Assignment1 Problem Statement-9

Problem Statement: Knowledge Graph Application

Objective:

Develop a web-based Knowledge Graph application that allows users to input and visualize relationships between entities. The application should provide a user-friendly interface to add relationships, query the graph, and display the results.

Group Number: 26

Group Members:

- DIVYANSHU SINGH (2022AC05011)
- SOURAV PANI (2022AC05206)
- SWARAJYA RANJAN BHANJA (2022AC05599)
- BALAJI T (2022AC05612)

Application Overview-

The knowledge graph application allows users to add, visualize, and query relationships between various entities. Users can manually input relationships, upload CSV files for bulk data entry, and query specific entities to see their connections. The application features an interactive graph visualization, a data table for managing relationships, and responsive design elements.

Index-

Instructions for Running the Application Locally

Prerequisites

1. **Python**: Ensure you have Python installed (preferably version 3.6 or higher).
2. **pip**: Ensure you have `pip` installed to manage Python packages.
3. **Virtual Environment**: It's recommended to use a virtual environment to manage dependencies.

Step-by-Step Guide

1. **Clone the Repository** If the project is in a repository, clone it using:

   ```
   bash
   Copy code
   git clone <repository_url>
   cd <repository_directory>
   ```

2. **Set Up Virtual Environment** Create and activate a virtual environment:

   ```
   bash
   Copy code
   python -m venv venv
   source venv/bin/activate  # On Windows: venv\Scripts\activate
   ```

3. **Install Dependencies** Install the required Python packages:

   ```
   bash
   Copy code
   pip install flask networkx
   ```

4. **Directory Structure** Ensure your directory structure looks like this:

   ```
   your-flask-app/
   ├── app.py
   ├── templates/
   │   └── index.html
   ├── static/
   │   ├── image.png
   │   ├── styles.css
   │   └── script.js
   └── ...
   ```

5. **Add Frontend Files** Place your frontend files in the appropriate directories:
   - `index.html` in the `templates/` directory
   - `styles.css` and `script.js` in the `static/` directory
   - `image.png` (the logo image) in the `static/` directory

6. **Run the Flask Application** Start the Flask development server:

```
bash
Copy code
python app.py
```

   You should see output indicating that the Flask server is running on http://127.0.0.1:5000/.

```
C:\Users\divysing\Downloads\Final NLPA\NLPA Assignment 1>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 504-024-609
```

7. **Access the Application** Open your web browser and navigate to http://127.0.0.1:5000/ to access the application.

Design Choices and Challenges

Design Choices

1. **Flask Framework**:
    - **Reason**: Flask is a lightweight and flexible web framework for Python. It provides the essentials needed to build web applications while allowing developers to use their preferred tools and libraries. Flask's minimalistic approach makes it easy to learn and use, making it ideal for this project.
    - **Usage**: Flask was used to set up the server, handle routing, and serve the HTML and static files. It also handled API requests for adding relationships, uploading CSV files, querying nodes, and deleting the graph.

2. **NetworkX for Graph Management**:
    - **Reason**: NetworkX is a comprehensive library for the creation, manipulation, and study of complex networks. It supports operations on graphs and is highly compatible with Python, making it a suitable choice for backend graph management.
    - **Usage**: NetworkX was used to maintain the graph data structure, perform operations like adding nodes and edges, and query relationships between nodes. The graph was then converted to a dictionary format for easy JSON serialization and transmission to the frontend.

3. **Bootstrap for Styling**:
    - **Reason**: Bootstrap is a popular CSS framework that provides a responsive grid system and pre-designed components, making it easy to create modern and responsive web pages. It helps ensure the application looks good on various devices.
    - **Usage**: Bootstrap was used for the overall layout, styling forms, buttons, tables, and other UI components. It provided a clean and professional look with minimal custom CSS needed.

4. **Vis.js for Graph Visualization**:
    - **Reason**: Vis.js is a dynamic, browser-based visualization library that is perfect for visualizing data and interactive graphs. It offers rich interaction and customization options, making it ideal for displaying the knowledge graph.
    - **Usage**: Vis.js was used to render the knowledge graph in the browser. Nodes and edges were dynamically added or removed based on user interactions like adding relationships, uploading CSV files, or querying nodes.
    -

5. **DataTables for Tabular Data**:
    - **Reason**: DataTables is a jQuery plugin that enhances HTML tables with advanced features like search, pagination, and sorting. It provides an interactive way to display tabular data.
    - **Usage**: DataTables was used to display the list of relationships in a searchable and sortable table. This helped in managing and editing relationships efficiently.

6. **jQuery for DOM Manipulation**:
    - **Reason**: jQuery simplifies HTML DOM tree traversal and manipulation, event handling, and AJAX interactions. It is widely used and compatible with many plugins.
    - **Usage**: jQuery was used for handling form submissions, making AJAX requests to the backend, and updating the DOM based on the responses.

Challenges Faced

1. **Integrating Different Libraries**:
   - **Challenge**: Ensuring compatibility and smooth interaction between Flask (backend), NetworkX (graph management), Vis.js (visualization), and DataTables (tabular data) was challenging.
   - **Solution**: Careful planning and testing were required to ensure all components worked together seamlessly. Documentation and community forums were helpful in resolving integration issues.

2. **Graph Visualization Performance**:
   - **Challenge**: Handling large datasets efficiently for real-time visualization posed a performance challenge. Large graphs could lead to slow rendering and interactions.
   - **Solution**: Optimized data fetching and rendering strategies were employed. For example, limiting the number of nodes displayed initially and providing options to query and focus on specific parts of the graph helped manage performance.

3. **User Interface Design**:
   - **Challenge**: Creating a user-friendly interface that is both functional and visually appealing required iterative design and user feedback.
   - **Solution**: Utilizing Bootstrap for responsive design and consistent styling helped maintain a clean UI. Custom CSS was added where necessary to meet specific design requirements. Regular testing and feedback loops were critical in refining the UI.

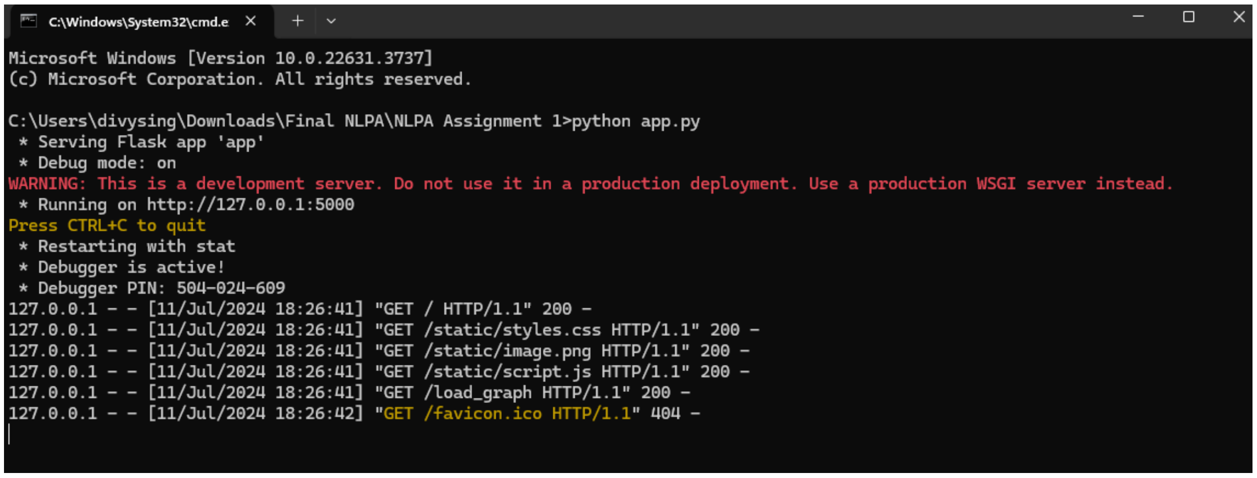
4. **Error Handling**:
   - **Challenge**: Ensuring the application gracefully handles various errors, such as invalid input, network issues, or server errors, was essential for a robust user experience.
   - **Solution**: Comprehensive error handling was implemented in both the frontend and backend. User-friendly error messages and alerts were provided, and the application was designed to remain stable under error conditions.

5. **Data Consistency and Validation**:
   - **Challenge**: Maintaining data consistency and validating user input to prevent invalid data entries or duplicates required careful planning.
   - **Solution**: Validation checks were implemented in the backend to ensure data integrity. User input was validated for required fields, and duplicate entries were prevented by checking existing graph data before adding new relationships.

6. **Cross-Browser Compatibility**:
   - **Challenge**: Ensuring the application works consistently across different web browsers required thorough testing and adjustments.
   - **Solution**: The application was tested on major browsers like Chrome, Firefox, Safari, and Edge. Any browser-specific issues were addressed using appropriate CSS and JavaScript adjustments.

Screenshots of the Application

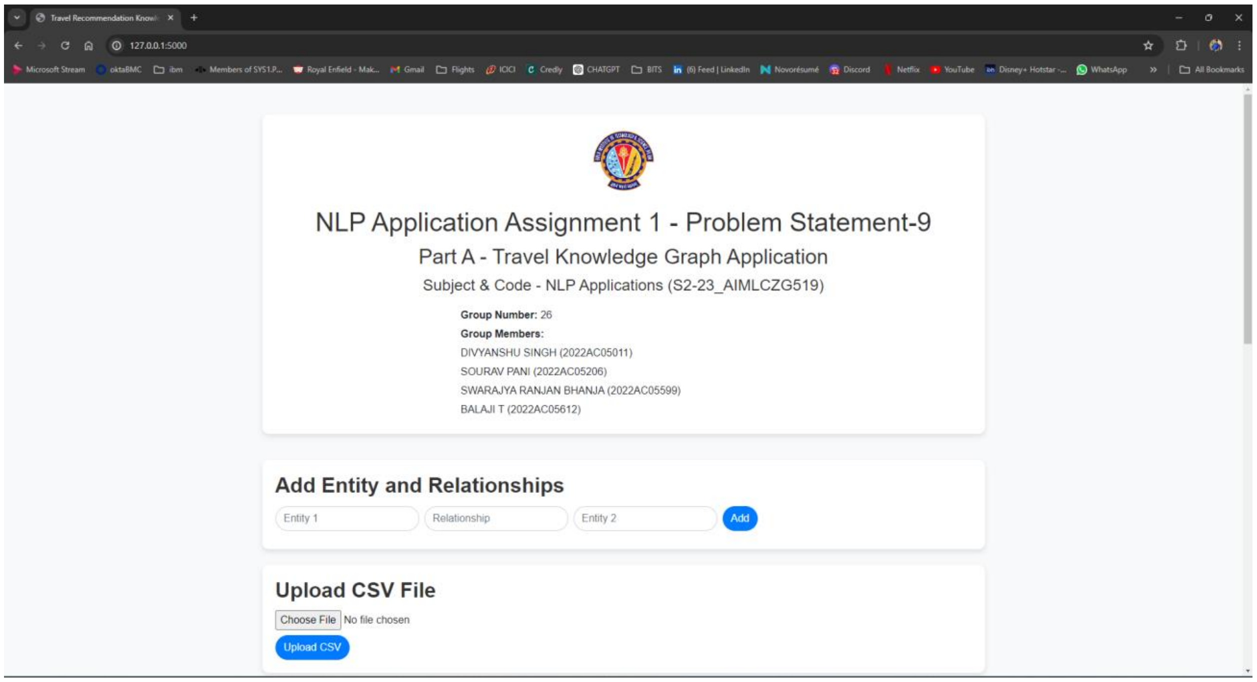
1. **Starting the Python App**
   This image shows the command-line interface (CLI) where the Flask application is started using the command `python app.py`. The output indicates that the Flask server is running in debug mode on `http://127.0.0.1:5000/`. This step is crucial to initialize the backend server for the application.



```
Microsoft Windows [Version 10.0.22631.3737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\divysing\Downloads\Final NLPA\NLPA Assignment 1>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 504-024-609
127.0.0.1 - - [11/Jul/2024 18:26:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [11/Jul/2024 18:26:41] "GET /static/styles.css HTTP/1.1" 200 -
127.0.0.1 - - [11/Jul/2024 18:26:41] "GET /static/image.png HTTP/1.1" 200 -
127.0.0.1 - - [11/Jul/2024 18:26:41] "GET /static/script.js HTTP/1.1" 200 -
127.0.0.1 - - [11/Jul/2024 18:26:41] "GET /load_graph HTTP/1.1" 200 -
127.0.0.1 - - [11/Jul/2024 18:26:42] "GET /favicon.ico HTTP/1.1" 404 -
```

2. **Overview of Webpage**
   This image provides a high-level overview of the knowledge graph application webpage. The header includes details about the assignment, group members, and the application purpose. Below the header, there are sections for adding entity relationships, uploading CSV files, querying the graph, viewing the data table, and visualizing the knowledge graph.



NLP Application Assignment 1 - Problem Statement-9

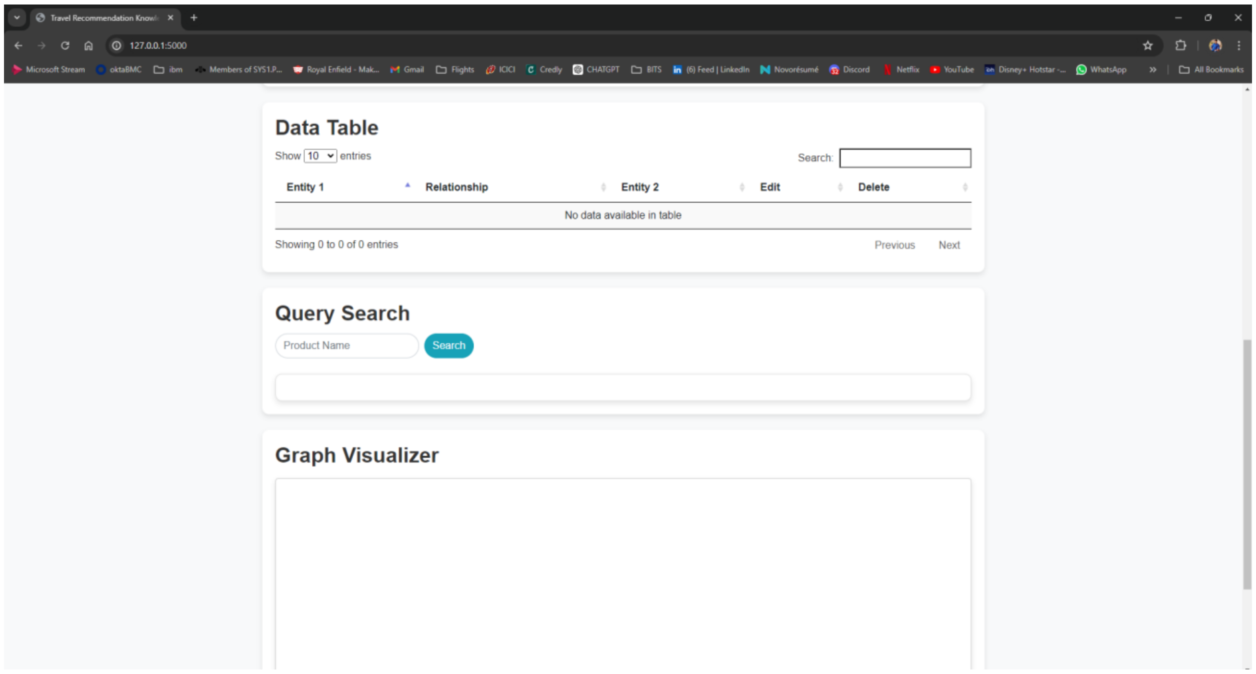
Part A - Travel Knowledge Graph Application

Subject & Code - NLP Applications (S2-23_AIMLCZG519)

Group Number: 26
Group Members:
DIVYANSHU SINGH (2022AC05011)
SOURAV PANI (2022AC05206)
SWARAJYA RANJAN BHANJA (2022AC05599)
BALAJI T (2022AC05612)

Add Entity and Relationships

Upload CSV File



Data Table

Show 10 entries

Search:

Entity 1	Relationship	Entity 2	Edit	Delete
No data available in table				

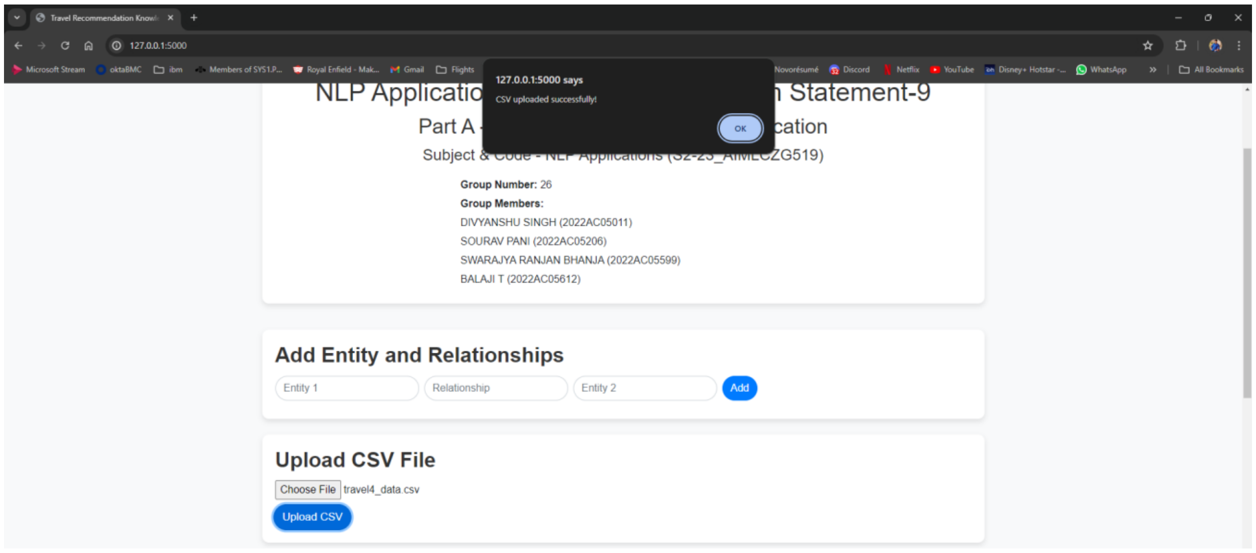
Showing 0 to 0 of 0 entries

Query Search

Graph Visualizer

3. Adding CSV File

This image shows the user in the process of uploading a CSV file. The file input allows users to select a CSV file from their computer, and the "Upload CSV" button sends the file to the server for processing. A success message confirms that the CSV was uploaded successfully.



127.0.0.1:5000 says

CSV uploaded successfully!

NLP Applicatio n Statement-9

Part A - cation

Subject & Code - NLP Applications (S2-23_AIMLCZG519)

Group Number: 26

Group Members:

DIVYANSHU SINGH (2022AC05011)

SOURAV PANI (2022AC05206)

SWARAJYA RANJAN BHANJA (2022AC05599)

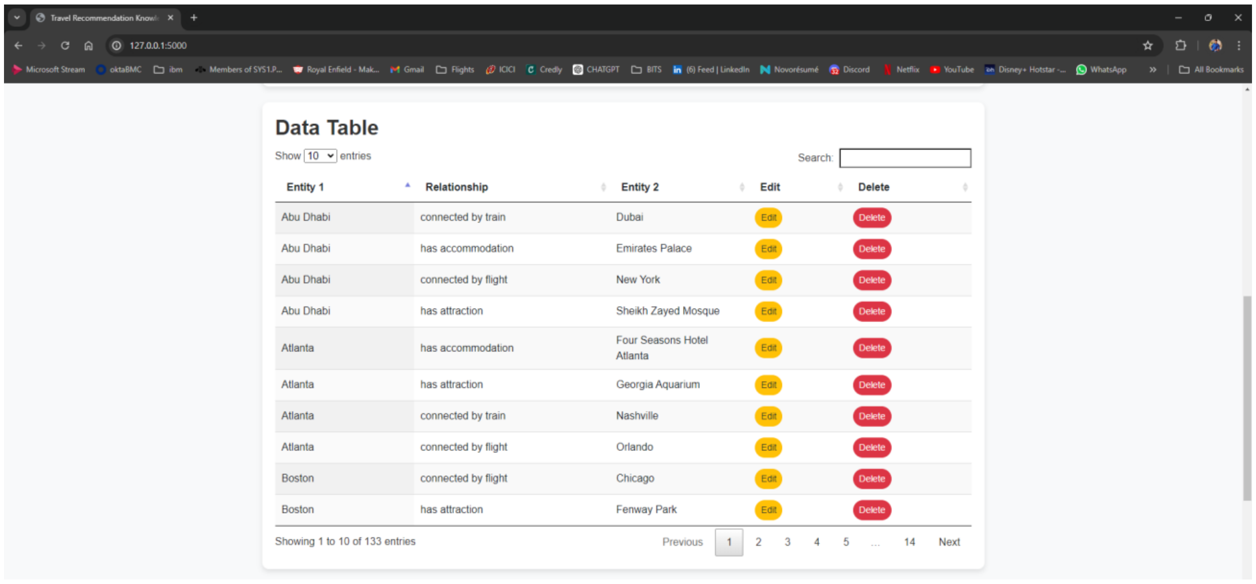
BALAJI T (2022AC05612)

Add Entity and Relationships

Upload CSV File

Choose File travel4_data.csv

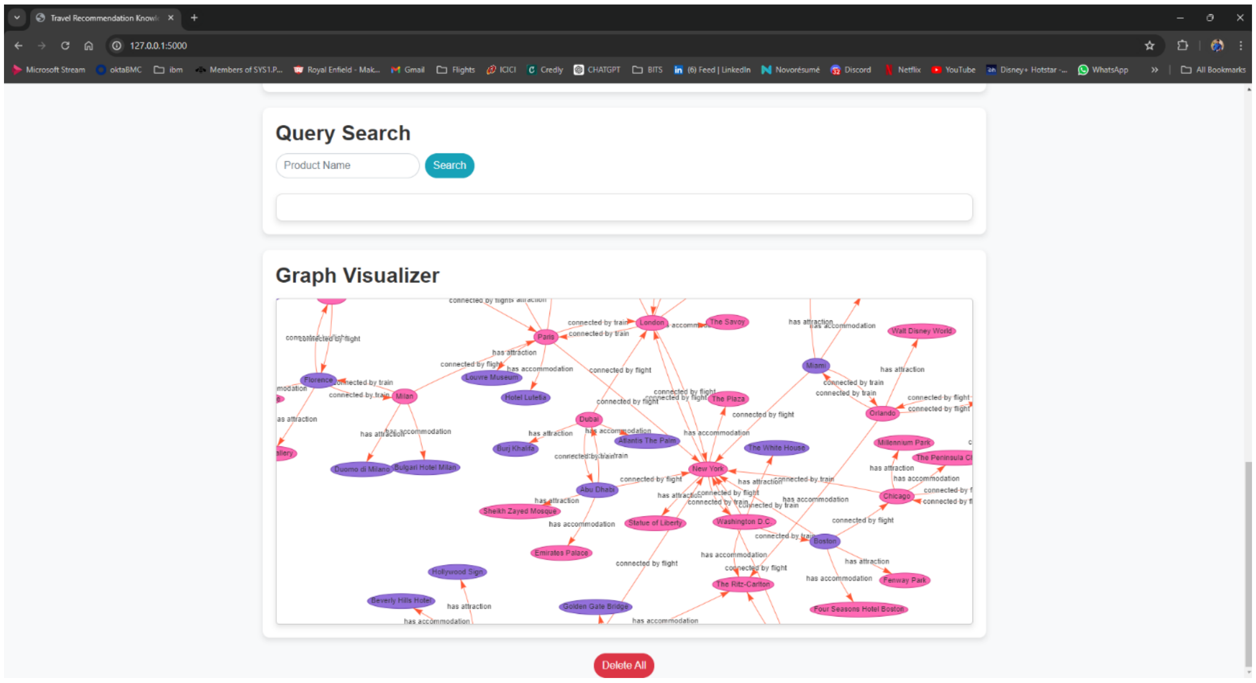
4. **Populated Table of CSV**
This image shows the data table populated with entries from the uploaded CSV file. Each row represents a relationship between two entities, along with buttons to edit or delete each entry. The table supports searching, sorting, and pagination for better data management.



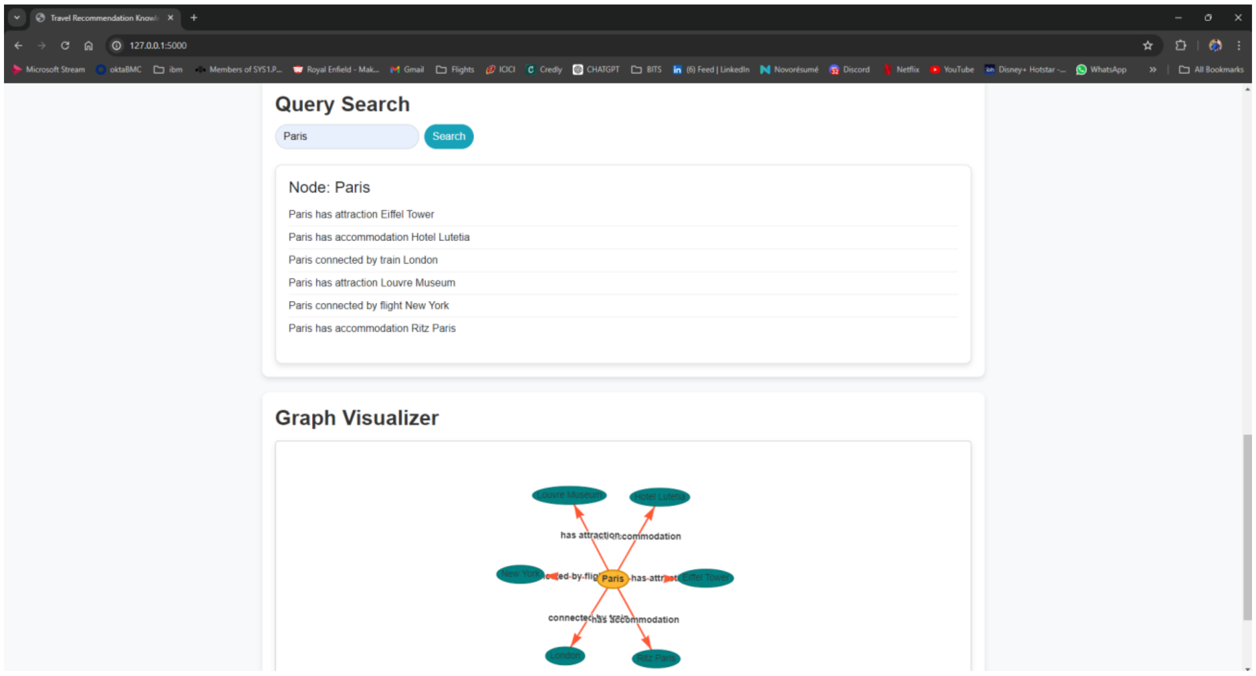
Entity 1	Relationship	Entity 2	Edit	Delete
Abu Dhabi	connected by train	Dubai	Edit	Delete
Abu Dhabi	has accommodation	Emirates Palace	Edit	Delete
Abu Dhabi	connected by flight	New York	Edit	Delete
Abu Dhabi	has attraction	Sheikh Zayed Mosque	Edit	Delete
Atlanta	has accommodation	Four Seasons Hotel Atlanta	Edit	Delete
Atlanta	has attraction	Georgia Aquarium	Edit	Delete
Atlanta	connected by train	Nashville	Edit	Delete
Atlanta	connected by flight	Orlando	Edit	Delete
Boston	connected by flight	Chicago	Edit	Delete
Boston	has attraction	Fenway Park	Edit	Delete

Showing 1 to 10 of 133 entries

5. **Knowledge Graph Visualization**
This image displays the knowledge graph visualization with nodes and edges representing entities and their relationships. Different colors are used to distinguish between entity types and relationships. This interactive graph allows users to explore the connections visually.



6. **Query Search Result**
This image shows the result of a query search for the entity "Paris". The result section lists all relationships involving Paris, and the graph visualizer displays only the nodes and edges relevant to Paris, providing a focused view of the queried entity's connections.



Query Search

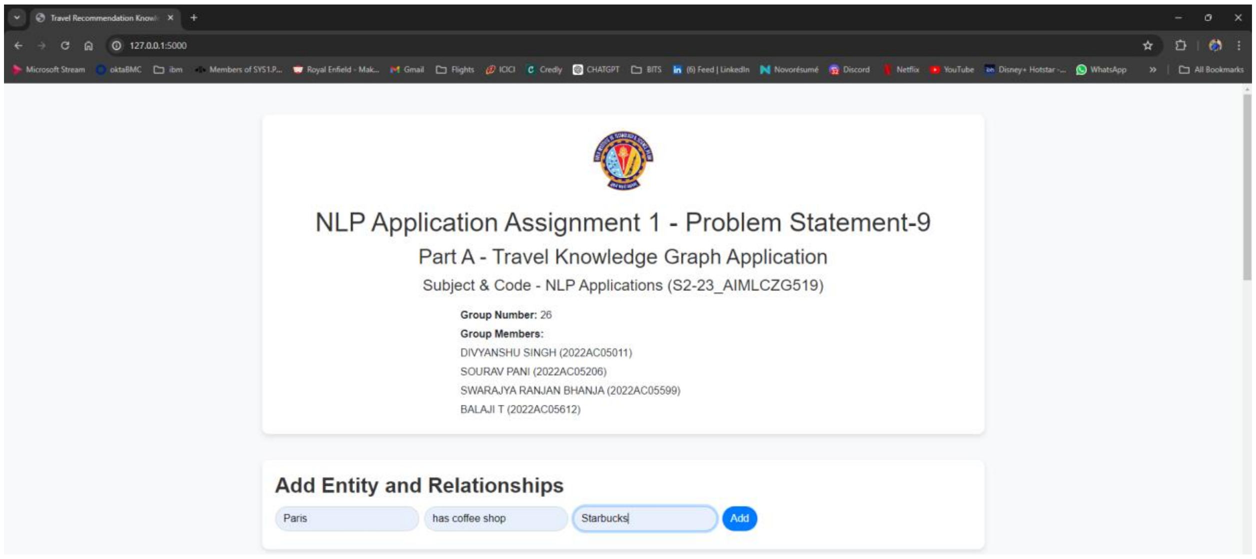
Paris | Search

Node: Paris

Paris has attraction Eiffel Tower
Paris has accommodation Hotel Lutetia
Paris connected by train London
Paris has attraction Louvre Museum
Paris connected by flight New York
Paris has accommodation Ritz Paris

Graph Visualizer

7. **Adding a Relationship**
This image shows the user adding a new relationship between entities using the form provided. The user inputs "Paris" as Entity 1, "has coffee shop" as the relationship, and "Starbucks" as Entity 2. The "Add" button submits the form to add the relationship to the graph.



NLP Application Assignment 1 - Problem Statement-9

Part A - Travel Knowledge Graph Application

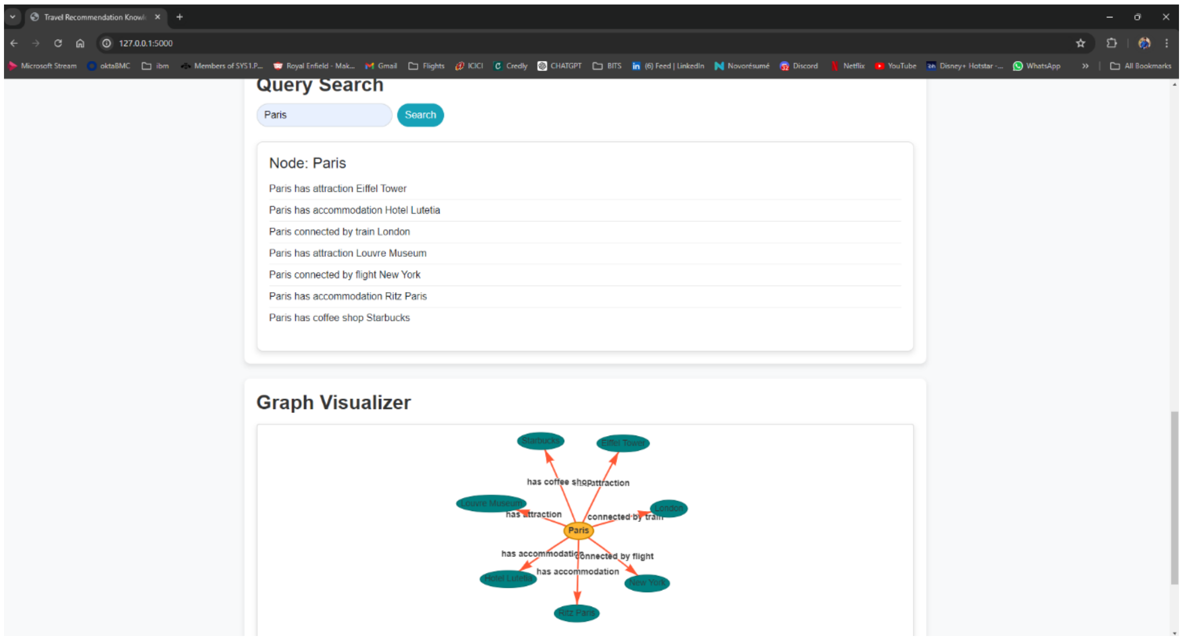
Subject & Code - NLP Applications (S2-23_AIMLCZG519)

Group Number: 26
Group Members:
DIVYANSHU SINGH (2022AC05011)
SOURAV PANI (2022AC05206)
SWARAJYA RANJAN BHANJA (2022AC05599)
BALAJI T (2022AC05612)

Add Entity and Relationships

Paris | has coffee shop | Starbucks | Add

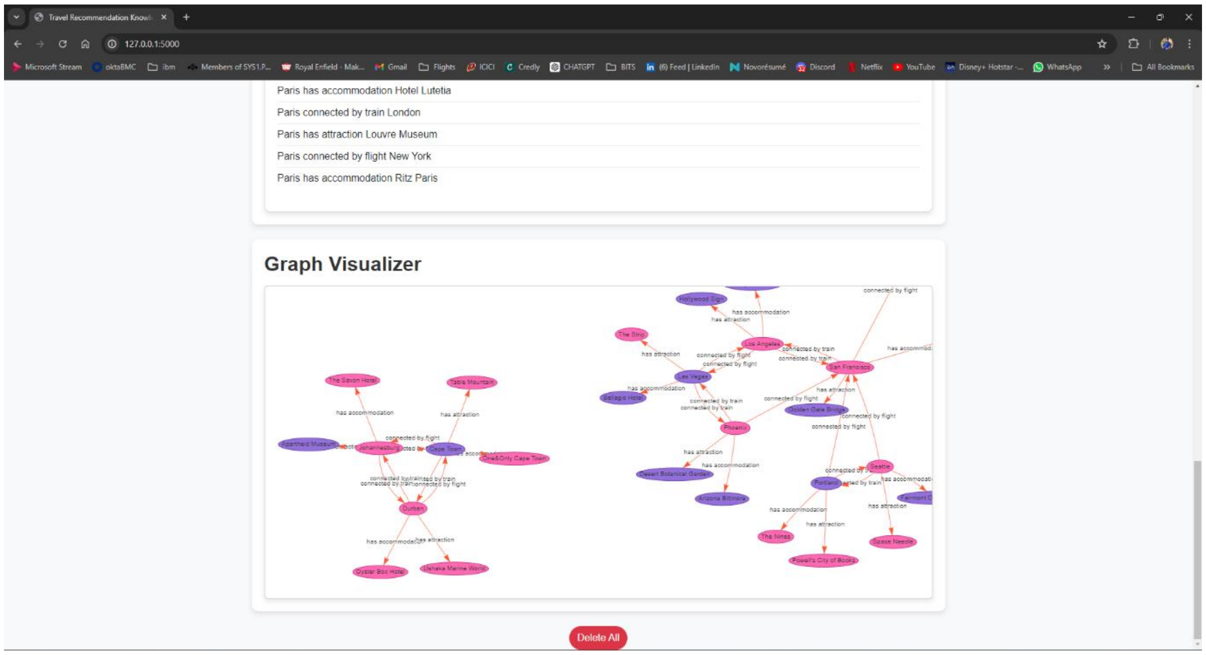
8. **Addition Result**
This image displays the result after adding the new relationship between "Paris" and "Starbucks". The relationship is shown in the graph visualizer, updating in real-time to reflect the new connection, demonstrating the interactive nature of the application.



Node: Paris

Paris has attraction Eiffel Tower
Paris has accommodation Hotel Lutetia
Paris connected by train London
Paris has attraction Louvre Museum
Paris connected by flight New York
Paris has accommodation Ritz Paris
Paris has coffee shop Starbucks

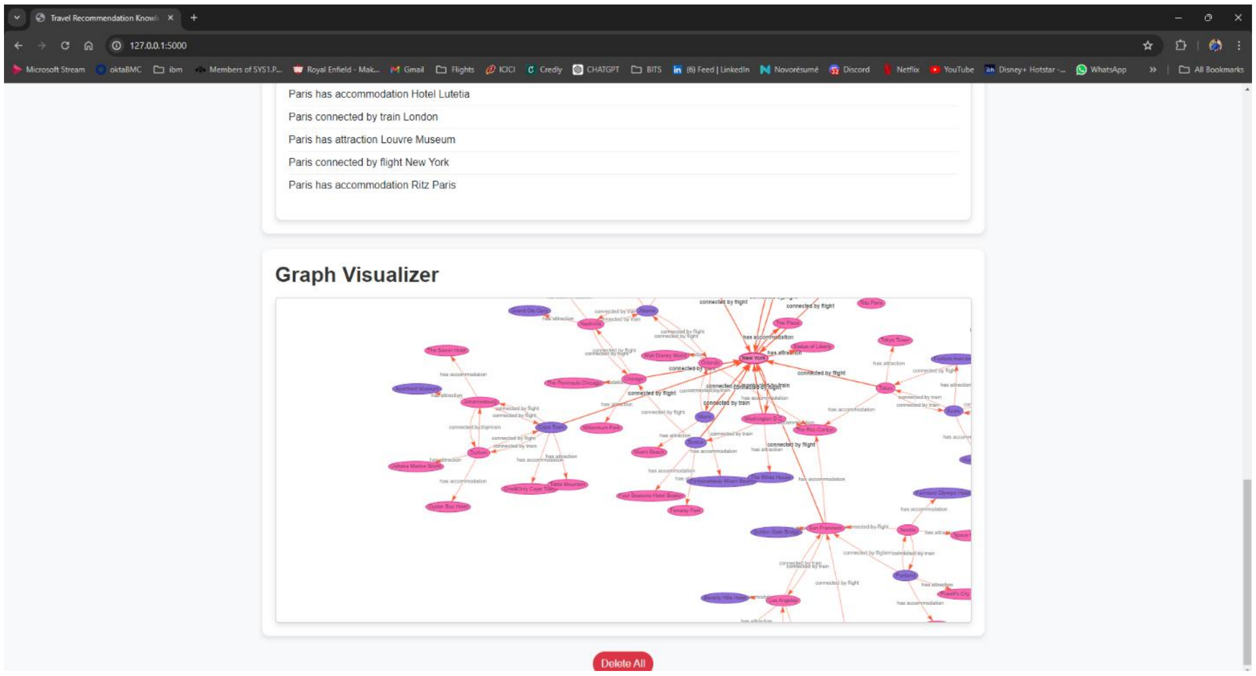
9. **Separate Graph Visualization**
This image shows a section of the knowledge graph where Cape Town has a separate graph and is not connected to the Main division graph. It highlights how the graph can display different sets of relationships in a visually distinct manner.



Paris has accommodation Hotel Lutetia
Paris connected by train London
Paris has attraction Louvre Museum
Paris connected by flight New York
Paris has accommodation Ritz Paris

10. Merged Graph Visualization

This image displays the interconnected view of the knowledge graph, showing Cape Town now connected to New York and their relationships merged into a single, cohesive visualization. It demonstrates the application's capability to handle and display complex networks of data.



Paris has accommodation Hotel Lutetia
Paris connected by train London
Paris has attraction Louvre Museum
Paris connected by flight New York
Paris has accommodation Ritz Paris

Graph Visualizer

Conclusion

The knowledge graph application successfully integrates various technologies to provide a robust and user-friendly tool for managing and visualizing relationships between entities. Flask serves as the backbone, managing server-side operations and routing, while NetworkX handles the graph data structure. The frontend leverages Bootstrap for styling, Vis.js for interactive graph visualization, and DataTables for managing tabular data.

Challenges such as integrating different libraries, managing performance, designing a user-friendly interface, handling errors, ensuring data consistency, and achieving cross-browser compatibility were addressed through careful planning, iterative testing, and leveraging community resources and documentation. The resulting application is a powerful tool for creating and exploring knowledge graphs, with a clean and responsive user interface.