ChefGPT

Table of Contents

Objective 2

Steps to Run the Project 2

Dataset 2

Chatbot Stages in Action 3

  System Design 3

  Stage 1: Engagement with the User 3

  Stage 2: Recipe Filtering 3

  Stage 3: Recipe Presentation 3

Important Files 4

- app.py 4
- functions.py 5
- food_recipe.py 7

Flow Chart 9

Example Conversation 10

Conclusion 13

  Important Lessons learned 13

  Areas of Improvement 13

Objective

The goal of this project is to implement a chatbot that can understand user preferences for food recipes and provide suggestions from a pre-existing dataset.

Steps to Run the Project

1. **Ensure OpenAI Account and API Keys:**
    - Create an account on OpenAI if you don't have one.
    - Make sure you have enough credits to use the 'gpt-4o' model.
    - Obtain your API keys and save them in a file named **Open_AI_Key.txt**.
2. **Open the Project in Visual Studio Code:**
    - Launch Microsoft Visual Studio Code on your computer.
    - Open the project folder in Visual Studio Code.
3. **Run the Main Application File:**
    - Locate the **app.py** file in the project directory.
    - Execute the **app.py** file to start the chatbot. This file contains the initial flow of the code and will set everything in motion.
4. You can open the webpage at http://127.0.0.1:5000/ to converse with the ChefGPT.

Note:

- If you encounter any missing libraries, please ensure you install them using a package manager like **pip**. You can install the required libraries by running **pip install -r requirements.txt**, or by installing each library individually as needed.
- Since we are using online LLM models, sometimes response might be not as expected, so try again by clicking on ‘End Conversation’ button or restarting the application.

By following these steps, you should be able to successfully run the ChefGPT project and interact with the chatbot to get personalized food recipe suggestions.

Dataset

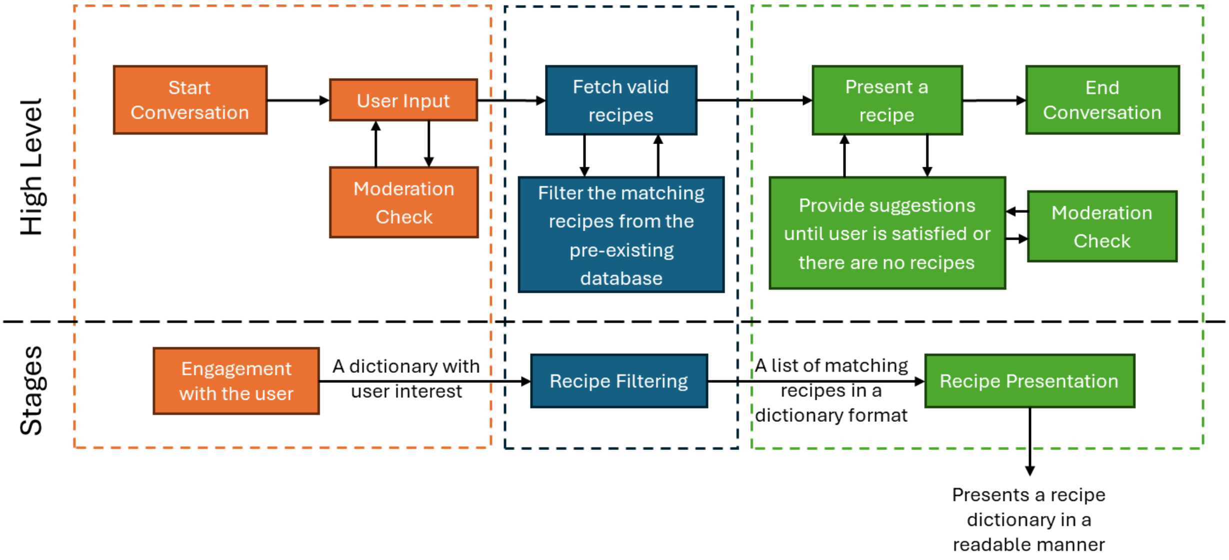
The dataset used in this project is a refined version of the 6000+ Indian Food Recipes Dataset (kaggle.com) available on Kaggle.

Modifications made to the original dataset:

1) Filtered for English Content
2) The dataset was streamlined to include only the following relevant columns ('Srno', 'Recipe Name', 'Ingredients', 'PrepTimeInMins', 'CookTimeInMins', 'TotalTimeInMins', 'Servings', 'Cuisine', 'Course', 'Diet', 'Instructions', 'URL')

Chatbot Stages in Action

System Design


High Level
Stages
Start Conversation
User Input
Moderation Check
Fetch valid recipes
Filter the matching recipes from the pre-existing database
Present a recipe
End Conversation
Provide suggestions until user is satisfied or there are no recipes
Moderation Check
Engagement with the user
A dictionary with user interest
Recipe Filtering
A list of matching recipes in a dictionary format
Recipe Presentation
Presents a recipe dictionary in a readable manner

Stage 1: Engagement with the User

The chatbot initiates a conversation with the user to understand their preferences regarding recipes. It asks about various aspects such as cuisine preference, type of course (appetizer, main course, dessert, etc.), dietary restrictions or preferences, preferred preparation and cooking times, and the number of servings desired. This ensures that the recommendations provided are tailored to the user's specific requirements and tastes. **Function calling** technique is used here for extracting the information from the user.

Stage 2: Recipe Filtering

Once the chatbot gathers all the necessary information from the user, it sifts through its database of recipes. Using the criteria provided by the user, the chatbot filters out recipes that match the specified parameters. This step ensures that only relevant recipe options are presented to the user, saving time and effort in the selection process.

Stage 3: Recipe Presentation

With a refined list of suitable recipes, the chatbot proceeds to present them to the user one by one. Each recipe is showcased along with its ingredients, cooking instructions, and any additional relevant details. The chatbot continues this process until the user finds a recipe that piques their interest or satisfies their requirements. If the chatbot exhausts its list of suggestions without finding a suitable match, it notifies the user about the unavailability of further recommendations.

Important Files

➢ app.py

This file is a Flask web application designed to function as a chatbot that suggests food recipes based on user input. The key components and functionalities of the application are as follows:

Imports and Setup

- **Libraries and Modules**: Imports Flask for web framework, OpenAI for GPT-based responses, and custom functions for conversation initialization, recipe fetching, and moderation.
- **OpenAI API Key**: Reads the API key from a file for authenticating OpenAI API requests.

Global Variables

- *conversation_bot*: List to store the conversation history between the user and the bot for understanding user's recipe interest.
- *conversation_reco:* List to store the conversation history between the user and the bot for understanding user feedback on suggested recipe.
- *conversation*: Initializes the conversation state.
- *introduction*: Stores the initial welcome message from the chatbot.
- *recipes*: Stores the recipes fetched based on user input.
- *explained_recipes*: Tracks the recipes that have been explained to the user.

Flask Routes

1. **/ (default route):**
   Renders the main chat interface (index_chat.html) and displays the conversation history.

2. ***/end_conv* (end conversation):**
   Resets the global variables and redirects to the default route to start a new conversation.

3. ***/chat* (chat endpoint): -> Dialogue management system**
   - Handles user input from the chat interface.
   - Processes the input to understand user requirements and appends user messages to the conversation history.
   - Uses OpenAI's GPT model to generate responses from the chatbot.
   - Performs moderation checks to ensure appropriate content.
   - If all the user requirements are obtained, it fetches recipes, and starts the feedback loop.
   - If recipes are already available, it continues the conversation to get user feedback on the suggested recipes and makes additional suggestions if needed.

Detailed Functional Flow

- Moderation Check: Ensures that user input and bot responses adhere to content guidelines.
- Recipe Fetching and Suggestion:
  - Implements Open AI function calling technique to fetch recipes based on user-provided criteria (cuisine, course, diet, preparation time, cooking time, servings).
  - Because of the use of function calling, the chatbot takes care of making sure if the required inputs from the user are obtained.
  - Suggests a random recipe from the fetched list and starts the feedback process.
- Feedback Handling:
  - Manages user feedback to either provide additional recipe suggestions or conclude the conversation.

Execution

Runs the Flask application on 0.0.0.0 in debug mode when executed directly.

Summary

The app.py file implements a conversational chatbot using Flask and OpenAI's GPT model to assist users in finding and selecting food recipes. It handles user interactions, fetches recipes based on specified criteria, ensures content moderation, and processes user feedback to refine recipe suggestions or conclude the chat session. The application maintains the state of the conversation and dynamically updates the chat interface based on user interactions.

➢ functions.py

This file uses OpenAI's GPT model to interact with users, understand their recipe requirements, and suggest appropriate recipes. The key components and functionalities of the code are outlined below:

Imports and Setup

- **Libraries and Modules:** Imports necessary libraries such as OpenAI for GPT model interactions
- **Cuisine, Course, and Diet Lists:** Defines lists of unique cuisines, courses, and diets that the chatbot can suggest. This list is fetched from ‘food_recipe.xlsx’ file.
- **requirement_tools**: Defines a list containing a function *get_food_recipe* with parameters like cuisine, course, diet, preparation time, cooking time, and servings. These parameters help in fetching the appropriate recipe based on user input.

Functions

1. ***get_chat_model_completions(messages, needTools=True):***
   - Interacts with OpenAI's GPT model to get responses based on the conversation history (messages).
   - If *needTools* is True, the api call includes the defined requirement_tools to suggest recipes and implement function calling.

2. ***moderation_check(user_input):***
   - Checks the user input for any inappropriate content using OpenAI's moderation API.
   - Returns "Flagged" if the content is inappropriate, otherwise returns "Not Flagged".

3. ***initialize_conversation():***
   - Initializes the conversation with a system message that sets up the context for the chatbot.
   - The system message includes detailed instructions on how the chatbot should interact with the user to understand their recipe requirements.
   - The conversation starts with the chatbot role set to "system".

4. ***initialize_conv_reco():***
   - Initializes a separate conversation context for understanding user satisfaction with the recipe suggestions.
   - The system message instructs the chatbot to determine if the user is satisfied and respond with "Yes", "No", or "Reset".

System Messages:

- Provide detailed instructions to the chatbot on how to understand and fulfil user requests.
- Include example conversations to guide the chatbot's interactions.

Chain of Thoughts:

- The chatbot follows a structured approach to understand user requirements through a series of thoughtful questions.
- The goal is to fill out a dictionary with keys for Cuisine, Course, Diet, Preparation Time, Cooking Time, and Servings.
- Once the necessary information is gathered, the chatbot makes a function call to fetch the recipe.

Summary

The provided code defines the core logic and structure for a food recipe chatbot. It uses OpenAI's GPT model to engage with users, understand their food preferences, and suggest appropriate recipes. The chatbot is designed to follow a systematic approach to gather user requirements, perform moderation checks, and handle user feedback effectively. The functions initialize_conversation and initialize_conv_reco set up the initial context for recipe suggestions and user satisfaction assessment, respectively.

➢ food_recipe.py

The file is part of a recipe suggestion system that filters recipes based on user requirements and generates detailed HTML reports for these recipes. The key components and functionalities of the code are as follows:

Imports and Data Setup

- **Libraries**: Imports necessary libraries including pandas for data manipulation, random for random selection, and html for HTML escaping.
- **Data**: Reads a recipe database from an Excel file “food_recipe.xlsx” into a DataFrame recipe_db.

Functions

1. ***filterBasedOnPrepAndCookTime(userPrepTime, userCookTime, recipePrepTime, recipeCookTime, recipeTotalTime)***

- *Purpose*: Filters recipes based on user-provided preparation and cooking time preferences.
- *Logic*:
  - Calculates the ratios of preparation and cooking times to the total recipe time.
  - Checks if these ratios match the user's preferences (e.g., less prep time and more cook time).
  - Example, if prep time was mentioned less and cook time was mentioned more, then preparation time ratio should be less than cooking time ratio to return True.
  - Returns True if the recipe matches the user's time constraints, False otherwise.

2. ***get_food_recipes(recipe_requirement)***

- *Purpose*: Filters the recipes in the database based on user requirements such as cuisine, diet, course, preparation time, cooking time, and servings.
- *Logic*:
  - Filters recipes in recipe_db according to the specified cuisine, diet, and course.
  - Further filters these recipes based on the preparation and cooking time using the filterBasedOnPrepAndCookTime function.
  - Prepares a list of valid recipes in dictionary format by extracting and formatting relevant information (name, ingredients, preparation time, cooking time, instructions, URL, and servings).
  - Note: We can also provide a list of HTML formatted strings as output, but a dictionary is created so that it could be used later for any more processing at ease.

3. ***generate_recipe_report(recipe)***

- *Purpose*: Generates an HTML report for a given recipe.
- *Logic*:
  - Constructs an HTML formatted string that includes the recipe name, preparation time, cooking time, ingredients, instructions, and servings.
  - Servings will be updated with a disclaimer about the difference in the user requested servings and the recipe mentioned servings.
  - Includes a link to the recipe's URL.
  - Adds a note to adjust ingredient quantities if the recipe servings differ from the user's specified servings.

4. ***get_random_recipe(recipes, explained_recipes)***

- *Purpose*: Selects a random recipe from the list of valid recipes that hasn't been explained yet and generates its HTML report.
- *Logic*:
  - Randomly selects a recipe index from the list of recipes.
  - Ensures that the selected recipe has not been previously explained by checking against explained_recipes.
  - Generates an HTML report for the selected recipe using generate_recipe_report.
  - Returns the HTML report and updates the list of explained recipes.
  - If all recipes have been explained, returns None.

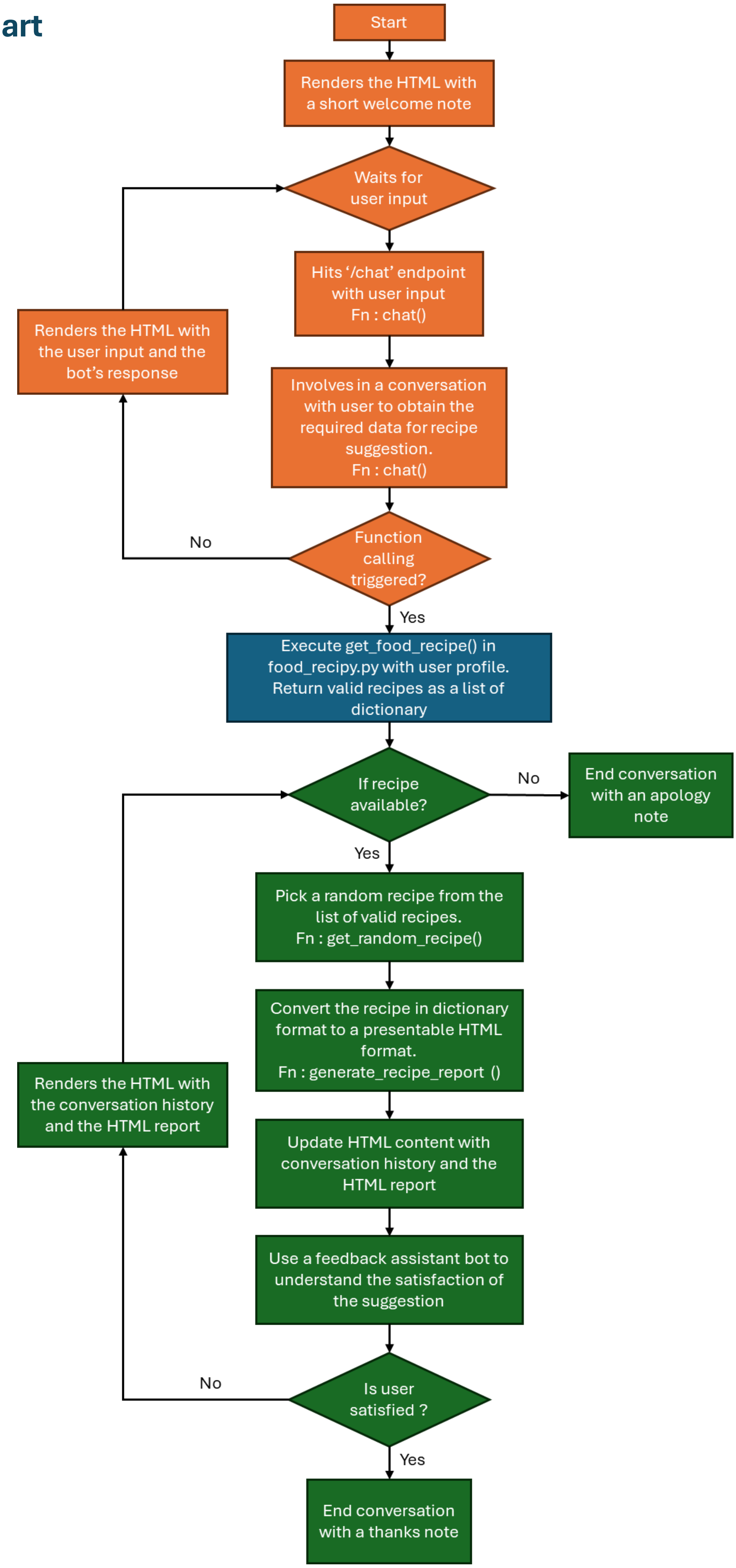
Summary

The code provides functionalities to:
1. Filter recipes based on user-specified preparation and cooking times.
2. Retrieve recipes that match user-defined criteria (cuisine, diet, course, etc.).
3. Generate detailed HTML reports for the selected recipes.
4. Randomly select recipes that have not been previously explained to the user.

This allows the system to interactively suggest and explain recipes based on user preferences, enhancing the user's experience in finding suitable recipes.

Based on the above understanding of each file, let us look at the overall flow of the code.

Flow Chart


Start
Renders the HTML with a short welcome note
Waits for user input
Hits '/chat' endpoint with user input Fn : chat()
Involves in a conversation with user to obtain the required data for recipe suggestion. Fn : chat()
Function calling triggered?
No
Renders the HTML with the user input and the bot's response
Yes
Execute get_food_recipe() in food_recipy.py with user profile. Return valid recipes as a list of dictionary
If recipe available?
No
End conversation with an apology note
Yes
Pick a random recipe from the list of valid recipes. Fn : get_random_recipe()
Convert the recipe in dictionary format to a presentable HTML format. Fn : generate_recipe_report ()
Update HTML content with conversation history and the HTML report
Use a feedback assistant bot to understand the satisfaction of the suggestion
Is user satisfied ?
No
Renders the HTML with the conversation history and the HTML report
Yes
End conversation with a thanks note

Example Conversation

The below table shows an example conversation experimented with the ChefGPT.

Role	Message
ChefGPT	Hello! I'm excited to help you find the perfect recipe. Could you please share some details about the type of cuisine or dish you're interested in?
User	I am looking for a Mexican lunch
ChefGPT	Fantastic choice! Mexican cuisine is vibrant and full of flavors. Are you looking for a vegetarian, non-vegetarian, or vegan lunch?
User	Non veg
ChefGPT	Got it! A non-vegetarian Mexican lunch sounds delicious. Do you have more or less time for preparation and cooking?
User	More time to cook than prepare
ChefGPT	Perfect! And how many people will be enjoying this meal?
User	5
ChefGPT	Thank you for providing all the information. Kindly wait, while I fetch the recipes...
ChefGPT	**Recipe Name:** Chicken Chilli Cheese Quesadillas Recipe **Preparation Time:** 10.0 **Cooking Time:** 60.0 **Ingredients:** • 250 grams Boneless chicken. • 1 tablespoon Chipotle sauce. • 1 cup Cheddar cheese - shredded. • 1 Onion - sliced. • 1 Green Bell Pepper (Capsicum) - sliced.

- 1/2 teaspoon Black pepper powder - crushed.
- Salt - as required.
- 1/2 cup Sweet corn.
- 2 teaspoons Roasted tomato pasta sauce.
- 1 teaspoon Lemon juice.
- 1 teaspoon Butter.
- 6 Tortillas.

Instructions:

- To begin making the Chicken Chilli Cheese Quesadillas recipe, firstly marinate chicken with chipotle paste, salt, crushed pepper and lime juice for minimum 40 minutes.
- Heat butter in a non stick pan, add onion and sliced green pepper and saute for a while.
- Now add the marinated chicken and saute for some more time.
- Add tomato sauce and mix well.
- Cook till chicken softens.
- Transfer the chicken mix in a microwave bowl and add in the cheese and corn.
- Mix well.
- Microwave on high for 2 minutes in oven and allow it to rest for a while.
- Take one tortilla sheet and add spoonful of chicken chilli cheese mix and spread evenly.
- Place another tortilla over it and grill on stove top grill pan.
- Carefully flip the other side and grill again by brushing a little butter on top.
- Transfer in a plate and carefully cut the quesadillas into four quadrants using a pizza cutter.
- Similarly make more quesadillas and serve hot with your favourite dip.
- Serve Chicken Chilli Cheese Quesadillas along with Baked Mexican Rice and Corn Pineapple Salsa Salad for a perfect Mexican dinner on a weekend night.

Servings:
Instructions are for 4.0 people. Remember to scale up ingredients accordingly.

For more details, you can refer at https://www.archanaskitchen.com/chicken-chilli-cheese-quesadillas-recipe

ChefGPT Did the recipe suggestion meet your expectations?

User Suggest something else

ChefGPT **Recipe Name:** Mexican Low Fat Fish Wraps Recipe

Preparation Time: 10.0

Cooking Time: 30.0

Ingredients:

- 1 Whole Wheat Flour Tortilla.
- 200 grams Basa fish - boneless and skinless.
- 2 Iceberg lettuce.
- 1 teaspoon Sunflower Oil.
- 2 tablespoons Black pepper powder.
- 1/4 teaspoon Salt.
- 2 tablespoon Lemon juice.
- 1 Tomato - finely chopped.
- 1 Onion - finely chopped.
- 2 tablespoon Coriander (Dhania) Leaves - finely chopped.
- 1 Green Chilli - finely chopped.
- 1 tablespoon Extra Virgin Olive Oil.
- 1 tablespoon Lemon juice.
- 1 pinch Salt.
- 1/2 cup Hung Curd (Greek Yogurt).
- 1 tablespoon Italian seasoning.

Instructions:

- To begin making the Mexican Low Fat Fish Wraps recipe, marinate the fish by applying black pepper powder, lemon juice and salt.
- Keep it aside for 30 minutes.
- For salsa, mix the ingredients mentioned for salsa in a mixing bowl and keep it aside.
- Mix yogurt and seasoning for yogurt sauce.
- Pre-heat the oven for 5 minutes.
- Place the marinated fish on the skewers, brush them with oil and grill them for 20 Minutes at 200 C.
- To assemble the Mexican fish wraps, heat the tortilla for few seconds.
- Then place it on a plate.
- In the centre spread the lettuce.
- Over it spread 2 tablespoon of salsa.
- Over this, place the grilled fish pieces.
- Spread 2 tablespoons of yogurt sauce on the top of the fish.
- Roll the tortillas by bringing the opposite sides together.
- If the roll is very big, cut it in halves and serve with yoghurt sauce and salsa.

- Serve Mexican Low Fat Fish Wraps along with Spicy Mexican Salsa Verde and a glass of Pomegranate Mint Cocktail Recipe for a perfect weekend dinner.

Servings:
Instructions are for 4.0 people. Remember to scale up ingredients accordingly.

For more details, you can refer at https://www.archanaskitchen.com/mexican-low-fat-fish-wraps-recipe

ChefGPT How about this recipe?

User Perfect!

ChefGPT Thank you for using the assistance ! Enjoy your food !

Conclusion

With the ChefGPT developed, the user can interact with the bot to get a recipe of interest.

Important Lessons learned

- Developed an end-to-end system starting from data source till recipe suggestion.
- Developed a feedback system to understand if the user is satisfied with the suggestion or else keep suggesting till we have enough data.
- Implemented Function Calling API and biggest advantage is that it takes care of filling all the required data.
- Experimented with different prompts to improvise the LLM model's response.
- Implemented this whole chatbot on a web interface.

Areas of Improvement

- Currently the knowledge of the chatbot is with respect to the food_recipe.xlsx file. We can expand that to fetch details from online sources.
- Understanding the user requirements can be improved to fetch more details, like specific ingredients of choice or specific recipe names.
- Feedback assistant could be improved to better understand the user satisfaction.