Georgio Zeilaa

[Email address]

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

Development of a Web Application for Visualising Geospatial Knowledge Graphs

[Document subtitle]

Table of Contents

[Chapter 1: Project Statement, Aim, and Objectives 3](#_Toc205393825)

[1.1 Introduction 4](#_Toc205393826)

[Chapter 2: Literature Review 5](#_Toc205393827)

[2.1 Introduction to knowledge graphs 5](#_Toc205393828)

[2.2 Existing web applications for data visualisation using knowledge graphs 5](#_Toc205393829)

[2.3 NoSQL Databases 6](#_Toc205393830)

[Chapter 3: Background 7](#_Toc205393831)

[3.1 Choosing a web framework 7](#_Toc205393832)

[3.2 ReactJS 7](#_Toc205393833)

[3.2.1 Examples of map components 7](#_Toc205393834)

[3.2.2 GUI components 7](#_Toc205393835)

[3.3 Neo4j to manage and store the data 8](#_Toc205393836)

[Chapter 4: Design 9](#_Toc205393837)

[4.1 Overview 9](#_Toc205393838)

[4.2 Requirements 9](#_Toc205393839)

[4.2.1 The data provided 9](#_Toc205393840)

[4.2.2 Use cases 9](#_Toc205393841)

[4.2.3 GUI 10](#_Toc205393842)

[4.3 Modelling 10](#_Toc205393843)

[4.3.1 Data modelling 10](#_Toc205393844)

[4.3.2 Graph modelling 11](#_Toc205393845)

[Chapter 5: Implementation 12](#_Toc205393846)

[5.1 Programming Language 12](#_Toc205393847)

[5.2 Version control 12](#_Toc205393848)

[5.3 Neo4J 12](#_Toc205393849)

[5.3.1 Importing data 13](#_Toc205393850)

[5.4 Cypher query 18](#_Toc205393851)

[5.5 Overview ReactJS 20](#_Toc205393852)

[5.5.1 Use cases GUI implementation 21](#_Toc205393853)

[5.5.2 Neo4J Driver 22](#_Toc205393854)

[5.5.3 React-Leaf 23](#_Toc205393855)

[5.5.4 DeckGL 23](#_Toc205393856)

[Chapter 6: Evaluation & Conclusion 24](#_Toc205393857)

[Chapter 7: Future Work 25](#_Toc205393858)

[Chapter 8: Reflection 26](#_Toc205393859)

[References 27](#_Toc205393860)

[List of Figures 29](#_Toc205393861)

[List of Tables 30](#_Toc205393862)

[Appendices 31](#_Toc205393863)

# Chapter 1: Project Statement, Aim, and Objectives

**Project statement:**

To produce a webapp that provides different ways of visualising the geospatial knowledge graph.

**Aim:**

To be able to produce a webapp that visualises the geospatial knowledge graphs with different types of visuals.

**Objectives:**

* To understand what it takes to create a webapp that previews the geospatial knowledge graphs.
  + Background research on existing solutions of the current problem.
  + Attempt to use an existing webapp that has a similar solution.
* Compare different types of visuals that represent the geospatial knowledge graph.
  + Produce a solution for each visual type.
  + Analyse and compare the outcome of each visual type.
* Compare webapp frameworks.
  + Research webapp frameworks that work best for visualising knowledge graphs.
  + Implement a webapp framework with geospatial knowledge graph visuals.
* To test the webapp with knowledge graphs loaded.
  + To test the functionality of the webapp.
  + To explore how well it is for the mobile screen size and how this can be done in the future as an improvement.

## 1.1 Introduction

# Chapter 2: Literature Review

## 2.1 Introduction to knowledge graphs

Knowledge graphs represents real world knowledge for example a map with cities and countries. One country can be close to another country and that country has multiple cities. The connection between the two countries (nodes) can be the direction of where the other country is, this is called an edge (the connection). Knowledge base was used a long time ago, it was used in the 1970s reasoning and problem solving, MYCIN. This had 600 rules based on an expert system for medical diagnosis. The concept of a knowledge graph gained popularity when google search engine introduced their knowledge fusion framework called knowledge vault. [1] The knowledge vault is meant to get information from various sources across the web, this can be people, places, organisations which are in the knowledge graph. [2]

## 2.2 Existing web applications for data visualisation using knowledge graphs

Kepler.gl provides various ways to load geospatial data onto a map. The data can be in a form of files, URLs, or load from cloud storage such as google drive. Once the data is loaded, there is a filter option where it can show data if a value is higher than a specific number. For example, loading the new york city taxi data, there is a field called passenger\_count which can be used to show specific number of passengers who used the taxi in new york. [3]

Kepler is built on react and redux web framework technologies, the project also has embedded ability so it can be added to other websites as a small sized window to view the map with the loaded data points of your choice. The project is available on GitHub and it is open source, it is built upon mapbox which is a paid service that provides maps, navigation, and more. Kepler can also be integrated into react web projects by installing it as a component. [3]

Using kepler.gl,

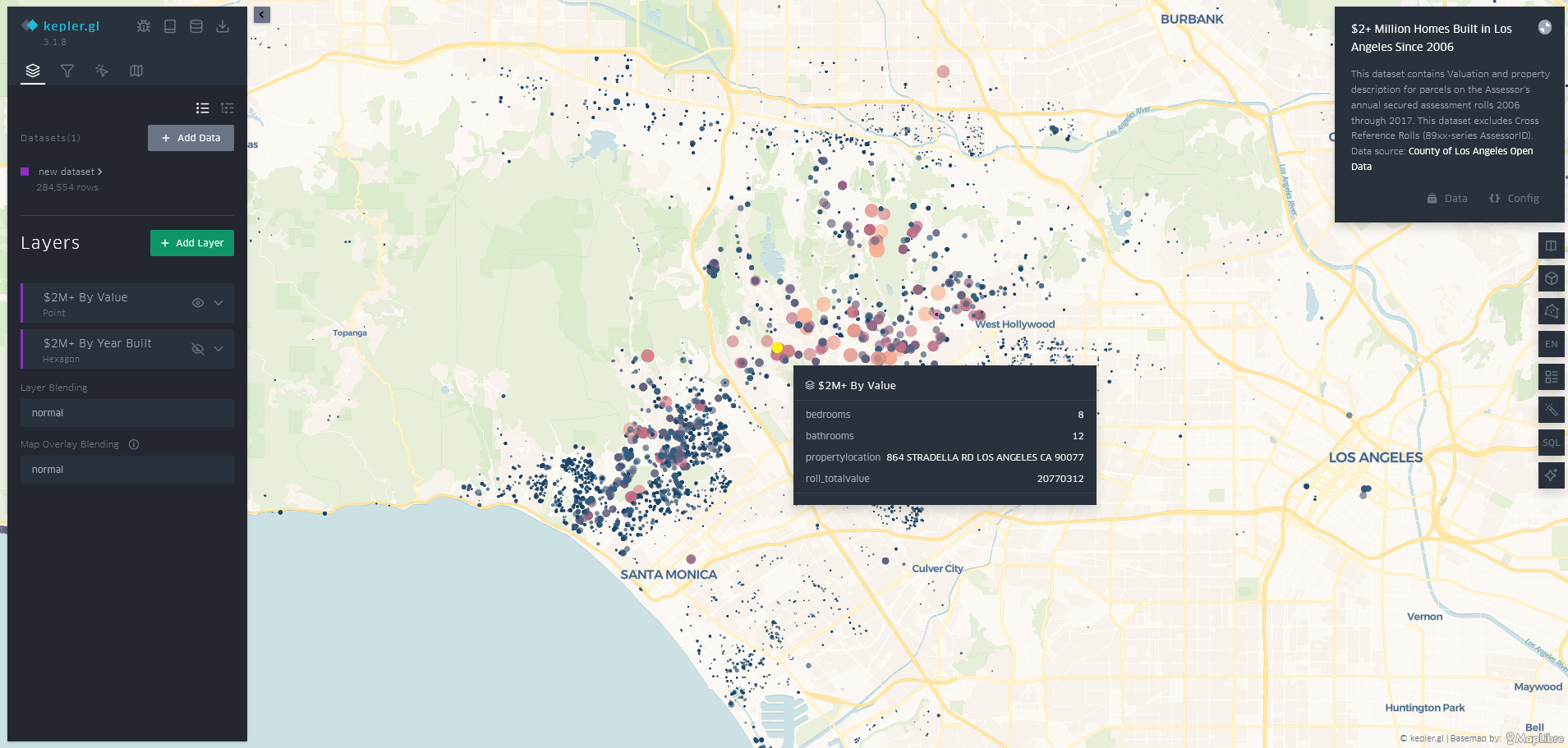


Figure 1 Sample data loaded in Kepler.gl

The above figure shows data points loaded onto a map. Those data points are a sample of data which represents the “$2+ Million Homes Built in Los Angeles Since 2006”. When clicking on one of the points on the map information about that point appears. It shows information about the house built such as, number of bedrooms, bathrooms, the location of the property (address) and the total value.

One the left-hand side there is a panel that allows for filtering, adding more layers such as a line between two points. In the figure below, there is an option for hexagon layer to show the year built by having a different colour represent 2 years incrementally.

A map of different colored hexagons

AI-generated content may be incorrect.

Figure 2 Layer example in Kepler.gl

## 2.3 NoSQL Databases

This is a graph database management system, this allows the use of knowledge graphs to be stored and utilised in Neo4J.

The paper “State-of-the-Art Geospatial Information Processing in NoSQL Databases” goes in depth into NoSQL databases, including Neo4J. Neo4J offers feature specific to knowledge graphs such as specific functionality to traverse through the graph in an efficient manner. It even has a spatial feature called Neo4J Spatial, it offers support for seven geometry types: point, linestring, polygon, multipoint, multilinestring, multipolygon, and geometrycollection. This indicated as to how in depth the feature Neo4J has to offer and how powerful this tool can be when used with geospatial data. There are other options than Neo4J, an example is MongoDB, this is also a NoSQL database, it has some advantages which include the queries of “within” and “intersection” in their time to take to load.

The query language that is used in Neo4J is cypher, it is SQL-like but graph oriented, it is easy to understand since it is readable and expressive. MongoDB uses JSON based syntax which can be more difficult than cypher in Neo4J since it can get verbose and nested. [4]

# Chapter 3: Background

## 3.1 Choosing a web framework

Why ReactJS and how it can be used to show knowledge graphs and components:

<https://blog.tomsawyer.com/react-js-graph-visualization>

Choosing a web framework to run the visuals of the geospatial knowledge graph data need to be optimised for reliability and versatile. Reliability when it comes to speed of loading the webpage and versatility in terms of flexibility when it comes to adding features to the website by using existing components. Components are an example of plugins, for example it can be react leaf which is a map library that can be added to a ReactJS project, this helps speed up the development time. ReactJS uses JavaScript language, this helps by having the code modular, simple to use, and having a strong community by having third party libraries widely accessible to ReactJS. Compared to other frameworks such as vue or angular, angular was actually faster than ReactJS according to Redware study. Overall, this is a compromise in potential speed loss versus the flexibility of the community support and reliability is a trade-off that might be worth making. [5]

## 3.2 ReactJS

### 3.2.1 Examples of map components

An example of where ReactJS was used with geospatial knowledge graph an article titled: “GeoGraphVis: A Knowledge Graph and Geovisualization Empowered Cyberinfrastructure to Support Disaster Response and Humanitarian Aid”. It provides the use of ReactJS with a third party library called DeckGL for adding a map to the website to load the data visually see it. It allows the selection of data variables on the map and filter options. It is used in the article to filter by disaster type and ability to view the results as a bar chart. [6]

Another example of a ReactJS map component is react-leaf, this is an open source project which provides map component for ReactJS and other frameworks. It is quiet simple to implement compared to DeckGL since it has 2D rendering and it’s great for small to medium sized data. It has basic maps and interactive UI but not as detailed as DeckGL with its 3D maps and animated features. [7]

### 3.2.2 GUI components

GUI components in ReactJS are very helpful in terms of the plug and play capabilities. Normally a developer would need to develop a CSS file, this file contains the styles that will be used in the webapp which in this case would be ReactJS. Using libraries such as MUI, this can help the developer in spending less time worrying about the styles and more time developing other parts of the webapp. GUI components that would be suitable for this project would need to have components such as a table, preferably a collapsible one too. This will then be able to hold more data than just a regular table. MUI which is implements Google’s material design with pre-built component set, the package offers highly extensive theming support and robust community with documentation. There are other options as well, a lot of them offer similar options to MUI however MUI seems to be the most polished component library where it offers a lot of components that should meet the project’s aim and objectives. [16]

## 3.3 Neo4j to manage and store the data

Neo4j is a NoSQL, this is an example of an advanced and flexible database system. NoSQL is not only structured query language, it is used for when the normal relational database structures are too limited for the use case scenario, for example a knowledge graph is well represented when stored in Neo4J since it can store graph database. Neo4J uses Cypher as its query language which makes relationship querying intuitive compared to SQL by having simpler and more understandable querying syntax. When it comes to traversal, it is much quicker to hop from one node to another using Neo4J than the traditional SQL join clause. [8]

# Chapter 4: Design

## 4.1 Overview

The program will be designed to contain two types of geospatial visualisations of the knowledge graph data. These will be points on the map with edges connecting those points. The two types of visualisations will be in the form on ReactJS components, one is React-Leaf and the other is DeckGL.

Understanding the data provided and then modelling them is important. By modelling the data, this will enable the queries which are in the form of use cases to be ran quicker than just using a general data model.

After producing the data models, graph models will be put in place using Neo4J.

## 4.2 Requirements

### 4.2.1 The data provided

The data provided are in the form of csv format. There are 6 files which contain the following:

* SF\_List
  + It contains the spatial features (SF) such as buildings, malls, landmarks, etc. This includes both geographical coordinates and semantic classifications.
* ED\_SF\_OtherPoints\_Containment
  + Contains the relationship between electoral divisions (EDs) and spatial features like railway stations, buildings, or other local points of interest.
* ED\_Wales\_ProximityData
  + Contains directional adjacency graph between electoral divisions.
* SF\_OtherPoints\_ProximityData
  + Contains directional proximity relationships between spatial features like buildings, transport points, landmarks, or even businesses.
* UA\_ED\_Containment
  + Contains administrative hierarchy: which Electoral Divisions belong to which Unitary Authority (UA).
* UA\_Wales\_ProximityData
  + Contains the directional proximity relationships between Unitary Authorities.

Overall, the files collectively represent a multi-layer geospatial knowledge graph, compromising of spatial features, electoral divisions, unitary authorities. Along with relationships such as proximity and containment. The nodes are defined with details of coordinates and spatial features.

### 4.2.2 Use cases

The use cases will represent examples of what to expect the users to get out of the data.

1. Searching by place name, for example search for a place called Cardiff University, or called Cardiff.
2. Search places by place type, for example search for universities within Cardiff.

### 4.2.3 GUI

The UI would need a map to plot the points on the map then it would require the edges which connects those points to each other if more than one point is visible on the map.

The user would require the ability to do a search using the use cases listed above so at least one input box with a search button is required. This will then update the map with the results.

Having just a map would not be sufficient for the user to have the ability to view the data differently and directly. Directly, meaning the user can read all the information about the location or locations found, this will include subjects, longitude and latitude and more. This can be in a form of table with rows and columns. If one data point meaning one location is found, then it will show one record in that table and if more than one location is found then it will list them all.

The user will be able to switch the views from the two view map components that would be implemented, which are React Leaf and DeckGL.

There will be a search feature. It will have the ability to search by entering information in an input field then clicking the search button would be a way of sending the data to Neo4J. Different kind of searches would require the GUI to implement some sort of options such as dropdown menu, this will allow the user to choose what type of search would they like to perform then the necessary input fields will appear to prompt the user to enter the data then click search.

## 4.3 Modelling

### 4.3.1 Data modelling

The data provided include nearly all the correct data needed to build a graph model. One file where the data needs to be converted to longitude and latitude is the SF\_List file. The file contains X and Y labels which cannot be used in the ReactJS components such as react leaf so converting them would make it much easier to work with them plotting them on a map in ReactJS. Creating a python script to convert them and the outcome will be a new csv file with two new fields of longitude and latitude. It will use transformer from pyproj, the library is used for geographic data and converting coordinates. The data in the original csv file is in British national grid format with X and Y values, this is represented by the code “EPSG:27700” and conversion to longitude and latitude would be to the code “EPSG:4326”. [14]

The below figure shows a small sample of the converted data:

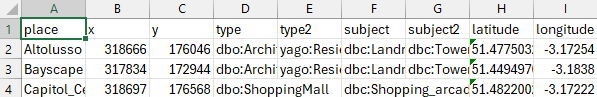


Figure 3 Converted x and y to latitude and longitude

### 4.3.2 Graph modelling

The graph models will be created to meet the use case scenarios. There can be one data model for all scenarios however this might slow down the queries since it would be searching in a much bigger data model. Creating tailored data models for each of the scenarios can lead to the queries taking less time to process.

The general data model that contains all the data includes all the csv files and connected edges and nodes. The connected edges can either be proximity, contains or direction. This is shown in the figure below.

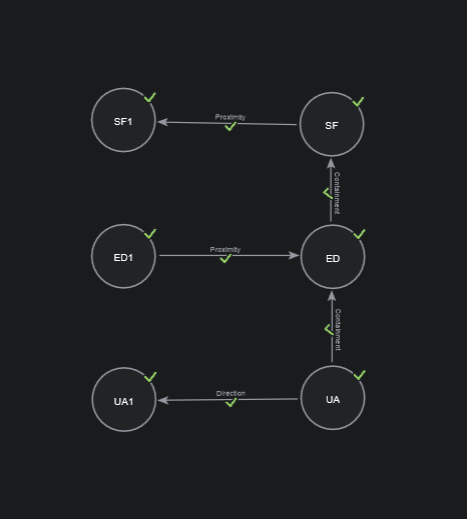


Figure 4 Neo4J Graph Model

The scenarios and what graph models would look like:

1. Regional information, for example: List all places in Cardiff.
   1. To use have the following data:
      1. SF list is needed but to get the overall picture with more detailed data, all data would be required.
         1. This will list all the places and points of interest in Cardiff, such as ED and shops.
2. To search by place, type, and relationship type. For example, what hospitals are in the north of Cardiff.
   1. The data required:
      1. All the data since it will check for hospitals in the north of Cardiff and it will use the directly connected nodes to show them too for a more in-depth output.
         1. This will list all the hospitals in the north of Cardiff.

# Chapter 5: Implementation

## 5.1 Programming Language

Choosing JavaScript as the programming language when creating a website will help expand the project quicker due to its simplicity of the programming language. ReactJS utilises the language with a folder and code structure that is intuitive to follow. The availability of third-party libraries which are called components in ReactJS is massive.

## 5.2 Version control

Using version control helps when it comes to detailed history of every change so it is possible to see what changes are applied, where, and why. Allows the developer to create different branches, meaning control of what features are done can be separated so if there is a mistake or something new, it is on its own separated from the original working code hence experimenting can be done safely. Automation and deployment can be added, for example when having a ReactJS web app project, this can be deployed to a server for hosting when changes are detected in a specific branch. This will help in deploying the project quickly.

Using Cardiff University GitLab to store and host the ReactJS project on a private repository.

## 5.3 Neo4J

Setup of Neo4J is done using Neo4J aura, creating an instance that is for free which suits the project since it won’t need scalable options like they have in offer in the paid tiers. [13]

The instance is running online so it is accessible from any computer around the world, if the credentials are used, figure below shows the instance running on the cloud.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5 Neo4J Aura Cloud Instance

### 5.3.1 Importing data

Importing data into the data sources section, this can be from another database or JSON files which is used in this project. The figure below shows a graph model with an empty data source.

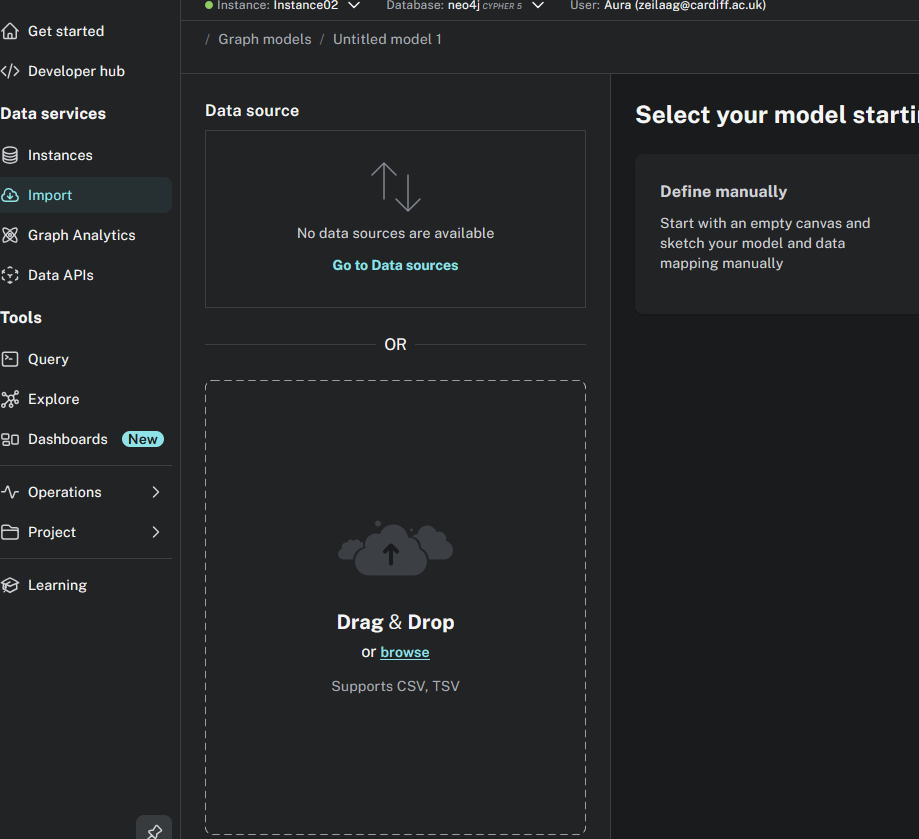


Figure 6 Neo4J Empty Graph Model

#### 5.3.1.1 Converting CSV into JSON

Converting the csv files into two JSON files, one will contain the list of nodes, and the other will contain the list of relationships (edges).

This will help when importing them into Neo4J and it will make sure that all nodes with relationships are imported by using cypher query.

Creating a Python script to convert the csv files into JSON files. First script is to create the nodes.json file. There is a csv file that contains all the places called SF\_List. This was already converted into longitude and latitude by another Python script since React Leaf component (the map library in ReactJS) cannot use the default x and y that it came with. The final csv file adds two columns of longitude and latitude. Code available in appendix A.

Since we have the places csv file now ready, converting it into nodes with the rest of the csv files since not all nodes exist as a place in the place csv file. The base of the landmarks\_converted.csv file is first loaded into Python then it checks if the node is not already added, this makes sure that no duplicate nodes are added, shown in the figure below.

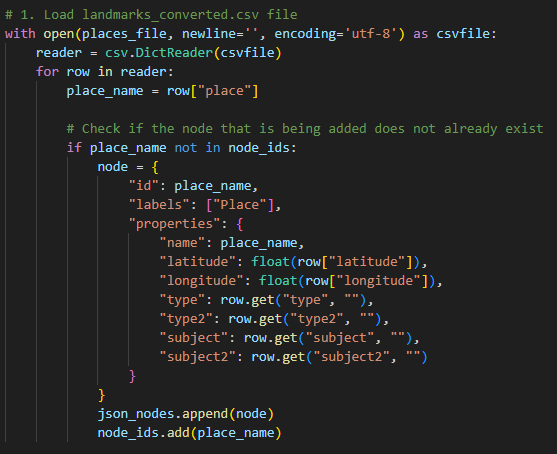


Figure 7 Nodes Python Script Converter

Once the places are loaded, it then adds the rest of the data which contain for example shops and roads. These will not contain any longitude, latitude, type, type2, subject, subject2 hence loading them with just name is enough. These are needed to be loaded as nodes since the relationship later needs them as they are listed in there and as the graph model that was shown in the design chapter shows the relationship between them.

467 nodes are created and added into a JSON file called nodes.json.

Loading the relationships into a JSON file from all but one csv file using a Python script. The csv files must contain the relationship, for example direction and contains. The places csv file will not be used here since it only contains list of places and no relationships. The relationship JSON file will contain the from, to and type where the property of relationship will be added. The Python script also checks if there are any invalid rows, for example if the row number is not count of 3 then it will ignore and not add that row since it does not define a relationship. When running the Python script, there are no invalid rows and 2631 relationships are created. In the figure below, it shows what the template will look like in the JSON relationship file. The rest of the code is available in appendix A.

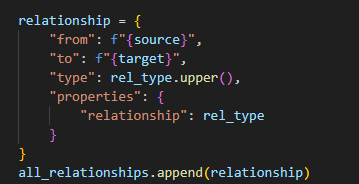


Figure 8 Relationship Python Script Converter

#### 5.3.1.2 Loading JSON files into Neo4J

Using cypher query to load the JSON file into the cloud database instance of Neo4J will require to load it into an online accessible link that represents the JSON file, using Pastebin allows the JSON file to be accessed via Neo4J cloud database instance.

First loading the JSON file and representing the output as value and then passing the value to the create node where the labels are loaded with the id and properties then node value is passed and returned as the output. The figure below shows the full cypher query used to import nodes.json file into Neo4J.



Figure 9 Load nodes.json into Neo4J cypher query

467 nodes are loaded, this represents the same value of nodes that was created when converting from csv file into json file using the Python script. The figure below shows all the nodes created, this is currently without any edges.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 10 Nodes in Neo4J without edges

Loading the relationships.json file into Neo4J cloud database instance required the same steps as loading the nodes.json file but with a different cypher query. The cypher query will load the json file from Pastebin and pass the value to then match the id with the values of “from” and “to”. Those are then passed as a, b, value. Then use of relationship will be created based on the “a”, which is “from” and value “type” which is the type of relationship, properties, and value “b” which is “to”, this is then returned to preview the results. 2631 relationships are imported and created into Neo4J, this value is the same value when creating the relationship.json file using the Python converter hence no relationships are missed. The figure below shows the cypher query.

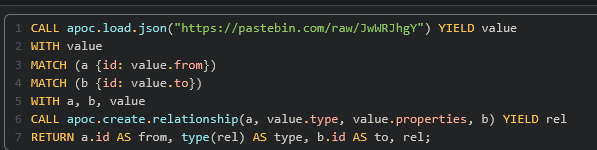


Figure 11 Load relationships.json into Neo4J cypher query

A small sample of the output in Neo4J after the relationship.json import and creation shown in the figure below.



Figure 12 Sample relationship import data into Neo4J

To show an overview of all the nodes and edges imported into Neo4J cloud database instance, running a cypher query to load all nodes and edges shown in the figure below.

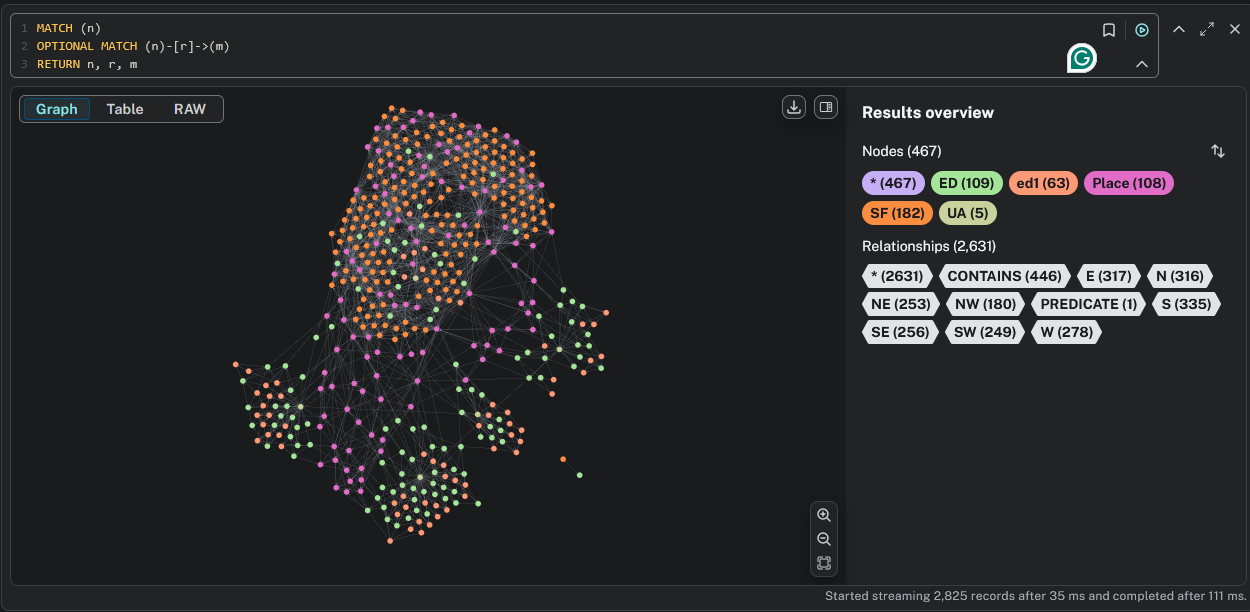


Figure 13 All nodes and edges in Neo4J

Overall, the nodes and edges are connected however there are two nodes where they are not connected. The figure below shows a zoomed in view of the nodes by themselves.

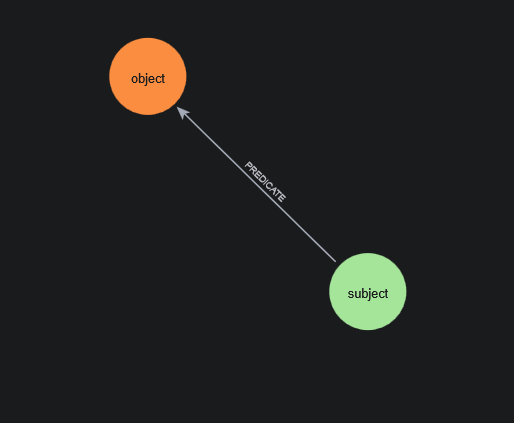


Figure 14 Nodes by themselves in Neo4J

This was imported from the csv file into JSON and is located in ED\_SF\_OtherPoints\_Containment.csv file. Opening the file in excel shows the incorrect data.



Figure 15 Incorrect data in ED\_SF\_OtherPoint\_Containment.csv

By removing the row, then running all the Python converting scripts and importing all nodes and relationships, the node count is now reduced by two and the relationships are reduced by one. And there is no longer two nodes and an edge by themselves.

## 5.4 Cypher query

To meet the use cases listed in the design chapter, the queries need to be designed to give an answer to the use cases.

The use cases:

1. Regional information, for example: list all places in Cardiff.
   1. The cypher query used:
      1. Looking at the data, it shows that Cardiff is in SF\_List which is under the header place and also it can be under the subject title where it can mention Cardiff in the text for example Cardiff.
      2. A screen shot of a computer code

         AI-generated content may be incorrect.

Figure 16 Use case 1 cypher query

* + 1. The cypher in the figure above finds all nodes in the subject field contain the text “Cardiff” and it makes sure to return all the results that include a longitude and a latitude, this is done to be able to plot the results on a map in ReactJS later. Then by using the output of the nodes of “Cardiff”, it is used to check the relationship to other nodes and include those, this relationship is directly connected and nothing more so the end result will include all nodes that have “Cardiff” as a subject and all related nodes.
  1. The outcome:
     1. A screenshot of a computer

        AI-generated content may be incorrect.

Figure 17 Use case 1 cypher query output

* + 1. The nodes contain all the Cardiff text in them will appear first then all the direct relation nodes will appear, hence the number of nodes appearing of 263 which is about half of the total nodes, this data is in Wales, mostly south Wales so it would be expected to have a lot of nodes appearing in this type of search.

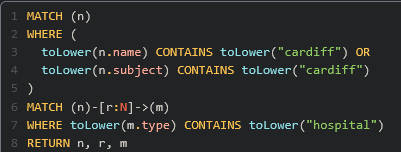
1. To search by place, type, and relationship type. For example, what hospitals are in the north of Cardiff.
   1. The cypher query used:
      1. The data that contains the place names is in the name and subject fields, then the relationship is the North or N as saved in the data. For the type, the hospital is an example in the type field.
      2. 

Figure 18 Use case 2 cypher query

* + 1. The cypher query returns all the nodes and edges which are related to places containing the word “cardiff” with the relationship of “N” (north), and place type containing the word “hospital”. The n is for the node of place, r is for the relationship, and m is for the place type.
  1. The outcome:
     1. A screenshot of a computer

        AI-generated content may be incorrect.

Figure 19 Use case 2 cypher query output

* + 1. The query output shows hospitals in north of Cardiff that are connected to the north of various locations such as Cathays railway station. In total there are 3 hospitals that are in the north of Cardiff.

## 5.5 Overview ReactJS

Creating the ReactJS app requires NodeJS to be installed. This can be done by downloading then installing the NodeJS installer from their official website. This can be installed on Windows, macOS, and Linux. [10]

Creating the react app by running the following command in the command line prompt: “npm create-react-app my-app”. This will download all the necessary files to get started. It will have an organised structure that we can follow later when adding more files to the website. [11]

The design library with UI components that is going to be used is called Material UI. This will represent the input boxes, search buttons, tables, and any other UI components except for the maps. Choosing this library since it is free to use and offers wide variety of UI elements such as sliders, tables, dialogs, and more. The UI components are highly responsive hence having a mobile friendly UI would become much easier. Customisations of the UI elements is also possible by adjusting size, colour, and actions such as on hover colour and effects. Adding the MUI component requires some packages to be installed, running the following command “npm install @mui/material @emotion/react @emotion/styled” installs all the necessary packages. [15]

The UI components used to meet the UI design requirements will be buttons, input boxes, and tables. The button would be used for to click search. The input boxes would be required to contain value, for example a place. If a value is there, then it will send it with the cypher query to Neo4J to process and output an answer. Once the answer is received from Neo4J in ReactJS, the data will be processed to be added to the map of both React Leaf and DeckGL, and table component which is from MUI.

The table component would be used to list all the output in a table format since not all data can be plotted on the map because not all data contain longitude and latitude. Within the table, there is a collapsible option, this will show all the nodes that have an edge to that node that was clicked on. In the rows, there are the following headings: Name, type, latitude, longitude, number of edges, shown in the figure below:

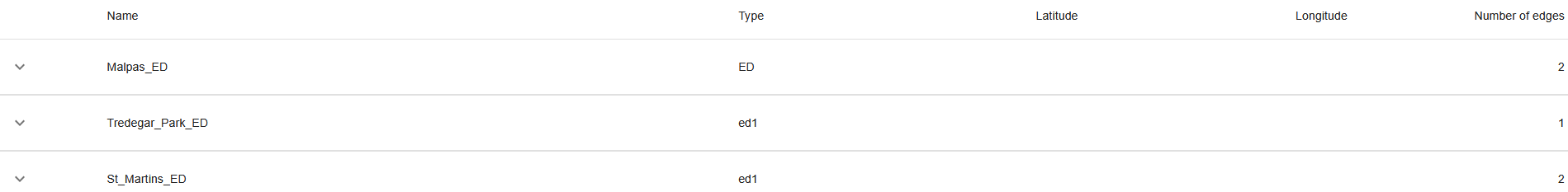


Figure 20 Table with examples of nodes

Example of places in the table:

A screenshot of a phone

AI-generated content may be incorrect.

Figure 21 Table with example of place nodes

Example of a place with its edges, this is the collapsible option:

A white page with black lines

AI-generated content may be incorrect.

Figure 22 Table with example of expanded node that lists its edges

The input boxes and search button to be dynamically done so in the future, more use cases can be added. Using a dropdown menu option for search type will help in allowing more search options to be added. In this case, all the searches in the use cases to be implemented in this project is done under the dropdown menu.

### 5.5.1 Use cases: GUI implementation

For the search by place use case, which is the first one, the UI is shown in the figure below where the “Search by Place” option is chosen, and a place can be entered then the search button will be clickable.



Figure 23 Search by Place GUI

This will allow the cypher query to contain all the data that it needs to then be sent to Neo4J to get a result back to ReactJS so then it can be used to display in the graph as listed above and in the map components which will be shown below in section 5.5.3 and 5.5.4.

The second use case which is to search by place, place type and relationship is shown in the figure below.

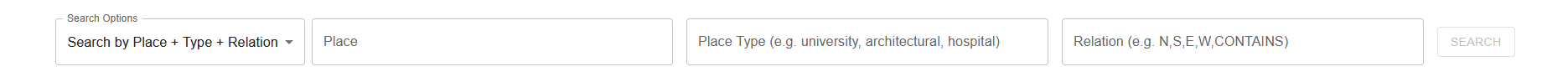


Figure 24 Search by Place, Place Type, and Relationship GUI

This will also do the same functionality when the search button is clickable, this is when the data values are entered which will then enable the search button.

### 5.5.2 Neo4J Driver

Implementing Neo4J Driver required the installation of driver in ReactJS. Running the following command on the ReactJS project: “npm install neo4j-driver” to install the Neo4J Driver. This will allow the frontend to run queries to the database. The database has the geospatial knowledge graphs. [9]

Since the credentials are needed to connect to the Neo4J database, storing the credentials in a secure manner would be necessary. Separating the credentials in a JSON file then by adding the file to the git ignore list, the credentials would never be passed to the git repositories.

Producing a file in components under the folder services for Neo4J service will be useful so that it can be used in the react project. This file has the credentials connection details and the query to send and receive the results from Neo4J. This is shown in the figure below. [9]

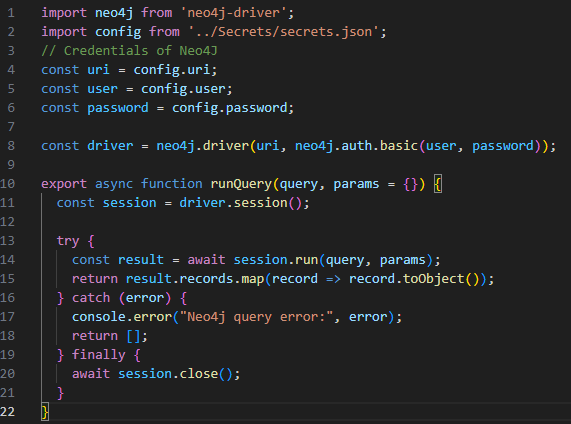


Figure 25 Neo4J driver implementation in ReactJS

The above figure shows the credentials being used to get the driver a session which can then be used to run the query with the parameters to get a result, this might return an error if for example the connection cannot be established so it is shown in the console.

Finally, the session is then closed.

The same cypher query that was made in Neo4J would be implemented in ReactJS, this is shown in the figure below.

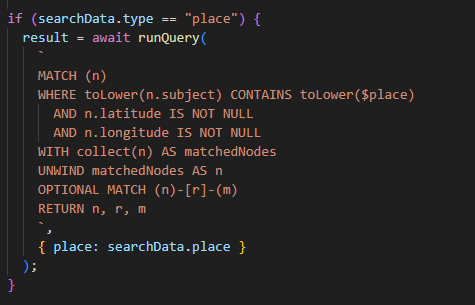


Figure 26 Cypher query in ReactJS

The above figure contains the cypher query for the first use case which is searching by place. This $ symbol represents a parameter of place that will be given by the user then it would be sent to the Neo4J service function which will send all the cypher with the parameter to Neo4J to process and return a result of nodes(n), edges(r), and end nodes(m).

### 5.5.3 React-Leaf

Implementing React-Leaf by first installing the components required using “npm install react@rc react-dom@rc leaflet” and “npm install react-leaflet” command. [12]

Making a new file in the ReactJS project called ReactLeaf.js, to contain the map, nodes, edges, and sending search cypher queries to the neo4j service function.

Using the MapContainer component from ReactLeaf to show the map with the first node’s coordinates specified and specifying the zoom level which 9 seem to be the ideal zoom level, this can be adjusted by the end user by clicking on the + or – buttons to adjust the zoom level.

The Marker component from ReactLeaf adds the points on the map with the Popup component to add the labels when clicking on the points. Then for the edges, there is a Polyline component from ReactLeaf which takes from and to nodes and has an option to adjust the colour in the code, currently set to the colour blue.

The data of nodes and edges are the results of the cypher query from Neo4J. Then the nodes are loaded using the .map functionality in javascript which allows to iterate though the nodes array. In the iteration, there is the repeated Market component that is called on every node that is added. The same logic is done for the edges but with the iteration on the edges array and the repetition of the Polyline component. The end result will look like this figure below.

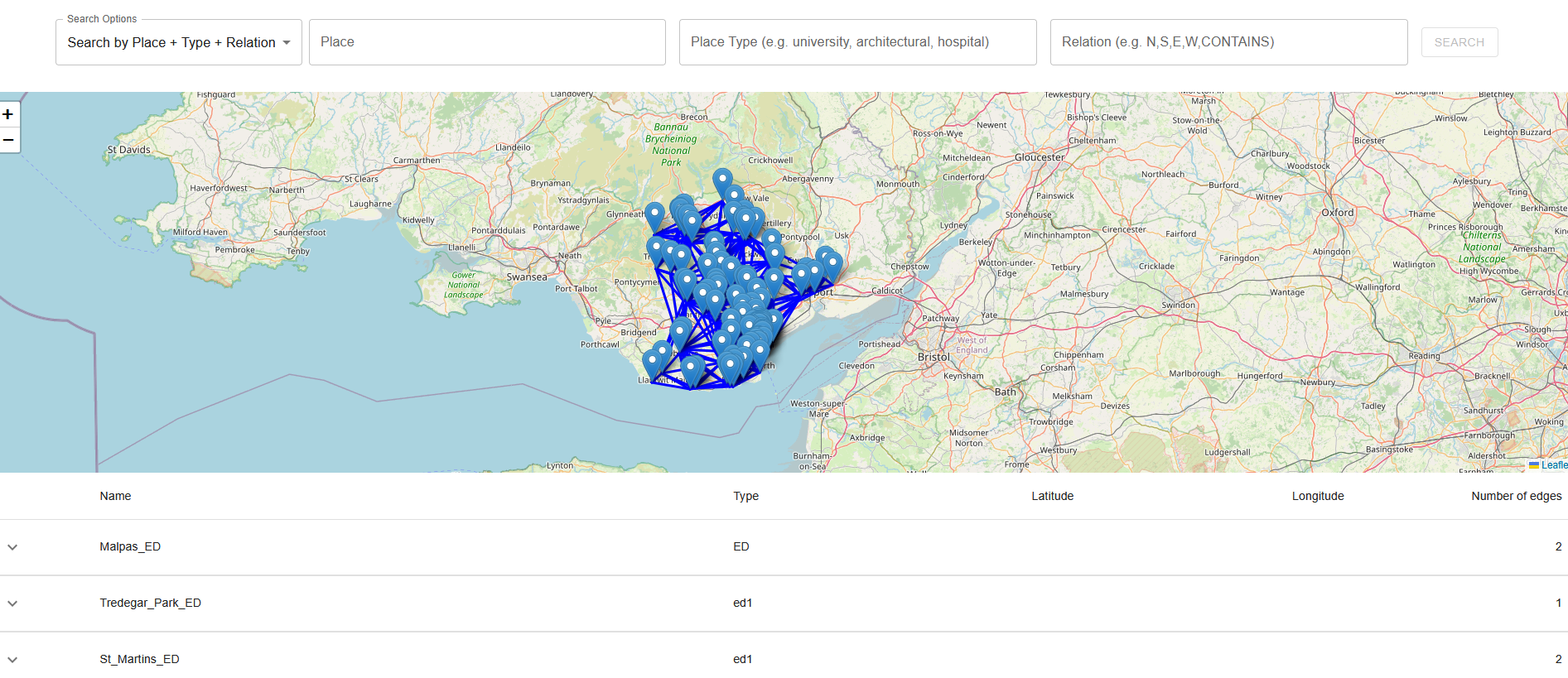


Figure 27 ReactLeaf example map of all data

### 5.5.4 DeckGL

# Chapter 6: Evaluation & Conclusion

# Chapter 7: Future Work

# Chapter 8: Reflection

# References

[1] Ji, S., Pan, S., Cambria, E., Marttinen, P. and Yu, P.S. (2022). A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. IEEE Transactions on Neural Networks and Learning Systems, [online] 33(2), pp.494–514. doi:https://doi.org/10.1109/TNNLS.2021.3070843.

[2] support.google.com. (n.d.). About knowledge panels - Knowledge Panel Help. [online] Available at: https://support.google.com/knowledgepanel/answer/9163198?hl=en [Accessed 12 Jul. 2025].

[3] kepler.gl. (2019). Large-scale WebGL-powered Geospatial Data Visualization Tool. [online] Available at: https://kepler.gl/ [Accessed 12 Jul. 2025].

[4] Guo, D. and Onstein, E. (2020). State-of-the-Art Geospatial Information Processing in NoSQL Databases. ISPRS International Journal of Geo-Information, 9(5), p.331. doi:https://doi.org/10.3390/ijgi9050331.

[5] Rajiv Tulsyan, Shukla, P., Singh, T. and Kumar, A. (2024). The Impact of JavaScript Frameworks on Website Performance and User Experience. [online] pp.299–305. doi:https://doi.org/10.1109/icbdml60909.2024.10697529.

[6] Li, W., Wang, S., Chen, X., Tian, Y., Gu, Z., Lopez-Carr, A., Schroeder, A., Currier, K., Schildhauer, M. and Zhu, R. (2023). GeoGraphVis: A Knowledge Graph and Geovisualization Empowered Cyberinfrastructure to Support Disaster Response and Humanitarian Aid. ISPRS International Journal of Geo-Information, [online] 12(3), p.112. doi:https://doi.org/10.3390/ijgi12030112.

[7] react-leaflet.js.org. (n.d.). Introduction | React Leaflet. [online] Available at: https://react-leaflet.js.org/docs/start-introduction/ [Accessed 14 Jul. 2025].

[8] Pourabbas, E. (2025). Geographical Information Systems. [online] Google Books. Available at: https://books.google.co.uk/books?hl=en&lr=&id=dcuSAwAAQBAJ&oi=fnd&pg=PA73&dq=neo4j+geospatial&ots=G8ASJx9DE3&sig=a1MSTyvYGB69fC7Liq8WX97iZX8&redir\_esc=y#v=onepage&q&f=true [Accessed 14 Jul. 2025].

[9] Neo4j Graph Data Platform. (n.d.). Build applications with Neo4j and JavaScript - Neo4j JavaScript Driver Manual. [online] Available at: https://neo4j.com/docs/javascript-manual/current/ [Accessed 25 Jul. 2025].

[10] Node.js. (n.d.). Download. [online] Available at: https://nodejs.org/en/download [Accessed 25 Jul. 2025].

[11] Create-react-app.dev. (2019). Create React App · Set up a modern web app by running one command. [online] Available at: https://create-react-app.dev/docs/getting-started/ [Accessed 25 Jul. 2025].

[12] react-leaflet.js.org. (n.d.). Introduction | React Leaflet. [online] Available at: https://react-leaflet.js.org/docs/start-introduction/ [Accessed 25 Jul. 2025].

[13] Graph Database & Analytics. (2024). Neo4j Pricing. [online] Available at: https://neo4j.com/pricing/#graph-database [Accessed 26 Jul. 2025].

[14] Github.io. (2019). Transformer - pyproj 3.7.1 documentation. [online] Available at: https://pyproj4.github.io/pyproj/stable/api/transformer.html [Accessed 29 Jul. 2025].

[15] mui.com. (n.d.). Installation - Material UI. [online] Available at: https://mui.com/material-ui/getting-started/installation/ [Accessed 1 Aug. 2025].

[16] Giri, N. (2024). Best 7 React UI Libraries for 2024. [online] Nishangiri.dev. Available at: https://nishangiri.dev/blog/best-react-ui-library [Accessed 4 Aug. 2025].

# List of Figures

[Figure 1 Sample data loaded in Kepler.gl 5](#_Toc205288350)

[Figure 2 Layer example in Kepler.gl 6](#_Toc205288351)

[Figure 3 Converted x and y to latitude and longitude 10](#_Toc205288352)

[Figure 4 Neo4J Graph Model 11](#_Toc205288353)

[Figure 5 Neo4J Aura Cloud Instance 12](#_Toc205288354)

[Figure 6 Neo4J Empty Graph Model 13](#_Toc205288355)

[Figure 7 Nodes Python Script Converter 14](#_Toc205288356)

[Figure 8 Relationship Python Script Converter 15](#_Toc205288357)

[Figure 9 Load nodes.json into Neo4J cypher query 15](#_Toc205288358)

[Figure 10 Nodes in Neo4J without edges 15](#_Toc205288359)

[Figure 11 Load relationships.json into Neo4J cypher query 16](#_Toc205288360)

[Figure 12 Sample relationship import data into Neo4J 16](#_Toc205288361)

[Figure 13 All nodes and edges in Neo4J 17](#_Toc205288362)

[Figure 14 Nodes by themselves in Neo4J 17](#_Toc205288363)

[Figure 15 Incorrect data in ED\_SF\_OtherPoint\_Containment.csv 18](#_Toc205288364)

[Figure 17 Use case 1 cypher query output 19](#_Toc205288365)

[Figure 18 Table with examples of nodes 20](#_Toc205288366)

[Figure 19 Table with example of place nodes 20](#_Toc205288367)

[Figure 20 Table with example of expanded node that lists its edges 20](#_Toc205288368)

# List of Tables

# Appendices

Appendix A – GitLab Project, contains the converters, ReactJS webapp, csv, and json files.

<https://git.cardiff.ac.uk/student-projects/msc-projects/zeilaa-2025>