**PROJECT REPORT ON**

“Recipe Organizer”

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Title** | **Page No.** |
| 01 | INTRODUCTION | 03 |
| 02 | OBJECTIVE | 04 |
| 03 | BACKGROUND | 05-11 |
| 04 | HARDWARE AND SOFTWARE REQUIREMENTS | 12 |
| 05 | CODING | 13-24 |
| 06 | OUTPUT SCREENSHOTS | 25-36 |
| 07 | FUTURE SCOPE | 37 |
| 08 | CONCLUSION | 38 |
| 09 | REFERENCES AND BIBLIOGRAPHY | 39 |

**INTRODUCTION**

Managing and organizing recipes efficiently becomes a significant task. Recipe Organizer is a GUI interface which is made by using Python library tkinter and the programming language PYTHON is used for the coding purpose. It has basically a GUI interface where it includes “Add Recipe”, “View Recipe”, “Edit Recipe”, “Delete Recipe”, “Sort Recipe”, “Search Recipe” button. There is a listbox just above all these buttons in the “Recipe GUI” window where all the Recipes will be added. And if any user want to Edit a particular recipe then user have to first select it then by clicking on Edit button user can update the existing recipe. To delete a particular recipe user have to select it first then click on delete button. To arrange the recipe in alphabetical manner and also according to category wise user have to click on Sort Recipe button. Search Recipe button will search the random recipe from Edamam api and then in a new window which is having title means Recipe name, ingredients and a Recipe Link will be also there where after clicking on Recipe link button user will navigate to a browser tab where all the details of recipe will be written.

**OBJECTIVE**

The primary objectives of the Recipe Organizer application are as follows:

* **User-Friendly Interface**: Develop an intuitive graphical user interface (GUI) that allows users to interact with their recipe collection seamlessly.
* **Recipe Management**: Enable users to add new recipes, view details of existing recipes, edit recipes, and delete recipes as needed.
* **Search Functionality**: Implement a search feature that allows users to find recipes based on keywords, enhancing accessibility to specific recipes.
* **Sorting Mechanism**: Provide a sorting mechanism to arrange recipes alphabetically, facilitating efficient organization.
* **Integration with Edamam API**: Utilize the Edamam API to fetch additional details and enrich the application's recipe database.
* **PIL Library**: Understanding the concept of Pillow Image Library in python
* **Webbrowser module**: Understanding how webbrowser module work with any url
* **Requests module**:

**BACKGROUND**

The Recipe Organizer application is built using the Tkinter library, a widely used GUI toolkit for Python. Tkinter provides a robust foundation for developing graphical applications with a user-friendly interface. The application employs various components such as labels, entry fields, buttons, and listboxes to create an organized and visually appealing layout.

Furthermore, the integration with the Edamam API adds a dynamic element to the application, allowing users to search for recipes beyond their personal collection. This API fetches additional details and images, enhancing the overall user experience.

MODULES USED:

* **tkinter**: The primary GUI library in Python. It provides functions and classes to create windows, buttons, labels, etc., for building graphical applications.
* **simpledialog, messagebox**: These modules from Tkinter provide simple dialog boxes for user input and message boxes for displaying various types of messages.
* **io, BytesIO**: The io module is used for handling streams, and BytesIO is used here to read/write bytes in memory. This is often used for image handling.
* **PIL (Python Imaging Library) - ImageTk, Image**: These modules from the Pillow library (a fork of PIL) are used for working with images. ImageTk is used to handle images in a Tkinter-compatible way.
* **requests**: A popular Python library for making HTTP requests. In this script, it is likely used for fetching images from a URL.
* **webbrowser**: A module that provides a high-level interface to allow displaying Web-based documents to users. It's used here to open URLs in the default web browser.

CLASS

* **LinkLabel class**: This is a custom class that inherits from the Tkinter Label class. It is designed to act as a hyperlink. It has methods to set the URL and open the link when clicked.
* **RecipeInfo class**: This class is a simple data container for storing details of a recipe. It has a constructor that initializes the attributes such as name, ingredients, instructions, and category.
* **RecipeOrganizer class**: This is the main class responsible for managing the recipe organizer GUI and functionality. It has a constructor that sets up the main window (root) and its properties.

CONSTRUCTOR USED:

* **Constructor in the LinkLabel Class:**

Overview:

* + This constructor initializes an instance of the LinkLabel class.
  + It calls the constructor of the superclass (Label) using super().\_\_init\_\_(master, \*\*kwargs).
  + Configures the label with properties such as cursor, text, background color, foreground color, font, and underline.
* **Constructor in the RecipeOrganizer Class:**

**Overview:**

* This constructor initializes an instance of the RecipeOrganizer class.
* It takes a root parameter, which is the main window of the application.
* Sets up properties for the main window, including dimensions (800x700), title ("Recipe GUI"), and background color ("black").
* Constructor in the **RecipeInfo** class:

Overview:

* \_\_init\_\_ Method:
* This is the special method in Python classes that is automatically called when an object of the class is created.
* It initializes the state of the object by setting the values of its attributes.

Methods in RecipeOrganizer Class:

* **add\_recipe:** Opens a new window for adding a new recipe with entry fields for name, ingredients, instructions, and category.
* **save\_recipe:** Saves the entered recipe details and updates the recipe list.
* **view\_recipe:** Displays details of the selected recipe in a pop-up window.
* **edit\_recipe:** Opens a new window for editing the selected recipe.
* **save\_changes:** Saves the changes made to the edited recipe.
* **delete\_recipe:** Deletes the selected recipe after user confirmation.
* **sort\_recipes:** Sorts the recipes alphabetically and displays them in a new window categorized by recipe category.
* **search\_recipe:** Opens a new window for searching recipes.
* **perform\_search:** Performs a search operation using the Edamam API.
* **display\_recipe\_details\_edamam:** Displays recipe details obtained from the **Edamam API**.
* **update\_listbox:** Updates the listbox with the current list of recipes.
* **download\_image:** Function to set the background image for the main window of Recipe organizer.

GUI WINDOW:

* **Main Window (Recipe Organizer):**
  + This is the primary window that serves as the Recipe Organizer application.
  + It includes a background image, labels for welcome and styling, a listbox for displaying recipes, and buttons for adding, viewing, editing, deleting, sorting, and searching recipes.
* **Add Recipe Window:**
  + This window is created when the user clicks the "Add Recipe" button on the main window.
  + It allows the user to input details for a new recipe, including name, ingredients, instructions, and category.
  + It includes labels, entry fields, and a button for saving the recipe.
* **Edit Recipe Window:**
  + This window is created when the user clicks the "Edit Recipe" button on the main window after selecting a recipe.
  + It allows the user to edit the details of the selected recipe.
  + Similar to the "Add Recipe" window, it includes labels, entry fields, and a button for saving the changes.
* **Sorted Recipes Window:**
  + This window is created when the user clicks the "Sort Recipes" button on the main window.
  + It displays the recipes sorted by category. Each category has its own frame containing a label for the category and a listbox displaying recipes in that category.
  + The window includes a button to close the window.
* **Search Recipe Window:**
  + This window is created when the user clicks the "Search Recipe" button on the main window.
  + It allows the user to enter a recipe name for searching using an external API (Edamam API).
  + The window includes a label, an entry field for the search query, and a button to perform the search.
* **Display Recipe Details Window:**
  + This window is created when a recipe is selected after searching using the Edamam API.
  + It displays details of the selected recipe, including an image (if available), title, ingredients, and a link to the source.
  + The window includes labels, a text widget for displaying ingredients, and a clickable link for the source URL.

BUTTONS USED:

* **Add Recipe Button:**
  + Purpose: Initiates the process of adding a new recipe.
  + Location: Located on the main window.
  + Usage: When clicked, it opens a new window (Add\_Recipe\_window) where the user can input details for a new recipe.
* **View Recipe Button:**
  + Purpose: Displays detailed information about the selected recipe.
  + Location: Located on the main window.
  + Usage: When a recipe is selected in the listbox, clicking this button opens a messagebox with details such as recipe name, ingredients, instructions, and category.
* **Edit Recipe Button:**
  + Purpose: Allows the user to edit the details of the selected recipe.
  + Location: Located on the main window.
  + Usage: When a recipe is selected in the listbox, clicking this button opens a new window (edit\_recipe\_window) with pre-filled details for editing.
* **Delete Recipe Button:**
  + Purpose: Deletes the selected recipe after user confirmation.
  + Location: Located on the main window.
  + Usage: When a recipe is selected in the listbox, clicking this button prompts the user to confirm deletion. If confirmed, the recipe is removed from the list.
* **Sort Recipes Button:**
  + Purpose: Sorts the recipes alphabetically by name.
  + Location: Located on the main window.
  + Usage: When clicked, it sorts the recipes and opens a new window (sort\_window) displaying the sorted recipes categorized by recipe type.
* **Search Recipe Button:**
  + Purpose: Initiates a search for recipes using an external API (Edamam API).
  + Location: Located on the main window.
  + Usage: When clicked, it opens a new window (search\_window) with an entry field for the user to enter a recipe name to search for.
* **Save Recipe Button (in Add Recipe Window):**
  + Purpose: Saves the newly added recipe.
  + Location: Located on the "Add Recipe" window (Add\_Recipe\_window).
  + Usage: When clicked, it validates the input and saves the recipe to the list.
* **Save Changes Button (in Edit Recipe Window):**
  + Purpose: Saves the edited details of the selected recipe.
  + Location: Located on the "Edit Recipe" window (edit\_recipe\_window).
  + Usage: When clicked, it saves the changes made to the selected recipe.
* **Close Button (in Sort Recipes Window):**
  + Purpose: Closes the window displaying sorted recipes.
  + Location: Located on the "Sort Recipes" window (sort\_window).
  + Usage: When clicked, it closes the window.
* **Search Button (in Search Recipe Window):**
  + Purpose: Initiates the search for recipes using the entered query.
  + Location: Located on the "Search Recipe" window (search\_window).
  + Usage: When clicked, it triggers the search operation based on the entered query.

Edamam API:

* The Edamam API is a food and recipe API that allows developers to access a large database of recipes, nutrition information, and more.
* It provides endpoints for recipe search, nutrition analysis, and food database lookup.
* In this code, the recipe search endpoint is used to find recipes based on a user-entered query.
* **API Key:** To use the Edamam API, developers need to obtain an API key.
* **Code Integration:**
* The code includes a function named perform\_search within the RecipeOrganizer class. This function constructs a URL with the search query and the Edamam application ID and API key.
* It then sends an HTTP GET request to the Edamam API using the requests library.
* The response is processed to extract recipe details.
* **Handling Errors:** The code includes error handling to manage cases where the API request fails. If there's an issue with the request or if no matching recipes are found, appropriate error messages are displayed using the messagebox module.
* **Displaying Recipe Details:**
* When a matching recipe is found, the details are displayed in a new window. The details include the recipe name, ingredients (with bullet points), and a link to the source URL.
* The link is clickable and opens the recipe's source URL in the default web browser.

**HARDWARE AND SOFTWARE REQUIREMENTS**

**HARDWARE REQUIREMENTS:**

|  |  |
| --- | --- |
| **Hardware Tools** | **Minimum Requirements** |
| OS Name | Microsoft Windows 10 |
| System Type | x64 based PC |
| Processor | Intel(R) Core i5 |
| RAM | 8GB |
| Hardware | 10GB |

**SOFTWARE REQUIREMENTS:**

|  |  |
| --- | --- |
| **SOFTWARE TOOLS** | **MINIMUM REQUIREMENTS** |
| Platform | Windows, MacOs or Linux |
| Operating System | Windows, MacOs or Linux |
| Technology | Python |
| Technology version | Python 3.12.0 |
| IDE | VsCode |

**CODING**

recipe\_organizer.py

from tkinter import \*

from tkinter import simpledialog, messagebox

from io import BytesIO

from PIL import ImageTk, Image

import requests

import webbrowser

# class LinkLabel is a label acting as a hyperlink.

class LinkLabel(Label):

    def \_\_init\_\_(self, master=None, \*\*kwargs):

        super().\_\_init\_\_(master, \*\*kwargs)

        self.config(cursor="hand2", text="Recipe Link", bg="white",fg="blue", font=("Times New Roman", 15,"bold"),underline=-1)

    #  Method sets the URL for the hyperlink.

    def set\_link(self, url):

        self.url = url

    # Method opens the URL in the default web browser when the label is clicked.

    def open\_link(self, event):

        webbrowser.open\_new(self.url)

# class RecipeInfo containing all the details of Recipe

class RecipeInfo:

    # Constructor initializes the recipe details such as name, ingredients, instructions, and category.

    def \_\_init\_\_(self, name, ingredients, instructions, category):

        self.name = name

        self.ingredients = ingredients

        self.instructions = instructions

        self.category = category

# class RecipeOrganizer responsible for managing the recipe organizer GUI and functionality.

class RecipeOrganizer:

    # Constructor initializes the main window (root) and sets its properties.

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.geometry("800x700")

        self.root.title("Recipe GUI")

        self.root.configure(bg="black")

        # List to store recipes

        self.recipes = []

        # setting background image for Main window

        # Replace "your\_image\_url" with the URL of the image you want to use

        image\_url = "https://images.unsplash.com/photo-1490645935967-10de6ba17061?w=600&auto=format&fit=crop&q=60&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxzZWFyY2h8MTF8fHJlY2lwZXxlbnwwfHwwfHx8MA%3D%3D"

        self.bg\_image = self.download\_image(image\_url)

        # Set background image for the main window

        background\_label = Label(self.root, image=self.bg\_image,bg="black")

        background\_label.place(relwidth=1, relheight=0.5)

        # GUI components

        Upper\_Label = Label(self.root, text="Welcome to the Recipe Organizer", bg="grey", font=("Times New Roman", 18))

        Upper\_Label.pack(side=TOP, fill=X)

        Left\_Label = Label(self.root, bg="grey", width=3)

        Left\_Label.pack(side=LEFT, fill=Y)

        Right\_Label = Label(self.root, bg="grey", width=3)

        Right\_Label.pack(side=RIGHT, fill=Y)

        Bottom\_Label = Label(self.root, bg="grey", width=5)

        Bottom\_Label.pack(side=BOTTOM, fill=X)

        # Frame for Listbox

        listbox\_frame = Frame(self.root, bg="black")

        listbox\_frame.place(relx=0.5, rely=0.7, anchor=CENTER)

        # Listbox where the Recipe will be added

        self.recipe\_listbox = Listbox(listbox\_frame, selectmode=SINGLE)

        self.recipe\_listbox.configure(width=40)

        self.recipe\_listbox.pack(pady=10)

        # Frame for Buttons

        button\_frame = Frame(self.root, bg="black")

        button\_frame.place(relx=0.5, rely=0.9, anchor=CENTER)

        # Add button to add New recipe

        add\_button = Button(button\_frame, text="Add Recipe", command=self.add\_recipe, bg="lightblue", font=("Times New Roman", 13, "bold"), fg="black")

        add\_button.pack(side=LEFT, padx=5)

        # View button to see the information of selected Recipe

        view\_button = Button(button\_frame, text="View Recipe", command=self.view\_recipe, bg="lightblue", font=("Times New Roman", 13, "bold"), fg="black")

        view\_button.pack(side=LEFT, padx=5)

        # Edit button to edit the previous recipe that added inside the Listbox

        edit\_button = Button(button\_frame, text="Edit Recipe", command=self.edit\_recipe, bg="lightblue", font=("Times New Roman", 13, "bold"), fg="black")

        edit\_button.pack(side=LEFT, padx=5)

        # Delete button to delete the selected recipe of the Listbox

        delete\_button = Button(button\_frame, text="Delete Recipe", command=self.delete\_recipe, bg="lightblue", font=("Times New Roman", 13, "bold"), fg="black")

        delete\_button.pack(side=LEFT, padx=5)

        # Button for sorting recipes of recipe listbox in alphabetical manner

        sort\_button = Button(button\_frame, text="Sort Recipes", command=self.sort\_recipes, bg="lightblue", font=("Times New Roman", 13, "bold"), fg="black")

        sort\_button.pack(side=LEFT, padx=5)

        # Add the Search button

        search\_button = Button(button\_frame, text="Search Recipe", command=self.search\_recipe, bg="lightblue", font=("Times New Roman", 13, "bold"), fg="black")

        search\_button.pack(side=LEFT, padx=5)

    # Function to add a new recipe to the list

    def add\_recipe(self):

        # Create a new Toplevel window for the custom dialog

        Add\_Recipe\_window = Toplevel()

        Add\_Recipe\_window.title("Add Recipe")

        Add\_Recipe\_window.geometry("800x600")

        Add\_Recipe\_window.configure(bg="#FFC5C5")

        # Variables to store the input values

        name\_var = StringVar()

        category\_var = StringVar()

        # Labels and Entry fields for the recipe details

        Label(Add\_Recipe\_window, text="Recipe Name:", font=("Times New Roman", 13,"bold"), bg="#96EFFF",width=30).grid(row=0, column=0, padx=10, pady=10)

        Entry(Add\_Recipe\_window, textvariable=name\_var, font=("Times New Roman", 11), width=50).grid(row=0, column=1, padx=10, pady=10)

        Label(Add\_Recipe\_window, text="Recipe Ingredients (Numbered Order):", font=("Times New Roman", 13,"bold"), bg="#96EFFF",width=30).grid(row=1, column=0, padx=10, pady=10)

        ingredients\_entry = Text(Add\_Recipe\_window, height=7, font=("Times New Roman", 11), width=50)

        ingredients\_entry.grid(row=1, column=1, padx=10, pady=10)

        Label(Add\_Recipe\_window, text="Recipe Instructions (Step Manner):", font=("Times New Roman", 13,"bold"), bg="#96EFFF",width=30).grid(row=2, column=0, padx=10, pady=10)

        instructions\_entry = Text(Add\_Recipe\_window, height=7, font=("Times New Roman", 11), width=50)

        instructions\_entry.grid(row=2, column=1, padx=10, pady=10)

        Label(Add\_Recipe\_window, text="Recipe Category:", font=("Times New Roman", 13,"bold"), bg="#96EFFF",width=30).grid(row=3, column=0, padx=10, pady=10)

        Entry(Add\_Recipe\_window, textvariable=category\_var, font=("Times New Roman", 11), width=50).grid(row=3, column=1, padx=10, pady=10)

        # Button to save the recipe that user wants to add inside the Listbox

        save\_button = Button(

            Add\_Recipe\_window,

            text="Save Recipe",

            command=lambda: self.save\_recipe(

                Add\_Recipe\_window, name\_var, ingredients\_entry, instructions\_entry, category\_var

            ),

            fg="white",

            bg="black",

            font=("Times New Roman", 13)

        )

        save\_button.grid(row=4, column=1, pady=10)

    # Function to save the recipe and update the list

    def save\_recipe(self, Add\_Recipe\_window, name\_var, ingredients\_entry, instructions\_entry, category\_var):

        name = name\_var.get()

        ingredients = ingredients\_entry.get("1.0", END).strip()

        instructions = instructions\_entry.get("1.0", END).strip()

        category = category\_var.get()

        if name:

            recipe = RecipeInfo(name, ingredients, instructions, category)

            self.recipes.append(recipe)

            self.update\_listbox()

            messagebox.showinfo("Recipe Added", f"Recipe '{name}' added successfully!")

            Add\_Recipe\_window.destroy()  # Close the Add\_Recipe\_window

    # Function to view the selected recipe

    def view\_recipe(self):

        selected\_recipe\_index = self.recipe\_listbox.curselection()

        if selected\_recipe\_index:

            selected\_recipe = self.recipes[selected\_recipe\_index[0]]

            messagebox.showinfo("Recipe", f"Recipe Name: {selected\_recipe.name}\n\n"

                                          f"Ingredients: \n{selected\_recipe.ingredients}\n\n"

                                          f"Instructions: \n{selected\_recipe.instructions}\n\n"

                                          f"Category: {selected\_recipe.category}")

    # # Function to edit the selected recipe

    def edit\_recipe(self):

        selected\_recipe\_index = self.recipe\_listbox.curselection()

        if selected\_recipe\_index:

            selected\_recipe = self.recipes[selected\_recipe\_index[0]]

            # Create a new Toplevel window for the custom dialog for editing the recipe

            edit\_recipe\_window = Toplevel()

            edit\_recipe\_window.title("Edit Recipe")

            edit\_recipe\_window.geometry("800x600")

            edit\_recipe\_window.configure(bg="#FFC5C5")

            # Variables to store the input values

            name\_var = StringVar(value=selected\_recipe.name)

            category\_var = StringVar(value=selected\_recipe.category)

            # Labels and Entry fields for the recipe details which user will edit

            Label(edit\_recipe\_window, text="Recipe Name:", font=("Times New Roman", 13,"bold"), bg="#96EFFF",width=30).grid(row=0, column=0, padx=10, pady=10)

            Entry(edit\_recipe\_window, textvariable=name\_var, font=("Times New Roman", 11), width=50).grid(row=0, column=1, padx=10, pady=10)

            Label(edit\_recipe\_window, text="Recipe Ingredients (Numbered Order):", font=("Times New Roman", 13,"bold"), bg="#96EFFF",width=30).grid(row=1, column=0, padx=10, pady=10)

            ingredients\_entry = Text(edit\_recipe\_window, height=7, font=("Times New Roman", 11), width=50)

            ingredients\_entry.insert(END, selected\_recipe.ingredients)  # Pre-fill with existing ingredients

            ingredients\_entry.grid(row=1, column=1, padx=10, pady=10)

            Label(edit\_recipe\_window, text="Recipe Instructions (Step Manner):", font=("Times New Roman", 13,"bold"), bg="#96EFFF",width=30).grid(row=2, column=0, padx=10, pady=10)

            instructions\_entry = Text(edit\_recipe\_window, height=7, font=("Times New Roman", 11), width=50)

            instructions\_entry.insert(END, selected\_recipe.instructions)  # Pre-fill with existing instructions

            instructions\_entry.grid(row=2, column=1, padx=10, pady=10)

            Label(edit\_recipe\_window, text="Recipe Category:", font=("Times New Roman", 13,"bold"), bg="#96EFFF",width=30).grid(row=3, column=0, padx=10, pady=10)

            Entry(edit\_recipe\_window, textvariable=category\_var, font=("Times New Roman", 11), width=50).grid(row=3, column=1, padx=10, pady=10)

            # Button to save the edited recipe

            save\_button = Button(

                edit\_recipe\_window,

                text="Save Changes",

                command=lambda: self.save\_changes(

                    selected\_recipe, edit\_recipe\_window, name\_var, ingredients\_entry, instructions\_entry, category\_var

                ),

                fg="white",

                bg="black",

                font=("Times New Roman", 13)

            )

            save\_button.grid(row=4, column=1, pady=10)

    def save\_changes(self, recipe, edit\_recipe\_window, name\_var, ingredients\_entry, instructions\_entry, category\_var):

        recipe.name = name\_var.get()

        recipe.ingredients = ingredients\_entry.get("1.0", END).strip()  # Get all content from line 1 to the end

        recipe.instructions = instructions\_entry.get("1.0", END).strip()  # Get all content from line 1 to the end

        recipe.category = category\_var.get()

        self.update\_listbox()

        messagebox.showinfo("Recipe Updated", f"Recipe '{recipe.name}' updated successfully!")

        edit\_recipe\_window.destroy()  # Close the edit\_recipe\_window window

    # Function to delete the selected recipe by user confirmation

    def delete\_recipe(self):

        selected\_recipe\_index = self.recipe\_listbox.curselection()

        # if else condition to ask from the user that whether user want to delete the selected Recipe or not by clicking Yes or No

        if selected\_recipe\_index:

            result = messagebox.askyesno("Delete Recipe", "Are you sure you want to delete the selected recipe from the recipe\_listbox?")

            if result:

                self.recipes.pop(selected\_recipe\_index[0])

                self.update\_listbox()

    # Function to sort recipes in alphabetical manner

    def sort\_recipes(self):

        self.recipes.sort(key=lambda x: x.name.lower())

        self.update\_listbox()

        # Create a new window for displaying sorted recipes by category

        sort\_window = Toplevel()

        sort\_window.title("Sorted Recipes by Category")

        sort\_window.geometry("400x800")

        sort\_window.configure(bg="#BB9CC0")

        # Create a dictionary to store frames for each category

        category\_frames = {}

        # Populate frames with recipes

        for recipe in self.recipes:

            category = recipe.category

            if category not in category\_frames:

            # Create a new frame for the category

                category\_frame = Frame(sort\_window, padx=10, pady=10)

                category\_frame.configure(bg="#BB9CC0")

                category\_frame.pack(side=TOP, fill=BOTH, expand=True)

                # Label for the category

                category\_label = Label(category\_frame, text=f"{category} Recipes", font=("Times New Roman", 14, "bold"))

                category\_label.configure(bg="#BB9CC0")

                category\_label.pack(side=TOP)

                # Listbox for displaying recipes in the category

                category\_listbox = Listbox(category\_frame, selectmode=SINGLE, font=("Times New Roman", 14))

                category\_listbox.pack(side=TOP, fill=BOTH, expand=True)

                # Save the frame in the dictionary

                category\_frames[category] = category\_frame

            else:

                # Use existing frame for the category

                category\_frame = category\_frames[category]

                category\_listbox = category\_frame.winfo\_children()[1]  # Assuming label is the first and listbox is the second

            # Insert the recipe into the corresponding listbox

            category\_listbox.insert(END, recipe.name)

        # Button to close the sort window

        close\_button = Button(sort\_window, text="Close", command=sort\_window.destroy, bg="lightblue", font=("Times New Roman", 13, "bold"), fg="black")

        close\_button.pack(side=BOTTOM, pady=10)

    # Function to search recipes

    def search\_recipe(self):

        # Create a new Toplevel window for the search dialog

        search\_window = Toplevel()

        search\_window.title("Search Recipe")

        search\_window.geometry("610x100")

        search\_window.configure(bg="#FFF5C2")

        # Label and Entry for the search query

        Label(search\_window, text="Enter Recipe Name:", font=("Times New Roman", 14,"bold"),bg="#7071E8").grid(row=0, column=0, padx=10, pady=10)

        search\_entry = Entry(search\_window, font=("Times New Roman", 14), width=30)

        search\_entry.grid(row=0, column=1, padx=10, pady=10)

        # Button to perform the search

        search\_button = Button(search\_window, text="Search", command=lambda: self.perform\_search(search\_entry.get()), font=("Times New Roman", 15,"bold"), fg="white", bg="black")

        search\_button.grid(row=0, column=2, pady=10, padx=10)

    # function to perform search operation

    def perform\_search(self, query):

        # Edamam application ID and apikey

        edamam\_app\_id = 'ad79754f'

        api\_key = '9a6f2888ad1b2265659bd91a0195297c'

        url = f"https://api.edamam.com/search?q={query}&app\_id={edamam\_app\_id}&app\_key={api\_key}"

        try:

            response = requests.get(url)

            data = response.json()

            if 'hits' in data:

                # Display the first recipe found in a new window

                if data['hits']:

                    recipe = data['hits'][0]['recipe']

                    self.display\_recipe\_details\_edamam(recipe)

                else:

                    messagebox.showinfo("Search Result", "No matching recipes found.")

            else:

                messagebox.showerror("Error", "Failed to fetch data from the Edamam API.")

        except Exception as e:

            messagebox.showerror("Error", f"An error occurred: {str(e)}")

    # Function to display the recipe details from Edamam API

    def display\_recipe\_details\_edamam(self, recipe):

        edamam\_app\_id = 'ad79754f'

        api\_key = '9a6f2888ad1b2265659bd91a0195297c'

        url = f"https://api.edamam.com/search?q={recipe.get('label', '')}&app\_id={edamam\_app\_id}&app\_key={api\_key}"

        try:

            response = requests.get(url)

            data = response.json()

            if 'hits' in data and data['hits']:

                recipe\_details = data['hits'][0]['recipe']

                # Display recipe details in a new window

                recipe\_details\_window = Toplevel()

                recipe\_details\_window.title(recipe\_details['label'])

                recipe\_details\_window.geometry("500x700")

                recipe\_details\_window.configure(bg="#3081D0")

                # Example: Display recipe image

                if 'image' in recipe\_details:

                    image\_url = recipe\_details['image']

                    response = requests.get(image\_url)

                    img\_data = response.content

                    image = Image.open(BytesIO(img\_data))

                    image = ImageTk.PhotoImage(image)

                    image\_label = Label(recipe\_details\_window, image=image)

                    image\_label.image = image

                    # Set border properties of recipe image

                    image\_label.config(highlightthickness=2,highlightbackground="black")

                    image\_label.pack()

                # Create a frame for the details

                details\_frame = Frame(recipe\_details\_window, bg="grey")

                details\_frame.pack(pady=10)

                # Example: Display other details (modify as needed)

                Label(details\_frame, text=f"Title: {recipe\_details['label']}", bg="grey",font=("Times New Roman", 12,"bold"), justify=LEFT).pack()

                # Display ingredients with bullet points and left alignment

            if 'ingredients' in recipe\_details:

                ingredients\_label = Label(details\_frame, text="Ingredients:", bg="grey", font=("Times New Roman", 12, "bold"), justify=LEFT)

                ingredients\_label.pack()

                # Use a Text widget to display ingredients

                ingredients\_text = Text(details\_frame, wrap=WORD, font=("Times New Roman", 12), height=10, width=40)

                ingredients\_text.pack()

                for index, ingredient in enumerate(recipe\_details['ingredients'], start=1):

                    # Insert each ingredient with a bullet point

                    ingredients\_text.insert(END, f"• {ingredient['text']}\n")

                # Display instructions

                if 'url' in recipe\_details:

                    url\_label = LinkLabel(recipe\_details\_window, text="Source URL", fg="black", cursor="hand2")

                    url\_label.pack()

                    url\_label.bind("<Button-1>", lambda event: webbrowser.open\_new(recipe\_details['url']))

            else:

                messagebox.showerror("Error", "Failed to fetch recipe details from the API.")

        except requests.exceptions.HTTPError as http\_err:

            messagebox.showerror("HTTP Error", f"HTTP error occurred: {http\_err}")

        except Exception as e:

            messagebox.showerror("Error", f"An error occurred: {str(e)}")

    # Function to update the listbox with the current recipe list

    def update\_listbox(self):

        self.recipe\_listbox.delete(0, END)

        for recipe in self.recipes:

            self.recipe\_listbox.insert(END, f"{recipe.name} - {recipe.category}")

    # Function to set the background image for the main window of Recipe organizer

    def download\_image(self, url):

        response = requests.get(url)

        img\_data = response.content

        image = Image.open(BytesIO(img\_data))

        return ImageTk.PhotoImage(image)

# Create the main window

root = Tk()

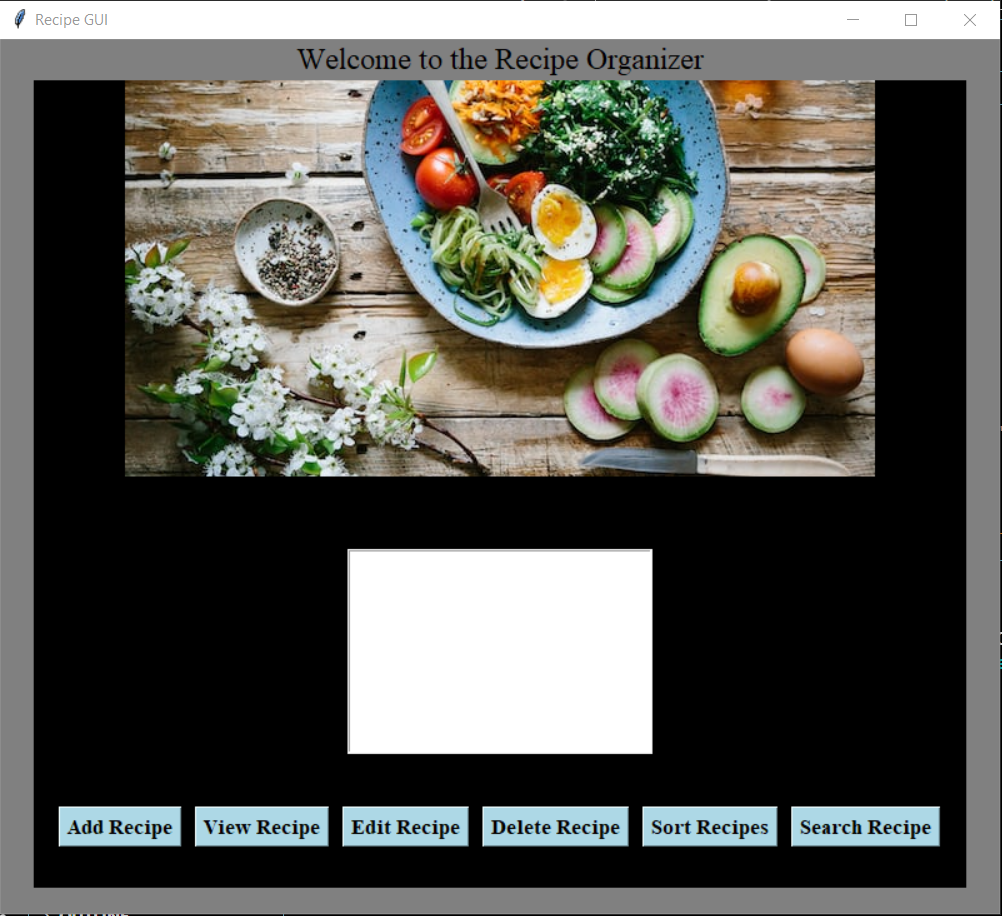
# object of class RecipeOrganizer

recipe = RecipeOrganizer(root)

root.mainloop()

**OUTPUT SCREENSHOT**

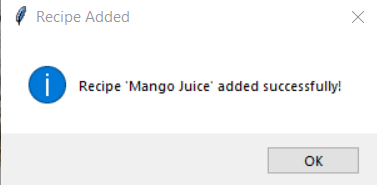
* Main window of Recipe Organizer GUI



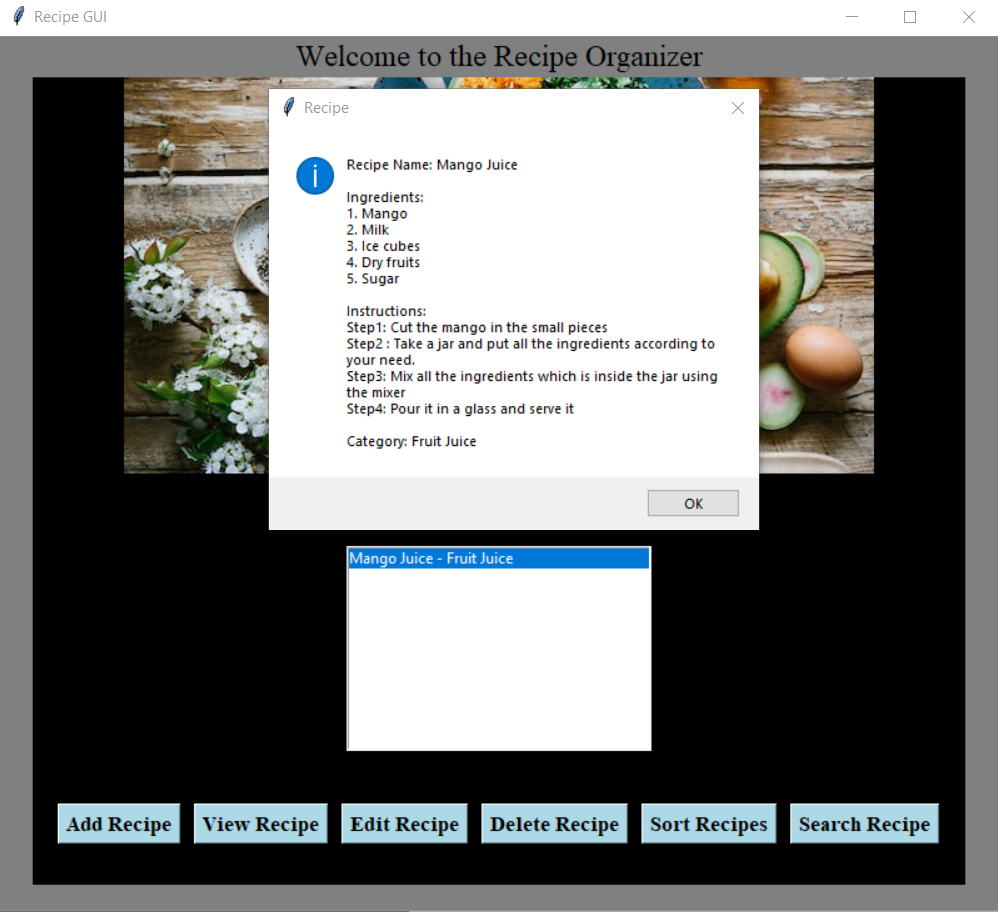
* Add\_recipe\_window whose title is Add Recipe



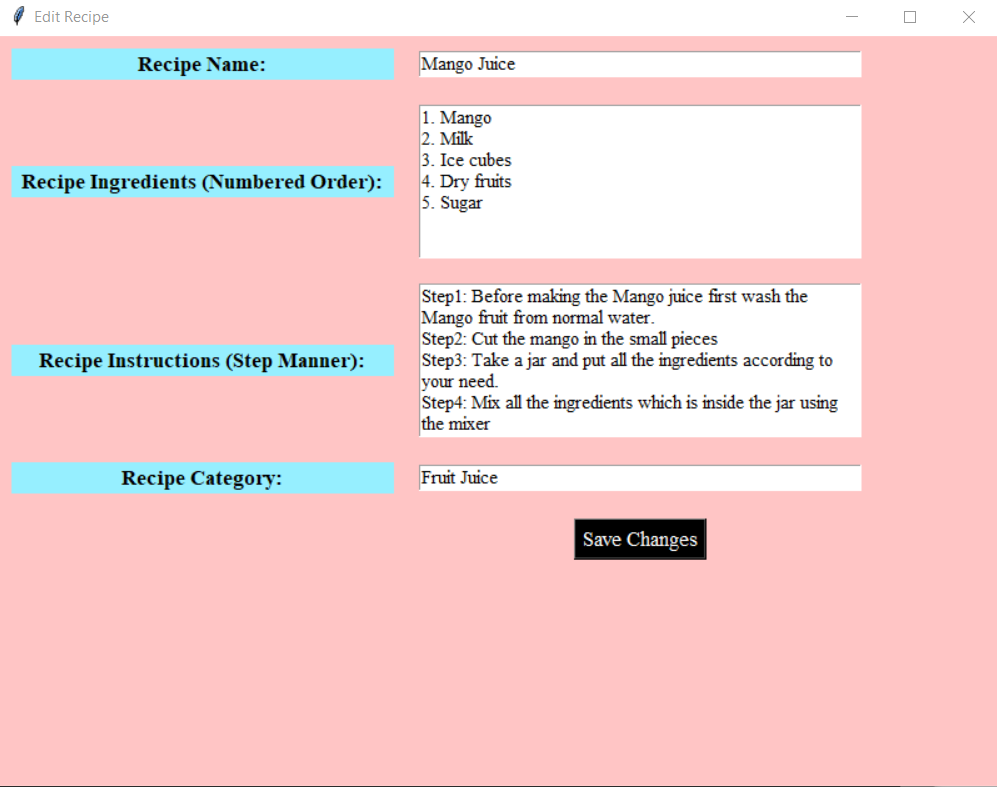
* After clicking on Save Recipe button a messagebox will be open like this an title is Recipe Added



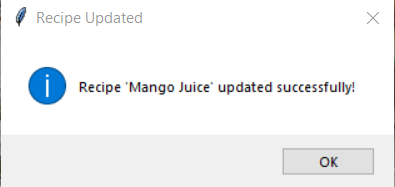
* To view the recipe that added inside the listbox, user have to first select the recipe from the listbox that he/she want to see and click on the View Recipe button then Messagebox will appear whose title is Recipe. And all the information related to recipe such as Recipe Name, ingredients, instructions, category will be shown in the messagebox.

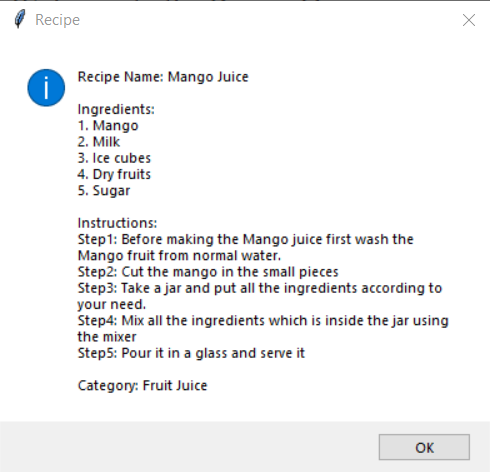


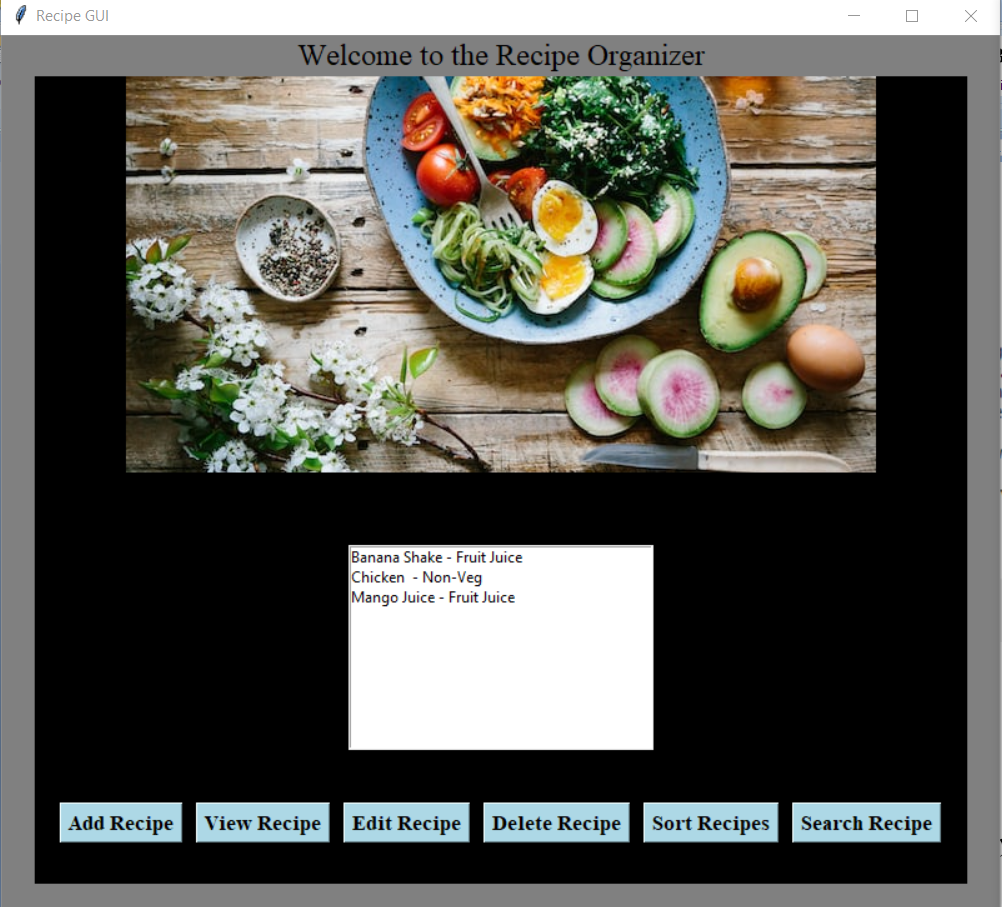
* Edit Recipe window will be open when user want to edit any recipe from the listbox. User have to select the recipe that he or she want to edit. Here, I have change some instruction or steps like step1.

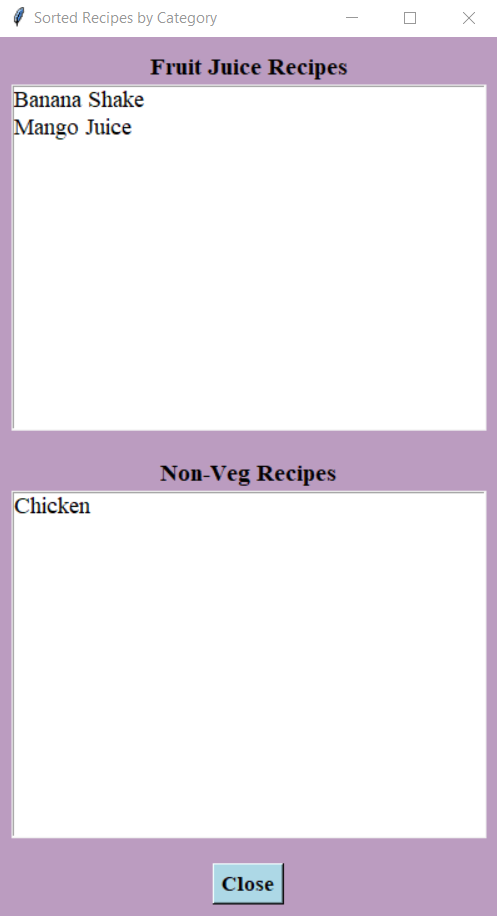


* When user will click on Save Changes button then a mesagebox will appear on the screen having title Recipe Updated and again when we will see the Recipe details by clicking on View Button then

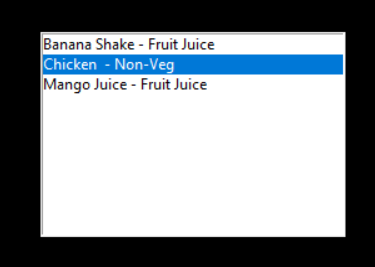




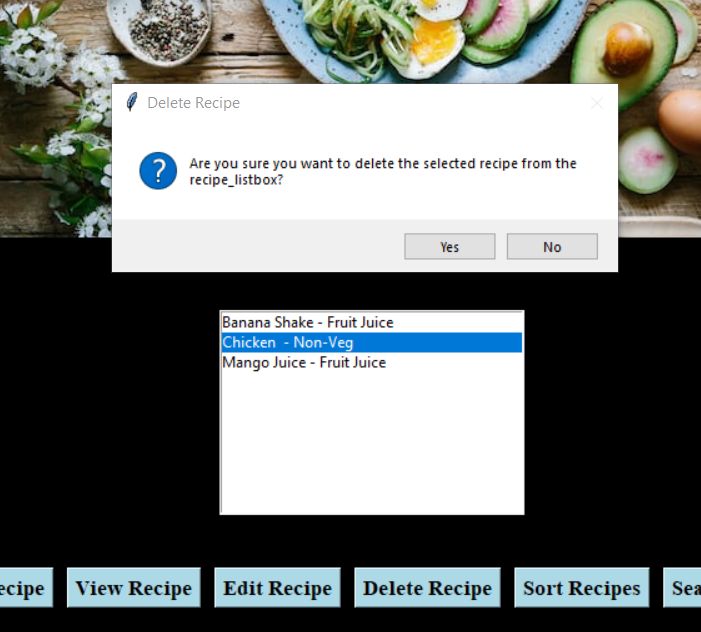
* Sort Recipes Button is used to sort the recipe listed inside the listbox in alphabeticalmanner.  
  
* A new Sorted Recipes by Category window will appear on the screen when Sort Recipes button will be clicked where all the recipes will be sorted according to category wise. For Eg: here, Fruit juice is the category having 2 Recipes inside it one is Mango Juice and another is Banana shake whereas Chicken is in a different category i.e., Non-Veg.
* When I will click on the Close button which is inside the Sorted Recipes by Category window the window will be destroyed.



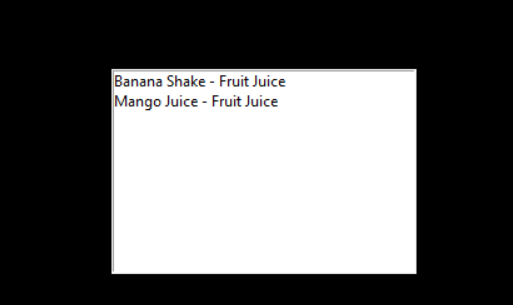
* User have to select the Recipe that he/she want to delete. When user will click on the Delete Recipes button the selected Recipe will be deleted from the listbox. Here I am deleting the Chicken recipe from the listbox.



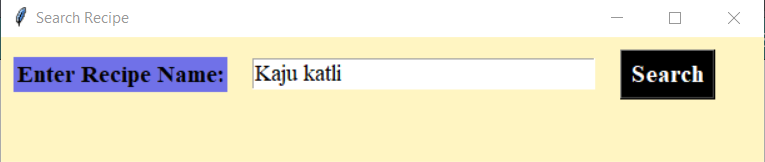
Then a messagebox will be appear where User have to choose from Yes or No whether user want to delete the selected recipe or not. If user will click on Yes Button then Recipe will be deleted and If click on No button then Recipe will not be deleted.



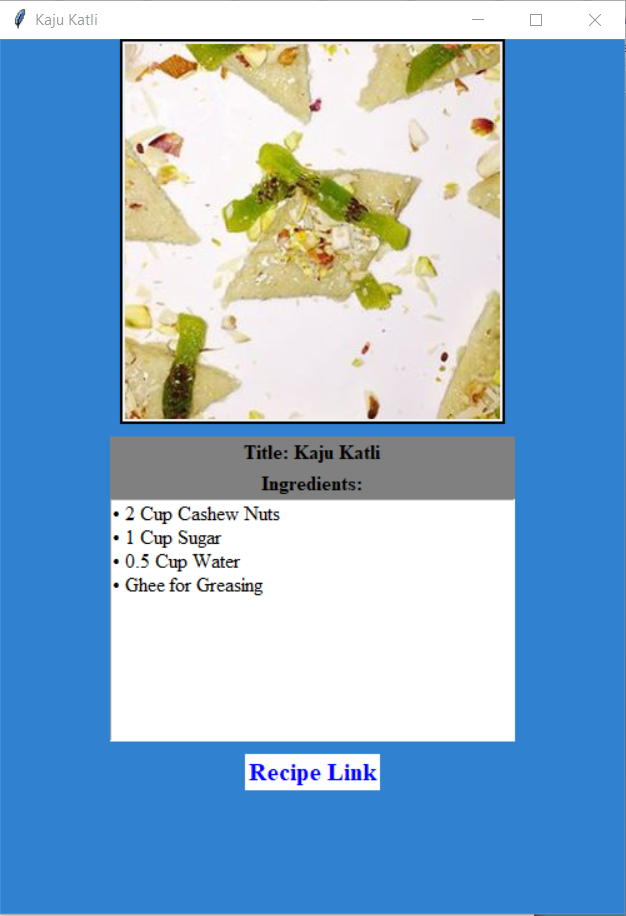
* See there is no Chicken Recipe in the Listbox of main window Recipe GUI



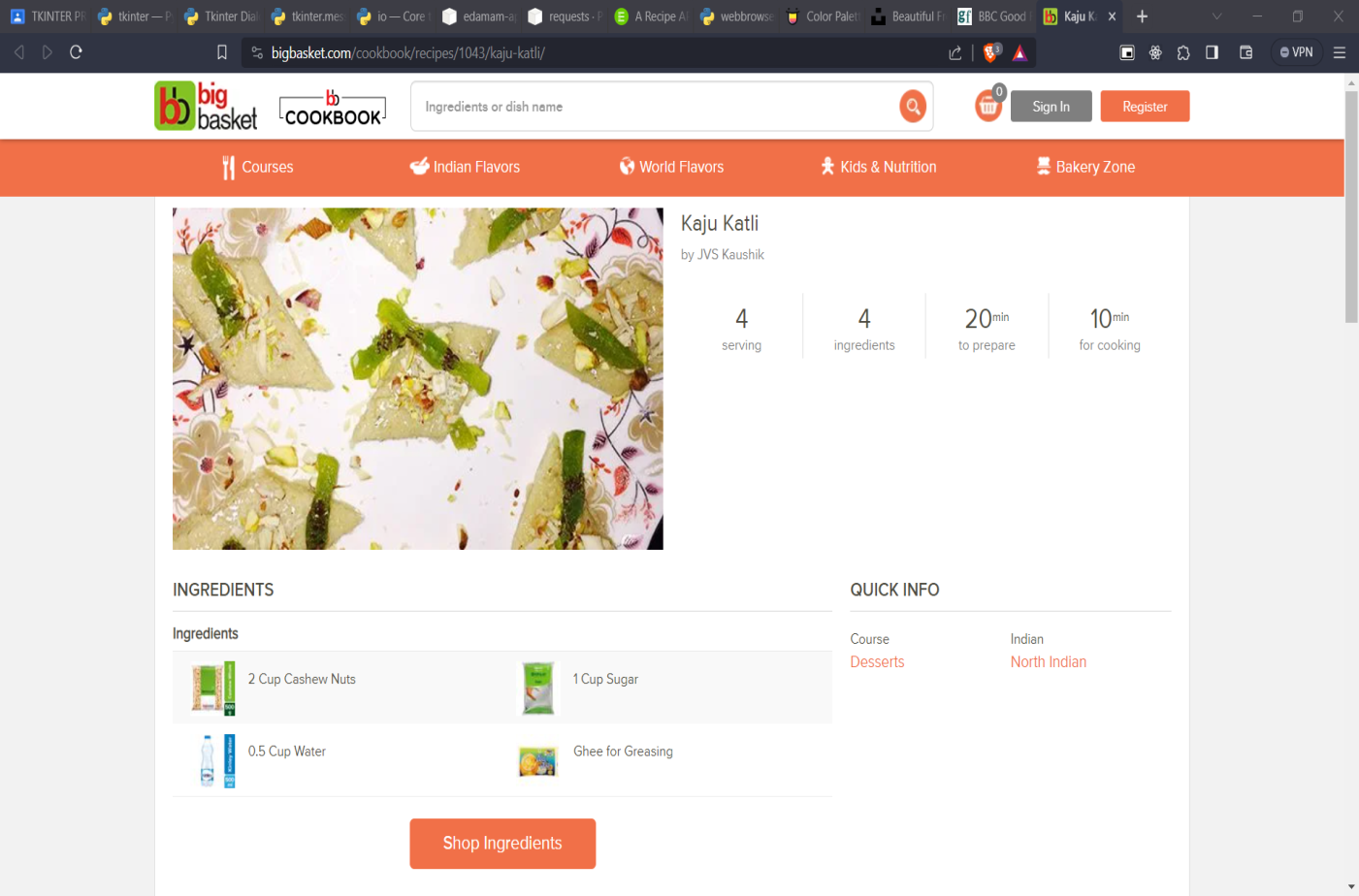
* Clicking Search Recipe button will open a new window of title Search Recipe there user have to enter the Food name which recipe they want then user have to click on Search button.

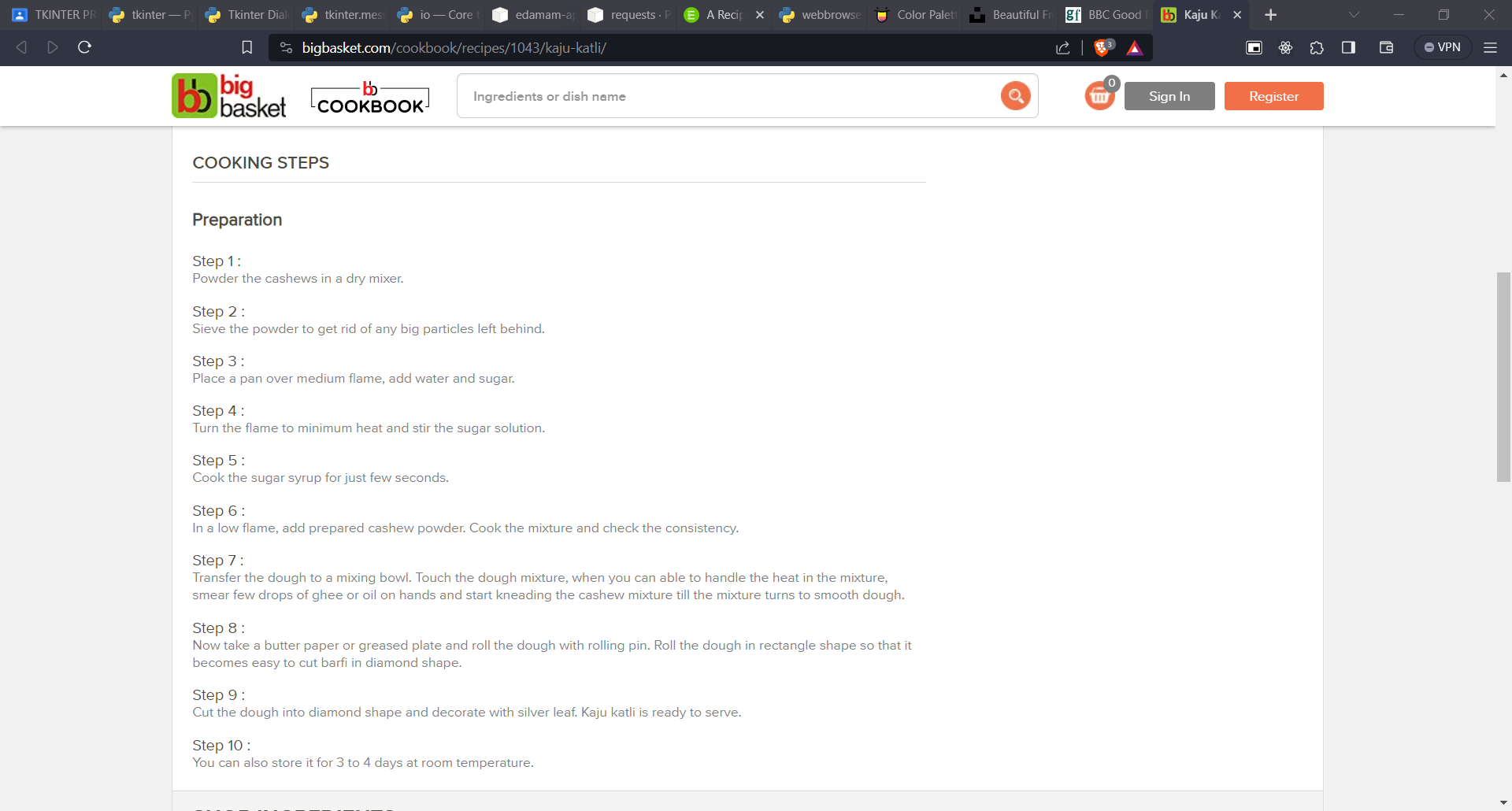


* After clicking on Search button again a new window of title “Recipe Name” will open there user will be able to see the Image of Recipe, Title that means name of Recipe, Ingredients as well and there will be a button Recipe Link on which when user click then a browser will open where whatever the particular recipe user had searched there instruction will be written.



* After clicking a Recipe Link button a browser is open where recipe ingredients and how to prepare the recipe it is also written in steps manner.





**FUTURE SCOPE**

The Recipe Organizer application can be extended and enhanced in several ways for future releases:

* **User Authentication**: Implement user authentication mechanisms to allow multiple users to maintain separate recipe collections.
* **Categorization and Tags**: Introduce a feature for categorizing recipes and adding tags, enabling users to filter recipes based on specific criteria.
* **Export and Import Functionality**: Allow users to export their recipe collections to share with others or import recipes from external sources.
* **Nutritional Information**: Integrate nutritional analysis features, providing users with insights into the health aspects of their recipes.
* **Cross-Platform Compatibility**: Develop versions of the application for different operating systems, ensuring a broader user base.
* **Login Validation**: If any user is coming to the Recipe Organizer GUI then user should be able to Login/Signup into the GUI and their data will be saved inside the database so that if existing user will come to the same GUI then user will only need to login and their validation/verification will be happen through datatbase whether they are existing or not.
* **Databse**: Whatever the Recipe that user is searching, adding, editing, deleting, that should be updated inside the database.

**CONCLUSION**

The Recipe Organizer application is a practical tool for individuals passionate about cooking and recipe management. Its user-friendly interface, combined with features like searching and sorting, makes it a valuable asset for culinary enthusiasts. The integration with the Edamam API expands the application's capabilities, providing users with a vast and diverse recipe database. Recipe Organizer is basically a GUI platform where any user who is enthusiasts to cook they can add their recipe, delete, edit, view, sort the recipe in different category, and last but not the least they can search also.

As the application continues to evolve, incorporating user feedback and adding new features will contribute to its ongoing success in simplifying recipe organization and exploration.

**REFERENCES AND BIBLIOGRAPHY**

1. Python Documentation: (<https://www.python.org/downloads/>)
2. Vscode: (<https://code.visualstudio.com/download>)
3. Tkinter Python Library: (<https://docs.python.org/3/library/tkinter.html>)
4. Tkinter Simpledialog module: (<https://docs.python.org/3/library/dialog.html>)
5. Tkinter Messagebox module: (<https://docs.python.org/3/library/tkinter.messagebox.html>)
6. Io module in Python: (<https://docs.python.org/3/library/io.html>)
7. Edamam python: (<https://pypi.org/project/edamam-api/>)
8. Requests module in python: (<https://pypi.org/project/requests/>)
9. Webbrowser module in python: (https://docs.python.org/3/library/webbrowser.html)
10. Edamam api website: (<https://developer.edamam.com/edamam-recipe-api>)
11. Unsplash website for background image: (https://unsplash.com/)
12. Colorhunt website for color: (<https://colorhunt.co/>)