CS301-Software Engineering

Name: Shashwat Chandel

Roll No: 21BCS108

**Goal**: Our aim to create a cultural heritage destination that connects our specified target audience, celebrates the cultures of India and helps to provide a platform for emerging talents.

Therefore to achieve and be able to build a digital solution for the above aim we need to start of by following a few steps:

1. **Reasearch**: We first collect information about the cultural heritages of India, our target audience, existing cultural spaces, ect. This research would basically help us identify where the gap exists so that we can work on bridging that gap.
2. **Planning:** Now as we have completed the research and basic understanding of the requirements of each of the target groups, we can start working on the planning of financial projections, marketing strategy, operational plan, and the digital technology solutions that would be required to achieve the objectives.
3. **Digital Technology Solutions:**

This is the major part where all the features and aspects of our project are discussed and worked upon.

The key features that our digital solution can have are:

1. Event Management feature:

This is a feature that is required for help users organize and manage performances, events, exhibitions, ect.

CODE:

from flask import Flask, request

import requests

app = Flask(\_\_name\_\_)

# Define API endpoints

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

def get\_events():

    # Access public events API

    response = requests.get('https://api.publicapis.org/entries')

    events = response.json()

    return events

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

def create\_event():

    # Create a new event

    event\_data = request.json

    # Add event to database

    # ...

    return {'message': 'Event created successfully'}

@app.route('/events/<id>', methods=['PUT'])

def update\_event(id):

    # Update an existing event

    event\_data = request.json

    # Update event in database

    # ...

    return {'message': 'Event updated successfully'}

@app.route('/events/<id>', methods=['DELETE'])

def delete\_event(id):

    # Delete an event

    # Delete event from database

    # ...

    return {'message': 'Event deleted successfully'}

# Define testing functions

def test\_get\_events():

    response = app.test\_client().get('/events')

    assert response.status\_code == 200

    assert response.json is not None

def test\_create\_event():

    response = app.test\_client().post('/events', json={'name': 'New Event'})

    assert response.status\_code == 200

    assert response.json == {'message': 'Event created successfully'}

def test\_update\_event():

    response = app.test\_client().put('/events/1', json={'name': 'Updated Event'})

    assert response.status\_code == 200

    assert response.json == {'message': 'Event updated successfully'}

def test\_delete\_event():

    response = app.test\_client().delete('/events/1')

    assert response.status\_code == 200

    assert response.json == {'message': 'Event deleted successfully'}

# Run tests

test\_get\_events()

test\_create\_event()

test\_update\_event()

test\_delete\_event()

# Run application

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

    app.run(debug=True)

The above is the code for Event Scheduling and Management feature of our digital solution along with the testing code and use of API to access public domains and the data.

So let us understand each line of code above in depth:

from flask import Flask, request

import requests

First we import the required files in our code.

The “Flask module” is a lightweight web application framework used for building web applications.

The “request module” is used for handling incoming HTTP requests.

The “requests module” is used for making HTTP requests to other APIs.

app = Flask(\_\_name\_\_)

By the above code I am creating a new instance of the Flask application with the current module name as the argument.

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

def get\_events():

    # Access public events API

    response = requests.get('https://api.publicapis.org/entries')

    events = response.json()

    return events

In Flask, @app.route() decorator is used to bind a URL which is called when the specified URL is requested by a client.

The “methods” parameter is used to specify the allowed HTTP methods for each endpoint.

As we can see in the above “methods” parameter is set to ['GET'], which means that this endpoint only allows HTTP GET requests.

“get\_events()” function which is executed when a GET request is sent to the /events endpoint. The function sends an HTTP GET request to a public API, receives the response, converts the response into a JSON format, and returns it.

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

def create\_event():

    # Create a new event

    event\_data = request.json

    # Add event to database

    # ...

    return {'message': 'Event created successfully'}

The above is the code for creating an event. We set the methods parameter as POST. So, a POST request is sent to the /events endpoint. Our function would receive a JSON payload from the request and it would create a new event in the database and at the end display a message to indicate that the event was created successfully.

@app.route('/events/<id>', methods=['PUT'])

def update\_event(id):

    # Update an existing event

    event\_data = request.json

    # Update event in database

    # ...

    return {'message': 'Event updated successfully'}

When a PUT request is sent to the “/events/<id>” endpoint, where <id> is the unique identifier of the event being updated. The function receives a JSON payload from the request, and it accordingly updates the event in the database.

@app.route('/events/<id>', methods=['DELETE'])

def delete\_event(id):

    # Delete an event

    # Delete event from database

    # ...

    return {'message': 'Event deleted successfully'}

The above function gets executed when a DELETE request is sent to the “/events/<id>” endpoint, where <id> is the unique identifier of the event being deleted. The function deletes the event from the database.

1. NEXT IS THE TESTING CODE:

def test\_get\_events():

    response = app.test\_client().get('/events')

    assert response.status\_code == 200

    assert response.json is not None

def test\_create\_event():

    response = app.test\_client().post('/events', json={'name': 'New Event'})

    assert response.status\_code == 200

    assert response.json == {'message': 'Event created successfully'}

def test\_update\_event():

    response = app.test\_client().put('/events/1', json={'name': 'Updated Event'})

    assert response.status\_code == 200

    assert response.json == {'message': 'Event updated successfully'}

def test\_delete\_event():

    response = app.test\_client().delete('/events/1')

    assert response.status\_code == 200

    assert response.json == {'message': 'Event deleted successfully'}

So we test the get\_event(), create\_event(), update\_event(), delete\_event() all these using the above 4 functions.

These are a series of automated tests for an API that handles events. Each test sends a request to the API and checks whether the response is as expected.

1. Test\_get\_events():

This test sends a GET request to the '/events' endpoint and checks whether the response status code is 200 (OK) and whether the response body is not empty (i.e., response.json is not None).

1. Test\_create\_event():

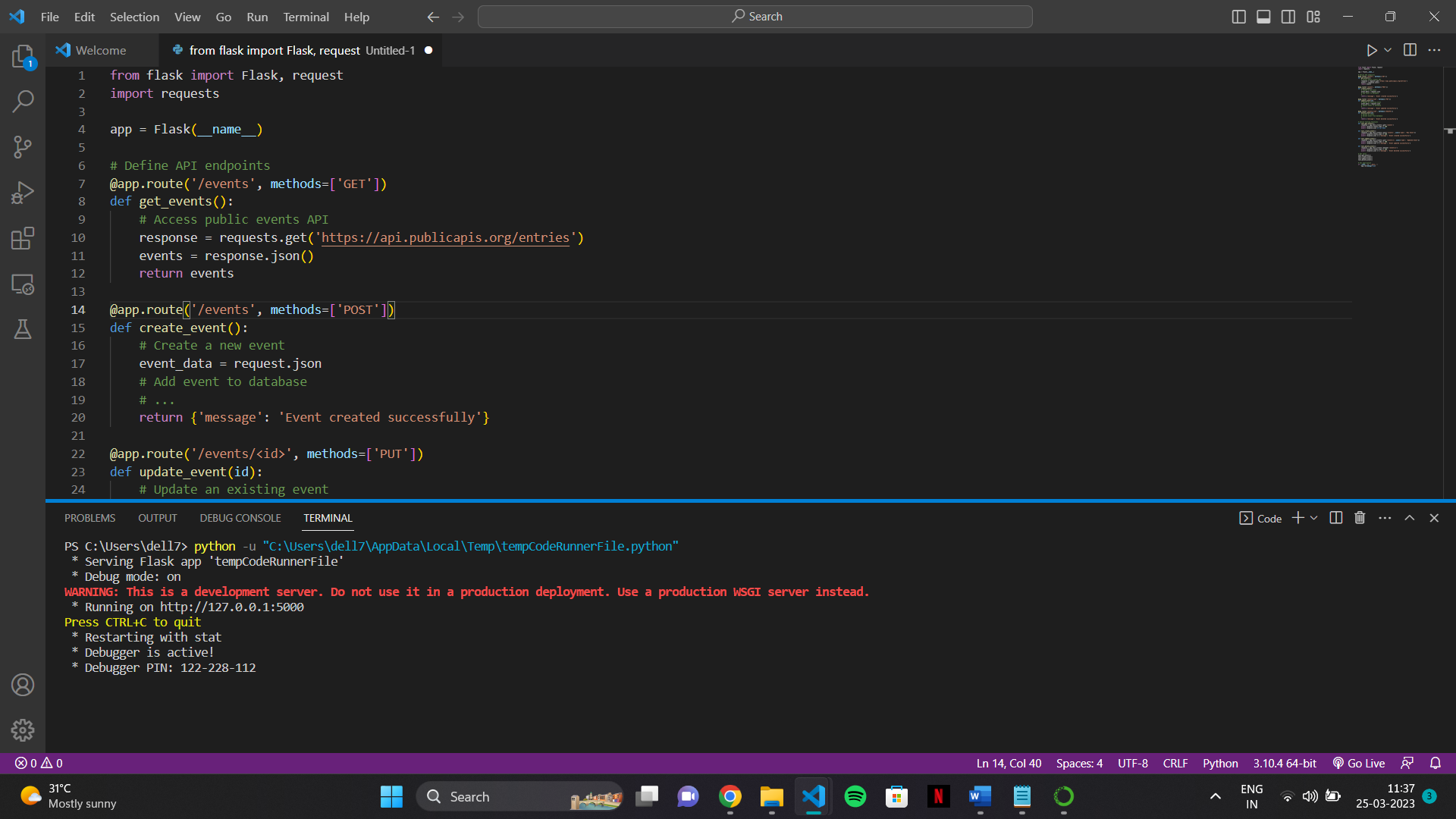
This test sends a POST request to the '/events' endpoint with a JSON payload containing the name of a new event. It checks whether the response status code is 200 and whether the response body is a JSON object containing a message indicating that the event was created successfully.

1. Test\_update\_event():

This test sends a PUT request to the '/events/1' endpoint with a JSON payload containing a new name for the event with ID 1. It checks whether the response status code is 200 and whether the response body is a JSON object containing a message indicating that the event was updated successfully.

1. Test\_delete\_event():

This test sends a DELETE request to the '/events/1' endpoint to delete the event with ID 1. It checks whether the response status code is 200 and whether the response body is a JSON object containing a message indicating that the event was deleted successfully.



The above is what will be seen in my console when I run the code.

So, the above image tells that the code is successfully executed and the flask is currently running on the local server. The debugger is also active which we can use to debug our code.