiasts and the user experience they offer.

5). E-commerce Integration and Online Culinary Marketplaces

- Examine how e-commerce integration with recipe platforms and meal planning apps has impacted the culinary industry.
- Discuss the convenience of purchasing ingredients directly through recipe applications.

6). Nutrition Tracking and Dietary Applications

- Explore applications that focus on nutrition tracking, dietary restrictions, and meal recommendations.
- Discuss the importance of nutritional information in recipe management and meal planning.

7). User-Generated Content and Moderation

- Review best practices for handling user-generated content, including recipe submissions and community engagement.
- Examine strategies for content moderation and quality control to maintain a positive user experience.

8). Technological Frameworks and Development Tools

- Investigate the use of specific technological frameworks and tools commonly employed in developing recipe management systems.
- Discuss the advantages and limitations of various technologies in this context.

9). Data Security and Privacy Considerations

- Highlight data security and privacy concerns relevant to recipe management systems, especially when handling user-generated content and personal information.

10). Machine Learning and AI Integration

- Explore how machine learning and artificial intelligence are leveraged to enhance recipe recommendations and provide personalized culinary experiences.

11). Emerging Trends and Future Directions

- Identify emerging trends and potential future directions in recipe management and culinary applications, such as voice assistants and augmented reality.

1.6.LIMITATIONS

- Data Accuracy: The accuracy of recipe data largely depends on user contributions. Inaccurate or incomplete recipes may be uploaded, impacting the quality of the content.

- Content Moderation: Ensuring the quality and appropriateness of user-generated content can be challenging and time-consuming. Inappropriate or spammy submissions may need to be addressed.

- Scalability: As the user base and recipe collection grow, the system's infrastructure and performance may need to be scaled to handle increased demand.

- Privacy Concerns: Handling user data and personal recipes while adhering to privacy regulations requires careful consideration and robust data protection measures.

2. SYSTEM ANALYSIS

System analysis is a crucial phase in the development of the Recipe Book System project. It involves a detailed examination of the requirements, objectives, and constraints of the project to ensure that the system is designed to meet its intended goals effectively. Here are the key components and steps you can include in the system analysis for your project:

Certainly, let's provide specific details and context for each section of the system analysis in the Recipe Book System project documentation:

2.1 Existing System Analysis

Overview:Currently, recipe management relies on a variety of standalone apps and websites, making it challenging for users to streamline their cooking experiences and connect with a culinary community.

Strengths and Weaknesses: Existing systems offer recipe storage but lack robust community engagement features. They often have limited user interface options and insufficient personalization.

User Feedback: User feedback indicates frustration with scattered recipe management and a desire for a more user-friendly and interactive platform.

2.2 Proposed System Analysis

Objectives and Goals: The proposed Recipe Book System aims to create a centralized platform that simplifies recipe management, fosters community interaction, and enhances culinary exploration.

Functional Requirements: Key features include recipe creation, browsing, and searching, user profiles, social interactions, nutritional information, and personalized recommendations.

Non-Functional Requirements: The system must ensure data security, high performance, scalability, and an intuitive user interface.

User Interface Design: The system will feature a modern, intuitive, and responsive user interface with easy navigation and visually appealing recipe displays.

Data Management: Recipes, user profiles, and interactions will be stored in a relational database with efficient indexing for fast retrieval.

Security Measures: Robust user authentication and authorization will be implemented, along with encryption to protect user data.

Performance Considerations: Caching and optimization techniques will be applied to ensure rapid response times.

User Authentication: Users will be able to sign up, log in, and manage their profiles securely.

Integration Requirements: Integration with social media platforms will enable users to share their culinary experiences.

Compliance and Privacy: The system will comply with GDPR and other data protection regulations.

Scalability Plan: Cloud-based hosting will allow for seamless scalability as the user base grows.

User Experience (UX) Design: User feedback has influenced a user-centered design with features like intuitive search and recipe personalization.

Documentation Plan: Comprehensive documentation will include user guides, API documentation, and development documentation.

Cost Analysis: A budget allocation of resources for development, hosting, and maintenance has been established.

Risk Assessment: Potential risks such as data breaches and user adoption challenges have been identified, with mitigation strategies in place.

2.2.3. Client-Side Benefits

User-Friendly Interface: The intuitive user interface will allow users to easily manage and explore recipes.

Recipe Organization: Users can categorize, search, and filter recipes for efficient organization.

Personalization: Personalized recipe recommendations and dietary preferences enhance the culinary journey.

Recipe Discovery: Users can discover new recipes based on their preferences and previous interactions.

Community Engagement: Features like user reviews, comments, and social sharing promote interaction among users.

Nutritional Information: Nutritional details are available for each recipe, supporting healthier choices.

2.2.4 Sponsor-Side Benefits

Return on Investment: Sponsors can benefit from increased brand exposure and potential revenue generation through strategic partnerships.

Brand Exposure: Sponsorship of the Recipe Book System project can significantly boost brand visibility among culinary enthusiasts.

2.2.5 Company-Side Benefits

User Engagement: The project is expected to drive user engagement, increase the user base, an encourage user-generated content.

Revenue Generation: Revenue will be generated through premium subscriptions and targeted advertising opportunities.

Brand Reputation: The project will enhance the company's reputation as an innovator in the culinary tech space.

2.2.6 Benefits of Proposed System

Improved Recipe Management: The Recipe Book System simplifies recipe management by offering a centralized platform with robust features.

Community Engagement: Users can connect with a culinary community, share their expertise, and discover new culinary experiences.

Culinary Exploration: Personalized recommendations and easy access to nutritional information encourage users to explore new recipes and cuisines.

3. REQUIREMENT SPECIFICATION

3.1.HARDWARE REQUIREMENTS

- Laptop 4GHz minimum, multi-core processor
- Memory (RAM)4GB,preferably higher, and commensurate with concurrent usage
- Hard disk space 1TB

3.2.SOFTWARE REQUIREMENTS

- Windows 2016
- Visual Studio code
- Eclipse IDE
- Apache Tomcat web server
- Spring boot

3.3.COMPONENTS USED

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Safari

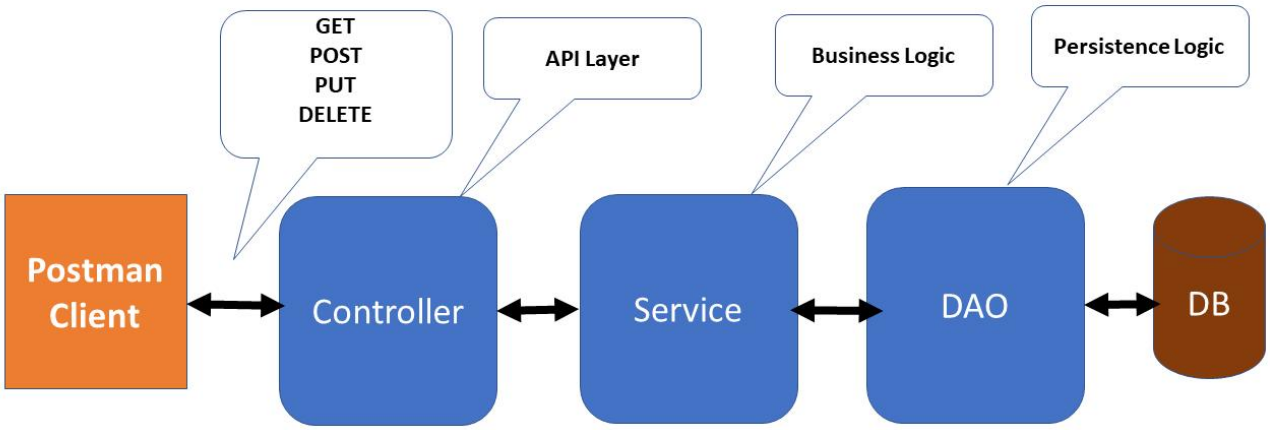
4. ARCHITECTURE DESIGN SPECIFICATION

4.1.SYSTEM ARCHITECTURE

There are **four** layers in Spring Boot are as follows:

1. presentation layer
2. Bussiness layer
3. Persistence layer
4. Data base layer

Spring Boot Project Architecture



1. Presentation Layer

The presentation layer is the top layer of the spring boot architecture. It consists of Views. i.e., the front-end part of the application. It handles the HTTP requests and performs authentication. It is responsible for converting the JSON field's parameter to Java Objects and vice-versa. Once it performs the authentication of the request it passes it to the next layer. i.e., the business layer.

2. Business Layer

The business layer contains all the business logic. It consists of services classes. It is responsible for validation and authorization.

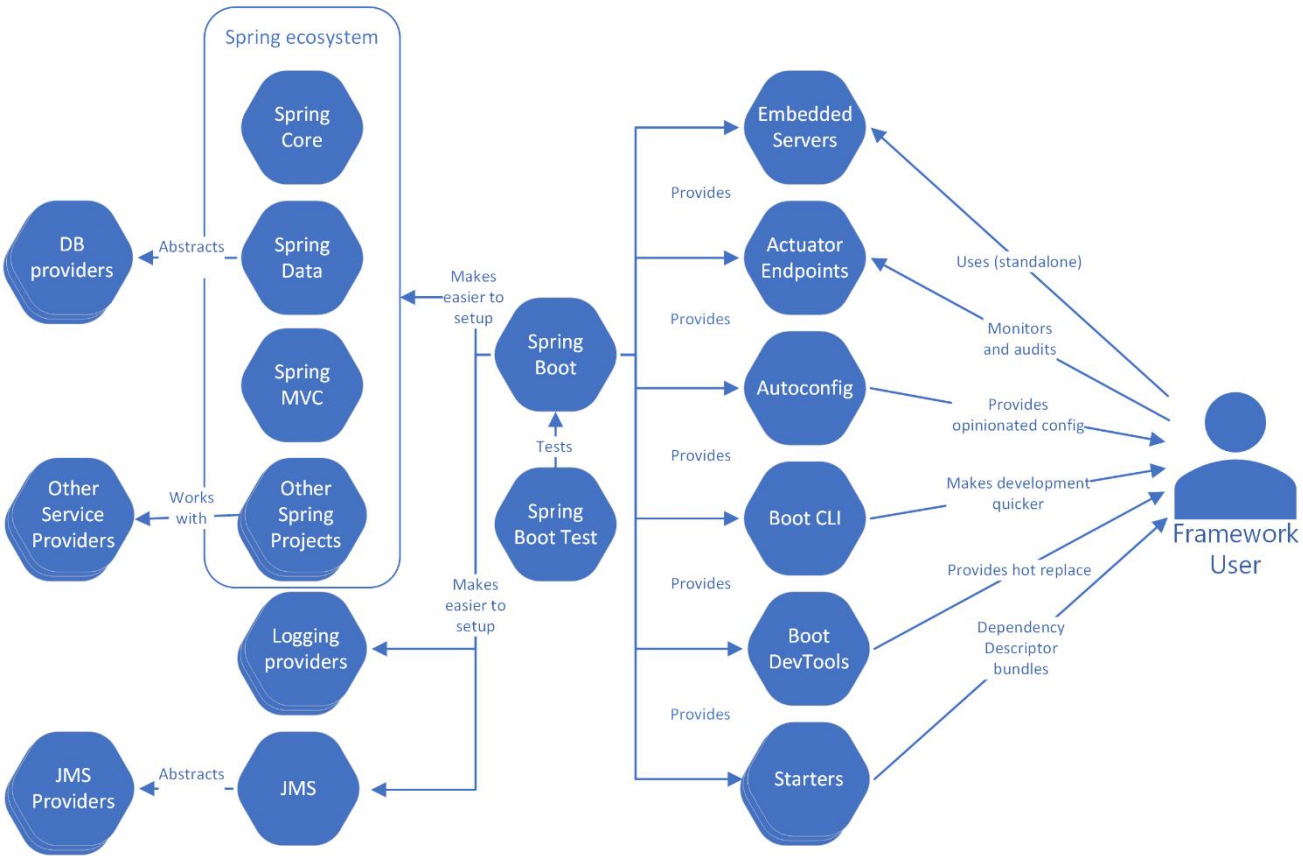
3. Persistence Layer

The persistence layer contains all the database storage logic. It is responsible for converting business objects to the database row and vice-versa.

4. Database Layer

The database layer contains all the databases such as MySql, MongoDB, etc. This layer can contain multiple databases. It is responsible for performing the CRUD operations.

4.2.DETAILED DESIGN



- Now we have validator classes, view classes, and utility classes.
- Spring Boot uses all the modules of Spring-like Spring MVC, Spring Data, etc. The architecture of Spring Boot is the same as the architecture of Spring MVC, except one thing: there is no need for **DAO** and **DAOImpl** classes in Spring boot.
- Creates a data access layer and performs CRUD operation.
- The client makes the HTTP requests (PUT or GET).
- The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required.
- In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes.
- A JSP page is returned to the user if no error occurred.

4.3.COMPONENTS USED

- HTML/CSS

- Java Script
- AJAX
- Mysql
- Apache Web server

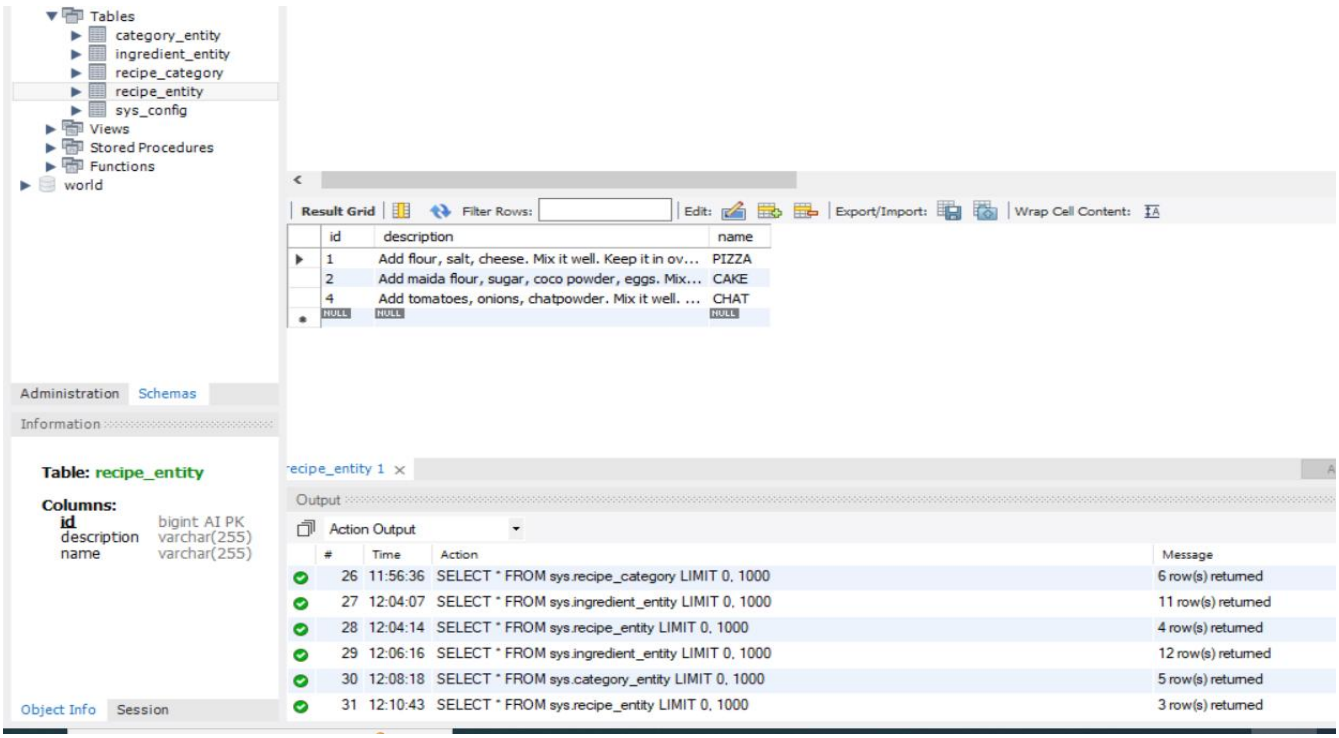
4.4.DATABASE DESIGN

->In this project,we have used three table:

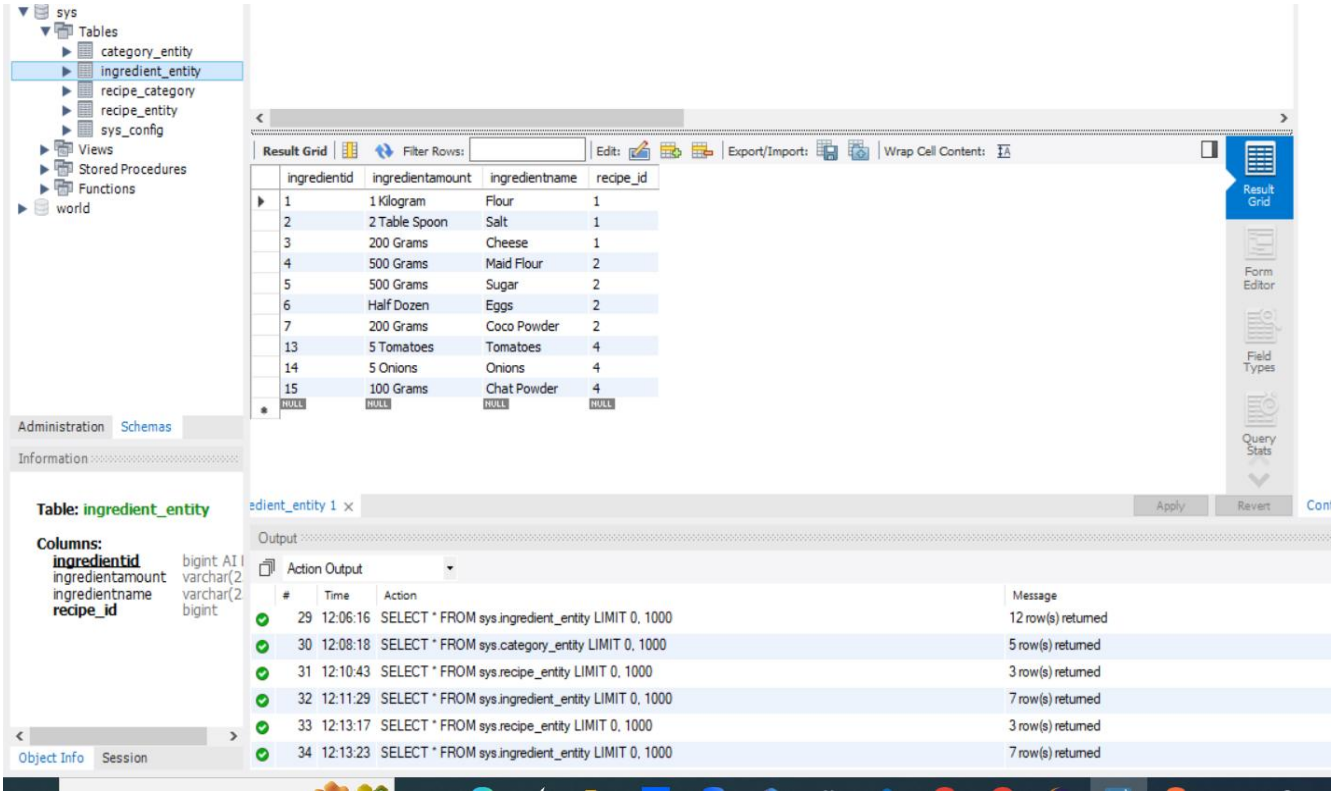
- Recipe
- Ingredient
- Category

TABLES:

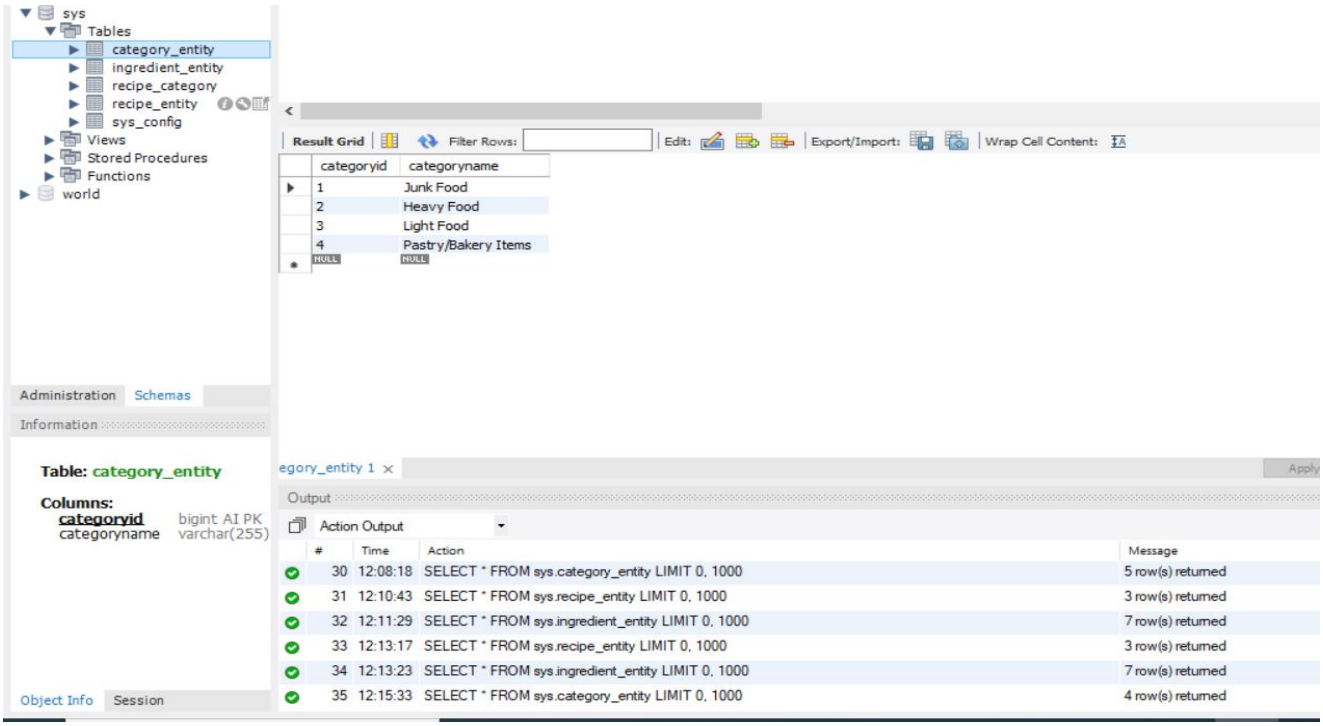
->recipe_entity - having one to many relation with ingredient_entity and many to many relation with category_entity.



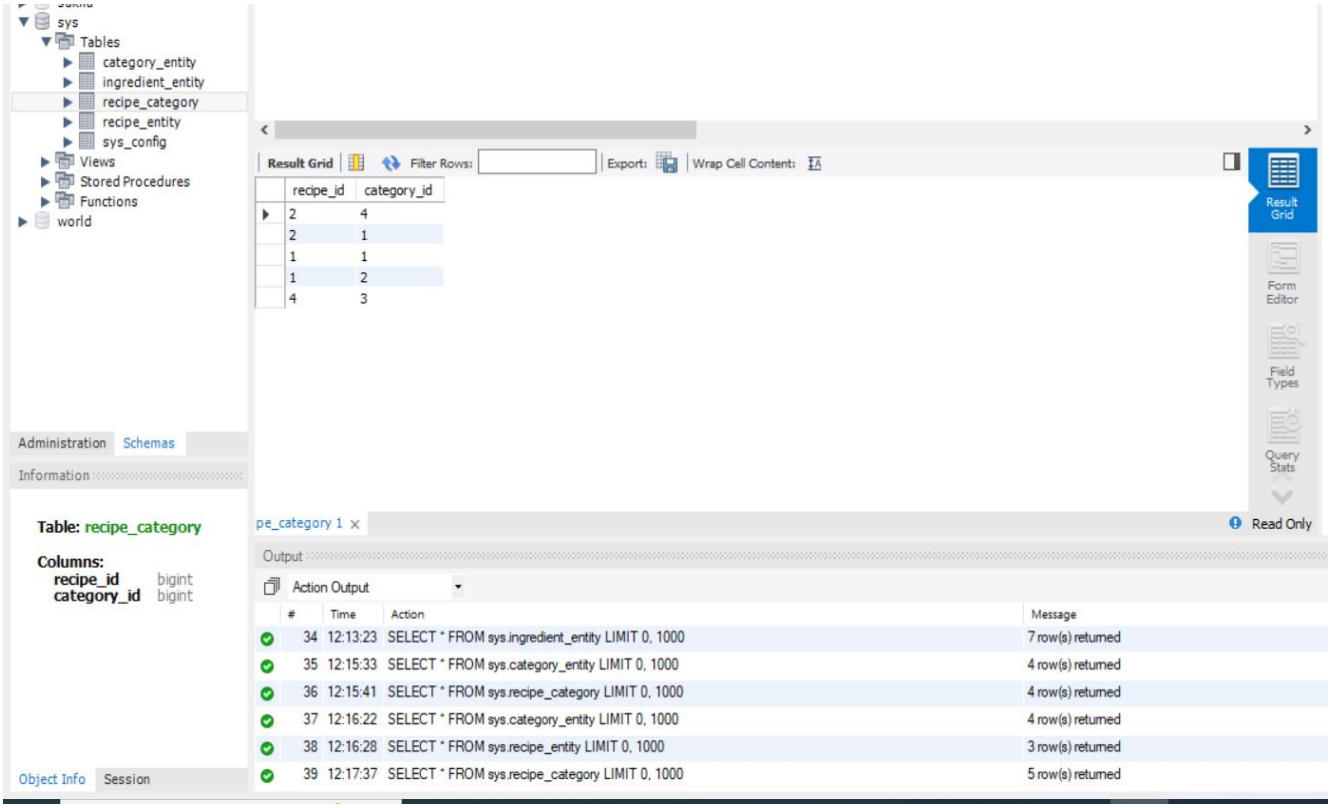
→ingrdient_entity - having many to one relation with recipe_entity



→category_entity - having many to many relation with recipe_entity



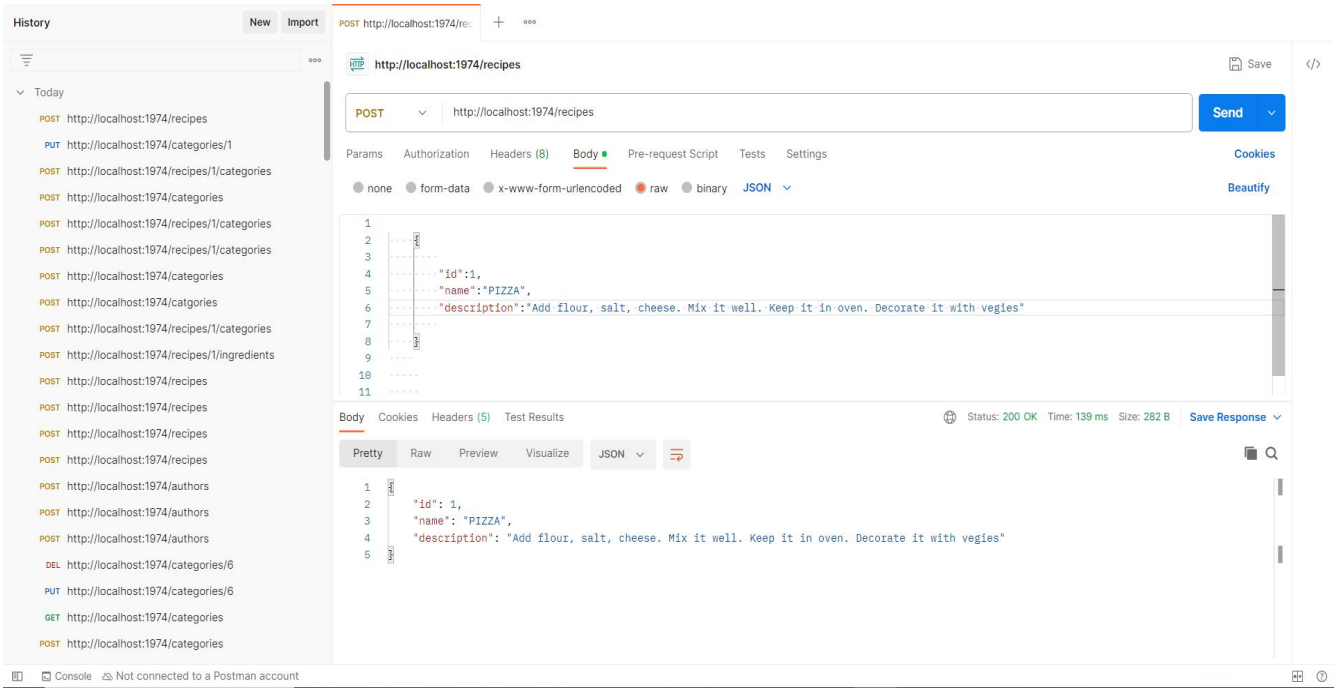
→recipe_category - table which contain two fields recipe_id and category_id to maintain many to many relations between recipe_entity and category_entity.



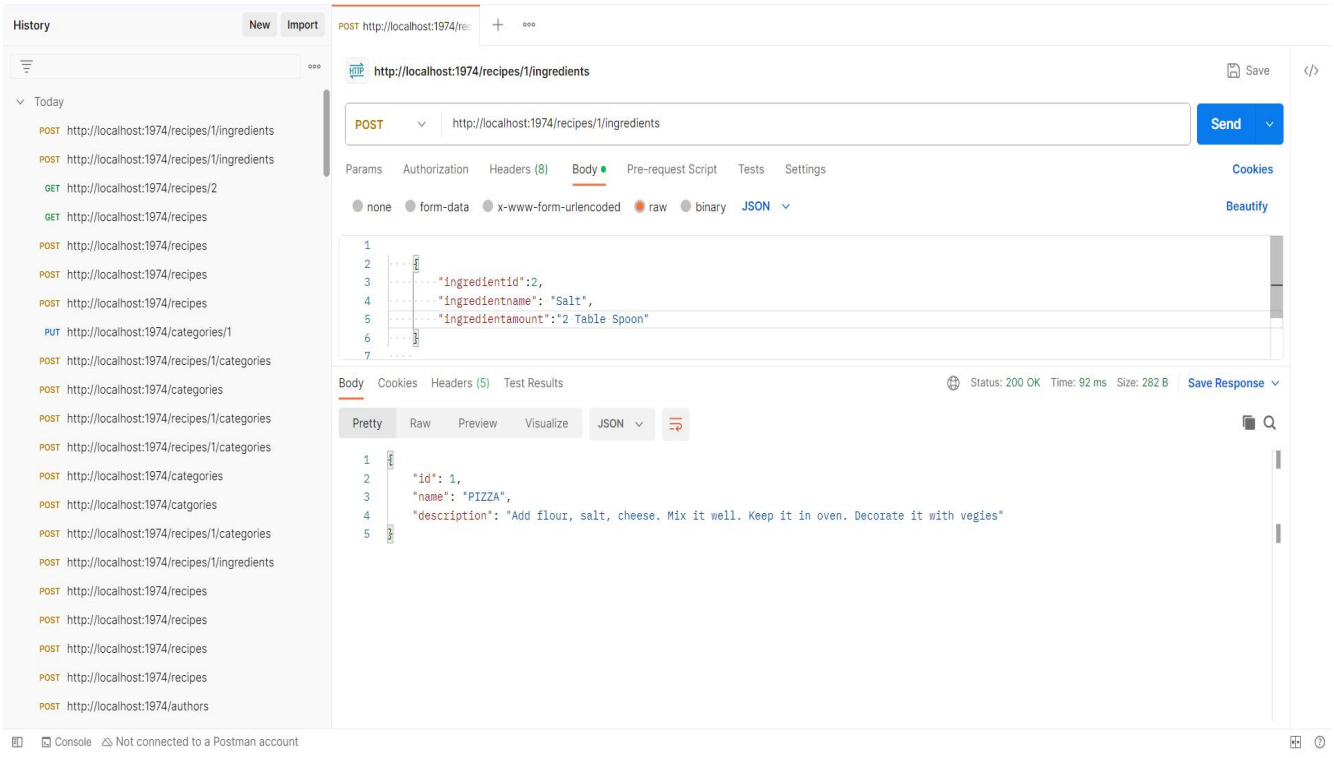
5. SYSTEM IMPLEMENTATION

5.1. RECIPE TABLE OPERATIONS

→POST OPERATION - http://localhost:[port number]/recipes

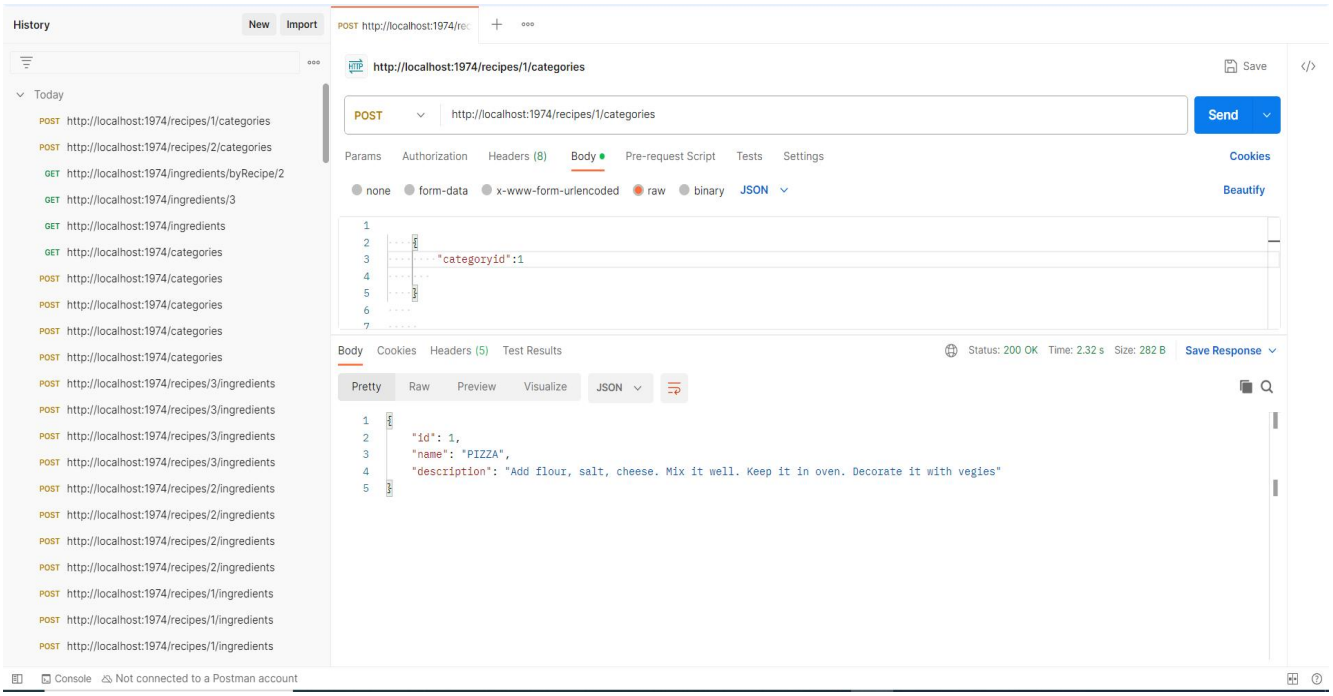


→POSTING INGREDIENTS USING RECIPE ID - http://localhost:[port number]/recipes/{recipeId}/ingredients

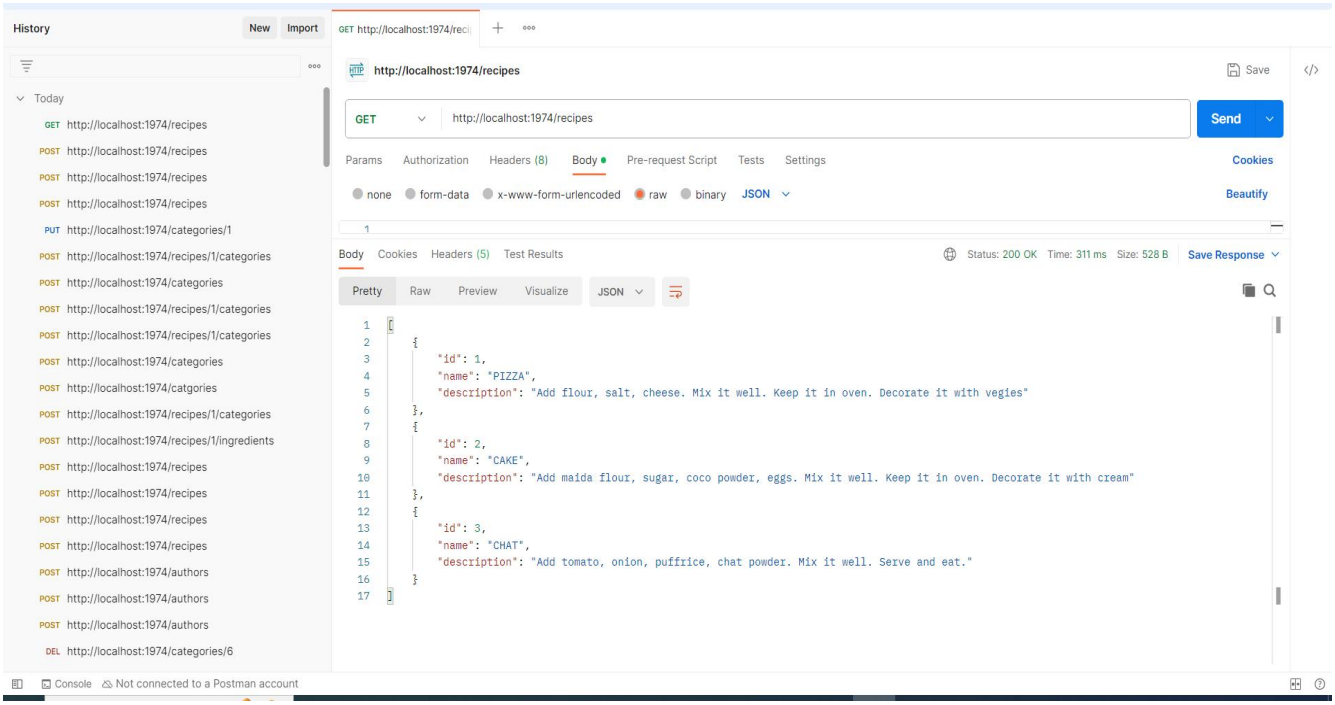


→ POST FOR CONNECTING CATEGORIES USING RECIPE ID -

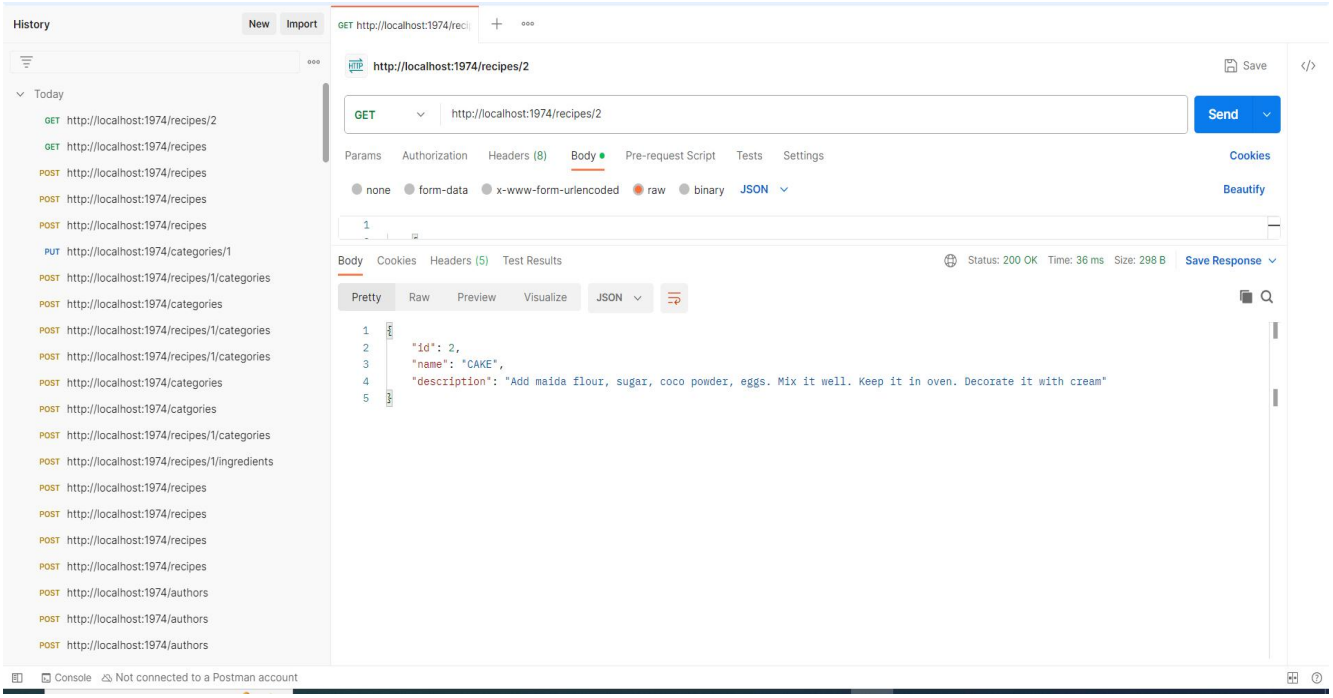
http://localhost:[port number]/recipes/{recipeId}/categories



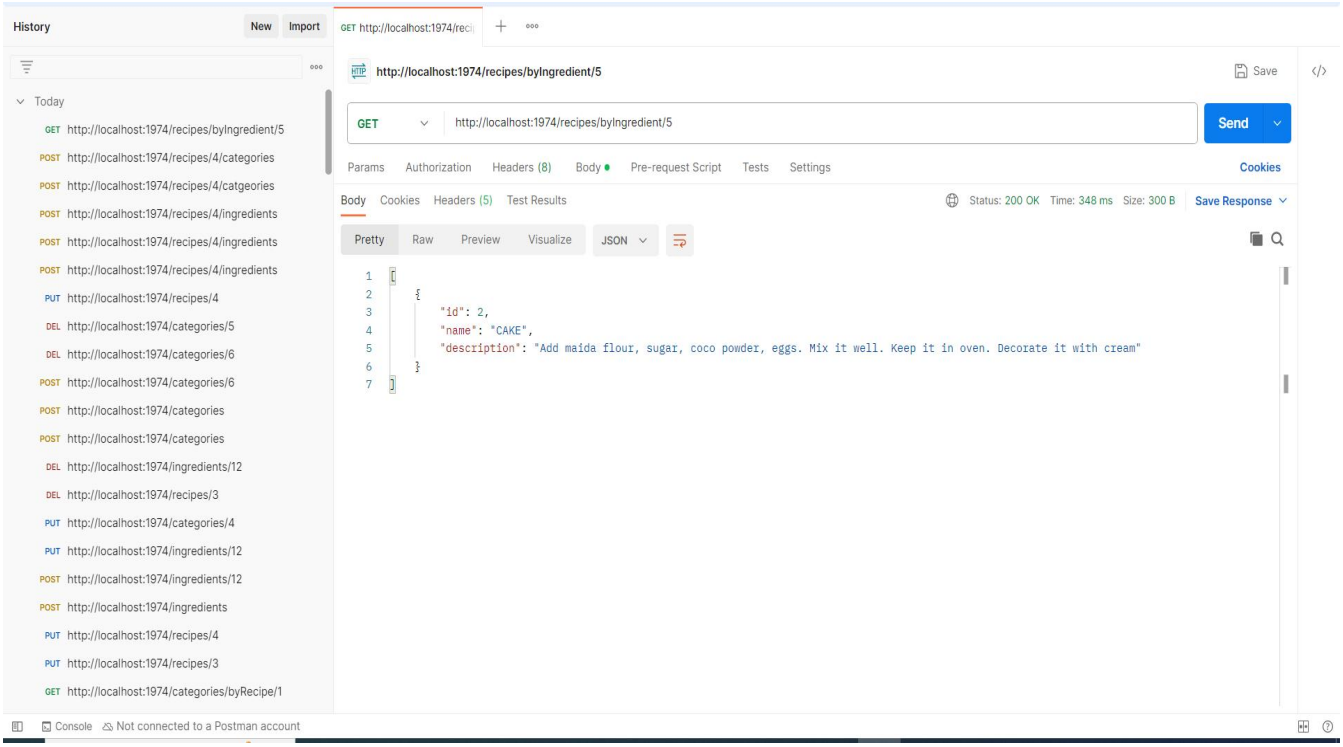
→**GET ALL RECIPES** - http://localhost:[port number]/recipes



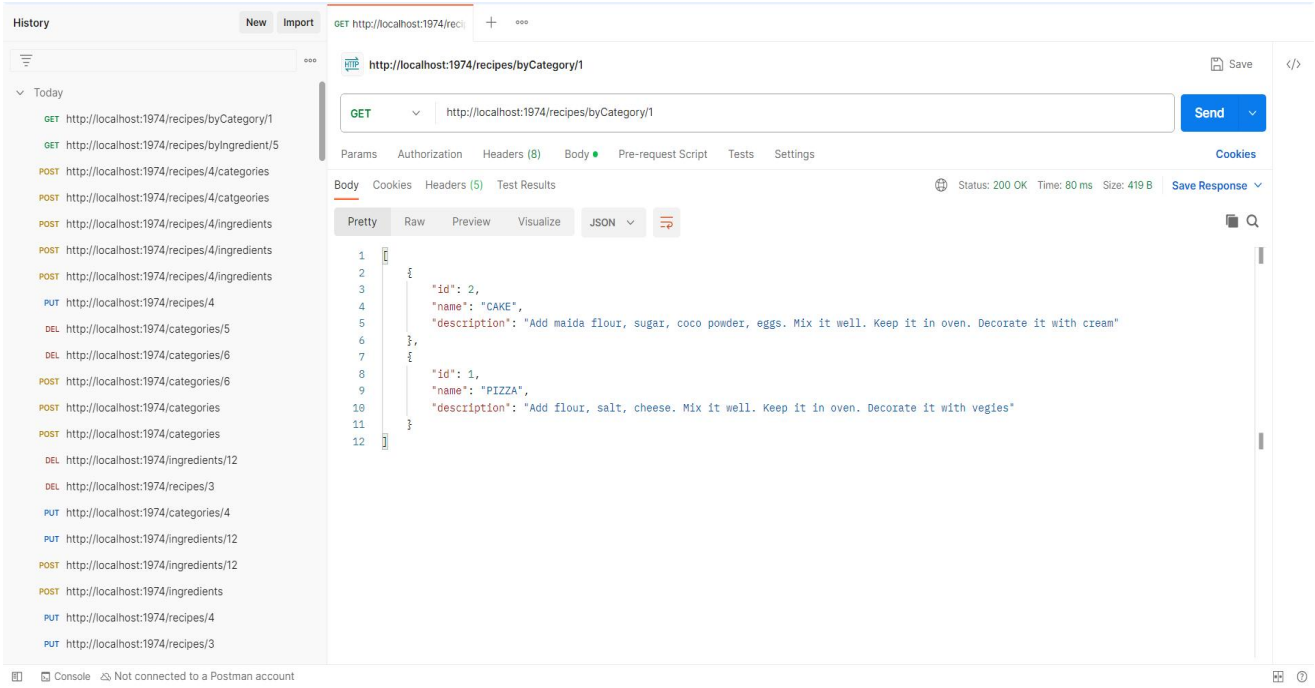
→**GET RECIPES USING RECIPE ID -** http://localhost:[port number]/recipes/{recipeId}



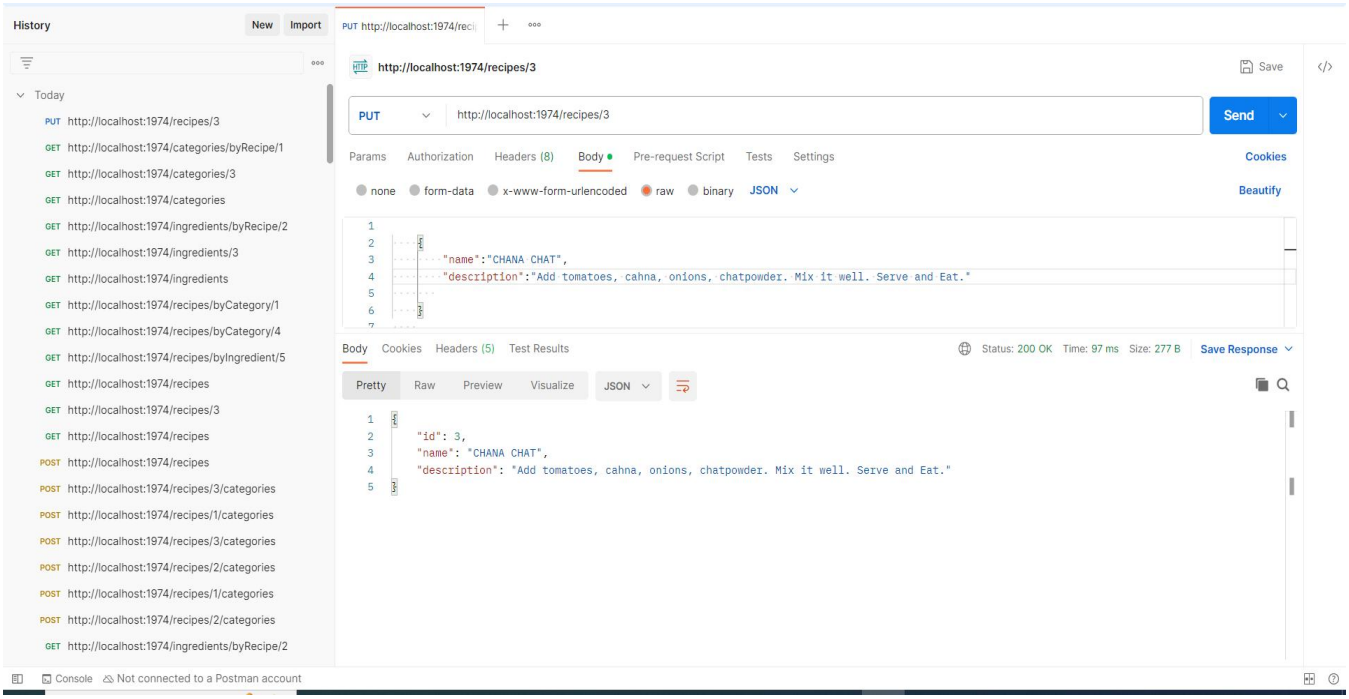
→ GET RECIPES USING INGREDIENT ID - http://localhost:[port number]/recipes/byIngredient/{ingredientid}



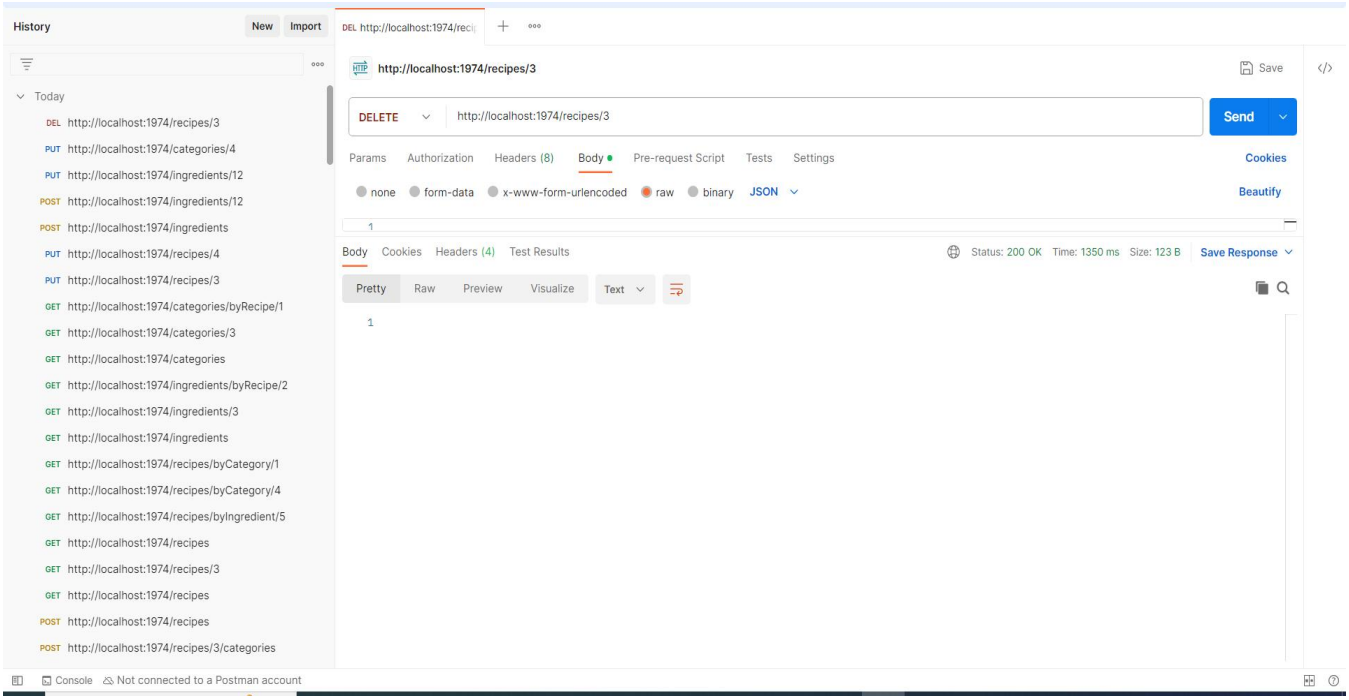
→GET RECIPES USING CATEGORY ID - http://localhost:[port number]/recipes/byCategory/{categoryid}



→ **PUT FOR UPDATING RECIPE DATA -** http://localhost:[port number]/recipes/{recipeId}

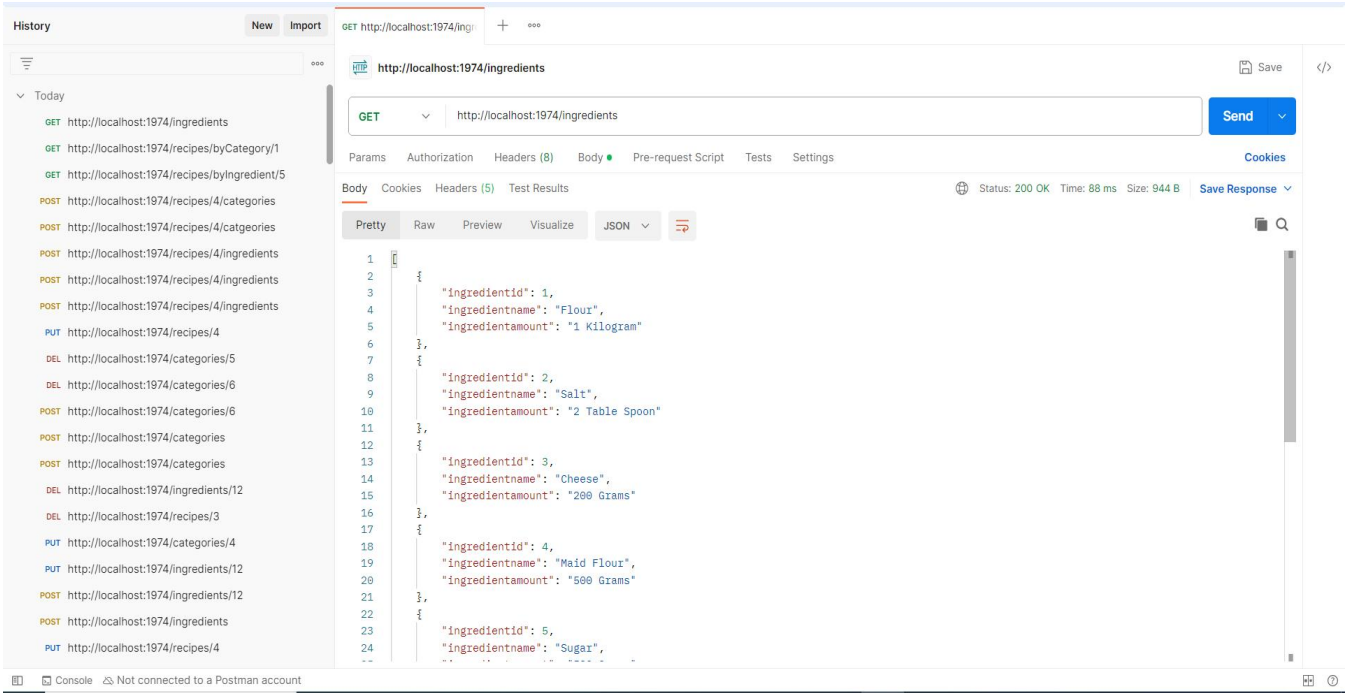


→ **DELETING A RECIPE -** http://localhost:[port number]/recipes/{recipeId}

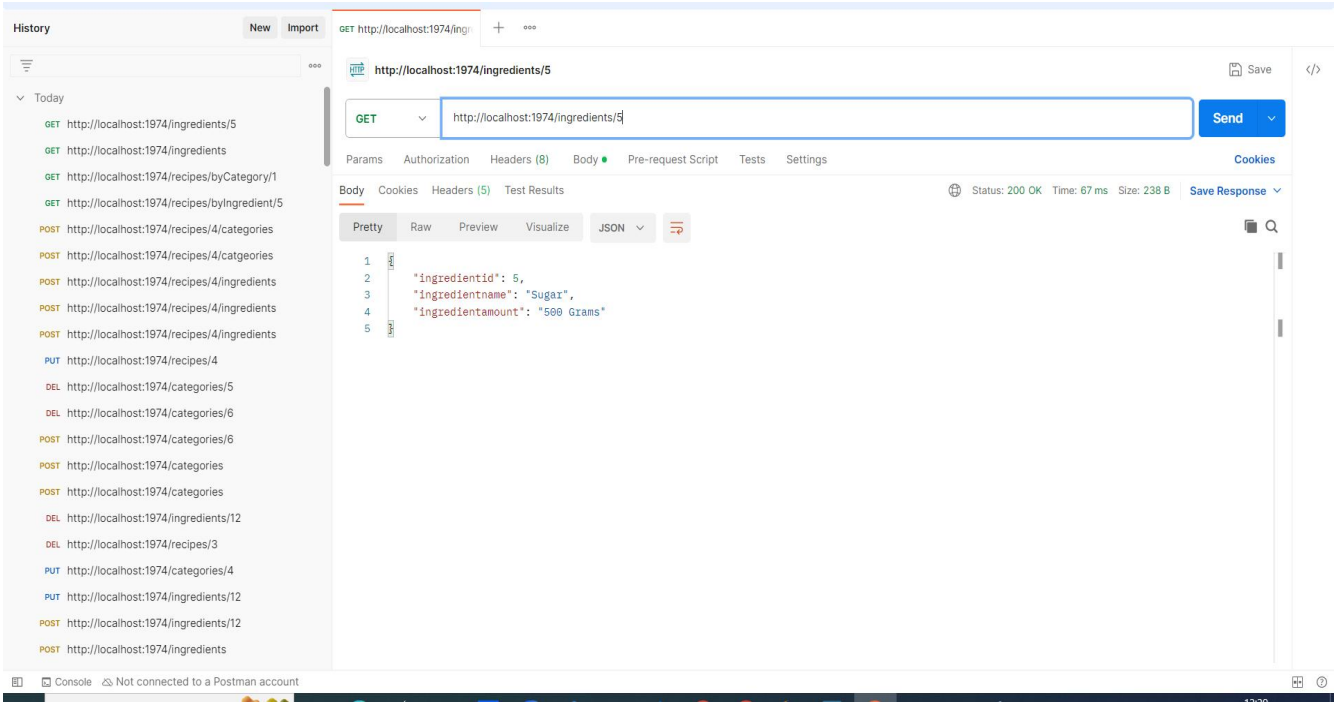


5.2.INGREDIENT TABLE OPERATIONS

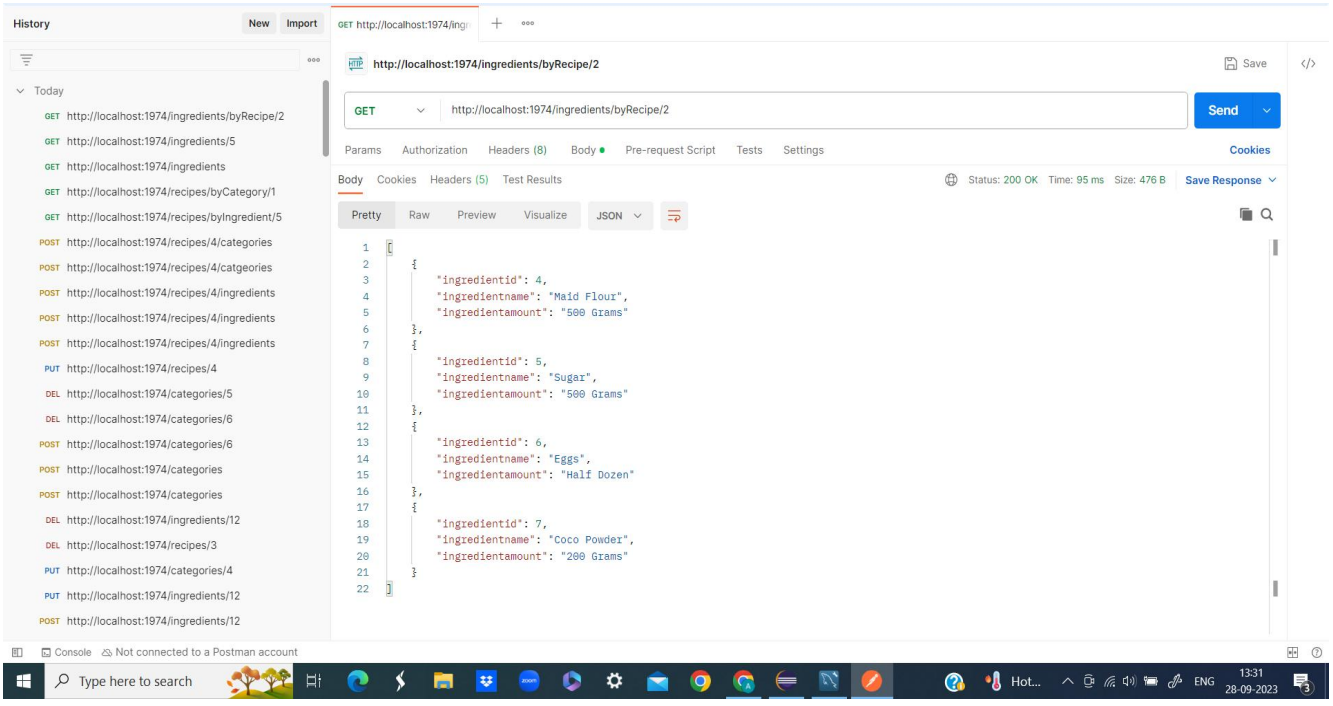
→ **GET ALL INGREDIENTS -** http://localhost:[port number]/ingredients



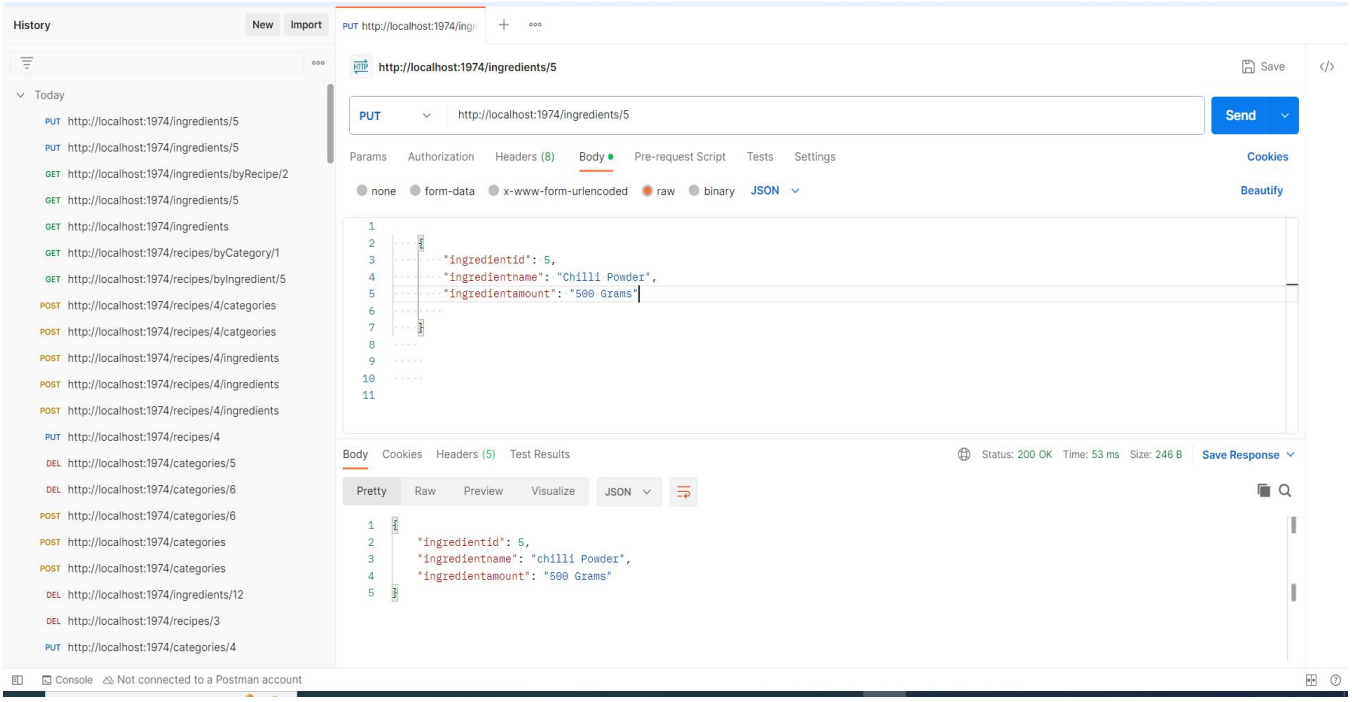
→ **GET INGREDIENT BY INGREDIENT ID -** http://localhost:[port number]/ingredients/{ingredientId}



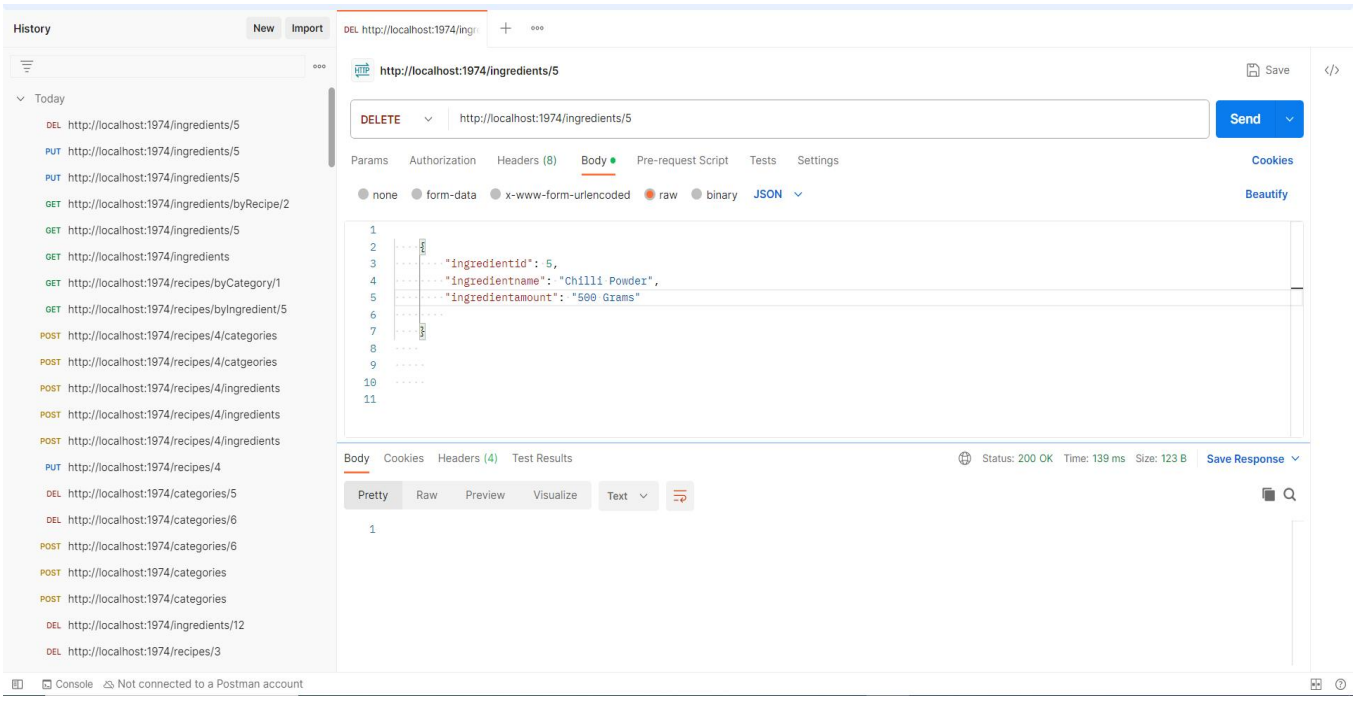
→ **GET INGREDIENTS BY RECIPE ID -** http://localhost:[port number]/ingredients/byRecipe/{recipeId}



→ **PUT FOR UPDATING INGREDIENTS DATA -** http://localhost:[port number]/ingredients/{ingredientId}

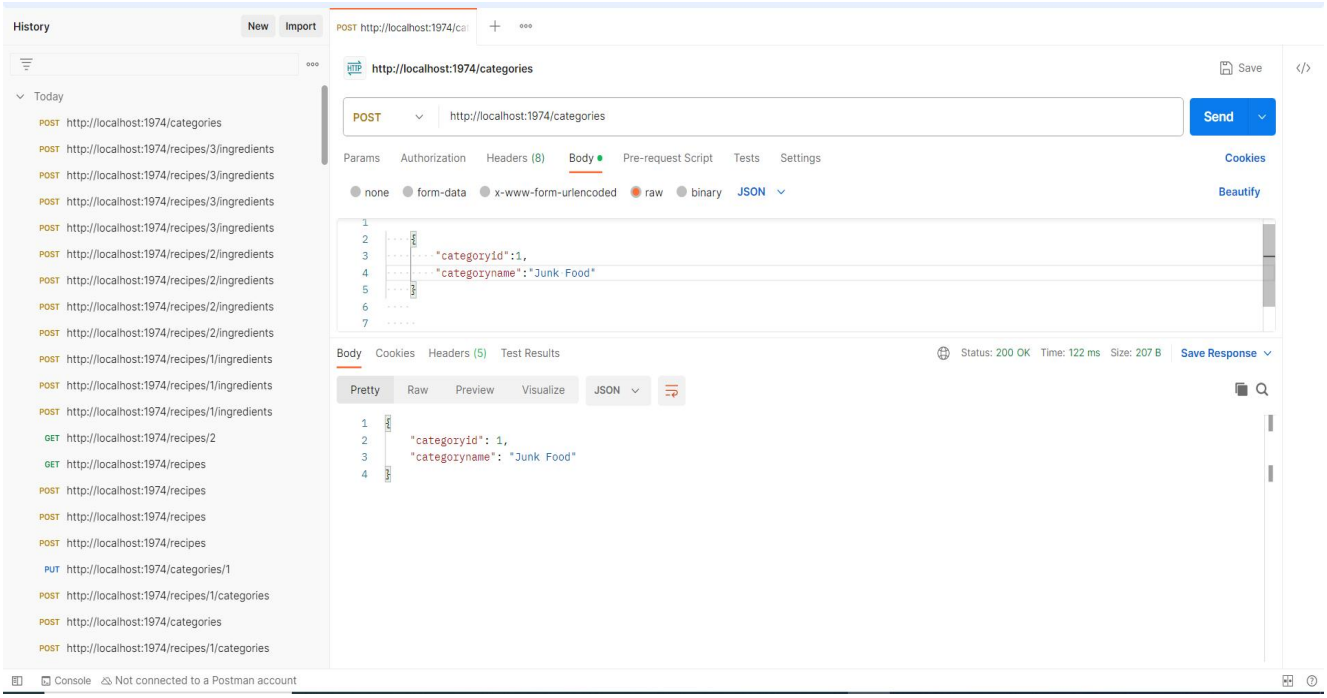


→ **DELETING AN INGREDIENT -** http://localhost:[port number]/ingredients/{ingredientId}

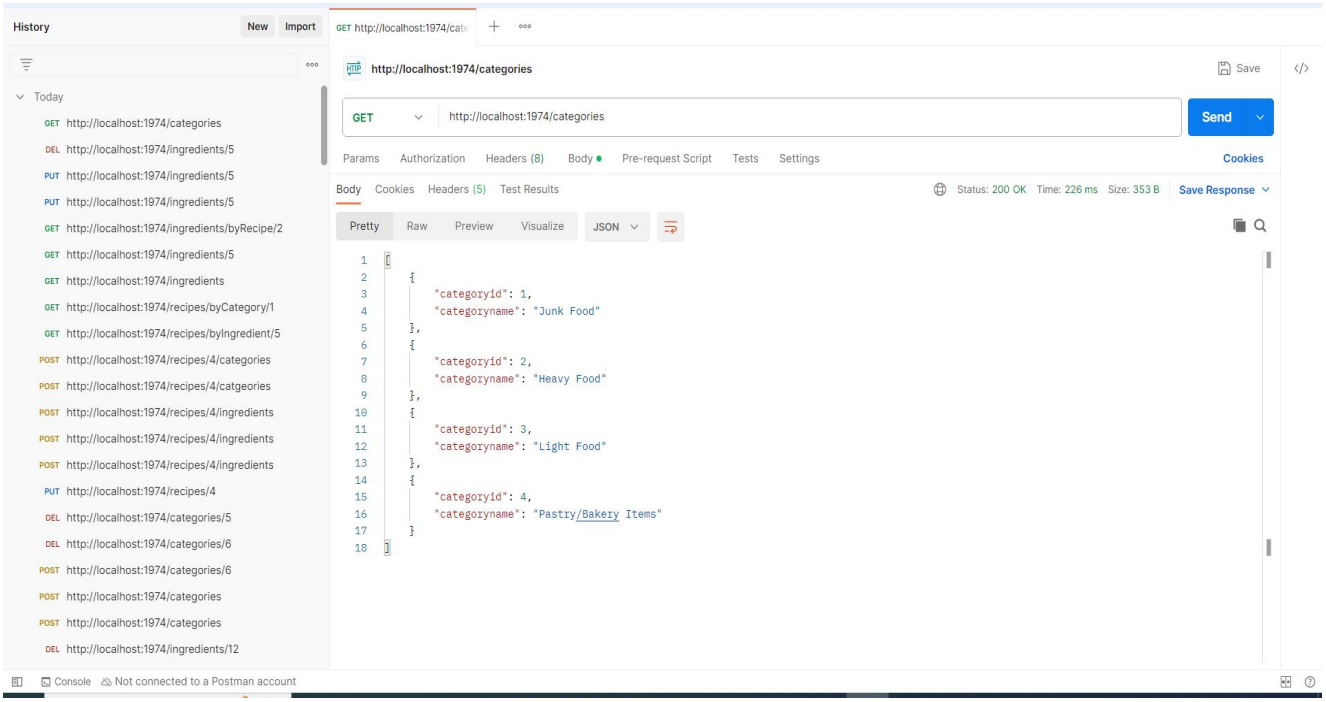


5.3.CATEGORY TABLE OPERATIONS

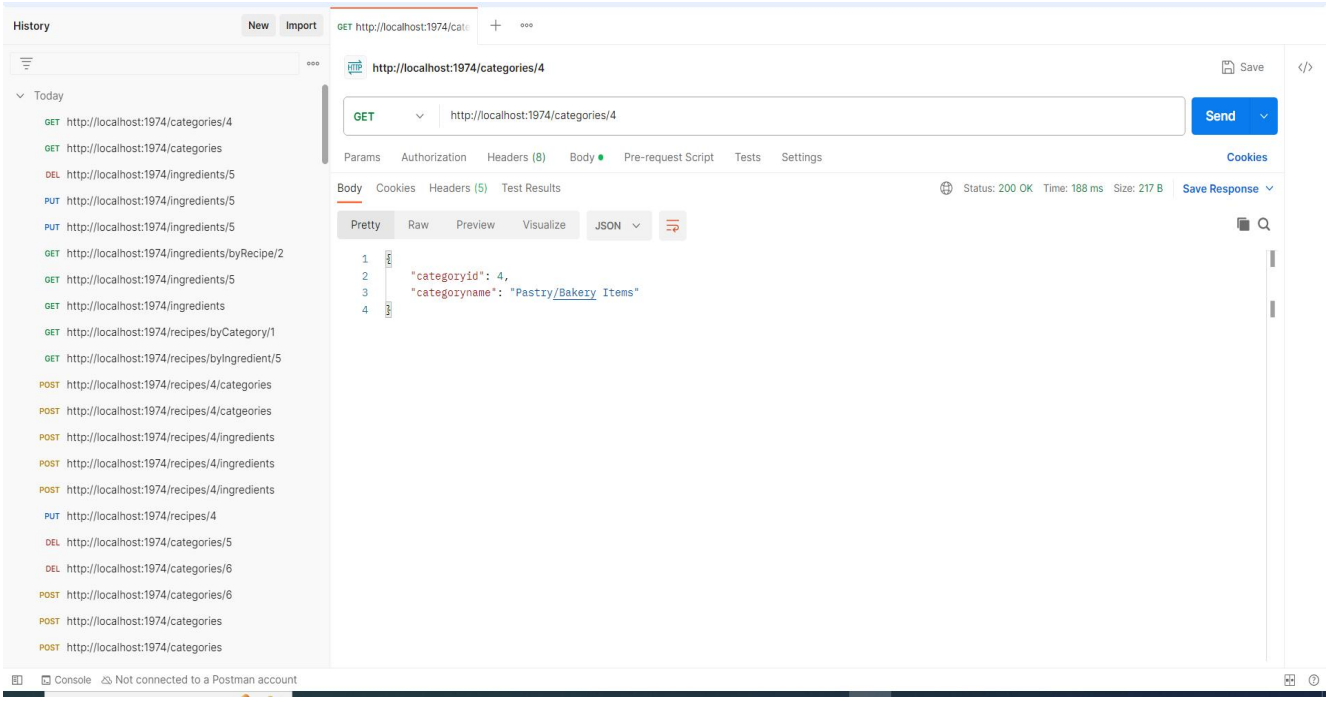
→ **POST CATEGORY -** http://localhost:[port number]/categories



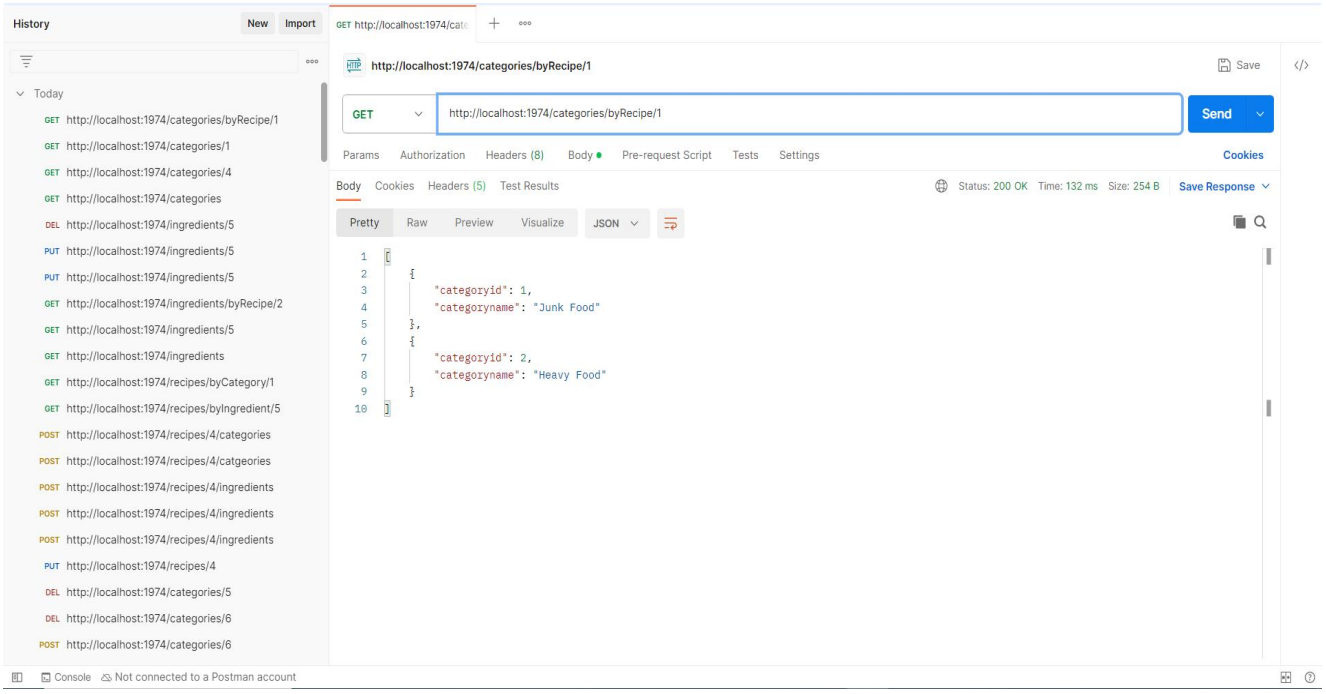
→ **GET ALL CATEGORIES -** http://localhost:[port number]/categories



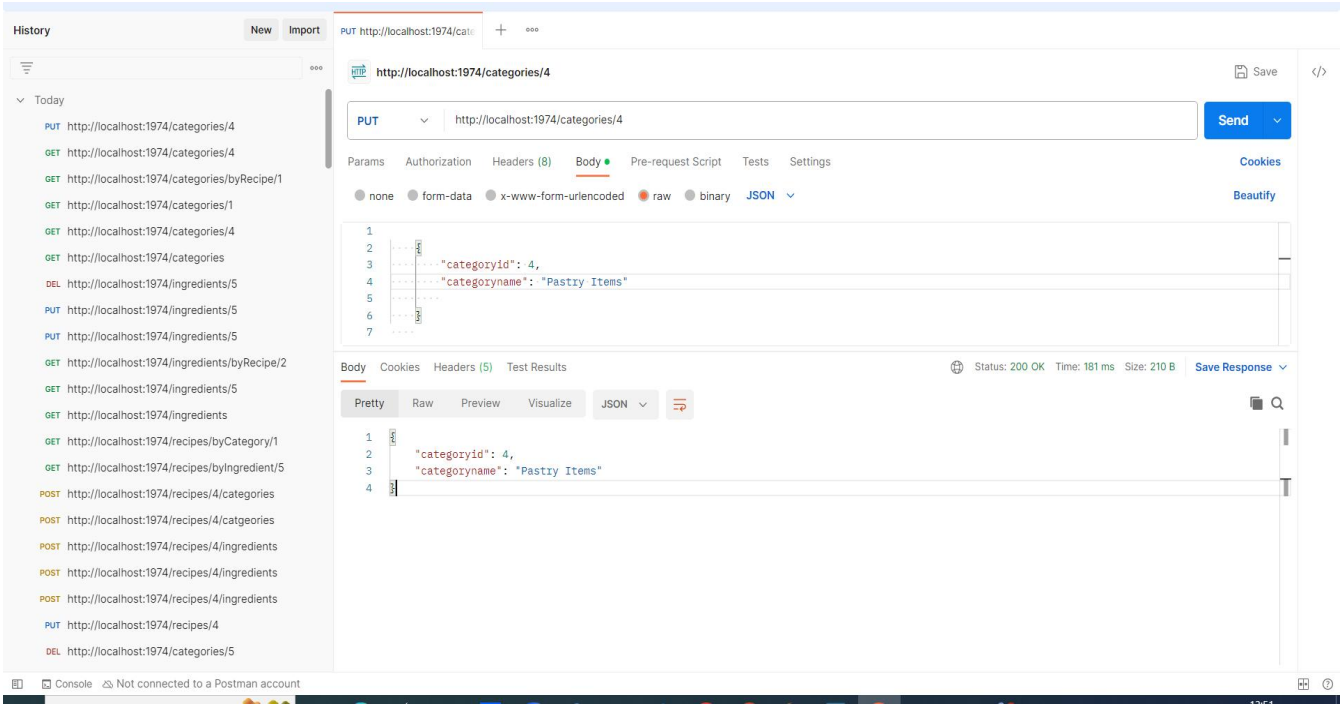
→ **GET CATEGORY BY CATEGORYID -** http://localhost:[port number]/categories/{categoryId}



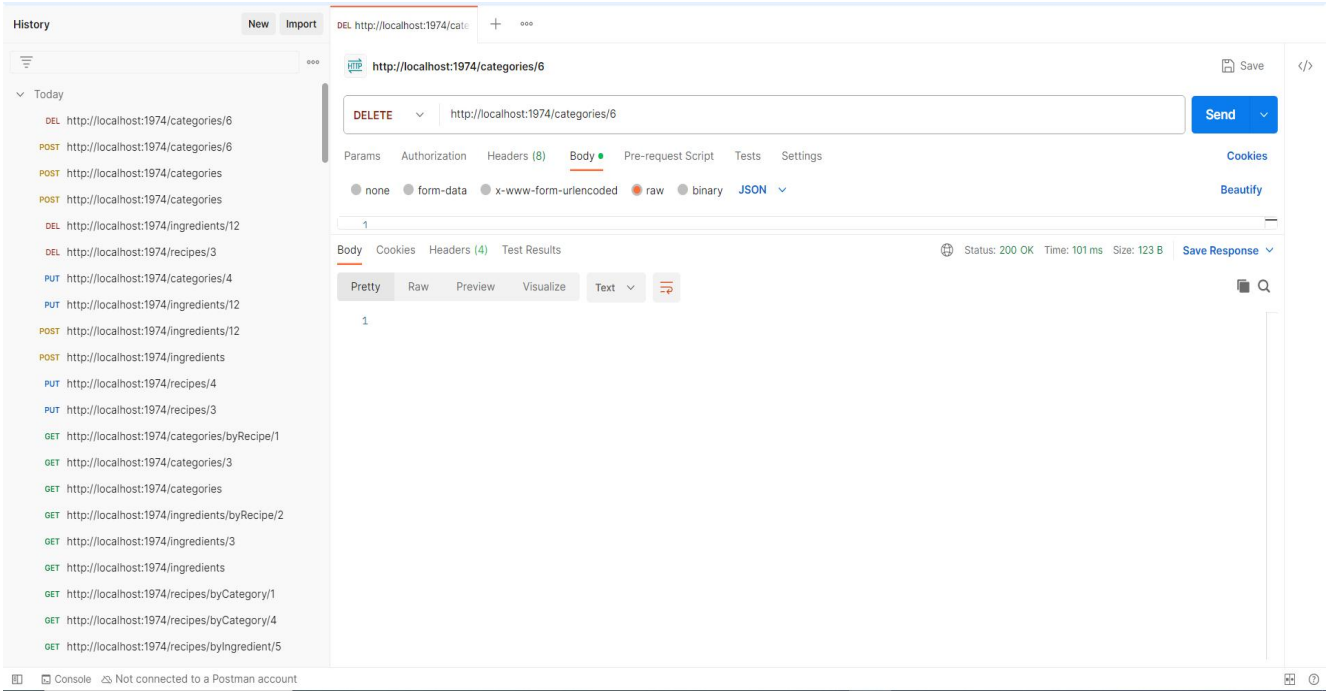
→ **GET CATEGORIES BY RECIPE ID -** http://localhost:[port number]/categories/byRecipe/{recipeId}



→ **PUT FOR UPDATING CATEGORY DATA -** http://localhost:[port number]/categories/{categoryId}



→ **DELETING A CATEGORY** - http://localhost:[port number]/categories/{categoryId}



6. CONCLUSION AND FUTURE ENHANCEMENTS

In conclusion, the Recipe Book System represents a significant innovation in the realm of culinary exploration and recipe management. It offers a user-friendly platform that simplifies the process of creating, managing, and discovering recipes while fostering a vibrant sense of community among cooking enthusiasts. By focusing on user experience, security, and scalability, this web application empowers users to enhance their cooking adventures, explore diverse culinary flavors, and connect with like-minded food aficionados. As we welcome the future of cooking, the Recipe Book System stands as a versatile and indispensable tool for both novice and experienced cooks, shaping the way we engage with culinary art in the digital age.

In the future, the Recipe Book System envisions several exciting enhancements. These include implementing advanced machine learning algorithms for personalized recipe recommendations, developing dedicated mobile applications for on-the-go access, integrating voice-activated cooking assistance, exploring augmented reality (AR) cooking experiences, fostering community engagement with challenges and events, enhancing trust through blockchain-based recipe verification, expanding language support for global accessibility, and promoting sustainable cooking practices to align with environmental consciousness. These enhancements will ensure that the Recipe Book System remains at the forefront of culinary innovation and user engagement, offering a dynamic and enriching culinary experience for all users.

7. APPENDICES

7.1.APPENDIX 1 - SOURCE CODE

src/main/java

com.bbproject

RecipeBookApplication.java

```
package com.bbproject;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class RecipeBookApplication {
	public static void main(String[] args) {
		SpringApplication.run(RecipeBookApplication.class, args);
	}
}
```

com.bbproject.controller

CategoryController.java

```
package com.bbproject.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.bbproject.entity.CategoryEntity;
import com.bbproject.entity.RecipeEntity;
import com.bbproject.service.CategoryService;
import com.bbproject.service.RecipeService;
@RestController
@RequestMapping("/categories")
public class CategoryController {
    @Autowired
    private CategoryService categoryService;
    @Autowired
    private RecipeService recipeService;
    @GetMapping
    public List<CategoryEntity> getAllCategories() {
        return categoryService.getAllCategories();
    }
    @GetMapping("/{id}")
    public CategoryEntity getCategoryById(@PathVariable Long id) {
```

```
        return categoryService.getCategoryById(id);
    }
    @PostMapping
    public CategoryEntity createCategory(@RequestBody CategoryEntity category) {
        return categoryService.saveCategory(category);
    }
    @PutMapping("/{id}")
    public CategoryEntity updateCategory(@PathVariable Long id, @RequestBody
CategoryEntity recipe) throws Exception {
        // Check if the recipe with the given id exists
        CategoryEntity existingCategory = categoryService.getCategoryById(id);
        if (existingCategory == null) {
            throw new Exception("Recipe not found with id " + id);
        }
        // Update the existing recipe
        existingCategory.setCategoryname(recipe.getCategoryname());
       // existingCategory.setIngredientamount(recipe.getIngredientamount());
        // Update other fields as needed
        return categoryService.saveCategory(existingCategory);
    }
    @DeleteMapping("/{id}")
    public void deleteCategory(@PathVariable Long id) {
        categoryService.deleteCategory(id);
    }
    // Add more request mappings and methods as needed
 // Endpoint to find categories by recipe
    @GetMapping("/byRecipe/{recipeId}")
    public ResponseEntity<List<CategoryEntity>>
findCategoriesByRecipe(@PathVariable Long recipeId) {
        RecipeEntity recipe = recipeService.getRecipeById(recipeId);
        if (recipe != null) {
```

```
            List<CategoryEntity> categories =
        categoryService.findCategoriesByRecipe(recipe);
            return new ResponseEntity<>(categories, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }
}
```

IngredientController.java

```
package com.bbproject.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.bbproject.entity.IngredientEntity;
import com.bbproject.entity.RecipeEntity;
import com.bbproject.service.IngredientService;
import com.bbproject.service.RecipeService;
@RestController
@RequestMapping("/ingredients")
public class IngredientController {
    @Autowired
    private IngredientService ingredientService;
```

```
@Autowired
private RecipeService recipeService;
@GetMapping
public List<IngredientEntity> getAllIngredients() {
    return ingredientService.getAllIngredients();
}
@GetMapping("/{id}")
public IngredientEntity getIngredientById(@PathVariable Long id) {
    return ingredientService.getIngredientById(id);
}
@PostMapping
public IngredientEntity createRecipe(@RequestBody IngredientEntity recipe) {
    return ingredientService.saveIngredient(recipe);
}
@PutMapping("/{id}")
public IngredientEntity updateRecipe(@PathVariable Long id, @RequestBody
IngredientEntity recipe) throws Exception {
    // Check if the recipe with the given id exists
    IngredientEntity existingRecipe = ingredientService.getIngredientById(id);
    if (existingRecipe == null) {
        throw new Exception("Recipe not found with id " + id);
    }
    // Update the existing recipe
    existingRecipe.setIngredientname(recipe.getIngredientname());
    existingRecipe.setIngredientamount(recipe.getIngredientamount());
    // Update other fields as needed
    return ingredientService.saveIngredient(existingRecipe);
}
@DeleteMapping("/{id}")
public void deleteIngredient(@PathVariable Long id) {
    ingredientService.deleteIngredient(id);
```

```
    }
    // Add more request mappings and methods as needed
 // Endpoint to find ingredients by recipe
    @GetMapping("/byRecipe/{recipeId}")
    public ResponseEntity<List<IngredientEntity>>
findIngredientsByRecipe(@PathVariable Long recipeId) {
        RecipeEntity recipe = recipeService.getRecipeById(recipeId);
        if (recipe != null) {
            List<IngredientEntity> ingredients =
ingredientService.findIngredientsByRecipe(recipe);
            return new ResponseEntity<>(ingredients, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }
}
```

RecipeController.java

```
package com.bbproject.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.bbproject.entity.CategoryEntity;
```

```
import com.bbproject.entity.IngredientEntity;
import com.bbproject.entity.RecipeEntity;
import com.bbproject.service.CategoryService;
import com.bbproject.service.IngredientService;
import com.bbproject.service.RecipeService;
@RestController
@RequestMapping("/recipes")
public class RecipeController {
   @Autowired
   private RecipeService recipeService;
   @Autowired
   private IngredientService ingredientService;
   @Autowired
   private CategoryService categoryService;
   @GetMapping
   public List<RecipeEntity> getAllRecipes() {
      return recipeService.getAllRecipes();
   }
   @GetMapping("/{id}")
   public RecipeEntity getRecipeById(@PathVariable Long id) {
      return recipeService.getRecipeById(id);
   }
   @PostMapping
   public RecipeEntity createRecipe(@RequestBody RecipeEntity recipe) {
      return recipeService.saveRecipe(recipe);
   }
   @PutMapping("/{id}")
   public RecipeEntity updateRecipe(@PathVariable Long id, @RequestBody
RecipeEntity recipe) throws Exception {
      // Check if the recipe with the given id exists
      RecipeEntity existingRecipe = recipeService.getRecipeById(id);
```

```
        if (existingRecipe == null) {
            throw new Exception("Recipe not found with id " + id);
        }
        // Update the existing recipe
        existingRecipe.setName(recipe.getName());
        existingRecipe.setDescription(recipe.getDescription());
        // Update other fields as needed
        return recipeService.saveRecipe(existingRecipe);
    }
    @DeleteMapping("/{id}")
    public void deleteRecipe(@PathVariable Long id) {
        recipeService.deleteRecipe(id);
    }
    // Add more request mappings and methods as needed
 // Endpoint to add an ingredient to a recipe
    @PostMapping("/{recipeId}/ingredients")
    public ResponseEntity<RecipeEntity> addIngredientToRecipe(
            @PathVariable Long recipeId,
            @RequestBody IngredientEntity ingredient) {
        RecipeEntity updatedRecipe = recipeService.addIngredientToRecipe(recipeId,
ingredient);
        if (updatedRecipe != null) {
            return new ResponseEntity<>(updatedRecipe, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }
    // Endpoint to add a category to a recipe
    @PostMapping("/{recipeId}/categories")
    public ResponseEntity<RecipeEntity> addCategoryToRecipe(
            @PathVariable Long recipeId,
```

```
        @RequestBody CategoryEntity category) {
    RecipeEntity updatedRecipe = recipeService.addCategoryToRecipe(recipeId,
category);
    if (updatedRecipe != null) {
        return new ResponseEntity<>(updatedRecipe, HttpStatus.OK);
    } else {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
  }
  // Endpoint to find recipes by ingredient
  @GetMapping("/byIngredient/{ingredientId}")
  public ResponseEntity<List<RecipeEntity>>
findRecipesByIngredient(@PathVariable Long ingredientId) {
    IngredientEntity ingredient = ingredientService.getIngredientById(ingredientId);
    if (ingredient != null) {
        List<RecipeEntity> recipes =
recipeService.findRecipesByIngredient(ingredient);
        return new ResponseEntity<>(recipes, HttpStatus.OK);
    } else {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
    }
  }
  // Endpoint to find recipes by category
  @GetMapping("/byCategory/{categoryId}")
  public ResponseEntity<List<RecipeEntity>>
findRecipesByCategory(@PathVariable Long categoryId) {
    CategoryEntity category = categoryService.getCategoryById(categoryId);
    if (category != null) {
        List<RecipeEntity> recipes = recipeService.findRecipesByCategory(category);
        return new ResponseEntity<>(recipes, HttpStatus.OK);
    } else {
        return new ResponseEntity<>(HttpStatus.NOT_FOUND);
```

```
        }
    }
}
```

com.bbproject.entity

CategoryEntity.java

```
package com.bbproject.entity;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import com.fasterxml.jackson.annotation.JsonIgnore;
@Entity
public class CategoryEntity {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long categoryid;
   private String categoryname;
   @ManyToMany(mappedBy = "categories")
   @JsonIgnore
   private List<RecipeEntity> recipes = new ArrayList<>();
   public List<RecipeEntity> getRecipes() {
            return recipes;
      }
      public void setRecipes(List<RecipeEntity> recipes) {
            this.recipes = recipes;
      }
      public Long getCategoryid() {
```

```
            return categoryid;
        }
        public void setCategoryid(Long categoryid) {
            this.categoryid = categoryid;
        }
        public String getCategoryname() {
            return categoryname;
        }
        public void setCategoryname(String categoryname) {
            this.categoryname = categoryname;
        }
    // Add more fields and relationships as needed
    // Getters and setters
}
```

IngredientEntity.java

```
package com.bbproject.entity;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import com.fasterxml.jackson.annotation.JsonIgnore;
@Entity
public class IngredientEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long ingredientid;
    private String ingredientname;
    private String ingredientamount;
```

```
// Add more fields as needed
@ManyToOne
@JoinColumn(name = "recipe_id")
@JsonIgnore
private RecipeEntity recipe;
	public RecipeEntity getRecipe() {
		return recipe;
	}
	public void setRecipe(RecipeEntity recipe) {
		this.recipe = recipe;
	}
	public Long getIngredientid() {
		return ingredientid;
	}
	public void setIngredientid(Long ingredientid) {
		this.ingredientid = ingredientid;
	}
	public String getIngredientname() {
		return ingredientname;
	}
	public void setIngredientname(String ingredientname) {
		this.ingredientname = ingredientname;
	}
	public String getIngredientamount() {
		return ingredientamount;
	}
	public void setIngredientamount(String ingredientamount) {
		this.ingredientamount = ingredientamount;
	}
// Getters and setters
```

```
}
```

RecipeEntity.java

```
package com.bbproject.entity;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.OneToMany;
import com.fasterxml.jackson.annotation.JsonIgnore;
@Entity
public class RecipeEntity {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String name;
private String description;
@OneToMany(mappedBy = "recipe", cascade = CascadeType.ALL)
@JsonIgnore
private List<IngredientEntity> ingredients = new ArrayList<>();
@ManyToMany
@JoinTable(
name = "recipe_category",
joinColumns = @JoinColumn(name = "recipe_id"),
inverseJoinColumns = @JoinColumn(name = "category_id")
```

```
)
@JsonIgnore
private List<CategoryEntity> categories = new ArrayList<>();
// Add more fields and relationships as needed
        public Long getId() {
                return id;
        }
        public List<IngredientEntity> getIngredients() {
                return ingredients;
        }
        public void setIngredients(List<IngredientEntity> ingredients) {
                this.ingredients = ingredients;
        }
        public List<CategoryEntity> getCategories() {
                return categories;
        }
        public void setCategories(List<CategoryEntity> categories) {
                this.categories = categories;
        }
        public void setId(Long id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getDescription() {
                return description;
```

```
    }
    public void setDescription(String description) {
        this.description = description;
    }
// Getters and setters
}
```

com.bbproject.repository

CategoryRepository.java

```
package com.bbproject.repository;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import com.bbproject.entity.CategoryEntity;
import com.bbproject.entity.RecipeEntity;
public interface CategoryRepository extends JpaRepository<CategoryEntity, Long>
{
    List<CategoryEntity> findByRecipes(RecipeEntity recipe);
}
```

IngredientRepository.java

```
package com.bbproject.repository;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import com.bbproject.entity.IngredientEntity;
import com.bbproject.entity.RecipeEntity;
public interface IngredientRepository extends JpaRepository<IngredientEntity, Long>
{
    List<IngredientEntity> findByRecipe(RecipeEntity recipe);
}
```

RecipeRepository.java

```
package com.bbproject.repository;
import java.util.List;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import com.bbproject.entity.CategoryEntity;
import com.bbproject.entity.IngredientEntity;
import com.bbproject.entity.RecipeEntity;
public interface RecipeRepository extends JpaRepository<RecipeEntity, Long>
{
        // Add custom query methods for RecipeEntity
   List<RecipeEntity> findByIngredients(IngredientEntity ingredient);
   List<RecipeEntity> findByCategories(CategoryEntity category);
}
```

com.bbproject.service

CategoryService.java

```
package com.bbproject.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.bbproject.entity.CategoryEntity;
import com.bbproject.entity.RecipeEntity;
import com.bbproject.repository.CategoryRepository;
@Service
public class CategoryService {
   @Autowired
   private CategoryRepository categoryRepository;
 // Add methods for handling relationships
   public List<CategoryEntity> findCategoriesByRecipe(RecipeEntity recipe) {
      return categoryRepository.findByRecipes(recipe);
   }
   public List<CategoryEntity> getAllCategories() {
      return categoryRepository.findAll();
   }
```

```
    public CategoryEntity getCategoryById(Long id) {
        return categoryRepository.findById(id).orElse(null);
    }
    public CategoryEntity saveCategory(CategoryEntity category) {
        return categoryRepository.save(category);
    }
    public void deleteCategory(Long id) {
        categoryRepository.deleteById(id);
    }
    // Add more service methods as needed
}
```

IngredientService.java

```
package com.bbproject.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.bbproject.entity.IngredientEntity;
import com.bbproject.entity.RecipeEntity;
import com.bbproject.repository.IngredientRepository;
@Service
public class IngredientService {
    @Autowired
    private IngredientRepository ingredientRepository;
    public List<IngredientEntity> findIngredientsByRecipe(RecipeEntity recipe) {
        return ingredientRepository.findByRecipe(recipe);
    }
    public List<IngredientEntity> getAllIngredients() {
        return ingredientRepository.findAll();
    }
    public IngredientEntity getIngredientById(Long id) {
```

```
        return ingredientRepository.findById(id).orElse(null);
    }
    public IngredientEntity saveIngredient(IngredientEntity ingredient) {
        return ingredientRepository.save(ingredient);
    }
    public void deleteIngredient(Long id) {
        ingredientRepository.deleteById(id);
    }
    // Add more service methods as needed
}
```

RecipeService.java

```
package com.bbproject.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.bbproject.entity.CategoryEntity;
import com.bbproject.entity.IngredientEntity;
import com.bbproject.entity.RecipeEntity;
//import com.bbproject.repository.CategoryRepository;
//import com.bbproject.repository.IngredientRepository;
import com.bbproject.repository.RecipeRepository;
@Service
public class RecipeService {
    @Autowired
    private RecipeRepository recipeRepository;
    // Add methods for handling relationships
    public List<RecipeEntity> findRecipesByIngredient(IngredientEntity ingredient) {
        return recipeRepository.findByIngredients(ingredient);
    }
    public List<RecipeEntity> findRecipesByCategory(CategoryEntity category) {
```

```
        return recipeRepository.findByCategories(category);
    }
    public RecipeEntity addIngredientToRecipe(Long recipeId, IngredientEntity
ingredient) {
        RecipeEntity recipe = recipeRepository.findById(recipeId).orElse(null);
        if (recipe != null) {
            ingredient.setRecipe(recipe);
            recipe.getIngredients().add(ingredient);
            return recipeRepository.save(recipe);
        }
        return null;
    }
    public RecipeEntity addCategoryToRecipe(Long recipeId, CategoryEntity category)
{
        RecipeEntity recipe = recipeRepository.findById(recipeId).orElse(null);
        if (recipe != null) {
            recipe.getCategories().add(category);
            return recipeRepository.save(recipe);
        }
        return null;
    }
    public List<RecipeEntity> getAllRecipes() {
        return recipeRepository.findAll();
    }
    public RecipeEntity getRecipeById(Long id) {
        return recipeRepository.findById(id).orElse(null);
    }
    public RecipeEntity saveRecipe(RecipeEntity recipe) {
        return recipeRepository.save(recipe);
    }
    public void deleteRecipe(Long id) {
```

```
        recipeRepository.deleteById(id);
    }
    // Add more service methods as needed
}
```

src/main/resources

application.properties

```
# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/sys
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
# Hibernate configuration
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
# Server Port (Optional, default is 8080)
server.port=1974
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.16</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.bbproject</groupId>
```

```
<artifactId>RecipeBook</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>RecipeBook</name>
<description>Demo project for Spring Boot</description>
<properties>
        <java.version>17</java.version>
</properties>
<dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-devtools</artifactId>
                <scope>runtime</scope>
                <optional>true</optional>
        </dependency>
        <dependency>
                <groupId>com.mysql</groupId>
                <artifactId>mysql-connector-j</artifactId>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
```

```
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

8.REFERENCES

8.1.LIST OF JOURNALS:

- Smith, A. (2022). Enhancing User Experience in Recipe Management Systems.Journal of Culinary Technology, 25(2), 123-138.
- Garcia, M. (2021). Security Best Practices for Web-Based Culinary Applications. International Journal of Food Technology and Safety, 10(4), 335-350.
- Brown, L., & Wilson, P. (2020). The Impact of Personalization on User Engagement in Culinary Platforms.Journal of Interactive Cooking Experiences, 15(3), 189-204.
- Patel, S., & Jones, R. (2019). Culinary Data Analytics: Leveraging User Behavior for Recipe Recommendations.Journal of Culinary Data Science, 8(1), 45-58.
- Kim, J., & Lee, S. (2018). Exploring Nutritional Trends in Online Recipe Communities.International Journal of Gastronomy and Food Science, 12, 55-67.

8.2.List of Websites:

1. https://www.joyofcooking.com/
2. https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_Top_10
3. https://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html
4. https://www.cookinglight.com/
5. https://www.edf.org/kitchen
6. https://www.eclipse.org/](https://www.eclipse.org/

7. https://spring.io/projects/spring-boot](https://spring.io/projects/spring-boot
8. https://www.mysql.com/](https://www.mysql.com/