

RecipeDB Backend

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

Collection Runner

File Edit View Help

Collection RunnerRun ResultsMy WorkspaceRun In Command LineDocs

8 PASSED0 FAILED

RecipeDB: Search for recipe, bookmark it, chec...No Environment

Run SummaryExport ResultsRetryNew

Iteration 1

POST	Login	http://localhost:5000/login	...it exists in user bookmarks / Login	200 OK	698 ms	349 B
PASS	login status code is 200					
GET	Search Recipes "pasta"	http://localhost:5000/rec...	...okmarks / Search Recipes "pasta"	200 OK	513 ms	4.139 KB
PASS	items returned from search					
GET	Get Selected Recipe	http://localhost:5000/rec...	...bookmarks / Get Selected Recipe	200 OK	505 ms	1.568 KB
PASS	recipe status code is 200					
POST	Bookmark Selected Recipe	http://localhost:5000/rec...	...arks / Bookmark Selected Recipe	201 CREATED	506 ms	35 B
PASS	bookmark status code is 201					
GET	Get Bookmarks	http://localhost:5000/bo...	...user bookmarks / Get Bookmarks	200 OK	504 ms	337 B
PASS	bookmarked item exists in bookmarks					
DELETE	Unbookmark Recipe	http://localhost:5000/rec...	...bookmarks / Unbookmark Recipe	200 OK	505 ms	37 B
PASS	successfully removed from bookmarks (200)					
GET	Get Bookmarks Again	http://localhost:5000/bo...	...bookmarks / Get Bookmarks Again	200 OK	505 ms	3 B
PASS	bookmarked item does not exist in bookmarks					
POST	Logout	http://localhost:5000/log...	...exists in user bookmarks / Logout	200 OK	504 ms	38 B

Collection Runner

File Edit View Help

Collection RunnerRun ResultsMy WorkspaceRun In Command LineDocs

10 PASSED0 FAILED

RecipeDB: Comment on top recipeNo Environment

Run SummaryExport ResultsRetryNew

Iteration 1

POST	Login	http://localhost:5000/login	...B: Comment on top recipe / Login	200 OK	699 ms	349 B
PASS	login successful (200)					
GET	Top Recipes	http://localhost:5000/rec...	...ment on top recipe / Top Recipes	200 OK	506 ms	1.707 KB
PASS	top recipes status code 200					
PASS	items returned from top recipes query					
POST	Comment on Selected Recipe	http://localhost:5000/rec...	...pe / Comment on Selected Recipe	200 OK	505 ms	185 B
PASS	add comment success (200)					
PASS	one comment added					
PASS	comment body is "new"					
PUT	Update Comment on Selected Recipe	http://localhost:5000/rec...	...ate Comment on Selected Recipe	200 OK	507 ms	188 B
PASS	update comment success (200)					
PASS	comment body is now "update"					
DELETE	Delete Comment on Selected Recipe	http://localhost:5000/rec...	...lete Comment on Selected Recipe	200 OK	506 ms	3 B
PASS	delete comment success (200)					
PASS	comments length is now 0					

Collection Runner

File Edit View Help

Collection Runner Run Results My Workspace Run In Command Line Docs

2 PASSED 0 FAILED RecipeDB: Duplicate registration No Environment just now Run Summary Export Results Retry New

Iteration 1

POST	Register First Run	http://localhost:5000/reg...	...te registration / Register First Run	200 OK	702 ms	38 B
PASS	first register status code is 200					
POST	Register Second Run	http://localhost:5000/reg...	...registration / Register Second Run	400 BAD REQUEST	504 ms	71 B
PASS	second register status code is 400					

Collection Runner

File Edit View Help

Collection Runner Run Results My Workspace Run In Command Line Docs

5 PASSED 0 FAILED RecipeDB: Scrape Recipe from BBC, check it ex... No Environment just now Run Summary Export Results Retry New

Iteration 1

POST	Login as Admin	http://localhost:5000/login	...sts then delete it / Login as Admin	200 OK	696 ms	349 B
PASS	successfully logged in as admin (200)					
POST	Scrape New Recipe	http://localhost:5000/rec...	...hen delete it / Scrape New Recipe	200 OK	759 ms	45 B
PASS	successfully added recipe (200)					
GET	Get New Recipe	http://localhost:5000/rec...	...ts then delete it / Get New Recipe	200 OK	504 ms	1.566 KB
PASS	successfully got new recipe (200)					
DELETE	Delete New Recipe	http://localhost:5000/rec...	...then delete it / Delete New Recipe	200 OK	504 ms	35 B
PASS	successfully deleted recipe (200)					
GET	Get New Recipe Again	http://localhost:5000/rec...	...n delete it / Get New Recipe Again	404 NOT FOUND	504 ms	33 B
PASS	recipe should not exist (404)					

Collection Runner

File Edit View Help

Collection RunnerRun ResultsMy WorkspaceRun In Command LineDocs

2 PASSED0 FAILED

RecipeDB: Successful registration and loginNo Environmentjust now

Run SummaryExport ResultsRetryNew

Iteration 1

1

POST Registerhttp://localhost:5000/register...ul registration and login / Register200 OK698 ms38 B

PASSregister status code is 200

POST Loginhttp://localhost:5000/login...ssful registration and login / Login200 OK696 ms358 B

PASSlogin status code is 200

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.eyJfaWQiOiI1ZGY0MTljMmNmMGQ5NmQ1YTZlZWJjZDIiLCJ1c2VybmFtZSI6ImInJvYiIsImFkbWluIjp0cnV1LCJleHAiOjE1NzY4NjU2Mjh9.JEVVJvjQqM8EjIldt_ABjbkjBSkvMhMMz9xn0-6O9Zw",
  "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

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.eyJfaWQiOiI1ZGY0MTljMmNmMGQ5NmQ1YTEzNWFiZDliLWUiE_eHoUKmJw
	User's current JWT.

Example Response200 OK

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",
    "recipe": "5dfb063572714fbb11f11589",
    "comment": "This recipe is amazing!"
  }
]
```

```
    "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 Requestbookmarks

```
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"
}
```