Final Project SI 507

- Prekshah Kamleshkumar Shah
- 8645 1278

Project Code:

GitHub repo Link : [Preksha0420/FindMeTheRecipe]

README uploaded on GitHub.

Data Sources:

The program the Spoonacular API and its /analyzedInstruction extension to retrieve recipe suggestions, ingredients and recipe steps for recipes based on user preferences and inputs.

- Spoonacular API Recipe Search: "https://api.spoonacular.com/recipes/complexSearch?apiKey=" + API_key
- Spoonacular API analyzed instructions: "https://api.spoonacular.com/recipes/" + idd + "/analyzedInstructions/?apiKey=" + API_key
- All spoonacular API documentation can be found at: https://spoonacular.com/food-api/docs

The data is retrieved from these APIs using the requests package in Python. All data is retrieved from the API in JSON and cached appropriately.

The code caches the top recipe suggestions for a certain query with the key being “recipe name” + “cuisine” + “diet” and the recipes suggested as the value of this key. The code also caches the recipe instructions retrieved from the /analyzedInstruction extension by storing the instructions in the value of the “recipe_id” key.

We use the Recipe Search URL along with various parameters to search for recipe suggestions. Currently in the program, we use only 3 parameters (query, diet and cuisine) to search for recipes. Diet and Cuisine are provided by the user as they play along the initial two tree while query is a direct input of what recipe they would like to search. We use requests.get to get the JSON data and then json.loads to load everything in a usable python data type (dictionary). We then store each individual recupe as an object of the “Recipe” class we have defined.

Once the user wants to learn more about a particular suggested recipe, we will use its “recipe_id” to search with the analyzed instruction extension for recipe ingredients and instructions.

The spoonacular API has over 5000 recipes, of which for testing purposes, I have retrieved about 150 recipes. Below are screenshots of the fields you retrieve from the recipe search API and the analyzedInstruction API:

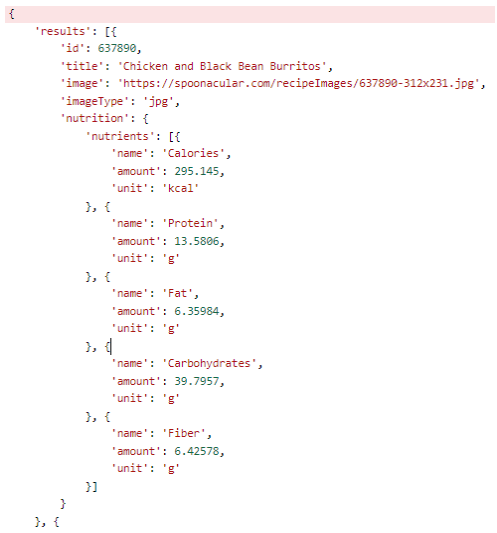


Figure 1: Output of recipe search step(1 recipe)

Figure 2: Output of recipe instructions search

For the recipe search step, we store the “id”, “title”, “image”, “Calories” amount, “Protein” amount, “Fat” amount, “Carbohydrates” amount and “Fiber” amount in our Recipe class.

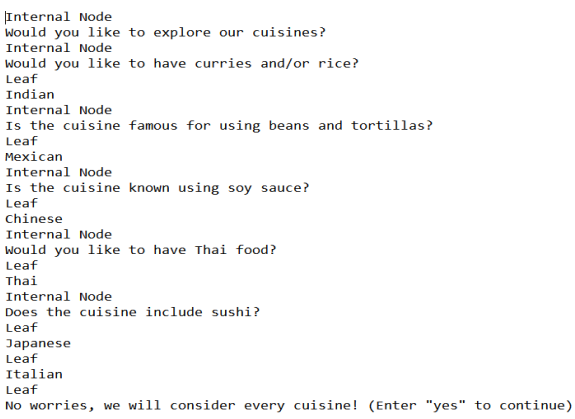
For the recipe instructions search, we store the “ingredients” “name” for each ingredient in a list and print the “step” for each step usign a function.

Data Structures:

README : Data structure description added to README.

Required Python and JSON files are added to GitHub Repository.

The program uses lists, dictionaries, classes and trees to store and navigate through the neccessary data. The program starts with two trees (Cuisine.txt and Diet.txt). The program loads the current tree from these files and plays it with the user to gather their preferences on cuisine and diet. The program saves the trees to the .txt file to make sure any changes are accounted for.



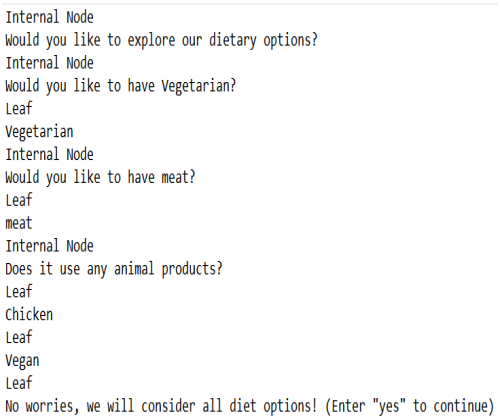


Figure 3: Tree .txt files (program loads and saves the tree in these files)

Then we use these parameters along with the recipe search query to search for recipe suggestions from the API. Each output recipe is stored as an object of the “Recipes” class. For queries that have multiple recipes, we store them as a list of objects.

```
class Recipe:
    def __init__(self, i_d, title, image, calories, protein, fat, carbs, fiber):
        self.i_d = i_d
        self.title = title
        self.image = image
        self.calories = calories
        self.protein = protein
        self.fat = fat
        self.carbs = carbs
        self.fiber = fiber
```

Figure 4: Recipe Class definition

```
rec = get_recipe_name(q, cuisine=cuisine, diet=diet)
i_ds = [results["id"] for results in rec["results"]]
titles = [results["title"] for results in rec["results"]]
cals = [str(results["nutrition"]["nutrients"][0]["amount"]) for results in rec["results"]]
pro = [str(results["nutrition"]["nutrients"][1]["amount"]) for results in rec["results"]]
f = [str(results["nutrition"]["nutrients"][2]["amount"]) for results in rec["results"]]
c = [str(results["nutrition"]["nutrients"][3]["amount"]) for results in rec["results"]]
fib = [str(results["nutrition"]["nutrients"][4]["amount"]) for results in rec["results"]]
img = [results["image"] for results in rec["results"]]

recipes = [Recipe(i_ds[i], titles[i], img[i], cals[i], pro[i], f[i], c[i], fib[i]) for i in range(0, len(titles))]
```

Figure 5: Storing Recipes as Objects of Recipe Class

The last step is to search for specific recipe instructions (using /analyzedInstruction). This search outputs for two useful data points, the ingredients and the recipe steps. The ingredients are stored in a list (removing any duplicates mentioned in the API JSON) and the steps are simply printed directly from the resulting dictionary.

Finally, we save all retrieved information in appropriate dictionaries and cache them as the recipes.json file. We first save the recipe suggestions as the value to a “recipe name” + “cuisine” + “diet” key and secondly, we save the analyzed instructions as the value to the “recipe_id” key which is specific for each recipe in the API. Below is an image of the JSON file highlighting the necessary information.


```
{"noodleschinesevegetarian": [{"i_d": 662669, "title": "Swiss Chard Linguine", "image": "https://spoonacular.com/recipeImages/662669-312x231.jpg", "calories": "71.8351", "protein":
"2.27227", "fat": "2.12989", "carbs": "9.876", "fiber": "1.13599"}, {"i_d": 644885, "title": "Gluten Free Vegan Gnocchi", "image": "https://spoonacular.com/recipeImages/644885-312x231.jpg",
"calories": "132.624", "protein": "3.54427", "fat": "0.373333", "carbs": "26.2111", "fiber": "3.25376"}, {"i_d": 655589, "title": "Penne with Goat Cheese and Basil", "image":
"https://spoonacular.com/recipeImages/655589-312x231.jpg", "calories": "151.211", "protein": "5.50989", "fat": "4.76624", "carbs": "20.2824", "fiber": "0.939185"}, {"i_d": 634437, "title":
"Basil Tagliatelle with Roasted Red Bell Pepper Salad", "image": "https://spoonacular.com/recipeImages/634437-312x231.jpg", "calories": "200.257", "protein": "5.16829", "fat": "11.9114",
"carbs": "15.7966", "fiber": "7.96785"}, {"i_d": 665527, "title": "Yellow Squash Noodles with Tomato Basil Sauce", "image": "https://spoonacular.com/recipeImages/665527-312x231.jpg",
"calories": "219.624", "protein": "6.93901", "fat": "12.8262", "carbs": "16.9605", "fiber": "8.2807"}, {"i_d": 633711, "title": "Baked Penne", "image":
"https://spoonacular.com/recipeImages/633711-312x231.jpg", "calories": "241.208", "protein": "10.511", "fat": "1.29692", "carbs": "43.0405", "fiber": "6.05634"}, {"i_d": 640693, "title":
"Creamy Ratatouille Over Penne", "image": "https://spoonacular.com/recipeImages/640693-312x231.jpg", "calories": "308.03", "protein": "8.9482", "fat": "11.9317", "carbs": "36.8335",
"fiber": "7.2875"}, {"i_d": 644205, "title": "Garden Vegetable Ravioli With Tomato Brodo", "image": "https://spoonacular.com/recipeImages/644205-312x231.jpg", "calories": "349.673",
"protein": "10.2907", "fat": "17.403", "carbs": "36.1682", "fiber": "5.06916"}, {"i_d": 1487863, "title": "Baked Macaroni and Cheese", "image":
"https://spoonacular.com/recipeImages/1487863-312x231.jpg", "calories": "366.872", "protein": "15.6076", "fat": "22.3432", "carbs": "24.8506", "fiber": "1.07956"}, {"i_d": 631732, "title":
"Caponata Style Celery Spaghetti", "image": "https://spoonacular.com/recipeImages/631732-312x231.png", "calories": "378.76", "protein": "9.29895", "fat": "15.2415", "carbs": "47.2253",
"fiber": "5.39225"}], "1487863": [{"name": "", "steps": [{"number": 1, "step": "Heat oven to 350 degrees Fahrenheit.\u00a0", "ingredients": [], "equipment": [{"id": 404784, "name": "oven",
"localizedName": "oven", "image": "oven.jpg", "temperature": {"number": 350.0, "unit": "Fahrenheit"}}]}, {"number": 2, "step": "Cook noodles until slightly tender but not fully cooked.
```


Figure 6: Recipe suggestions and instructions as dictionary in JSON file

Interaction and Presentation:

The program relies both on command line prompts and the command window interface to interact with the user. More than 60 percent of the program relies on user interaction and user preferences.

The program starts with two trees where the user needs to provide “yes” or “no” inputs based on the command line prompts to select their preferred cuisine and diet. If the cuisine and/or diet are not mentioned in the tree, the user can then input their cuinine and/or tree to the program along with a distinguishing question. This will be stored for future use.

Then the user needs to input a string recipe search query which is the recipe/food item they want to search for.

Recipe suggestions are displayed on the command window. The displayed results include a local recipe number, recipe name, total calories, protein, fats, carbs and fiber content for each recipe.

The user is then prompted to select any one of these recipes to learn more about it by entering the local recipe number. If the user does not wish to select any of these recipes, they can enter a string “exit” to exit the application.

If the user selects a recipe via its local number, then we play a tree with the user to see whether they have all the ingredients or not. The questions of this tree are simple “Do you have xx?: ”, where xx is the ingredient name. The user replies with a “yes” or “no”. If the user is missing any one of the ingredients, then we will end the tree and display that they cannot make the recipe, and give them an option to select a different number or click exit.

If the user has all the ingredients, we will display the steps required to make the recipe and finally give the user an option to either check out a different recipe or click exit. This is the final functionality of the code. We will navigate through the complete user interface in the demo window.

Demo Link:

https://umich.zoom.us/rec/share/VO9fvexHbtlyM7nwH2AV8wyMzc2KJADqquM0vK6xeDftDXCpcxb1H2ve_CwI4am0.OPHjJV9UjM8z5C4z