# RecipeDB Backend

|  |  |  |  |
| --- | --- | --- | --- |
| Endpoint | Method | Description | User Type |
| /register | POST | Registers a new user | any |
| /login | POST | Logs user in | any |
| /logout | POST | Logs user out | registered |
| /recipes/top | GET | Gets 6 random recipes with a five-star rating | any |
| /recipes/search | GET | Searches recipes for criteria in the recipe title | any |
| /recipes | POST | Scrapes given bbcgoodfood.com URL | admin |
| /recipe/<id> | GET | Gets a recipe’s detail, if user is logged in also returns whether or not the recipe is bookmarked | any/registered |
|  | DELETE | Deletes the recipe from system | admin |
| /recipe/<id>/comments | GET | Gets all comments for a recipe | any |
| POST | Adds a new comment to the recipe | registered |
| PUT | Edits an existing comment for a recipe | registered |
| DELETE | Deletes an existing comment from a recipe | registered |
| /recipe/<id>/bookmark | POST | Adds recipe to user’s bookmarks | registered |
| DELETE | Removes recipe from user’s bookmarks | registered |
| /bookmarks | GET | Gets all of a user’s bookmarks | registered |

## Code Listing

**C:\Users\rob\Documents\GitHub\RecipeDB\api\app.py**

from flask import Flask, request

from flask\_cors import CORS

from functools import wraps

from config import SECRET\_KEY, MONGO, MONGO\_INDEXES

from jwt import decode

from pymongo import MongoClient, TEXT

from util import response

from auth import login, logout, register

from recipes import get\_recipe, search\_recipes, new\_recipe, delete\_recipe, get\_top\_recipes

from comments import get\_recipe\_comments, new\_recipe\_comment, update\_recipe\_comment, delete\_recipe\_comment

from bookmarks import get\_bookmarks, bookmark\_recipe, unbookmark\_recipe

app = Flask('RecipeDB')

CORS(app)

client = MongoClient(MONGO)

db = client.RecipeDB

blacklist = db.blacklist

\_user = None

for index in MONGO\_INDEXES:

if index['name'] not in db.recipes.index\_information():

db.recipes.create\_index([(index['field'], TEXT)], name=index['name'], default\_language='english')

def jwt\_required(func):

@wraps(func)

def jwt\_required\_wrapper(\*args, \*\*kwargs):

global \_user

if 'x-access-token' in request.headers:

token = request.headers['x-access-token']

if not token:

return response(401, 'Token is missing')

try:

\_user = decode(token, SECRET\_KEY)

except:

return response(401, 'Token is invalid.')

bl\_token = blacklist.find\_one({"token": token})

if bl\_token is not None:

return response(401, 'Token is invalid.')

return func(\*args, \*\*kwargs)

return jwt\_required\_wrapper

def jwt\_optional(func):

@wraps(func)

def jwt\_optional\_wrapper(\*args, \*\*kwargs):

global \_user

if 'x-access-token' in request.headers:

token = request.headers['x-access-token']

try:

usr = decode(token, SECRET\_KEY)

bl\_token = blacklist.find\_one({"token": token})

if bl\_token is None:

\_user = usr

return func(\*args, \*\*kwargs)

except:

return func(\*args, \*\*kwargs)

return jwt\_optional\_wrapper

def admin\_required(func):

@wraps(func)

def admin\_required\_wrapper(\*args, \*\*kwargs):

token = request.headers['x-access-token']

data = decode(token, SECRET\_KEY)

if data["admin"]:

return func(\*args, \*\*kwargs)

else:

return response(401, 'Admin required')

return admin\_required\_wrapper

@app.route('/login', methods=['POST'])

def app\_login():

return login()

@app.route('/logout', methods=['POST'])

@jwt\_required

def app\_logout():

return logout()

@app.route('/register', methods=['POST'])

def app\_register():

return register()

@app.route('/recipes/top', methods=['GET'])

def app\_get\_top\_recipes():

return get\_top\_recipes()

@app.route('/recipes/search', methods=['GET'])

def app\_search\_recipes():

return search\_recipes()

@app.route('/recipe/<string:\_id>', methods=['GET'])

@jwt\_optional

def app\_get\_recipe(\_id):

global \_user

return get\_recipe(\_id, \_user)

@app.route('/recipe/<string:\_id>', methods=['DELETE'])

@admin\_required

def app\_delete\_recipe(\_id):

return delete\_recipe(\_id)

@app.route('/recipe/<string:\_id>/comments', methods=['GET'])

def app\_get\_recipe\_comments(\_id):

return get\_recipe\_comments(\_id)

@app.route('/recipe/<string:\_id>/comments', methods=['POST'])

@jwt\_required

def app\_new\_recipe\_comment(\_id):

global \_user

return new\_recipe\_comment(\_id, \_user)

@app.route('/recipe/<string:\_id>/comments', methods=['PUT'])

@jwt\_required

def app\_update\_recipe\_comment(\_id):

global \_user

return update\_recipe\_comment(\_id, \_user)

@app.route('/recipe/<string:\_id>/comments', methods=['DELETE'])

@jwt\_required

def app\_delete\_recipe\_comment(\_id):

global \_user

return delete\_recipe\_comment(\_id, \_user['\_id'])

@app.route('/recipe/<string:\_id>/bookmark', methods=['POST'])

@jwt\_required

def app\_bookmark\_recipe(\_id):

global \_user

return bookmark\_recipe(\_id, \_user)

@app.route('/recipe/<string:\_id>/bookmark', methods=['DELETE'])

@jwt\_required

def app\_unbookmark\_recipe(\_id):

global \_user

return unbookmark\_recipe(\_id, \_user)

@app.route('/bookmarks', methods=['GET'])

@jwt\_required

def app\_get\_bookmarks():

global \_user

return get\_bookmarks(\_user['\_id'])

@app.route('/recipes', methods=['POST'])

@admin\_required

def app\_new\_recipe():

return new\_recipe()

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**C:\Users\rob\Documents\GitHub\RecipeDB\api\auth.py**

from flask import request

from pymongo import MongoClient

from bcrypt import hashpw, gensalt, checkpw

from config import MONGO, SECRET\_KEY

from util import response

from datetime import datetime, timedelta

from jwt import encode

from json import dumps

client = MongoClient(MONGO)

db = client.RecipeDB

users = db.users

blacklist = db.blacklist

def login():

form = request.form

username = form['username']

password = form['password']

if username and password:

user = users.find\_one({'username': username})

if user is not None:

if checkpw(bytes(password, 'UTF-8'), user['password']):

exp = datetime.utcnow() + timedelta(hours=24)

token = encode({

'\_id': str(user['\_id']),

'username': username,

'admin': user['admin'],

'exp': exp

}, SECRET\_KEY)

exp\_clean = dumps(exp.isoformat())

exp\_clean = exp\_clean[1:-1]

return response(200, {

'id': str(user['\_id']),

'username': username,

'name': user['name'],

'token': token.decode('UTF-8'),

'exp': exp\_clean,

'admin': user['admin']

})

else:

return response(401, 'Incorrect password.')

else:

return response(401, 'Incorrect username.')

return response(401, 'Authentication required.')

def logout():

token = None

if 'x-access-token' in request.headers:

token = request.headers['x-access-token']

if not token:

return response(401, 'Token required.')

blacklist.insert\_one({'token': token})

return response(200, 'Logout successful.')

def register():

def is\_null(value):

return len(value) == 0

name = request.form['name']

username = request.form['username']

email = request.form['email']

password = request.form['password']

if is\_null(name):

return response(400, 'Name cannot be null.')

if is\_null(username):

return response(400, 'Username cannot be null.')

if is\_null(email):

return response(400, 'Email cannot be null.')

if is\_null(password):

return response(400, 'Password cannot be null.')

elif len(password) < 8:

return response(400, 'Password must be longer than 8 characters.')

user\_list = users.find({'username': username})

for u in user\_list:

if u['username'] == username:

return response(400, 'A user already exists with the username you provided.')

user = {

'name': name,

'email': email,

'username': username.lower(),

'password': hashpw(str(password).encode('UTF-8'), gensalt()),

'admin': False,

'bookmarks': []

}

users.insert\_one(user)

return response(200, 'Signup successful.')

**C:\Users\rob\Documents\GitHub\RecipeDB\api\bookmarks.py**

from pymongo import MongoClient

from config import MONGO

from bson import ObjectId

from util import response, trim

client = MongoClient(MONGO)

db = client.RecipeDB

recipes = db.recipes

users = db.users

def bookmark\_recipe(\_id, \_user):

user = users.find\_one({'\_id': ObjectId(\_user['\_id'])}, {'bookmarks': 1})

for bookmark in user['bookmarks']:

if bookmark['recipeId'] == \_id:

return response(202, 'This recipe is already in your bookmarks.')

bookmark = {

'\_id': ObjectId(),

'recipeId': \_id

}

users.update\_one({'\_id': ObjectId(\_user['\_id'])}, {'$push': {'bookmarks': bookmark}})

return response(201, 'Bookmark added.')

def unbookmark\_recipe(\_id, \_user):

user = users.find\_one({'\_id': ObjectId(\_user['\_id'])}, {'bookmarks': 1})

i = 0

removed = False

while i < len(user['bookmarks']):

bookmark = user['bookmarks'][i]

if str(bookmark['recipeId']) == \_id:

users.update\_one({'\_id': ObjectId(\_user['\_id'])}, {'$pull': {'bookmarks': {'\_id': bookmark['\_id']}}})

removed = True

if i + 1 == len(user['bookmarks']):

if removed:

return response(200, 'Bookmark removed.')

else:

return response(202, 'Not bookmarked.')

i += 1

return response(202, 'Recipe not bookmarked.')

def get\_bookmarks(\_id):

result = users.find\_one({'\_id': ObjectId(\_id)}, {'bookmarks': 1})

bookmark\_list = []

for bookmark in result['bookmarks']:

r = recipes.find\_one({'\_id': ObjectId(bookmark['recipeId'])},

{'title': 1, 'desc': 1, 'rating': 1, 'calories': 1})

if r is not None:

r['\_id'] = str(r['\_id'])

r['desc'] = trim(r['desc'], 160)

bookmark\_list.append(r)

else:

users.update\_one({'\_id': ObjectId(\_id)}, {'$pull': {'bookmarks': {'\_id': bookmark['\_id']}}})

return response(200, bookmark\_list)

**C:\Users\rob\Documents\GitHub\RecipeDB\api\comments.py**

from flask import request

from pymongo import MongoClient

from config import MONGO

from bson import ObjectId

from datetime import datetime

from util import response

client = MongoClient(MONGO)

db = client.RecipeDB

recipes = db.recipes

def get\_recipe\_comments(\_id):

result = recipes.find\_one({'\_id': ObjectId(\_id)}, {'comments': 1, '\_id': 0})

return\_data = []

if result is not None:

for comment in result['comments']:

comment['\_id'] = str(comment['\_id'])

return\_data.append(comment)

return response(200, return\_data)

def new\_recipe\_comment(\_id, user):

comment = {

'\_id': ObjectId(),

'user\_id': user['\_id'],

'user\_name': user['username'],

'body': request.form['body'],

'date': datetime.utcnow()

}

recipes.update\_one({'\_id': ObjectId(\_id)}, {'$push': {'comments': comment}})

return get\_recipe\_comments(\_id)

def update\_recipe\_comment(\_id, \_user):

try:

current\_comment = recipes.find\_one({'comments.\_id': ObjectId(\_id)}, {'\_id': 1, 'comments.$': 1})

if \_user['\_id'] != current\_comment['comments'][0]['user\_id']:

return response(400, 'This is not your comment to update.')

except:

return response(400, 'No comment found.')

updated\_comment = {

'comments.$.user\_id': current\_comment['comments'][0]['user\_id'],

'comments.$.body': request.form['body'],

'comments.$.date': datetime.utcnow()

}

recipes.update\_one({'comments.\_id': ObjectId(\_id)}, {'$set': updated\_comment})

return get\_recipe\_comments(current\_comment['\_id'])

def delete\_recipe\_comment(\_id, user\_id):

result = recipes.find\_one({'comments.\_id': ObjectId(\_id)}, {'comments.$': 1, '\_id': 1})

if result is not None:

recipe\_id = str(result['\_id'])

result = result['comments']

if len(result) == 1:

result = result[0]

if result['user\_id'] == user\_id:

recipes.update\_one({'\_id': ObjectId(recipe\_id)}, {'$pull': {'comments': {'\_id': ObjectId(result['\_id'])}}})

return get\_recipe\_comments(recipe\_id)

else:

return response(403, 'This is not your comment to delete.')

return response(404, 'No item found')

**C:\Users\rob\Documents\GitHub\RecipeDB\api\config.py**

SECRET\_KEY = '2c05f68c5b4b7a4088ff3561028252e95314f672ac8bcda016a9792e8dc35092'

ITEMS\_PER\_PAGE = 15

MONGO = 'mongodb://127.0.0.1:27017'

MONGO\_INDEXES = [

{

'name': 'search\_index',

'field': 'title'

}

]

HEADERS = {

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36'

}

**C:\Users\rob\Documents\GitHub\RecipeDB\api\recipes.py**

from flask import request

from pymongo import MongoClient

from config import ITEMS\_PER\_PAGE, MONGO, HEADERS

from bson import ObjectId

from random import shuffle, randrange

from bs4 import BeautifulSoup

from requests import get

from math import ceil

from util import response, trim

client = MongoClient(MONGO)

db = client.RecipeDB

recipes = db.recipes

users = db.users

def get\_recipe(\_id, \_user):

recipe = recipes.find\_one({'\_id': ObjectId(\_id)}, {'comments': 0})

if recipe is not None:

recipe['\_id'] = str(recipe['\_id'])

if \_user is not None:

user = users.find\_one({'\_id': ObjectId(\_user['\_id'])}, {'bookmarks': 1})

if user is not None:

if len(user['bookmarks']) > 0:

for bookmark in user['bookmarks']:

if bookmark['recipeId'] == \_id:

recipe['bookmarked'] = True

if recipe.get('bookmarked') is None:

recipe['bookmarked'] = False

return response(200, recipe)

else:

return response(404, 'No recipe found')

# unused endpoint

def get\_recipes():

page\_num = 1

if request.args.get('p'):

page\_num = int(request.args.get('p'))

start = ITEMS\_PER\_PAGE \* (page\_num - 1)

recipe\_list = []

results = recipes.find({}, {'title': 1, 'desc': 1, 'rating': 1, 'calories': 1}).skip(start).limit(ITEMS\_PER\_PAGE)

for recipe in results:

recipe['\_id'] = str(recipe['\_id'])

recipe['desc'] = trim(recipe['desc'], 160)

recipe\_list.append(recipe)

return response(200, recipe\_list)

def get\_top\_recipes():

results = recipes.find({'rating': 5}, {'title': 1, 'desc': 1, 'rating': 1, 'calories': 1}).limit(100)

recipe\_list = []

for recipe in results:

recipe['\_id'] = str(recipe['\_id'])

recipe['desc'] = trim(recipe['desc'], 160)

recipe\_list.append(recipe)

shuffle(recipe\_list)

return response(200, recipe\_list[:6])

def search\_recipes():

criteria = str(request.args.get('criteria'))

if len(criteria) != 0:

page\_num = 1

if request.args.get('p'):

page\_num = int(request.args.get('p'))

total = recipes.count\_documents({'title': {'$regex': criteria, "$options": "-i"}}, None)

page\_count = ceil(total / ITEMS\_PER\_PAGE)

if page\_num > page\_count:

page\_num = page\_count

start = ITEMS\_PER\_PAGE \* (page\_num - 1)

if start < 0:

start = 0

recipe\_list = []

for r in recipes.find({'title': {'$regex': criteria, "$options": "-i"}},

{'title': 1, 'desc': 1, 'rating': 1, 'calories': 1}).skip(start).limit(ITEMS\_PER\_PAGE):

r['\_id'] = str(r['\_id'])

r['desc'] = trim(r['desc'], 160)

recipe\_list.append(r)

return response(200, {

'data': recipe\_list,

'page': page\_num,

'pageCount': page\_count,

'perPage': ITEMS\_PER\_PAGE,

'total': total

})

else:

return response(400, 'No search criteria provided.')

def new\_recipe():

url = request.form['url']

if 'bbcgoodfood.com' not in url:

return response(400, 'Not a BBC link.')

if url is not None:

data = get(url, headers=HEADERS)

soup = BeautifulSoup(data.content, 'html.parser')

recipe = {

'\_id': ObjectId(),

'title': soup.find(class\_='recipe-header\_\_title').getText(),

'desc': soup.find(class\_='recipe-header\_\_description').getText(),

'comments': [],

'rating': randrange(2, 5),

'categories': []

}

for i in soup.find(class\_='nutrition'):

label = i.find(class\_='nutrition\_\_label').getText()

value = i.find(class\_='nutrition\_\_value').getText().replace('g', '')

if label == 'fat':

recipe['fat'] = value

elif label == 'protein':

recipe['protein'] = value

elif label == 'salt':

# 5g salt = 2000mg sodium (\* 400)

recipe['sodium'] = round(float(value) \* 400)

elif label == 'kcal':

recipe['calories'] = value

ingredients = []

for i in soup.find(class\_='ingredients-list\_\_group'):

ingredients.append(i.getText())

recipe['ingredients'] = ingredients

directions = []

for i in soup.find(class\_='method\_\_list'):

directions.append(i.getText())

recipe['directions'] = directions

recipes.insert\_one(recipe)

return response(200, {'inserted': str(recipe['\_id'])})

else:

return response(400, 'No suitable url provided.')

def delete\_recipe(\_id):

recipes.delete\_one({'\_id': ObjectId(\_id)})

return response(200, 'Recipe deleted.')

**C:\Users\rob\Documents\GitHub\RecipeDB\api\util\\_\_init\_\_.py**

from flask import make\_response, jsonify

def response(status\_code, data):

if isinstance(data, str):

if 199 < status\_code < 300:

data = {'message': data}

else:

data = {'error': data}

return make\_response(jsonify(data), status\_code)

def trim(value, length):

return (value[:length + 2] + '..') if len(value) > length else value

**C:\Users\rob\Documents\GitHub\RecipeDB\api\util\data\createUsers.py**

from pymongo import MongoClient

from bcrypt import hashpw, gensalt

client = MongoClient("mongodb://127.0.0.1:27017")

db = client.RecipeDB

users = db.users

data = [

{

'name': 'Rob Wilkie',

'username': 'rob',

'password': b'password1',

'email': 'rob@wilkie.io',

'bookmarks': [],

'admin': True

},

{

'name': 'Lola the Cat',

'username': 'lola',

'password': b'cat',

'email': 'lola@wilkie.io',

'bookmarks': [],

'admin': False

}

]

for new\_user in data:

new\_user["password"] = hashpw(new\_user["password"], gensalt())

users.insert\_one(new\_user)

**C:\Users\rob\Documents\GitHub\RecipeDB\api\util\data\importData.py**

import json

from pymongo import MongoClient

client = MongoClient('mongodb://127.0.0.1:27017')

db = client.RecipeDB

recipes = db.recipes

mongo\_data = []

with open('../../../data/full\_format\_recipes.json') as file:

content = json.load(file)

di = True

for recipe in content:

desc = recipe.get('desc')

calories = recipe.get('calories')

fat = recipe.get('fat')

protein = recipe.get('protein')

sodium = recipe.get('sodium')

if desc and sodium and fat and calories and protein:

recipe['comments'] = []

recipe['title'] = recipe.get('title').strip()

if len(recipe['directions']) == 1:

directions = recipe['directions'][0]

recipe['directions'] = directions.split('. ')

mongo\_data.append(recipe)

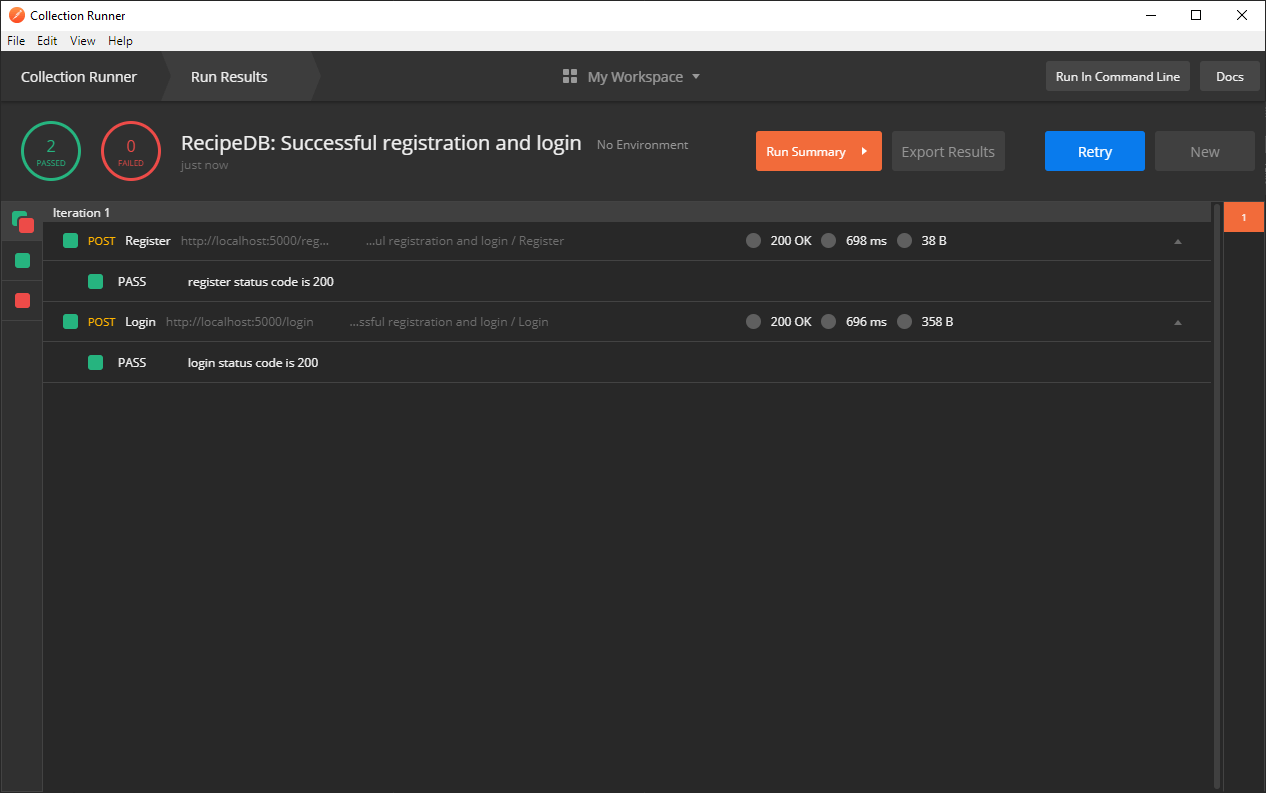
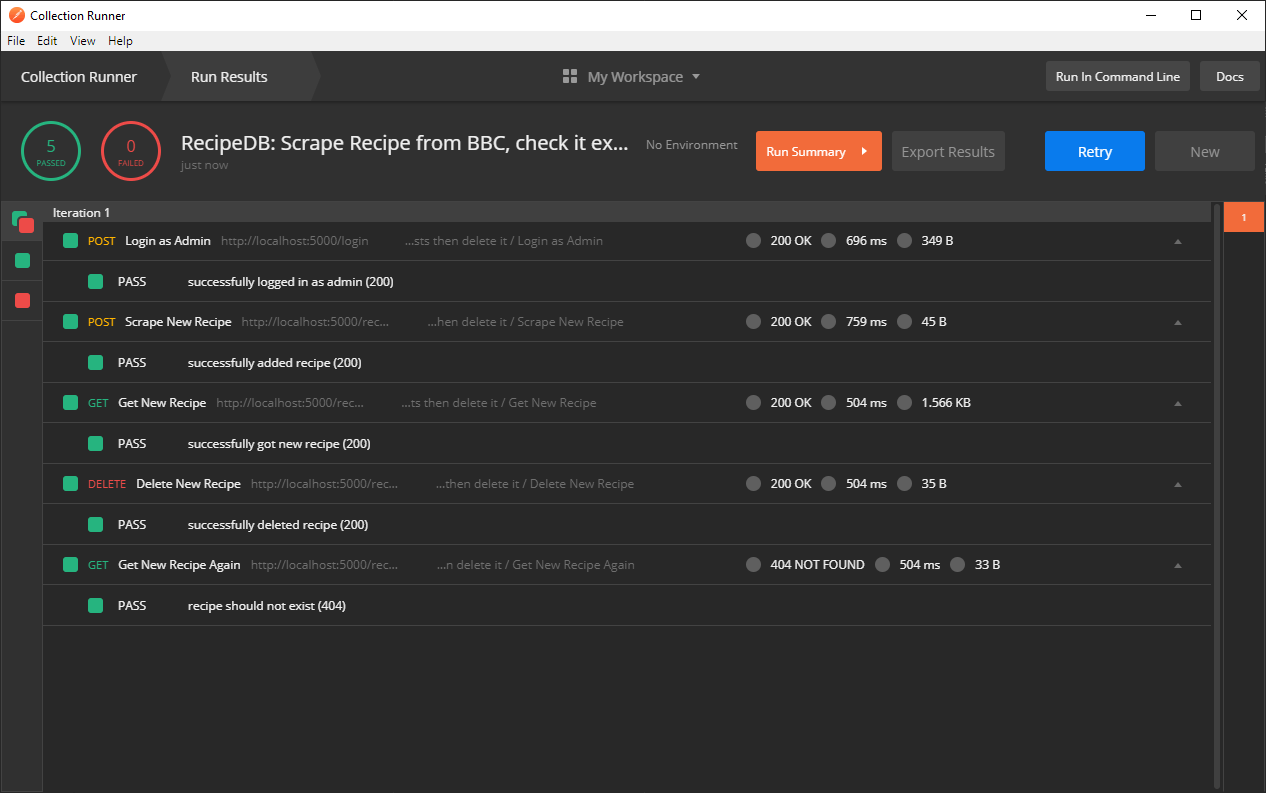
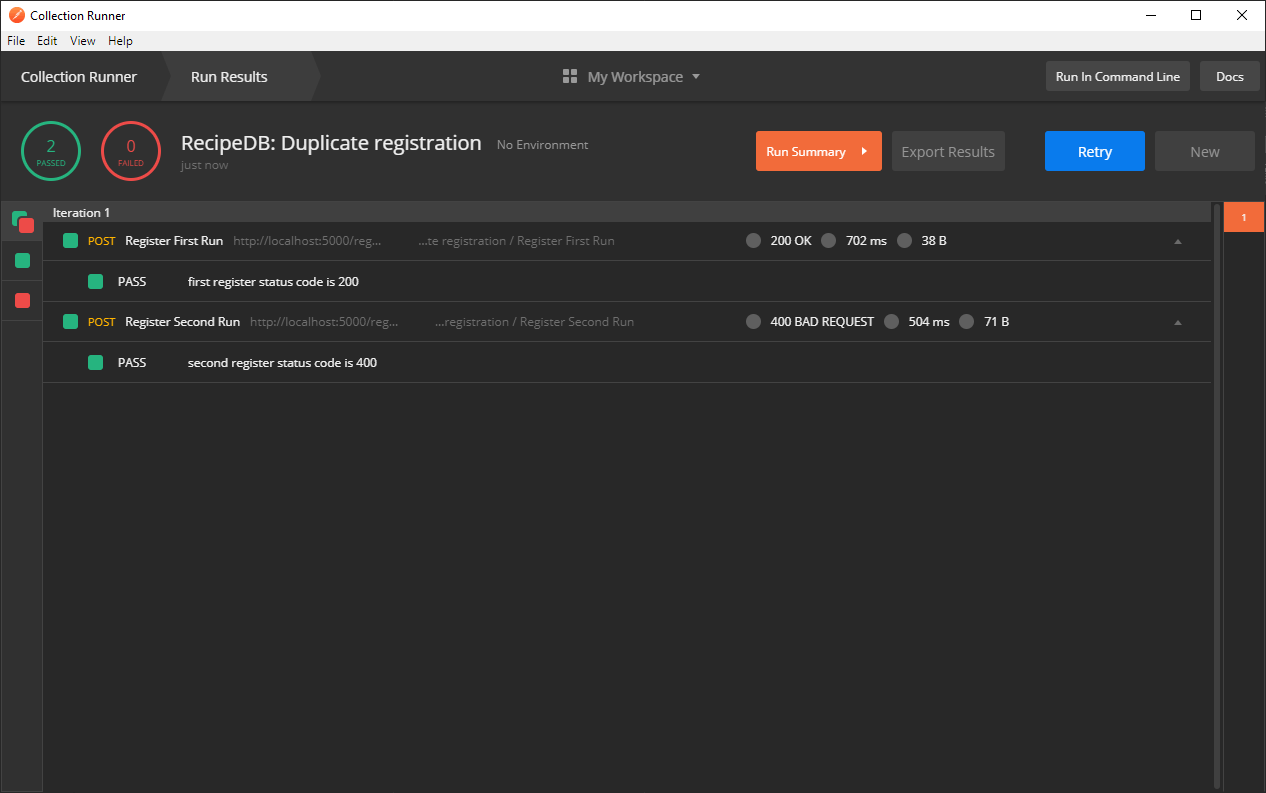
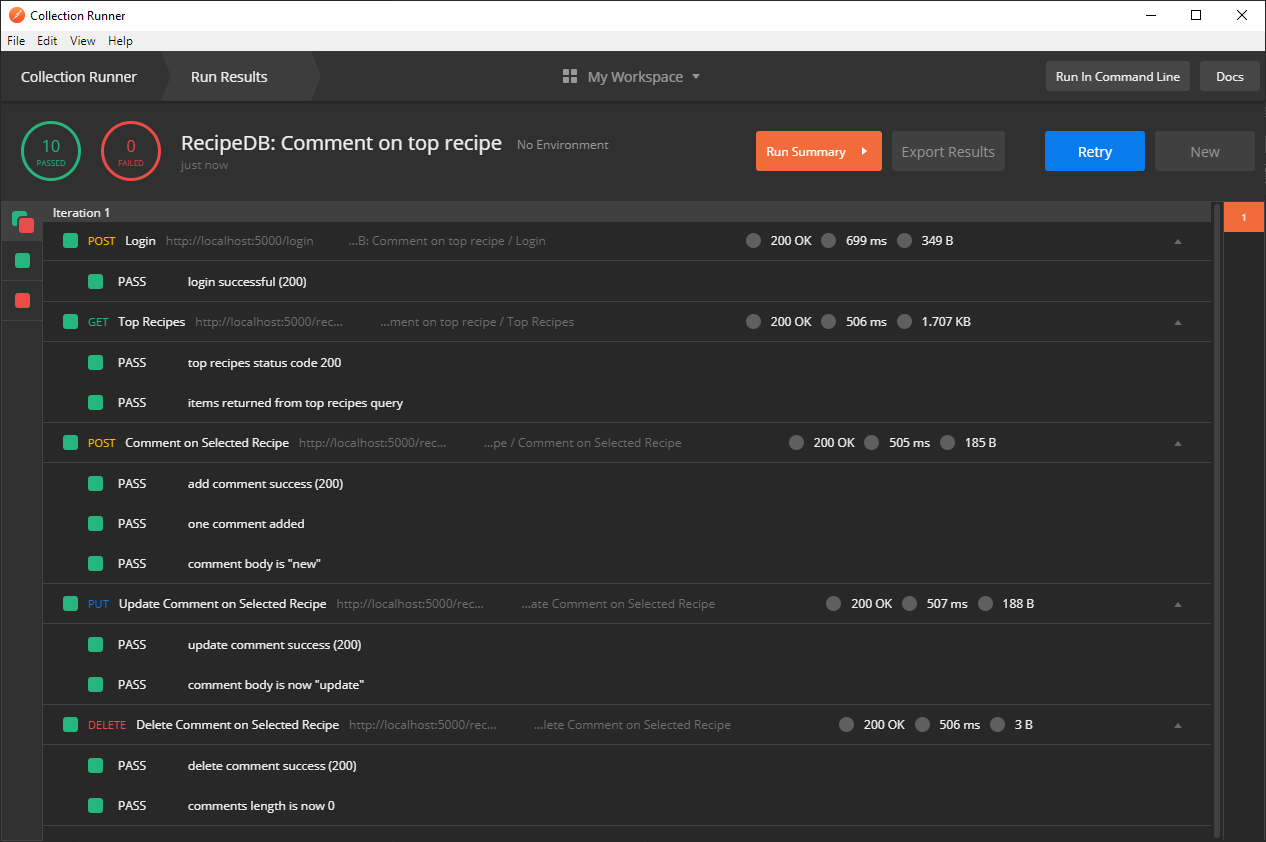
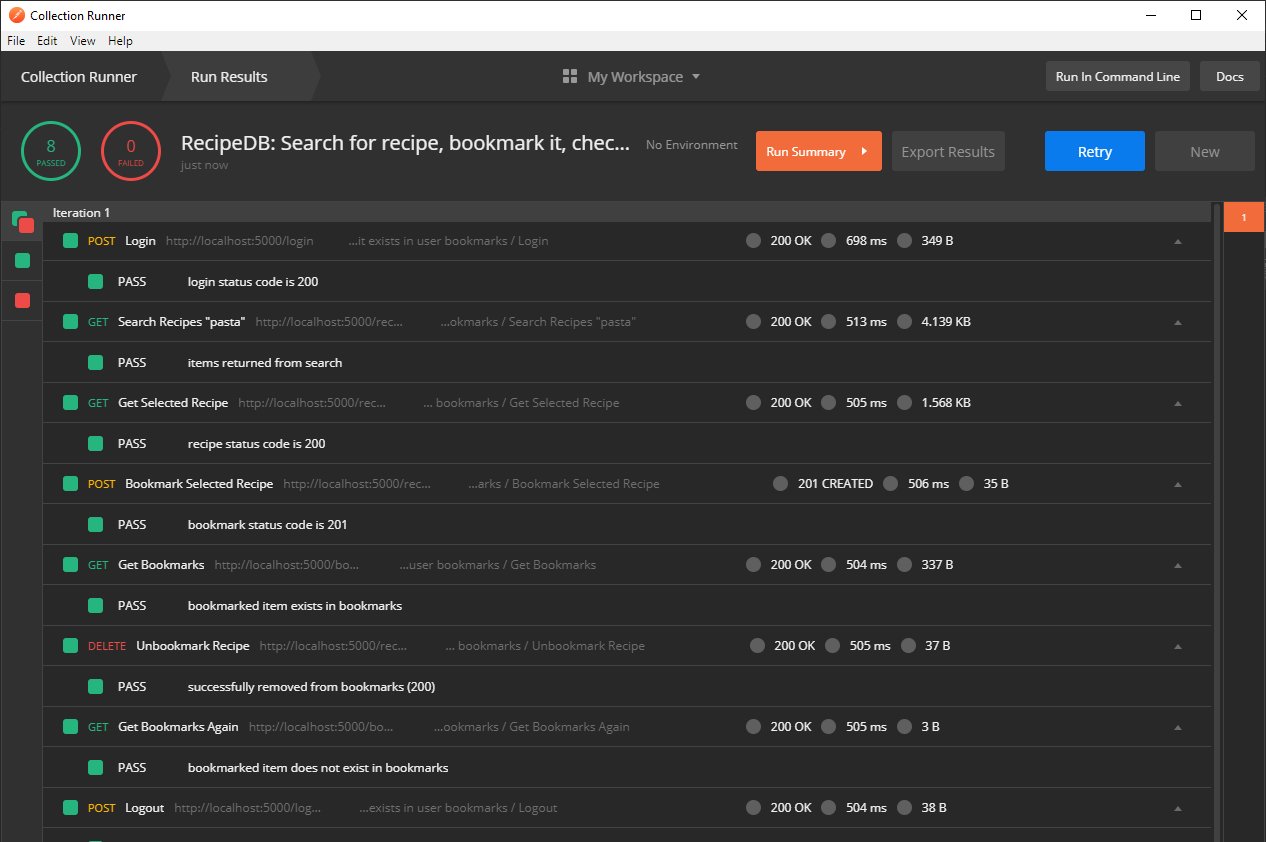
recipes.insert\_many(mongo\_data)

print(f'{len(content) - len(mongo\_data)} items removed due to insufficient data.')

## Endpoint Summary

|  |
| --- |
| **POST /register**  Registers a new user in the RecipeDB system. |
| **POST /login**  Logs user into RecipeDB system. Requires a valid username and password, returns a JWT for use in future API calls. |
| **POST /logout**  Logs user out of RecipeDB system, blacklists the given JWT; preventing further use of it. |
| **GET /recipes/top**  Gets 6 randomly selected recipes which have a rating of 5 (maximum). |
| **GET /recipes/search**  Searches recipes for any which title matches the given criteria. |
| **POST /recipes**  Adds a new recipe to the database, requires a valid bbcgoodfood.com recipe URL, scrapes the recipe and returns the ID of the inserted recipe. |
| **GET /recipe/<id>**  Gets a recipe by ID. |
| **DELETE /recipe/<id>**  Deletes a given recipe from the database by ID. |
| **GET /recipe/<id>/comments**  Gets all comments for a given recipe, by RECIPE ID. |
| **POST /recipe/<id>/comments**  Adds a new comment to a given recipe, by RECIPE ID. Returns updated list of comments for this recipe. |
| **PUT /recipe/<id>/comments**  Updates a comment on a given recipe, by COMMENT ID. Only updates comment if the comment belongs to the same user as the JWT used in this request. Returns updated list of comments for this recipe. |
| **DELETE /recipe/<id>/comments**  Deletes a comment on a given recipe, by COMMENT ID. Only deletes comment if the comment belongs to the same user as the JWT used in this request. Returns updated list of comments for this recipe. |
| **GET /bookmarks**  Gets a user's full list of bookmarks. |
| **POST /recipe/<id>/bookmark**  Adds a given recipe to a user's bookmarks. |
| **DELETE /recipe/<id>/bookmark**  Removes a given recipe from a user's bookmarks. |

## Tests



## API Summary

Full documentation available at: <https://documenter.getpostman.com/view/9830786/SWEDzZd5?version=latest>

**POST /login**

http://localhost:5000/login

Logs user into RecipeDB system. Requires a valid username and password, returns a JWT for use in future API calls.

**Headers**

|  |  |
| --- | --- |
| Content-Type | application/x-www-form-urlencoded |

**Body**

urlencoded

|  |  |
| --- | --- |
| username | rob  User's username |
| password | password1  User's password |

Example Response200 OK

{

"admin": true,

"exp": "2019-12-20T18:13:48.794088",

"id": "5df419c2cf0d96d5a135acd2",

"name": "Rob Wilkie",

"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1ZGY0MTljMmNmMGQ5NmQ1YTEzNWFjZDIiLCJ1c2VybmFtZSI6InJvYiIsImFkbWluIjp0cnVlLCJleHAiOjE1NzY4NjU2Mjh9.JEVVJvjQqM8EjIlDt\_ABjbkjBSkvMhMMz9xnO-6O9Zw",

"username": "rob"

}

**POST /register**

http://localhost:5000/register

Registers a new user in the RecipeDB system, all fields are required.

**Headers**

|  |  |
| --- | --- |
| Content-Type | application/x-www-form-urlencoded |

**Body**

urlencoded

|  |  |
| --- | --- |
| name | Rob Wilkie  User's full name. |
| email | rob@wilkie.io  User's email address. |
| username | wllkle  User's username. |
| password | password1  User's password, must be longer than 8 characters. |

Example Response200 OK

{

"message": "Signup successful."

}

**POST /logout**

http://localhost:5000/logout

Logs user out of RecipeDB system, blacklists the given JWT; preventing further use of it.

**Headers**

|  |  |
| --- | --- |
| x-access-token | eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJfaWQiOiI1ZGY0MTljMmNmMGQ5NmQ1YTEzNWFjZDIiLCJ1c2VybmFtZSI6InJvYiIsImFkbWluIjp0cnVlLCJleHAiOjE1NzY4NjA4NjB9.YBrTA1HaQysRVvsZIME\_Vl10rSxAST-uiE\_eHoUKmJw  User's current JWT. |

Example Response200 OK

{

"message": "Logout successful."

}

**GET /recipes/top**

http://localhost:5000/recipes/top

Gets 6 randomly selected recipes which have a rating of 5 (maximum).

Example Response200 OK

[

{

"\_id": "5dfb063572714fbb11f0fcf6",

"calories": 399,

"desc": "Charred to a crisp: Not a statement usually associated with salads, but delicious nonetheless in this Mexican-inspired romaine number.",

"rating": 5,

"title": "Charred Romaine with Tomatillo Dressing"

},

{

"\_id": "5dfb063572714fbb11f0fcb3",

**GET /recipes/search**

http://localhost:5000/recipes/search?criteria=pizza

Searches recipes for any which title matches the given criteria.

**Params**

|  |  |
| --- | --- |
| criteria | pizza  Search criteria used to find matches in recipe titles. |

Example Response200 OK

{

"data": [

{

"\_id": "5dfb063572714fbb11f0fb85",

"calories": 695,

"desc": "This recipe can be prepared in 45 minutes or less.",

"rating": 3.75,

"title": "Tossed Pizza Salad"

},

{

**GET /recipe/<id>**

http://localhost:5000/recipe/5dfbab3faecb1b082e88fe69

Gets a recipe by ID.

**Headers**

|  |  |
| --- | --- |
| x-access-token | {{token}}  Optional header, when present returns whether the recipe is bookmarked by the user. |

Example Response200 OK

{

"\_id": "5dfbab3faecb1b082e88fe69",

"bookmarked": false,

"calories": "445",

"categories": [],

"desc": "A Cajun-inspired rice pot recipe with spicy Spanish sausage, sweet peppers and tomatoes",

"directions": [

"Heat 1 tbsp olive oil in a large frying pan with a lid and brown 2 chopped chicken breasts for 5-8 mins until golden.",

"Remove and set aside. Tip in the 1 diced onion and cook for 3-4 mins until soft.",

"Add 1 thinly sliced red pepper, 2 crushed garlic cloves, 75g sliced chorizo and 1 tbsp Cajun seasoning, and cook for 5 mins more.",

**DELETE /recipe/<id>**

http://localhost:5000/recipe/5dfbab3faecb1b082e88fe69

Deletes a given recipe from the database by ID.

**Headers**

|  |  |
| --- | --- |
| x-access-token | {{admin\_token}}  JWT of an admin-level user. |

Example Response200 OK

{

"message": "Recipe deleted."

}

**GET /recipe/<id>/comments**

http://localhost:5000/recipe/5dfb063572714fbb11f11589/comments

Gets all comments for a given recipe, by RECIPE ID.

Example Response200 OK

[

{

"\_id": "5dfbcd7cba7c69b9757b87b3",

"body": "This was great, everyone loved it.",

"date": "Thu, 19 Dec 2019 19:20:28 GMT",

"user\_id": "5df419c2cf0d96d5a135acd2",

"user\_name": "rob"

}

]

**POST /recipe/<id>/comments**

http://localhost:5000/recipe/5dfb063572714fbb11f11589/comments

Adds a new comment to a given recipe, by RECIPE ID. Returns updated list of comments for this recipe.

**Headers**

|  |  |
| --- | --- |
| x-access-token | {{token}}  JWT belonging to commenting user. |
| Content-Type | application/x-www-form-urlencoded |

**Body**

urlencoded

|  |  |
| --- | --- |
| body | Tasty!  Comment string, must not be null or empty. |

Example Response200 OK

[

{

"\_id": "5dfbcd7cba7c69b9757b87b3",

"body": "This was great, everyone loved it.",

"date": "Thu, 19 Dec 2019 19:20:28 GMT",

"user\_id": "5df419c2cf0d96d5a135acd2",

"user\_name": "rob"

},

{

"\_id": "5dfbd052ba7c69b9757b87b4",

**DELETE /recipe/<id>/comments**

http://localhost:5000/recipe/5dfbcd7cba7c69b9757b87b3/comments

Deletes a comment on a given recipe, by COMMENT ID. Only deletes comment if the comment belongs to the same user as the JWT used in this request. Returns updated list of comments for this recipe.

**Headers**

|  |  |
| --- | --- |
| x-access-token | {{token}}  JWT belonging user deleting their comment. |

Example Response200 OK

[

{

"\_id": "5dfbd052ba7c69b9757b87b4",

"body": "Tasty!",

"date": "Thu, 19 Dec 2019 19:32:34 GMT",

"user\_id": "5df419c2cf0d96d5a135acd2",

"user\_name": "rob"

}

]

**PUT /recipe/<id>/comments**

http://localhost:5000/recipe/5dfbcd7cba7c69b9757b87b3/comments

Updates a comment on a given recipe, by COMMENT ID. Only updates comment if the comment belongs to the same user as the JWT used in this request. Returns updated list of comments for this recipe.

**Headers**

|  |  |
| --- | --- |
| x-access-token | {{token}}  JWT belonging user deleting their comment. |
| Content-Type | application/x-www-form-urlencoded |

**Body**

urlencoded

|  |  |
| --- | --- |
| body | Really tasty!  Updated comment string, must not be null or empty. |

Example Response200 OK

[

{

"\_id": "5dfbd052ba7c69b9757b87b4",

"body": "Tasty!",

"date": "Thu, 19 Dec 2019 19:32:34 GMT",

"user\_id": "5df419c2cf0d96d5a135acd2",

"user\_name": "rob"

}

]

**POST /recipe/<id>/bookmark**

http://localhost:5000/recipe/5dfb063572714fbb11f0fbd8/bookmark

Adds a given recipe to a user's bookmarks.

**Headers**

|  |  |
| --- | --- |
| x-access-token | {{token}}  JWT of user bookmarking the recipe. |

Example Response201 CREATED

{

"message": "Bookmark added."

}

**DELETE /recipe/<id>/bookmark**

http://localhost:5000/recipe/5dfb063572714fbb11f0fbd8/bookmark

Removes a given recipe from a user's bookmarks.

**Headers**

|  |  |
| --- | --- |
| x-access-token | {{token}}  JWT of user removing the recipe from their bookmarks. |

Example Response200 OK

{

"message": "Bookmark removed."

}

**GET /bookmarks**

http://localhost:5000/bookmarks

Gets a user's full list of bookmarks.

**Headers**

|  |  |
| --- | --- |
| x-access-token | {{token}}  JWT belonging to user accessing their bookmarks. |

Example Request/bookmarks

curl --location --request GET 'http://localhost:5000/bookmarks' \

--header 'x-access-token: {{token}}'

Example Response200 OK

[

{

"\_id": "5dfb063572714fbb11f0fbd8",

"calories": 687,

"desc": "The yogurt helps tenderize the chicken; the garlic, ginger, and spices in the marinade infuse it with lots of flavor.",

"rating": 5,

"title": "Chicken Tikka Masala"

},

{

"\_id": "5dfb063572714fbb11f108ce",

**POST /recipes**

http://localhost:5000/recipes

Adds a new recipe to the database, requires a valid bbcgoodfood.com recipe URL, scrapes the recipe and returns the ID of the inserted recipe.

**Headers**

|  |  |
| --- | --- |
| x-access-token | {{token}}  JWT belonging to user adding recipe to system, must be an admin. |
| Content-Type | application/x-www-form-urlencoded |

**Body**

urlencoded

|  |  |
| --- | --- |
| url | https://www.bbcgoodfood.com/recipes/beef-vegetable-casserole  Valid URL for a bbcgoodfood.com recipe. |

Example Response200 OK

{

"inserted": "5dfbd469ba7c69b9757b87b8"

}