COOKBOOK – RECIPE FOR KITCHEN

A project work
submitted for the partial fulfillment for the
award of degree in

NAAN MUDHALVAN- PROJECT DEVELOPMENT COURSE

COLLEGE CODE : UNM1441

BACHELOR OF SOFTWARE APPLICATION

BY

SUJITHRA. D

VIJAYALAKSHMI. P

VIMALA DEVI. B

VIGNESH. A

SREE MUTHUKUMARASWAMY COLLEGE

(AFFILIATED TO UNIVERSITY OF MADRES)

KODUNGAIYUR, CHENNAI – 600 118

APRIL 2025 EXAMINATIOS

BONAFIDE CERTIFICATED

This is to certify that the project entitled "**YOUR PROJECT TOPIC**" being subimtted to Sree Muthukumaraswamy College, College code : UNM1441 Kodungaiyur, Chennai – 600118, by group of satudents in partial fulfillment for the award of the degree of B.Sc., Software Applications is a bonafide record of the work carried out by her under my guidance and supervision.

Internal Guide	Head of the Department
(Mr. M. Kalaimani)	(Mrs. T. Merlin Jaba)

DECLARATION

I hereby declare that this project titled **“Your Project Topic”** submitted by me in partial fulfillment of the requirements for the Bachelor Degree of Software Applications has not formed a basis for the awared of any other deegree, diploma, associate, fellowships or other similar titles and this project developed by us.

NAME OF THE STUDENTS	**REGISTER NO**
1 SUJITHRA. D	222208944
2 VIJAYALAKSHMI. P	222208946
3 VIMALA DEVI. B	222208947
4 VIGNESH. A	222208926

Place **:** Chennai – 600 118

Date **: 11-02-2025**

Sree Muthukumaraswamy College Kodungaiyur

Chennai – 600 118

Project Documentation : Cookbook – Recipe for Kitchen

Team Members:

- SUJITHRA. D - Backend Development
- VIJAYALAKSHMI. P – Data Handling
- VIMALA DEVI. B – Frontend Development
- VIGNESH. A – User Interface

1. Introduction

Project Title: CookBook: Recipe for Kitchen

Welcome to CookBook: Recipe for Kitchen, your ultimate destination for all things food. Explore, create, and share recipes with a community of fellow food enthusiasts.

---

2. Project Overview

Purpose:

The Cookbook application is designed to help users discover, store, and share recipes. Users can browse a collection of recipes, add their own, and manage ingredients efficiently.

Features:

User authentication and profile management

Recipe creation, editing, and deletion

Categorization and tagging of recipes

Search and filter functionality

Ingredient and shopping list management

Favorites and rating system

API for integration with frontend (if applicable)

---

3. Architecture

Technology Stack:

Backend: Java (Spring Boot)

Database: MySQL / PostgreSQL

Frontend (if applicable): React.js / Angular

Authentication: JWT / OAuth2

Storage: Cloud storage for images (e.g., AWS S3, Firebase Storage)

Component Structure:

Controllers: Handle HTTP requests

Services: Contain business logic

Repositories: Communicate with the database

Models: Define entities (Recipe, User, Ingredient, etc.)

API Structure:

GET /recipes – Fetch all recipes

POST /recipes – Add a new recipe

GET /recipes/{id} – Fetch recipe details

PUT /recipes/{id} – Update a recipe

DELETE /recipes/{id} – Remove a recipe

---

4. Setup Instructions

Prerequisites:

Install Java (JDK 17+)

Install Maven or Gradle

Install MySQL/PostgreSQL

(Optional) Docker for containerized deployment

Installation:

1. Clone the repository:

```
git clone https://github.com/your-repo/cookbook.git
cd cookbook
```

2. Configure environment variables in application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/cookbook
spring.datasource.username=root
spring.datasource.password=yourpassword
```

3. Build and run the application:

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

---

5. Folder Structure

```
/src
  /main/java/com/example/cookbook
    /controllers   # Handles HTTP requests
    /services      # Business logic
    /repositories  # Database interactions
    /models        # Data models (Recipe, User, etc.)
  /main/resources
```

    application.properties # Configuration file

---

6. Running the Application

Local: mvn spring-boot:run

Docker:

```
docker build -t cookbook-app .
docker run -p 8080:8080 cookbook-app
```

API Documentation

API endpoints documented using Swagger (http://localhost:8080/swagger-ui.html)

Running the Cookbook Application

1. Prerequisites

Before running the application, ensure you have the following installed:

Java Development Kit (JDK 17+)

Maven or Gradle (for dependency management)

MySQL/PostgreSQL (for database)

Docker (optional, for containerized deployment)

---

2. Clone the Repository

If you haven't already, clone the project from your Git repository:

```
git clone https://github.com/your-repo/cookbook.git
cd cookbook
```

---

3. Configure the Database

Modify src/main/resources/application.properties (for MySQL example):

```
spring.datasource.url=jdbc:mysql://localhost:3306/cookbook
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
```

If using PostgreSQL:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/cookbook
spring.datasource.username=postgres
spring.datasource.password=yourpassword
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

---

4. Build the Application

Use Maven to build the project:

```
mvn clean install
```

If using Gradle, run:

```
gradle build
```

---

5. Run the Application

Start the application using:

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

or

```
java -jar target/cookbook-0.0.1-SNAPSHOT.jar
```

If using Gradle, run:

```
gradle bootRun
```

---

6. Access the Application

API Endpoints (Swagger UI):
Open http://localhost:8080/swagger-ui.html

If using a Frontend:
Ensure your frontend (React, Angular, etc.) is configured to call http://localhost:8080/api

---

7. Running with Docker (Optional)

If you want to run the application in a container:

1. Build the Docker Image:

```
docker build -t cookbook-app .
```

2. Run the Container:

```
docker run -p 8080:8080 cookbook-app
```

---

8. Stopping the Application

If running with Maven/Gradle, press CTRL + C to stop.

If running a JAR, find the process and kill it:

```
ps -ef | grep cookbook
kill -9 <PROCESS_ID>
```

If using Docker, stop the container:

```
docker ps
docker stop <CONTAINER_ID>
```

---

9. Logs and Debugging

To check logs, run:

```
tail -f logs/app.log
```

or view logs in the terminal using:

```
mvn spring-boot:run -Dspring-boot.run.arguments="--debug"
```

---

7. Component Documentation

1. Package Structure

The project follows a layered architecture:

```
/src/main/java/com/example/cookbook
  /controllers    # Handles HTTP requests (REST API)
  /services       # Business logic layer
  /repositories   # Data access layer (JPA repositories)
  /models         # Entity classes representing database tables
  /dto            # Data Transfer Objects for API responses
  /config         # Configuration files (CORS, security, etc.)
  /exceptions     # Custom exception handling
  /security       # Authentication and authorization logic
```

---

2. Component Documentation

2.1 Models (Entities)

These represent database tables and are managed by JPA.

Recipe.java

```java
@Entity
@Table(name = "recipes")
public class Recipe {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;
    @Column(columnDefinition = "TEXT")
    private String ingredients;

    @Column(columnDefinition = "TEXT")
    private String instructions;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User createdBy;

    // Getters and Setters
}
```

Purpose: Represents a recipe.

Relations: Linked to a User entity.

2.2 Repositories (Data Access Layer)

Repositories interact with the database using JPA.

RecipeRepository.java

```
@Repository
public interface RecipeRepository extends JpaRepository<Recipe, Long>
{
    List<Recipe> findByCreatedBy(User user);
    List<Recipe> findByTitleContainingIgnoreCase(String title);
}
```

Purpose: Provides methods to query the database for recipes.

Custom Methods:

Find recipes by user

Search recipes by title

2.3 Services (Business Logic)

Services contain the main logic for handling recipes.

RecipeService.java

```
@Service
public class RecipeService {
    @Autowired
    private RecipeRepository recipeRepository;

    public List<Recipe> getAllRecipes() {
        return recipeRepository.findAll();
    }

    public Recipe getRecipeById(Long id) {
        return recipeRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Recipe
not found"));
    }
    public Recipe addRecipe(Recipe recipe) {
        return recipeRepository.save(recipe);
    }

    public void deleteRecipe(Long id) {
        Recipe recipe = getRecipeById(id);
        recipeRepository.delete(recipe);
    }
}
```

Purpose: Handles all business logic related to recipes.

Methods: Fetch, add, and delete recipes.

2.4 Controllers (API Layer)

Controllers handle HTTP requests and return responses.

RecipeController.java

```
@RestController
@RequestMapping("/api/recipes")
public class RecipeController {
    @Autowired
    private RecipeService recipeService;

    @GetMapping
    public List<Recipe> getAllRecipes() {
        return recipeService.getAllRecipes();
    }
    @GetMapping("/{id}")
    public ResponseEntity<Recipe> getRecipeById(@PathVariable Long
id) {
        return ResponseEntity.ok(recipeService.getRecipeById(id));
    }

    @PostMapping
    public ResponseEntity<Recipe> addRecipe(@RequestBody Recipe
recipe) {
        return new ResponseEntity<>(recipeService.addRecipe(recipe),
HttpStatus.CREATED);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteRecipe(@PathVariable Long id) {
        recipeService.deleteRecipe(id);
        return ResponseEntity.noContent().build();
    }
}
```

Purpose: Exposes API endpoints for recipes.

Endpoints:

GET /api/recipes → Fetch all recipes

GET /api/recipes/{id} → Fetch a single recipe

POST /api/recipes → Add a new recipe

DELETE /api/recipes/{id} → Delete a recipe

2.5 DTOs (Data Transfer Objects)

DTOs structure API responses to avoid exposing sensitive data.

RecipeDTO.java

```
public class RecipeDTO {
    private Long id;
    private String title;
    private String ingredients;
    private String instructions;

    // Constructor, Getters, and Setters
}
```

Purpose: Used for transferring recipe data in API responses.

2.6 Exception Handling

Custom exceptions improve error handling.

ResourceNotFoundException.java

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String message) {
        super(message);
    }
}
```

Purpose: Throws an error when a requested resource is not found.

GlobalExceptionHandler.java

```
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<String>
handleResourceNotFound(ResourceNotFoundException ex) {
        return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }
}
```

Purpose: Centralized exception handling for API errors.

---

3. API Documentation (Swagger)

To document and test APIs, we integrate Swagger.

Swagger Configuration

SwaggerConfig.java

```
@Configuration
```

```
@EnableSwagger2
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
                .apis(RequestHandlerSelectors.basePackage("com.example.cook
book.controllers"))
                .paths(PathSelectors.any())
                .build();
    }
}
```

Access Swagger UI:
Open http://localhost:8080/swagger-ui.html

---

4. Security (JWT Authentication)

If authentication is needed, Spring Security and JWT can be used.

User Authentication with JWT

Login & Token Generation (/api/auth/login)

Secure Recipe Creation (POST /api/recipes requires authentication)

---

5. Future Enhancements

Implement pagination for large recipe lists.

Add image uploads for recipes.

Implement role-based access control (admin, user).

---

8. State management

1. Understanding State Management in Java

State management in a Spring Boot backend typically involves:

Session-based state (stored in memory, Redis, or database)

JWT-based authentication (stateless authentication)

Application state (managed using services, caches, or databases)

Types of State in Cookbook Application

---

2. Global State Management (Authentication & Session)

Global state management ensures that user authentication and session management persist across multiple requests.

Using JWT (Stateless Authentication)

Instead of storing session data in memory, we use JWT (JSON Web Token) for authentication.

JWT Token Generation

When a user logs in, a JWT token is issued.

```
@Service
public class JwtUtil {
   private String secretKey = "secret"; // Use environment variables for
security

   public String generateToken(String username) {
      return Jwts.builder()
            .setSubject(username)
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 1000 *
60 * 60)) // 1 hour expiration
            .signWith(SignatureAlgorithm.HS256, secretKey)
            .compact();
   }

   public String extractUsername(String token) {
      return Jwts.parser()
            .setSigningKey(secretKey)
            .parseClaimsJws(token)
            .getBody()
            .getSubject();
   }
}
```

JWT Authentication Filter

```
@Component
public class JwtRequestFilter extends OncePerRequestFilter {
    @Autowired
    private JwtUtil jwtUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain chain)
            throws ServletException, IOException {
        final String authorizationHeader =
request.getHeader("Authorization");

        if (authorizationHeader != null &&
authorizationHeader.startsWith("Bearer ")) {
            String token = authorizationHeader.substring(7);
            String username = jwtUtil.extractUsername(token);
            if (username != null) {
                UsernamePasswordAuthenticationToken auth = new
UsernamePasswordAuthenticationToken(username, null, new
ArrayList<>());
                SecurityContextHolder.getContext().setAuthentication(auth);
            }
        }
        chain.doFilter(request, response);
    }
}
```

How it works:

User logs in → JWT token is generated

}

JWT token is sent in Authorization header for every request

Backend extracts the token and authenticates the user

---

3. Persistent State Management (Database Storage)

All recipes, ingredients, and user data are stored in a relational database (MySQL/PostgreSQL) to maintain persistence.

Example: Storing Recipes in a Database

```
@Entity
@Table(name = "recipes")
public class Recipe {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String title;
   private String ingredients;
   private String instructions;

   @ManyToOne
   @JoinColumn(name = "user_id", nullable = false)
   private User createdBy;
}
```

Why this is persistent state?

Data is stored in MySQL/PostgreSQL

Survives application restarts

Repository for CRUD Operations

```
@Repository
public interface RecipeRepository extends JpaRepository<Recipe, Long>
{
    List<Recipe> findByCreatedBy(User user);
}
```

This ensures state persists across multiple users and sessions.

---

4. Local State Management (Temporary Data Storage)

Local state is short-lived and typically exists only during request processing.

Example: Using In-Memory State for Temporary Recipe Drafts

```
@Component
public class RecipeDraftStore {
    private Map<String, Recipe> drafts = new HashMap<>();

    public void saveDraft(String userId, Recipe recipe) {
        drafts.put(userId, recipe);
    }

    public Recipe getDraft(String userId) {
        return drafts.get(userId);
    }
}
```

Why use local state?

Doesn't require database interaction

Useful for temporary drafts or cache

---

5. Caching for Optimized State Management

To avoid frequent database queries, caching can be implemented using Redis.

Example: Caching Recipes Using Redis

```
@Service
public class RecipeCacheService {
    @Autowired
    private RedisTemplate<String, Recipe> redisTemplate;

    public void cacheRecipe(Long id, Recipe recipe) {
        redisTemplate.opsForValue().set("recipe_" + id, recipe, 10,
TimeUnit.MINUTES);
    }
    public Recipe getCachedRecipe(Long id) {
        return redisTemplate.opsForValue().get("recipe_" + id);
    }
}
```

Improves performance by reducing database queries

Expires automatically after a set time

---

6. State Flow in the Application

State Management Lifecycle in Cookbook App

1. User Logs In:

Token is generated (JWT)

Token is stored client-side and sent with every request

2. User Requests Recipes:

Check Redis cache → If present, return cached response

Otherwise, query database (RecipeRepository) and cache result

3. User Creates Recipe:

Recipe data is temporarily stored in memory (RecipeDraftStore)

When user submits, it's stored in the database

4. User Logs Out:

Token is removed client-side (since JWT is stateless)

9. User Interface

1. JavaFX User Interface (Desktop Application)

JavaFX provides a graphical user interface for a desktop-based Cookbook application.

Project Structure (JavaFX with Spring Boot)

```
/src/main/java/com/example/cookbook
  /controllers    # JavaFX Controllers
  /views          # FXML UI Files
  /models         # Java Classes for Recipe, User, etc.
  /services       # Business Logic
  /repositories  # Database Interactions
  /config         # Application Configuration
```

---

2. JavaFX UI Implementation

2.1 JavaFX UI for Recipe Management

Displays a list of recipes

Allows users to add, edit, and delete recipes

2.2 JavaFX FXML Layout (RecipeView.fxml)

```
<VBox alignment="TOP_CENTER" spacing="10"
xmlns="http://javafx.com/javafx"
      xmlns:fx="http://javafx.com/fxml"
      fx:controller="com.example.cookbook.controllers.RecipeController">

    <Label text="Cookbook Recipe Manager" style="-fx-font-size:
20px;"/>

    <TableView fx:id="recipeTable">
        <columns>
            <TableColumn text="Title" fx:id="titleColumn"/>
            <TableColumn text="Ingredients" fx:id="ingredientsColumn"/>
            <TableColumn text="Instructions" fx:id="instructionsColumn"/>
        </columns>
    </TableView>

    <HBox spacing="10">
        <Button text="Add Recipe" onAction="#handleAdd"/>
        <Button text="Edit Recipe" onAction="#handleEdit"/>
        <Button text="Delete Recipe" onAction="#handleDelete"/>
    </HBox>
</VBox>
```

2.3 JavaFX Controller (RecipeController.java)

```
package com.example.cookbook.controllers;

import com.example.cookbook.models.Recipe;
import com.example.cookbook.services.RecipeService;
import javafx.fxml.FXML;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;

public class RecipeController {

    @FXML private TableView<Recipe> recipeTable;
    @FXML private TableColumn<Recipe, String> titleColumn;
    @FXML private TableColumn<Recipe, String> ingredientsColumn;
    @FXML private TableColumn<Recipe, String> instructionsColumn;

    private RecipeService recipeService = new RecipeService();

    @FXML
    public void initialize() {
        titleColumn.setCellValueFactory(cellData ->
cellData.getValue().titleProperty());
        ingredientsColumn.setCellValueFactory(cellData ->
cellData.getValue().ingredientsProperty());
        instructionsColumn.setCellValueFactory(cellData ->
cellData.getValue().instructionsProperty());
```

```
        recipeTable.setItems(recipeService.getAllRecipes());
    }

    @FXML
    public void handleAdd() {
        System.out.println("Adding a new recipe...");
    }

    @FXML
    public void handleEdit() {
        System.out.println("Editing selected recipe...");
    }

    @FXML
    public void handleDelete() {
        System.out.println("Deleting selected recipe...");
    }
}
```

2.4 Running the JavaFX UI

```
mvn clean javafx:run
```

---

3. Web UI with Thymeleaf (Spring Boot)

For a web-based Cookbook, Thymeleaf provides dynamic HTML rendering.

3.1 Thymeleaf Template (recipe-list.html)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Cookbook Recipes</title>
</head>
<body>
    <h1>Cookbook Recipes</h1>
    <table>
        <tr>
            <th>Title</th>
            <th>Ingredients</th>
            <th>Instructions</th>
            <th>Actions</th>
        </tr>
        <tr th:each="recipe : ${recipes}">
            <td th:text="${recipe.title}"></td>
            <td th:text="${recipe.ingredients}"></td>
            <td th:text="${recipe.instructions}"></td>
            <td>
                <a th:href="@{/edit/{id}(id=${recipe.id})}">Edit</a>
                <a th:href="@{/delete/{id}(id=${recipe.id})}">Delete</a>
            </td>
        </tr>
    </table>
    <a href="/add">Add Recipe</a>
</body>
</html>
```

3.2 RecipeController (Spring Boot)

```
@Controller
@RequestMapping("/recipes")
public class RecipeController {

    @Autowired
    private RecipeService recipeService;

    @GetMapping
    public String listRecipes(Model model) {
        model.addAttribute("recipes", recipeService.getAllRecipes());
        return "recipe-list";
    }

    @GetMapping("/add")
    public String showAddForm(Model model) {
        model.addAttribute("recipe", new Recipe());
        return "recipe-form";
    }

    @PostMapping("/save")
    public String saveRecipe(@ModelAttribute Recipe recipe) {
        recipeService.addRecipe(recipe);
        return "redirect:/recipes";
    }
}
```

3.3 Running the Web UI

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

Access the web app: http://localhost:8080/recipes

---

4. UI Enhancements

4.1 Adding CSS to JavaFX

Create styles.css:

```
.root {
   -fx-background-color: #f4f4f4;
}

.button {
   -fx-background-color: #4CAF50;
   -fx-text-fill: white;
}
```

Apply it in start():

```
scene.getStylesheets().add(getClass().getResource("styles.css").toExternalForm());
```

4.2 Making the UI Responsive in Thymeleaf

Add Bootstrap:

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
```

---

5. Choosing Between JavaFX and Thymeleaf

---

10. Styling

1. Styling in JavaFX (Desktop UI)

JavaFX uses CSS for styling components. You can apply custom themes, animations, and layouts.

1.1 Creating a Stylesheet (styles.css)

Create a CSS file styles.css in the resources folder:

```
.root {
   -fx-background-color: #f4f4f4;
}

.label {
   -fx-font-size: 18px;
   -fx-text-fill: #333;
}

.button {
   -fx-background-color: #4CAF50;
   -fx-text-fill: white;
   -fx-padding: 10px;
   -fx-font-size: 14px;
   -fx-border-radius: 5px;
}
```

```
.table-view {
    -fx-border-color: #ddd;
    -fx-background-color: white;
}

.table-row-cell:selected {
    -fx-background-color: #a5d6a7;
    -fx-text-fill: black;
}
```

1.2 Applying Styles to JavaFX Scene

In start() method of JavaFX Application:

```
@Override
public void start(Stage primaryStage) throws Exception {
    Parent root =
FXMLLoader.load(getClass().getResource("RecipeView.fxml"));
    Scene scene = new Scene(root);

scene.getStylesheets().add(getClass().getResource("styles.css").toExternal
Form());

    primaryStage.setTitle("Cookbook");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

This ensures the JavaFX UI follows the defined styles

Buttons, labels, and tables will have a uniform theme

1.3 Inline Styling for JavaFX Components

You can also apply styles directly in Java:

```
Button addButton = new Button("Add Recipe");
addButton.setStyle("-fx-background-color: #ff9800; -fx-text-fill: white;
-fx-padding: 10px;");
```

Use this for dynamic styling or theme changes at runtime

---

2. Styling in Thymeleaf (Web UI with Spring Boot)

For web applications using Thymeleaf, CSS frameworks like Bootstrap or custom CSS can be used.

2.1 Adding Bootstrap for a Responsive UI

Include Bootstrap in recipe-list.html:

```
<head>
   <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.
css">
</head>
```

Bootstrap provides buttons, grids, and tables with modern design

---

2.2 Custom CSS for Web UI (static/styles.css)

Create a styles.css file in src/main/resources/static/:

```
body {
   background-color: #f4f4f4;
   font-family: Arial, sans-serif;
}

h1 {
   color: #4CAF50;
}

.table {
   background-color: white;
   border-radius: 5px;
}

.btn-primary {
   background-color: #4CAF50;
   border: none;
}

.btn-primary:hover {
   background-color: #45a049;
}
```

2.3 Linking CSS in Thymeleaf Template

Modify recipe-list.html:

```
<head>
   <link rel="stylesheet" th:href="@{/styles.css}">
</head>
```

Now all pages using this template will have the custom styles

---

3. Theming Support in JavaFX

JavaFX allows dynamic theme switching using CSS.

3.1 Creating Light and Dark Themes

Create light-theme.css:

```
.root {
   -fx-background-color: white;
   -fx-text-fill: black;
}
.button {
   -fx-background-color: #4CAF50;
}
```

Create dark-theme.css:

```
.root {
    -fx-background-color: #333;
    -fx-text-fill: white;
}
.button {
    -fx-background-color: #555;
}
```

3.2 Switching Themes Dynamically

```
public void switchTheme(String theme) {
    scene.getStylesheets().clear();
    if (theme.equals("dark")) {
        scene.getStylesheets().add(getClass().getResource("dark-
theme.css").toExternalForm());
    } else {
        scene.getStylesheets().add(getClass().getResource("light-
theme.css").toExternalForm());
    }
}
```

Call switchTheme("dark") or switchTheme("light") to change themes at runtime

---

4. Animations in JavaFX

JavaFX supports animations using TranslateTransition, FadeTransition, and RotateTransition.

4.1 Adding Fade Animation to Buttons

```
FadeTransition fade = new FadeTransition(Duration.millis(500),
addButton);
fade.setFromValue(0.5);
fade.setToValue(1.0);
fade.setCycleCount(Timeline.INDEFINITE);
fade.setAutoReverse(true);
fade.play();
```

Creates a pulsing animation effect on the "Add Recipe" button

---

5. Material Design UI (For Web UI)

For a modern look, integrate Google Material Design with Thymeleaf:

5.1 Using Material Icons

```
<head>
   <link rel="stylesheet" href="https://fonts.googleapis.com/icon?
family=Material+Icons">
</head>
<button class="btn btn-primary">
   <i class="material-icons">add</i> Add Recipe
</button>
```

---

11. Testing

1. Types of Testing

---

2. Unit Testing with JUnit & Mockito

Unit testing ensures individual methods work as expected.

2.1 Adding Dependencies (Maven)

```
<dependencies>
    <!-- JUnit 5 -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <scope>test</scope>
    </dependency>

    <!-- Mockito for mocking dependencies -->
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

2.2 Unit Test for RecipeService

RecipeService.java

```java
package com.example.cookbook.services;

import com.example.cookbook.models.Recipe;
import java.util.ArrayList;
import java.util.List;

public class RecipeService {
    private List<Recipe> recipes = new ArrayList<>();

    public void addRecipe(Recipe recipe) {
        recipes.add(recipe);
    }

    public List<Recipe> getAllRecipes() {
        return recipes;
    }

    public Recipe findRecipeByTitle(String title) {
        return recipes.stream()
                .filter(recipe -> recipe.getTitle().equalsIgnoreCase(title))
                .findFirst()
                .orElse(null);
    }
}
```

Unit Test for RecipeService

```java
package com.example.cookbook.services;

import com.example.cookbook.models.Recipe;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class RecipeServiceTest {
    private RecipeService recipeService;

    @BeforeEach
    void setUp() {
        recipeService = new RecipeService();
    }

    @Test
    void testAddRecipe() {
        Recipe recipe = new Recipe("Pasta", "Tomatoes, Garlic", "Boil
pasta");
        recipeService.addRecipe(recipe);
        assertEquals(1, recipeService.getAllRecipes().size());
    }

    @Test
    void testFindRecipeByTitle() {
        Recipe recipe = new Recipe("Pizza", "Cheese, Dough", "Bake in
oven");
        recipeService.addRecipe(recipe);
        Recipe foundRecipe = recipeService.findRecipeByTitle("Pizza");

        assertNotNull(foundRecipe);
        assertEquals("Pizza", foundRecipe.getTitle());
    }
}
```

✅ Run Tests:

```
mvn test
```

---

3. Integration Testing (Spring Boot)

Integration testing verifies database interactions, API endpoints, and service-layer logic.

3.1 Adding Dependencies (Spring Boot)

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

3.2 Testing REST API (RecipeControllerTest)

RecipeController.java

```java
@RestController
@RequestMapping("/recipes")
public class RecipeController {
    @Autowired
    private RecipeService recipeService;

    @GetMapping
    public List<Recipe> getAllRecipes() {
        return recipeService.getAllRecipes();
    }
    @PostMapping
    public ResponseEntity<String> addRecipe(@RequestBody Recipe
recipe) {
        recipeService.addRecipe(recipe);
        return ResponseEntity.ok("Recipe added successfully");
    }
}
```

Integration Test for API (RecipeControllerTest.java)

```
@SpringBootTest(webEnvironment =
SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureMockMvc
public class RecipeControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testAddRecipe() throws Exception {
        String recipeJson = "{ \"title\": \"Burger\", \"ingredients\": \"Bread,
Meat\", \"instructions\": \"Grill the meat\" }";

        mockMvc.perform(post("/recipes")
                .contentType(MediaType.APPLICATION_JSON)
                .content(recipeJson))
                .andExpect(status().isOk())
                .andExpect(content().string("Recipe added successfully"));
    }

    @Test
    void testGetAllRecipes() throws Exception {
        mockMvc.perform(get("/recipes"))
```

```
            .andExpect(status().isOk())
            .andExpect(jsonPath("$").isArray());
    }
}
```

✅ Run API Tests:

```
mvn test
```

---

4. UI Testing (JavaFX & Web UI)

4.1 JavaFX UI Testing with TestFX

Adding Dependencies

```
<dependency>
    <groupId>org.testfx</groupId>
    <artifactId>testfx-junit5</artifactId>
    <scope>test</scope>
</dependency>
```

Testing JavaFX Button Click

```
import static org.testfx.api.FxAssert.verifyThat;
import static org.testfx.matcher.control.LabeledMatchers.hasText;

public class RecipeUITest extends ApplicationTest {

    @Test
    void testAddRecipeButton() {
        clickOn("#addRecipeButton");
        verifyThat("#recipeTitleField", hasText(""));
    }
}
```

✅ Run UI Test:

```
mvn test
```

4.2 Web UI Testing with Selenium

Adding Dependencies

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <scope>test</scope>
</dependency>
```

Selenium Test for Web UI

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.By;
import org.junit.jupiter.api.*;

public class RecipeUITest {
    private WebDriver driver;

    @BeforeEach
    void setUp() {
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");
        driver = new ChromeDriver();
    }

    @Test
    void testAddRecipeForm() {
        driver.get("http://localhost:8080/recipes");
        driver.findElement(By.id("addRecipe")).click();
        driver.findElement(By.name("title")).sendKeys("Salad");
        driver.findElement(By.name("ingredients")).sendKeys("Lettuce,
Tomato");
        driver.findElement(By.name("instructions")).sendKeys("Mix
everything");
        driver.findElement(By.id("submit")).click();

        Assertions.assertTrue(driver.getPageSource().contains("Salad"));
    }

    @AfterEach
    void tearDown() {
        driver.quit();
    }
}
```

✅ Run Selenium Test:

```
mvn test
```

---

5. Code Coverage with JaCoCo

JaCoCo helps measure test coverage.

5.1 Add JaCoCo to pom.xml

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.7</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

5.2 Generate Coverage Report

```
mvn clean test jacoco:report
```

Reports saved in: target/site/jacoco/index.html
---

12. Screenshots or Demo

1. Taking Screenshots (JavaFX Desktop App)

If your Cookbook application is built with JavaFX, you can capture UI screenshots using JavaFX’s Robot API.

1.1 Programmatically Capture Screenshot

Add the following code to capture a JavaFX scene:

```
import javafx.scene.image.WritableImage;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.embed.swing.SwingFXUtils;
import javax.imageio.ImageIO;
import java.io.File;
import java.io.IOException;

public class ScreenshotUtil {
    public static void takeScreenshot(Scene scene, String filename) {
        WritableImage image = new WritableImage((int) scene.getWidth(),
(int) scene.getHeight());
        scene.snapshot(image);

        File file = new File(filename);
        try {
            ImageIO.write(SwingFXUtils.fromFXImage(image, null), "png",
file);
            System.out.println("Screenshot saved: " + filename);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

✅ Usage: Call ScreenshotUtil.takeScreenshot(scene, "cookbook.png"); inside your JavaFX application.

1.2 Manually Capture JavaFX UI

1. Run the Cookbook application

2. Press PrtSc (Windows) or Cmd + Shift + 4 (Mac)

3. Save and upload the screenshot

---

2. Recording a Demo (JavaFX Desktop App)

For a video demo, use:

OBS Studio (Free and open-source screen recorder)

Loom (Online screen recording tool)

Windows Game Bar (Win + G for screen recording)

---

3. Taking Screenshots (Web-Based Cookbook)

If your Cookbook is a web app (Spring Boot + Thymeleaf/React), capture it via:

1. Open Google Chrome or Firefox

2. Press Ctrl + Shift + S (Windows) or Cmd + Shift + 4 (Mac)

3. Save and share the screenshot

Alternatively, use Selenium WebDriver to take a screenshot programmatically.

3.1 Selenium Screenshot Code

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import java.io.File;
import java.io.IOException;
import org.apache.commons.io.FileUtils;

public class WebScreenshot {
    public static void main(String[] args) throws IOException {
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");
        WebDriver driver = new ChromeDriver();
        driver.get("http://localhost:8080/cookbook");

        File srcFile = ((TakesScreenshot)
driver).getScreenshotAs(OutputType.FILE);
        FileUtils.copyFile(srcFile, new File("cookbook-web.png"));

        System.out.println("Screenshot saved: cookbook-web.png");
        driver.quit();
    }
```

}

✅ Run it:

```
mvn test
```

---

4. Hosting a Live Demo

To showcase the Cookbook app online, deploy it to a cloud platform:

✅ Spring Boot Deployment Example on Heroku

```
heroku login
heroku create cookbook-app
git push heroku main
heroku open
```

---

5. Adding Screenshots to Documentation

If writing a project report or README, embed images like this:

5.1 Markdown (README.md)

```
# Cookbook Application

## Home Page Screenshot
![Home Page](screenshots/home.png)
```

## Recipe Details Page
![Recipe Page](screenshots/recipe.png)

5.2 HTML (Thymeleaf/Web App)

<img src="screenshots/home.png" alt="Cookbook Home Page" width="600px">

---

6. Sharing Video Demos

Upload your demo video to:

YouTube (Public or Unlisted)

Google Drive (Shareable link)

Loom (Quick screen recording)

Example YouTube Video Embed in Markdown:

[![Watch the demo](https://img.youtube.com/vi/YOUR_VIDEO_ID/maxresdefault.jpg)](https://www.youtube.com/watch?v=YOUR_VIDEO_ID)

---

7. Summary

✔ Screenshots: JavaFX Robot API, Selenium, or Manual Capture
✔ Demo Video: OBS, Loom, or Game Bar
✔ Live Hosting: Heroku, Render, Netlify
✔ Documentation: Embed images in Markdown/HTML
---

13. Known Issues

1. Functional Issues

1.1 Recipe Duplication Issue

Description: When adding a new recipe, users can add duplicate recipes without validation.
Impact: Results in multiple identical recipes in the list.
Possible Fix: Implement a duplicate check in RecipeService.

```
public void addRecipe(Recipe recipe) {
    if (recipes.stream().anyMatch(r ->
r.getTitle().equalsIgnoreCase(recipe.getTitle()))) {
        throw new IllegalArgumentException("Recipe already exists!");
    }
    recipes.add(recipe);
}
```

1.2 Search Function Case Sensitivity

Description: The search functionality is case-sensitive, so searching for “pasta” and “Pasta” gives different results.
Impact: Reduces user experience.
Possible Fix: Convert input and stored recipe titles to lowercase before

searching.

```
public Recipe findRecipeByTitle(String title) {
    return recipes.stream()
        .filter(recipe -> recipe.getTitle().equalsIgnoreCase(title))
        .findFirst()
        .orElse(null);
}
```

1.3 Slow Performance with Large Datasets

Description: The application slows down when handling a large number of recipes, especially in JavaFX TableView or when fetching data from the database.
Impact: Affects scalability and user experience.
Possible Fix: Implement pagination for recipe retrieval:

```
public List<Recipe> getRecipesPaginated(int page, int size) {
    return recipes.stream()
        .skip((long) page * size)
        .limit(size)
        .collect(Collectors.toList());
}
```

---

2. UI/UX Issues

2.1 UI Freezes When Fetching Recipes

Description: When loading a large list of recipes, the JavaFX UI freezes because fetching runs on the main thread.
Impact: Poor responsiveness.
Possible Fix: Run data retrieval in a background thread using Task<Void> in JavaFX.

```
Task<Void> task = new Task<>() {
    @Override
    protected Void call() {
        List<Recipe> recipes = recipeService.getAllRecipes();
        Platform.runLater(() ->
recipeTableView.setItems(FXCollections.observableList(recipes)));
        return null;
    }
};
new Thread(task).start();
```

2.2 Unresponsive Buttons in JavaFX

Description: Sometimes, buttons do not respond immediately after clicking.
Impact: Users need to double-click, leading to confusion.
Possible Fix: Ensure all UI updates are handled inside Platform.runLater().

```
button.setOnAction(event -> {
    Platform.runLater(() -> showRecipeDetails(selectedRecipe));
});
```

---

3. API Issues (For Web-Based Cookbook)

3.1 API Returns Incorrect Status Code

Description: The API sometimes returns 200 OK even when an error occurs.
Impact: Misleads the frontend, making error handling difficult.
Possible Fix: Ensure the correct HTTP status codes are returned.

```
@PostMapping("/add")
public ResponseEntity<?> addRecipe(@RequestBody Recipe recipe) {
    try {
        recipeService.addRecipe(recipe);
        return ResponseEntity.ok("Recipe added successfully");
    } catch (Exception e) {
        return
ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Error: " +
e.getMessage());
    }
}
```

3.2 Missing Validation for Empty Fields

Description: Users can submit empty recipe fields in a Spring Boot API, leading to database errors.
Impact: Causes incomplete or invalid data.
Possible Fix: Use Spring Validation to enforce required fields.

```
public class Recipe {
    @NotBlank(message = "Title is required")
    private String title;
    @NotBlank(message = "Ingredients cannot be empty")
    private String ingredients;

    @NotBlank(message = "Instructions cannot be empty")
    private String instructions;
}
```

---

4. Database Issues

4.1 SQL Injection Risk

Description: If the application directly concatenates user input into SQL queries, it becomes vulnerable to SQL injection.
Impact: Attackers could manipulate queries to delete or access unauthorized data.
Possible Fix: Use Prepared Statements with JDBC.

```
String sql = "SELECT * FROM recipes WHERE title = ?";
PreparedStatement stmt = connection.prepareStatement(sql);
stmt.setString(1, userInput);
ResultSet rs = stmt.executeQuery();
```

4.2 Data Not Persisting After Restart

Description: If using an in-memory database (H2, HSQLDB), all data is lost when the server restarts.
Impact: Causes data loss.

Possible Fix: Switch to a persistent database like MySQL or PostgreSQL. Update application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/cookbook
spring.datasource.username=root
spring.datasource.password=yourpassword
```

---

5. Security Issues

5.1 No User Authentication

Description: Anyone can add, delete, or modify recipes without authentication.
Impact: Unauthorized users can modify or delete data.
Possible Fix: Implement Spring Security for authentication and authorization.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http)
throws Exception {
        http.authorizeHttpRequests(auth -> auth
            .requestMatchers("/recipes/**").authenticated()
            .anyRequest().permitAll()
        ).formLogin(Customizer.withDefaults());
        return http.build();
    }
}
```

5.2 Hardcoded Credentials in Code

Description: Database credentials are hardcoded in Java code, making them visible in repositories.
Impact: Security risk if exposed.
Possible Fix: Store credentials in environment variables or a .env file.

```
String dbUser = System.getenv("DB_USER");
String dbPassword = System.getenv("DB_PASSWORD");
```

---

14. Future Enhancements

1. Functional Enhancements

1.1 User Authentication & Role-Based Access Control

Current Issue:

The app does not have authentication; anyone can add, edit, or delete recipes.

Enhancement:

Implement Spring Security with JWT authentication for user login and role-based access control.

Admin users can manage all recipes, while regular users can only manage their own.

✅ Example using Spring Security & JWT:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http)
throws Exception {
        http.authorizeHttpRequests(auth -> auth
            .requestMatchers("/admin/**").hasRole("ADMIN")
            .requestMatchers("/recipes/**").authenticated()
            .anyRequest().permitAll()
        ).formLogin(Customizer.withDefaults());
        return http.build();
    }
}
```

1.2 Recipe Rating & Reviews

Current Issue:

No way for users to rate recipes or leave reviews.

Enhancement:

Allow users to rate recipes (1–5 stars) and leave comments.

Store ratings in the database and calculate average ratings for each recipe.

✅ Example Recipe Entity Update:

```
@Entity
public class Recipe {
    private int ratingCount;
    private double averageRating;
}
```

✅ Sample API for Adding Reviews:

```
@PostMapping("/{recipeId}/rate")
public ResponseEntity<?> addRating(@PathVariable Long recipeId,
@RequestBody Rating rating) {
    recipeService.addRating(recipeId, rating);
    return ResponseEntity.ok("Rating added successfully");
}
```

1.3 Recipe Categories & Filtering

Current Issue:

All recipes are listed together, making it hard to find specific ones.

Enhancement:

Add categories (Vegetarian, Desserts, Beverages, etc.).

Allow users to filter recipes by category, ingredients, or cooking time.

✅ Example Filter Query in Spring Boot:

```
@GetMapping("/recipes")
public List<Recipe> getRecipesByCategory(@RequestParam String category) {
    return recipeRepository.findByCategory(category);
}
```

---

2. UI & UX Enhancements

2.1 Dark Mode & Theming

Current Issue:

The UI is static, with no customization options.

Enhancement:

Implement Dark Mode and allow users to switch themes.

✅ Example JavaFX CSS for Dark Mode:

```
.root {
    -fx-background-color: #2E2E2E;
    -fx-text-fill: white;
}
```

✅ Toggle Theme in JavaFX:

```
if (isDarkMode) {
    scene.getStylesheets().add("dark-theme.css");
} else {
    scene.getStylesheets().add("light-theme.css");
}
```

2.2 Responsive Web Design (For Web Version)

Current Issue:

The web version (if using Spring Boot + Thymeleaf/React) may not be mobile-friendly.

Enhancement:

Use Bootstrap or Material UI for a responsive layout.

Ensure pages look good on phones, tablets, and desktops.

✅ Example: Adding Bootstrap in Thymeleaf:

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.
css">
```

2.3 Meal Planning & Shopping List Feature

Current Issue:

Users can view recipes but cannot plan meals or generate a shopping list.

Enhancement:

Allow users to schedule recipes for the week.

Generate a shopping list with needed ingredients.

✅ Example Java Class for Meal Planning:

```
@Entity
public class MealPlan {
    @Id @GeneratedValue
    private Long id;
    private LocalDate date;

    @ManyToOne
    private Recipe recipe;
}
```

---

3. Performance & Optimization Enhancements

3.1 Implement Caching for Faster Load Times

Current Issue:

Fetching recipes from the database every time makes the app slow.

Enhancement:

Use Spring Cache to store recipe data in memory and reduce DB queries.

✅ Enable Caching in Spring Boot:

```
@EnableCaching
@SpringBootApplication
public class CookbookApplication {
}
```

✅ Cache Recipes for Faster Access:

```
@Cacheable("recipes")
public List<Recipe> getAllRecipes() {
    return recipeRepository.findAll();
}
```

3.2 Optimize Database Queries

Current Issue:

Large recipe collections slow down search and filtering.

Enhancement:

Use indexed columns and pagination in queries.

✅ Index Recipe Title for Faster Search:

```
CREATE INDEX idx_recipe_title ON recipe(title);
```

✅ Paginate Large Recipe Lists:

```
Page<Recipe> findAll(Pageable pageable);
```

---

4. New Features & Integrations

4.1 Export & Share Recipes

Current Issue:

Users cannot export or share their recipes easily.

Enhancement:

Allow exporting recipes in PDF, CSV, or JSON formats.

✅ Example PDF Export using iText:

```
Document document = new Document();
```

```
PdfWriter.getInstance(document, new FileOutputStream("recipe.pdf"));
document.open();
document.add(new Paragraph(recipe.getTitle()));
document.close();
```

4.2 Voice Command Integration

Current Issue:

Users must manually browse recipes, which can be inconvenient while cooking.

Enhancement:

Add voice commands using Java’s Speech API.

✅ Example Speech Recognition in Java:

```
Recognizer recognizer = Central.createRecognizer(new
EngineModeDesc(Locale.US));
recognizer.allocate();
```

4.3 AI-Powered Recipe Suggestions

Current Issue:

Users must manually search for recipes.

Enhancement:

Implement AI suggestions based on available ingredients.

✅ Example AI-based Recipe Recommendation:

```
public List<Recipe> suggestRecipes(List<String> ingredients) {
   return recipeRepository.findByIngredientsIn(ingredients);
}
```

---

5. Deployment & Scalability Enhancements

5.1 Deploy to Cloud (AWS, Heroku, Docker)

Current Issue:

The app is only available locally.

Enhancement:

Deploy it to AWS, Heroku, or Docker for public access.
