Search Engine for Recipe Dataset

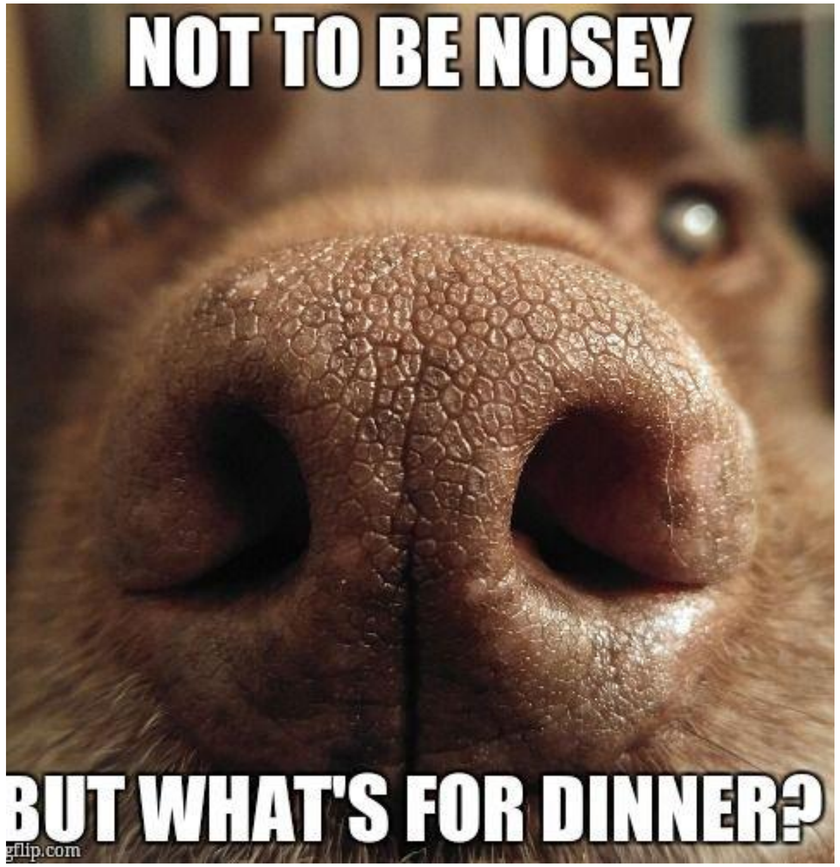
Team Members: Janet Dick, Jonilyn Dick, and Navodita Mathur

Goals and Importance

The goal for this project was to design a search engine for a large variety of recipes. Incorporated with the search engine is a user-friendly interface that allows users to execute their query and have recipes returned that can then be accessed.

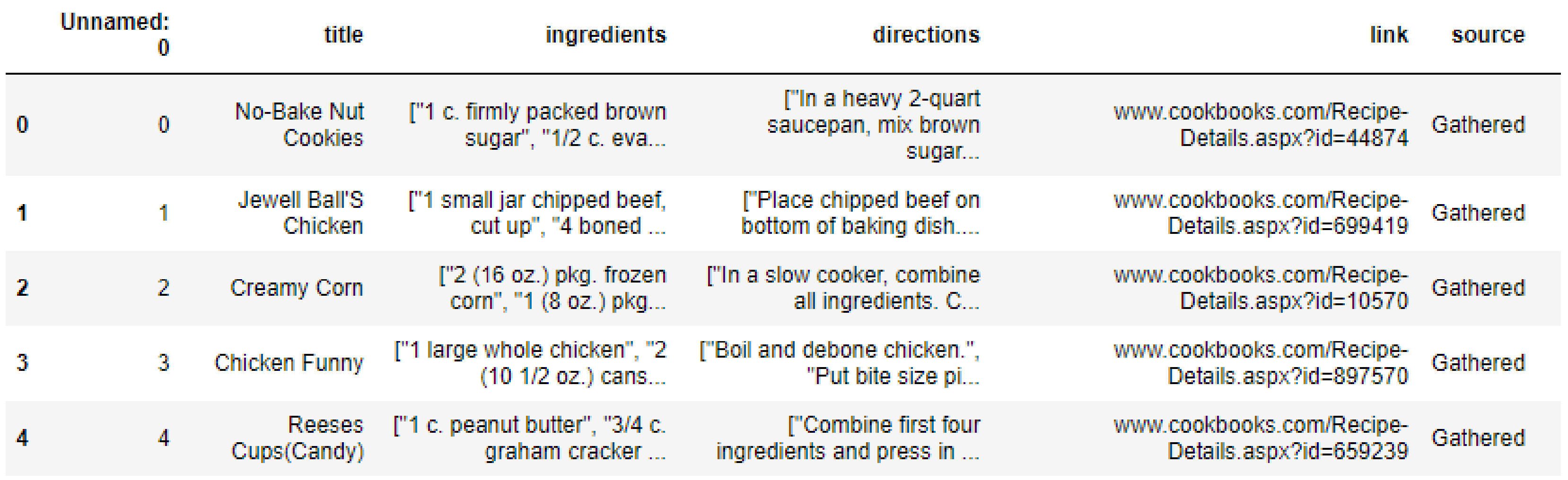
- The end users envisioned for this project are the aspiring home chefs looking for their next challenge in the kitchen, busy parents looking for fresh and quick meals to put on the table, and people that have made the switch from going out to staying in due to social climate.

- A recipe search engine is useful to a large and varied group of people, especially considering that the number of recipes that can exist is almost unlimited for as long as people have the access to explore new recipes and perhaps some inspiration. Without a search engine, looking for new recipes would be a time-consuming task that could possibly deter people from trying to cook at home!



Details of Approach – Initial Input

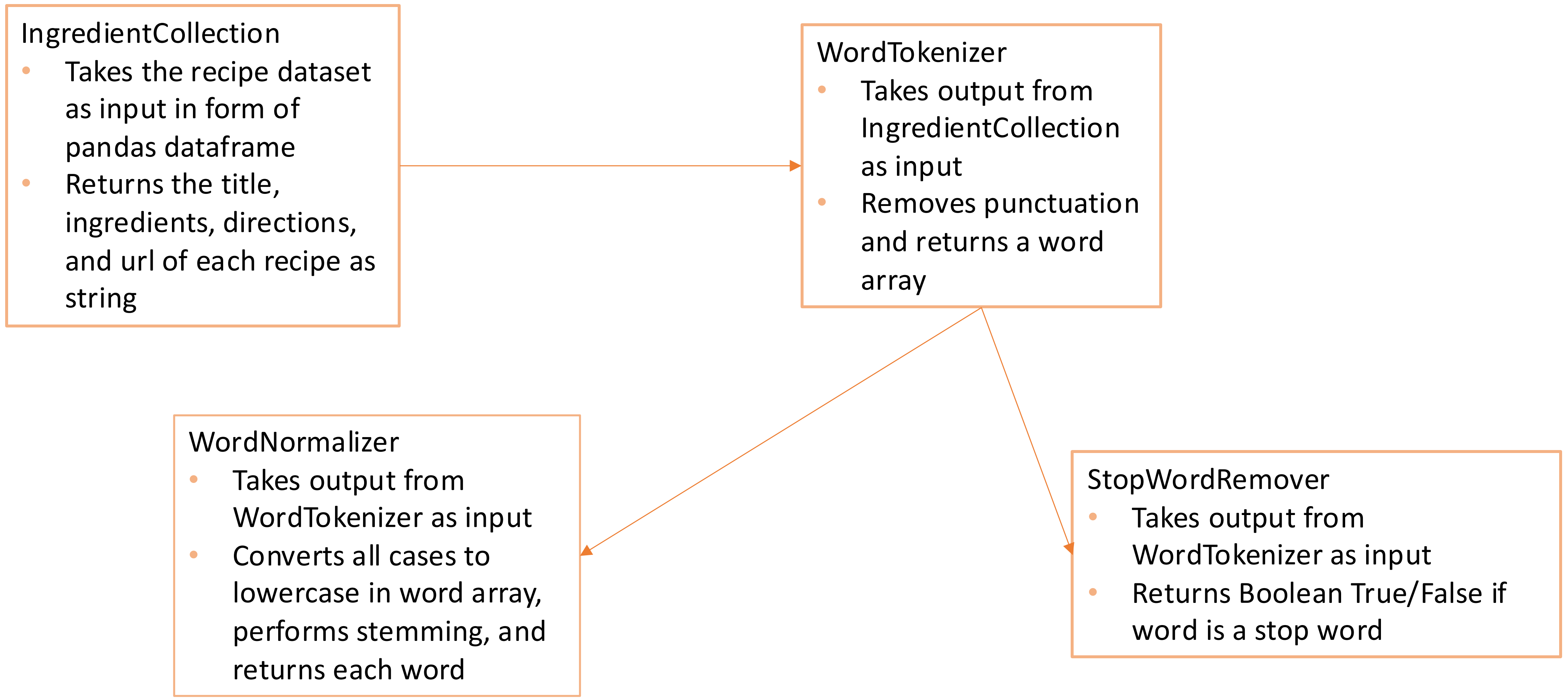
- The initial input for the search engine was sourced from Kaggle, it is a dataset containing over 2 million recipes gathered from multiple different sites across the web
  - Figure 1 is the first 5 rows of the dataset used



	Unnamed: 0	title	ingredients	directions	link	source
0	0	No-Bake Nut Cookies	["1 c. firmly packed brown sugar", "1/2 c. eva...	["In a heavy 2-quart saucepan, mix brown sugar...	www.cookbooks.com/Recipe-Details.aspx?id=44874	Gathered
1	1	Jewell Ball'S Chicken	["1 small jar chipped beef, cut up", "4 boned ...	["Place chipped beef on bottom of baking dish....	www.cookbooks.com/Recipe-Details.aspx?id=699419	Gathered
2	2	Creamy Corn	["2 (16 oz.) pkg. frozen corn", "1 (8 oz.) pkg...	["In a slow cooker, combine all ingredients. C...	www.cookbooks.com/Recipe-Details.aspx?id=10570	Gathered
3	3	Chicken Funny	["1 large whole chicken", "2 (10 1/2 oz.) cans...	["Boil and debone chicken.", "Put bite size pi...	www.cookbooks.com/Recipe-Details.aspx?id=897570	Gathered
4	4	Reeses Cups(Candy)	["1 c. peanut butter", "3/4 c. graham cracker ...	["Combine first four ingredients and press in ...	www.cookbooks.com/Recipe-Details.aspx?id=659239	Gathered

Figure 1: Dataset from Kaggle to be used for creating search engine - RecipeNLG (cooking recipes dataset) | Kaggle

Details of Approach – Processing Input



IngredientCollection

- Takes the recipe dataset as input in form of pandas dataframe
- Returns the title, ingredients, directions, and url of each recipe as string

WordTokenizer

- Takes output from IngredientCollection as input
- Removes punctuation and returns a word array

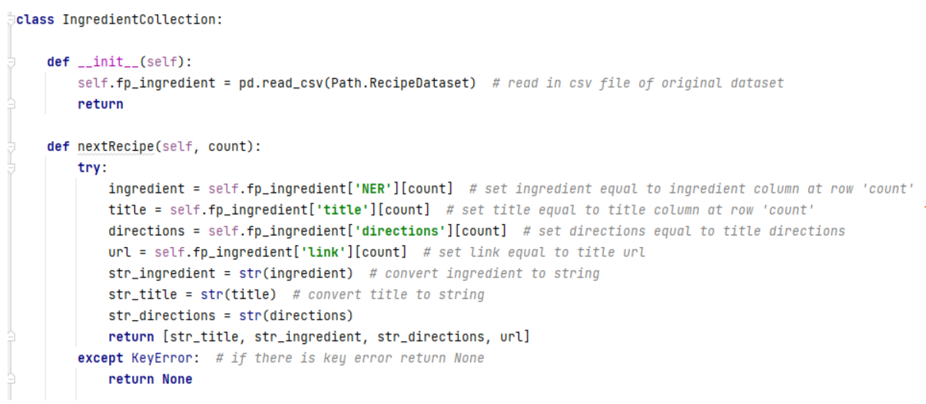
WordNormalizer

- Takes output from WordTokenizer as input
- Converts all cases to lowercase in word array, performs stemming, and returns each word

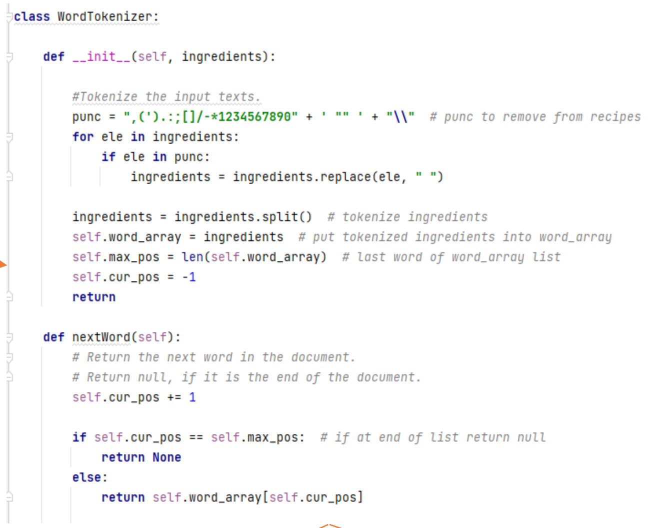
StopWordRemover

- Takes output from WordTokenizer as input
- Returns Boolean True/False if word is a stop word

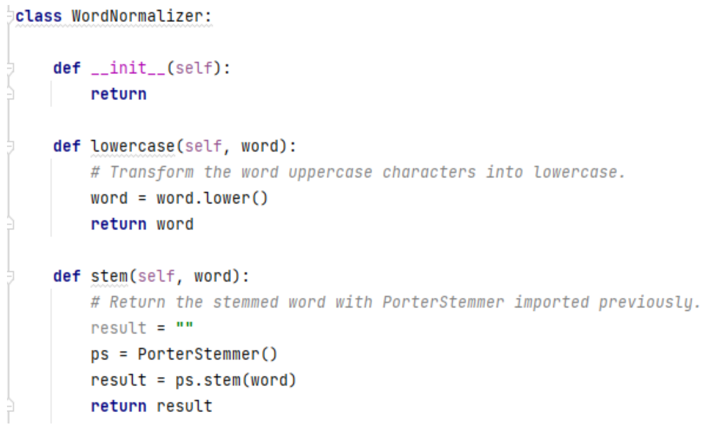
Details of Approach – Processing Input



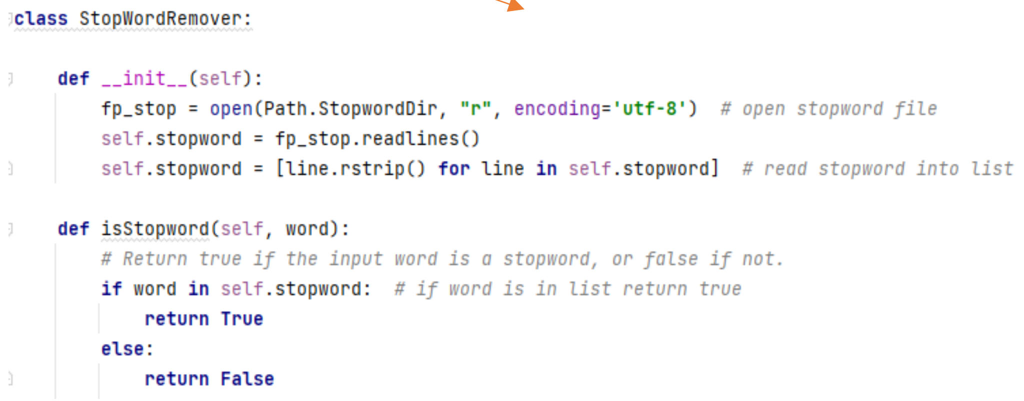
```python
class IngredientCollection:

    def __init__(self):
        self.fp_ingredient = pd.read_csv(Path.RecipeDataset)  # read in csv file of original dataset
        return

    def nextRecipe(self, count):
        try:
            ingredient = self.fp_ingredient['NER'][count]  # set ingredient equal to ingredient column at row 'count'
            title = self.fp_ingredient['title'][count]  # set title equal to title column at row 'count'
            directions = self.fp_ingredient['directions'][count]  # set directions equal to title directions
            url = self.fp_ingredient['link'][count]  # set link equal to title url
            str_ingredient = str(ingredient)  # convert ingredient to string
            str_title = str(title)  # convert title to string
            str_directions = str(directions)
            return [str_title, str_ingredient, str_directions, url]
        except KeyError:  # if there is key error return None
            return None
```



```python
class WordTokenizer:

    def __init__(self, ingredients):

        #Tokenize the input texts.
        punc = ",(').:;[]/-*1234567890" + ' "" ' + "\\"  # punc to remove from recipes
        for ele in ingredients:
            if ele in punc:
                ingredients = ingredients.replace(ele, " ")

        ingredients = ingredients.split()  # tokenize ingredients
        self.word_array = ingredients  # put tokenized ingredients into word_array
        self.max_pos = len(self.word_array)  # last word of word_array list
        self.cur_pos = -1
        return

    def nextWord(self):
        # Return the next word in the document.
        # Return null, if it is the end of the document.
        self.cur_pos += 1

        if self.cur_pos == self.max_pos:  # if at end of list return null
            return None
        else:
            return self.word_array[self.cur_pos]
```



```python
class WordNormalizer:

    def __init__(self):
        return

    def lowercase(self, word):
        # Transform the word uppercase characters into lowercase.
        word = word.lower()
        return word

    def stem(self, word):
        # Return the stemmed word with PorterStemmer imported previously.
        result = ""
        ps = PorterStemmer()
        result = ps.stem(word)
        return result
```



```python
class StopWordRemover:

    def __init__(self):
        fp_stop = open(Path.StopwordDir, "r", encoding='utf-8')  # open stopword file
        self.stopword = fp_stop.readlines()
        self.stopword = [line.rstrip() for line in self.stopword]  # read stopword into list

    def isStopword(self, word):
        # Return true if the input word is a stopword, or false if not.
        if word in self.stopword:  # if word is in list return true
            return True
        else:
            return False
```

Details of Approach – Processed Input

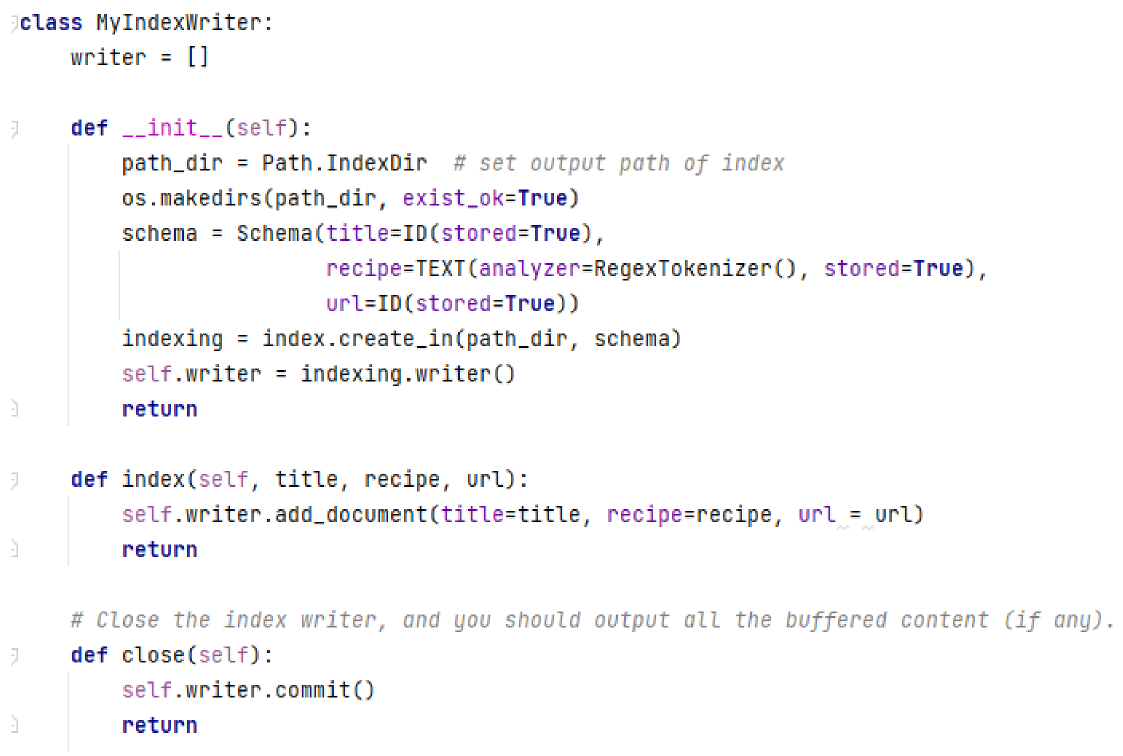
No-Bake Nut Cookies
bake nut cooki brown sugar milk vanilla nut butter bite size shred rice biscuit heavi quart saucepan mix brown sugar nut evapor milk butter margarin stir medium heat mixtur bubbl top boil stir minut heat stir vanilla cereal mix teaspoon drop shape cluster wax paper stand firm minut
www.cookbooks.com/Recipe-Details.aspx?id=44874
Jewell Ball'S Chicken
jewel ball chicken beef chicken breast cream mushroom soup sour cream place chip beef bottom bake dish place chicken top beef mix soup cream pour chicken bake uncov hour
www.cookbooks.com/Recipe-Details.aspx?id=699419
Creamy Corn
creami corn frozen corn cream chees butter garlic powder salt pepper slow cooker combin ingredi cover cook low hour heat chees melt stir serv yield serv
www.cookbooks.com/Recipe-Details.aspx?id=10570
Chicken Funny
chicken funni chicken chicken gravi cream mushroom soup shred chees boil debon chicken put bite size piec averag size squar casserol dish pour gravi cream mushroom soup chicken level make stuf instruct box make moist put stuf top chicken gravi level sprinkl shred chees top bake approxim minut golden bubbl
www.cookbooks.com/Recipe-Details.aspx?id=897570

Figure 2: Preprocessed Recipe dataset after normalization, tokenization, and stemming

Details of Approach – Writing the Index with Whoosh Library Python

- MyIndexWriter – uses preprocessed data to write an index with whoosh schema
  - ID : title
  - TEXT : recipe
  - ID: url
- Whoosh RegexTokenizer was used to tokenize TEXT : recipe
- Whoosh index used to build index

Details of Approach – Writing the Index with Whoosh Library Python

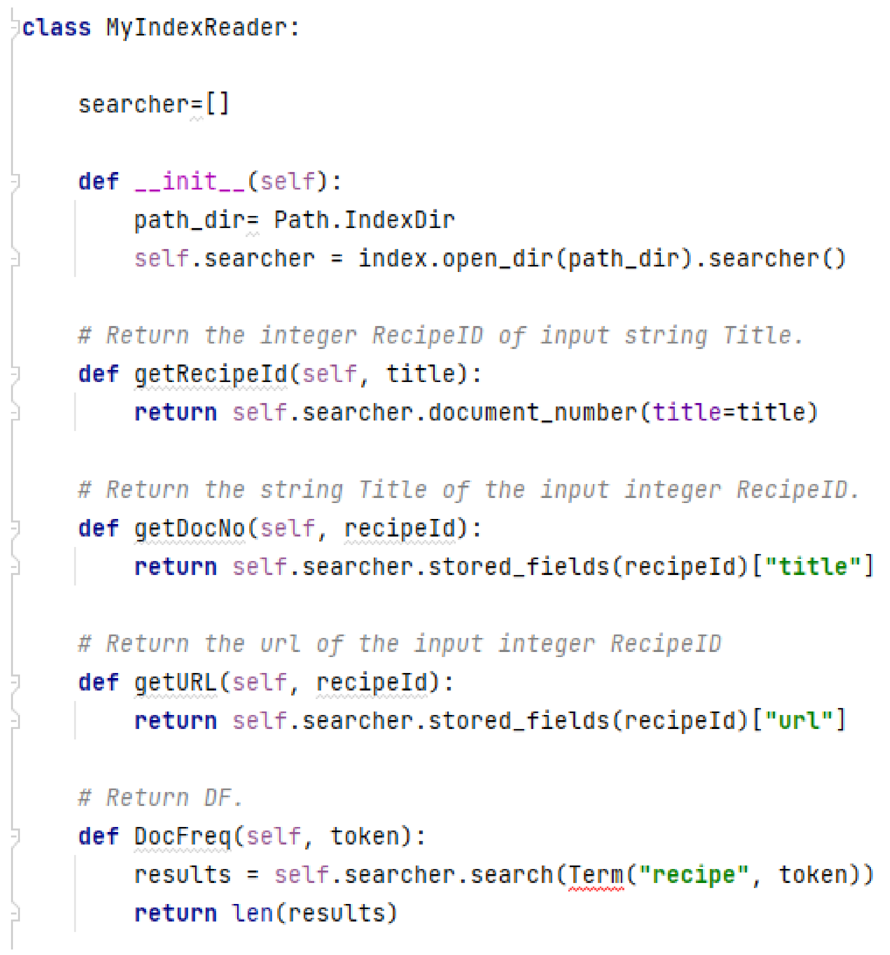


```python
class MyIndexWriter:
    writer = []

    def __init__(self):
        path_dir = Path.IndexDir  # set output path of index
        os.makedirs(path_dir, exist_ok=True)
        schema = Schema(title=ID(stored=True),
                        recipe=TEXT(analyzer=RegexTokenizer(), stored=True),
                        url=ID(stored=True))
        indexing = index.create_in(path_dir, schema)
        self.writer = indexing.writer()
        return

    def index(self, title, recipe, url):
        self.writer.add_document(title=title, recipe=recipe, url = url)
        return

    # Close the index writer, and you should output all the buffered content (if any).
    def close(self):
        self.writer.commit()
        return
```

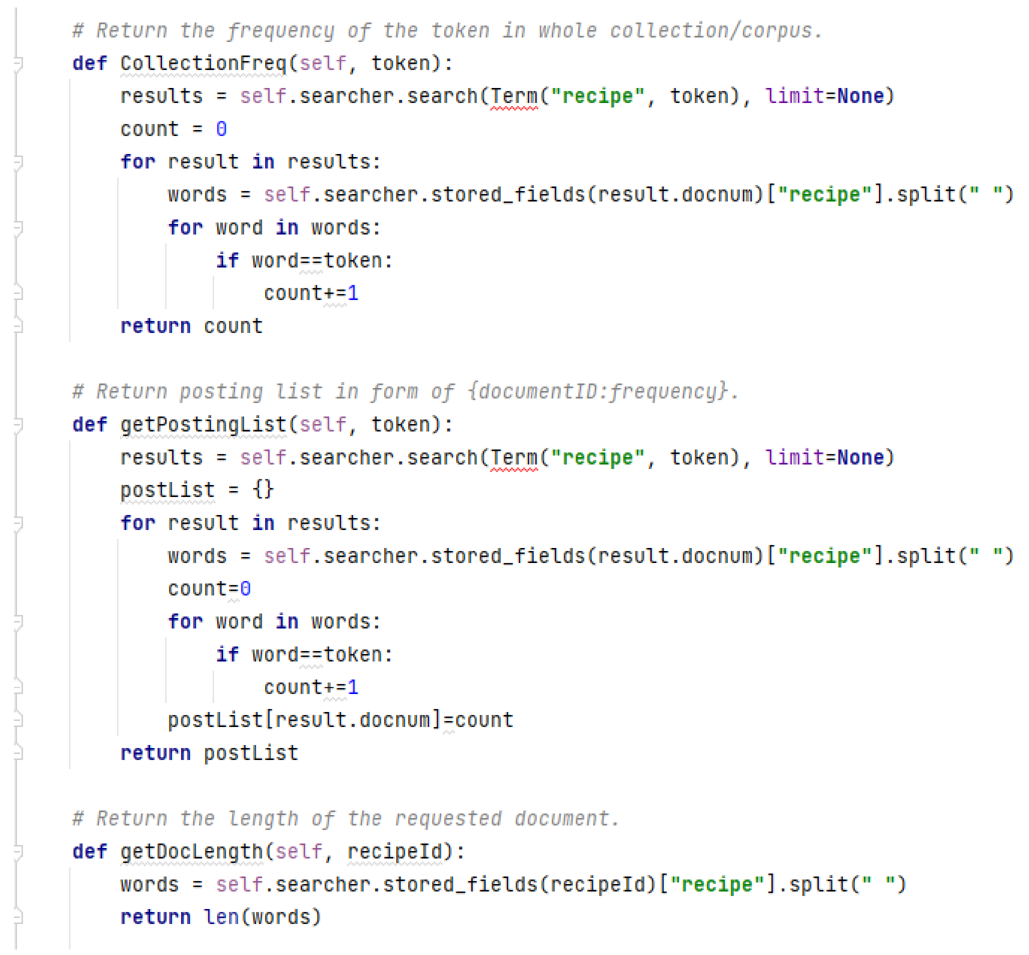
Details of Approach – Reading the Index with Whoosh Library Python

- MyIndexReader – uses whoosh searcher object in functions to return following…
  - recipeID (recipe title)
  - url
  - token’s document frequency
  - token’s collection frequency
  - recipe document’s length.
- It also contains the function that creates the postings list from the index using Whoosh searcher object to search index

Details of Approach – Reading the Index with Whoosh Library Python



```
class MyIndexReader:

    searcher=[]

    def __init__(self):
        path_dir= Path.IndexDir
        self.searcher = index.open_dir(path_dir).searcher()

    # Return the integer RecipeID of input string Title.
    def getRecipeId(self, title):
        return self.searcher.document_number(title=title)

    # Return the string Title of the input integer RecipeID.
    def getDocNo(self, recipeId):
        return self.searcher.stored_fields(recipeId)["title"]

    # Return the url of the input integer RecipeID
    def getURL(self, recipeId):
        return self.searcher.stored_fields(recipeId)["url"]

    # Return DF.
    def DocFreq(self, token):
        results = self.searcher.search(Term("recipe", token))
        return len(results)
```

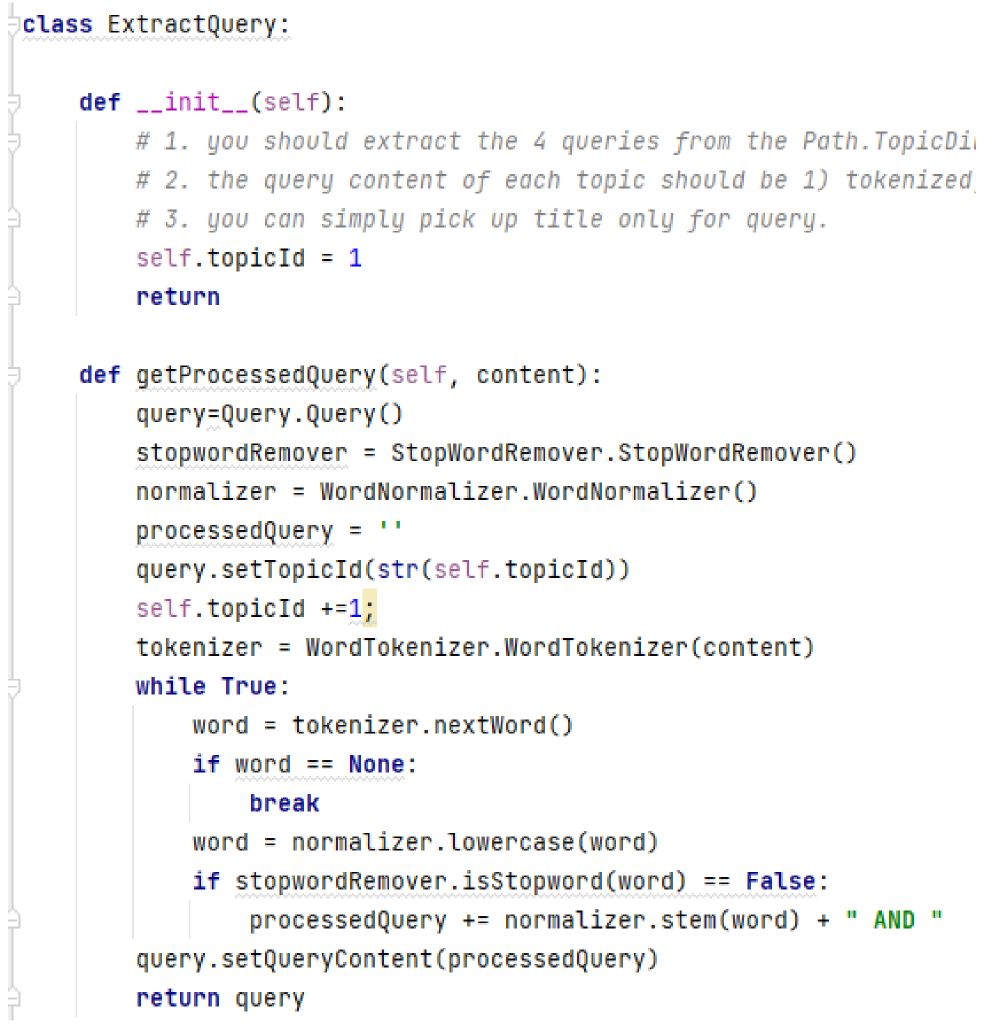


```
# Return the frequency of the token in whole collection/corpus.
def CollectionFreq(self, token):
    results = self.searcher.search(Term("recipe", token), limit=None)
    count = 0
    for result in results:
        words = self.searcher.stored_fields(result.docnum)["recipe"].split(" ")
        for word in words:
            if word==token:
                count+=1
    return count

# Return posting list in form of {documentID:frequency}.
def getPostingList(self, token):
    results = self.searcher.search(Term("recipe", token), limit=None)
    postList = {}
    for result in results:
        words = self.searcher.stored_fields(result.docnum)["recipe"].split(" ")
        count=0
        for word in words:
            if word==token:
                count+=1
        postList[result.docnum]=count
    return postList

# Return the length of the requested document.
def getDocLength(self, recipeId):
    words = self.searcher.stored_fields(recipeId)["recipe"].split(" ")
    return len(words)
```

Details of Approach – Query Extraction

ExtractQuery
- Takes user query as input
- returns the processed query

- getProcessedQuery
  - Uses the same preprocssing approach as performed on the recipe dataset input
  - Includes tokenization, stop word removal, and normalization of the query input

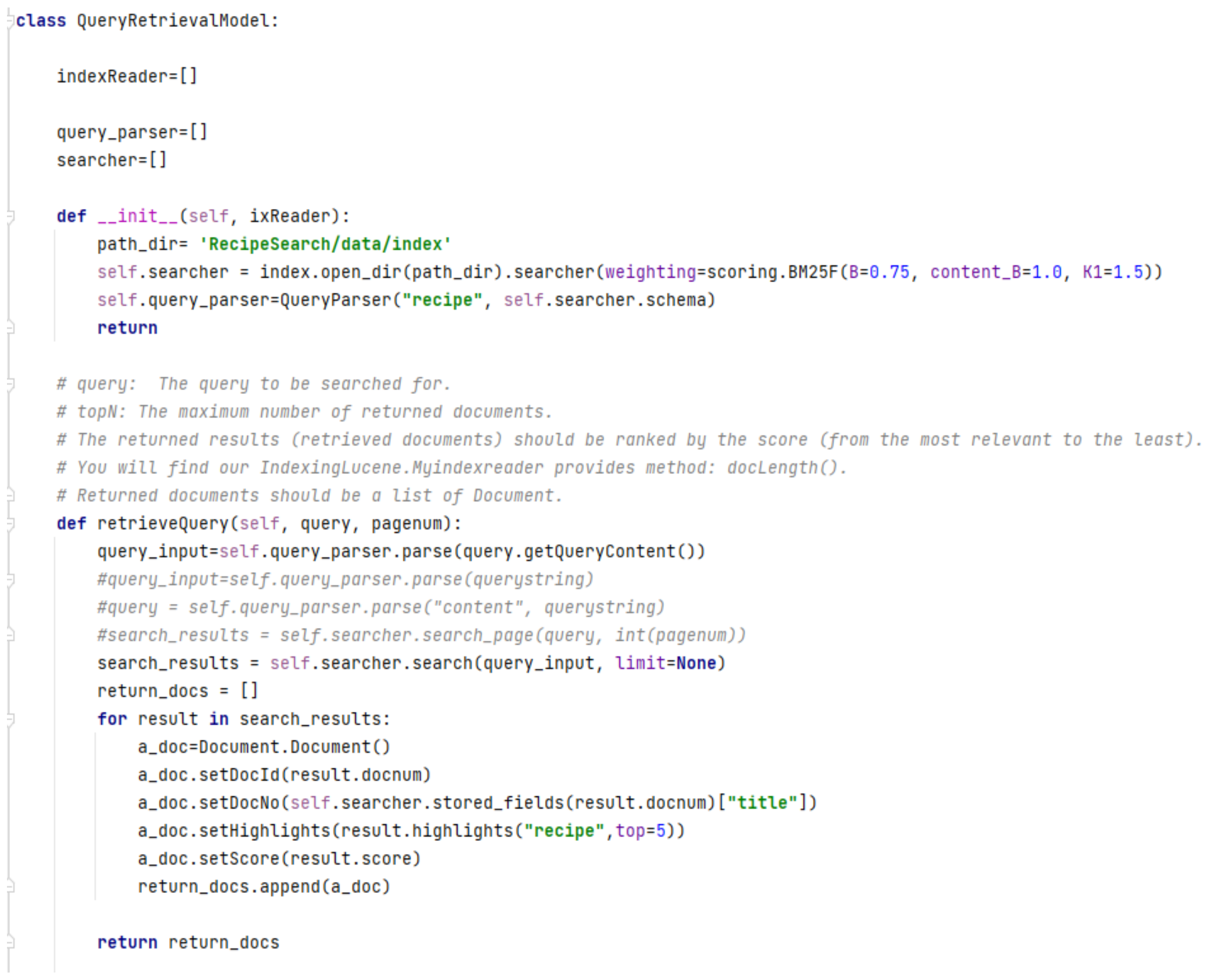


```
class ExtractQuery:

    def __init__(self):
        # 1. you should extract the 4 queries from the Path.TopicDi
        # 2. the query content of each topic should be 1) tokenized
        # 3. you can simply pick up title only for query.
        self.topicId = 1
        return

    def getProcessedQuery(self, content):
        query=Query.Query()
        stopwordRemover = StopWordRemover.StopWordRemover()
        normalizer = WordNormalizer.WordNormalizer()
        processedQuery = ''
        query.setTopicId(str(self.topicId))
        self.topicId +=1;
        tokenizer = WordTokenizer.WordTokenizer(content)
        while True:
            word = tokenizer.nextWord()
            if word == None:
                break
            word = normalizer.lowercase(word)
            if stopwordRemover.isStopword(word) == False:
                processedQuery += normalizer.stem(word) + " AND "
        query.setQueryContent(processedQuery)
        return query
```

Details of Approach – Query Retrieval and Scoring with Whoosh Library Python

QueryRetrievalModel

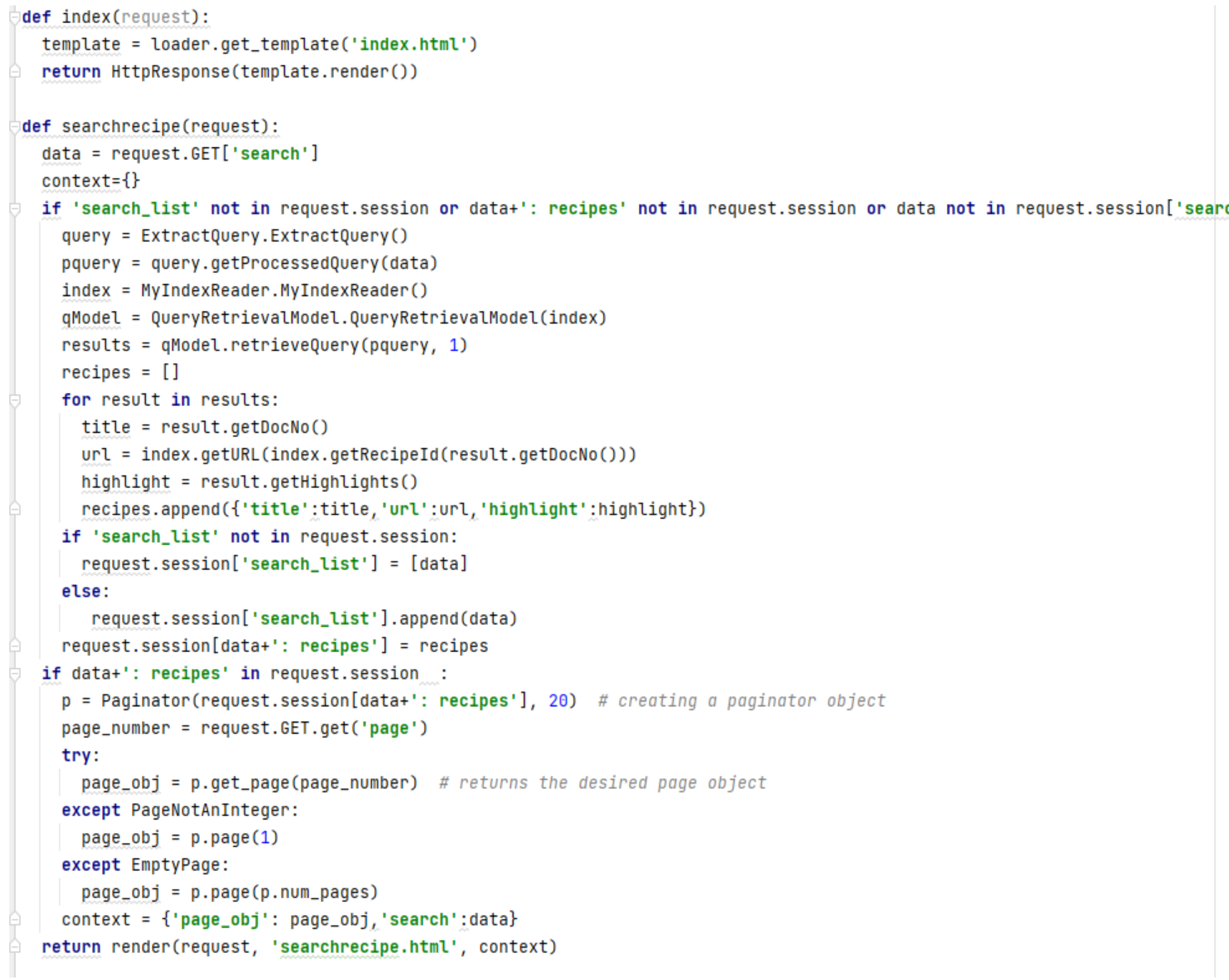
- Takes user query as input
- Returns the set number of recipe documents determined by the scoring
- Whoosh QueryParser to convert query string into query object
- Whoosh searcher object used for searching and scoring results from index
- Whoosh BM25F algorithm used for scoring the returned recipe documents.
    - Default values of B=0.75 and K1 = 1.5 used



```python
class QueryRetrievalModel:

    indexReader=[]

    query_parser=[]
    searcher=[]

    def __init__(self, ixReader):
        path_dir= 'RecipeSearch/data/index'
        self.searcher = index.open_dir(path_dir).searcher(weighting=scoring.BM25F(B=0.75, content_B=1.0, K1=1.5))
        self.query_parser=QueryParser("recipe", self.searcher.schema)
        return

    # query:  The query to be searched for.
    # topN: The maximum number of returned documents.
    # The returned results (retrieved documents) should be ranked by the score (from the most relevant to the least).
    # You will find our IndexingLucene.Myindexreader provides method: docLength().
    # Returned documents should be a list of Document.
    def retrieveQuery(self, query, pagenum):
        query_input=self.query_parser.parse(query.getQueryContent())
        #query_input=self.query_parser.parse(querystring)
        #query = self.query_parser.parse("content", querystring)
        #search_results = self.searcher.search_page(query, int(pagenum))
        search_results = self.searcher.search(query_input, limit=None)
        return_docs = []
        for result in search_results:
            a_doc=Document.Document()
            a_doc.setDocId(result.docnum)
            a_doc.setDocNo(self.searcher.stored_fields(result.docnum)["title"])
            a_doc.setHighlights(result.highlights("recipe",top=5))
            a_doc.setScore(result.score)
            return_docs.append(a_doc)

        return return_docs
```

Details of Approach – Creating web page and integrating backend using Django

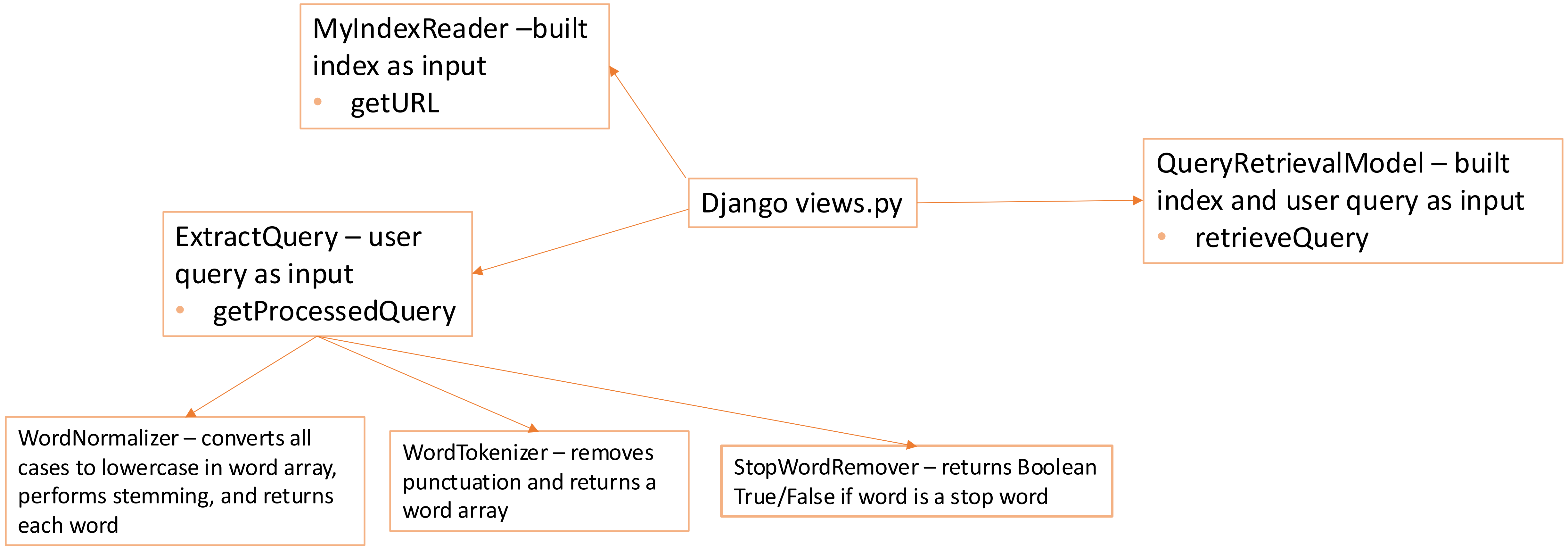
- Frontend html web page created using Django library
- Views.py Django file includes:
  - Web page creation and display of results
  - Query extraction and retrieval
  - Reading of the Index



```
def index(request):
  template = loader.get_template('index.html')
  return HttpResponse(template.render())

def searchrecipe(request):
  data = request.GET['search']
  context={}
  if 'search_list' not in request.session or data+': recipes' not in request.session or data not in request.session['search_list']:
    query = ExtractQuery.ExtractQuery()
    pquery = query.getProcessedQuery(data)
    index = MyIndexReader.MyIndexReader()
    qModel = QueryRetrievalModel.QueryRetrievalModel(index)
    results = qModel.retrieveQuery(pquery, 1)
    recipes = []
    for result in results:
      title = result.getDocNo()
      url = index.getURL(index.getRecipeId(result.getDocNo()))
      highlight = result.getHighlights()
      recipes.append({'title':title,'url':url,'highlight':highlight})
    if 'search_list' not in request.session:
      request.session['search_list'] = [data]
    else:
       request.session['search_list'].append(data)
    request.session[data+': recipes'] = recipes
  if data+': recipes' in request.session  :
    p = Paginator(request.session[data+': recipes'], 20)  # creating a paginator object
    page_number = request.GET.get('page')
    try:
      page_obj = p.get_page(page_number)  # returns the desired page object
    except PageNotAnInteger:
      page_obj = p.page(1)
    except EmptyPage:
      page_obj = p.page(p.num_pages)
    context = {'page_obj': page_obj,'search':data}
  return render(request, 'searchrecipe.html', context)
```

Figure 3: views.py

Details of Approach – Overview


MyIndexReader –built index as input
• getURL
Django views.py
QueryRetrievalModel – built index and user query as input
• retrieveQuery
ExtractQuery – user query as input
• getProcessedQuery
WordNormalizer – converts all cases to lowercase in word array, performs stemming, and returns each word
WordTokenizer – removes punctuation and returns a word array
StopWordRemover – returns Boolean True/False if word is a stop word

Performance Evaluation

- The system performs well in the following ways:
  - Final project output is a user-friendly search engine that allows user to search for a specific ingredient or recipe title
  - System returns recipes results with Title and description along with clickable url that leads to the recipe instructions
  - System highlights the searched query in the returned results results

- Improvements
  - The title of the recipe was preprocessed and indexed along with the rest of the content , there is no separate search capability for the title only.
    - Ex: a query for the title “funny funny chicken” returns the recipe titled as “funny funny chicken” as the second highest result
  - There is no filtering capability for the results returned