DEVELOPMENT OF A WEB APPLICATION FOR VISUALISING GEOSPATIAL KNOWLEDGE GRAPHS

Author: Georgio Zeilaa - C23101178

School of Computer Science and Informatics,
Cardiff University

Supervisor: Alia l Abdelmoty
MSc Advanced Computer Science

September 2025

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Alia l Abdelmoty, for their extremely helpful guidance and advice throughout this project. I am also grateful to my friends and family for their advice and encouragement.

Abstract

To create a website that visualises the geospatial knowledge graph by utilising various features including maps, tables, and detailed panels. There are different approaches to this problem, choosing the most appropriate solutions would be done by researching existing products and choosing wisely the most suitable features, keeping in mind that future expandability would be needed. Designing the website project to implement features with scalability is a factor hence a plan must be in place to satisfy scalability. Exploring the features implemented by testing them with comparisons made with the original geospatial knowledge graph data would mean a better result for the project where solutions were implemented and explored of how well they are implemented. At the end, concluding how well the website runs with the features added and more featured that can be added.

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

Knowledge graphs are a representation of organised information so that concepts such as people, places, or objects are represented as nodes and the relationships between them are represented as edges. Focusing on geospatial knowledge graphs which represent places and connections between them.

The data represented would need to be in a form of database that is able to extract data whenever a result is requested. Researching existing databases that work best with geospatial knowledge graphs is one key point to achieve in order to produce a working database.

Producing websites can be done in a lot of various ways hence looking into what can be the best choice for this project which makes use of the geospatial knowledge graph database plus the graphical user elements which needs to be implemented as well. The graphical user elements would represent the geospatial data visually. This can be in various forms of visualisation.

Looking at existing products that represent a geospatial knowledge graph in a website, this would help the project in implementing visualisation features on the website. Adding more features would be explored when it comes to meeting the aim by following the objectives.

Checking the solutions implemented to make sure they function as intended and they are correctly outputting results that are accurate. This will be done by comparing results on the website with the chosen database. Making sure that there is a base line of accurate results which is in the database. This helps in making a fair comparison.

Future improvements are possible since this project focuses on the implementation of the website with scalability in mind and visualisation elements. Scalability helps ensure that project code can be easily expanded with new features or enhancements, while minimizing the need to modify existing code.

Chapter 2: Literature Review

This chapter reviews existing solutions and compares them with potential alternatives that could be implemented in this project. Particular attention is given to evaluating the importance of using a specific database solution in relation to other available options. By identifying and analysing these approaches, the review aims to provide a clearer understanding of how they can be implemented and the ways in which they add value to the end user. The discussion is organised thematically, beginning with an overview understanding of knowledge graphs, followed by a comparison of their strengths and limitations of database solutions, and concluding with an assessment of their applicability to the project.

2.1 Introduction to knowledge graphs

Knowledge graphs represent 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. (Ji et al., 2022) 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. (support.google.com, n.d.)

2.2 Existing web applications for data visualisation using knowledge graphs

2.2.1 Kepler.gl

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. (kepler.gl, 2019)

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. (kepler.gl, 2019)

Using ReactJS is a consideration that this project can utilise since it is a web framework solution that is used in such a big open source project which has been maintained through many years, looking at the github repository, it is still being updated regularly and the community are contributing to the project as well. (keplergl, 2018)

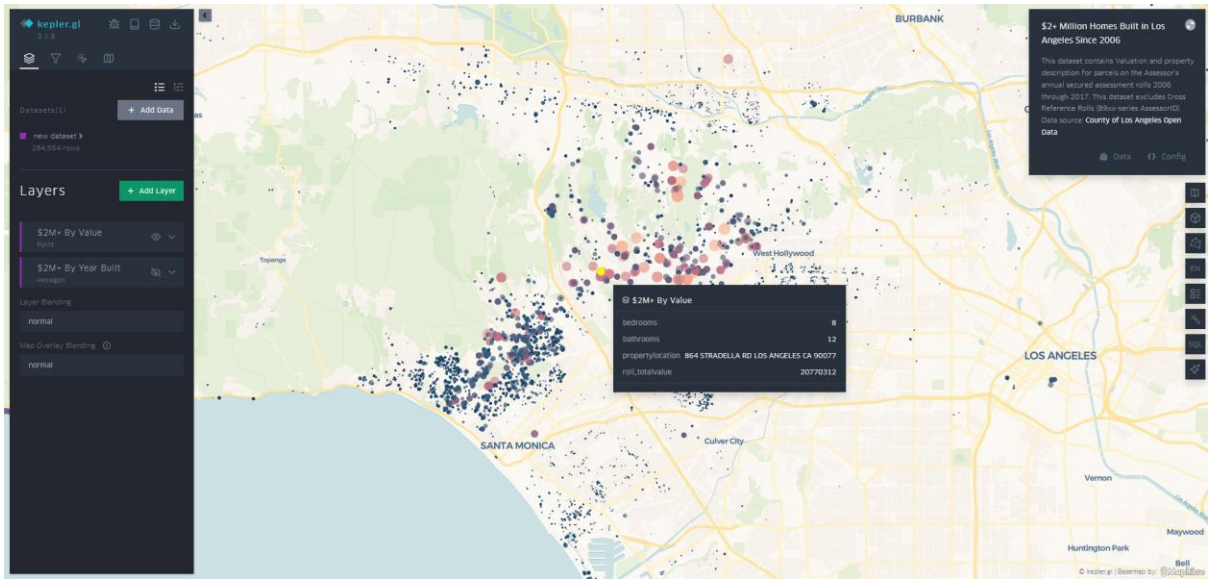


Figure 1 Sample data loaded in Kepler.gl

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.

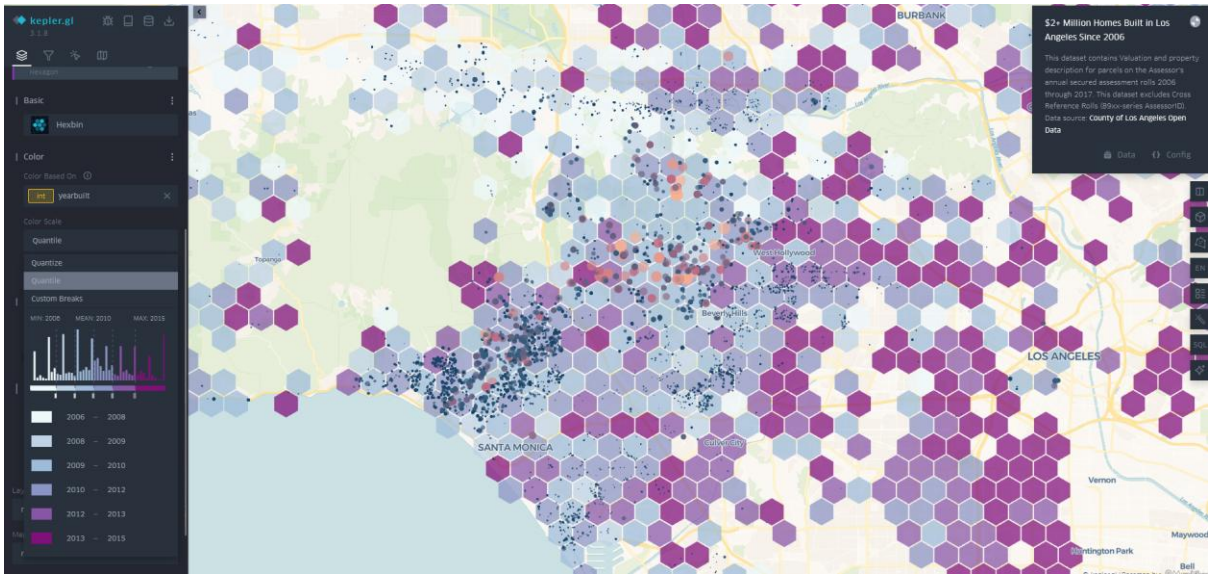


Figure 2 Layer example in Kepler.gl

2.2.2 QGIS

QGIS is an open-source platform that visually shows geospatial data, editing the data, and analysis. It is highly versatile for many applications such as bushfire hazard mapping and monitoring tigers using camera. (QGIS, 2025)

The use of QGIS grew a lot in the past years, specifically between 2005 and 2020, the number of publications utilising QGIS went up by 40.3% annually. Rosas-Chavoya et al., 2022)

Its versatility also is possible by supporting plugins and tools. An example of a tool is FREEWAT, a plugin used for groundwater and surface water management. This project was financed hence the extensive development that went into it is a lot. (Rosas-Chavoya et al., 2022)

QGIS has a desktop application which allows the loading of map data. Focusing on the visual aspects, QGIS has an option for filtering by layers. The layers can be enabled and disabled which displays them on the map. Producing visually unique maps is also possible using the desktop application, for example here's a vintage map of Rotterdam. (Qgis.org, 2025) It is shown in the figure below.

Figure 3 QGIS Vintage Map Of Rotterdam

The map shown above was created by the use of a plugin called TopoTijdReis. This plugin contains all the historic maps from 1815-2025. Using another plugin called MapTiler to adjust the colours of features with the samples. This is not easily possible without the use of plugins which QGIS have a wide variety of plugins to choose from. (Qgis.org, 2025)

Overall, the two explored options do have a filter feature in common which makes it easier for the user to understand the data shown on the map. They are both open source, it helps in having a community to support the development of either project.

2.3 NoSQL Databases

The geospatial knowledge graph information must be stored in a location that is accessible. This can be in a form of database or even a direct text file access. There are ways to improve access to the data by utilising features in a database which is produced to handle knowledge graphs, especially geospatial data.

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. (Guo and Onstein, 2020)
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. (Guo and Onstein, 2020)
Using Cypher queries rather than SQL queries would make this project easier to implement and maintain, especially when future developers work on it.

Neo4J and MongoDB both seem to be good options however Neo4J does use cypher query which would make development of queries quicker when implementing search features which is to be investigated.

Chapter 3: Background

Choosing a web framework which will be the main part of the project where the website will be developed in. This will be done by comparing other solutions to existing solutions on what frameworks they use. Researching on what could be the best solution for implementing the GUI by utilising the chosen web framework would be beneficial in terms of versatility which would enable expandability in the future.

3.1 Choosing a web framework

Implementing a website can be done in a lot of ways. A simple approach but not used much these days would be to implement a html and CSS website. This approach is simple but lacks scalability and does not easily support third party libraries. On the other hand, a JavaScript based web framework would allow the use of JavaScript code which is simple yet powerful with its data manipulation possibilities and third-party support.

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. Kepler.gl which is mentioned in chapter 2 is based on ReactJS. This shows how quickly the kepler.gl solutions works and how many contributors are contributing to the project. 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 faster than ReactJS according to Redware study. (Rajiv Tulsyan et al., 2024)

Overall, the compromise in potential speed loss in ReactJS versus the flexibility of the community support and reliability which is a trade-off that might be worth making when choosing ReactJS over Vue. (Rajiv Tulsyan et al., 2024)

3.2 ReactJS

3.2.1 Examples of map components

Visualizing geospatial data is incomplete without a map therefore considering a few map components as options would greatly benefit the project.

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. (Li et al., 2023)

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 quite 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. (react-leaflet.js.org, n.d.)

The implementation of react-leaf is simple; by installing the package (library) by using the node package manager (npm), this allows the installation of any library. The features which can be used for the project are the points on the map, the edges by adding lines on the map and the clickable event on the points. This will allow the project to implement other components that can perform an action when a node on the map is selected. (react-leaflet.js.org, n.d.)

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. (Giri, 2024)

3.3 Neo4j to manage and store the data

The geospatial data of nodes and edges need to be stored in a database. There are specific databases that handle knowledge graphs better, enabling features such as finding the closest node to a specific one from a specified edge.

SQL query is normally used across a broad range of database systems. Neo4j is a NoSQL, this is an example of an advanced and flexible database system. NoSQL is not only structured query language but also 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. This is done 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. (Google Books, 2025)

The project would benefit greatly from using a NoSQL database such as Neo4J. This would enable the geospatial knowledge graph to be stored and accessed efficiently by

leveraging Neo4J's capabilities, which are specifically designed for working with and querying knowledge graphs.

Chapter 4: Design

4.1 Introduction

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 ReactLeaf, 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, a graph model will be put in place using Neo4J. Then to produce a graphical user interface on the ReactJS webapp that meets the requirements.

4.2 Requirements

Understanding the data provided which can then enable to implementation of the use cases. This helps in implementing a solution that is focused on what is available without going out of scope.

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 search Use Cases required to be implemented:

1. **Regional information, for example: List all places in Cardiff.**
   a. 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.**
   a. Checking 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. This will list all the hospitals in the north of Cardiff.

3. **Find all types in a place. For example, find all hotels in Cardiff**.
   a. This will list all places in Cardiff with the hotel nodes connected to it.

The use cases would be considered during the implementation phase; this would require the GUI to be able to facilitate the use cases. By having three different types of searches, the need of implementing them on the GUI is a important. Allowing the user to search using three options seamlessly.

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.
The first design plan, representing the simple base structure, is shown in the figure below.

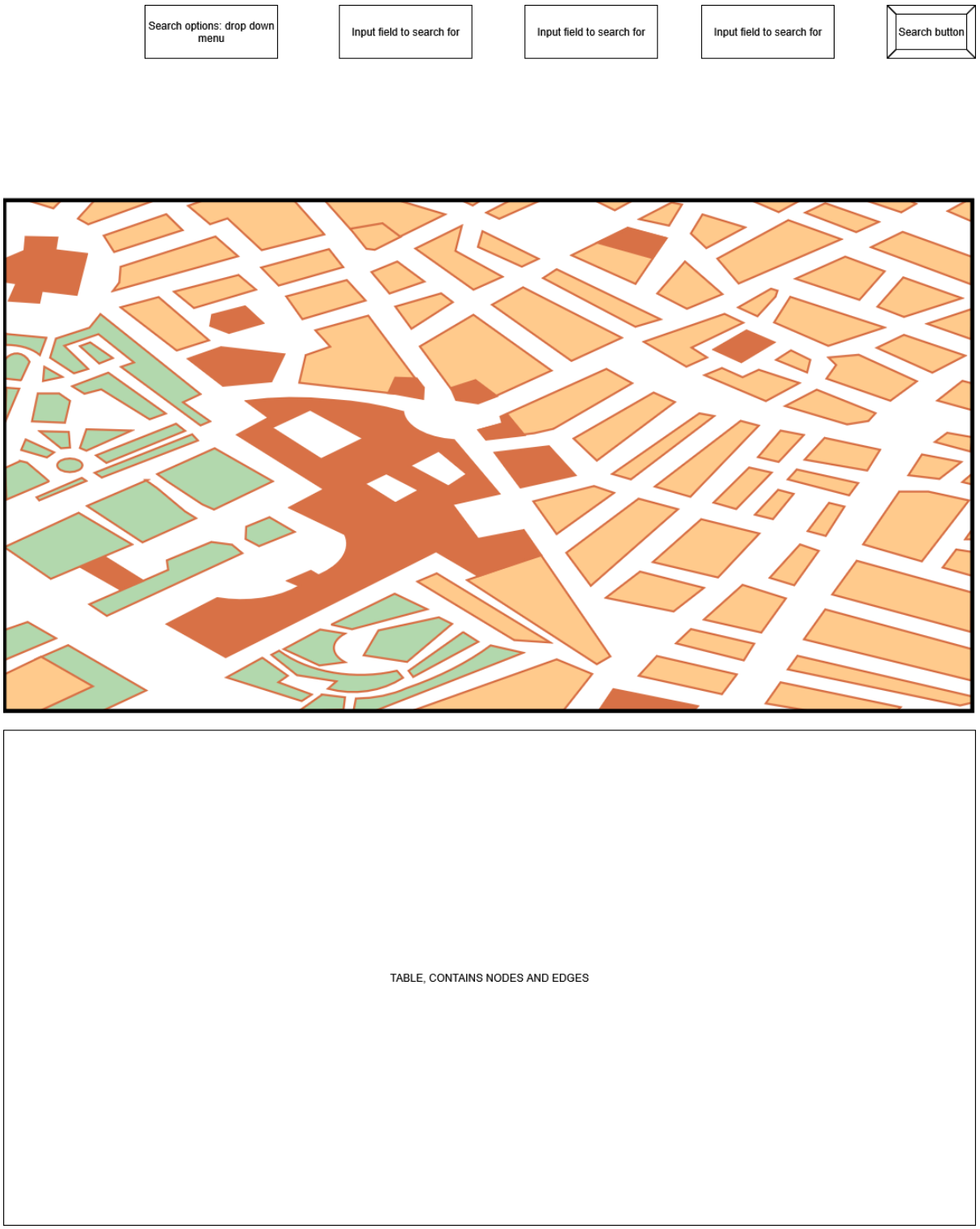


Figure 4 First Demo Plan Of Website

The above figure shows the design with a map, table, and search options. The below figure is an improvement where more features were added which was done due to the feedback from the supervisor of the project. The supervisor was a user that was able to provide constructive feedback which meant improvements could be made which are tailored to the users' requests.

The design which includes the buttons such as search, input boxes for the search, maps, table, and a detailed panel is shown in the figure below.

Figure 5 Design Demo Of Website

Other possible design was considered at first, this is shown in the figure above the "Design Demo Of Website" figure, the design included only a map with the points on the map being clickable to show what the place is called. This was good for a start however not all the data contain longitude and latitude hence the use of other forms of

visualisation components would be required. This would then show all the missing data from the map.

Adding a detailed panel meant more information would be shown. This enabled the use of clicking on the map which then sent the data to the detailed panel to show specific information on that selected node.
After adding a lot of features, a reset button was added since the user was unable to reset the website's data if they wanted to start from scratch.

The overall design of the website and its components was to consider the intractability between the components where they communicate to produce different types of information which can then be used by the end user for various requests.

4.3 Modelling

Insights into the data were gathered in section 4.2 of the requirements, which helped guide the data modelling process. Additionally, having a preliminary understanding of the GUI layout provided a clearer view of the data needed for each section of the website, ensuring that the data modelling outcomes are more precise and aligned with achieving an optimal implementation.

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, by allowing to plot the coordinates on a map in ReactJS.
Creating a python script to convert the X and Y values which the outcome will be a new csv file with two new fields of longitude and latitude. The python script will use a transformer from pyproj library. 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". (Github.io, 2019)

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

	A	B	C	D	E	F	G	H	I
1	place	x	y	type	type2	subject	subject2	latitude	longitude
2	Altolusso	318666	176046	dbo:Archi	yago:Resi	dbc:Landr	dbc:Towe	51.477503	-3.17254
3	Bayscape	317834	172944	dbo:Archi	yago:Resi	dbc:Landr	dbc:Towe	51.449497	-3.1838
4	Capitol_C	318697	176568	dbo:ShoppingMall		dbc:Shopping_arcac		51.482200	-3.17222

Figure 6 Converted x and y to latitude and longitude

By having the latitude and longitude, allowing the process to be seamless when using the data to be plotted on the map in ReactJS.

4.3.2 Graph models and Use Cases

The graph models will be created to meet the use case scenarios. There can be one data model for all scenarios however this will be custom made data model so each of the

scenarios can lead to the queries fully functioning as expected with a correct outcome that considers all the data.

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.

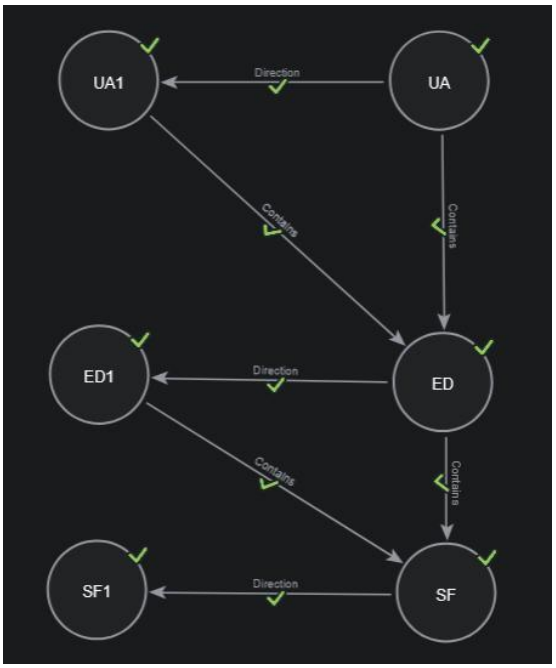


Figure 7 Neo4J Graph Model

The data model that is represented in a graph model contains all the data from the csv files. By having a good graph model, this means it represents all the data correctly with an effective knowledge graph that is shown in an organised and meaningful manner.

Chapter 5: Implementation

5.1 Overview

The solution will be implemented using an appropriate programming language, specifically ReactJS, to develop the user interface. Version control will be used to track changes and maintain an organised development process. Additionally, Neo4j will be integrated with ReactJS components, including the GUI and map features, to ensure seamless interaction between the data and visual elements.

5.1.1 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. By having a massive choice of components (libraries) which will enhance the development time by having prebuilt components.

5.1.2 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.

5.2 Neo4J

The database of choice to implement is Neo4J, this is where the data will be stored and processed. Since the setup would be to have Neo4J as the backend where the website would directly connect to Neo4J so a good database that has low processing times would be beneficial.

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. (Graph Database & Analytics, 2024)

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.

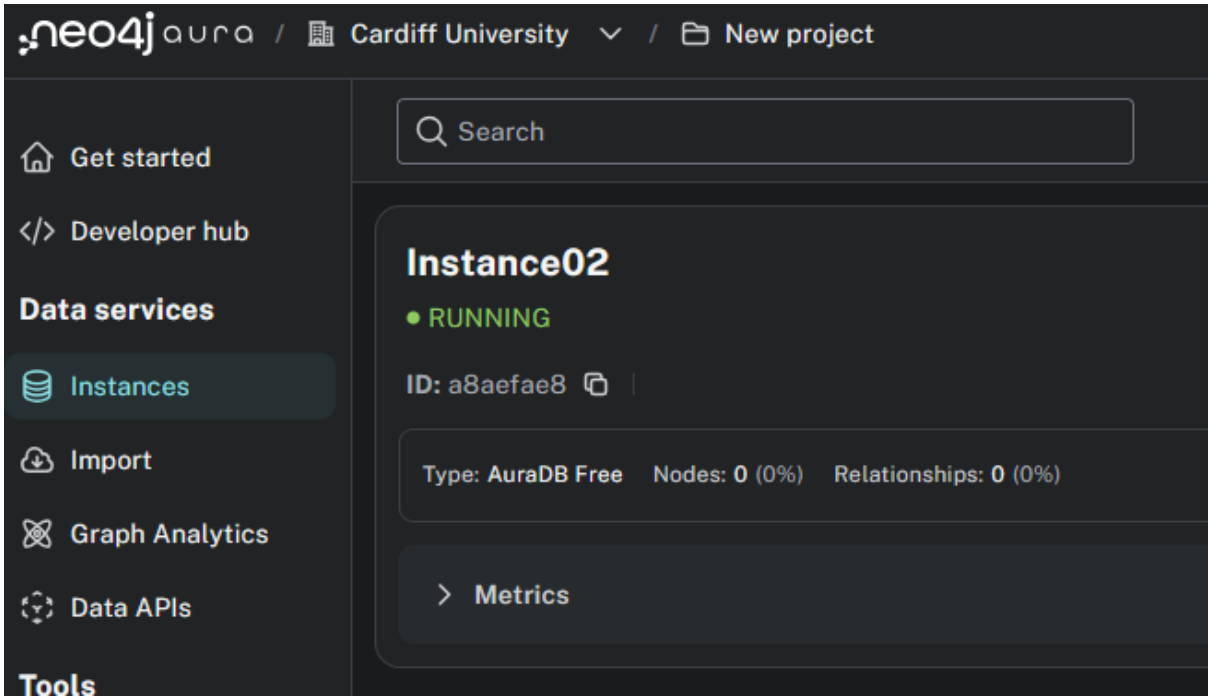


Figure 8 Neo4J Aura Cloud Instance

5.2.1 Importing data

Neo4J instance by default does not have any 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.

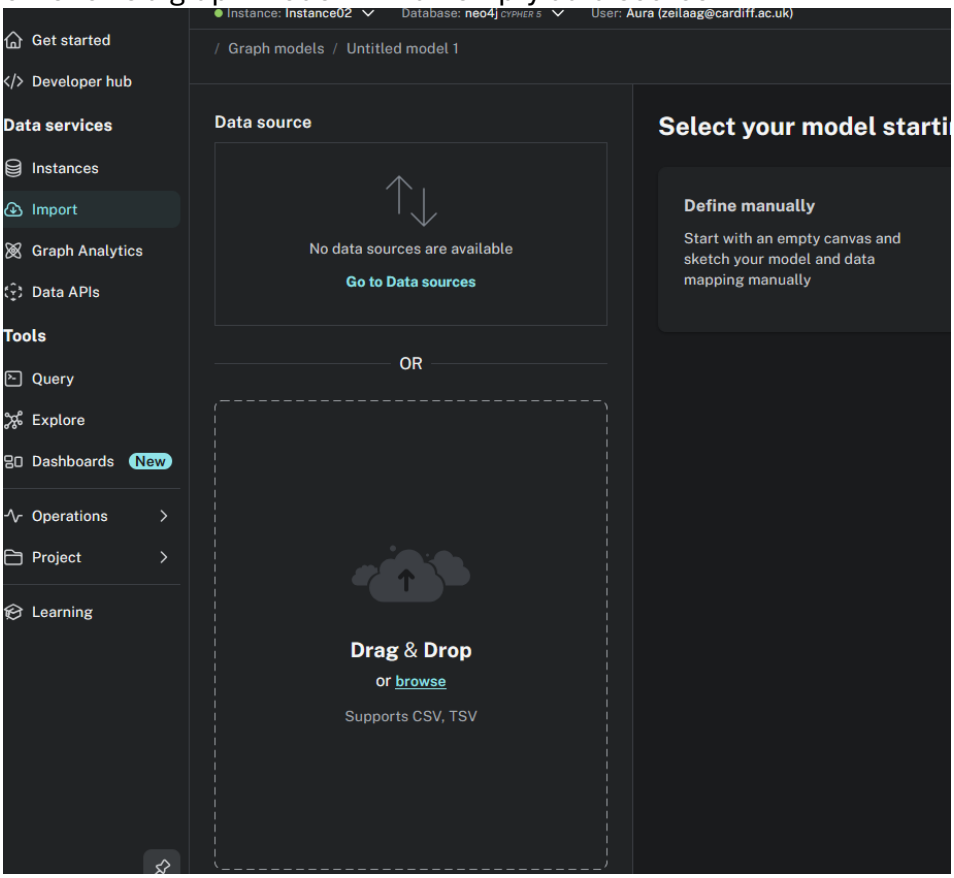


Figure 9 Neo4J Empty Graph Model

5.2.1.1 Converting CSV into JSON

Having csv files is not enough to be properly imported into Neo4J. The csv files are limited when it comes to representing a graph structure, this comes with the limitations of not supporting nested data. Nested data represents the relationships which are part of the geospatial knowledge graph. This will help when importing them into Neo4J and it will make sure that all nodes with relationships are imported by using cypher query.

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).
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 the places csv file is 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.

```
# 1. Load landmarks_converted.csv file
with open(places_file, newline='', encoding='utf-8') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        place_name = row["place"]

        # Check if the node that is being added does not already exist
        if place_name not in node_ids:
            node = {
                "id": place_name,
                "labels": ["Place"],
                "properties": {
                    "name": place_name,
                    "latitude": float(row["latitude"]),
                    "longitude": float(row["longitude"]),
                    "type": row.get("type", ""),
                    "type2": row.get("type2", ""),
                    "subject": row.get("subject", ""),
                    "subject2": row.get("subject2", "")
                }
            }
            json_nodes.append(node)
            node_ids.add(place_name)
```

Figure 10 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.

```
relationship = {
    "from": f"{source}",
    "to": f"{target}",
    "type": rel_type.upper(),
    "properties": {
        "relationship": rel_type
    }
}
all_relationships.append(relationship)
```

Figure 11 Relationship Python Script Converter

5.2.1.2 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.

```
CALL apoc.load.json("https://pastebin.com/raw/pXCr8xQc") YIELD value
WITH value
CALL apoc.create.node(value.labels, apoc.map.merge({id: value.id}, value.properties)) YIELD node
RETURN node;
```

Figure 12 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.

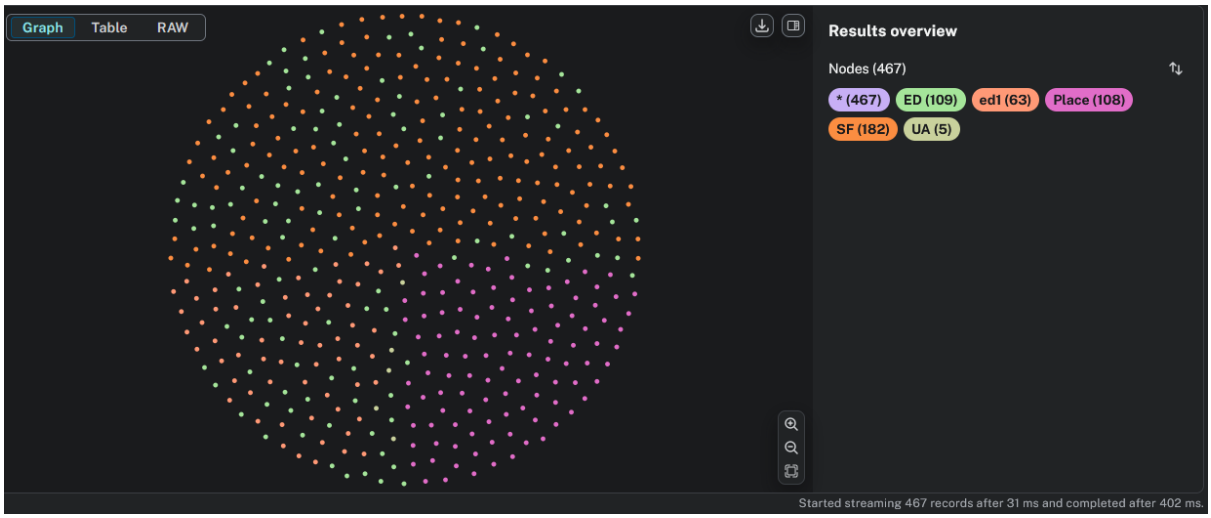


Figure 13 Nodes in Neo4J without edges

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.

```
CALL apoc.load.json("https://pastebin.com/raw/JwWRJhgY") YIELD value
WITH value
MATCH (a {id: value.from})
MATCH (b {id: value.to})
WITH a, b, value
CALL apoc.create.relationship(a, value.type, value.properties, b) YIELD rel
RETURN a.id AS from, type(rel) AS type, b.id AS to, rel;
```

Figure 14 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 15 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 16 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.

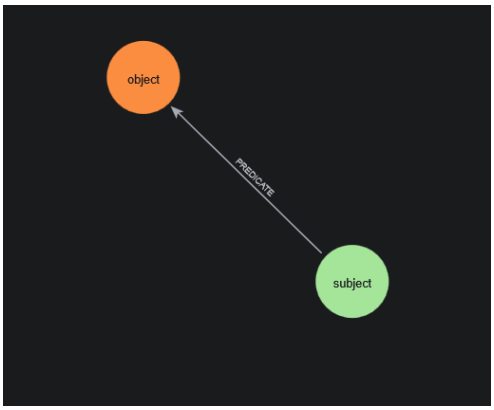


Figure 17 Nodes by themselves in Neo4J

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.

	A	B	C
248	Riverside_ED	contains	Boiler_Plant_Maintenance_Glamorgan_Ltd
249	Riverside_ED	contains	Tuck_in_Grill_House
250	Riverside_ED	contains	St_Mary_of_the_Angels_R_C_Church_Canton
251	Rogerstone_ED	contains	Bassaleg_Junction_railway_station
252	Splott_ED	contains	Sundial
253	Splott_ED	contains	Health_Visitors_Office
254	Splott_ED	contains	Lighting_Tower
255	Splott_ED	contains	Multiple_Sclerosis_Society
256	Splott_ED	contains	Gypsy_Caravan_Site
257	Splott_ED	contains	Four_Elms_Medical_Centres
258	Splott_ED	contains	RAF_Pengam_Moors
259	St._Athan_ED	contains	Aberthaw_power_stations
260	St._Augustine's_ED	contains	Dingle_Road_railway_station
261	St._James_ED	contains	Ruperra_Castle
262	St._Martins_ED	contains	Aber_railway_station
263	St._Martins_ED	contains	Caerphilly_railway_station
264	Stow_Hill_ED	contains	Kingsway_Shopping_Centre
265	subject	predicate	object
266	Sully_ED	contains	Barry_Power_Station
267	Tonyrefail_East_ED	contains	Coed-Ely
268	Tonyrefail_West_ED	contains	Coed_Ely_railway_station
269	Trealaw_ED	contains	Dinas_Rhondda_railway_station
270	Trowbridge_ED	contains	Display_One_Ltd
271	Trowbridge_ED	contains	Pump
272	Twyn_Carno_ED	contains	Bute_Town
273	Wenvoe_ED	contains	Copthorne_Hotel_Cardiff
274	Wenvoe_ED	contains	Dyffryn_Gardens
275	Whitchurch_and_Tongwynlais_ED	contains	Tank
276	Whitchurch_and_Tongwynlais_ED	contains	H_T_Fox_Roofing_&_Leadwork
277	Whitchurch_and_Tongwynlais_ED	contains	Institute_of_Cancer_&_Genetics
278	Whitchurch_and_Tongwynlais_ED	contains	Slot_Cars_Wales_Ltd
279	Whitchurch_and_Tongwynlais_ED	contains	Ashvale_Contractors_Ltd
280	Whitchurch_and_Tongwynlais_ED	contains	Cancer_Research_UK

Figure 18 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 are no longer two nodes and an edge by themselves.

5.2.2 Cypher query

To meet the requirements of the use cases listed in the design chapter, the queries need to be designed to give an answer to the use cases. Implementing the cypher query and looking at the outcome in Neo4J which then the same cypher query be used in the ReactJS project. The cypher query can be used in the ReactJS project since the Neo4J driver library is implemented which communicates directly to Neo4J database instance. The use cases:

1. Regional information, for example: list all places in Cardiff.
    a. The cypher query used:
        i. Looking at the data, it shows that Cardiff is in SF_List which is under the header place name.
        ii.

```
MATCH (n)-[r]-(m)
WHERE n.name = "Cardiff"
RETURN n, r, m
```

Figure 19 Use Case 1 cypher Query

iii. The cypher in the figure above finds all nodes in the name field in the place “Cardiff” and it makes sure to return all the results that have connected nodes to the place “Cardiff.

b. The outcome:

i. 

Figure 20 Use Case 1 Cypher Query Output

ii. The node of Cardiff will appear first then all the direct relation nodes will appear, hence finding all the connected nodes directly to Cardiff with different relationship types since this was not specified in the cypher query.

2. To search by place, type, and relationship type. For example, what hospitals are in the north of Cardiff.
   a. The cypher query used:
      i. 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.
      ii.
```
MATCH (n)
WHERE (
  toLower(n.name) CONTAINS toLower("cardiff") OR
  toLower(n.subject) CONTAINS toLower("cardiff")
)
MATCH (n)-[r:N]->(m)
WHERE toLower(m.type) CONTAINS toLower("hospital")
RETURN n, r, m
```

Figure 21 Use Case 2 Cypher Query

      iii. 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.

   b. The outcome:

i.

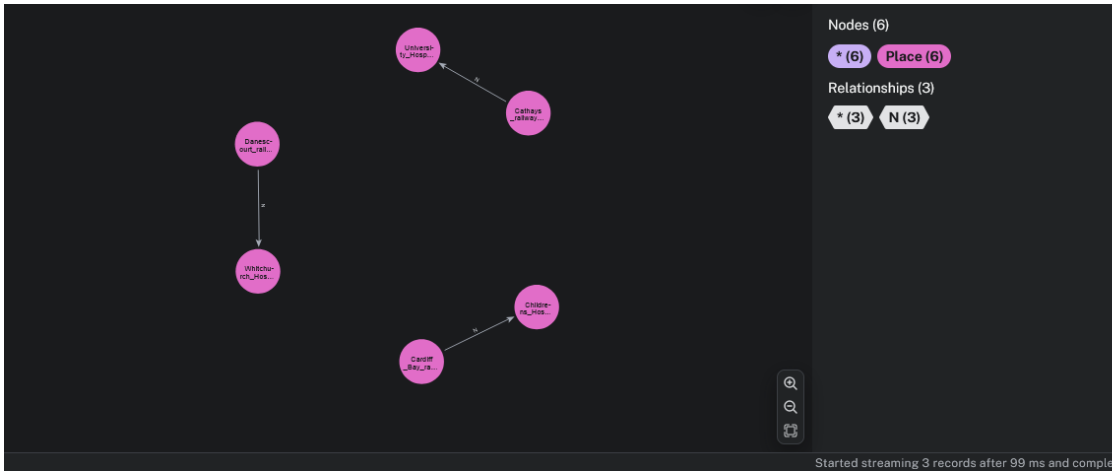


Figure 22 Use Case 2 Cypher Query Output

ii. 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.

3. Find all types in place, for example find all hotels in Cardiff.
    a. The cypher query used:
        i. The data has the places names which will contain the word Cardiff and finding all hotels in Cardiff would mean checking the nodes that are directly related to with the name, subject, or type to contain the word hotel.

ii.

```
MATCH (n)-[r]-(m)
WHERE toLower(n.name) CONTAINS toLower("cardiff")
  AND (
    toLower(m.name) CONTAINS toLower("hotel")
    OR toLower(m.type) CONTAINS toLower("hotel")
    OR toLower(m.type2) CONTAINS toLower("hotel")
    OR toLower(m.subject) CONTAINS toLower("hotel")
    OR toLower(m.subject2) CONTAINS toLower("hotel")
  )
RETURN n, r, m
```

Figure 23 Use Case 3 Cypher Query

iii. The cypher query searches all the places with the name Cardiff and gets the connected nodes that have the name, type, type2, subject, or subject2 that contains the word hotel. This will return a list of

Cardiff nodes which connect to nodes that have the word hotel in them.

b. The outcome:

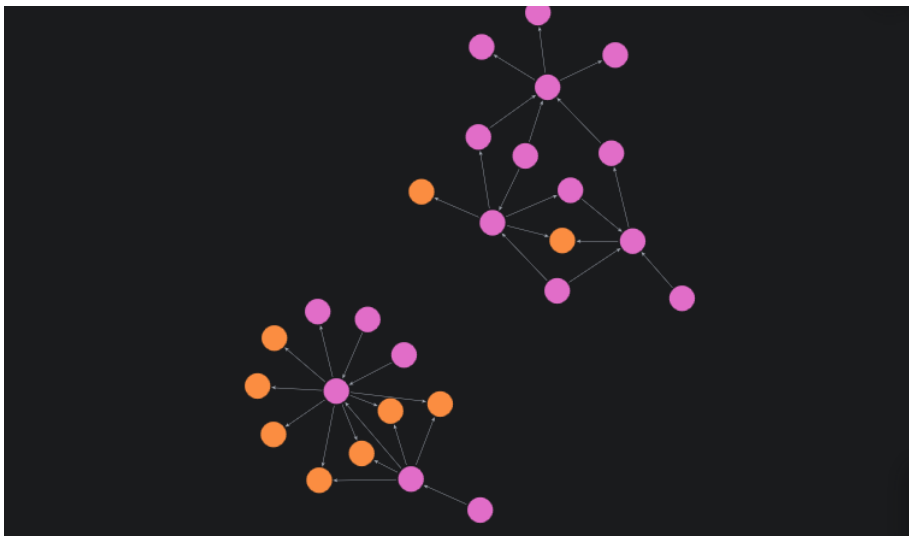
i. 

Figure 24 Use Case 3 Cypher Query Output

ii. There are 5 hotels which multiple nodes connected to them. The nodes connected to them are related to the place Cardiff.

All the use cases have been implemented with an outcome that matches the requirements of the use cases. The next stage is to add the cypher query to the ReactJS implementation phase.

5.3 ReactJS

ReactJS is the choice for the website framework to be created on since this was found to be the most used and well maintained by the community compared to other researched frameworks.

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. (Node.js, n.d.)

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 can be followed later when adding more files to the website. (Create-react-app.dev, 2019)

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. (mui.com, n.d.)

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 overall structure of the ReactJS project code would follow the possibility of creating a component to separate the code wherever possible. There are 4 folders created in src. The folders are:

Components:
- Collapsible Table
  - The table below the map, this shows the list of nodes and edges.
- Detailed Panel
  - Showing the information about a selected node from the map.
- Input Form
  - Search types and input fields with the search and reset buttons section.

Pages:
- DeckGL
  - Containing the DeckGL component.
- Map Tabs
  - Two tabs where it switches between the two map components available.
- ReactLeaf
  - Containing the ReactLeaf component.

Secrets:
- Credentials Neo4J
  - Contains the login information needed to connect to Neo4J database.

Services:
- Neo4J service
  - Sets up a connection to Neo4J which can then send cypher queries to Neo4J to then expect a result back.

Collapsible Table

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:

Name	Type	Subject	Subject2	Type	Type2	Latitude	Longitude	Number of edges
Malpas_ED	ED							2
Tredegar_Park_ED	ed1							1
St_Martins_ED	ed1							2

Figure 25 Table with examples of nodes

Example of places in the table:

Cardiff_town_walls	Place	dbc:History_of_Cardiff		dbo:ArchitecturalStructure dbo:MilitaryStructure	51.48209933923276	-3.1786951538655592	9
Castell_Coch	Place	dbc:Castles_in_Cardiff	dbc:Historic_house_museums_in_Wales	yago:CastlesInCardiff dbo:ArchitecturalStructure	51.53589938112881	-3.2548143325488823	56
Childrens_Hospital_for_Wales	Place	dbc:Hospitals_in_Cardiff		dbo:Hospital	51.4849994778276	-3.1620059054516645	15
Firing_Line_Cardiff_Castle_Museum_of_the_Welsh_Soldier	Place	dbc:Museums_in_Cardiff		dbo:Museum	51.48200424039941	-3.180996888068443	13
Helmont_House	Place	dbc:Hotels_in_Cardiff	dbc:Towers_in_Wales	dbo:ArchitecturalStructure yago:HotelsInCardiff	51.48099845566724	-3.1710055164027473	17

Figure 26 Table with example of place nodes

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

Childrens_Hospital_for_Wales	Place	dbc:Hospitals_in_Cardiff	dbo:Hospital	51.4849994778276	-3.1620059054516645 15

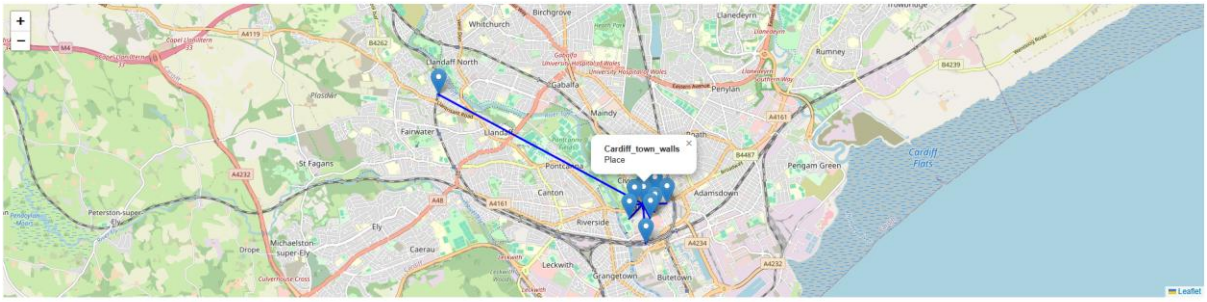
Edges

Type	End Node	End Node Label	Latitude	Longitude
SW	Ty_Pont_Haearn	Place	51.47779635009881	-3.170203498646891
SW	Adam_Street	SF		
SW	Cash_Machine-Bank_Machine_Limited	SF		
SW	Tommys_Bar_Uwic	SF		
W	Cathays_railway_station	Place	51.48909802286611	-3.1792933709291322
W	Cardiff_Alcohol_&_Drugs_Team	SF		
NW	Public_Telephone	SF		
SE	System_Street_MOT_Station	SF		
E	Sundial	SF		
E	Health_Visitors_Office	SF		
E	Lighting_Tower	SF		
E	Multiple_Sclerosis_Society	SF		
E	Four_Elms_Medical_Centres	SF		
S	Back_on_the_Move	SF		
N	Community_Addictions_Unit	SF		

Figure 27 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.

The Names and End Nodes fields are clickable; this is where a cypher search query will take place to show all the information about the selected node. For example, when selecting Cardiff town walls, it will show all the locations of where the edges related are and the Cardiff town walls too. This is shown in the figure below.



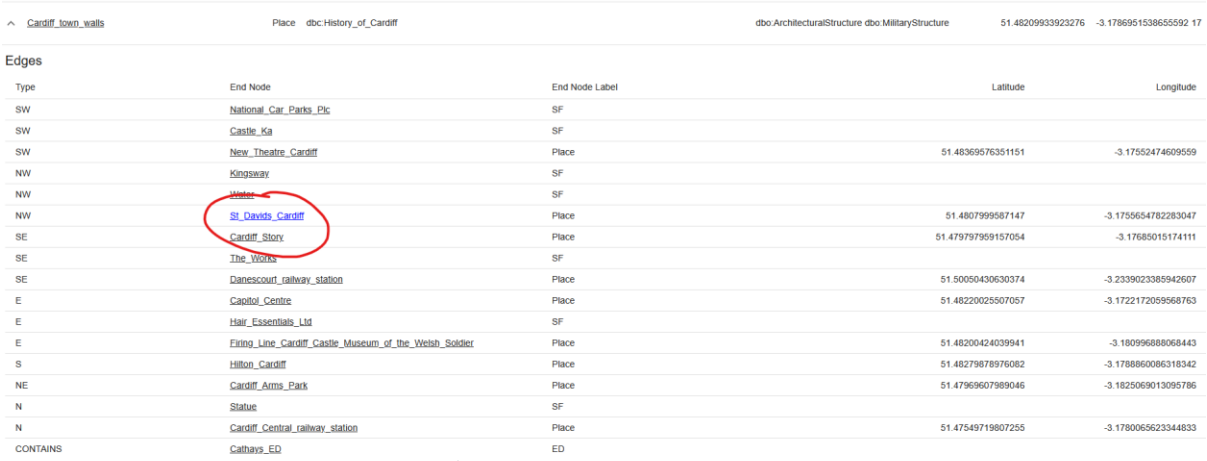
Name	Type	Subject	Subject2	Type	Type2	Latitude	Longitude	Numb of edges
Cardiff_town_walls	Place	dbc:History_of_Cardiff		dbo:ArchitecturalStructure dbo:MilitaryStructure		51.48209933923276	-3.1786951538655592	17

Edges

Type	End Node	End Node Label	Latitude	Longitude
SW	National_Car_Parks_Plc	SF		
SW	Castle_Ka	SF		
SW	New_Theatre_Cardiff	Place	51.48369576351151	-3.17552474609559

Figure 28 Selecting a node to give more info

The End Nodes are also selectable, for example selecting St David Cardiff shows all the directly related edges with nodes, this is shown in the two figures below.

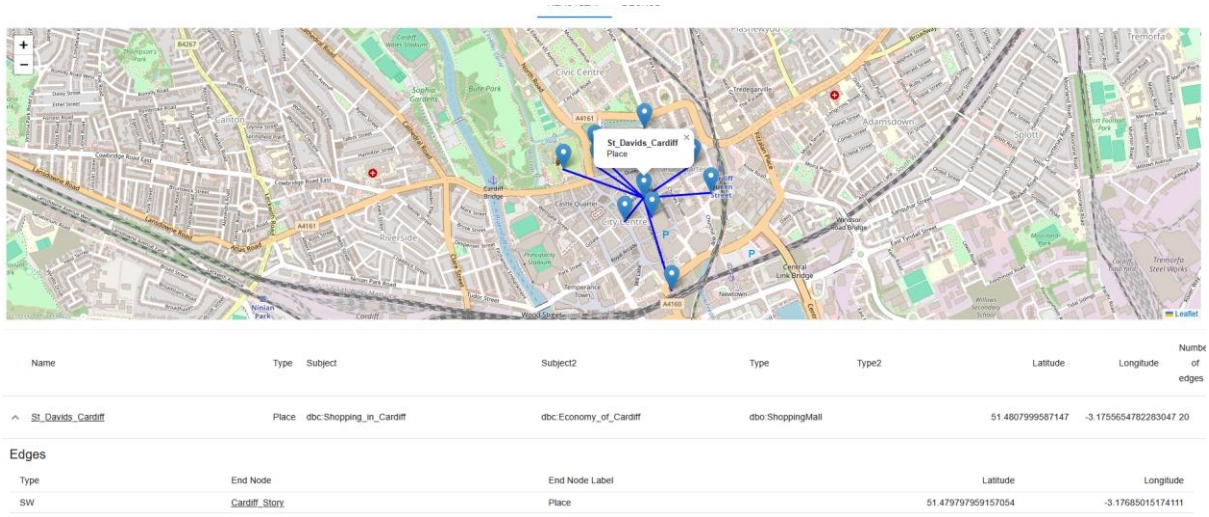


Cardiff_town_walls | Place | dbc:History_of_Cardiff | dbo:ArchitecturalStructure dbo:MilitaryStructure | 51.48209933923276 | -3.17869515386555592 17

Edges

Type	End Node	End Node Label	Latitude	Longitude
SW	National_Car_Parks_Plc	SF		
SW	Castle_Ka	SF		
SW	New_Theatre_Cardiff	Place	51.48369576351151	-3.17552474609559
NW	Kingsway	SF		
NW	Wales	SF		
NW	St_Davids_Cardiff	Place	51.4807999587147	-3.1755654782283047
SE	Cardiff_Story	Place	51.479797959157054	-3.17685015174111
SE	The_Works	SF		
SE	Danescourt_railway_station	Place	51.50050430630374	-3.2339023385942607
E	Capitol_Centre	Place	51.48220025507057	-3.1722172059568763
E	Hair_Essentials_Ltd	SF		
E	Firing_Line_Cardiff_Castle_Museum_of_the_Welsh_Soldier	Place	51.48200424039941	-3.180996888068443
S	Hilton_Cardiff	Place	51.48279878976082	-3.1788860086318342
NE	Cardiff_Arms_Park	Place	51.47969607989046	-3.1825069013095786
N	Statue	SF		
N	Cardiff_Central_railway_station	Place	51.47549719807255	-3.1780065623344833
CONTAINS	Cathays_ED	ED		

Figure 29 Selectable End Nodes



Name	Type	Subject	Subject2	Type	Type2	Latitude	Longitude	Number of edges
St_Davids_Cardiff	Place	dbc:Shopping_in_Cardiff	dbc:Economy_of_Cardiff	dbo:ShoppingMall		51.4807999587147	-3.1755654782283047	20

Edges

Type	End Node	End Node Label	Latitude	Longitude
SW	Cardiff_Story	Place	51.479797959157054	-3.17685015174111

Figure 30 St Davids Cardiff Node Selected

In the figure above where it shows the St David Cardiff, it also shows information on the subject fields which is shopping in Cardiff and Economy of Cardiff with a type of Shopping Mall.

Detail Panel

Having a detail panel on the right-hand side, showing the details of a selected node on the map. The figure below shows what it looks like when the webpage is loaded for the first time.

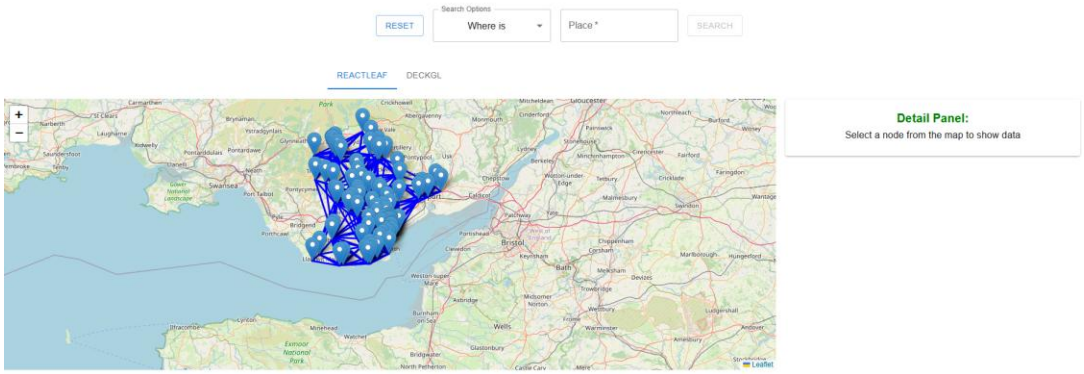


Figure 31 Default View Of Webpage

Showing the details of a selected node is done by clicking on a point in the map. The detail panel will show name, type, location, and properties. It will also show the relationships available. This is shown in the figure below.

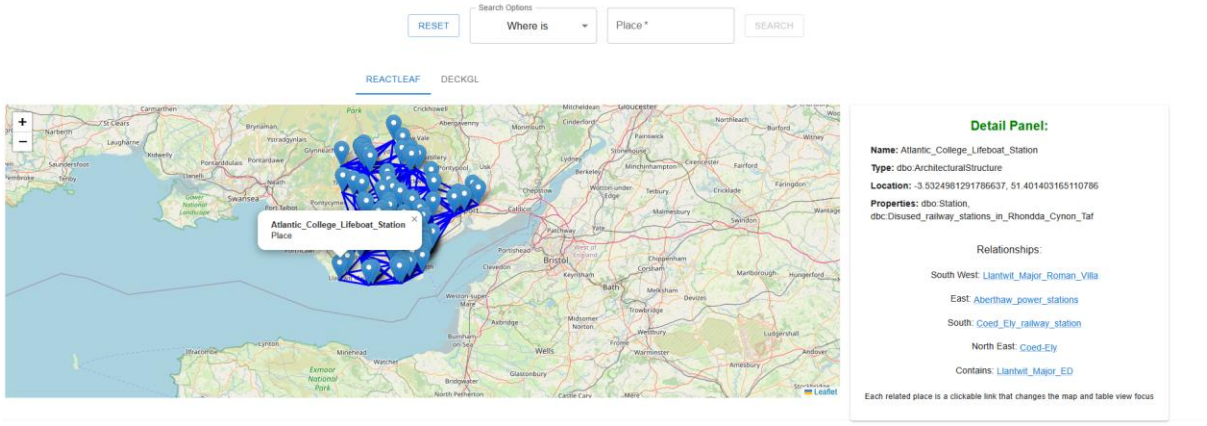


Figure 32 Node Selected, Detail Panel Populated

The relationships appearing is possible because the data is gathered from Neo4J. The node name, type, location, and properties are already available in the point on the map hence passing that data to the detail panel component would be a better choice than having to look for it again in Neo4J. The only missing data would be the edges so, running a cypher query to get the edges of that node which is shown in the figure below.

```
let result = await runQuery(
  `
    MATCH (n {name: $nodeName})-[r]-(m)
    WITH type(r) AS relationshipType, collect(m.name) AS name
    RETURN relationshipType, name
  `,
  { nodeName: selectedNodeFromMap.name }
);
```

Figure 33 Cypher Query Returns RelationshipType And Name

After receiving the edges and their names, they are already grouped by relationshipType so this would make it easier to show in the detail panel.
Every name in the relationships section is clickable, this does the same functionality as a clickable node name in the table below.

5.3.1 Use cases 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.

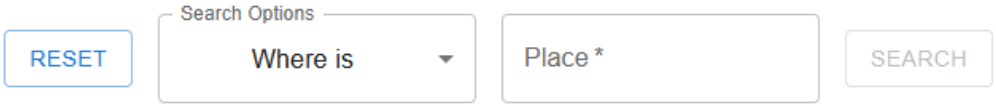


Figure 34 Search by Place GUI

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 5.4.3 and 5.4.4.

Implementing suggestions, when the user types a place then a suggested place names would be shown under the input box. For example, if the user types Cardiff then list of places with the name Cardiff would appear. This is shown in the figure below.

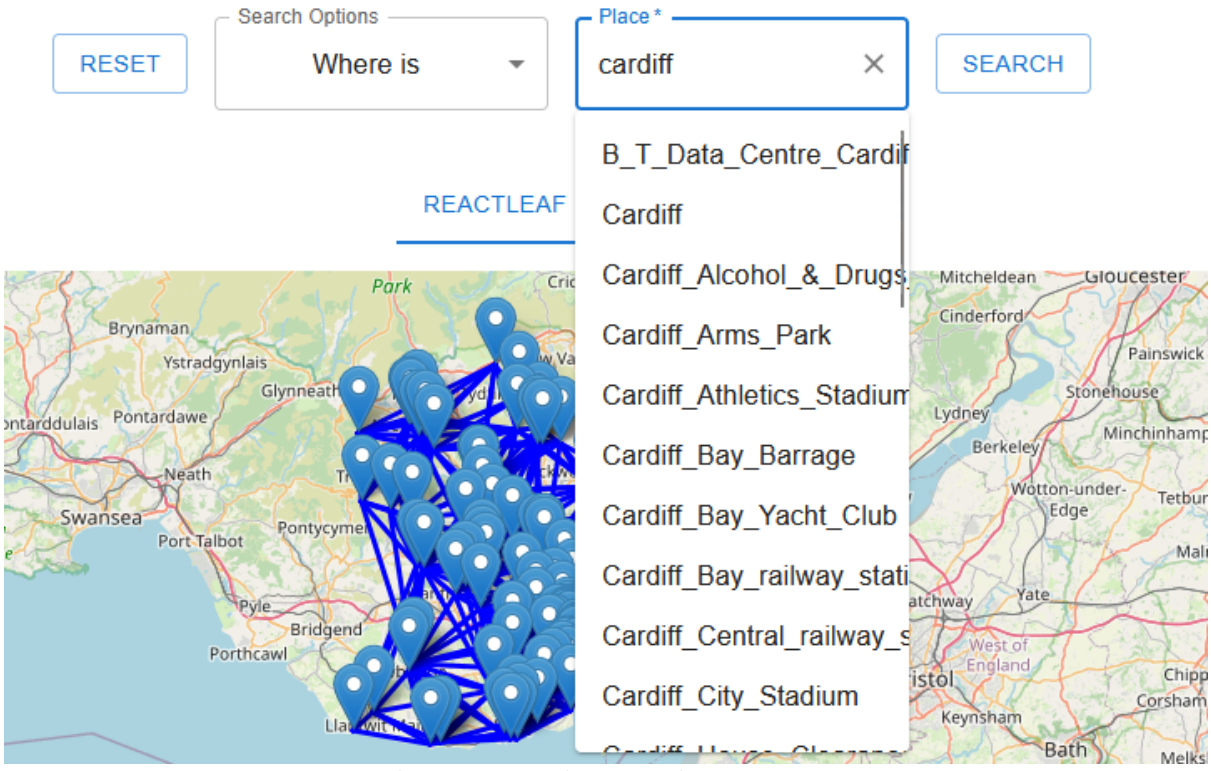


Figure 35 Suggestions under inputbox place

The suggestion uses a component from MUI called Autocomplete. This component takes in the list of data that it would suggest. The suggested data is provided from the result of a cypher query to Neo4J. The Cypher query returns all places’ names shown in the figure below.

```
const autoCompletePlace = await runQuery(
  `
  MATCH (n)
  RETURN DISTINCT n.name AS name
  ORDER BY name
  `
);
```

Figure 36 Autocomplete cypher query for places

The second use case which is to search by place, place type and relationship. This is where relationship input field also has suggestions implemented, this is shown in the figure below.

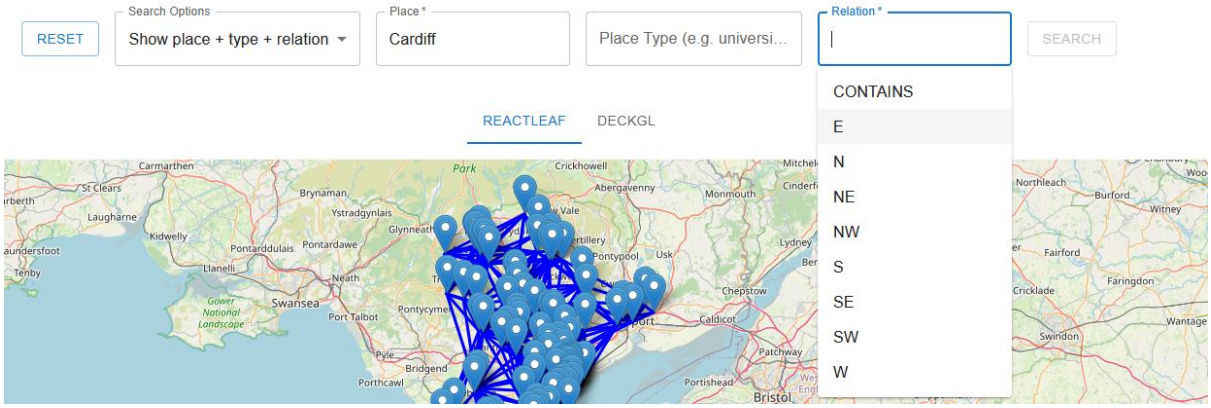


Figure 37 Search by Place, Place Type, and Relationship GUI

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.

The Reset button is implemented so that the user can reset all the search results and input fields to return the data back to the default which shows all the data. This takes into account what search option the user has chosen and resets the input fields only for that search option.

5.3.2 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. (Neo4j Graph Data Platform, n.d.)

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. (Neo4j Graph Data Platform, n.d.)

```
import neo4j from 'neo4j-driver';
import config from '../Secrets/secrets.json';
// Credentials of Neo4J
const uri = config.uri;
const user = config.user;
const password = config.password;

const driver = neo4j.driver(uri, neo4j.auth.basic(user, password));

export async function runQuery(query, params = {}) {
  const session = driver.session();

  try {
    const result = await session.run(query, params);
    return result.records.map(record => record.toObject());
  } catch (error) {
    console.error("Neo4j query error:", error);
    return [];
  } finally {
    await session.close();
  }
}
```

Figure 38 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.

```
result = await runQuery(
  `
  MATCH (n)-[r]-(m)
  WHERE n.name = $place
  RETURN n, r, m
  `,
  { place }
);
```

Figure 39 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).

For the second use case, the same cypher query that was used in Neo4J at chapter 5.4 is also used in ReactJS, this returns the same variables of nodes, edges, and end nodes.

More cypher queries can be added since the base functionality is now in place. By adding a search option then checking if that search option is chosen by checking the search type which then can apply the tailored cypher query for that search type to Neo4J.

Having a specific file that does all the Neo4J functionality helps get access to the Neo4J service. This is done by importing the file into a component then utilising the service by sending the cypher query and expecting a result.

5.3.3 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. (react-leaflet.js.org, n.d.)

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 result will look like what is shown on the figure below.

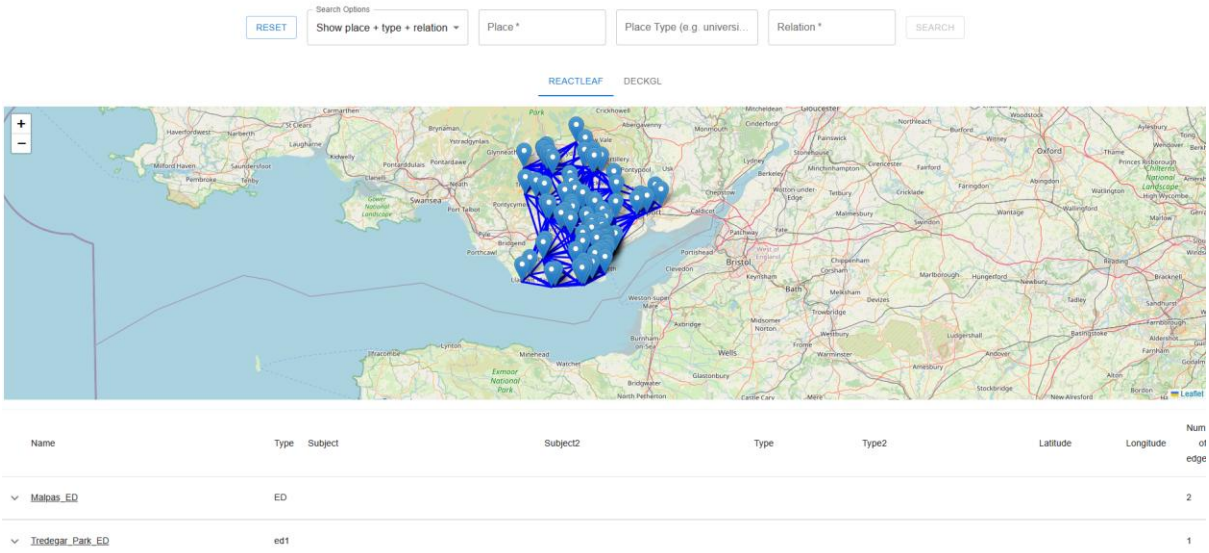


Figure 40 ReactLeaf example map of all data

The map ReactLeaf component would take the following parameters: nodes, edges, nodeMap, loading, set selected node from map.
This data is gathered from Neo4J then processed in ReactJS which then sends it to the map to show. The nodeMap is used for the edges to appear from one node to another. The loading is used to indicate that the data has not returned yet from Neo4J so it shows loading text instead of showing a map with nothing on it. The last one is for the detail panel, when a user clicks on a point in the map, the node data is then sent back to the parent component which sends to the detail panel component to show the details of the selected node.

5.3.4 DeckGL

Another type of map component is DeckGL. To install the components, this command is used: “npm install deck.gl –save”. (Deck.gl, 2025) To get it working with the current setup of ReactLeaf, a GUI adjustment must be made to then accommodate both map components. A new component called MapTabs is created which will handle the tabs that can switch between the two map components, this will allow for a comparison of two map components. The figure below shows the tabs.

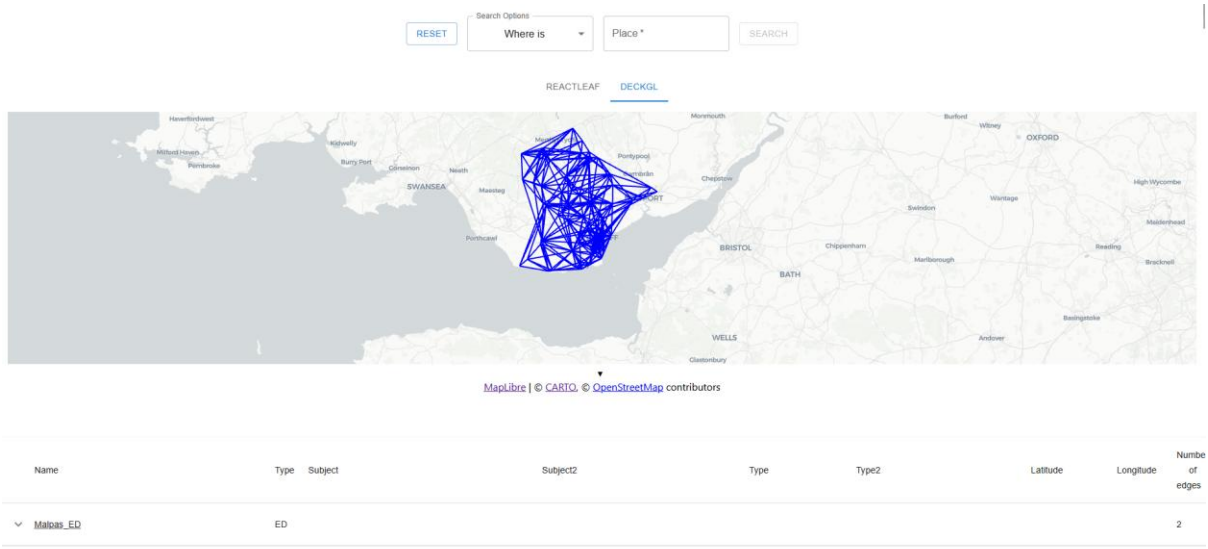


Figure 41 Tabs in ReactJS

The search and results will be in its own custom component, and it will return the result from Neo4J to then send to the map components and the table components. This result data can be used to be sent to other components as well in the future. The DeckGL uses the DeckGL component itself to then show a map using the data from OpenStreetMap. Using layers which are scatter layer and line layer, the scatter layer contains the nodes which plots them on the map using the coordinates of longitude and latitude. The line layer contains the edges which shows the lines between points when they are visible on the map and when there's a relationship.

5.4 Project Source Code and Setup Instructions

The project source code of ReactJS, along with instructions to run it locally, is available in the GitLab repository referenced in Appendix A, which also includes scripts for data conversion, JSON node and relationship files, and guidance on configuring a local Neo4J database.

Chapter 6: Analysis and Testing

6.1 Overview

When checking a system, a few key features need to be looked at: if it works as it should (functional accuracy), if the data is correct and shown in the right way (data and visualization accuracy), and if it is easy for people to use (usability). Testing is also important to make sure the data is correct from the start and displayed properly, so users can trust the results and understand them clearly.

6.2 Analysis

6.2.1 Functional – use cases

The use cases implemented in chapter 5.4.1 meet the requirements by producing various search options which accommodate the use cases' requirements. This is done by having a choice of search options which allows the user to choose from and to do a search with an expected result back. The search options are not limited to the ones implemented since more can be added.

The search section has its own component. There is a variable option called search option. The search option is used to identify which option was chosen from the drop-down menu, once an option is selected, then it identifies which cypher query to choose for the selected search option. Adding another search option would mean to define what data is required then to understand how the cypher query would work best. At the end, adding the search option in the search section component by adding the necessary input fields and cypher query, this will produce a new search option.

6.2.2 Data and Visualisation - Accuracy of knowledge graph visualisation

Checking the accuracy of the knowledge graph visualisation of how well the nodes which contain longitude and latitude align on the map is important. This makes sure that the data being used is accurate.

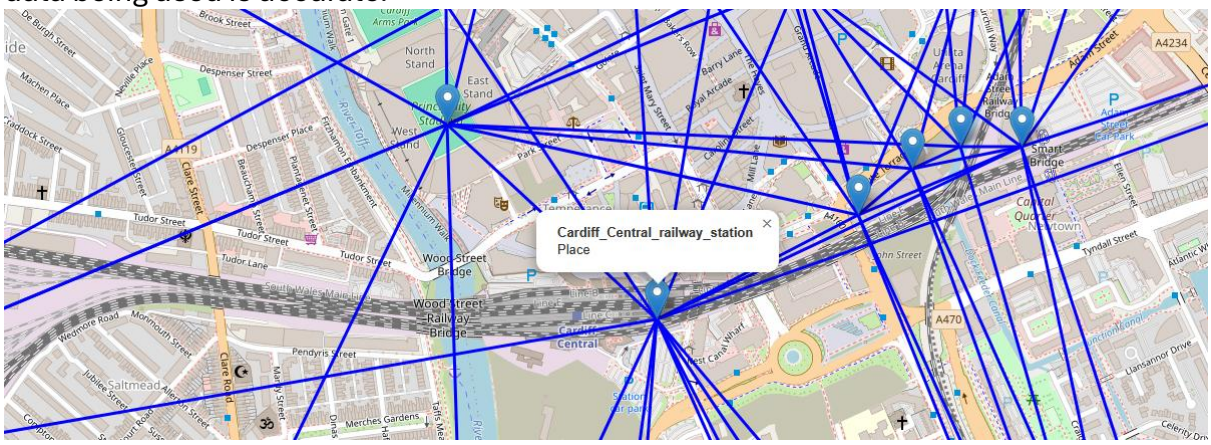


Figure 42 A Point On The Map Demonstrating The Accuracy

In the above figure, this is taken from the ReactLeaf map component and clicking on a place which is Cardiff Central Railway Station. This is accurate, when comparing this to google maps, this shows the exact location of how it is shown in the ReactJS website. The google map is shown in the figure below.

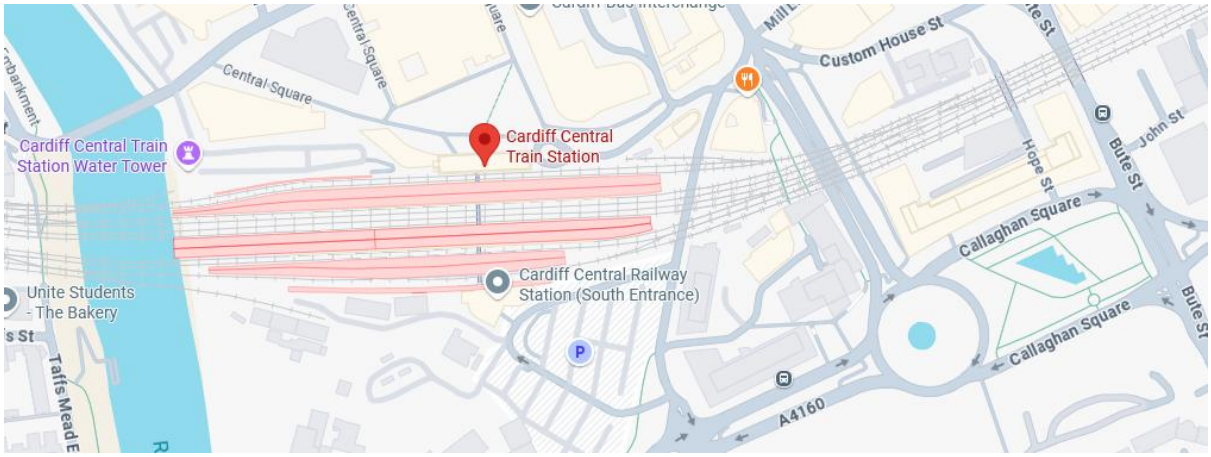


Figure 43 Google Maps - A Point On The Map

6.2.3 Usability

Users can view the geospatial knowledge graph data via different views for different reasons. Having a map view gives the places which contain longitudes and latitudes to appear on the map with points and the connections between them to also appear on the map.

The detail panel shows the details of a selected node on the map. This helps the user understand what this point has. For example, clicking on Kingsway Shopping Centre shows the type, properties it has, and the relationships it has. This is shown in the figure below.

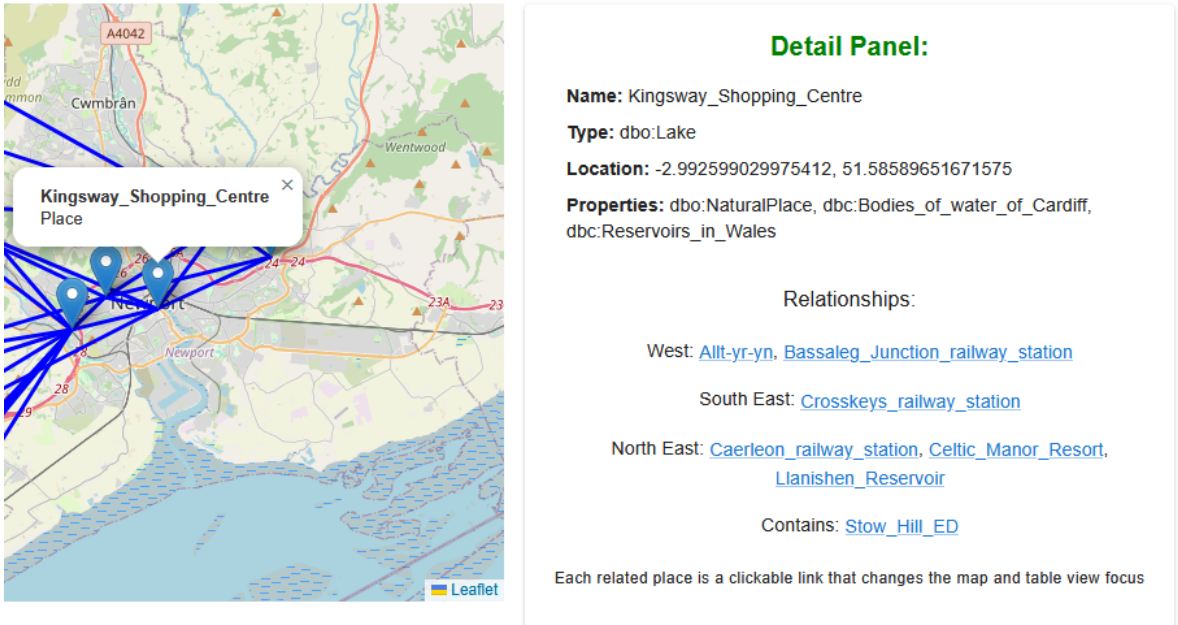
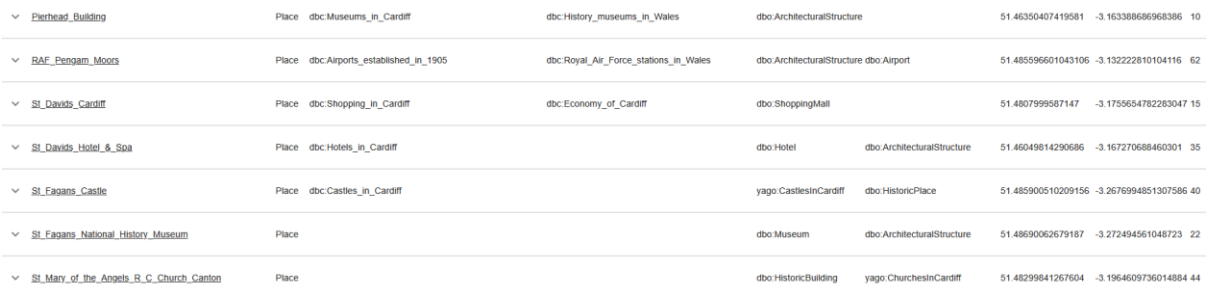


Figure 44 Kingway Shopping Centre - Detail Panel

Every location that is connected with the node that was selected is clickable in the detail panel. This allows for the user to explore locations easier.

The table below shows the information of all nodes and edges that appear on the map. This is shown in the figure below.



Pierhead_Building	Place	dbc:Museums_in_Cardiff	dbc:History_museums_in_Wales	dbo:ArchitecturalStructure		51.46350407419581	-3.163388686968386 10
RAF_Pengam_Moors	Place	dbc:Airports_established_in_1905	dbc:Royal_Air_Force_stations_in_Wales	dbo:ArchitecturalStructure	dbo:Airport	51.485596601043106	-3.132222810104116 62
St_Davids_Cardiff	Place	dbc:Shopping_in_Cardiff	dbc:Economy_of_Cardiff	dbo:ShoppingMall		51.4807999587147	-3.1755654782283047 15
St_Davids_Hotel_&_Spa	Place	dbc:Hotels_in_Cardiff		dbo:Hotel	dbo:ArchitecturalStructure	51.46049814290686	-3.167270688460301 35
St_Fagans_Castle	Place	dbc:Castles_in_Cardiff		yago:CastlesInCardiff	dbo:HistoricPlace	51.485900510209156	-3.2676994851307586 40
St_Fagans_National_History_Museum	Place			dbo:Museum	dbo:ArchitecturalStructure	51.48690062679187	-3.272494561048723 22
St_Mary_of_the_Angels_R_C_Church_Canton	Place			dbo:HistoricBuilding	yago:ChurchesInCardiff	51.48299841267604	-3.1964609736014884 44

Figure 45 Table - Showing Nodes

By expanding the node, it can show the edges connected to it and the names are all clickable so it will allow the user to explore easier than having to search for it manually.

Overall, usability has been a focus to allow the user to explore the data in more convenient ways than just having one option that does one thing. These features are all interconnected by adapting to what the user clicks or searches.

6.3 Testing

6.3.1 Map components

Loading the data accuracy, this is coming from Neo4J using the following cypher query in ReactJS, shown in the figure below.

```
result = await runQuery(`
  MATCH (n)
  OPTIONAL MATCH (n)-[r]->(m)
  RETURN n, r, m
`);
```

Figure 46 Default Cypher Query

The same cypher query can be applied in Neo4J Aura Query, returns 465 nodes with 2630 edges. The same values are loaded in ReactJS as shown in the figure below.

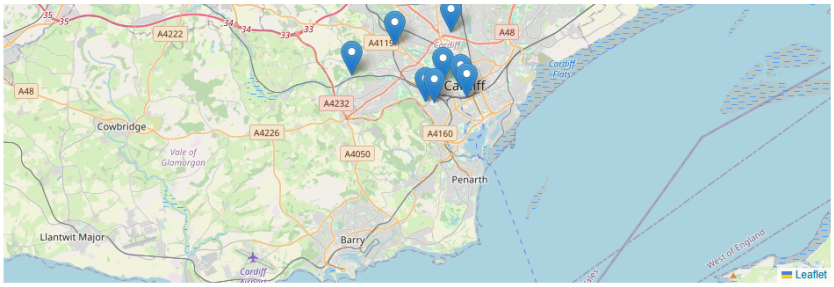
```
nodeList length total 465
nodeList length only lng,lat:  108
edgeList length 2630
```

Figure 47 Console Log The Node With/Without Longitude/Latitude And Edge List Length

The same exact values are loaded into ReactJS, the nodes and edges are all available to use. Less nodes are there once they are filtered to places that contain longitude and latitude values since these will be loaded onto the maps of ReactLeaf and DeckGL.

6.3.2 Table

The table will contain all the data that is shown on the map plus the points that cannot appear since they do not have longitude and latitude. When searching for Tesco as a place, it shows all the results from Neo4J in the table, shown in the figure below.



Name	Type	Subject	Subject2	Type	Type2	Latitude	Longitude	Nu of edg
Tesco	SF							9

Edges

Type	End Node	End Node Label	Latitude	Longitude
SW	St_Mary_of_the_Angels_R_C_Church_Canton	Place	51.48299841267604	-3.1964609736014884
W	Cardiff_Arms_Park	Place	51.47969607989046	-3.1825069013095786
NW	Cardiff_Central_railway_station	Place	51.47549719807255	-3.1780065623344833
SE	Danescourt_railway_station	Place	51.50050430630374	-3.2339023385942607
E	St_Fagans_Castle	Place	51.485900510209156	-3.2676994851307586
S	University_Hospital_of_Wales	Place	51.50700061831285	-3.1900004693274173
NE	Cardiff_International_Sports_Stadium	Place	51.47310324665489	-3.209995673970293
N	Cardiff_Athletics_Stadium	Place	51.472796945302925	-3.2030619869754307
CONTAINS	Riverside_ED	ED		

Figure 48 Tesco Search Results

In the figure above there are 8 places with latitude and longitude, the same is reflected on the map above with 8 points appearing. There are two points which do not appear, the node itself of Tesco and the edge node of Riverside, both do not contain longitude and latitude.

6.3.3 Detail Panel

The panel shows the information of the selected node on the map. This uses the data provided from Neo4J, either from the default loading all nodes and edges or when performing a search. The edges are loaded from Neo4J after selecting a node, this uses a cypher query. Producing a cypher query in Neo4J to return the results that will appear the same in the detail panel. The figure below shows the Neo4J cypher query and the results of the place “Cardiff_Central_railway_station”.

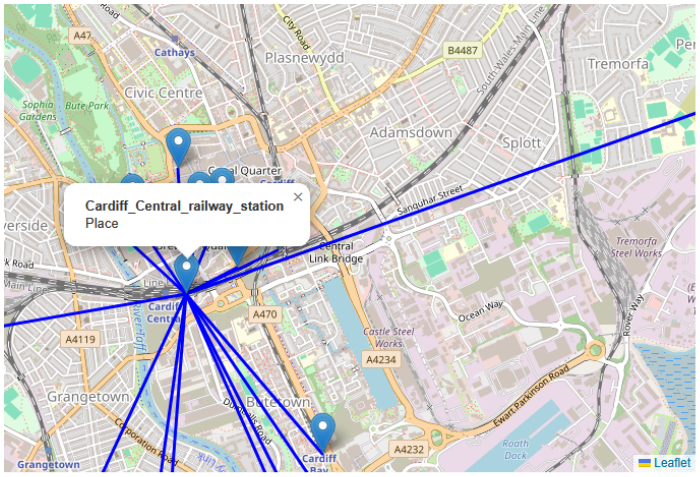
```
MATCH (n {name: "Cardiff_Central_railway_station"})
OPTIONAL MATCH (n)-[r]-(m)
RETURN
  n.name AS name,
  n.type AS type1,
  n.latitude AS Latitude,
  n.longitude AS longitude,
  n.type2 AS type2,
  n.subject AS subject,
  n.subject2 AS subject2,
  collect({ relType: type(r), target: m.name }) AS edges
```

Table | RAW

name	type1	Latitude	longitude	type2	subject	subject2	edges
"Cardiff_Central_railway_station"	"dbo:ArchitecturalStructure"	51.47549719807255	-3.1780065623344833	"dbo:Station"	"dbc:Railway_stations_in_Cardiff"	""	[{ relType: "W", target: "Maplin_E s" }, { relType: "W", target: "Bus_Depo }, { relType: "W", target: "RAF_Peng }, { relType: "NW", target: "Cardiff_ }, { relType: "NW", target: "Millenni m" }, { relType: "NW", target: "National

Figure 49 Cypher Query Search By Name

The below detail panel shows the same data as the figure above from Neo4J cypher query.



Detail Panel:

Name: Cardiff_Central_railway_station

Type: dbo:ArchitecturalStructure

Location: -3.1780065623344833, 51.47549719807255

Properties: dbo:Station, dbc:Railway_stations_in_Cardiff

Relationships:

West: Maplin_Electronics, Bus_Depot, RAF_Pengam_Moors

North West: Cardiff_Arms_Park, Millennium_Stadium, National_Car_Parks_Plc, Way_Ahead, Rugby_Football_Ground, Pure_Gym, Vue_Cinemas, Ben_&_Jerrys, Cash_Machine-Royal_Bank_of_Scotland, Tesco, Tuck_in_Grill_House, Cardiff_Bay_railway_station

South East: Mermaid_Quay, St_Davids_Hotel_&_Spa, Techniquest, Road_Tunnel, Mermaid_Quay_Waterbus, Stephen_Doughty_MP-Cardiff_South_&_Penarth, Cadwyn_Capital_LLP, Recreation_Ground, Bishop_&_Olden_Solicitors, Travelling_Crane, Adelaide_Street, Institute_of_Welsh_Affairs, Graving_Dock_No-3, Depot, Cadwaladers_Ice_Cream

East: Ing_Direct, Cardiff_Athletics_Stadium

South: Bayscape, Cardiff_International_Pool, Business_Centre, Taxi_Vans_Cardiff, Cardiff_House_Clearance, Cardiff_Yacht_Club, Sewage_Pumping_Station, Watkiss_Way-1, Weighbridge, Appletree_Nursery, Frankie_&_Bennys, B_T_Data_Centre_Cardiff_Bay, Cardiff_Ice_Rink, Planet_Ice_Cardiff_Arena, Watkiss_Way-2, Gantry

North East: Meridian_Gate_Cardiff, Ty_Pont_Haearn, Castle_Quarter, The_Golden_Cross, Bute_Terrace, Eastbrook_railway_station

North: Cardiff_Story, Cardiff_town_walls, Kingsway, The_Works

Contains: Butetown_ED

Each related place is a clickable link that changes the map and table view focus

Figure 50 ReactJS Detail Panel Node Selected

The values of both figures above, from Neo4J Aura and from ReactJS Detail Panel match. For example, the type and other properties are the exact same values, and the relationships of West are 3 places which also appear 3 places in Neo4J Aura result from the cypher query.

Chapter 7: Evaluation & Conclusion

7.1 Evaluation

Choosing a web framework was difficult decision since there are a lot of very well-made frameworks. ReactJS was chosen because it has a big community backing, meaning it has a lot of third-party libraries. This is beneficial for this project since it utilises GUI components, it is essential to visualise the geospatial knowledge graph in various unique ways. GUI was generally done by material UI, which is based on google material UI. The library has a wide range of options that the developer can choose from hence the expandability can be achieved by choosing MUI library. Implementing them was not difficult since clear instructions are provided on MUI's official website. Documentations also include live demonstrations of most GUI components offered.

There are many database options available for managing geospatial knowledge graphs. Comparing database solutions by researching what is the best choice for this project. The project required a simple yet effective database solution. Effective meaning that the database is made to handle knowledge graphs, especially geospatial data. Utilising the relationships between the nodes is a minimum requirement. Neo4J has all the checkboxes checked for this project by providing a robust, reliable, and well maintained since it also needs to be accessible in ReactJS. ReactJS can access the Neo4J database directly by using the Neo4J-driver library provided by Neo4J. This made the implementation much easier and quicker than implementing a backend API.

Importing the data into Neo4J was not straightforward since the data was in csv format. Converting them required using a simple yet effective Python script. Producing 2 JSON files that contain nodes and edges which then can be imported easier into Neo4J. By checking the data integrity, it assures that they are maintained throughout the process of converting and importing. At the end, the data is shown in ReactJS website. Also checking if the data is correct by having a cypher query get the results, running it directly in Neo4j Aura and comparing the results with the data that appears on the website.

Representing the data on the ReactJS website was done in different ways. Map, table, and detail panel. The map shows the locations by having points with edges appear, helping the user to get a good picture of where the locations are shown. The table gives more information since not all the data has coordinates, allowing the user to see all the data plus the missing ones from the map. Detail panel gives more information on the selected point on the map. If the user would like to know more about a location on the map, they can click on it and detailed information would appear, giving the user a better understanding of locations. Having search options on the website gives the user the power to filter what locations they want to see, this gives the user three search options, search for: a place, a place plus type, a place plus type plus relation. The result of the search is reflected on the map and table.

More GUI capabilities could be implemented since the MUI library has many more components. And more search options can be added where the ReactJS code has been done in such a way that more options can be easily added without having to modify any

of the code significantly. This was done so the project can be further developed in the future, with numerous possibilities for extension.

7.2 Conclusion

Meeting the overall aim first required achieving the individual objectives. The first objective is to understand what it takes to create a webapp that has the capability of visualising a geospatial knowledge graph, this was achieved by the background research that was done. Exploring the possible web frameworks that contain features such as scalability, easy to develop where it does not compromise on expandability. Assessing an existing product such as kepler.gl which shows geospatial knowledge graphs provided valuable insights into the requirements and expectations of end users in terms of usability and functionality.

The second objective is to compare different types of visuals that represent the geospatial knowledge graph. The objective was achieved by implementing two maps by using different libraries, a table that shows the data in more detail, and a detailed panel. Having three different types of visual components integrated into the webapp provides the end user with various possible solutions. Analysing and testing each visual type was done to verify the accuracy of the data since there were numerous steps taken to making the data usable.

Comparing web frameworks is the third objective which was achieved by researching several possible frameworks. There were some requirements to be met, spending less time on developing the website but more time on implementing the required components, enabling the project to have numerous unique visuals implemented. Not compromising on quality of the code by having the possibility of adding different visual types, can be accomplished through having the data readily accessible throughout the code and each visual type has a separate component making it loosely coupled.

The final objective is to test the web application using the knowledge graph data. This can be done once the data is available, allowing the verification of its accuracy. Comparison with the Neo4j data can be performed both on the website and directly in the database. The testing was completed successfully, confirming that the data is correct. Checking the maps implemented, a comparison was done with well-known existing map, Google Maps.

To conclude, a well-structured and functional code has been developed, enabling future implementation and demonstrating the geospatial knowledge graph using various techniques to address the end-user queries.

Chapter 8: Future Work

Features that could not be implemented which could be done in the future are more variety of map types and adding search feature to the table.

During the exploring of existing products, kepler.gl has many features which could be implemented in this project. The features that can be investigated are the simple filtering options and heat maps. Adding a filter option on the side of the map that include options such as showing all places so filtering by type or showing all schools so filtering by subject. This can introduce a quick and easy simple use cases that the users can utilise. Adding a heat map option would show where most of the locations are located at or it can be used to show how dense the population in specific locations are. Implementing those features is possible and easily done since this project uses components in ReactJS. Implementing components which can interact with others by sending necessary data back to the parent component which communicates with its children's components. This enabled the development of this feature in a quick and easy way.

Currently the table shows all the data from the search results and by default it shows all the nodes and edges available. By showing a lot of data, the user can be overwhelmed hence implementing a search feature or even adding sorting feature on the existing fields could help the user in identifying the point of interest for them. Using MUI for GUI in ReactJS means a simple implementation by utilising a component from MUI that features the search component in a table and the sorting feature in the table.

This is just an example of the features possible to implement, as the project was built from the ground up to make adding new features fast and easy.

Chapter 9: Reflection

Prior planning was required to understand how much this project requires to have a complete solution. This is why making a Gantt chart helped in having a good idea on what needs to be done in a specific timeframe. The plan is shown in appendix B. This is where I was able to stay on track and recognise when I was starting to drift off it. This has allowed me to maintain steady progress, which in the end lead to a detailed report and a programmed solution.

Programming the ReactJS website solution presented some challenges, but they were easy to overcome thanks to the extensive documentation and the availability of helpful third-party libraries, which are also utilised in this project. The third-party libraries mainly helped in implementing the GUI and the map components, then I was able to connect them together by the data. This has allowed me to take control in implementing a solution that can be worked on in the future. Allowing expandability in a program is a challenge since it might take extra time during the implementation stage compared to straight up implementing a solution without considering other factors such as scalability.

There are more features that can be added in the program which may have an impact on usability. This could have been studied by allowing trials to wide range of users which they can provide feedback of their experience. It could have helped in understanding the missing features which users may want or even know the limitations in more depth of the existing features. This is not a deal breaker since working on the project in the future is made easy by utilising various features as mentioned previously.

Using Neo4J was generally good, no limitations that I ran into during this project however other solutions could have been explored in more depth by implementing them. This was the stage where I began to feel the time constraints. Implementing an alternative database solution would have required considerable time, since this project is primarily focused on delivering a visual website solution.

I did not have a broad understanding of knowledge graphs, particularly geospatial graphs. Through this project, I learned the most in this area by conducting research and expanding my knowledge into geospatial knowledge graphs and by having regular meetings which helped be guided correctly, the meeting notes are provided in appendix C.

References

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].

Deck.gl. (2025). Installing and Running Examples | deck.gl. [online] Available at: https://deck.gl/docs/get-started/getting-started [Accessed 7 Aug. 2025].

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].

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].

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

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.

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.

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

keplergl (2018). GitHub - keplergl/kepler.gl: Kepler.gl is a powerful open source geospatial analysis tool for large-scale data sets. [online] GitHub. Available at: https://github.com/keplergl/kepler.gl [Accessed 28 Aug. 2025].

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.

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

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].

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

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].

QGIS (2025). *QGIS*. [online] qgis.org. Available at: https://qgis.org/ [Accessed 28 Aug. 2025].

Qgis.org. (2025). *QGIS Map*. [online] Available at: https://hub.qgis.org/map-gallery/19/ [Accessed 28 Aug. 2025].

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.

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

Rosas-Chavoya, M., Gallardo-Salazar, J.L., López-Serrano, P.M., Alcántara-Concepción, P.C. and León-Miranda, A.K. (2022). QGIS a constantly growing free and open-source geospatial software contributing to scientific development. *Cuadernos de Investigación Geográfica*, 48(1), pp.197–213. doi:https://doi.org/10.18172/cig.5143.

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].

List of Figures

Appendices

Appendix A - GitLab Project, contains the converters, ReactJS webapp, csv, json files, instructions on how to run the project

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

Appendix B – Gantt Chart, project plan

Task Name	Assignment	Start date	Due date	Bucket	Progress	Priority	Labels	Quick look
~~Project initialisation~~		6/23/2025	6/30/2025	Planning	Completed	Medium	Add label	2/2
~~Literature Review~~		7/7/2025	7/23/2025	Initiating	Completed	Medium	Add label	2/2
~~Background Research~~		7/14/2025	7/23/2025	Initiating	Completed	Medium	Add label	4/4
~~Design~~		7/23/2025	7/28/2025	Initiating	Completed	Medium	Add label	1/1
~~Implementation~~		7/28/2025	8/15/2025	Initiating	Completed	Medium	Add label	4/4
~~Testing~~		8/16/2025	8/18/2025	Initiating	Completed	Medium	Add label	
~~Evaluation & Conclusion~~		8/19/2025	8/22/2025	Initiating	Completed	Medium	Add label	2/2
~~Reflection & Future Work~~		8/23/2025	8/26/2025	Initiating	Completed	Medium	Add label	

Appendix C – Meeting notes

Meeting 1

Date: 16/06/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

Discussed:

- What projects I am interested in.
- How to choose a project.

For next week:

- Research on possible projects with geospatial knowledge graphs and webapps.

Meeting 2

Date: 23/06/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

Discussed:

- Showed a project that uses knowledge graph geospatial
- Web application, have a database, neo4j (used to store the data and query to get the data,
- Look at examples of how the visualisation is done:
    - Knowwheregraph.org
    - Github: relfinder
- Use the datasets in Ethan's project
- Start with simple files of csv and link to front, link it to neo4j, frontend and backend.

For next week:

- Write the project title, aim, objectives, Gantt chart plan.

Meeting 3

Date: 30/06/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

To discuss:

- Project plan, title, statement, objectives and aim.

Questions:

- To check if the objectives and aim are suitable for the project.

Discussed:

- Gitlab with University.
- Hosting a reactjs webapp using university servers.
- To add a video recording of the dissertation which will help with the video report.

For next week:

- To check neo4j or mongodb cloud options and how that can connect to reactjs.
- Communicate with Alia to check on the database model.

Meeting 4

Date: 17/07/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

To discuss:

- Able to connect neo4j and reactjs with the data appear, trial is now done.
- Need a way to host reactjs website.
- To implement two solutions in the website project using two different components that can load knowledge graphs visually on a map with different options, one is called: react leaf and the other is called: DeckGL

Questions:

- Compare react leaf and deckgl? Yes
- Should I setup pipeline on gitlab for reactjs webapp to run? Alia asking about domain
- Request an account on neo4j from Cardiff uni IT team? No need, use free version.

Discussed:

- Choosing the map components:
    - Check the previous project on their solution with how they show the knowledge graph.
    - Show the graph, a textual version of it and show objects on the map.
    - OpenStreetMap is completely free. (To add to the research)
- Download Neo4J on a local machine.

For next week:

- How to create Neo4J graph, and access graph.
- OpenStreetMap.
- ReactJS implement react leaf and deckgl.

Meeting 5

Date: 24/07/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

To discuss:

- I am able to connect nodes and edges from the data that is given to me into neo4j, I am using the neo4j cloud database so I am able to connect to it online.

Questions:

- Should I contact Shancang Li (MSc Project Coordinator) about the new title for pats website? No need for now, this can be done at the end.
- Any info on reactjs hosting? IT department got the message from Alia, awaiting their response.

Discussed:

- How am I going to model the data and how to build the interface to view the data?
- Three types of relationships, topological (a place contains a place, intersects between others), directional (north east etc), proximity (distance).

For next week:

- To check if neo4j database instance cloud stays running at all times.
- Add a chapter to modelling the data (Check modelling data in neo4j) and requirements and loading the data and develop the database and displaying the data
    - Requirements: Show the requirements of what needs to be implemented to meet the requirements, for example need to find the city that has the closest cities to it.
    - Modelling data: How different options in data models impacts the speed of queries.
        - Neo4j can return how long it took to query.
    - Loading the data: Read csv files and load them to neo4j into a database.
    - Develop the database.
    - Display the data in reactjs, design the interface.

Meeting 6

Date: 31/07/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

To discuss:

- Completed the graph models, where 5 nodes and edges exist.
- Added use cases to find specific data.
- Data model, x and y in sf_list converted to lat and long for it to work in react leaf.
- Created a cypher query for one use case and indeed I would only need specific graph models for specific queries.
- Able to view the nodes and edges on a map in ReactJS using react leaf.

Questions:

- To start implementing the UI with the agreed use cases.

Discussed:

- The data I have: csv file, list of places that shows other places connected to it.
- The next steps to create JSON, load Neo4J, implement UI.

For next week:

- Create JSON file: place name, type, properties such as contains, place id, east of, west of
- Loading the JSON file into Neo4J, make it a section for this.
- Create Neo4J graph using the JSON file, check if all nodes are connected so they have edges, proper naming conventions etc.
- User use case, design of the interface:
    - Browsing the graph and searching for places on the graph.
    - Search the graph by place name, for example search for a place called Cardiff university, this will return a focused map on Cardiff university, there will be two kind of views:
        - A map view, hovering over the pin will show the place name.
        - A list (this could be the information about Cardiff university, relationship contains in etc.
            - Possibly add a clickable option on the returned values in the list.
    - Search places by place type, search for place type within a place
        - Search for universities within Wales, another example search for universities north of Cardiff, to show the results in:
            - A map view.
            - The list the information, this will be a scrollable view of all pins found.

Meeting 7

Date: 07/08/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

To discuss:

- Completed csv to JSON converters of nodes and edges.
- Imported them into Neo4J cloud database instance.
- Added the cypher queries to ReactJS, both use cases are functional.
- Added GUI components to ReactJS, collapsible table with all nodes and edges appearing or when search is active, it shows the results only.

Questions:

- How will the testing be done in such a way that it is sufficient to cover the data.

Discussed:

- Conceptual design of the actual data, to check if the system is working correctly.
- Justify the design for the user, for example this is a strong feature of having a map and a table.

For next week:

- Aurom survey, interface, ethan project check.
- Example of GUI functionality: make the place name clickable.
    - Link the table with the map, make a name clickable.

Meeting 8

Date: 14/08/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

To discuss:

- Implemented better cypher queries and more GUI options and functionality in ReactJS:
    - Added functionality to the table where names of nodes are clickable and it will update on the map and table.
    - Added another search option.
- Implemented DeckGL via a tab switch option.

Questions:

- ReactJS hosting, any updates from the University IT department?
- Using Ethan's data, unable to change to another set of data since the UI in the ReactJS is built upon that data set and the report is more than halfway complete with 2 weeks left, unable to change dataset.

Discussed:

- How to improve the GUI.

For next week:

- Fix the overlay text issue in the table.
- Add next to the Type column in the table an explanation of the type, UA, ED, place.
- Replace End Node Label with End Node Type text in the column in the table.
- Explain how the ReactJS project can be transformed by adding more features.
- Implement detail panel, make node names clickable.
- Zoom levels of the map, show only 80% relationship, specify bigger places as in regions and smaller places as in Tesco. This only applies when it shows all nodes.

Meeting 9

Date: 21/08/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

To discuss:

- Implemented detailed panel.
- Unable to implement a working zoom level with showing specific nodes.
- Completed the testing by comparing the data in Neo4J to the website made with ReactJS.

Questions:

- Any progress on hosting the ReactJS on university website?
- How will the hosting for Neo4J be done for submission?

Discussed:

- Will host the ReactJS and Neo4J locally, add instructions on how to run the project.

For next week:

- Add to readme.md, send on teams message to check.
- Add backup from neo4j.
- Make a video demonstration of the website, gitlab, neo4j database.

Meeting 10

Date: 28/08/2025

Attendees: Alia Abdelmoty, Georgio Zeilaa

To discuss:

- The report, to receive feedback.

Questions:

- How would you like me to provide the video demonstration? Via link in the report appendix and mention it in the report?

Discussed:

- In literature review, add how kepler.gl is related to the project, add detail in design for reason of what I have learnt from kepler.gl.
- Data modelling, discuss the modelling options with sample data that I have. Why implementing all nodes as nodes and relationships instead of only places being nodes and rest relationships.
- In the implementation, why I chose the options and why not others.
- Add introductions in the chapters.
- Add the revised options on the website which was based on feedback.