Assignment

Venkatesh Jaiswal

21bcs131

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
a.Generate code and run the program in Goggle Colab/Jupiter Notebook/Visual Code/PyCharm
b.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.

Requirement 1: Registration and Profile Management

Users should be able to create accounts, update their profile information, and view their past activity.

Code:

```
class User:

   def __init__(self, username, password, email):

       self.username = username

       self.password = password

       self.email = email

       self.activity_history = []



   def update_profile(self, new_username, new_email):

       self.username = new_username

       self.email = new_email



   def add_activity(self, activity):

       self.activity_history.append(activity)



class AccountManager:

   def __init__(self):
       self.users = []



   def register(self, username, password, email):

       new_user = User(username, password, email)

       self.users.append(new_user)



   def login(self, username, password):

       for user in self.users:

           if user.username == username and user.password ==
password:

               return user

       return None

```

Requirement 2: Artwork Management

Artists should be able to submit their artwork and manage their submissions. Curators should be able to review artwork submissions and select pieces for display.

Code:

```
class Artwork:

    def __init__(self, title, artist, medium, year, image_url):

        self.title = title
```

```python
        self.artist = artist

        self.medium = medium

        self.year = year

        self.image_url = image_url

        self.submission_date = None

        self.status = "Pending"



    def submit(self):

        self.submission_date = datetime.now()

        self.status = "Submitted"



    def approve(self):

        self.status = "Approved"



    def reject(self):

        self.status = "Rejected"



class ArtworkManager:

    def __init__(self):
        self.artworks = []



    def submit_artwork(self, title, artist, medium, year,
image_url):

        new_artwork = Artwork(title, artist, medium, year,
image_url)

        new_artwork.submit()

        self.artworks.append(new_artwork)



    def approve_artwork(self, artwork):

        artwork.approve()



    def reject_artwork(self, artwork):

        artwork.reject()



    def get_pending_artwork(self):

        pending_artworks = []

        for artwork in self.artworks:

            if artwork.status == "Pending":

                pending_artworks.append(artwork)
```

```
       return pending_artworks
```

Requirement 3: Event Management

Users should be able to view upcoming events, purchase tickets, and view their past event attendance.

Code:

```
class Event:
   def __init__(self, name, date, location, description,
image_url, ticket_price):
       self.name = name
       self.date = date
       self.location = location
       self.description = description
       self.image_url = image_url
       self.ticket_price = ticket_price
       self.attendees = []

   def purchase_ticket(self, user):
       user.add_activity(self.name)
       self.attendees.append(user)

class EventManager:
   def __init__(self):
       self.events = []

   def create_event(self, name, date, location, description,
image_url, ticket_price):
       new_event = Event(name, date, location, description,
image_url, ticket_price)
```

```
        self.events.append(new_event)

    def get_upcoming_events(self):
        upcoming_events = []
        for event in self.events:
            if event.date > datetime.now():
                upcoming_events.append(event)
        return upcoming_events

    def get_past_events(self):
        past_events = []
        for event in self.events:
            if event.date < datetime.now():
                past_events.append(event)
        return past_events
```

For integration testing, we will create a script that simulates user actions and verifies the expected output.

Code:

```
# Create sample users and events
user1 = User("john_doe", "password", "johndoe@example.com")
user2 = User("jane_doe", "password", "janedoe@example.com")
account_manager = AccountManager()
account_manager.register(user1.username, user1.password,
user1.email)
account_manager.register(user2.username, user2.password,
user2.email)

event1 = Event("Concert", datetime(2023, 5, 1, 19, 0), "Mumbai",
"A concert featuring top Indian musicians", "concert.jpg", 1000)
event2 = Event("Play", datetime(2023, 4, 10, 14, 0), "Delhi", "A
play featuring regional theatre actors", "play.jpg", 500)
event_manager = EventManager()
```

```
event_manager.create_event(event1.name, event1.date,
event1.location, event1.description, event1.image_url,
event1.ticket_price)
event_manager.create_event(event2.name, event2.date,
event2.location, event2.description, event2.image_url,
event2.ticket_price)

# Simulate user actions
user1 = account_manager.login(user1.username, user1.password)
user2 = account_manager.login(user2.username, user2.password)
event_manager.get_upcoming_events() # Should return [event2,
event1]
event_manager.get_past_events() # Should return []
event2.purchase_ticket(user1)
user1.activity_history # Should return ["Play"]
event2.attendees # Should return [user1]

# Verify outputs
assert event_manager.get_upcoming_events() == [event2, event1]
assert event_manager.get_past_events() == []
assert user1.activity_history == ["Play"]
assert event2.attendees == [user1]
```

For API testing, we will write a script that requests data from the API and verifies the expected response.

Code:

```
import requests

# Test artwork submission API
url = "https://myartgallery.com/api/artwork/submit"
data = {"title": "Sunflowers", "artist": "Vincent Van Gogh",
"medium": "Oil on canvas", "year": 1888, "image_url":
"sunflowers.jpg"}
```

```
response = requests.post(url, data=data)
assert response.status_code == 200
assert response.json()["status"] == "Submitted"

# Test event retrieval API
url = "https://myartgallery.com/api/events/upcoming"
response = requests.get(url)
assert response.status_code == 200
assert len(response.json()) == 2
assert response.json()[0]["name"] == "Play"

# Test user profile update API
url = "https://myartgallery.com/api/user/profile/update"
data = {"username": "john_doe2", "email": "johndoe2@example.com"}
headers = {"Authorization": "Bearer abc123"}
response = requests.put(url, data=data, headers=headers)
assert response.status_code == 200
assert response.json()["username"] == "john_doe2"
assert response.json()["email"] == "johndoe2@example.com"
```