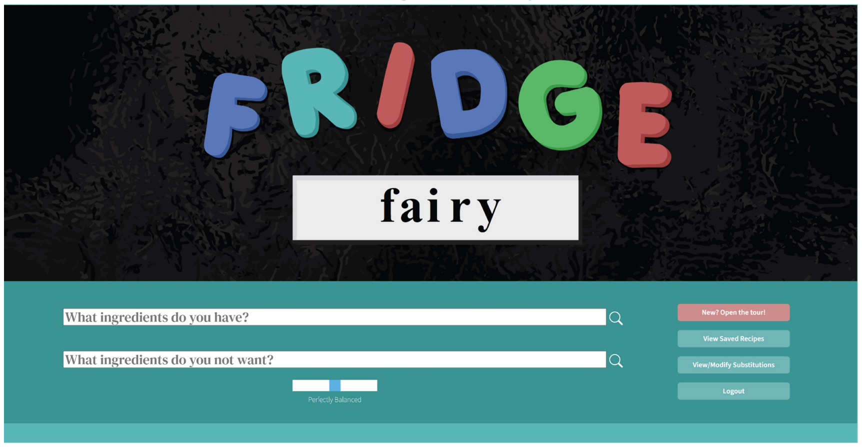
Fridge Fairy


FRIDGE
fairy
What ingredients do you have?
What ingredients do you not want?
Perfectly Balanced
New? Open the tour!
View Saved Recipes
View/Modify Substitutions
Logout

FridgeFairy is a website dedicated to discovering recipes based on input ingredients. The user inputs the ingredients that they would like to use and the FridgeFairy searches over 1,000 recipes to find matches. FridgeFairy uses ElasticSearch to store recipes. ElasticSearch is an API that takes in a large amount of data in the form of JSON and allows that data to be queried in a dynamic manner. The backend was coded in Python and interfaces with the ElasticSearch rest API.

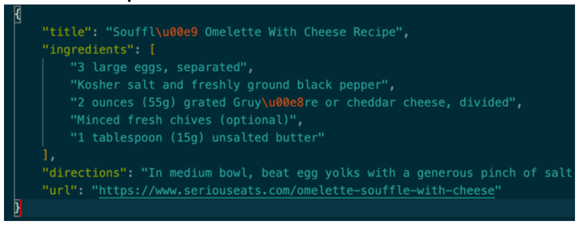
Features

Slider:
FridgeFairy has two modes: survival and creative. In survival mode, all of the user's ingredients must be in the resulting recipes. In creative mode, only a small percentage of ingredients must be in each result, but recipes with more ingredient matches appear first.

Meal Chaining:
After a search, the user can choose what ingredients have been used and remove them. The ingredients are saved, so they can perform a new search and save recipes for their meal chains.

Substitutions:
Users can input ingredient that they would like to substitute for another ingredient. For example, almond, coconut, and oat flour can be substitutes for regular flour. This, along with the user including search terms they don't want, is our method of accommodating allergies.

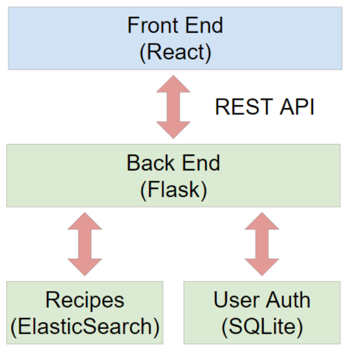
JSON recipe structure:



```
{
    "title": "Souffl\u00e9 Omelette With Cheese Recipe",
    "ingredients": [
        "3 large eggs, separated",
        "Kosher salt and freshly ground black pepper",
        "2 ounces (55g) grated Gruy\u00e8re or cheddar cheese, divided",
        "Minced fresh chives (optional)",
        "1 tablespoon (15g) unsalted butter"
    ],
    "directions": "In medium bowl, beat egg yolks with a generous pinch of salt
    "url": "https://www.seriouseats.com/omelette-souffle-with-cheese"
}
```

Design

Overall design:



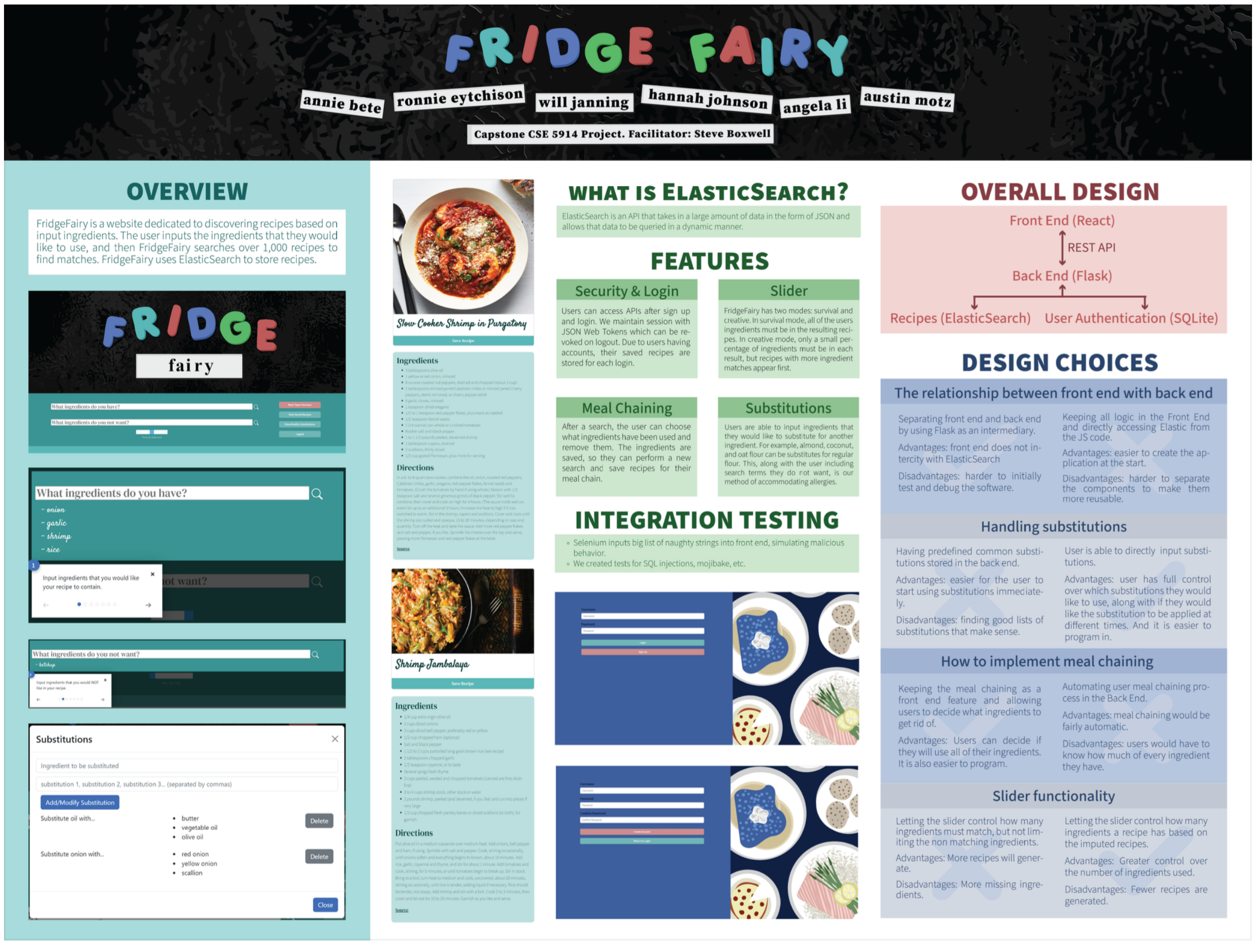
Design Choices:

The relationship between front end with back end	
Separating front end and back end by using Flask as an intermediary. Advantages: front end does not intercity with ElasticSearch Disadvantages: harder to initially test and debug the software.	Keeping all logic in the Front End and directly accessing Elastic from the JS code. Advantages: easier to create the application at the start. Disadvantages: harder to separate the components to make them more reusable.
Handling substitutions	
Having predefined common substitutions stored in the back end. Advantages: easier for the user to start using substitutions immediately. Disadvantages: finding good lists of substitutions that make sense.	User can directly input substitutions. Advantages: user has full control over which substitutions they would like to use, along with if they would like the substitution to be applied at different times. And it is easier to program in.
How to implement meal chaining	
Keeping the meal chaining as a front-end feature and allowing users to decide what ingredients to get rid of. Advantages: Users can decide if they will use all their ingredients. It is also easier to program.	Automating user meal chaining process in the Back End. Advantages: meal chaining would be fairly automatic. Disadvantages: users would have to know how much of every ingredient they have.
Slider functionality	
Letting the slider control how many ingredients must match, but not limiting the non-matching ingredients. Advantages: More recipes will generate. Disadvantages: More missing ingredients.	Letting the slider control how many ingredients a recipe has based on the imputed recipes. Advantages: Greater control over the number of ingredients used. Disadvantages: Fewer recipes are generated

My Role

My main role in the project was in the overall ideation and design, the slider, and the backend helper classes. I had the initial idea of creating the recipe generator. Additionally, I proposed the structure of the backend to help reduce coupling and increase flexibility of our product. I researched the API of ElasticSearch to understand the best functions to use for our goals. I was a big proponent in creating flexibility for users with different needs, suggesting the functionalities of the slider and the substitutions. In terms of code, I wrote the query and search functions as well as the slider functionality.

Poster



FRIDGE FAIRY

annie bete, ronnie eytchison, will janning, hannah johnson, angela li, austin motz

Capstone CSE 5914 Project. Facilitator: Steve Boxwell

OVERVIEW

FridgeFairy is a website dedicated to discovering recipes based on input ingredients. The user inputs the ingredients that they would like to use, and then FridgeFairy searches over 1,000 recipes to find matches. FridgeFairy uses ElasticSearch to store recipes.

What ingredients do you have?
- onion
- garlic
- shrimp
- rice

Input ingredients that you would like your recipe to contain.

What ingredients do you not want?
- ketchup

Substitutions

Ingredient to be substituted

substitution 1, substitution 2, substitution 3... (separated by commas)

Add/Modify Substitution

Substitute oil with... butter, vegetable oil, olive oil — Delete

Substitute onion with... red onion, yellow onion, scallion — Delete

Close

Slow Cooker Shrimp in Purgatory

Shrimp Jambalaya

WHAT IS ELASTICSEARCH?

ElasticSearch is an API that takes in a large amount of data in the form of JSON and allows that data to be queried in a dynamic manner.

FEATURES

Security & Login

Users can access APIs after sign up and login. We maintain session with JSON Web Tokens which can be revoked on logout. Due to users having accounts, their saved recipes are stored for each login.

Slider

FridgeFairy has two modes: survival and creative. In survival mode, all of the users ingredients must be in the resulting recipes. In creative mode, only a small percentage of ingredients must be in each result, but recipes with more ingredient matches appear first.

Meal Chaining

After a search, the user can choose what ingredients have been used and remove them. The ingredients are saved, so they can perform a new search and save recipes for their meal chain.

Substitutions

Users are able to input ingredients that they would like to substitute for another ingredient. For example, almond, coconut, and oat flour can be substitutes for regular flour. This, along with the user including search terms they do not want, is our method of accommodating allergies.

INTEGRATION TESTING

- Selenium inputs big list of naughty strings into front end, simulating malicious behavior.
- We created tests for SQL injections, mojibake, etc.

OVERALL DESIGN

$$\begin{tikzcd} & \text{Front End (React)} \arrow[d, leftrightarrow, "\text{REST API}"] & \\ \text{Recipes (ElasticSearch)} & \text{Back End (Flask)} \arrow[l, leftrightarrow] \arrow[r, leftrightarrow] & \text{User Authentication (SQLite)} \end{tikzcd}$$

DESIGN CHOICES

The relationship between front end with back end

Separating front end and back end by using Flask as an intermediary.

Advantages: front end does not intercity with ElasticSearch

Disadvantages: harder to initially test and debug the software.

Keeping all logic in the Front End and directly accessing Elastic from the JS code.

Advantages: easier to create the application at the start.

Disadvantages: harder to separate the components to make them more reusable.

Handling substitutions

Having predefined common substitutions stored in the back end.

Advantages: easier for the user to start using substitutions immediately.

Disadvantages: finding good lists of substitutions that make sense.

User is able to directly input substitutions.

Advantages: user has full control over which substitutions they would like to use, along with if they would like the substitution to be applied at different times. And it is easier to program in.

How to implement meal chaining

Keeping the meal chaining as a front end feature and allowing users to decide what ingredients to get rid of.

Advantages: Users can decide if they will use all of their ingredients. It is also easier to program.

Automating user meal chaining process in the Back End.

Advantages: meal chaining would be fairly automatic.

Disadvantages: users would have to know how much of every ingredient they have.

Slider functionality

Letting the slider control how many ingredients must match, but not limiting the non matching ingredients.

Advantages: More recipes will generate.

Disadvantages: More missing ingredients.

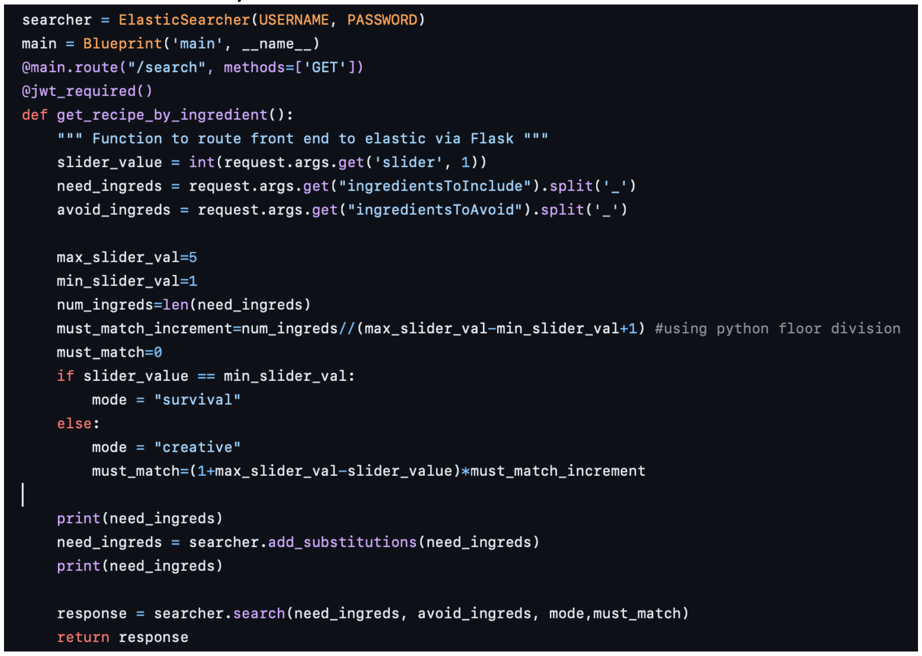
Letting the slider control how many ingredients a recipe has based on the imputed recipes.

Advantages: Greater control over the number of ingredients used.

Disadvantages: Fewer recipes are generated.

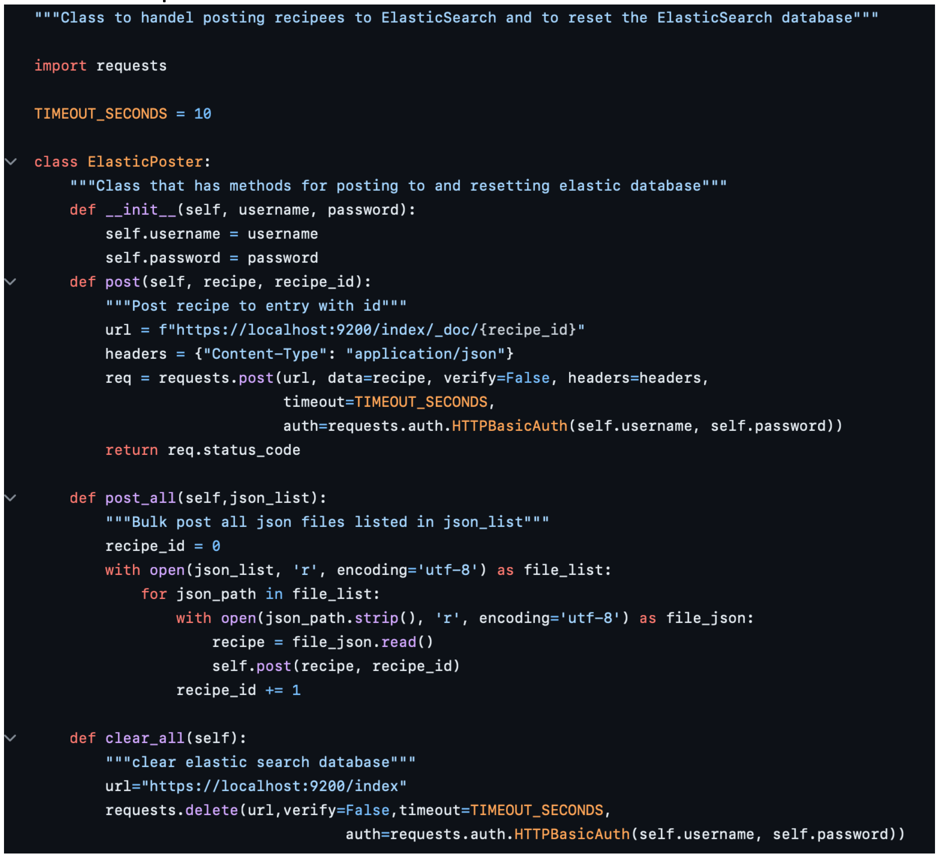
Code Samples

Slider functionality:

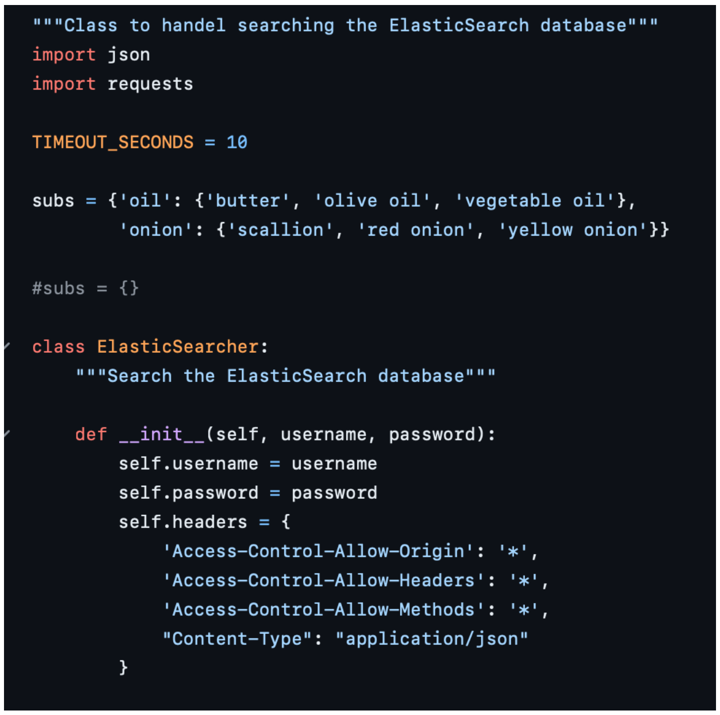


```python
searcher = ElasticSearcher(USERNAME, PASSWORD)
main = Blueprint('main', __name__)
@main.route("/search", methods=['GET'])
@jwt_required()
def get_recipe_by_ingredient():
    """ Function to route front end to elastic via Flask """
    slider_value = int(request.args.get('slider', 1))
    need_ingreds = request.args.get("ingredientsToInclude").split('_')
    avoid_ingreds = request.args.get("ingredientsToAvoid").split('_')

    max_slider_val=5
    min_slider_val=1
    num_ingreds=len(need_ingreds)
    must_match_increment=num_ingreds//(max_slider_val-min_slider_val+1) #using python floor division
    must_match=0
    if slider_value == min_slider_val:
        mode = "survival"
    else:
        mode = "creative"
        must_match=(1+max_slider_val-slider_value)*must_match_increment

    print(need_ingreds)
    need_ingreds = searcher.add_substitutions(need_ingreds)
    print(need_ingreds)

    response = searcher.search(need_ingreds, avoid_ingreds, mode,must_match)
    return response
```

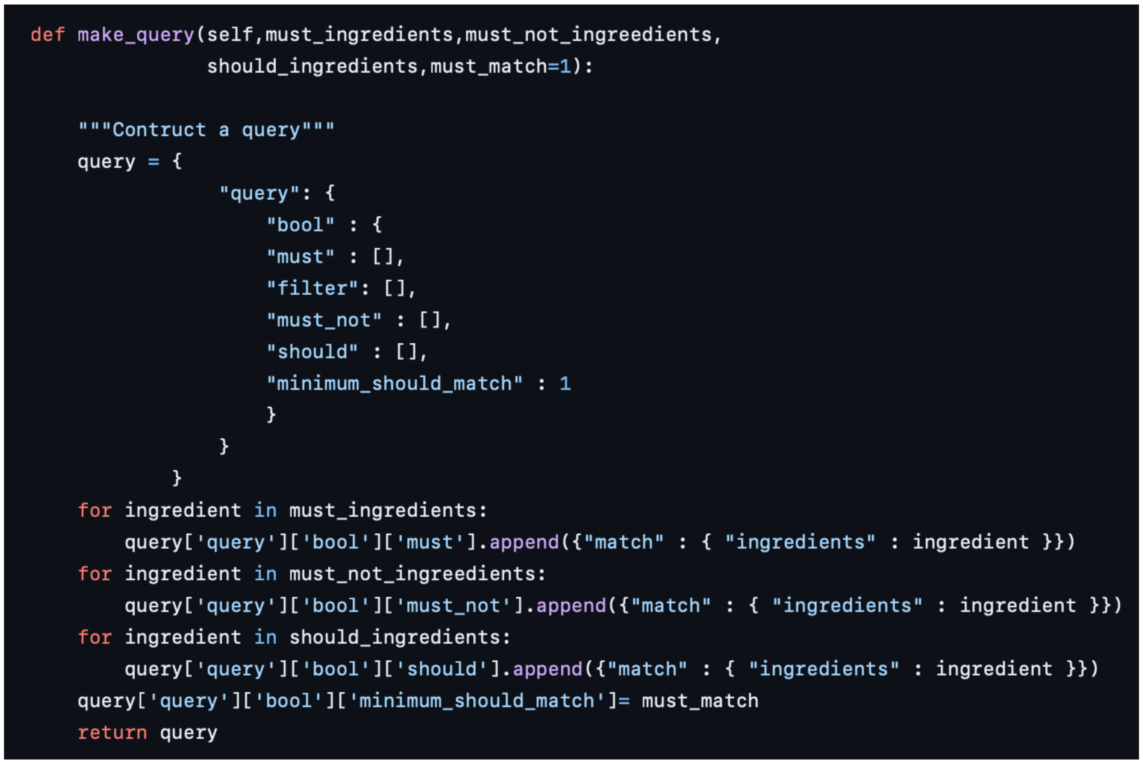
Backend helper classes:



```python
"""Class to handel posting recipees to ElasticSearch and to reset the ElasticSearch database"""

import requests

TIMEOUT_SECONDS = 10

class ElasticPoster:
    """Class that has methods for posting to and resetting elastic database"""
    def __init__(self, username, password):
        self.username = username
        self.password = password
    def post(self, recipe, recipe_id):
        """Post recipe to entry with id"""
        url = f"https://localhost:9200/index/_doc/{recipe_id}"
        headers = {"Content-Type": "application/json"}
        req = requests.post(url, data=recipe, verify=False, headers=headers,
                            timeout=TIMEOUT_SECONDS,
                            auth=requests.auth.HTTPBasicAuth(self.username, self.password))
        return req.status_code

    def post_all(self,json_list):
        """Bulk post all json files listed in json_list"""
        recipe_id = 0
        with open(json_list, 'r', encoding='utf-8') as file_list:
            for json_path in file_list:
                with open(json_path.strip(), 'r', encoding='utf-8') as file_json:
                    recipe = file_json.read()
                    self.post(recipe, recipe_id)
                recipe_id += 1

    def clear_all(self):
        """clear elastic search database"""
        url="https://localhost:9200/index"
        requests.delete(url,verify=False,timeout=TIMEOUT_SECONDS,
                                   auth=requests.auth.HTTPBasicAuth(self.username, self.password))
```



```python
"""Class to handel searching the ElasticSearch database"""
import json
import requests

TIMEOUT_SECONDS = 10

subs = {'oil': {'butter', 'olive oil', 'vegetable oil'},
        'onion': {'scallion', 'red onion', 'yellow onion'}}

#subs = {}

class ElasticSearcher:
    """Search the ElasticSearch database"""

    def __init__(self, username, password):
        self.username = username
        self.password = password
        self.headers = {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': '*',
            'Access-Control-Allow-Methods': '*',
            "Content-Type": "application/json"
        }
```



```python
def make_query(self,must_ingredients,must_not_ingreedients,
               should_ingredients,must_match=1):

    """Contruct a query"""
    query = {
                "query": {
                    "bool" : {
                    "must" : [],
                    "filter": [],
                    "must_not" : [],
                    "should" : [],
                    "minimum_should_match" : 1
                    }
                }
            }
    for ingredient in must_ingredients:
        query['query']['bool']['must'].append({"match" : { "ingredients" : ingredient }})
    for ingredient in must_not_ingreedients:
        query['query']['bool']['must_not'].append({"match" : { "ingredients" : ingredient }})
    for ingredient in should_ingredients:
        query['query']['bool']['should'].append({"match" : { "ingredients" : ingredient }})
    query['query']['bool']['minimum_should_match']= must_match
    return query
```

```python
def search_one(self, keyword):
    """Testing Get recipes whose ingredients match a single keyword"""
    url = "https://localhost:9200/index/_search/"
    query = {
            "query" : {
                "match" : {
                    "ingredients": ""
                }
            }
        }
    query['query']['match']['ingredients'] = keyword
    req = requests.get(url, verify=False, headers=self.headers, json=query,
                    timeout=TIMEOUT_SECONDS,
                    auth=requests.auth.HTTPBasicAuth(self.username, self.password))
    if not req.ok:
        return []
    content = json.loads(req.content)
    hits = content['hits']['hits']
    return hits
```

```python
def search(self, need_ingreds, avoid_ingreds, mode,must_match):
    """Get recipes whose ingredients match a list of keywords based on the mode"""
    if mode=="survival":
        print("survival")
        return self.survival_search(need_ingreds, avoid_ingreds,must_match)
    # mode=="creative"
    print("creative")
    return self.creative_search(need_ingreds, avoid_ingreds,must_match)

def survival_search(self,keywords,filter_out_words,must_match):
    """Get recipes whose ingredients match all elements in a list of keywords
    (ie use and)
    and dont include any filterwords"""
    url = "https://localhost:9200/index/_search/"
    query = self.make_query(keywords,filter_out_words,[],must_match)
    req = requests.get(url, verify=False, headers=self.headers, json=query,
                       timeout=TIMEOUT_SECONDS,
                    auth=requests.auth.HTTPBasicAuth(self.username, self.password))
    if not req.ok:
        return []
    content = json.loads(req.content)
    hits = content['hits']['hits']
    return hits
```

```python
def creative_search(self,keywords,filter_out_words,must_match):
    """Get recipes whose ingredients match some elements in a list of
    keywords (ie use or)
    and dont include filterwords"""
    url = "https://localhost:9200/index/_search/"
    query = self.make_query([],filter_out_words,keywords, must_match)
    req = requests.get(url, verify=False, headers=self.headers, json=query,
                       timeout=TIMEOUT_SECONDS,
                    auth=requests.auth.HTTPBasicAuth(self.username, self.password))
    if not req.ok:
        return []
    content = json.loads(req.content)
    hits = content['hits']['hits']
    return hits
```