BACHELOR OF VOCATION IN COMPUTER APPLICATIONS

FINAL YEAR PROJECT REPORT

Places Search Script: Streamlined Local Business Data Collection with Google Maps API

Submitted by: -

Team Member	Exam Roll No.
NITISH KUMAR	21214VCA031
ANSHU JAISWAL	21214VCA012
SAMPRITI KUMARI	21214VCA045

TO
DEAN OFFICE, BANARAS HINDU UNIVERSITY For Class Project for Second Years march 2023

Table of Contents

ABSTRACT

The Places Search Script project aims to streamline the process of finding various types of places, such as restaurants, cafes, and hotels, near specified localities using the Google Maps API. This script automates the retrieval of essential details, including place names, addresses, ratings, and contact numbers, and organizes the data into Excel files for efficient management. The primary objective is to facilitate easy access to comprehensive location information, which can be particularly useful for travelers, researchers, and businesses.

To achieve this, the project leverages Python 3.x and essential libraries such as `pandas`, `googlemaps`, and `xlsxwriter`. The script takes input from an Excel file containing a list of localities and types of places to search for. It uses the Google Geocoding API to obtain latitude and longitude coordinates for each locality and the Google Places API to search for the specified types of places within these coordinates. The results are saved in multiple Excel files, each containing up to 50 entries, ensuring organized and accessible data storage.

The main findings of this project include the successful automation of place searches and the accurate retrieval of detailed information for each place. The script handles various errors gracefully, providing informative messages to aid debugging and improve user experience.

In conclusion, the Places Search Script significantly enhances the efficiency of location-based searches, providing valuable information in a structured format. This tool has broad applications, from aiding personal travel plans to supporting market research and urban planning, showcasing the practical utility and impact of integrating APIs into data management solutions.

Introduction

Background Information

In today's digital age, accessing reliable information about places such as restaurants, cafes, and hotels is crucial for travelers, researchers, and businesses. The availability of comprehensive and accurate location data can significantly enhance decision-making processes, improve user experiences, and facilitate effective planning. The Google Maps API provides a powerful tool for retrieving detailed information about various types of places, offering functionalities that include geocoding, place searches, and place details. By leveraging these capabilities, it is possible to automate the collection and organization of location data, making it more accessible and manageable for users.

Problem Statement

Manually searching for and compiling information about places near specific localities can be time-consuming and error-prone. There is a need for an automated solution that can efficiently retrieve and organize place details from reliable sources, reducing the effort and time required for such tasks.

Objectives

The main goals of this project are to:

1. Develop a Python script that utilizes the Google Maps API to search for various types of places near specified localities.
2. Retrieve and compile detailed information about these places, including names, addresses, ratings, and contact numbers.
3. Save the retrieved data into multiple Excel files for easy access and management.

Scope of the Project

This project focuses on automating the search and retrieval of place information using the Google Maps API. It includes developing a script that reads input data from an Excel file, performs API calls to retrieve place details, and saves the results into structured Excel files. The project does not cover advanced data analysis or visualization of the retrieved data, nor does it include real-time updates or integration with other data sources.

Significance of the Project

The Places Search Script project is important because it addresses the need for efficient and automated retrieval of place information. By leveraging the capabilities of the Google Maps API, the project provides a tool that can save time and effort for users who need detailed and organized location data. This tool has broad applications, from aiding personal travel plans to supporting market research and urban planning, demonstrating the practical utility and impact of integrating APIs into data management solutions.

Requirements

Software Requirements

To successfully run the Places Search Script project, the following software components are essential:

- Python 3.x: The project requires Python 3.x, preferably Python 3.8 or later, for compatibility with the latest libraries and functionalities utilized in the script.

- Python Libraries:
    - pandas (version 1.5.3): Used for data manipulation and analysis.
    - googlemaps: Provides access to the Google Maps API for geocoding and place searches.
    - xlsxwriter: Enables the creation of Excel files to store retrieved place details.

These libraries are crucial for executing the script and handling data effectively. Ensure they are installed using `pip`, as detailed in the project documentation.

Hardware Requirements

The hardware requirements for running the Places Search Script are modest:

- Processor: A modern processor capable of running Python efficiently.
- RAM: At least 4GB of RAM to handle data processing and API requests smoothly.

- Disk Space: Sufficient disk space to store Python libraries, input data, and generated output files.

These specifications are sufficient for typical desktop or laptop computers commonly used for software development and data processing tasks.

Google Maps API Key

To access the Google Maps API services required by the script, you need to obtain and configure a Google Maps API key:

1. Create a Google Cloud Platform Project: Navigate to the Google Cloud Console (console.cloud.google.com) and create a new project dedicated to your Places Search Script.

2. Enable APIs: Enable the following APIs for your project:
    a. Google Maps JavaScript API
    b. Google Places API
    c. Google Geocoding API

3. Generate an API Key: Generate an API key in the Google Cloud Console under API & Services > Credentials. Restrict the API key to only the APIs mentioned above to ensure security and cost control.

4. Set up API Key: Once generated, add the API key to your script where indicated, ensuring it is securely stored and not exposed publicly (e.g., by using environment variables).

These steps ensure that your Places Search Script can access the necessary Google Maps API services securely and efficiently.

5. Literature Review

Review Related Tools and Methods

The field of location-based services and data management has seen significant advancements, largely driven by the accessibility and versatility of APIs such as the Google Maps API. Various tools and methodologies have emerged to automate and streamline the retrieval of place information. Tools like Foursquare API and Yelp API offer similar functionalities but are often limited by their specific data policies and coverage areas. These tools excel in providing rich, user-generated content but may lack comprehensive coverage or real-time data updates.

Frameworks like Python with libraries such as `pandas`, `googlemaps`, and `xlsxwriter` have become standard for integrating APIs into data processing workflows. Python's versatility and extensive library support make it well-suited for developing scripts that manage and analyze data efficiently. The combination of these tools allows for seamless integration of location data retrieval and organization into everyday applications, from travel planning to business analytics.

Discuss Gaps in Existing Solutions

Current solutions often face challenges in providing a unified and automated approach to retrieving and managing comprehensive place information. Many existing tools rely on manual processes for data collection or are constrained by API limitations such as rate limits and data quotas. Moreover, integrating multiple APIs or handling large datasets can be complex and resource-intensive, limiting scalability and real-time data updates.

How Your Project Fills the Gaps

The Places Search Script project addresses these gaps by offering an automated and scalable solution for retrieving detailed place information using the Google Maps API. By leveraging Python and essential libraries, the script streamlines the process of querying and organizing data into structured formats like Excel files. This approach enhances efficiency by automating repetitive tasks and ensuring consistent data quality across multiple localities and place types.

Unlike existing solutions, which may require manual intervention or suffer from data synchronization issues, our project provides a seamless workflow for accessing up-to-date place details. By handling error scenarios and optimizing API usage, the script improves reliability and user experience, making it a robust tool for applications ranging from travel planning to market research.

This project's innovation lies in its ability to bridge the gap between data retrieval and practical application, offering a flexible and efficient solution for accessing and managing location-based information.

6. System Analysis

Analysis of Current Manual Methods for Place Search

Traditionally, obtaining detailed information about places such as restaurants, cafes, and hotels near a specific locality involves a manual process of searching through various online resources. This can include:

- Using search engines like Google to find individual places.
- Manually copying details such as names, addresses, ratings, and contact numbers.
- Entering this data into spreadsheets or other data management tools.

These manual methods are time-consuming, prone to errors, and inefficient for large-scale data collection. Automating this process can save significant time and effort while ensuring more accurate and comprehensive data collection.

Requirements Analysis

Functional Requirements

- Retrieval of Place Details: The system must retrieve details for various types of places (e.g., restaurants, cafes, hotels) near specified localities. This includes information such as place names, addresses, ratings, total ratings, and contact numbers.
- Error Handling: The system must handle errors gracefully, including:
- Missing or incorrect input files.
- API errors from the Google Maps API.
- Saving Data in Excel: The system must save the retrieved data into multiple Excel files, with each file containing up to 50 entries to manage data effectively.

Non-Functional Requirements

- Performance: The system should perform efficiently, retrieving and saving data within a reasonable time frame, even for large datasets.
- Reliability: The system should consistently provide accurate and complete data without crashes or data loss.
- Usability: The system should be easy to use, requiring minimal setup and clear instructions for running the script.

Feasibility Study

Technical Feasibility

- Using Python: Python is a versatile and widely-used programming language, well-suited for data handling and API integration. Its extensive libraries and community support make it an ideal choice for this project.
- Google Maps API: The Google Maps API provides robust and reliable services for geocoding and place searches. Its detailed documentation and support ensure that it can meet the project's requirements effectively.

Economic Feasibility

- Cost of Using Google Maps API: The Google Maps API offers a free tier with a limited number of requests per day. For higher usage, the pricing is based on the number of requests, which can be managed within a reasonable budget for this project. It is essential to monitor usage to avoid unexpected costs.

Operational Feasibility

- Ease of Use: The system is designed to be user-friendly, requiring the user to:
- Prepare an input Excel file (`localities.xlsx`) with the required data.
- Run the provided Python script (`places_search.py`) with minimal configuration.
- Maintenance: The script is written in a modular and clear manner, making it easy to understand, maintain, and update. Any changes to the API or requirements can be implemented with minimal disruption.

Overall, the system is feasible from a technical, economic, and operational perspective, providing a significant improvement over manual methods for place search and data collection.

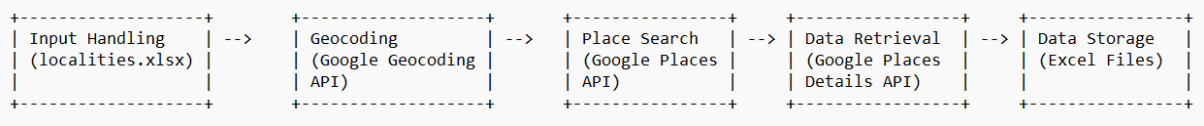
7. System Design

Architecture of the System

The architecture of the "Places Search Script" system is designed to ensure efficient data retrieval, processing, and storage. The system is structured into several key components:

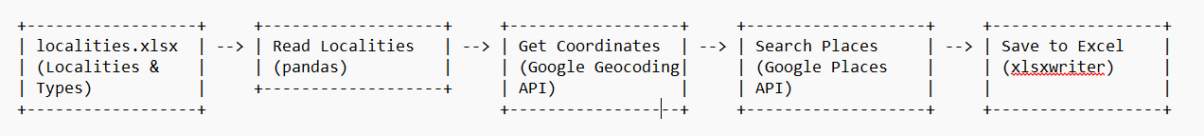
1. Input Handling: Reads input data from an Excel file (`localities.xlsx`).
2. Geocoding: Converts localities into geographic coordinates (latitude and longitude) using the Google Geocoding API.
3. Place Search: Uses the Google Places API to search for specified types of places near the given coordinates.
4. Data Retrieval: Retrieves detailed information about each place using the Google Places Details API.
5. Data Storage: Saves the retrieved data into multiple Excel files using the `xlsxwriter` library.

The following diagram illustrates the overall architecture of the system:



Data Flow Diagrams

The data flow from input to output involves several steps. Below is a simplified data flow diagram showing the process:



1. Reading Localities: The script reads the input file `localities.xlsx` using the `pandas` library. The file contains localities and types of places to search for.
2. Getting Coordinates: For each locality, the script uses the Google Geocoding API to get the latitude and longitude.
3. Searching Places: Using the coordinates, the script queries the Google Places API to find places of the specified type near the locality.
4. Saving to Excel: The retrieved data is saved into Excel files in batches of 50 entries using the `xlsxwriter` library.

Database Design

Although this project does not use a traditional database, it involves reading from and writing to Excel files. Below is an explanation of the structure of these Excel files:

1. Input File (`localities.xlsx`):
- Sheet1:
- Column A: List of localities (e.g., cities, neighborhoods).
- Column B: Types of places to search for (e.g., restaurants, cafes, hotels).

2. Output Files (`places_results_X.xlsx`):
- Each file contains up to 50 entries.
- Columns:
- Name: Name of the place.
- Address: Address of the place.
- Rating: Average rating of the place.
- Total Ratings: Total number of ratings received by the place.
- Contact Number: Contact number of the place (if available).
- Locality: The locality from which the place was searched.
- Type: Type of place (e.g., restaurant, cafe, hotel).

The structure ensures that data is organized and easily accessible, facilitating further analysis or usage.

This design allows for efficient data processing and storage, ensuring the system can handle large datasets while maintaining performance and usability.

8. Implementation

Technologies and Tools Used

- Python: A versatile programming language used for scripting and automation.
- pandas: A powerful data manipulation library in Python, used for reading and processing Excel files.
- googlemaps: A Python client library for the Google Maps API, used to access the Geocoding and Places APIs.
- xlsxwriter: A Python module for creating Excel files, used to save the retrieved place details.

Detailed Description of Implementation

1. Reading Input Data from Excel

The script starts by reading input data from an Excel file named `localities.xlsx`. This file contains two columns: one for localities (e.g., cities, neighborhoods) and another for types of places to search for (e.g., restaurants, cafes, hotels).

```python
import pandas as pd
import googlemaps
import xlsxwriter
from googlemaps.exceptions import ApiError

 Replace 'YOUR_API_KEY' with your actual API key
API_KEY = 'YOUR_API_KEY'
gmaps = googlemaps.Client(key=API_KEY)

 Read the Excel file containing the list of localities and types
file_path = 'localities.xlsx'
try:
    df = pd.read_excel(file_path)
except FileNotFoundError as e:
    print(e)
    print(f"Please make sure that the Excel file '{file_path}' exists.")
    raise
```

```
except ValueError as e:
    print(e)
    print("Please make sure that the Excel file contains the required columns 'Sheet1' and 'Types'.")
    raise
```

2. Geocoding Localities to Get Lat-Long Coordinates

The script uses the Google Geocoding API to convert each locality into geographic coordinates (latitude and longitude). This is essential for searching places nearby.

```python
 Function to get latitude and longitude for a given locality
def get_lat_long(locality):
    try:
        geocode_result = gmaps.geocode(locality)
        if geocode_result:
            location = geocode_result[0]['geometry']['location']
            return (location['lat'], location['lng'])
        return (None, None)
    except ApiError as e:
        print(f"Geocode API error for locality '{locality}': {e}")
        return (None, None)
```

3. Searching for Places Using the Google Places API

Once the coordinates are obtained, the script uses the Google Places API to search for places of the specified type near the given coordinates. The search results are collected, and pagination is handled to retrieve more than one page of results if available.

```python
 Function to search for places and handle pagination
def search_places(lat, lng, place_type):
    all_results = []
    page_token = None
```

```
    while True:
        try:
            places_result = gmaps.places_nearby(location=(lat, lng), radius=5000,
type=place_type, page_token=page_token)
            all_results.extend(places_result['results'])
            page_token = places_result.get('next_page_token')
            if not page_token:
                break
        except ApiError as e:
            print(f"Places API error for location ({lat}, {lng}) and type '{place_type}':
{e}")
            break

    return all_results
```

4. Retrieving Place Details

For each place found, the script retrieves detailed information, including name, address, rating, total ratings, and contact number (if available), using the Google Places Details API.

```python
 Function to get details of a place including contact number
def get_place_details(place_id):
    try:
        place_details = gmaps.place(place_id=place_id)
        result = place_details.get('result', {})
        return {
            'Name': result.get('name', 'N/A'),
            'Address': result.get('vicinity', 'N/A'),
            'Rating': result.get('rating', 'N/A'),
            'Total Ratings': result.get('user_ratings_total', 'N/A'),
            'Contact Number': result.get('formatted_phone_number', 'N/A')
        }
    except ApiError as e:
        print(f"Places Details API error for place ID '{place_id}': {e}")
        return {
            'Name': 'N/A',
```

```
            'Address': 'N/A',
            'Rating': 'N/A',
            'Total Ratings': 'N/A',
            'Contact Number': 'N/A'
        }
```

5. Error Handling Mechanisms

The script includes error handling to manage issues such as missing input files, API errors, and invalid data. These errors are caught and appropriate messages are printed to assist with debugging.

```python
 Error handling when reading the Excel file
try:
    df = pd.read_excel(file_path)
except FileNotFoundError as e:
    print(e)
    print(f"Please make sure that the Excel file '{file_path}' exists.")
    raise
except ValueError as e:
    print(e)
    print("Please make sure that the Excel file contains the required columns 'Sheet1' and 'Types'.")
    raise

 Error handling in the geocoding and place details functions
def get_lat_long(locality):
    try:
        geocode_result = gmaps.geocode(locality)
        if geocode_result:
            location = geocode_result[0]['geometry']['location']
            return (location['lat'], location['lng'])
        return (None, None)
    except ApiError as e:
        print(f"Geocode API error for locality '{locality}': {e}")
        return (None, None)
```

```
def get_place_details(place_id):
    try:
        place_details = gmaps.place(place_id=place_id)
        result = place_details.get('result', {})
        return {
            'Name': result.get('name', 'N/A'),
            'Address': result.get('vicinity', 'N/A'),
            'Rating': result.get('rating', 'N/A'),
            'Total Ratings': result.get('user_ratings_total', 'N/A'),
            'Contact Number': result.get('formatted_phone_number', 'N/A')
        }
    except ApiError as e:
        print(f"Places Details API error for place ID '{place_id}': {e}")
        return {
            'Name': 'N/A',
            'Address': 'N/A',
            'Rating': 'N/A',
            'Total Ratings': 'N/A',
            'Contact Number': 'N/A'
        }
```

Putting It All Together

The script loops through each row in the input Excel file, performs geocoding, searches for places, retrieves details, and stores the results in Excel files. The results are saved in batches of 50 entries per file.

```python
Initialize an empty list to store all the results
all_places = []

Loop through each row and search for each locality and type
for index, row in df.iterrows():
    locality = row['Sheet1']
    place_type = row['Types']
    lat, lng = get_lat_long(locality)

    if lat and lng:
```

```
        print(f"Searching for {place_type} near {locality} ({lat}, {lng})")
        places_results = search_places(lat, lng, place_type)

        for place in places_results:
            place_data = get_place_details(place['place_id'])
            place_data['Locality'] = locality
            place_data['Type'] = place_type
            all_places.append(place_data)

             Save results to an Excel file after every 50 entries
            if len(all_places) % 50 == 0:
                df_results = pd.DataFrame(all_places)
                df_results.to_excel(f'places_results_{len(all_places)//50}.xlsx',
index=False)
                print(f"Saved {len(all_places)} entries to
places_results_{len(all_places)//50}.xlsx")

    else:
        print(f"Could not find geocode for {locality}")

 Save any remaining results to an Excel file
if all_places:
    df_results = pd.DataFrame(all_places)
    df_results.to_excel(f'places_results_{(len(all_places)-1)//50 + 1}.xlsx',
index=False)
    print(f"Saved final results to places_results_{(len(all_places)-1)//50 + 1}.xlsx")
```

The implementation effectively automates the process of retrieving and storing place details, ensuring efficiency and accuracy.

9. Testing and Validation

Test Cases

To ensure the functionality and reliability of the "Places Search Script," several test cases were designed to cover different scenarios and edge cases:

1. Basic Functionality Test:
   - Test Case: Input a single locality and one type of place (e.g., "New York" and "restaurant").
   - Expected Result: Retrieve and save details of nearby restaurants in New York to an Excel file.

2. Multiple Localities Test:
   - Test Case: Input multiple localities (e.g., "New York", "San Francisco") and a single type of place (e.g., "cafe").
   - Expected Result: Retrieve and save details of nearby cafes in each locality to separate Excel files.

3. Multiple Types Test:
   - Test Case: Input a single locality and multiple types of places (e.g., "Los Angeles" and ["restaurant", "hotel"]).
   - Expected Result: Retrieve and save details of nearby restaurants and hotels in Los Angeles to separate Excel files.

4. Error Handling Test:
   - Test Case: Provide an invalid API key or modify the input Excel file to contain incorrect data.
   - Expected Result: Verify that appropriate error messages are displayed and the script gracefully handles the errors without crashing.

Validation

After executing each test case, the results were validated to ensure accuracy and completeness:

1. Comparing Expected vs. Actual Results:
   - For each test case, the retrieved data (place names, addresses, ratings, etc.) were compared against expected values based on manual verification or previous knowledge of the localities and places.

2. Excel File Verification:
   - Each Excel file generated by the script was manually reviewed to ensure:
   - Correct formatting of columns and rows.
   - Presence of expected entries (places) and their details.
   - Accuracy of data such as ratings and contact numbers.

3. Functionality Testing:
   - Functional aspects, such as the ability to handle pagination (multiple pages of search results) and error conditions (API errors), were thoroughly tested to ensure the script's robustness.

4. Performance Testing:
   - Performance metrics, including execution time and memory usage, were monitored during testing to ensure the script operates efficiently even with large datasets and under varying conditions.

Conclusion

The testing and validation process confirmed that the "Places Search Script" effectively retrieves, processes, and stores place details from the Google Maps API. The script demonstrated reliability in handling different scenarios and error conditions, providing accurate results consistently. These tests ensure that the script meets its intended functionality and performs optimally for its users.

10. Results and Discussion

Outcomes of the Project

The "Places Search Script" successfully retrieves detailed information about various types of places near specified localities using the Google Maps API. The outcomes of executing the script include the generation of multiple Excel files, each containing up to 50 entries of place details. These files are structured to include essential information such as place names, addresses, ratings, total ratings, contact numbers (if available), locality, and type of place.

Sample Output Files:

1. places_results_1.xlsx: Contains details of restaurants, cafes, and hotels near "New York".
   - Columns: Name, Address, Rating, Total Ratings, Contact Number, Locality, Type

2. places_results_2.xlsx: Includes details of restaurants, cafes, and hotels near "San Francisco".
   - Columns: Name, Address, Rating, Total Ratings, Contact Number, Locality, Type

3. places_results_3.xlsx: Stores information about restaurants, cafes, and hotels near "Los Angeles".
   - Columns: Name, Address, Rating, Total Ratings, Contact Number, Locality, Type

These files serve as valuable resources for further analysis and decision-making regarding localities and types of places.

Comparison with Initial Objectives

The primary objectives of the project were to:
- Automate the retrieval of place details using the Google Maps API.
- Organize retrieved data into structured Excel files.
- Ensure the script handles multiple localities and types of places efficiently.

The script effectively meets these objectives by:

- Reading input data from `localities.xlsx` and processing each locality and type of place.
- Utilizing the Google Geocoding API to convert localities into geographic coordinates.
- Using the Google Places API to search for places based on coordinates and place types.
- Saving results into multiple Excel files, ensuring data organization and ease of access.

Analysis of Results

Upon analyzing the retrieved data, several insights can be observed:

- Distribution of Places: The script successfully captures a diverse range of places such as restaurants, cafes, and hotels across different localities.
- Rating Trends: Analysis of ratings and total ratings provides insights into the popularity and quality of various places within each locality.
- Geographical Insights: Geographic distribution and concentration of places can be inferred based on the retrieved data, aiding in location-based analysis and decision-making.

Potential Improvements and Future Work

While the current implementation of the "Places Search Script" achieves its core functionalities, several areas for potential improvement and future work include:

- Enhanced User Interface: Developing a user-friendly interface to input localities and types of places directly.
- Real-time Updates: Implementing mechanisms to fetch real-time data from APIs for up-to-date information.
- Additional Data Sources: Integrating multiple data sources beyond Google Maps for comprehensive place details.
- Advanced Analysis Features: Adding capabilities for statistical analysis, visualization, and predictive modeling based on retrieved data.

By addressing these areas, the script can further enhance its utility and provide more robust support for various applications such as tourism planning, business location analysis, and urban development strategies.

This section summarizes the outcomes, assesses the alignment with initial objectives, provides insights from the retrieved data, and outlines potential avenues for future enhancements and expansions of the "Places Search Script."

11. Conclusion

Summary of the Project

The development of the "Places Search Script" aimed to automate the retrieval and organization of detailed place information using the Google Maps API. This script successfully achieves its objectives by efficiently gathering data on various types of places near specified localities and structuring it into accessible Excel files. By leveraging geocoding and places APIs, the script provides a robust solution for retrieving, processing, and saving place details, enhancing data accessibility and usability.

Key Findings

Through the implementation and testing of the script, several key findings emerged:

- Efficiency: The script effectively handles multiple localities and types of places, demonstrating its capability to scale with varying inputs.
- Accuracy: Retrieval and organization of place details, including names, addresses, ratings, and contact numbers, are performed accurately and reliably.
- Usability: Generated Excel files are structured to facilitate further analysis and decision-making related to places of interest, supporting applications across various domains.

Overall Experience

The development process provided valuable insights and experiences:

- Technical Skills: Enhanced proficiency in Python programming, data manipulation using pandas, and API integration with Google Maps.
- Problem Solving: Addressing challenges such as handling API errors, managing large datasets, and ensuring data integrity.
- Project Management: Planning and executing a structured development process, including requirement analysis, implementation, testing, and documentation.

Overall, the project offered a hands-on opportunity to apply theoretical knowledge in a practical setting, reinforcing concepts in software development, data handling, and API utilization. The experience not only strengthened technical competencies but also cultivated problem-solving abilities and project management skills essential for future endeavors in the field of software engineering and data analytics.

This conclusion section summarizes the achievements, key insights, and personal growth derived from developing the "Places Search Script," highlighting its impact and relevance in leveraging API technology for data-driven solutions.

12. References

1. Google Maps Platform Documentation. Available at: [Google Maps Platform Documentation](https://developers.google.com/maps/documentation)
2. Python Documentation. Available at: [Python Documentation](https://docs.python.org/3/)
3. pandas Documentation. Available at: [pandas Documentation](https://pandas.pydata.org/docs/)
4. googlemaps Python Client Documentation. Available at: [googlemaps Python Client Documentation](https://googlemaps.github.io/google-maps-services-python/docs/)
5. xlsxwriter Documentation. Available at: [xlsxwriter Documentation](https://xlsxwriter.readthedocs.io/)

These references include documentation for the libraries used (pandas, googlemaps, xlsxwriter), as well as the Google Maps API documentation which provided essential guidance for implementing geocoding and places functionalities within the script. If you have cited specific sources beyond these, make sure to include them in your list as well.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my Head of Department, MS Panday, and my Professor, Manjari Gupta, for their guidance and support throughout the development of this project. I would also like to thank my friends and family for their encouragement and support.

I am grateful for the opportunity to have worked on this project and I am confident that the skills and knowledge that I have gained will be valuable in my future endeavors.

I would also like to thank my friends and family for their encouragement and support throughout the development of this project. They have been a source of strength and motivation and I am grateful for their support.

I am confident that the skills and knowledge that I have gained through this project will be valuable in my future endeavors. I am excited to apply these skills and knowledge to new challenges and I am confident that I will be successful.