FOODILICIOUS

kitchen

Foodilicious

A Recipe Search Engine

Phase 1 Implementation

—

J Kavitha, 17PW19
A Kowshika Rani, 17PW20

Overview

In this phase one we have implemented data collection through scrapping and cleaned the processed data and then indexed the recipes for better retrieval.

Milestones

I. Scraping

II. Preprocessing

III. Indexing

Scraping

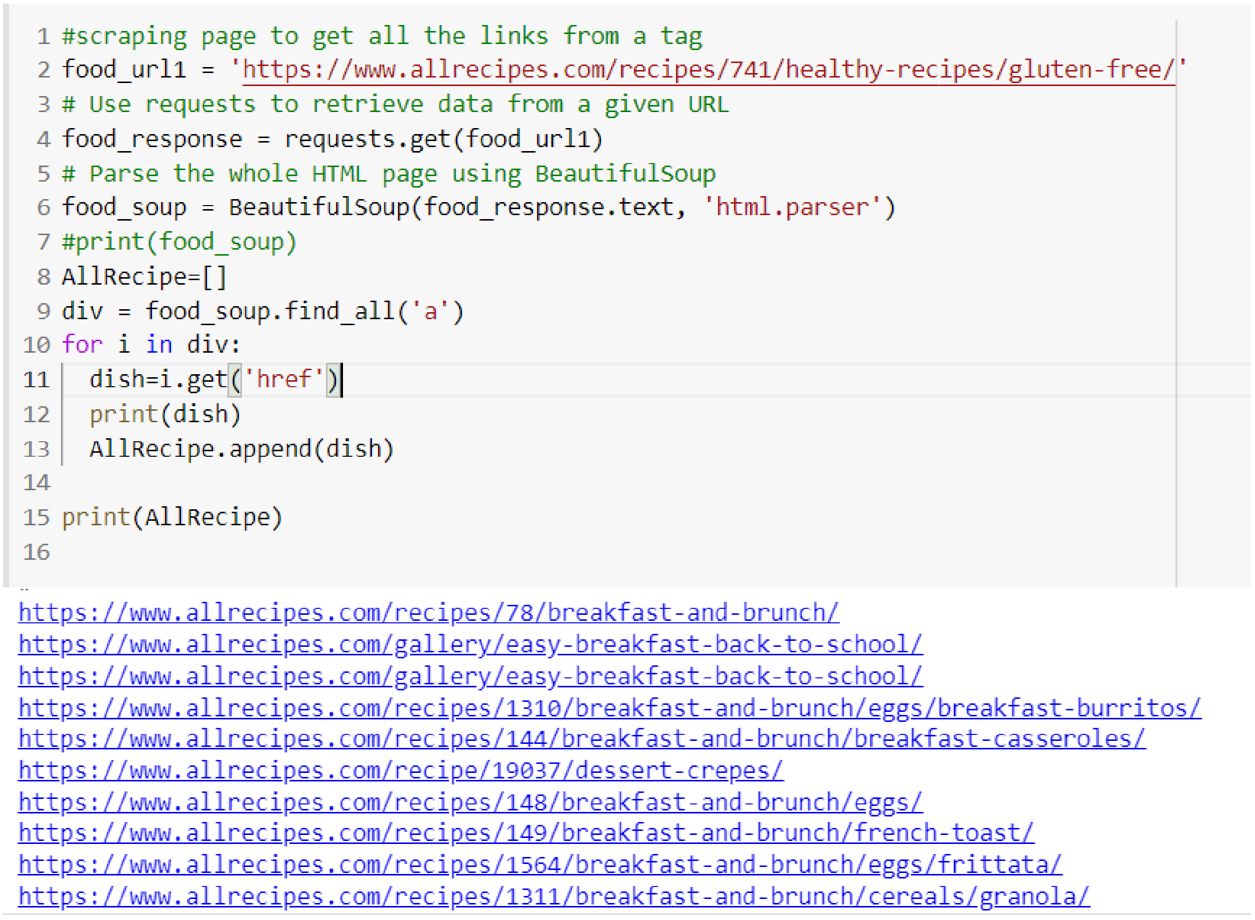
The recipes are scraped from various websites like

- Allrecipes.com
- Foodista.com
- SpoonacularAPI
- Cookpad.com

The data is scraped in the **json** format. Python is used for scraping the data. BeautifulSoup, a python library, is used to scrap the data out of HTML. Other necessary libraries are imported, and the recipes are scrapped.

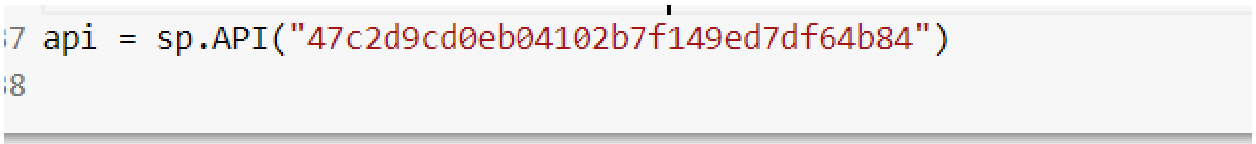
AllRecipe.com

The below snippets are used to scrape all the recipe links, then using the API key of spoonacular, the data is obtained in json and written in the output file respectively.

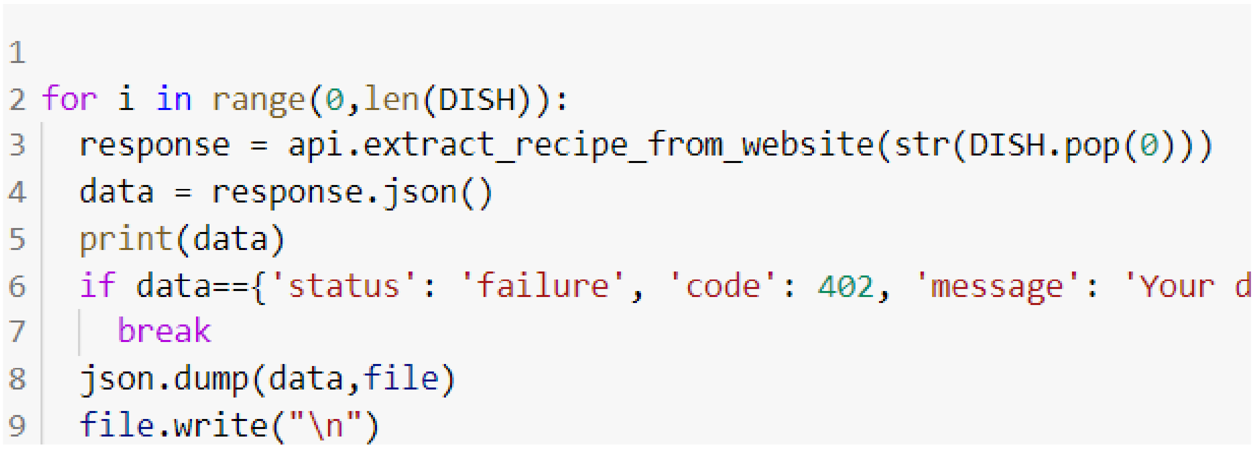


```
#scraping page to get all the links from a tag
food_url1 = 'https://www.allrecipes.com/recipes/741/healthy-recipes/gluten-free/'
# Use requests to retrieve data from a given URL
food_response = requests.get(food_url1)
# Parse the whole HTML page using BeautifulSoup
food_soup = BeautifulSoup(food_response.text, 'html.parser')
#print(food_soup)
AllRecipe=[]
div = food_soup.find_all('a')
for i in div:
    dish=i.get('href')
    print(dish)
    AllRecipe.append(dish)

print(AllRecipe)

```

```
https://www.allrecipes.com/recipes/78/breakfast-and-brunch/
https://www.allrecipes.com/gallery/easy-breakfast-back-to-school/
https://www.allrecipes.com/gallery/easy-breakfast-back-to-school/
https://www.allrecipes.com/recipes/1310/breakfast-and-brunch/eggs/breakfast-burritos/
https://www.allrecipes.com/recipes/144/breakfast-and-brunch/breakfast-casseroles/
https://www.allrecipes.com/recipe/19037/dessert-crepes/
https://www.allrecipes.com/recipes/148/breakfast-and-brunch/eggs/
https://www.allrecipes.com/recipes/149/breakfast-and-brunch/french-toast/
https://www.allrecipes.com/recipes/1564/breakfast-and-brunch/eggs/frittata/
https://www.allrecipes.com/recipes/1311/breakfast-and-brunch/cereals/granola/
```



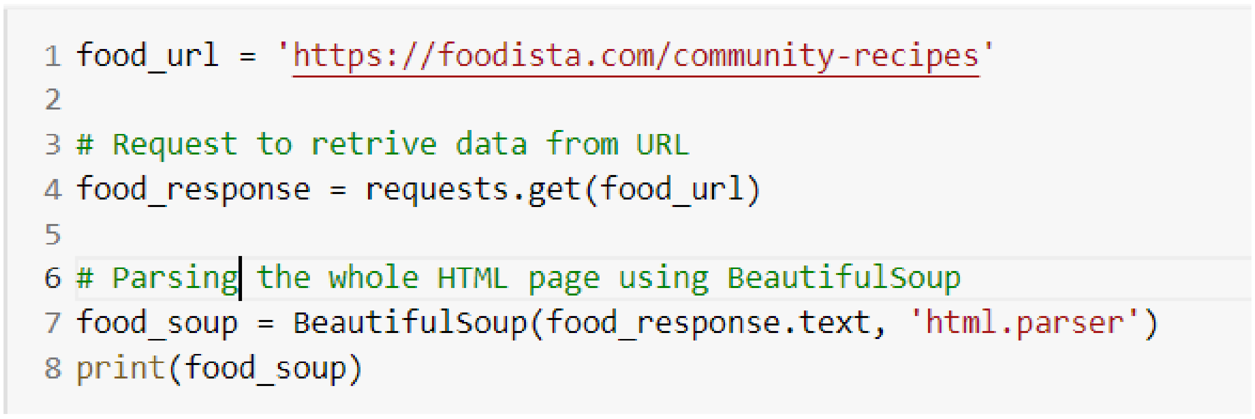
```
api = sp.API("47c2d9cd0eb04102b7f149ed7df64b84")

```



```

for i in range(0,len(DISH)):
    response = api.extract_recipe_from_website(str(DISH.pop(0)))
    data = response.json()
    print(data)
    if data=={'status': 'failure', 'code': 402, 'message': 'Your d
        break
    json.dump(data,file)
    file.write("\n")
```

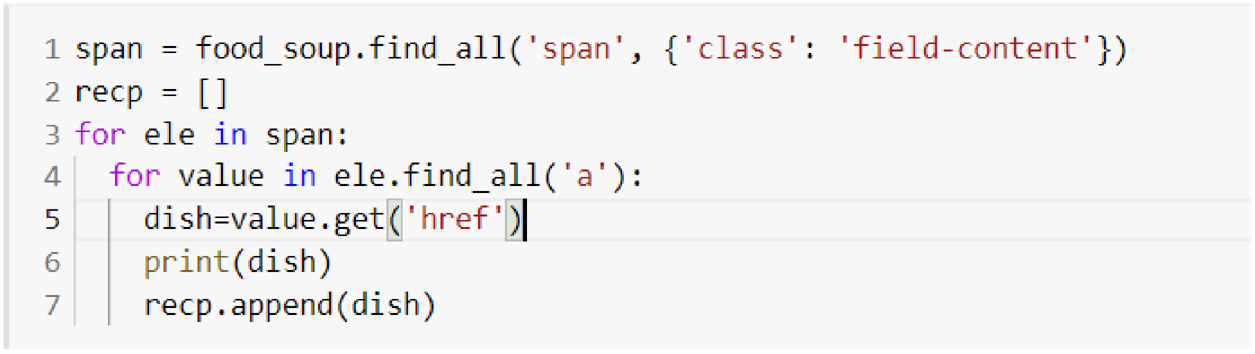
Foodista.com :

The whole HTML page data is scraped with the below code:



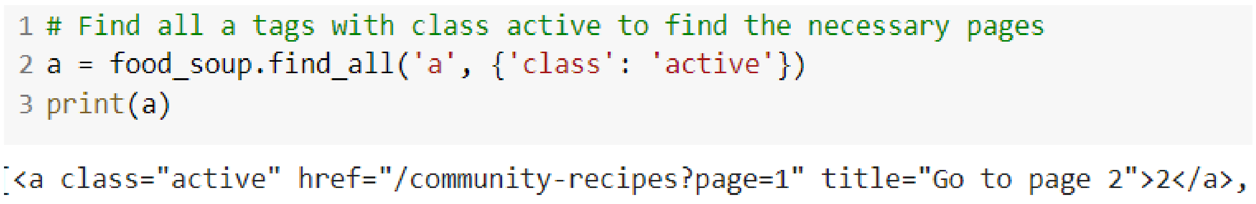
```python
food_url = 'https://foodista.com/community-recipes'

# Request to retrive data from URL
food_response = requests.get(food_url)

# Parsing the whole HTML page using BeautifulSoup
food_soup = BeautifulSoup(food_response.text, 'html.parser')
print(food_soup)
```

There were several pages of recipes on this website. To get each recipe from a page, the below snippet of code is used:



```python
span = food_soup.find_all('span', {'class': 'field-content'})
recp = []
for ele in span:
    for value in ele.find_all('a'):
        dish=value.get('href')
        print(dish)
        recp.append(dish)
```

The output will have the relative path of all recipes in that page. To get the available page links, below code is used:

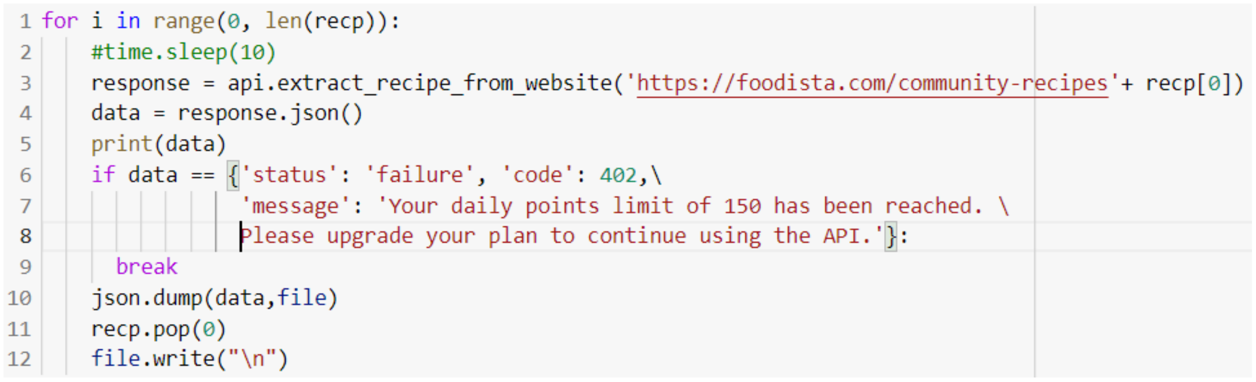


```python
# Find all a tags with class active to find the necessary pages
a = food_soup.find_all('a', {'class': 'active'})
print(a)
```

```
[<a class="active" href="/community-recipes?page=1" title="Go to page 2">2</a>,
```

The output consists of a tag along with href containing the link to the pages available. From the output, the href link is taken so that we can scrape data from those available pages.

The recipe links are given to Spoonacular API “Extract Recipe from Website” so as to get the necessary recipe information in the form of json.

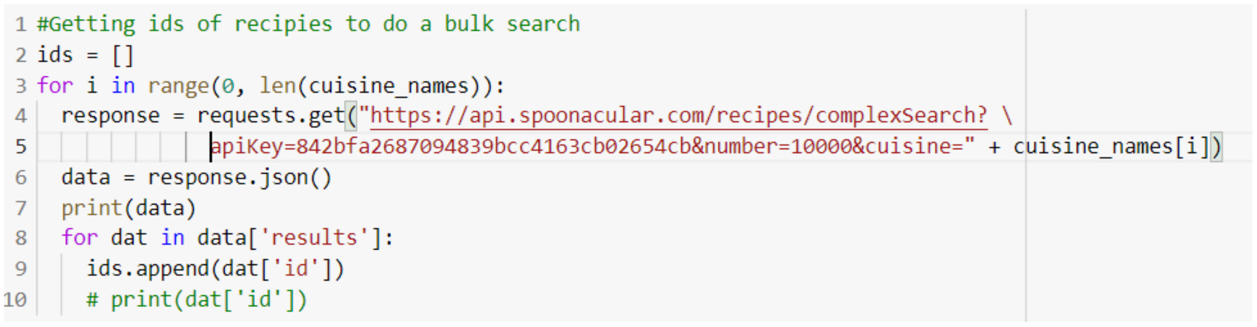


```python
for i in range(0, len(recp)):
    #time.sleep(10)
    response = api.extract_recipe_from_website('https://foodista.com/community-recipes'+ recp[0])
    data = response.json()
    print(data)
    if data == {'status': 'failure', 'code': 402,\
                'message': 'Your daily points limit of 150 has been reached. \
                Please upgrade your plan to continue using the API.'}:
        break
    json.dump(data,file)
    recp.pop(0)
    file.write("\n")
```

Spoonacular API:

There are several cuisines under which all the recipes reside. Some cuisine names are Indian, African, American and Mexican. Using these cuisine names, the recipes are scraped through the API provided by Spoonacular. There are two different APIs used for this purpose.

- Search API - The parameter includes cuisine name and returns all the recipes belonging to that cuisine.

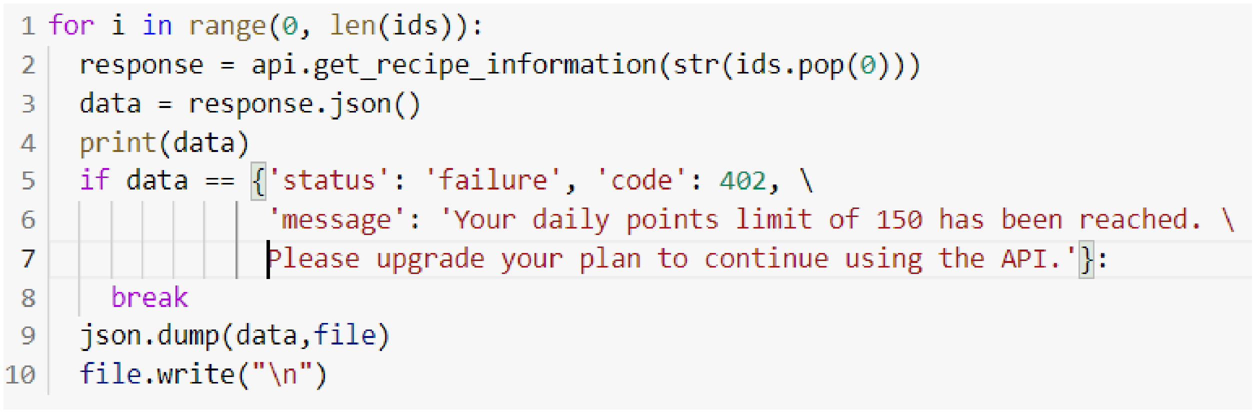


```python
#Getting ids of recipies to do a bulk search
ids = []
for i in range(0, len(cuisine_names)):
  response = requests.get("https://api.spoonacular.com/recipes/complexSearch? \
                          apiKey=842bfa2687094839bcc4163cb02654cb&number=10000&cuisine=" + cuisine_names[i])
  data = response.json()
  print(data)
  for dat in data['results']:
    ids.append(dat['id'])
    # print(dat['id'])
```

The output of this API will have the list of recipe IDs along with their title, source URL, image and image type for every cuisine. The Ids of these recipes will be given to the next API to get the particular recipe.

- Extract Recipe by ID - Extracts recipe with the given ID.

The recipe IDs are taken from the output of Search API of a particular cuisine and the whole recipe is taken with the other API.

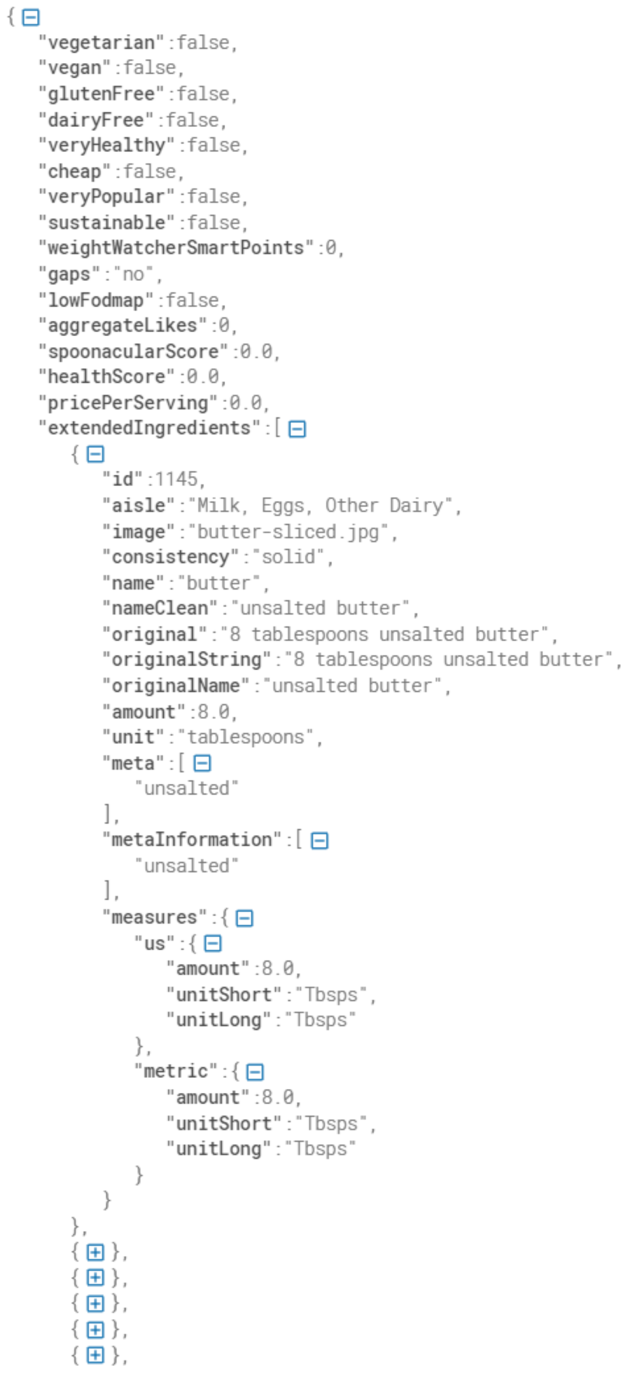


```python
for i in range(0, len(ids)):
  response = api.get_recipe_information(str(ids.pop(0)))
  data = response.json()
  print(data)
  if data == {'status': 'failure', 'code': 402, \
              'message': 'Your daily points limit of 150 has been reached. \
              Please upgrade your plan to continue using the API.'}:
    break
  json.dump(data,file)
  file.write("\n")
```

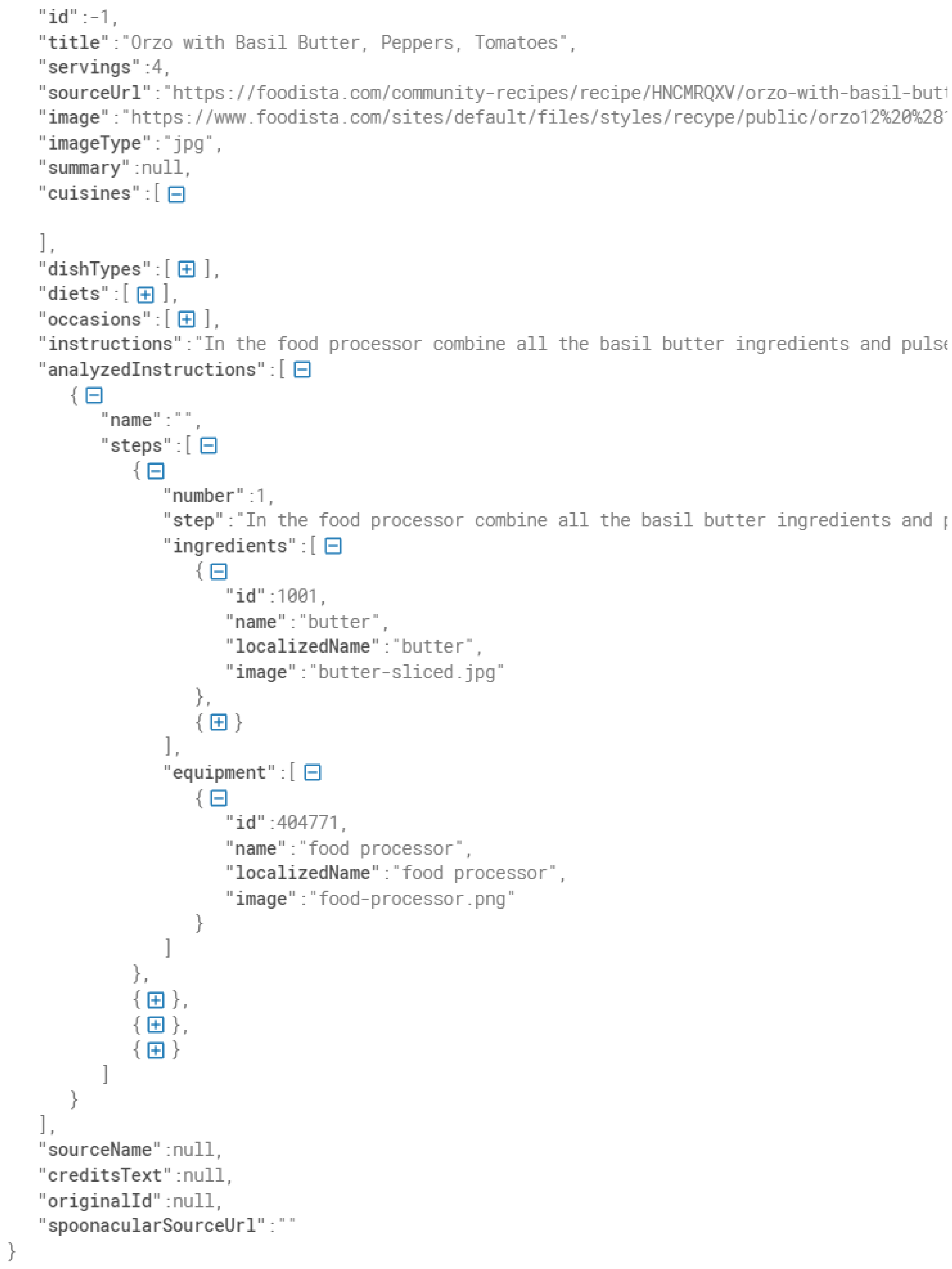
Similarly cookpad website is also scraped,The recipes are then written to the output file to get processed.

Preprocessing

An image of the recipe before preprocessing is shown below(raw data).



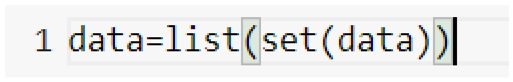
```
{ ⊟
   "vegetarian":false,
   "vegan":false,
   "glutenFree":false,
   "dairyFree":false,
   "veryHealthy":false,
   "cheap":false,
   "veryPopular":false,
   "sustainable":false,
   "weightWatcherSmartPoints":0,
   "gaps":"no",
   "lowFodmap":false,
   "aggregateLikes":0,
   "spoonacularScore":0.0,
   "healthScore":0.0,
   "pricePerServing":0.0,
   "extendedIngredients":[ ⊟
      { ⊟
         "id":1145,
         "aisle":"Milk, Eggs, Other Dairy",
         "image":"butter-sliced.jpg",
         "consistency":"solid",
         "name":"butter",
         "nameClean":"unsalted butter",
         "original":"8 tablespoons unsalted butter",
         "originalString":"8 tablespoons unsalted butter",
         "originalName":"unsalted butter",
         "amount":8.0,
         "unit":"tablespoons",
         "meta":[ ⊟
            "unsalted"
         ],
         "metaInformation":[ ⊟
            "unsalted"
         ],
         "measures":{ ⊟
            "us":{ ⊟
               "amount":8.0,
               "unitShort":"Tbsps",
               "unitLong":"Tbsps"
            },
            "metric":{ ⊟
               "amount":8.0,
               "unitShort":"Tbsps",
               "unitLong":"Tbsps"
            }
         }
      },
      { ⊞ },
      { ⊞ },
      { ⊞ },
      { ⊞ },
      { ⊞ },
```



```
    "id":-1,
    "title":"Orzo with Basil Butter, Peppers, Tomatoes",
    "servings":4,
    "sourceUrl":"https://foodista.com/community-recipes/recipe/HNCMRQXV/orzo-with-basil-butt
    "image":"https://www.foodista.com/sites/default/files/styles/recype/public/orzo12%20%28
    "imageType":"jpg",
    "summary":null,
    "cuisines":[ ⊟

    ],
    "dishTypes":[ ⊞ ],
    "diets":[ ⊞ ],
    "occasions":[ ⊞ ],
    "instructions":"In the food processor combine all the basil butter ingredients and pulse
    "analyzedInstructions":[ ⊟
       { ⊟
          "name":"",
          "steps":[ ⊟
             { ⊟
                "number":1,
                "step":"In the food processor combine all the basil butter ingredients and 
                "ingredients":[ ⊟
                   { ⊟
                      "id":1001,
                      "name":"butter",
                      "localizedName":"butter",
                      "image":"butter-sliced.jpg"
                   },
                   { ⊞ }
                ],
                "equipment":[ ⊟
                   { ⊟
                      "id":404771,
                      "name":"food processor",
                      "localizedName":"food processor",
                      "image":"food-processor.png"
                   }
                ]
             },
             { ⊞ },
             { ⊞ },
             { ⊞ }
          ]
       }
    ],
    "sourceName":null,
    "creditsText":null,
    "originalId":null,
    "spoonacularSourceUrl":""
}
```

Preprocessing is done in two main stages.

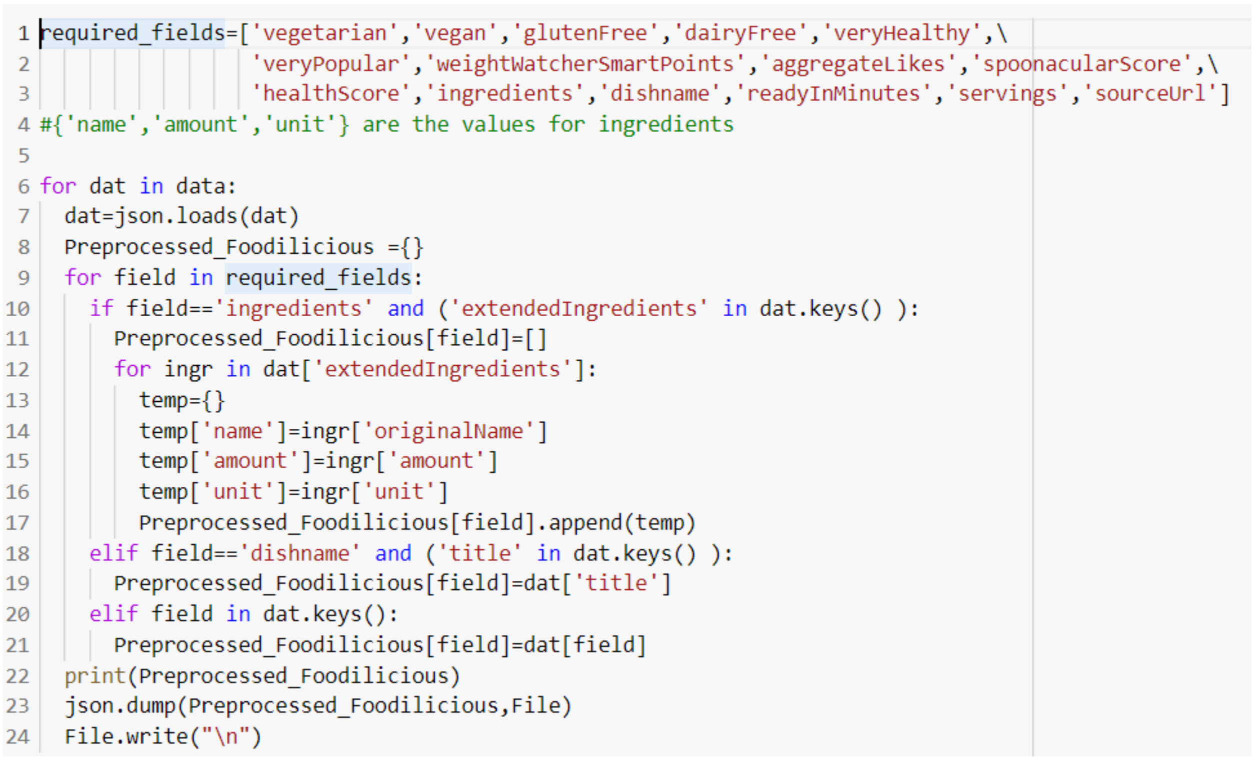
- In the first stage duplicate data in the same website is removed.



```
data=list(set(data))
```

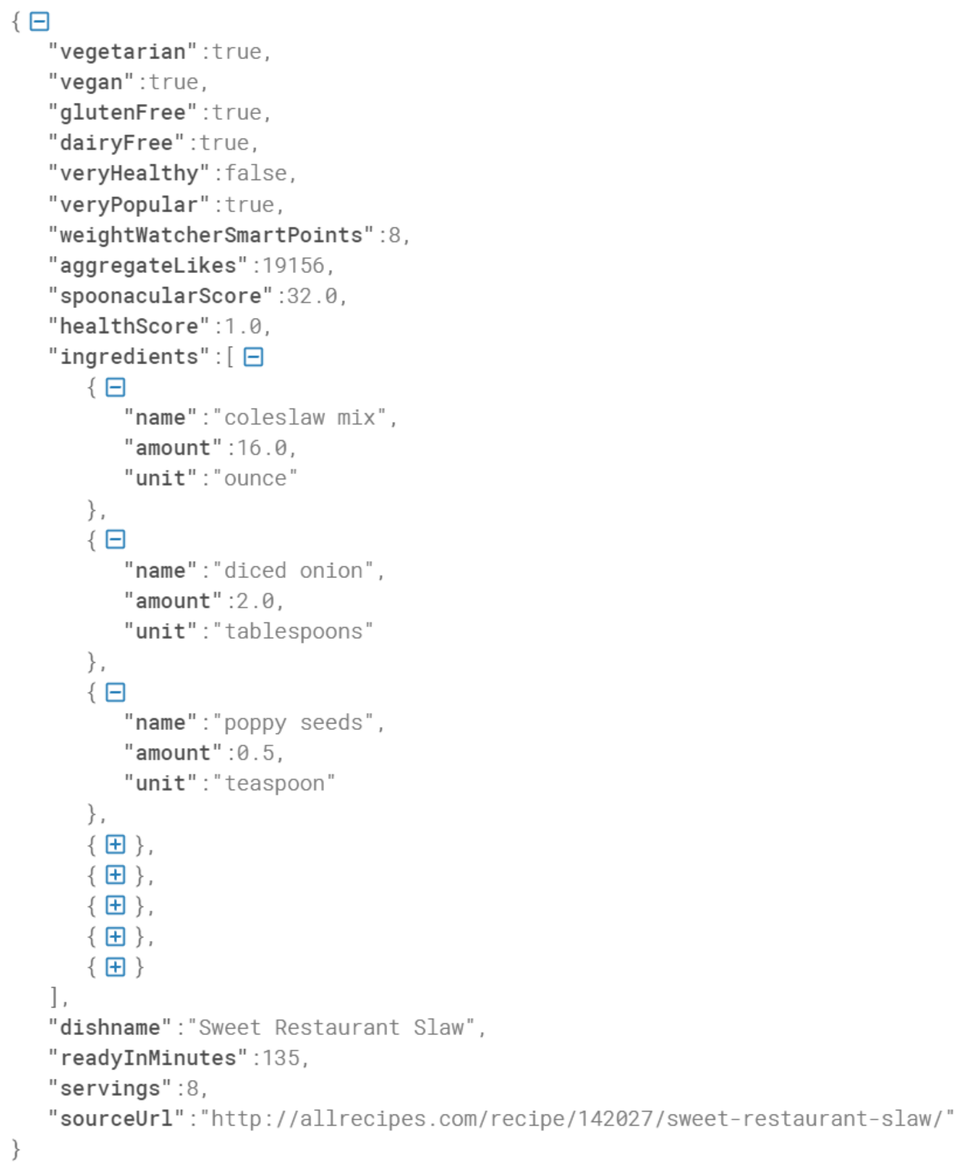
The variable data will have the data from the output file of the previous stage. Since it is a list, to remove duplicates, set functionality is used.

- In the second stage, only a few terms are picked for categorization such as ingredients list including the amount and unit, recipe name, vegetarian, gluten free, serving, cooking time, health score, aggregate likes, recipe site.
- As the final step, the incomplete recipes(those without title or ingredients) are not taken for the next stage. An image of the recipe after preprocessing is shown below.



```
required_fields=['vegetarian','vegan','glutenFree','dairyFree','veryHealthy',\
                 'veryPopular','weightWatcherSmartPoints','aggregateLikes','spoonacularScore',\
                 'healthScore','ingredients','dishname','readyInMinutes','servings','sourceUrl']
#{'name','amount','unit'} are the values for ingredients

for dat in data:
    dat=json.loads(dat)
    Preprocessed_Foodilicious ={}
    for field in required_fields:
        if field=='ingredients' and ('extendedIngredients' in dat.keys() ):
            Preprocessed_Foodilicious[field]=[]
            for ingr in dat['extendedIngredients']:
                temp={}
                temp['name']=ingr['originalName']
                temp['amount']=ingr['amount']
                temp['unit']=ingr['unit']
                Preprocessed_Foodilicious[field].append(temp)
        elif field=='dishname' and ('title' in dat.keys() ):
            Preprocessed_Foodilicious[field]=dat['title']
        elif field in dat.keys():
            Preprocessed_Foodilicious[field]=dat[field]
    print(Preprocessed_Foodilicious)
    json.dump(Preprocessed_Foodilicious,File)
    File.write("\n")
```

The actual data in json format will have the ingredients in the extendedIngredients field. The necessary ingredients data like name, amount and unit(Eg: olive oil, tbsp and 4) are fetched from that field; rest are discarded. The preprocessed recipe looks like the below snippet.



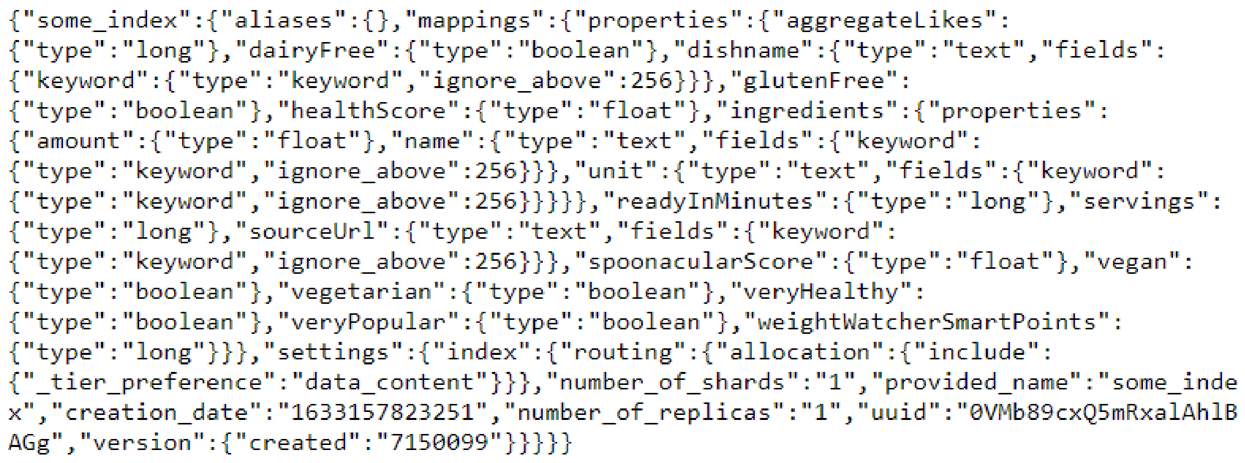
```
{
   "vegetarian":true,
   "vegan":true,
   "glutenFree":true,
   "dairyFree":true,
   "veryHealthy":false,
   "veryPopular":true,
   "weightWatcherSmartPoints":8,
   "aggregateLikes":19156,
   "spoonacularScore":32.0,
   "healthScore":1.0,
   "ingredients":[
      {
         "name":"coleslaw mix",
         "amount":16.0,
         "unit":"ounce"
      },
      {
         "name":"diced onion",
         "amount":2.0,
         "unit":"tablespoons"
      },
      {
         "name":"poppy seeds",
         "amount":0.5,
         "unit":"teaspoon"
      },
      { },
      { },
      { },
      { },
      { }
   ],
   "dishname":"Sweet Restaurant Slaw",
   "readyInMinutes":135,
   "servings":8,
   "sourceUrl":"http://allrecipes.com/recipe/142027/sweet-restaurant-slaw/"
}
```

Indexing

For Indexing elastic search is used. Elasticsearch is a distributed, free and open source and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. Elasticsearch is built on

Apache Lucene. It is known for its simple REST APIs, distributed nature, speed, and scalability.

Elastic search server is started which runs at the port 9200 of localhost. Elasticsearch client library for python is used to do indexing with elastic search through python. The input files are fed as a json list through the helper function which indexes each of the given json(single recipe) separately and stores them. We can view the indexes of the file through http://127.0.0.1:9200/_all.



```
{"some_index":{"aliases":{},"mappings":{"properties":{"aggregateLikes":
{"type":"long"},"dairyFree":{"type":"boolean"},"dishname":{"type":"text","fields":
{"keyword":{"type":"keyword","ignore_above":256}}},"glutenFree":
{"type":"boolean"},"healthScore":{"type":"float"},"ingredients":{"properties":
{"amount":{"type":"float"},"name":{"type":"text","fields":{"keyword":
{"type":"keyword","ignore_above":256}}},"unit":{"type":"text","fields":{"keyword":
{"type":"keyword","ignore_above":256}}}}},"readyInMinutes":{"type":"long"},"servings":
{"type":"long"},"sourceUrl":{"type":"text","fields":{"keyword":
{"type":"keyword","ignore_above":256}}},"spoonacularScore":{"type":"float"},"vegan":
{"type":"boolean"},"vegetarian":{"type":"boolean"},"veryHealthy":
{"type":"boolean"},"veryPopular":{"type":"boolean"},"weightWatcherSmartPoints":
{"type":"long"}}},"settings":{"index":{"routing":{"allocation":{"include":
{"_tier_preference":"data_content"}}},"number_of_shards":"1","provided_name":"some_inde
x","creation_date":"1633157823251","number_of_replicas":"1","uuid":"0VMb89cxQ5mRxalAhlB
AGg","version":{"created":"7150099"}}}}}
```

Github link:

The entire code of phase one is present in the following git repository: Foodilicious