FITNESS FULLSTACK APPLICATION-BACKEND

Initializing the Database

//__init__.py

```
from flask import Flask, jsonify, request
from flask_cors import CORS
from pymongo import MongoClient
import json
import http.client


app = Flask(__name__)
CORS(app)

app.config.from_mapping(
     SECRET_KEY="Secret_key ",

 )
# Connect to MongoDB
mongo = MongoClient("mongodb://localhost:27017/")
db = mongo.HealthFitness
users_collection = db['users']
Categories=db.Categories
Fitness_Program=db.Fitness_program



from App import routes
```

routes.py

```
from bson import ObjectId
from flask import render_template, request, jsonify, make_response,session
```

```python
from flask_bcrypt import Bcrypt
from App import app, users_collection,Categories,Fitness_Program
from App import Fitness
import requests


bcrypt = Bcrypt(app)

@app.route("/", methods=['GET'])
def index():
    return render_template("index.html")

@app.route("/register", methods=["POST"])
def register():
    data = request.json
    username = data.get('username')
    email = data.get('email')
    password = data.get('password')
    if users_collection.find_one({'email': email}):
        return make_response(jsonify({'error': 'Email already exists'}), 400)

    # Hash the password
    hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')

    # Register the user by inserting into the database
    new_user = {'username': username,
                'email': email,
                'password': hashed_password}
    users_collection.insert_one(new_user)

    # Return success message
    return make_response(jsonify({'message': 'User registered successfully'}),
201)




@app.route("/login", methods=['POST'])
def login():
    data = request.get_json()
    email = data.get('email')
    password = data.get('password')


    user = users_collection.find_one({"email": email})
```

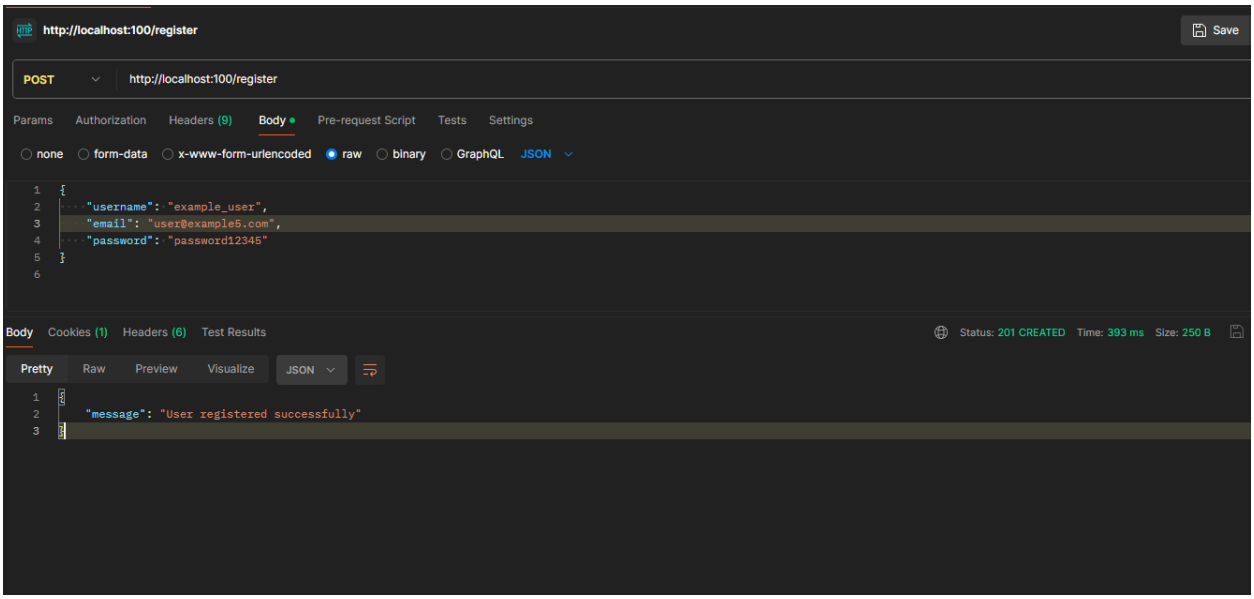
```
    if user:

        if bcrypt.check_password_hash(user['password'], password):

            session['email'] = user['email']

            return make_response(jsonify({'message': 'Login successful'}), 200)
        else:

            return make_response(jsonify({'error': 'Incorrect password'}), 401)
    else:

        return make_response(jsonify({'error': 'User does not exist'}), 404)

# body parts to get fit (subjective) to bodypart
@app.route("/categories/bodypart", methods=["GET"])
def get_body_part():
    exercise_api=Fitness.ExerciseAPI()
    body_part_list = exercise_api.get_body_part_list()

    return jsonify({"body_parts": body_part_list})




#exercises for specific body part
@app.route("/exercise/<body_part>", methods=["GET"])
def get_body_part_exercise(body_part):
    exercise_api=Fitness.ExerciseAPI()

    try:
        exercise_data = exercise_api.get_body_part_exercises(body_part, limit=15)
        return make_response(exercise_data, 200)  # Assuming exercise_data is
already JSON
    except Exception as e:
        return make_response(jsonify({"error": str(e)}), 500)


# Route to create a new fitness program
@app.route("/fitness_program", methods=["POST"])
def create_fitness_program():
    data = request.json
    program_name = data.get('programName')
    body_part = data.get('bodyPart')
```

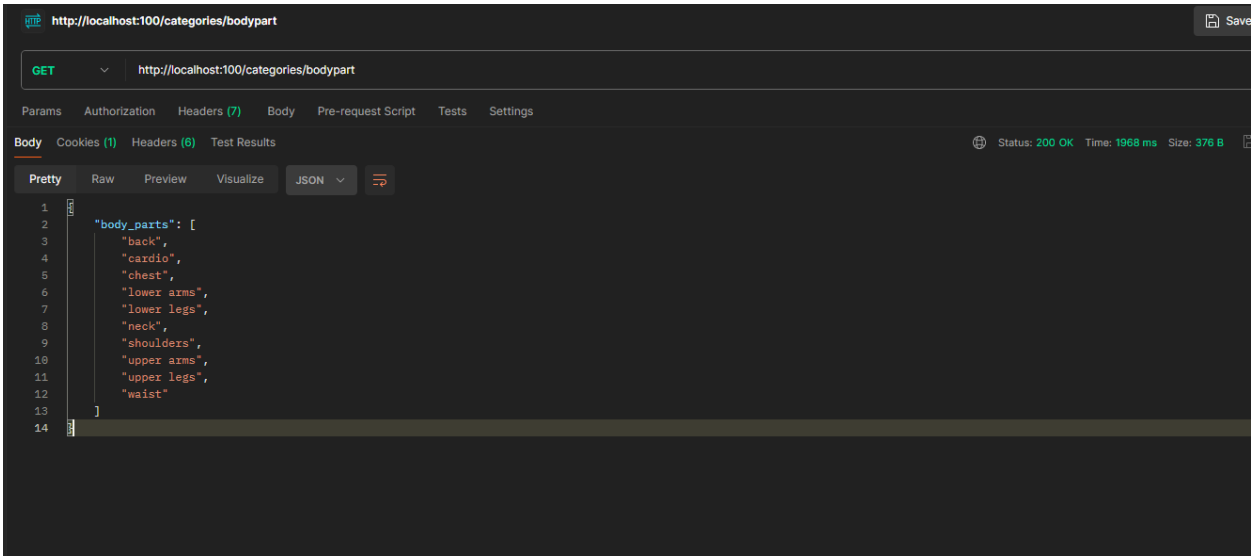
```
    user_id = data.get('userId')  # Get user ID from request
    hours_per_week = data.get('hoursPerWeek')
    days_per_week = data.get('daysPerWeek')
    fitness_program = {
        "user_id": user_id,
        "program_name": program_name,
        "body_part": body_part,
        "hours_per_week": hours_per_week,
        "days_per_week": days_per_week,
        "progress": 0
    }
    Fitness_Program.insert_one(fitness_program)
    return jsonify({"message": "Fitness program created successfully"}), 201



@app.route("/fitness_programs", methods=["GET"])
def get_fitness_programs():
    fitness_programs = list(Fitness_Program.find({}))

    # Convert ObjectId fields to string representations
    for program in fitness_programs:
        program['_id'] = str(program['_id'])

    # Return just the list of fitness programs
    return jsonify(fitness_programs), 200

# Route to update an existing fitness program
@app.route("/fitness_program/<program_id>", methods=["PUT"])
def update_fitness_program(program_id):
    data = request.json
    program_name = data.get('programName')
    body_part = data.get('bodyPart')
    hours_per_week = data.get('hoursPerWeek')  # Get updated hours per week
    days_per_week = data.get('daysPerWeek')    # Get updated days per week

    # Construct the update query including hours_per_week and days_per_week
    update_query = {"$set": {"program_name": program_name, "body_part":
body_part,
                             "hours_per_week": hours_per_week, "days_per_week":
days_per_week}}

    # Update the fitness program based on the program_id
    Fitness_Program.update_one({"_id": ObjectId(program_id)}, update_query)
```

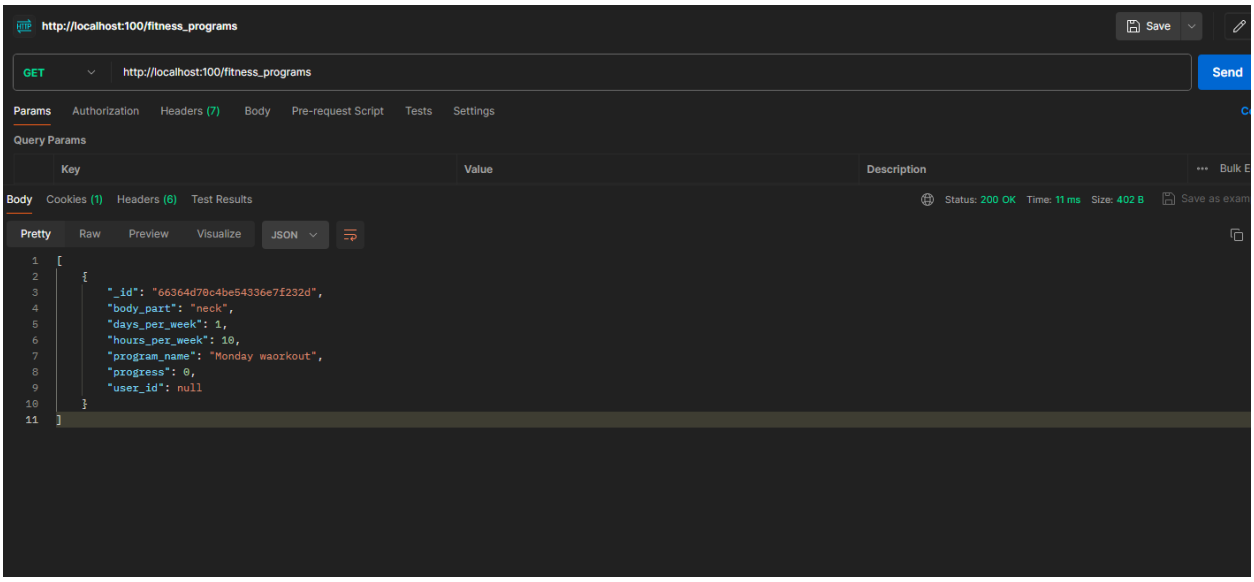
```
    return jsonify({"message": "Fitness program updated successfully"}), 200


# Route to delete an existing fitness program
@app.route("/fitness_program/<program_id>", methods=["DELETE"])
def delete_fitness_program(program_id):
    Fitness_Program.delete_one({"_id": ObjectId(program_id)})
    return jsonify({"message": "Fitness program deleted successfully"}), 200



@app.route("/fitness_programs/count", methods=["GET"])
def get_fitness_programs_count():
    total_count = Fitness_Program.count_documents({})
    return jsonify({"total_count": total_count}), 200
```

```
from App import app

if __name__ == "__main__":
    app.run(debug=True, port = 100)
```

Postman Tests







General Steps

1. Instantiating the MongoDB database.
    - This involved setting up the database collections.
2. Configure the API
    - This involved creating endpoints e.g, REGISTER, LOGIN, CREATE, DELETE.
    - The database details are needed here to ensure connection to our mongo collections.
    - The endpoints are created using flask.
    - The server is instantiated using the command Script/Activate in a virtual environment to demonstrate communication with the database.
    - Postman is used for testing the endpoints during development.
3. Creating the Front-End

- Angular 17 requires strict code modularization.
- Stand-alone components have been created for each page.
- Services have been used to abstract functionality away.
- The Front-End contains Imagery and graphics fetched through the API.

4. Running on our Front-End Server