

# RecipeFinder

Software code submission with documentation

by

Team "TasteBuddies"

Vijayaragavan Selvaraj (VS27) - Team Leader

Sathyanarayanan Gokarnesan (SG53)

Karthika Gopalakrishnan (KG24)

## 1. Application – Live Demo link

Please see the live demo of the “RecipeFinder” application running at the following location.

<http://54.81.52.188/>

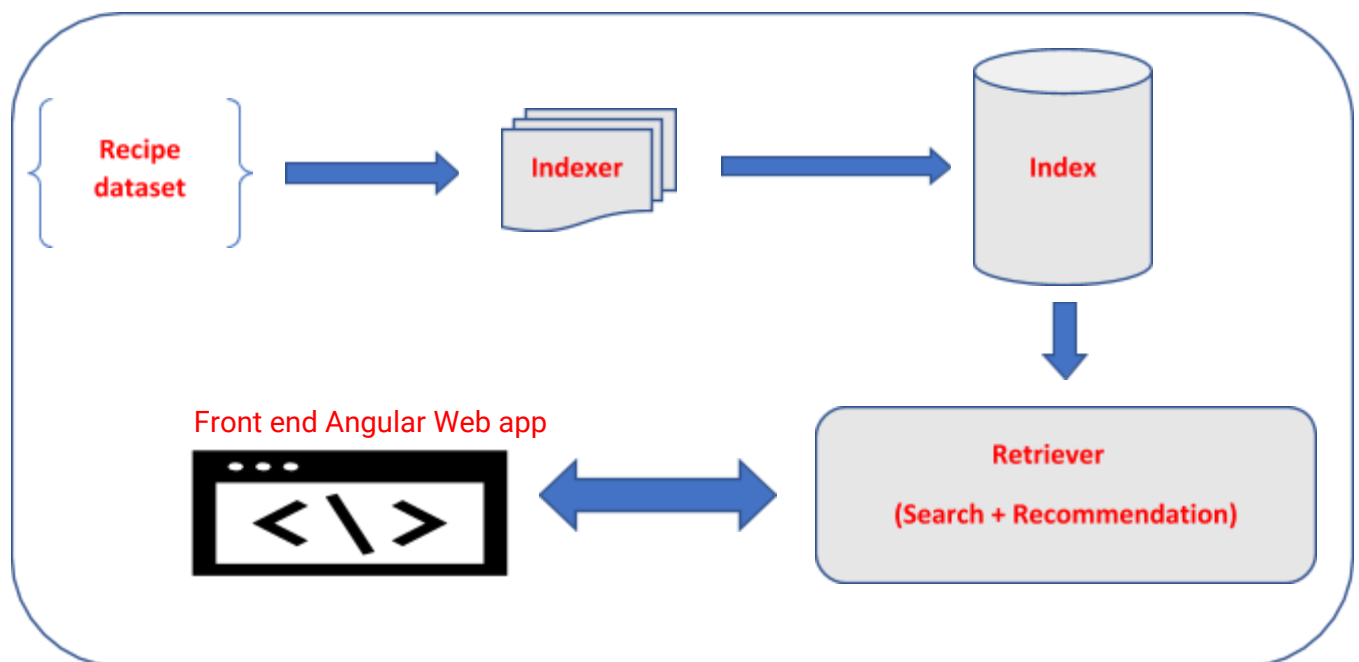
## 2. Overview

“RecipeFinder” is a web-based tool built using novel information retrieval techniques to search popular recipes.

It searches recipes based on the ingredients of the recipes. When the user searches with an ingredient name, it lists the top 20 recipes that contain the given ingredient. For each of the recipes in the search results, users can also see recipe details including the cooking directions, other ingredients of the recipe, nutritional values, timings, and ratings of the recipe. Users are also recommended with 4 similar recipes based on the nutritional value. Content based similarity approach has been used for this project.

## 3. Architecture and Source Code

Following is the high-level architecture of “RecipeFinder”.



Source code for the “Recipe Finder” application is present in the following github location:  
<https://github.com/vs27-illinois/CourseProject.git>

Source code for our project can be broadly categorized into three sections.

- ❖ BackEnd
- ❖ FrontEnd
- ❖ Infrastructure

#### Backend:

Following are the main files in the backend system.

##### 1. Indexer.py

This python file is responsible for taking the “Recipe Finder” dataset, parsing the csv file for different fields, creating indexes on different fields, and writing the same in an index document. We used Apache PyLucene to index the dataset.

We used the dataset from Kaggle at the following location for our project:  
[https://www.kaggle.com/elisaxygao/foodrecsysv1?select=raw-data\\_recipe.csv](https://www.kaggle.com/elisaxygao/foodrecsysv1?select=raw-data_recipe.csv)

Following are the columns in the index.

S.No	Column	Index Type
1	id	IndexOptions.DOCS
2	name	IndexOptions.NONE
3	image	IndexOptions.NONE
4	avg_rating	IndexOptions.NONE
5	total_reviews	IndexOptions.NONE
6	ingredients	IndexOptions.DOCS_AND_FREQS_AND_POSITIONS
7	time_taken	IndexOptions.NONE
8	nutrition	IndexOptions.NONE
9	calories	IndexOptions.NONE
10	carbohydrates	IndexOptions.NONE

11	protein	IndexOptions.NONE
12	fat	IndexOptions.NONE

We used MMapDirectory to load the index files and indexed the ingredients and recipe id field by using EnglishAnalyzer since the dataset is in English. We stored the rest of the fields in the indexed document. Following are some of the main code snippets of indexer.py.

```
def index_data():
    t1 = get_field_type()
    t1.setIndexOptions(IndexOptions.NONE)

    t2 = get_field_type()
    t2.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS)

    t3 = get_numeric_field_type()
    t3.setIndexOptions(IndexOptions.DOCS)

    t4 = get_numeric_field_type()
    t4.setIndexOptions(IndexOptions.NONE)

    index = 0
    for df in pd.read_csv('dataset/recipe.csv', chunksize=3000, iterator=True):
        df.drop("reviews", axis=1, inplace=True)

        index += 1
        mm_dir = MMapDirectory(Paths.get('index'))
        writer = IndexWriter(mm_dir, IndexWriterConfig(EnglishAnalyzer()))
        print(f"Opening index {index} with {writer.numRamDocs()} docs...")

        for num, row in df.iterrows():
            print(f"Indexing row {num+1}...")

            doc = Document()
            doc.add(Field("id", row["recipe_id"], t3))
            doc.add(Field("name", row["recipe_name"], t1))
            doc.add(Field("image", row["image_url"], t1))
            doc.add(Field("avg_rating", row["aver_rate"], t4))
            doc.add(Field("total_reviews", row["review_nums"], t4))
```

## 2. Retriever.py

Following are the functionalities of retriever.py.

- It is responsible for taking the “ingredient” input from the user, searching the index for 20 popular recipes based on the ingredient, converting the results (list of recipes) into json format to be rendered in the UI.
- It is responsible for taking the recipe id from the user to provide a detailed view of the recipe.
- It also returns the list of top 4 recipes based on similar nutritional value for the recipe which the user wants to see the details.

Following are the API calls involved.

S.No	End Point	Method	Output
1	<p><code>http://&lt;ipaddress&gt;/recipe/search/{ingredient}</code></p> <p>Ex:</p> <p><code>http://54.81.52.188/recipe/search/chocolate</code></p>	GET	<pre>[ {   "avg_rating": 4.34615373611,   "calories": 274.0809,   "carbohydrates": 45.78727,   "fat": 10.5331,   "id": 32482,   "image":   "https://images.media-allrecipes.com/userphotos/   250x250/710487.jpg",   "ingredients": [     "&lt;strong&gt;chocolate&lt;/strong&gt; chips",     "powdered &lt;strong&gt;chocolate&lt;/strong&gt; drink     mix",     "&lt;strong&gt;chocolate&lt;/strong&gt; syrup",     "scoops &lt;strong&gt;chocolate&lt;/strong&gt; ice cream",     "milk",     "ice"   ],   "name": "Chocolate Surprise Milkshake",   "protein": 3.052425,   "total_reviews": 17 },... ]</pre>
2	<p><code>http://&lt;ipaddress&gt;/recipe/details/{recipeId}</code></p> <p>Ex:</p> <p><code>http://54.81.52.188/recipe/details/220725</code></p>	GET	<pre>{   "avg_rating": 4.26016616821,   "calories": 123.5964,   "carbohydrates": 19.74722,   "directions": [     "Prepare the cake mix according to package     directions using any of the recommended pan     sizes. When cake is done, crumble while warm into     a large bowl, and stir in the frosting until well     blended.",     "Melt chocolate coating in a glass bowl in the     microwave, or in a metal bowl over a pan of     simmering water, stirring occasionally until     smooth.",     "Use a melon baller or small scoop to form balls     of the chocolate cake mixture. Dip the balls in</pre>

			<p>chocolate using a toothpick or fork to hold them. Place on waxed paper to set."</p> <p>],</p> <p>"fat": 5.188236,</p> <p>"id": 67656,</p> <p>"image":</p> <p>"http://images.media-allrecipes.com/userphotos/720x405/599097.jpg",</p> <p>"ingredients": [</p> <p>"chocolate cake mix",</p> <p>"prepared chocolate frosting",</p> <p>"bar chocolate flavored confectioners coating"</p> <p>],</p> <p>"name": "Cake Balls",</p> <p>"nutrition": {</p> <p>"calcium": {</p> <p>"amount": 27.23583,</p> <p>"displayValue": "27",</p> <p>"hasCompleteData": true,</p> <p>"name": "Calcium",</p> <p>"percentDailyValue": "3",</p> <p>"unit": "mg"</p> <p>},</p> <p>"calories": {</p> <p>"amount": 123.5964,</p> <p>"displayValue": "124",</p> <p>"hasCompleteData": true,</p> <p>"name": "Calories",</p> <p>"percentDailyValue": "6",</p> <p>"unit": "kcal"</p> <p>},</p> <p>},</p> <p>"protein": 1.122792,</p> <p>"time_taken": [</p> <p>"Prep",</p> <p>"40 m",</p> <p>"Cook",</p> <p>"30 m",</p> <p>"Ready In",</p> <p>"3 h 10 m"</p> <p>],</p> <p>"total_reviews": 1867</p> <p>}</p>
3	<p>http://&lt;ipaddress&gt;/recipe/recommend/{recipeId}</p> <p>Ex:</p> <p>http://54.81.52.188/recipe/recommend/220725</p>	GET	<p>[</p> <p>{</p> <p>"avg_rating": 3.5,</p> <p>"calories": 124.4117,</p> <p>"carbohydrates": 19.98162,</p> <p>"fat": 4.989575,</p> <p>"id": 15463,</p> <p>"image":</p> <p>"http://images.media-allrecipes.com/userphotos/720x405/1115423.jpg",</p> <p>"ingredients": [</p> <p>"canola oil",</p> <p>"honey",</p> <p>"packed brown sugar",</p> <p>"egg whites",</p> <p>"vanilla extract",</p> <p>"water",</p> <p>"wheat flour",</p> <p>"all-purpose flour",</p> <p>"baking powder",</p> <p>"salt",</p> <p>"ground cinnamon",</p> <p>"semisweet chocolate chips"</p>

			<pre> }, "name": "No Cholesterol Chocolate Chip", "protein": 1.515088, "total_reviews": 15 }, </pre>
--	--	--	--

We maintained the IndexSearcher object as a global variable to boost the performance of the search. We used Lucene Highlighter to highlight the search terms in the search results in the website. We maintained a global variable of pandas dataframe that is the L2 normalized form of nutritional values that we used in the recommendation service. The recommendation is done based on the nutritional values of calories and macronutrients (i.e., protein, carbohydrates and fat). Following are some of the main code snippets for retriever.py.

```

# Initializes the Flask App and Lucene Searcher
app = Flask(__name__)
vm = lucene.initVM()
mmDir = MMapDirectory(Paths.get('index'))
searcher = IndexSearcher(DirectoryReader.open(mmDir))

def convert_to_list(doc, key, highlight=False, query=None):
    if key == "ingredients" and highlight is True and query is not None:
        analyzer = EnglishAnalyzer()
        hl = Highlighter(SimpleHTMLFormatter('<strong>', '</strong>'), QueryScorer(query))

        values = []
        for text in doc.getValues(key):
            ts = analyzer.tokenStream("ingredients", StringReader(text))
            value = hl.getBestFragment(ts, text)
            if value is None:
                values.append(text)
            else:
                values.insert(0, value)

        return values
    else:
        return [value for value in doc.getValues(key)]

```

```

def get_all_recipes():
    hits = searcher.search(MatchAllDocsQuery(), 50000)

    recipe_list = {}
    recipes = []
    for hit in hits.scoreDocs:
        doc = searcher.doc(hit.doc)

        recipe = convert_to_json(doc)
        recipe_id = recipe['id']
        recipe_list[recipe_id] = recipe

        new_recipe = {
            'id': recipe['id'],
            'calories': recipe['calories'],
            'protein': recipe['protein'],
            'carbohydrates': recipe['carbohydrates'],
            'fat': recipe['fat']
        }
        recipes.append(new_recipe)

    df_pre = pd.DataFrame(recipes)
    df = df_pre.drop('id', axis=1)
    df.index = df_pre['id']
    df_norm = pd.DataFrame(normalize(df, axis=0))
    df_norm.columns = df.columns
    df_norm.index = df.index

    return recipe_list, df

# Initializes the Recipe List and Normalized Dataframe for the Recommendation Service
all_recipes, data_frame = get_all_recipes()

```

```

@app.route('/recipe/search/<ingredient>')
def get_recipes(ingredient):
    vm.attachCurrentThread()
    recipes = []
    ingredient = re.sub('[^a-zA-Z0-9 ]', '', ingredient).strip()
    if len(ingredient) > 0:
        query_parser = QueryParser("ingredients", EnglishAnalyzer())
        query_parser.setSplitOnWhitespace(True)
        query_parser.setAutoGeneratePhraseQueries(True)

        sort = Sort([SortField.FIELD_SCORE,
                     SortField("total_reviews", SortField.Type.FLOAT, True),
                     SortField("avg_rating", SortField.Type.FLOAT, True)])

        query = query_parser.parse(ingredient)
        hits = searcher.search(query, 20, sort)

        for hit in hits.scoreDocs:
            doc = searcher.doc(hit.doc)
            recipe = convert_to_json(doc, highlight=True, query=query)
            recipes.append(recipe)

    return jsonify(recipes)

```



```

@app.route('/recipe/details/<recipe_id>')
def get_recipe_details(recipe_id):
    vm.attachCurrentThread()
    query_parser = QueryParser("id", StandardAnalyzer())
    query = query_parser.parse(recipe_id)
    hits = searcher.search(query, 1)

    recipe = {}
    for hit in hits.scoreDocs:
        doc = searcher.doc(hit.doc)
        recipe = convert_to_json(doc)
        recipe['time_taken'] = convert_to_list(doc, "time_taken")
        recipe['directions'] = convert_to_list(doc, "directions")
        recipe['nutrition'] = json.loads(doc.get('nutrition'))

    return jsonify(recipe)

@app.route('/recipe/recommend/<recipe_id>')
def get_recommended_recipes(recipe_id):
    base_id = int(recipe_id)
    indices = pd.DataFrame(data_frame.index)
    indices = indices[indices.id != base_id]
    indices['distance'] = indices['id'].apply(lambda x: euclidean(data_frame.loc[base_id], data_frame.loc[x]))
    result = indices.sort_values(['distance']).head(4).sort_values(by=['distance', 'id'])

    recipes = []
    for index in result.id:
        recipes.append(all_recipes[index])

    return jsonify(recipes)

```

## Frontend:

The frontend of the code to display the search results and recommended recipes are based on Angular. It makes http calls to retrieve the search results, recipe details and recommended recipes. The frontend is a single page application and the code is located in the “src/app” folder.

## Infrastructure:

Following are the infrastructure related files.

### 1. DockerFile

We used Docker to containerize our application and used the following image as the base to install PyLucene: <https://hub.docker.com/r/coady/pylucene>. The Dockerfile in the project folder copies all the required backend and frontend files and deploys them in a standalone container.

### 2. startup.sh

This shell script file is responsible for creating the docker image (recipefinder:1.0) and running the image as a docker container.

## 4. Setup and Installation Instructions

Following are the technologies used in the project.

- Python 3.9.0
- Apache PyLucene 8.6.1
- Flask 1.1.2
- Angular 11.0
- Docker 2.5

Some of the python packages used are:

- Numpy 1.19.4
- Pandas 1.1.5
- scikit-learn 0.23.2
- sklearn
- scipy 1.5.4

Angular modules used:

- angular-material
- ng-bootstrap

Since we used Docker, the project can be installed in either a local environment or on any cloud instances. Following are the steps:

1. Install and set up Docker.
2. Clone the project from the github location <https://github.com/vs27-illinois/CourseProject.git>
3. Open the shell script and run the following command: `sh startup.sh`
4. If the application is running on a cloud environment, enable the http port (80) in the host machine, so that the Flask application running in the docker container can be exposed to the internet. Sample image below.

EC2 > Security Groups > sg-0d4d4ca9cf59cf6c - launch-wizard-5

### sg-0d4d4ca9cf59cf6c - launch-wizard-5 Actions ▾

**Details**

Security group name launch-wizard-5	Security group ID sg-0d4d4ca9cf59cf6c	Description launch-wizard-5 created 2020-12-03T20:08:07.845-05:00	VPC ID vpc-3fd21842
Owner 516179495762	Inbound rules count 2 Permission entries	Outbound rules count 1 Permission entry	

**Inbound rules** | Outbound rules | Tags

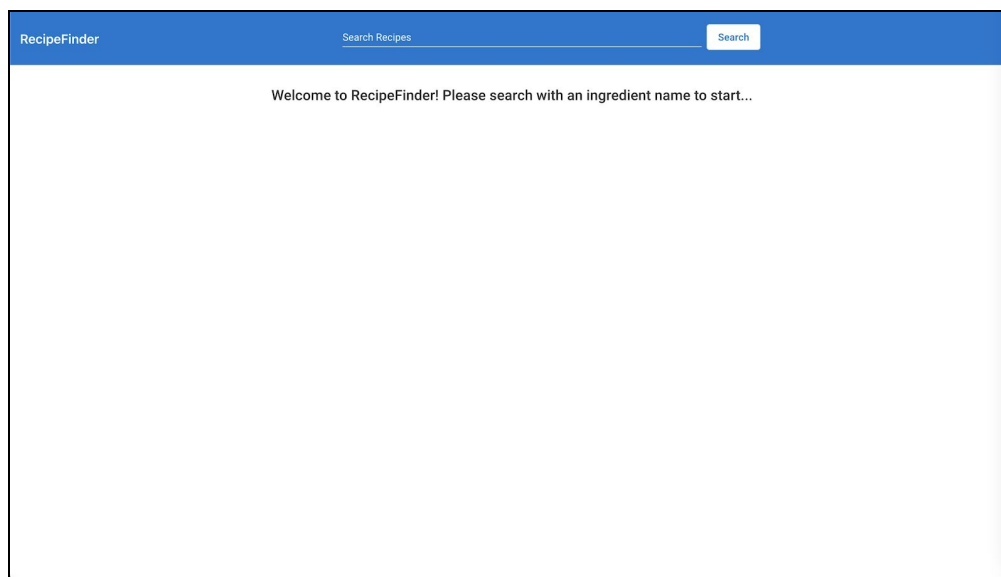
**Inbound rules** Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	0.0.0.0/0	http for flask
SSH	TCP	22	0.0.0.0/0	-

5. Open your favorite browser and go to `http://127.0.0.1/` (if the app is running in a local environment) or `http://<ipaddress>/` (if the app is running in a cloud environment).

## 5. Snapshots

Following are the snapshots of the “Recipe Finder” application.





RecipeFinder

Search Recipes  
chocolate

Search

- all chocolate flavored confections coating

### Directions

- Prepare the cake mix according to package directions using any of the recommended pan sizes. When cake is done, crumble while warm into a large bowl, and stir in the frosting until well blended.
- Melt chocolate coating in a glass bowl in the microwave, or in a metal bowl over a pan of simmering water, stirring occasionally until smooth.
- Use a melon baller or small scoop to form balls of the chocolate cake mixture. Dip the balls in chocolate using a toothpick or fork to hold them. Place on waxed paper to set.


### Nutrition

124 calories; protein 1g 2% DV; carbohydrates 20g 6% DV; fat 5g 8% DV.

### Recipes With Similar Nutritional Value

#### No Cholesterol Chocolate Chip


♥♥♥♥♥ 15 124 Calories



**Ingredients:** canola oil, honey, packed brown sugar, egg whites, vanilla extract, water, wheat flour, all-purpose flour, baking powder...

#### Esther's Christmas Cookies


♥♥♥♥♥ 2 123 Calories



**Ingredients:** butter, white sugar, eggs, vanilla extract, milk, candied cherries chopped, candied pineapple, chopped walnuts, golden...

#### Orange Cream Cookies


♥♥♥♥♥ 17 125 Calories



**Ingredients:** shortening, packed brown sugar, eggs, vanilla extract, all-purpose flour, orange flavored drink mix (e.g. Tang®), baking...

#### Peanut Butter Fudge II

♥♥♥♥♥ 15 123 Calories



**Ingredients:** white sugar, butter, evaporated milk, peanut butter chips, marshmallow creme, vanilla extract

## 6. Further Improvements

Within the given timeframe, we implemented all the functionalities that we have initially proposed for this project. We even fairly optimized the response time of the APIs by improving the performance of the search and recommendation service of the application. As a future enhancement, the performance of the recommendation service can be improved (currently it takes ~20 seconds to provide results). Moreover, we attempted to modify the recommendation service to use other fields like ingredients and faced memory limitations in our EC2 instance (we used free tier). It can also be tried as a further development of this project.

## 7. References

[https://lucene.apache.org/core/8\\_6\\_1/](https://lucene.apache.org/core/8_6_1/)

<https://docs.scipy.org/doc/scipy/reference/spatial.distance.html>

<https://www.kaggle.com/elisaxxygao/foodrecsysv1>

<https://github.com/coady/docker>

<https://material.angular.io/>

## 8. Contribution of Team Members

Vijayaragavan Selvaraj (VS27) - Team Leader

- Retriever (Search)
- Docker
- Angular

Sathyanarayanan Gokarnesan (SG53)

- Indexer
- EC2 setup
- Angular

Karthika Gopalakrishnan (KG24)

- Retriever (Recommendation service)
- Documentation
- Presentation