NAME:R.Arun

ROLL NO:21bcs093

various requirements for the program initiative and suggest some possible digital solutions for each requirement.

1. Requirements for the program initiative:

a. Create a platform for artists to showcase their work

b. Provide a way for the audience to discover and engage with the art

c. Enable artists to collaborate with each other

d. Facilitate ticket sales for events and performances

e. Collect feedback from the audience and artists to improve the platform f. Provide a way for artists to monetize their work

2. Possible digital solutions: a. Create a website or mobile app where artists can create profiles and upload their work. Users can browse through the art and engage with the artists through comments or direct messaging. b. Implement a recommendation system that suggests art to users based on their preferences and previous interactions with the platform. c. Develop a collaboration feature where artists can invite other artists to work on a project together. d. Implement an online ticketing system for events and performances, allowing users to purchase tickets and receive them digitally. e. Collect feedback through surveys, ratings, and reviews. Use this data to improve the platform's user experience and make it more attractive to artists and audiences. f. Enable artists to monetize their work through a variety of methods, such as selling prints, offering commissions, or accepting donations from users who enjoy their work.

3. For integrated testing and API testing, you can use testing frameworks like PyTest or Unittest to create test cases for each requirement and ensure the program works as expected. You can also use tools like Postman to test the APIs and ensure they return the correct data.

import mysql.connector

from datetime import datetime

# Define the database connection details

config = {

    'user': 'root',

    'password': 'samsam',

    'host': 'localhost',

    'database': 'booking'

}

# Create a new database connection

def get\_db():

    db = mysql.connector.connect(\*\*config)

    cursor = db.cursor()

    cursor.execute('''

CREATE TABLE IF NOT EXISTS bookings (

    id INTEGER PRIMARY KEY AUTO\_INCREMENT,

    name TEXT,

    email TEXT(255) ,

    event TEXT,

    date DATE,

    quantity INTEGER

);

    ''')

    return db

# Create a new booking

def create\_booking(name, email, event, date, quantity):

    db = get\_db()

    cursor = db.cursor()

    cursor.execute('INSERT INTO bookings (name, email, event, date, quantity) VALUES (%s, %s, %s, %s, %s)',

                   (name, email, event, date, quantity))

    db.commit()

    db.close()

# Get all bookings for a particular event

def get\_bookings(event):

    db = get\_db()

    cursor = db.cursor()

    cursor.execute('SELECT \* FROM bookings WHERE event = %s', (event,))

    bookings = cursor.fetchall()

    db.close()

    return bookings

# Check if a booking already exists for the given email and event

def booking\_exists(email, event, date):

    db = get\_db()

    cursor = db.cursor()

    cursor.execute('SELECT COUNT(\*) FROM bookings WHERE email = %s AND event = %s AND date = %s',(email, event, date))

    result = cursor.fetchone()

    db.close()

    return result[0] > 0

# Validate the booking details

def validate\_booking(name, email, event, date, quantity):

    errors = []

    # Check if name is not empty

    if not name:

        errors.append('Name cannot be empty')

    # Check if email is valid

    if not email:

        errors.append('Email cannot be empty')

    elif not '@' in email or not '.' in email:

        errors.append('Invalid email')

    # Check if event is not empty

    if not event:

        errors.append('Event cannot be empty')

    # Check if date is not empty and is in the correct format

    if not date:

        errors.append('Date cannot be empty')

    else:

        try:

            datetime.strptime(date, '%Y-%m-%d')

        except ValueError:

            errors.append('Invalid date format. Must be YYYY-MM-DD')

    # Check if quantity is not empty and is greater than 0

    if not quantity or int(quantity) < 1:

        errors.append('Quantity must be greater than 0')

    return errors

# Main function to create a new booking

def main():

    print('Welcome to the online booking system.')

    print('Please enter your details below.')

    # Get the user's details

    name = input('Name: ')

    email = input('Email: ')

    event = input('Event: ')

    date = input('Date (YYYY-MM-DD): ')

    quantity = input('Quantity: ')

    # Validate the details

    errors = validate\_booking(name, email, event, date, quantity)

    if errors:

        print('The following errors were found:')

        for error in errors:

            print('- ' + error)

        return

    # Check if a booking already exists

    if booking\_exists(email, event, date):

        print('A booking already exists for this email address and event on this date.')

        return

    # Create the booking

    create\_booking(name, email, event, date, quantity)

    print('Booking created successfully.')

    bookings = get\_bookings(event)

    print('Bookings for ' + event + ':')

    for booking in bookings:

        print(booking)

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

    main()

for unittest

import unittest

from datetime import datetime

from booking import validate\_booking

class TestBookingSystem(unittest.TestCase):

    def test\_validate\_booking\_with\_valid\_details(self):

        name = 'John'

        email = 'john@example.com'

        event = 'Concert'

        date = '2022-05-15'

        quantity = '2'

        errors = validate\_booking(name, email, event, date, quantity)

        if len(errors) > 0:

            print(f"Error in test\_validate\_booking\_with\_valid\_details: {errors}")

        self.assertEqual(len(errors), 0)

    def test\_validate\_booking\_with\_empty\_name(self):

        name = ''

        email = 'john@example.com'

        event = 'Concert'

        date = '2022-05-15'

        quantity = '2'

        errors = validate\_booking(name, email, event, date, quantity)

        if len(errors) > 0:

            print(f"Error in test\_validate\_booking\_with\_empty\_name: {errors}")

        self.assertEqual(len(errors), 1)

        self.assertEqual(errors[0], 'Name cannot be empty')

    def test\_validate\_booking\_with\_invalid\_email(self):

        name = 'John'

        email = 'johnexample.com'

        event = 'Concert'

        date = '2022-05-15'

        quantity = '2'

        errors = validate\_booking(name, email, event, date, quantity)

        if len(errors) > 0:

            print(f"Error in test\_validate\_booking\_with\_invalid\_email: {errors}")

        self.assertEqual(len(errors), 1)

        self.assertEqual(errors[0], 'Invalid email')

    def test\_validate\_booking\_with\_empty\_event(self):

        name = 'John'

        email = 'john@example.com'

        event = ''

        date = '2022-05-15'

        quantity = '2'

        errors = validate\_booking(name, email, event, date, quantity)

        if len(errors) > 0:

            print(f"Error in test\_validate\_booking\_with\_empty\_event: {errors}")

        self.assertEqual(len(errors), 1)

        self.assertEqual(errors[0], 'Event cannot be empty')

    def test\_validate\_booking\_with\_empty\_date(self):

        name = 'John'

        email = 'john@example.com'

        event = 'Concert'

        date = ''

        quantity = '2'

        errors = validate\_booking(name, email, event, date, quantity)

        if len(errors) > 0:

            print(f"Error in test\_validate\_booking\_with\_empty\_date: {errors}")

        self.assertEqual(len(errors), 1)

        self.assertEqual(errors[0], 'Date cannot be empty')

    def test\_validate\_booking\_with\_invalid\_date\_format(self):

        name = 'John'

        email = 'john@example.com'

        event = 'Concert'

        date = '2022/05/15'

        quantity = '2'

        errors = validate\_booking(name, email, event, date, quantity)

        if len(errors) > 0:

            print(f"Error in test\_validate\_booking\_with\_invalid\_date\_format: {errors}")

        self.assertEqual(len(errors), 1)

        self.assertEqual(errors[0], 'Invalid date format. Must be YYYY-MM-DD')

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

    unittest.main()

Report

The code provided is a Python script that implements an online booking system. The system is designed to create new bookings and retrieve bookings for a particular event.

The code uses the mysql-connector-python library to interact with a MySQL database. The connection details for the database are provided in a dictionary named "config", which contains the user name, password, host, and database name.

The script defines four functions:

1. get\_db() - This function creates a new database connection and returns the connection object. It also creates a new table named "bookings" if it doesn't already exist.

2. create\_booking(name, email, event, date, quantity) - This function creates a new booking by inserting the provided details into the "bookings" table.

3. get\_bookings(event) - This function retrieves all bookings for a particular event by executing a SELECT query on the "bookings" table with the provided event name as the filter.

4. booking\_exists(email, event, date) - This function checks if a booking already exists for the provided email, event, and date by executing a SELECT query on the "bookings" table with the provided details as the filters.

The script also defines a function named "validate\_booking" that takes in the name, email, event, date, and quantity of a booking as inputs and returns a list of errors if any of the inputs are invalid. The validation checks include ensuring that the name, email, and event fields are not empty, the email is valid, the date is in the correct format, and the quantity is greater than zero.

Finally, the script defines a "main" function that interacts with the user to create a new booking. The user is prompted to enter their name, email, event name, date, and quantity. The function then validates the inputs using the "validate\_booking" function and checks if a booking already exists for the provided email, event, and date using the "booking\_exists" function. If the inputs are valid and no booking exists for the provided details, the "create\_booking" function is called to create a new booking. The function then retrieves all bookings for the provided event using the "get\_bookings" function and displays them to the user.

In summary, the provided code implements an online booking system that allows users to create new bookings and retrieve existing bookings for a particular event. It validates user inputs and interacts with a MySQL database to store and retrieve booking details.