Microservice Architecture Programming
Practical 2

Submitted by:
Ishan Tewari
18BCE080

Task: Implementing REST API using any preferred language
Language Chosen: Python
Framework Chosen: Flask

Code:

1. settings.py: Contains the code for setting up the app and database.

```
from flask import Flask, jsonify, request, Response
from flask_sqlalchemy import SQLAlchemy

# creating flask app
app = Flask(__name__)

# configuring the database
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

2. movies.py: Contains code for setting up the database and methods for interacting with the database

```
from settings import *
import json

# initializing the database
db = SQLAlchemy(app)

# creating movie class corresponding to the movies table in our database
class Movie(db.Model):
   __tablename__ = 'movies' # giving the table a name

   id = db.Column(db.Integer, primary_key= True)
   title = db.Column(db.String(80), nullable= False)
   year = db.Column(db.Integer, nullable= False)
   genre = db.Column(db.String(80), nullable= False)

   # function to display output as json
   def json(self):
       return {
           'id': self.id,
           'title': self.title,
           'year': self.year,
           'genre': self.genre
       }

   # function to add a movie to the database
```

```
    def add_movie(_title, _year, _genre):
        movie = Movie(title = _title,year = _year,genre = _genre)
        db.session.add(movie)
        db.session.commit()

    # function to return all the movies in the database
    def get_all_movies():
        return [Movie.json(movie) for movie in Movie.query.all()]

    # function to return the movie by id
    def get_movie(_id):
        return [Movie.json(Movie.query.filter_by(id = _id).first())]

    # function to update the movie using id
    def update_movie(_id, _title, _year, _genre):
        movie_to_update = Movie.query.filter_by(id = _id).first()
        movie_to_update.title = _title
        movie_to_update.year = _year
        movie_to_update.genre = _genre
        db.session.commit()

    # function to delete a movie from database
    def delete_movie(_id):
        Movie.query.filter_by(id = _id).delete()
        db.session.commit()
```

3. api.py: Contains code for performing REST API calls and CRUD Operations

```
from operator import ge
from movies import *

# read
@app.route('/movies', methods=['GET'])
def get_all_movies():
   return jsonify({'Movies': Movie.get_all_movies()})

# read
@app.route('/movies/<int:id>', methods=['GET'])
def get_movie(id):
   query_result = Movie.get_movie(id)
   return jsonify(query_result)

# create
```

```python
@app.route('/movies', methods=['POST'])
def add_movie():

   # request data of movie to be added from the client
   request_data = request.get_json()

   # add the movie with data given by client
   Movie.add_movie(
       request_data['title'],
       request_data['year'],
       request_data['genre']
   )

   # give response on successfull addition
   reponse = Response("Movie Added",201, mimetype="application/json")
   return reponse

# update
@app.route('/movies/<int:id>', methods=['PUT'])
def update_movie(id):

   # request data of movie to be added from the client
   request_data = request.get_json()

   # update the movie with data given by client
   Movie.update_movie(
       id,
       request_data['title'],
       request_data['year'],
       request_data['genre']
   )

   # give response on successfull updation
   reponse = Response("Movie Updated",200, mimetype="application/json")
   return reponse

# delete
@app.route('/movies/<int:id>', methods=['DELETE'])
def delete_movie(id):
   Movie.delete_movie(id)

   # give response on successfull deletion
```

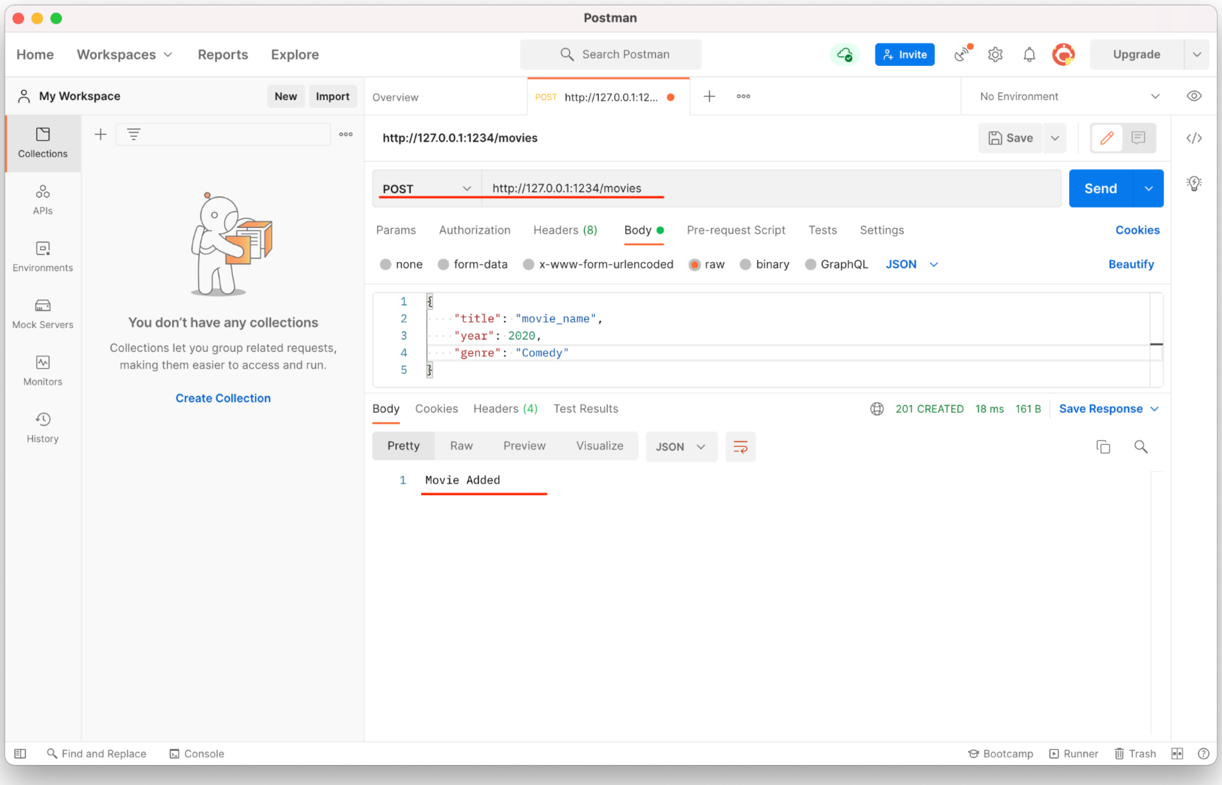
```
    reponse = Response("Movie Deleted",200, mimetype="application/json")
    return reponse



if __name__ == "__main__":
    app.run(port=1234, debug=True)
```

REST API calls using CRUD Operations:

1. CREATE:



POST http://127.0.0.1:1234/movies

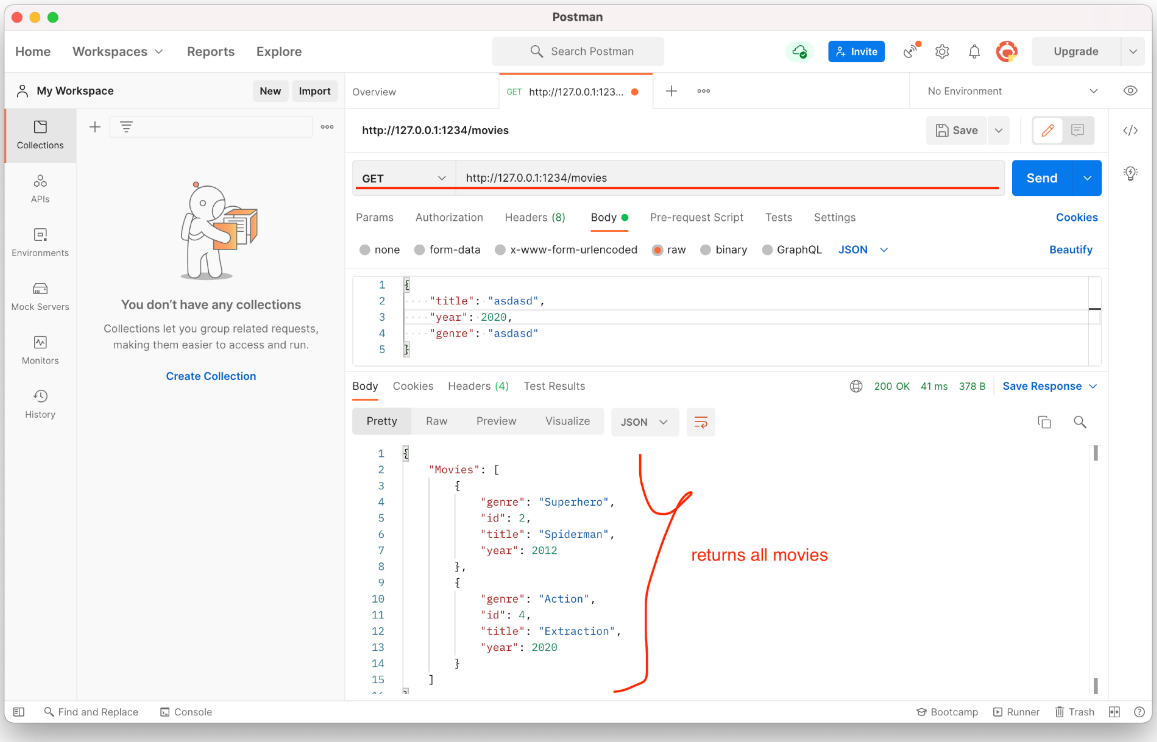
```
{
    "title": "movie_name",
    "year": 2020,
    "genre": "Comedy"
}
```

201 CREATED 18 ms 161 B

```
Movie Added
```

2. READ:

   a. Get all movies



http://127.0.0.1:1234/movies

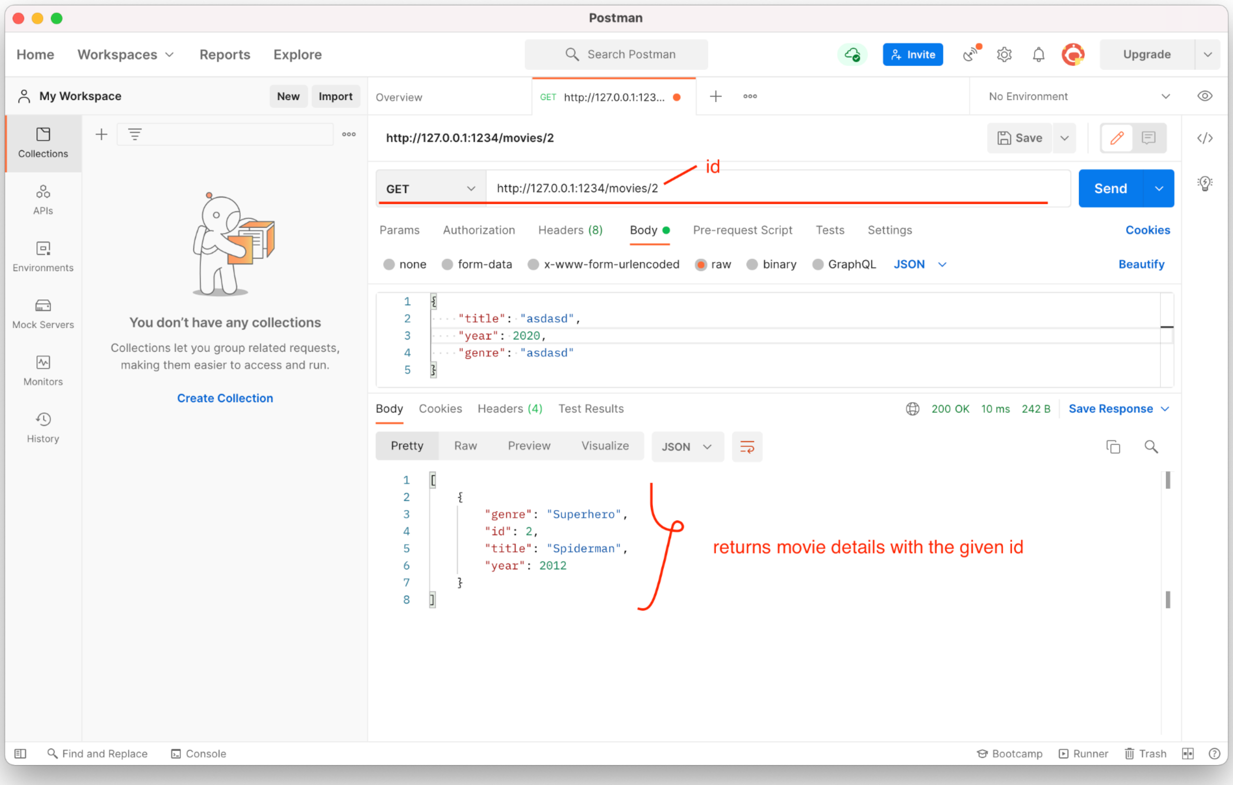
GET http://127.0.0.1:1234/movies

```
{
    "title": "asdasd",
    "year": 2020,
    "genre": "asdasd"
}
```

200 OK 41 ms 378 B

```
{
    "Movies": [
        {
            "genre": "Superhero",
            "id": 2,
            "title": "Spiderman",
            "year": 2012
        },
        {
            "genre": "Action",
            "id": 4,
            "title": "Extraction",
            "year": 2020
        }
    ]
```

   b. Get movie with a specific id



http://127.0.0.1:1234/movies/2

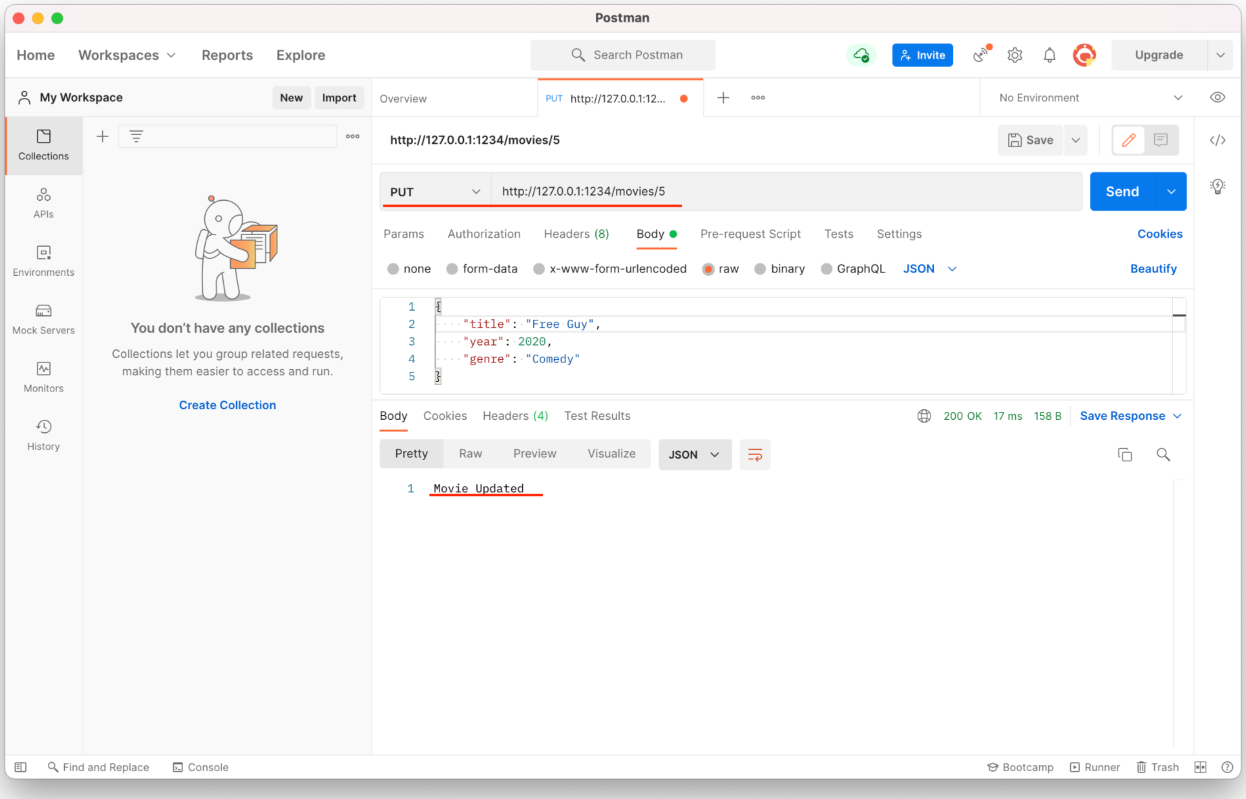
GET http://127.0.0.1:1234/movies/2

```
{
    "title": "asdasd",
    "year": 2020,
    "genre": "asdasd"
}
```

200 OK 10 ms 242 B

```
[
    {
        "genre": "Superhero",
        "id": 2,
        "title": "Spiderman",
        "year": 2012
    }
]
```

3. UPDATE:



http://127.0.0.1:1234/movies/5

PUT http://127.0.0.1:1234/movies/5

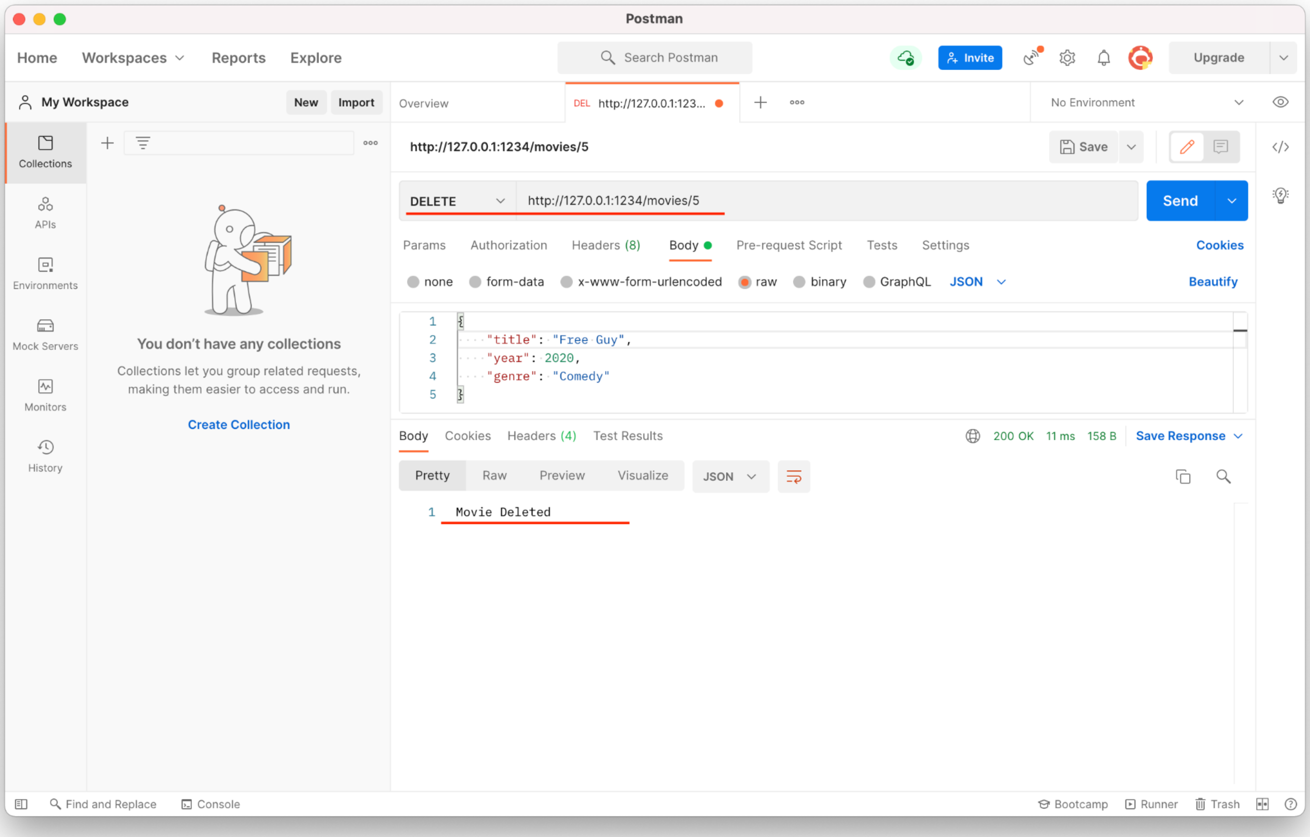
Body: raw JSON

```
{
    "title": "Free Guy",
    "year": 2020,
    "genre": "Comedy"
}
```

Body — 200 OK 17 ms 158 B

```
Movie Updated
```

4. DELETE:



http://127.0.0.1:1234/movies/5

DELETE http://127.0.0.1:1234/movies/5

Body: raw JSON

```
{
    "title": "Free Guy",
    "year": 2020,
    "genre": "Comedy"
}
```

Body — 200 OK 11 ms 158 B

```
Movie Deleted
```