PROJECT REPORT: RECIPE MANAGER

Name: Nandhu Jayan

Reg No: 22PMC139

BRIEF DESCRIPTION

The Recipe Manager is a Java-based recipe management system that allows users to efficiently manage their recipes, create shopping lists, and search for recipes by ingredients. This project provides a graphical user interface (GUI) for users to interact with the system. The Recipe Manager aims to simplify the process of organizing and accessing cooking recipes, making it a valuable tool for home cooks and culinary enthusiasts.

FEATURES

1. RECIPE MANAGEMENT

- **Add Recipe:** Users can add new recipes to the system by providing a name, list of ingredients, and cooking instructions. The system stores these recipes for future reference.
- **Edit Recipe:** Existing recipes can be edited, allowing users to update the name, ingredients, or cooking instructions of a recipe.
- **Delete Recipe:** Users can remove recipes from the system, with a confirmation prompt to avoid accidental deletions.

2. RECIPE CATEGORIZATION

- **Recipe Categories:** Recipes can be categorized into different groups or categories. This feature helps users organize their recipes based on meal types, cuisines, or any custom categories they define.

3. SHOPPING LIST

- **Add to Shopping List:** Users can select ingredients from recipes and add them to a shopping list. This list helps users keep track of the ingredients they need to purchase for their planned recipes.

4. SEARCH FUNCTIONALITY

- **Search by Ingredients**: Users can search for recipes by specifying ingredients they have on hand. The system displays matching recipes, making it easy for users to find recipes that suit the ingredients they have available.
- **Search by Name**: Users can search for recipes by their names. This feature is particularly useful when users remember the name of a recipe but not the ingredients.

IMPLEMENTATION

The Recipe Manager is implemented as a Java application using the JavaFX library for the graphical user interface. Here are the key components and technologies used:

- **JavaFX**: The GUI is built using JavaFX, which provides a user-friendly interface with various elements such as text fields, buttons, and lists.
- **ObservableList**: The `ObservableList` data structure is used to manage and display recipes, recipe categories, and shopping lists. It allows dynamic updates to reflect changes in real-time.
- **Dialogs and Alerts**: Dialogs and alerts are used to interact with the user for actions like adding, editing, and deleting recipes, confirming deletions, and adding ingredients to the shopping list.
- **Event Handling**: Event listeners are employed to handle user interactions with buttons and text fields, ensuring that the application responds to user actions.
- **Search Functionality**: The search functionality is implemented by filtering recipes based on user input (ingredients or recipe names) and updating the displayed recipes accordingly.

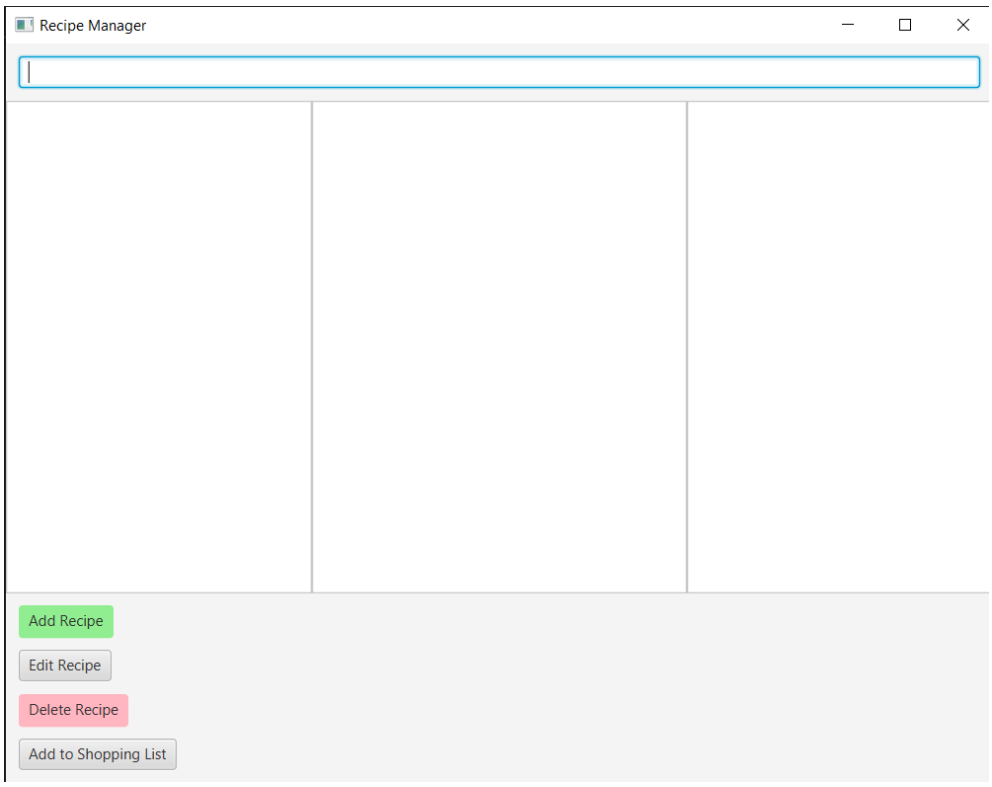
WIDGETS USED

In the Recipe Manager project, several JavaFX widgets are used to create the graphical user interface (GUI). Here are the different widgets and controls used in the project:

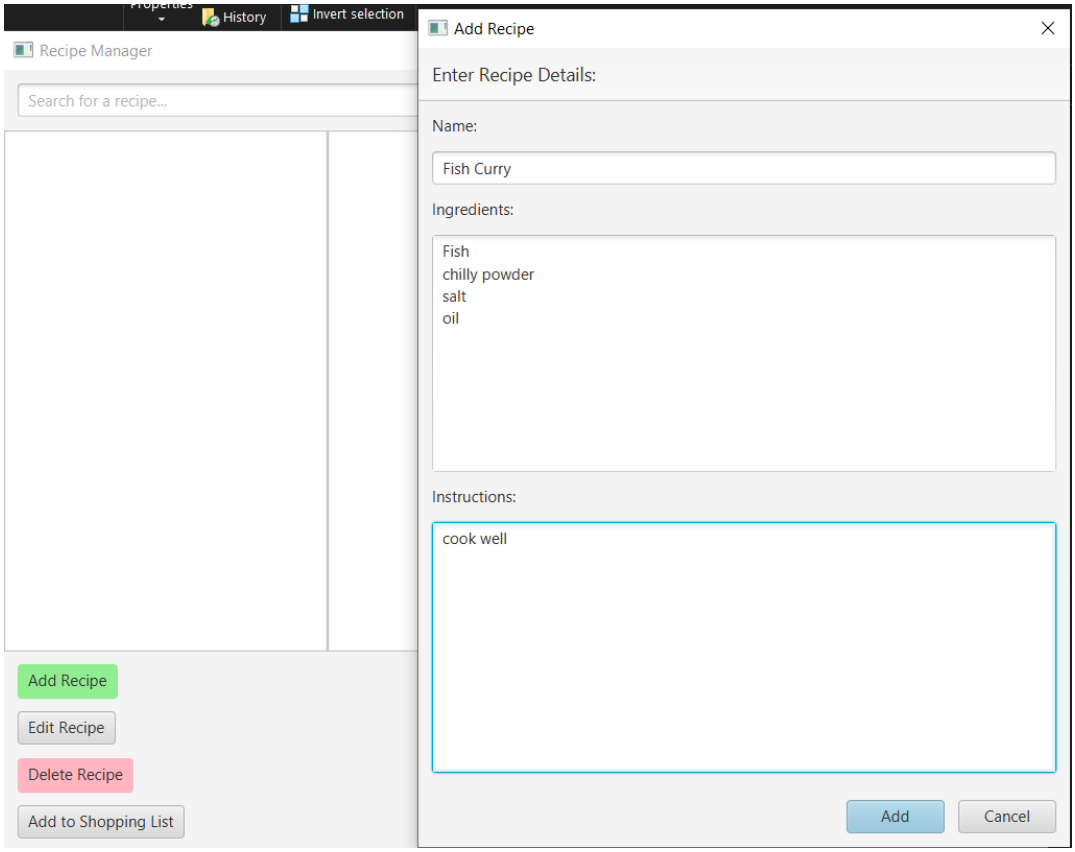
- **ListView**: ListView is used to display lists of items. In this project, it's used to display lists of recipes, recipe categories, and the shopping list.
- **Button**: Buttons are used to trigger actions in the application, such as adding, editing, and deleting recipes, as well as adding ingredients to the shopping list.
- **TextField**: TextField is used for user input, such as the search bar where users can enter keywords to search for recipes.
- **TextArea**: TextArea is used to input and display multi-line text. In this project, it's used for entering recipe details like ingredients and instructions.
- **ChoiceDialog**: ChoiceDialog is used to create a dialog box that presents users with a list of choices. In the project, it's used to select ingredients to add to the shopping list.
- **Alert**: The Alert control is used to display pop-up alerts, including confirmation dialogs when deleting recipes.
- **VBox**: VBox (Vertical Box) is a layout container used to stack UI components vertically. It's used to organize elements like buttons and text fields in a vertical arrangement.
- **BorderPane**: BorderPane is a layout container used to arrange UI elements in five regions: top, right, bottom, left, and center. It's used to create the main layout of the application, placing the search bar at the top, recipe lists on the left and center, and buttons on the bottom and right.

SCREENSHOT

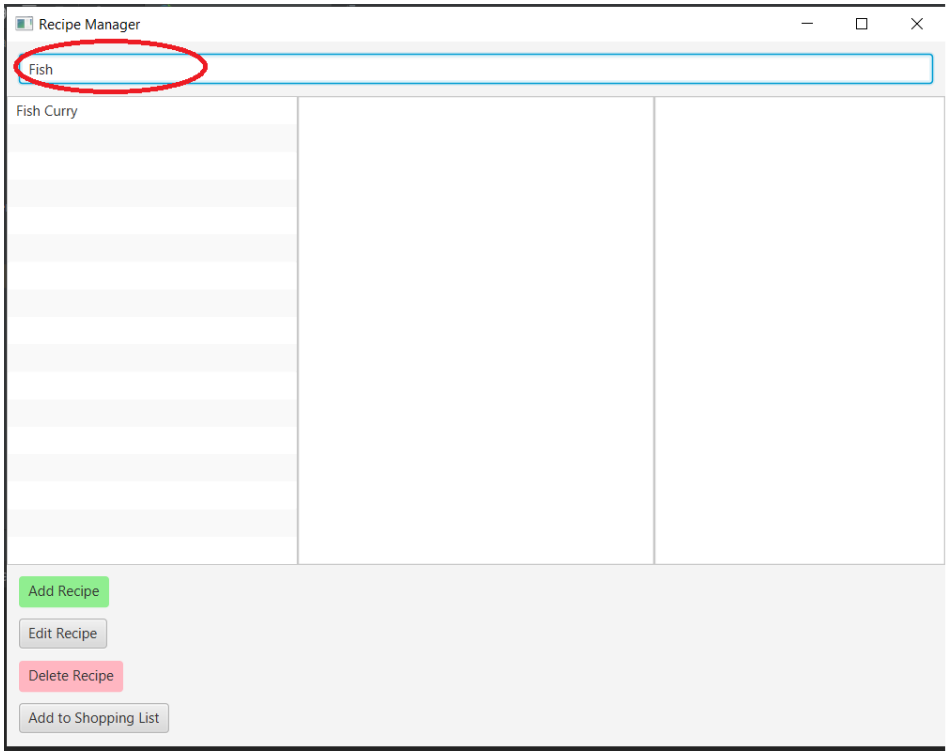
1. Home Page



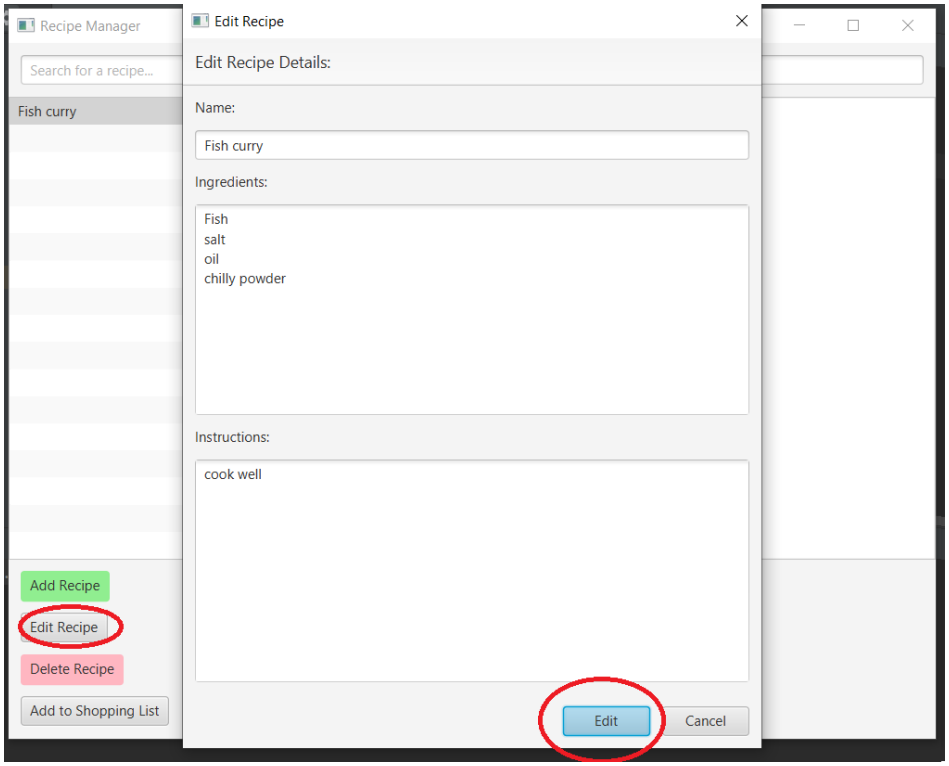
2. Add Recipe



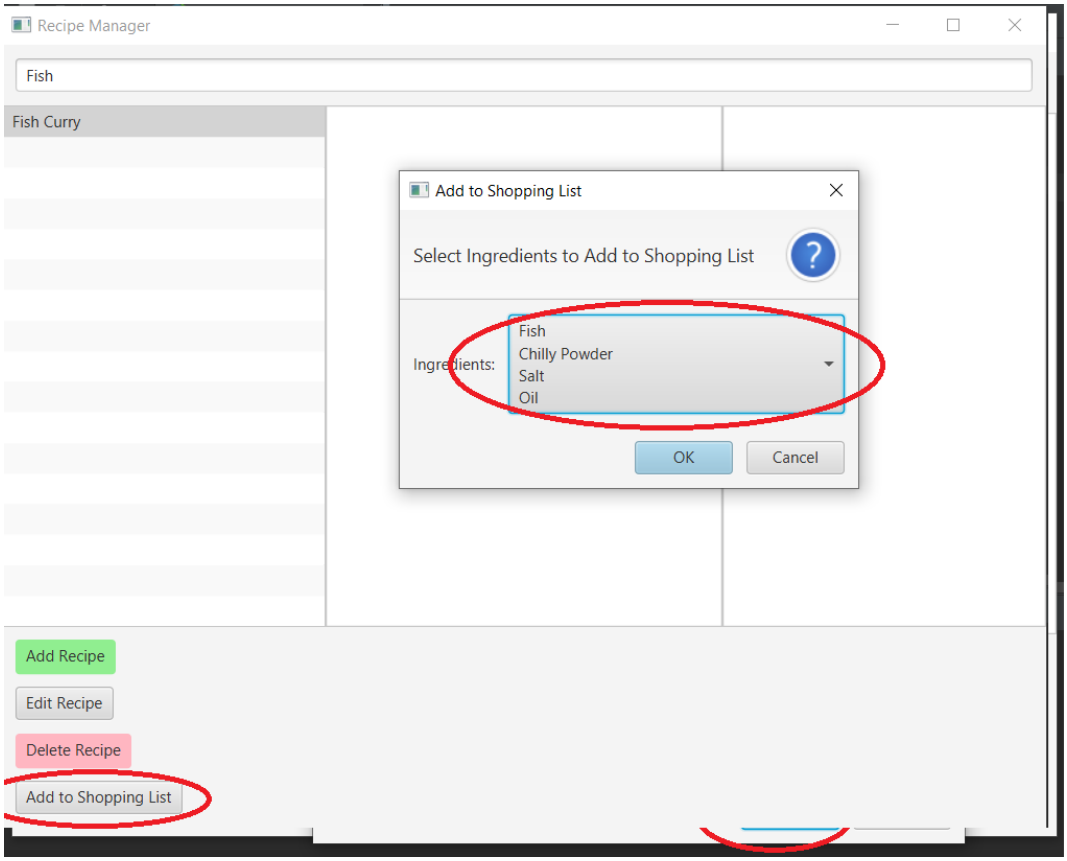
3.Search Bar



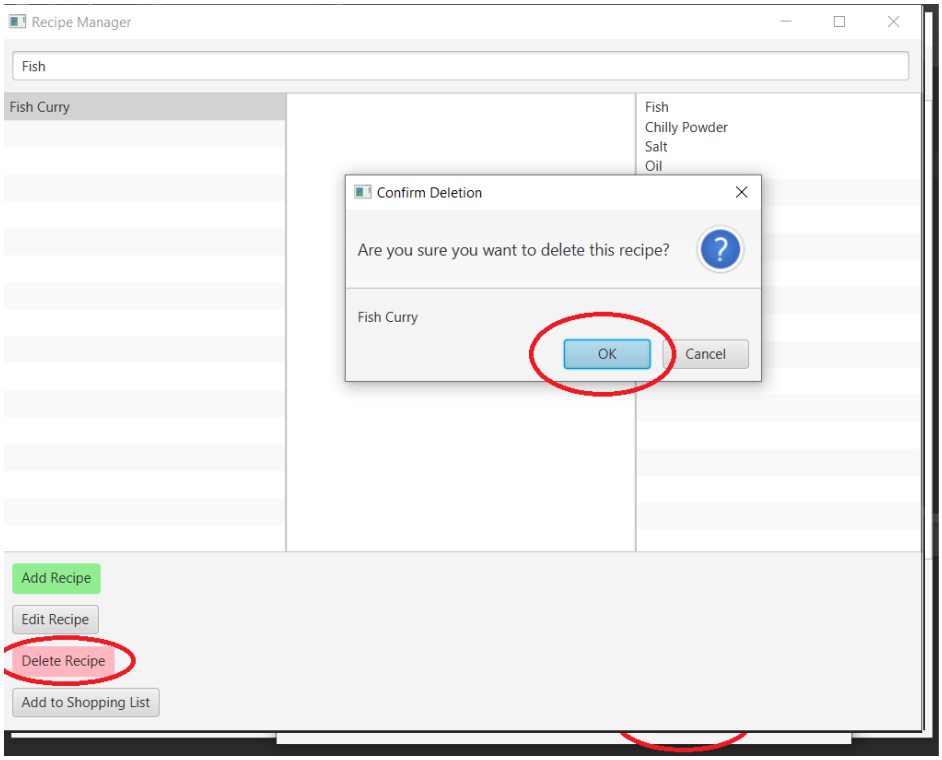
4. Edit Recipe



5.Add To shopping List



6.Delete Recipe



CODE

```
package com.example.java_assignment1;
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;


public class RecipeManager extends Application {

    private ObservableList<Recipe> recipes;
    private ListView<Recipe> recipeListView;
    private ListView<RecipeCategory> categoryListView;
    private ListView<String> shoppingListView;

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Recipe Manager");

        // Initialize the recipes list
        recipes = FXCollections.observableArrayList();

        // Create a ListView to display recipes
        recipeListView = new ListView<>();
        recipeListView.setItems(recipes);

        // Create a ListView to display recipe categories
        categoryListView = new ListView<>();
        categoryListView.setItems(FXCollections.observableArrayList());

        // Create a ListView to display shopping list items
        shoppingListView = new ListView<>();
        shoppingListView.setItems(FXCollections.observableArrayList());

        // Create buttons for adding, editing, and deleting recipes
        Button addButton = new Button("Add Recipe");
        addButton.setStyle("-fx-background-color: #90EE90;"); // Set the
background color to green
        addButton.setOnAction(e -> addRecipe());


        Button editButton = new Button("Edit Recipe");
        editButton.getStyleClass().add("button-edit");

        editButton.setOnAction(e -> editRecipe());
```

```
        Button deleteButton = new Button("Delete Recipe");
        deleteButton.getStyleClass().add("button-delete");
        deleteButton.setStyle("-fx-background-color: #FFB6C1;"); // Light
red color
        deleteButton.setOnAction(e -> deleteRecipe());

        Button addToShoppingListButton = new Button("Add to Shopping
List");
        addToShoppingListButton.getStyleClass().add("button-add-to-list");
        addToShoppingListButton.setOnAction(e -> addToShoppingList());

        TextField searchField = new TextField();
        searchField.setPromptText("Search for a recipe...");
        searchField.setOnKeyReleased(e ->
searchRecipes(searchField.getText()));

        // Create a VBox to hold the search bar
        VBox searchBarBox = new VBox(10);
        searchBarBox.setPadding(new Insets(10));
        searchBarBox.getChildren().addAll(searchField);

        // Create a VBox to hold the buttons
        VBox buttonBox = new VBox(10);
        buttonBox.setPadding(new Insets(10));
        buttonBox.getChildren().addAll(addButton, editButton, deleteButton,
addToShoppingListButton);

        // Create a BorderPane to arrange the UI elements
        BorderPane mainLayout = new BorderPane();
        mainLayout.setTop(searchBarBox); // Place the search bar at the top
        mainLayout.setLeft(recipeListView);
        mainLayout.setCenter(categoryListView);
        mainLayout.setRight(shoppingListView);
        mainLayout.setBottom(buttonBox);

        Scene scene = new Scene(mainLayout, 800, 600);
        //
scene.getStylesheets().add(getClass().getResource("styles.css").toExternalF
orm());

        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void addRecipe() {
        // Create a dialog to add a new recipe
        Dialog<Recipe> dialog = new Dialog<>();
        dialog.setTitle("Add Recipe");
        dialog.setHeaderText("Enter Recipe Details:");

        // Set the button types (OK and Cancel)
        ButtonType addButton = new ButtonType("Add",
ButtonBar.ButtonData.OK_DONE);
        dialog.getDialogPane().getButtonTypes().addAll(addButton,
ButtonType.CANCEL);

        // Create and configure the recipe details form
        VBox form = new VBox();
        form.setSpacing(10);
        form.setPadding(new Insets(10));
```

```
        TextField nameField = new TextField();
        nameField.setPromptText("Name");

        TextArea ingredientsField = new TextArea();
        ingredientsField.setPromptText("Ingredients");
        ingredientsField.setWrapText(true);

        TextArea instructionsField = new TextArea();
        instructionsField.setPromptText("Instructions");
        instructionsField.setWrapText(true);

        form.getChildren().addAll(new Label("Name:"), nameField, new
Label("Ingredients:"), ingredientsField, new Label("Instructions:"),
instructionsField);

        dialog.getDialogPane().setContent(form);

        // Convert the result to a recipe when the Add button is clicked
        dialog.setResultConverter(dialogButton -> {
            if (dialogButton == addButton) {
                String name = nameField.getText().trim();
                String ingredientsText = ingredientsField.getText().trim();
                String instructions = instructionsField.getText().trim();

                // Split ingredients into a list
                List<String> ingredients = new
ArrayList<>(Arrays.asList(ingredientsText.split(",")));

                return new Recipe(name, ingredients, instructions);
            }
            return null;
        });

        Optional<Recipe> result = dialog.showAndWait();
        result.ifPresent(newRecipe -> recipes.add(newRecipe));
    }

    private void editRecipe() {
        // Get the selected recipe
        Recipe selectedRecipe =
recipeListView.getSelectionModel().getSelectedItem();
        if (selectedRecipe != null) {
            // Create a dialog to edit the recipe
            Dialog<Recipe> dialog = new Dialog<>();
            dialog.setTitle("Edit Recipe");
            dialog.setHeaderText("Edit Recipe Details:");

            // Set the button types (OK and Cancel)
            ButtonType editButton = new ButtonType("Edit",
ButtonBar.ButtonData.OK_DONE);
            dialog.getDialogPane().getButtonTypes().addAll(editButton,
ButtonType.CANCEL);

            // Create and configure the recipe details form
            VBox form = new VBox();
            form.setSpacing(10);
            form.setPadding(new Insets(10));

            TextField nameField = new TextField(selectedRecipe.getName());
            nameField.setPromptText("Name");
```

```
            TextArea ingredientsField = new TextArea(String.join(", ",
selectedRecipe.getIngredients()));
            ingredientsField.setPromptText("Ingredients");
            ingredientsField.setWrapText(true);

            TextArea instructionsField = new
TextArea(selectedRecipe.getInstructions());
            instructionsField.setPromptText("Instructions");
            instructionsField.setWrapText(true);

            form.getChildren().addAll(new Label("Name:"), nameField, new
Label("Ingredients:"), ingredientsField, new Label("Instructions:"),
instructionsField);

            dialog.getDialogPane().setContent(form);

            // Convert the result to a recipe when the Edit button is
clicked
            dialog.setResultConverter(dialogButton -> {
                if (dialogButton == editButton) {
                    String name = nameField.getText().trim();
                    String ingredientsText =
ingredientsField.getText().trim();
                    String instructions =
instructionsField.getText().trim();

                    // Split ingredients into a list
                    List<String> ingredients = new
ArrayList<>(Arrays.asList(ingredientsText.split(",")));

                    selectedRecipe.setName(name);
                    selectedRecipe.setIngredients(ingredients);
                    selectedRecipe.setInstructions(instructions);

                    recipeListView.refresh(); // Update the list view
                }
                return null;
            });

            dialog.showAndWait();
        }
    }

    private void deleteRecipe() {
        // Get the selected recipe
        Recipe selectedRecipe =
recipeListView.getSelectionModel().getSelectedItem();
        if (selectedRecipe != null) {
            // Ask the user for confirmation
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Confirm Deletion");
            alert.setHeaderText("Are you sure you want to delete this
recipe?");
            alert.setContentText(selectedRecipe.getName());

            Optional<ButtonType> result = alert.showAndWait();
            if (result.isPresent() && result.get() == ButtonType.OK) {
                // Remove the recipe from the recipes list
                recipes.remove(selectedRecipe);

                // Remove ingredients of the deleted recipe from the
```

```
shopping list
                List<String> ingredientsToRemove =
selectedRecipe.getIngredients();
                shoppingListView.getItems().removeAll(ingredientsToRemove);
            }
        }
    }


    private void addToShoppingList() {
        // Get the selected recipe
        Recipe selectedRecipe =
recipeListView.getSelectionModel().getSelectedItem();
        if (selectedRecipe != null) {
            // Get the ingredients of the selected recipe
            List<String> ingredients = selectedRecipe.getIngredients();

            // Create a dialog to select specific ingredients to add to the
shopping list
            ChoiceDialog<String> dialog = new ChoiceDialog<>(null,
ingredients);
            dialog.setTitle("Add to Shopping List");
            dialog.setHeaderText("Select Ingredients to Add to Shopping
List");

            // Create a checkbox to allow selecting all ingredients
            CheckBox selectAllCheckBox = new CheckBox("Select All");

            // Show the dialog and handle the user's selection
            dialog.getDialogPane().setContent(new VBox(selectAllCheckBox,
dialog.getDialogPane().getContent()));
            dialog.getDialogPane().setContentText("Ingredients:");

            selectAllCheckBox.setOnAction(event -> {
                if (selectAllCheckBox.isSelected()) {
                    dialog.getItems().addAll(ingredients);
                } else {
                    dialog.getItems().clear();
                }
            });

            dialog.showAndWait().ifPresent(selectedIngredient -> {
                // Check if the selected ingredient is not null
                if (selectedIngredient != null) {
                    shoppingListView.getItems().add(selectedIngredient);
                }
            });
        }
    }

    private void searchRecipes(String query) {
        // Implement your search logic here and update the recipeListView
accordingly
        ObservableList<Recipe> searchResults =
FXCollections.observableArrayList();

        for (Recipe recipe : recipes) {
            String recipeName = recipe.getName().toLowerCase();
            String ingredientsText = String.join(", ",
recipe.getIngredients()).toLowerCase();
            if (query.isEmpty() || recipeName.contains(query.toLowerCase())
```

```
|| ingredientsText.contains(query.toLowerCase())) {
                searchResults.add(recipe);
            }
        }

        recipeListView.setItems(searchResults);
    }

    // Define the Recipe class here
    public static class Recipe {
        private String name;
        private List<String> ingredients;
        private String instructions;

        public Recipe(String name, List<String> ingredients, String
instructions) {
            this.name = name;
            this.ingredients = ingredients;
            this.instructions = instructions;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public List<String> getIngredients() {
            return ingredients;
        }

        public void setIngredients(List<String> ingredients) {
            this.ingredients = ingredients;
        }

        public String getInstructions() {
            return instructions;
        }

        public void setInstructions(String instructions) {
            this.instructions = instructions;
        }

        @Override
        public String toString() {
            return name;
        }
    }

    // Define the RecipeCategory class here
    public class RecipeCategory {
        private String name;

        public RecipeCategory(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
```

```
            }
        }
}
```

CONCLUSION

The Recipe Manager project provides an efficient solution for managing recipes, creating shopping lists, and finding recipes by ingredients. Its intuitive user interface and feature-rich functionality make it a valuable tool for anyone who enjoys cooking and wants to streamline their recipe organization process. This project can serve as a foundation for further enhancements, such as user accounts, meal planning, and more advanced categorization features.

GITHUB REPOSITORY

https://github.com/nandhujayan/Recipe-Manager.git

PROMPT USED

1. create a javafx fully functional project Recipe Manager: Develop a recipe management system where users can add, edit and search for recipes by ingredients.

2. implement additional features like organize recipes, create shopping lists into this project

3. Bring this search bar on the top

4. integrate a delete option into this

5. change the color of a button in JavaFX without creating a new CSS file

6. modify this code by adding a checkbox to the add to shopping list where incgrediants are choosed by ticking a checkbox. provide me the full code for this.