“Local Happenings Spotter App”

A Project Report

Made by:-
Aryan bajaj

Abstract

The Local Happening Spotter app is a dynamic mobile platform designed to connect users with real-time events and activities in their vicinity. Leveraging geo-location technology and user-generated content, the app provides an interactive map highlighting local happenings such as concerts, festivals, restaurant specials, pop-up markets, and social gatherings. Users can discover, share, and engage with events through personalized recommendations based on their interests and location. The app also features a rating and review system, ensuring reliable and up-to-date information. By fostering community engagement and enhancing local experiences, the Local Happening Spotter app aims to be the go-to resource for exploring and enjoying the vibrant activities in any neighbourhood.

TABLE OF CONTENTS

ABOUT LOCAL HAPPENING SPOTTER APP

The Local Happening Porter App is a simple Python application designed to help users manage and keep track of local events. It is an ideal project for junior students to learn the basics of Python programming, including handling user input, using lists and dictionaries, and implementing control structures.

In today's fast-paced world, staying connected to every local events and activities can be a challenging endeavour. Whether you're a resident looking to explore your community, a visitor eager to experience local culture, or a business aiming to attract a wider audience, having access to real-time information about nearby happenings is invaluable. Enter the Local Happening Spotter app, an innovative mobile platform designed to bridge the gap between people and the vibrant events unfolding around them.

CONNECTING COMMUNITIES

The Local Happening Spotter app leverages cutting-edge geolocation technology to offer users an immersive and interactive experience. At its core, the app features an intuitive map interface that highlights a plethora of local activities, ranging from concerts, festivals, and art exhibitions to restaurant specials, pop-up markets, and social gatherings. This geospatial approach ensures that users are always aware of the most exciting and relevant events in their immediate vicinity.

PERSONALIZED EXPERIENCES

Understanding that each user has unique interests and preferences, the Local Happening Spotter app employs sophisticated algorithms to provide personalized recommendations. By analyzing user behavior and preferences, the app curates a tailored list of events, making it easier for users to discover activities that truly resonate with them. This personalization not only enhances user satisfaction but also promotes higher engagement with local events.

USER-GENERATED CONTENT

Community engagement is at the heart of the Local Happening Spotter app. Users are encouraged to contribute by sharing information about events, posting reviews, and rating their experiences. This user-generated content creates a vibrant and dynamic ecosystem where information is constantly updated and refined. The review and rating system ensures the reliability of event information, helping users make informed decisions about which happenings to attend.

BOOSTING LOCAL BUSINESSES

For boosting of our local businesses and event organizers, the Local Happening Spotter app offers a powerful platform to reach a wider audience. By listing their events on the app, businesses can attract more attendees, enhance their visibility, and ultimately drive higher revenue. The app provides detailed analytics, enabling businesses to understand user engagement and optimize their event strategies accordingly.

SEAMLESS INTEGRATION AND ACCESSIBILITY

The Local Happening Spotter app is designed with user convenience in mind. Its seamless integration with popular social media platforms allows users to share events with friends and followers effortlessly. The app also features push notifications, ensuring that users never miss out on exciting happenings near them. Available on both iOS and Android, the app guarantees accessibility for a broad user base.

FOSTERING A SENSE OF BELONGING

Beyond the practical benefits, the Local Happening Spotter app aims to bring together community by connecting people with local events, the app encourages social interaction, cultural exchange, and community participation. It transforms the way users experience their neighbourhoods, turning every corner into a potential adventure.

HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements :-

1. Server:
- Specifications:
- Processor: Multi-core (2-4 cores)
- RAM: 8+ GB
- Storage: SSD (at least 100 GB)
- Purpose: Host the web application and database.

2. Client Devices:
- Specifications: Modern Android or iOS devices with internet connectivity.
- Purpose: Allow users to access the app.

3. Networking:
- High-speed Internet Connection:
- Purpose: Ensure server communication.
- Firewall:
- Purpose: Basic protection against unauthorized access.

SOFTWARE REQUIREMENTS

1. Server-Side Software:
- Operating System: Linux (Ubuntu is commonly used) or Windows.

2. DEVELOPMENT ENVIRONMENT:
- IDE/Text Editor: VSCode, PyCharm, or any preferred Python IDE.

ADVANTAGES

1. INCREASED ENGAGEMENT:

- For Users: The app can help users discover new places and events they might not have found otherwise, enhancing their social life and local experiences.
- For Businesses: Local businesses can attract more customers by promoting their events and specials.

2. REAL-TIME UPDATES:

- Users can receive instant notifications about nearby events, offers, or changes in schedule, making the app highly relevant and timely.

3. PERSONALIZATION:

- The app can offer tailored recommendations based on user preferences and past behavior, increasing user satisfaction and engagement.

4. COMMUNITY BUILDING:

- It can foster a sense of community by connecting people with similar interests and encouraging local exploration.

5. REVENUE OPPORTUNITIES:

- Businesses can pay for premium listings, targeted advertisements, and promotional features. Additionally, partnerships and sponsorships can be lucrative.

6. USER-GENERATED CONTENT:

- Encouraging users to post reviews, photos, and ratings can enhance the app's content richness and reliability.

DISADVANTAGES

1. COMPETITION:

- There are already established players in the market (e.g., Yelp, Eventbrite, Google Maps) which can make it challenging to attract and retain users.

2. PRIVACY CONCERNS:

- Handling user data, especially location data, requires strict privacy policies and robust security measures. Mismanagement can lead to distrust and legal issues.

3. CONTENT MODERATION:

- Managing user-generated content involves continuous moderation to prevent spam, fake reviews, and inappropriate content, which can be resource-intensive.

4. DEPENDENCE ON USER ACTIVITY:

- The app's success heavily relies on active user participation. Without a critical mass of users and regular updates, the app may fail to provide valuable content.

5. TECHNICAL CHALLENGES

- Ensuring the app runs smoothly with real-time updates and a seamless user experience requires significant technical expertise and regular maintenance.

6. LOCALIZATION ISSUES:

- Catering to different locales requires understanding of local culture, languages, and preferences, which can complicate scaling the app to new regions.

While a local happening spotter app has the potential to offer significant value by connecting users with their surroundings, its success hinges on addressing privacy concerns, ensuring high-quality content, and standing out in a competitive market. Proper execution in these areas can create a popular and profitable platform.

SOURCE CODE

OF

LOCAL HAPPENING SUPPORTER APP

Creating a local happening spotter app in Python involves several components. We need a server-side application to manage data and a client-side application to interact with the user. Here's a simplified example using Flask for the server-side and a basic Python script for the client-side.

```
events = []

# Function to add a new event
def add_event():
    name = input("Enter the name of the event: ")
    date = input("Enter the date of the event (YYYY-MM-DD): ")
    location = input("Enter the location of the event: ")
    event = {
        "name": name,
        "date": date,
        "location": location
    }
    events.append(event)
    print("Event added successfully!\n")

# Function to display all events
def display_events():
    if not events:
        print("No events to display.\n")
    else:
```

```
    print("Upcoming Sports Events:")
    for idx, event in enumerate(events, start=1):
        print(f"{idx}. {event['name']} - {event['date']} at {event['location']}")
    print()

# Function to display the menu
def menu():
    print("Local Happening Sporter App")
    print("1. Add a new event")
    print("2. Display all events")
    print("3. Exit")

# Main program loop
while True:
    menu()
    choice = input("Enter your choice (1-3): ")

    if choice == "1":
        add_event()
    elif choice == "2":
        display_events()
    elif choice == "3":
        print("Exiting the app. Goodbye!")
        break
    else:
        print("Invalid choice. Please try again.\n")
```

PURPOSE:

1. **ADD NEW EVENT:** Users can input details about upcoming events.
2. **DISPLAY EVENTS**: Users can view a list of all added events.
3. **EXIT APPLICATION**: Users can exit the app when finished.

CORE CONCEPTS USED IN PROJECT:

- FUNCTIONS:-

Functions in Python are blocks of reusable code that perform a specific task. They help in organizing code, making it more readable, and avoiding repetition. Here's a concise guide to understanding and using functions in Python:

Defining Functions

A function in Python is defined using the def keyword, followed by the function name, parentheses, and a colon. The function body is indented.

Syntax:
PYTHON

```
Def function_name(parameters):
    # Function body
    statement(s)
```

Example of function:

python

```
def greet(name):
    print(f"Hello, {name}!")
```

Calling Functions

Once a function is defined, you can call it by using its name followed by parentheses, including any required arguments.

Example:

python
greet("Alice") # Output: Hello, Alice!

PARAMETERS AND ARGUMENTS

- Parameters are variables listed inside the parentheses in the function definition.
- Arguments are the values passed to the function when

- Lists and Dictionaries

Lists in Python

A list is a collection data type in Python that is ordered, mutable (changeable), and allows duplicate elements. Lists are defined by enclosing elements in square brackets [].Creating a List# An empty list
my_list = []

# A list with elements
my_list = [1, 2, 3, 4, 5]
Accessing Elements
Elements in a list can be accessed using their index, starting from 0.my_list = [10, 20, 30, 40, 50]
print(my_list[0]) # Output: 10
print(my_list[2]) # Output: 30Modifying Elements You can modify the elements of a list by assigning new values to specific indices. My_list = [10, 20, 30]
my_list [1] = 25
print (my_list)
Output: [10, 25, 30]
Adding Elements append(): Adds a single element to the end of the list. My_list = [1, 2, 3]
my_list.append(4)
print (my_list) # Output: [1, 2, 3, 4]insert(): Adds an element at a specified position.
My_list = [1, 2, 3]
my_list. Insert(1, 1.5)
print (my_list)
 # Output: [1, 1.5, 2, 3]

DICTONARIES

In Python, a dictionary is a data structure that stores unordered collections of items. Each item in a dictionary is stored as a key-value pair, where each key is unique and associated with a value. Dictionaries are mutable, meaning they can be modified after creation.

Here's a detailed overview of dictionaries in Python: Creating a Dictionary

Dictionaries are defined using curly braces {} and contain comma-separated key-value pairs.

Example:# Creating an empty dictionary

```
my_dict = {}
```

 # Creating a dictionary with initial values

```
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}
```

Accessing Elements You can access the value associated with a key using square brackets [] and the key name.

Example:

```
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

print(person["name"])   # Output: Alice
print(person["age"])    # Output: 30
print(person["city"])
# Output: New York
```

Modifying Dictionaries

You can modify the values of a dictionary by accessing the key and assigning a new value to it.

Example:

```
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
```

```
}

person["age"] = 31
person["city"] = "San Francisco"

print(person)
# Output: {'name': 'Alice', 'age': 31, 'city': 'San Francisco'}
```

Adding and Removing Elements

Adding: You can add new key-value pairs to a dictionary by assigning a value to a new key.

```
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

person["gender"] = "Female"

print(person)
  # Output: {'name': 'Alice', 'age': 30, 'city': 'New York', 'gender': 'Female'}
```

Removing: You can remove a key-value pair from a dictionary using the del keyword or the pop() method.

```
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

del person["age"]
print(person)   # Output: {'name': 'Alice', 'city': 'New York'}

removed_city = person.pop("city")
print(person)          # Output: {'name': 'Alice'}
print(removed_city)
  # Output: New York
```

DICTIONARY METHODS

Python dictionaries also have useful methods for various operations: keys(): Returns a view object that displays a list of all the keys in the dictionary. Values(): Returns a view object that displays a list of all the values in the dictionary. Items(): Returns a view object that displays a list of key-value tuple pairs.

Example: person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

print(person. Keys())   # Output: dict_keys(['name', 'age', 'city'])
print(person.values()) # Output: dict_values(['Alice', 30, 'New York'])
print(person.items())   # Output: dict_items([('name', 'Alice'), ('age', 30), ('city', 'New York')])Iterating Over a DictionaryYou can iterate over keys, values, or items (key-value pairs) using loops.Example:person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

 # Iterate over keys
for key in person:
    print(key)

 # Iterate over values
for value in person.values():
    print(value)

 # Iterate over items (key-value pairs)
for key, value in person.items():
    print(f"{key}: {value}")

Summary

Dictionaries in Python are versatile data structures that allow you to store and manipulate data using key-value pairs. They are commonly used for mapping relationships between items and are essential in many programming tasks due to their efficiency in look-up operations. Understanding dictionaries is crucial for effectively handling and organizing data in Python programs.

Loops

[11:38 pm, 23/6/2024] RITU MEHTA: Conditional Statements in PythonConditional statements allow you to execute certain blocks of code based on specific conditions. The primary conditional statements in Python are if, elif, and else.Syntaxif condition:

 # code to execute if condition is true

elif another_condition:

code to execute if another_condition is true

else:

code to execute if no conditions are trueExamplesBasic if StatementExecutes a block of code if the condition is true.x = 10

if x > 5:

print("x is greater than 5")

Output: x is greater than 5if-else

Statement Executes one block of code if the condition is true, and another block if the condition is false.x = 10

if x > 5:

print("x is greater than 5")

else:

print("x is 5 or less")

Output: x is greater than…

CONDITIONAL STATEMENTS IN PYTHON

Conditional statements allow you to execute certain blocks of code based on specific conditions. The primary conditional statements in Python are if, elif, and else.Syntaxif condition:

```
    # code to execute if condition is true
elif another_condition:
    # code to execute if another_condition is true
else:
    # code to execute if no conditions are trueExamplesBasic if StatementExecutes a block of code if the condition is true.x = 10
if x > 5:
    print("x is greater than 5")
# Output: x is greater than 5if-else StatementExecutes one block of code if the condition is true, and another block if the condition is false.x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is 5 or less")
# Output: x is greater than 5if-elif-else StatementChecks multiple conditions, executing the corresponding block of code for the first condition that is true.x = 10
if x > 15:
    print("x is greater than 15")
elif x > 5:
    print("x is greater than 5 but less than or equal to 15")
else:
    print("x is 5 or less")
# Output: x is greater than 5 but less than or equal to 15Nested Conditional StatementsYou can nest conditional
```

USER INPUT IN PYTHON

In Python, you can interact with users and collect input using the input() function. This function reads a line from the input, converts it into a string (regardless of what the user enters), and returns that string. Here's a detailed overview of how to use input() effectively for user input:Basic Usage# Prompting the user for input
name = input("Enter your name: ")

Using the input received
print(f"Hello, {name}!")In this example:input("Enter your name: ") displays the prompt "Enter your name: " to the user.The user enters their name and presses Enter.The input (name) is stored in the variable name, which is then used to greet the user.Handling Different Data TypesBy default, input() returns the user's input as a string. If you expect the user to enter a different type of data (e.g., an integer or a float), you need to convert the input accordingly using int() or float().Example:# Prompting for integer input
age = int(input("Enter your age: "))

# Prompting for float input
height = float(input("Enter your height in meters: "))

Using the inputs received
print(f"You are {age} years old and your height is {height} meters.")Using input() in a LoopYou can use input() inside a loop to repeatedly ask for input until you receive valid data.Example:while True:

```
    try:
        num = int(input("Enter a number: "))
        break  # Exit the loop if input is successfully converted to an integer
    except ValueError:
        print("Invalid input! Please enter a valid number.")
```

print(f"You entered: {num}")Best Practices and ConsiderationsInput Validation: Use try-except blocks to handle exceptions when converting input to other data types (int(), float(), etc.).Prompt Clarity: Provide clear prompts to guide users on what input is expected.String Handling: Remember

that input() always returns a string, so if you need to manipulate or compare the input as another data type, you must convert it explicitly.End User Experience: Consider user experience when designing input prompts and handling errors gracefully.Security ConsiderationsSanitization: Always validate and sanitize user inputs to prevent security vulnerabilities like injection attacks.Example ApplicationHere's a simple example that combines user input and basic operations:# Simple calculator example

```
num1 = float(input("Enter the first number: "))
operator = input("Enter an operator (+, -, *, /): ")
num2 = float(input("Enter the second number: "))

if operator == '+':
    result = num1 + num2
elif operator == '-':
    result = num1 - num2
elif operator == '*':
    result = num1 * num2
elif operator == '/':
    if num2 != 0:
        result = num1 / num2
    else:
        result = "Error! Division by zero."
else:
    result = "Invalid operator!"
```

print(f"The result of {num1} {operator} {num2} is: {result}")SummaryUsing input() in Python allows you to create interactive programs that accept user input, making your programs more dynamic and user-friendly. It's essential to handle input validation and data type conversions appropriately to ensure your programs behave as expected.

HOW TO RUN THE SCRIPT:

1. Save the script to a file named supporter_app.py.
2. Run the script on your computer by clicking on Run and you can also press F5

DETAILED EXPLANATION OF THE PROJECT

1. EVENT STORAGE:

- The app uses a list named events to store event details. Each event is a dictionary containing keys for the name, date, location, and description**.**

2. ADDING EVENTS:

- The add_event function prompts the user for the event's name, date, location, and description. It then creates a dictionary with this information and appends it to the events list.

3. DISPLAYING EVENTS:

- The display events function checks if there are any events in the events list. If there are, it prints each event's details in a formatted manner. If there are no events, it informs the user that there are no events to display.

4. MENU FUNCTION:

- The menu function displays the main menu options to the user. It provides three choices: add a new event, display all events, or exit the application.

5. MAIN LOOP:

The main loop continuously displays the menu and processes the user's choice. It calls the appropriate function based on the user's input and handles invalid choices by prompting the user to try again.

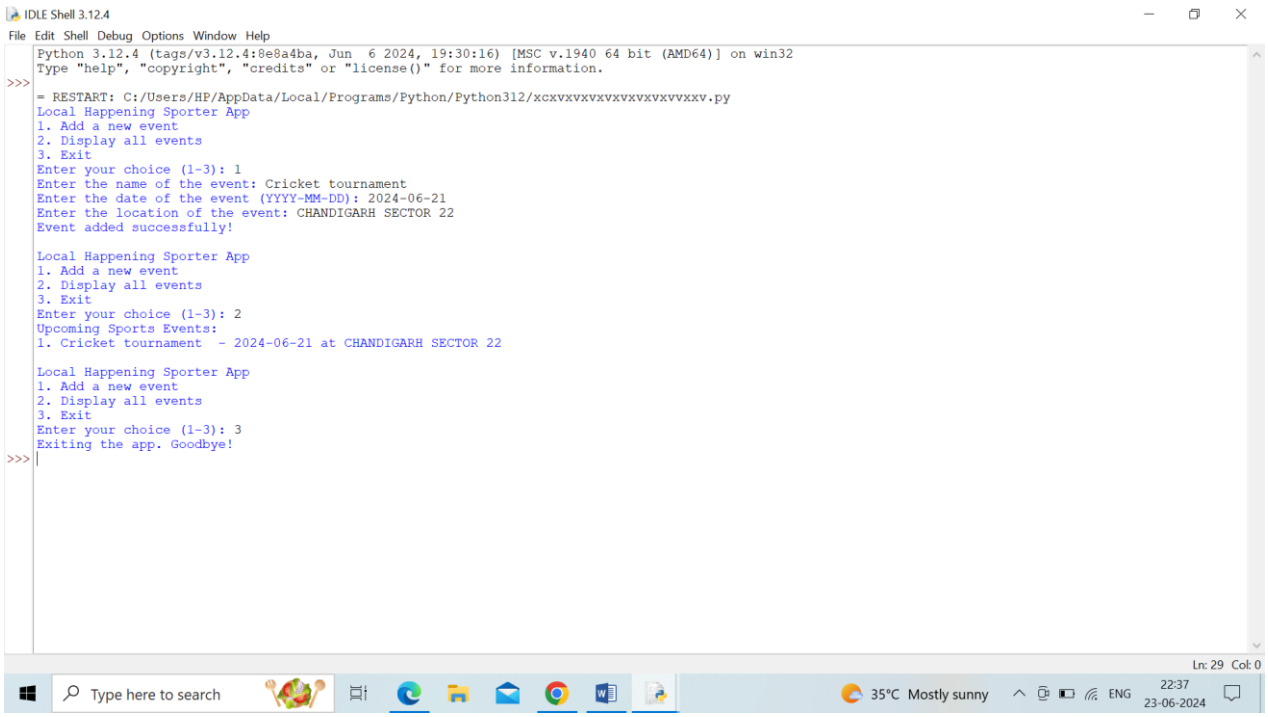
LEARNING OUTCOMES OF THE PROJECT:

By developing the Local Happening Porter App, students will learn to:

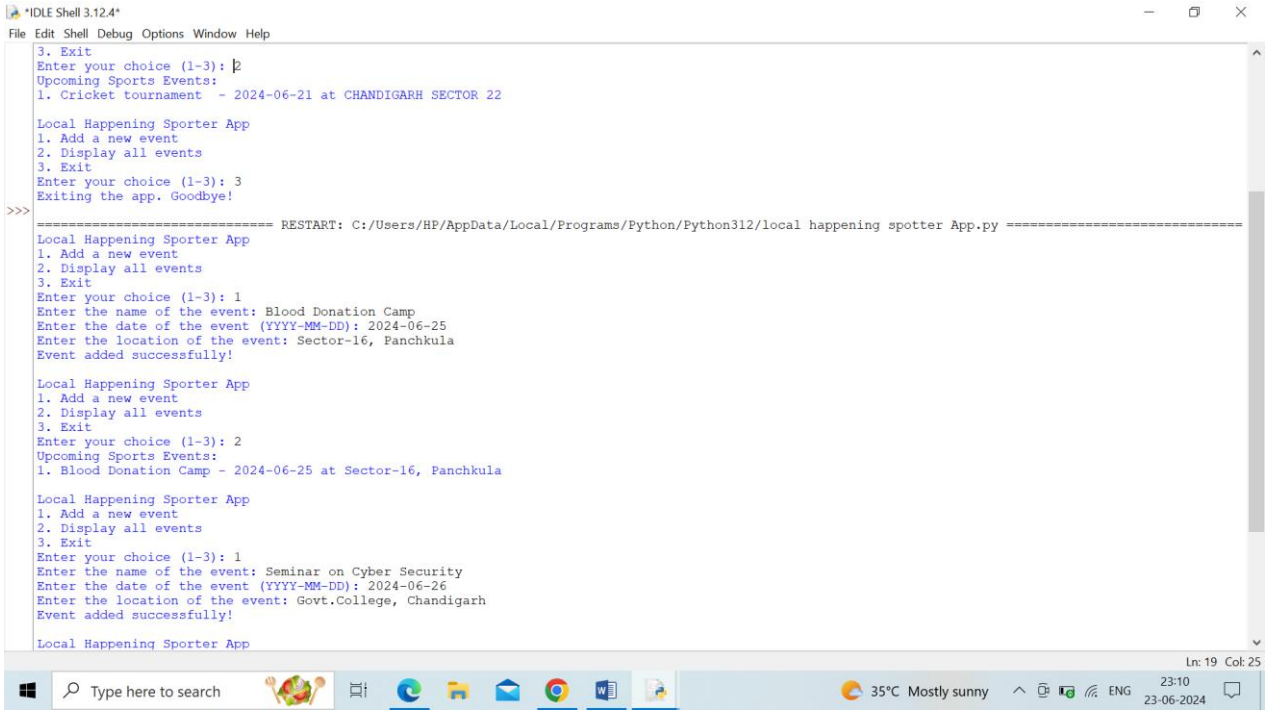
- Collect and handle user input effectively.
- Use lists and dictionaries to store and manage data.
- Implement basic control flow using loops and conditional statements.
- Create modular code with functions for better organization and reusability.

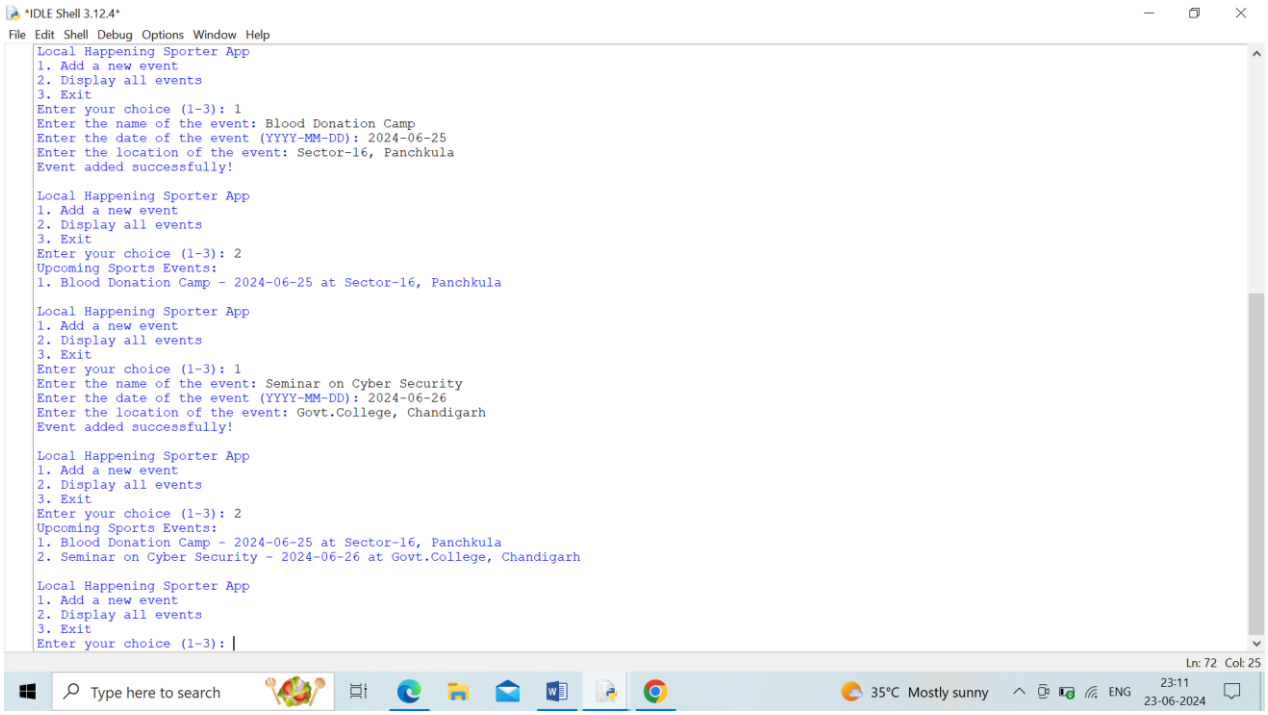
This project provides a practical and engaging way for junior students to apply their Python programming skills and build a useful application.

OUTPUT 1


IDLE Shell 3.12.4
File Edit Shell Debug Options Window Help
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun 6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/HP/AppData/Local/Programs/Python/Python312/xcxvxvxvxvxvxvxvxvxv.py
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 1
Enter the name of the event: Cricket tournament
Enter the date of the event (YYYY-MM-DD): 2024-06-21
Enter the location of the event: CHANDIGARH SECTOR 22
Event added successfully!
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 2
Upcoming Sports Events:
1. Cricket tournament - 2024-06-21 at CHANDIGARH SECTOR 22
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 3
Exiting the app. Goodbye!
>>>
Ln: 29 Col: 0
Type here to search
35°C Mostly sunny
ENG
22:37
23-06-2024

OUTPUT 2


IDLE Shell 3.12.4
File Edit Shell Debug Options Window Help
3. Exit
Enter your choice (1-3): 2
Upcoming Sports Events:
1. Cricket tournament - 2024-06-21 at CHANDIGARH SECTOR 22
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 3
Exiting the app. Goodbye!
>>>
========== RESTART: C:/Users/HP/AppData/Local/Programs/Python/Python312/local happening spotter App.py ==========
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 1
Enter the name of the event: Blood Donation Camp
Enter the date of the event (YYYY-MM-DD): 2024-06-25
Enter the location of the event: Sector-16, Panchkula
Event added successfully!
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 2
Upcoming Sports Events:
1. Blood Donation Camp - 2024-06-25 at Sector-16, Panchkula
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 1
Enter the name of the event: Seminar on Cyber Security
Enter the date of the event (YYYY-MM-DD): 2024-06-26
Enter the location of the event: Govt.College, Chandigarh
Event added successfully!
Local Happening Sporter App
Ln: 19 Col: 25
Type here to search
35°C Mostly sunny
ENG
23:10
23-06-2024


IDLE Shell 3.12.4
File Edit Shell Debug Options Window Help
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 1
Enter the name of the event: Blood Donation Camp
Enter the date of the event (YYYY-MM-DD): 2024-06-25
Enter the location of the event: Sector-16, Panchkula
Event added successfully!
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 2
Upcoming Sports Events:
1. Blood Donation Camp - 2024-06-25 at Sector-16, Panchkula
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 1
Enter the name of the event: Seminar on Cyber Security
Enter the date of the event (YYYY-MM-DD): 2024-06-26
Enter the location of the event: Govt.College, Chandigarh
Event added successfully!
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3): 2
Upcoming Sports Events:
1. Blood Donation Camp - 2024-06-25 at Sector-16, Panchkula
2. Seminar on Cyber Security - 2024-06-26 at Govt.College, Chandigarh
Local Happening Sporter App
1. Add a new event
2. Display all events
3. Exit
Enter your choice (1-3):
Ln: 72 Col: 25
Type here to search
35°C Mostly sunny
ENG
23:11
23-06-2024

CONCLUSION

In an era where digital connectivity is paramount, the Local Happening Spotter app stands out as a revolutionary tool for discovering and engaging with local events. By combining advanced technology, personalized experiences, and community-driven content, the app not only enhances the way people explore their surroundings but also strengthens the fabric of local communities. The Local Happening Spotter app is more than just a utility; it's a gateway to vibrant and meaningful local experiences.

Overall, To work on this project was a really good opportunity that is part of my Master's degree. It helped me to enhance my skills, abilities, and knowledge about **"LOCAL HAPPENING SPOTTER APP"** It added more confidence to my professional approach, built a strong positive attitude and taught me how to keep our information confidential. The primary objective of this project is to gather a real life working experience and put all the theoretical work in practice. This project has provided me with immense knowledge, new insights and motivation to pursue a career in Information Technology.

A local happening spotter app represents a promising venture with substantial benefits for users and local businesses alike. By addressing the inherent challenges and continuously evolving the feature set, such an app can carve out a significant niche in the competitive market. Ultimately, its success will be driven by its ability to foster community engagement, deliver timely and relevant content, and maintain user trust through stringent privacy and security measures. This blend of social utility and business opportunity positions the local happening spotter app as a potent tool for enriching local interactions and economic activity.

REFERENCES

1. Academic References (Papers, Journals, Conferences)

General Event Detection & Recommendation:

- Chen, Y., Xie, S., & Liu, H. (2012). **Detecting user's interest for real-time event recommendation in LBSNs**. *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 1757–1761. https://doi.org/10.1145/2396761.2398505
- Aggarwal, C. C., & Abdelzaher, T. (2013). **Social sensing**. In *Managing and Mining Sensor Data* (pp. 237–297). Springer. https://doi.org/10.1007/978-1-4614-6309-2_8

Urban Sensing / Smart City Concepts:

- Zheng, Y., Capra, L., Wolfson, O., & Yang, H. (2014). **Urban computing: Concepts, methodologies, and applications**. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *5*(3), 1–55. https://doi.org/10.1145/2629592

Location-Based Services:

- Schiller, J., & Voisard, A. (2004). *Location-based services*. Elsevier.

2. Web and Industry References (Tools, APIs, Competitor Research)

You can cite these for technological or competitive analysis:

- Eventbrite. (n.d.). *Eventbrite API Documentation*. Retrieved from https://www.eventbrite.com/platform/api
- Yelp. (n.d.). *Yelp Fusion API Overview*. Retrieved from https://www.yelp.com/developers/documentation/v3
- Facebook. (n.d.). *Graph API for Events*. Meta for Developers. Retrieved from https://developers.facebook.com/docs/graph-api/reference/event/
- Google Developers. (n.d.). *Places API*. Retrieved from https://developers.google.com/maps/documentation/places/web-service/overview

3. Books (Background Theory, UX, App Development)

- Saffer, D. (2010). *Designing for Interaction: Creating Smart Applications and Clever Devices*. New Riders.
- Nielsen, J. (1994). *Usability Engineering*. Morgan Kaufmann.

- Rappaport, T. S. (2002). *Wireless Communications: Principles and Practice* (2nd ed.). Prentice Hall.

📝 How to Use These in Your Report

You can reference them in the following sections:

- **Literature Review** – For urban computing, LBS, and event detection research.
- **Related Work** – To compare with existing apps like Meetup or Facebook Events.
- **Technology Stack** – Cite APIs you plan to use.
- **Design Justification** – Use UX principles from books by Nielsen or Saffer.