Name:-Rakshit Somkuwar

Regno:-21Bcs122

**Theme : Create new cultural destination to celebrate the heritage of India and provide a platform for emerging Talents using Digital Technology solutions**

**Aim :**

* Creating doors for a first-of-its-kind, multi-disciplinary space for the Arts in cities
* Encourage Visual art space and captivating array of public art
* Bring together communities through a dynamic programming of epic theatricals , regional theatre, music , dance , spoken word etc.
* Major attraction is to provide a platform for emerging talent and showcases the vibrance of India’s heritage
* Generate source of income for the Art communities through collaborations, aggregators and accelerators investments

**Target audiences :**

* Home to Art, Artists, the audience from India and around the world.

**Assignment scope :**

1. Identify various requirements for the above program initiative that can be developed as a digital solutions
2. Use ChatGPT platform an generate code for the above requirements
   1. Generate code and run the program in Goggle Colab/Jupiter Notebook/Visual Code/PyCharm
   2. Perform integrated testing. Add integration testing code in the same program.
3. Modify the same program. Write APIs to access the data from the public domain and test the program for regression testing the same program

**Deliverables :**

Working Program with test scripts embedded in the same program.

Here's an example Python code that could be used as the backbone of a digital platform for emerging talents in India:

from flask import Flask, render\_template, request, redirect, url\_for

import sqlite3

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

# Connect to the database

conn = sqlite3.connect('cultural\_destinations.db')

c = conn.cursor()

# Create a table to store information about the talents

c.execute('''CREATE TABLE IF NOT EXISTS talents

(id INTEGER PRIMARY KEY,

name TEXT NOT NULL,

category TEXT NOT NULL,

description TEXT NOT NULL,

image TEXT NOT NULL,

video TEXT NOT NULL,

email TEXT NOT NULL)''')

# Define routes for the app

@app.route('/')

def index():

# Display a list of talents

talents = c.execute('SELECT \* FROM talents').fetchall()

return render\_template('index.html', talents=talents)

@app.route('/talent/new', methods=['GET', 'POST'])

def new\_talent():

if request.method == 'POST':

# Add a new talent to the database

name = request.form['name']

category = request.form['category']

description = request.form['description']

image = request.form['image']

video = request.form['video']

email = request.form['email']

c.execute('INSERT INTO talents (name, category, description, image, video, email) VALUES (?, ?, ?, ?, ?, ?)',

(name, category, description, image, video, email))

conn.commit()

return redirect(url\_for('index'))

else:

# Display a form for adding a new talent

return render\_template('new\_talent.html')

# Run the app

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

app.run()

This code creates a Flask app that connects to a SQLite database to store information about emerging talents in India. The app has two routes:

/ displays a list of talents stored in the database.

/talent/new allows users to add a new talent to the database by filling out a form.

This app could be deployed to a server and used as a platform for emerging talents in India to showcase their work and connect with potential collaborators and audiences. The app could also be expanded to include additional features, such as a search function or a social media integration.

Identify various requirements for the above program initiative that can be developed as a digital solutions

Here are some possible requirements for a digital solution that could support the program initiative of creating a new cultural destination in India and providing a platform for emerging talents:

1. User Authentication: The platform should have a secure user authentication system to allow only registered users to access the content and features of the platform. The authentication system should be easy to use and allow users to create and manage their profiles.
2. Content Management System: The platform should have a content management system that allows administrators and users to easily add, edit, and delete content, including text, images, and videos.
3. Talent Management System: The platform should have a talent management system that allows users to submit their profiles, including their name, category, description, image, video, and contact information, to be displayed on the platform. The talent management system should be easy to use and allow users to manage their profiles.
4. Search Functionality: The platform should have a search functionality that allows users to search for talents based on different criteria, such as category, location, and keywords.
5. Collaboration Tools: The platform should have collaboration tools that allow users to connect and collaborate with other users, such as messaging, commenting, and sharing features.
6. Social Media Integration: The platform should have social media integration that allows users to share their content and profiles on social media platforms, such as Facebook, Twitter, and Instagram.
7. Analytics and Reporting: The platform should have analytics and reporting features that allow administrators to track user activity and engagement, such as number of users, page views, and content downloads.
8. Mobile Compatibility: The platform should be mobile-friendly and compatible with different devices and platforms, including desktops, laptops, tablets, and smartphones.
9. Payment Gateway: If the platform plans to generate revenue from its services, it should have a payment gateway integration that allows users to pay for premium features or content, such as sponsorships, advertisements, and premium profiles.
10. Scalability and Security: The platform should be scalable and secure, able to handle large amounts of traffic and data, and protected against security threats, such as hacking, phishing, and data breaches.
11. An online platform that empowers emerging artists to showcase their work, connect with potential collaborators and investors, and reach a wider audience
12. A comprehensive digital archive of past performances and exhibitions that allows visitors to revisit and share their favorite moments.

Perform integrated testing. Add integration testing code in the same program:

Integration testing involves testing the interactions between different components or modules of a system to ensure that they work together as intended. Here's an example of how integration testing code could be added to the existing program:

import unittest

from flask import Flask, render\_template, request, redirect, url\_for

import sqlite3

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

# Connect to the database

conn = sqlite3.connect('cultural\_destinations.db')

c = conn.cursor()

# Create a table to store information about the talents

c.execute('''CREATE TABLE IF NOT EXISTS talents

(id INTEGER PRIMARY KEY,

name TEXT NOT NULL,

category TEXT NOT NULL,

description TEXT NOT NULL,

image TEXT NOT NULL,

video TEXT NOT NULL,

email TEXT NOT NULL)''')

# Define routes for the app

@app.route('/')

def index():

# Display a list of talents

talents = c.execute('SELECT \* FROM talents').fetchall()

return render\_template('index.html', talents=talents)

@app.route('/talent/new', methods=['GET', 'POST'])

def new\_talent():

if request.method == 'POST':

# Add a new talent to the database

name = request.form['name']

category = request.form['category']

description = request.form['description']

image = request.form['image']

video = request.form['video']

email = request.form['email']

c.execute('INSERT INTO talents (name, category, description, image, video, email) VALUES (?, ?, ?, ?, ?, ?)',

(name, category, description, image, video, email))

conn.commit()

return redirect(url\_for('index'))

else:

# Display a form for adding a new talent

return render\_template('new\_talent.html')

# Integration testing code

class TestCulturalDestinations(unittest.TestCase):

def setUp(self):

app.testing = True

self.app = app.test\_client()

def test\_index(self):

# Test the index page

response = self.app.get('/')

self.assertEqual(response.status\_code, 200)

def test\_new\_talent(self):

# Test adding a new talent

data = {'name': 'John Doe',

'category': 'Musician',

'description': 'Lorem ipsum dolor sit amet.',

'image': 'https://example.com/image.jpg',

'video': 'https://example.com/video.mp4',

'email': 'johndoe@example.com'}

response = self.app.post('/talent/new', data=data)

self.assertEqual(response.status\_code, 302)

talents = c.execute('SELECT \* FROM talents').fetchall()

self.assertEqual(len(talents), 1)

talent = talents[0]

self.assertEqual(talent[1], 'John Doe')

self.assertEqual(talent[2], 'Musician')

self.assertEqual(talent[3], 'Lorem ipsum dolor sit amet.')

self.assertEqual(talent[4], 'https://example.com/image.jpg')

self.assertEqual(talent[5], 'https://example.com/video.mp4')

self.assertEqual(talent[6], 'johndoe@example.com')

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

unittest.main

In this example, we've added an integration testing class TestCulturalDestinations that inherits from unittest.TestCase. The setUp() method sets the testing attribute of the Flask app to True and creates a test client for making requests to the app.

The test\_index() method tests the index page by making a GET request and checking that the response status code is 200.

The test\_new\_talent() method tests adding a new()

Modify the same program. Write APIs to access the data from the public domain and test the program for regression testing the same program:

import requests

import json

import unittest

from flask import Flask, render\_template, request, redirect, url\_for, jsonify

import sqlite3

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

# Connect to the database

conn = sqlite3.connect('cultural\_destinations.db')

c = conn.cursor()

# Create a table to store information about the talents

c.execute('''CREATE TABLE IF NOT EXISTS talents

(id INTEGER PRIMARY KEY,

name TEXT NOT NULL,

category TEXT NOT NULL,

description TEXT NOT NULL,

image TEXT NOT NULL,

video TEXT NOT NULL,

email TEXT NOT NULL)''')

# Define routes for the app

@app.route('/')

def index():

# Display a list of talents

talents = c.execute('SELECT \* FROM talents').fetchall()

return render\_template('index.html', talents=talents)

@app.route('/talent/new', methods=['GET', 'POST'])

def new\_talent():

if request.method == 'POST':

# Add a new talent to the database

name = request.form['name']

category = request.form['category']

description = request.form['description']

image = request.form['image']

video = request.form['video']

email = request.form['email']

c.execute('INSERT INTO talents (name, category, description, image, video, email) VALUES (?, ?, ?, ?, ?, ?)',

(name, category, description, image, video, email))

conn.commit()

return redirect(url\_for('index'))

else:

# Display a form for adding a new talent

return render\_template('new\_talent.html')

# Define new routes for accessing data via APIs

@app.route('/api/talents')

def get\_talents():

# Return a JSON response containing information about all talents

talents = c.execute('SELECT \* FROM talents').fetchall()

talent\_list = []

for talent in talents:

talent\_dict = {

'id': talent[0],

'name': talent[1],

'category': talent[2],

'description': talent[3],

'image': talent[4],

'video': talent[5],

'email': talent[6]

}

talent\_list.append(talent\_dict)

return jsonify(talent\_list)

@app.route('/api/talents/<int:talent\_id>')

def get\_talent(talent\_id):

# Return a JSON response containing information about a specific talent

talent = c.execute('SELECT \* FROM talents WHERE id = ?', (talent\_id,)).fetchone()

if talent is not None:

talent\_dict = {

'id': talent[0],

'name': talent[1],

'category': talent[2],

'description': talent[3],

'image': talent[4],

'video': talent[5],

'email': talent[6]

}

return jsonify(talent\_dict)

else:

return jsonify({'error': 'Talent not found'}), 404

# Regression testing code

class TestCulturalDestinations(unittest.TestCase):

def setUp(self):

app.testing = True

self.app = app.test\_client()

def test\_index(self):

# Test the index page

response = self.app.get('/')

self.assertEqual(response.status\_code, 200)

def test\_new\_talent(self):

# Test adding a new talent

data = {'name': 'John Doe',

'category': 'Musician',