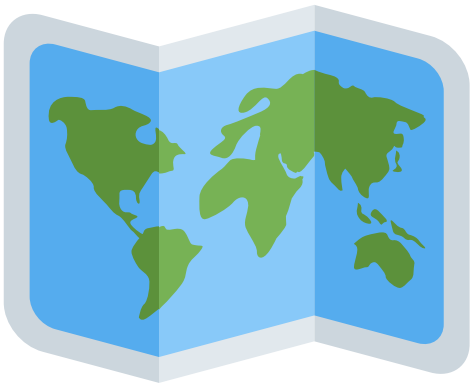


Untitled Mapping Project

Peter Warrington's NEA project

World map emoji icon sourced from the Twemoji project - Copyright 2020 Twitter, Inc and other contributors - licensed under the CC-BY 4.0 license.

Table of Contents

Untitled Mapping Project 1

Analysis 5

The problem 5

Problem research 5

Objectives 7

Prototype 8

Modelling 8

High level project structure 9

Documented design 10

Main data containers 11

Map objects 12

Database layout 13

Path finding pseudocode 13

HCI 17

Technical solution 22

Shared (used by both server and client) 22

src/shared/sharedStructures.js 22

Web client 39

src/web/index/index.js 39

Server 71

src/index.js 71

src/convertFromOSM.js 73

src/logging.js 81

src/dbFromQuery.js 82

src/findRoute.js 87

src/pointSearch.js 91

src/getDBfromFile.js 93

src/getTestDB.js 94

Abandoned – Tile mapping 96

src/MapAreaToImage.js [excerpt] 96

src/shared/sharedStructures.js [excerpt] 98

Testing 99

Testing video 99

Objective 1 (“Web client is able to send requests and receive appropriate data back to/from backend”) 102

Server successfully starts up 102
Web application successfully loads 103
Corresponding JSON data is returned for requests to GET /api/GetDBfromQuery 104

Objective 2 (“The backend is able to read from OpenStreetMap data and return this to clients via this API”) 107
convertFromOSM.js on OSM data for Cambridgeshire 107

Objective 3 (“The web client is then able to display and represent data returned from the server on a map that can be panned and zoomed”) 109
Pan without zoom to map extreme 109
Zoom out to show whole of map 111
Zoom in from initial position 111
Zoom in, pan, zoom out 114
Zoom out, pan, zoom in 116

Objective 4 (“A search function is available on this web client to be able to find locations using the API”) 119
API request to /api//PointSearch, searching for “museum” 119
API request to /api/PointSearch for a specific point ID 120
Searching for “museum” via web app 121
Search for specific point via web app 123

Objective 5 (“A function will be available to find an efficient route from one location on the map to another using path theory. This will take into account distance and this route must be traversable in the real world.”) 125
API request to /api/FindRoute searching between “Cambridge station” and “Cambridge north” 125
Search for route between “Cambridge station” and “Cambridge north” via web app 126

Objective 6 (“Function serviced by database on the backend to allow a user to sign-in/register and save places/routes”) 127

Objective 7a (“Testers reports that they are able to easily find their home using the search function”) 127

Objective 7b (“Testers reports that they are able to easily generate a route”) 127

Objectives 7c and 7d (“Testers report that they are able to easily register and log in, save their location and route to their account”) 128

Evaluation 128

Evaluating against set objectives 128

Possible improvements to project 131

Apendix 132

Analysis 132
Graph theory research from start of project: 132
Planned database/object structure 132
Version of analysis documentation during early stages of project 134
Early project purpose documentation 135

Project development screenshots over time 137
02/07/2021 137
18/11/2022 138
17/02/2022 138
20/02/2022 139
21/02/2022 140
23/02/2022 140
24/02/2022 141
26/02/2022 142
27/02/2022 142
Early testing 143
27/12/2021 143
Code 143
Final project - index.html 143
Final project - Index.css 146
Git commit log 155

Analysis

The problem

For my NEA, I have decided to create a self-hosted mapping solution, accessible from web clients via an API. This is as opposed to commonly used proprietary cloud services that are hosted by third parties, such as Google Maps, which cannot be hosted on-site, and which do not allow organisations and individuals the high level of control a self-hosted system provides.

This may be useful for individuals that have access to such services blocked or censored due to Government influence, where a decentralised self-hosted service allows people to avoid this. For example, mapping services offered in China face strict legislation that mandate these to meet Government conditions, for example to reflect their government's own disputed position on territorial boundaries.[1]

It may also be useful for organisations that want to manage and share their own confidential map data securely with authorised parties.

This service should be fully accessible from a variety of platforms, including desktops, smartphones and tablets in order to provide a mapping service for users when they need them, much like Google Maps. It was for this reason that I decided to create a web application for this service that communicates with the server in order to meet these requirements.

Problem research

In order to begin to understand how a solution to this problem could be approached, I broke down the problem into several questions.

1. What map data can be used in order to build a reliable, real-world map hosting system?

The most well-known Internet mapping service is of course Google Maps, at least in western countries such as the United States.[2] The map data underlying the service however available publicly, unless using Google's API under strict conditions banning external re-use.[3] Data from this service therefore could not be utilised in this project.

I therefore decided to use OpenStreetMap data as this is publicly available under an open source licence[4], as well as being comprehensive and reliable[5], available in machine readable XML format.[6]

---

[1] https://www.reuters.com/article/us-china-lawmaking-idUSKBN17T1S4
[2] https://www.statista.com/statistics/865413/most-popular-us-mapping-apps-ranked-by-audience/
[3] https://cloud.google.com/maps-platform/terms#:~:text=3.2.3%C2%A0Restrictions%20Against%20Misusing%20the%20Services
[4] https://www.openstreetmap.org/copyright
[5] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0180698
[6] https://wiki.openstreetmap.org/w/index.php?title=OSM_XML&oldid=2265532

I considered using Ordnance Survey data for this purpose, which is publicly available under the open-source Open Government License and meets those conditions met above, being very reliable given it is the UK's national mapping agency.[7] I however decided to use OpenStreetMap data as this is globally comprehensive, meaning adding OSM support would allow the importation of existing worldwide map data, not just for the UK.

2. How can map data be plotted on a map?

OSM provides data for streets, bodies of water, and land use, alongside nearly a hundred other map features.[8] As part of my research, I analysed the OSM XML file format and proposed an algorithm to be able to convert this into a custom database format included in my appendix ("OSM Conversion Algorithm").

OSM data which I used in my project is defined in terms of points specifying latitude and longitude, which must be displayed on a map using a projection in order to map the globe onto a rectangular plane like ours. The Mercator projection is one of the simplest of these, being regarded as much of a standard, with calculations to map latitude and longitude onto x and y coordinates being straight-forward.[9]

From a more implementation focused perspective, there are then two main ways for map data to then be displayed on a client. The simplest of these is to return raw map data to the client that is then processed and drawn onto the display accordingly. This can be resource intensive however, especially when drawing lots of map elements to screen, something that can lead to lag. Alternatively, a process of tile-mapping can be used, where a server returns pre-generated images of the map to the client, something that can lead to faster load times. This approach is used by Google maps.[10] As part of my project, I decided it was more feasible to get an implementation which doesn't use map tiling working first as it made map manipulation and map display implementation easier during the development process, leaving open the option of implementing tiles later.

3. How can map data be queried?

OSM provides comprehensive metadata for map elements[11], which can easily be queried and searched by implementing a database.

4. How can routes be plotted using map data?

Plotting routes requires the use of a path-finding algorithm. The main algorithm that I researched was Dijkstra's, as this enables path finding between connecting nodes from a start node to an end node reliably and reasonably efficiently. I considered the use of the A* path search algorithm, which is generally more efficient than Dijkstra's, but this requires the use of a heuristic function which I was unsure how to define. I therefore considered it more feasible to implement a Dijkstra's path finding algorithm within my project time frame

---

[7] https://www.ordnancesurvey.co.uk/business-government/products/open-map-local
[8] https://wiki.openstreetmap.org/wiki/Map_features
[9] https://stackoverflow.com/a/14457180
[10] https://www.e-education.psu.edu/geog585/node/706
[11] https://taginfo.openstreetmap.org/keys

and researched into this towards the start of the project as included in my appendix under "Graph theory research from start of project".

Objectives

During the analysis stage of my project, I outlined the following objectives that I wanted to achieve:

1. Web client is able to send requests and receive appropriate data back to/from backend
2. The backend is able to read from OpenStreetMap (OSM xml) data and return this to clients via this API
3. The web client is then able to display and represent data returned from the server on a map that can be panned and zoomed
4. A search function is available on this web client to be able to find locations using the API
5. A function will be available to find a efficient route from one location on the map to another using path theory. This will take into account distance and this route must be traversable in the real world.
    5. a. **Extension:** Take into account traffic, whether that be based on a custom specified metric or the location of traffic lights along a route
6. Function serviced by database on the backend to allow a user to sign-in/register and save places/routes
7. Testers reports that they are able to easily:
    a. Find their home using the search function
    - Generate a route
    - Register and log-in
    - Save this location and route to their account
8. Data structures and important algorithms are well-documented using comments
9. API is well documented
10. Effective use of object-oriented programming to be able to create effective data structures shared across the project, including the appropriate use of inheritance.
11. Normalised database structure to store the data mentioned above.

Extension:

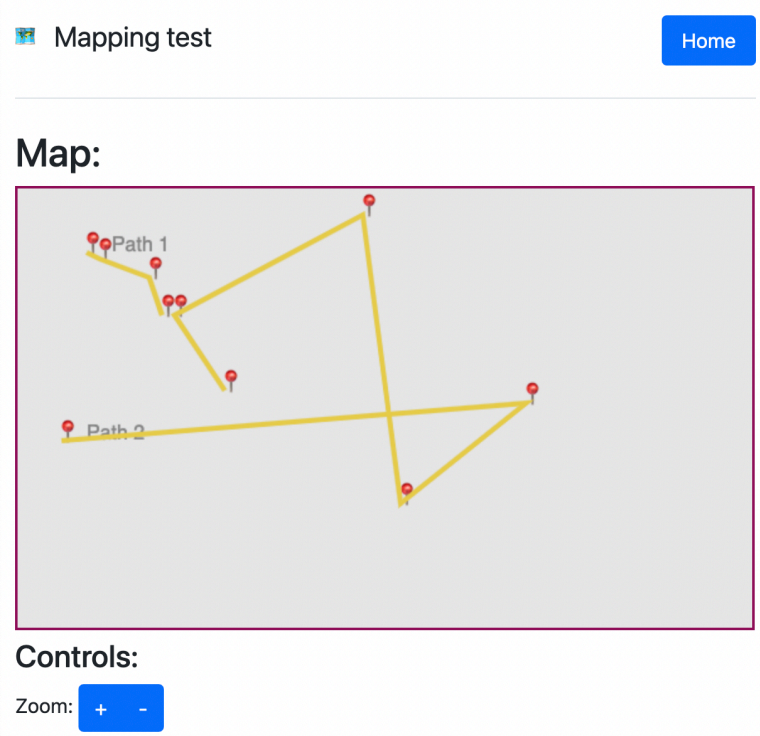
12. Cache the most frequently accessed places directly on the database
13. Function to be able to plot new data on map that is then made available

14. Ability to add notes to the map
15. Ability to add other annotations to the map

Prototype

During my analysis stage, I designed a basic prototype of a web application that can display points and paths locally on a translatable map. The data structures and methods that I created here went on to serve the basis of much of my finished implementation, establishing classes for the displayed canvas, map points and paths. This did not yet make use of a server, but demonstrated that much of the project was achievable, and served as a firm foundation from which I further developed the project.

A screenshot of it is included below, while the JavaScript code for this prototype is included in the appendix (Prototype - src/web/index/index.js).

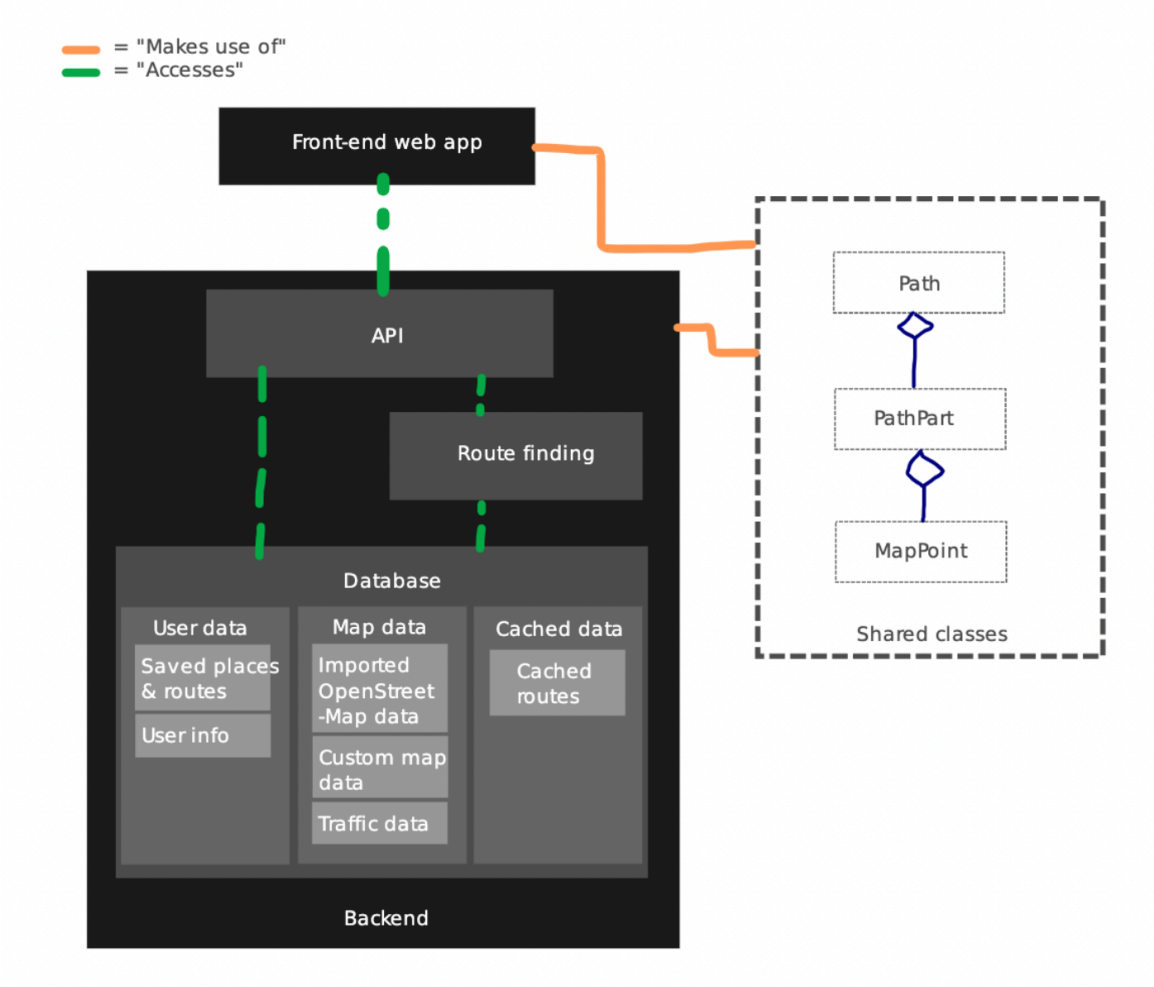


Modelling

During the analysis stage of my project having completed the prototype, I decided to draw up a project structure model, included below, describing the planned components of the project and how I expected them to interact.

High level project structure

This document outlines an initial plan for the main components of my NEA, and how these will interact.



The frontend

The frontend will be a web app built using JavaScript, HTML5 and CSS3 technologies. It will

- Display a map that is scalable and translatable
- Communicate with the backend using an API
- Support a variety of devices such as desktops, tablets and smartphones

It will share many of the data structures that are used on the backend representing map elements and paths. It will require extensive testing to be able to ensure it can reliably perform these functions, where serving as a web app will make user testing easier.

The backend

The backend will be built in JavaScript using the Node.JS framework. At its most basic level, it will:

- Be able to read OpenStreetMap data that will be imported into a database
- Serve this data via an API to the frontend
- Calculate paths and serve this via an API to the frontend

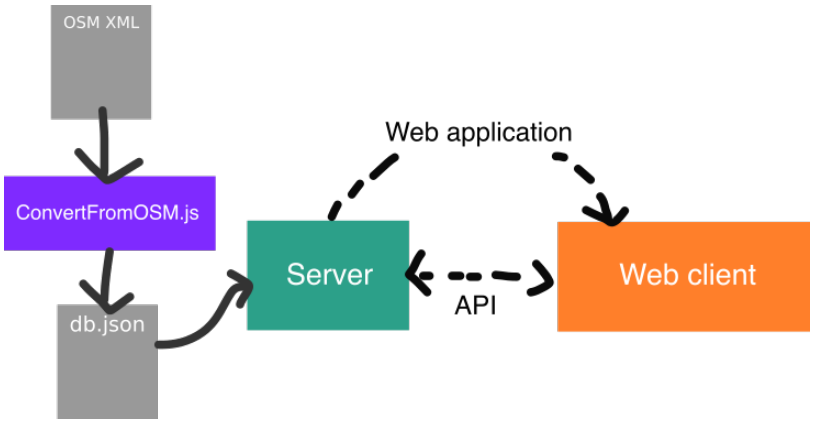
It will therefore require a database and the use of path theory which I will have to research.

Documented design

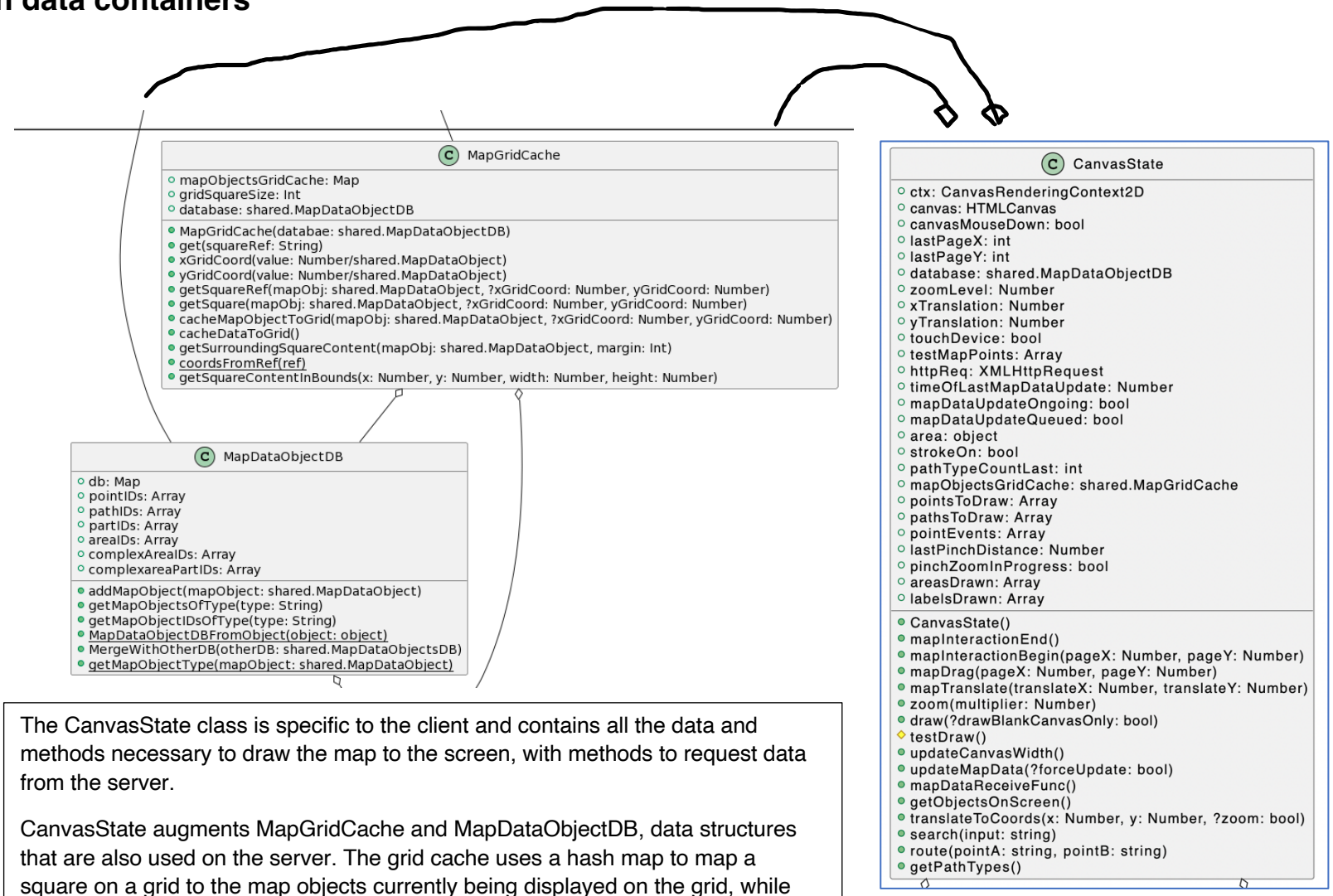
The design of my project is organised into three main components:

1. **convertFromOSM.js**
   This is manually ran in order to convert data from a source OSM XML file into the custom database format (db.json) that the project uses. This can take up to around 5 minutes as it is often a large dataset. Throughout development I have been using OSM XML map data of Cambridgeshire that is imported using this script with success.
   The resultant db.json is in a normalised form, and is significantly faster to process than just the OSM, as it does not have the same XML overhead and can be read natively into a JavaScript object on both the client and server, both of which use JavaScript.
2. **The server**
   This is the largest part of the project and is responsible for operating an API from which the client requests views of the database using queries, as well as conducting searches and route finding. It is additionally responsible for hosting the web application.
3. **The web application**
   This is responsible for requesting and displaying map data to the user, as well as presenting the user interface.

A diagram of these components and how they interact is shown below:



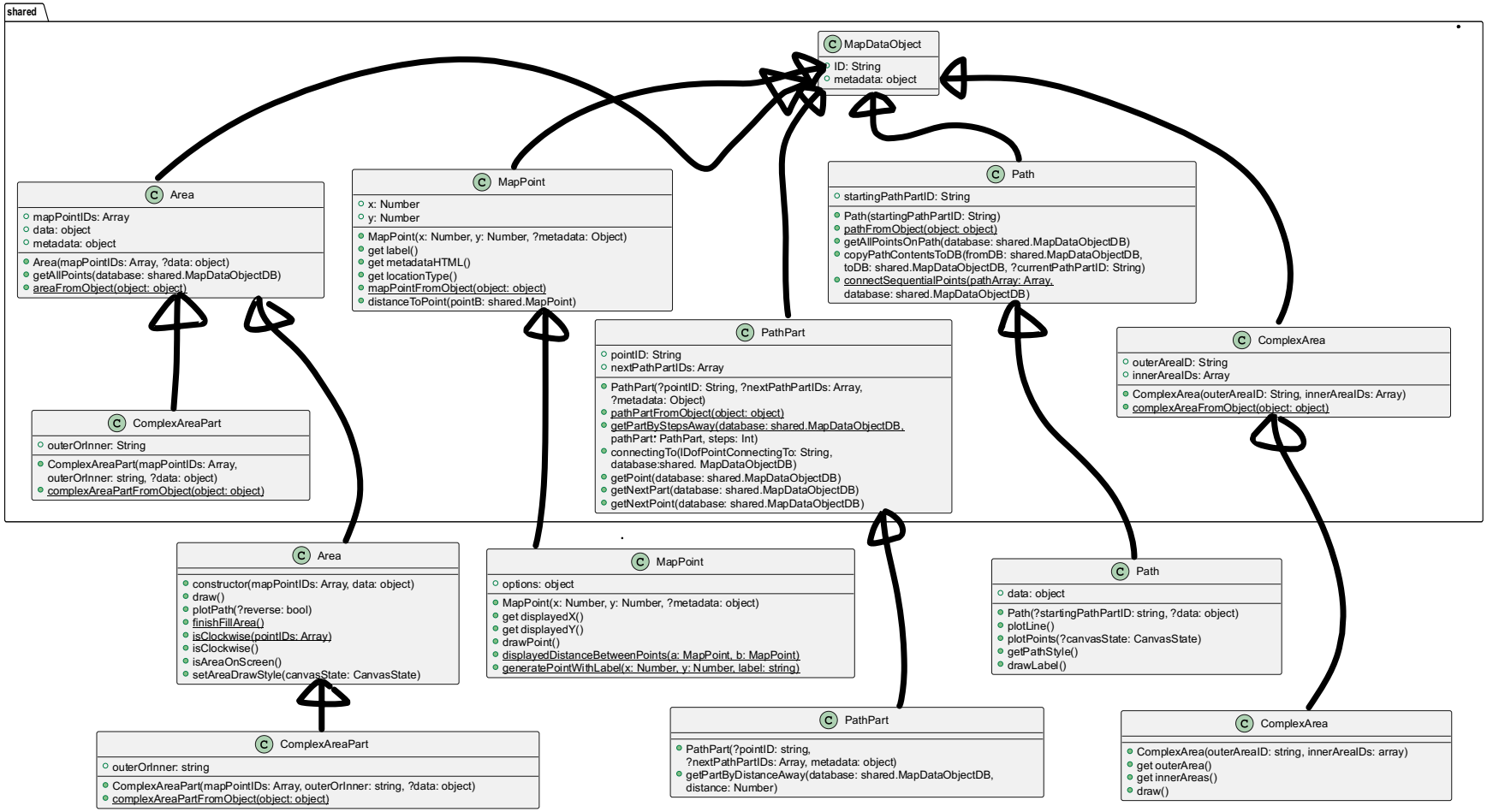
Main data containers



The CanvasState class is specific to the client and contains all the data and methods necessary to draw the map to the screen, with methods to request data from the server.

CanvasState augments MapGridCache and MapDataObjectDB, data structures that are also used on the server. The grid cache uses a hash map to map a square on a grid to the map objects currently being displayed on the grid, while the object DB is a custom database implementation that stores map objects so they can be accessed and queried, on both the server and client.

Map objects



The shared map objects represent the server representation of map elements such as points and paths, with those properties and methods that are required by the server. The client inherits from these shared map objects to add those properties and methods necessary to draw them to screen, with the exception of ComplexAreaPart which has to be fully duplicated from the shared structures unlike the other classes, because otherwise it inherits shared.Area without drawing functions, rather than just Area with drawing functions.

Database layout

The database used by my project is organised into keys, where each key is the unique ID of a map object, and the beginning of each ID indicates the table it is a part of. These are: POINT, PATH, PART, AREA, COMPLEX-AREA, and COMPLEX-AREA-PART. For example, an ID for a specific point might be POINT_123AbC. Where the last characters uniquely identify the point.

This database is in JSON key-value pairs as a file but is stored as a hash-map when loaded into memory both on the server and client.

The database caches the specific IDs that are in use for each map object type, so that all map objects of a specific type can quickly be iterated through, for example when conducting a search.

Map objects reference map objects in other “tables” using their IDs as foreign keys. For example, an Area contains an array of point IDs that make up that area.

Path finding pseudocode

One of the key features of my project is route finding of real world map data. Pseudocode detailing the algorithm used in my implementation is explained below:

```
// This function is called when a API request to find a route between two
points is
// received.
START FUNCTION findRoute(startingPointSearchTerm, destinationPointSearchTerm,
database, gridCache)
    // Conduct searches of the search terms to find their corresponding points
    startingPointID = Search(startingPointSearchTerm, database)
    destinationPointID = Search(destinationPointSearchTerm, database)

    startingPoint = database.get(startingPointID)
    destinationPoint = database.get(destinationPointID)

    // Get the point nearest to found points that is on a highway
    // From which we can conduct routing. We limit the types of highways
    // that we can route using to decrease nodes and therefore
    // decrease processing time into reasonable bounds
    startingPoint = nearestHighwayPoint(startingPoint, database, gridCache)
    destinationPoint = nearestHighwayPoint(destinationPoint, database,
gridCache)

    // Begin dijkstra's now that we have our starting and destination points
finalised
    // Define sets
    visitedNodes = new HashMap()
    unvisitedNodes = new HashMap()
```

```
    // Set distance of starting point to 0
    startingPoint.distance = 0

    allDatabasePaths =  database.getMapObjectsOfType("PATH")
    FOR EACH path IN allDatabasePaths
        IF isAcceptedHighway(path)
            FOR EACH point IN path
                // Set default properties for all nodes at start of dijkstra
                point.distance = Infinity
                point.visited = false
                point.parentNode = None
                unvisitedNodes[point.ID] = point
            END FOR
        END IF
    END FOR

    currentNode = startingPoint

    WHILE destinationPoint.visited == false DO
        // Get same points in same 10x10 square as currentNode
        neighbours = []
        neighbours += gridCache.get(currentNode.x, currentNode.y)
        neighbours +=
getPathContainingPoint(currentNode).getPointAfter(currentNode)
        neighbours +=
getPathContainingPoint(currentNode).getPointBefore(currentNode)

        // If all neighbours have Infinite distance, a route has not been
        // successfully found
        IF EVERY neighbour IN neighbours HAS neighbour.distance == Infinity
            return []
        END IF

        FOR EACH neighbour IN neighbours
            // Calculate the distance to each neighbour, changing its recorded
distance if this
            // calculated value is less than that recorded
            tentativeDistance = currentNode.distance +
distanceToPoint(neighbour)
            IF tentativeDistance < neighbour.distance THEN
                neighbour.distance = tentativeDistance
            END IF
            // Set current node as the parent of the neighbour to allow for
            // back-tracking
            IF neighbour.ID != currentNode.ID THEN
                neighbour.parentID = currentNode.ID
            END IF
        END FOR
```

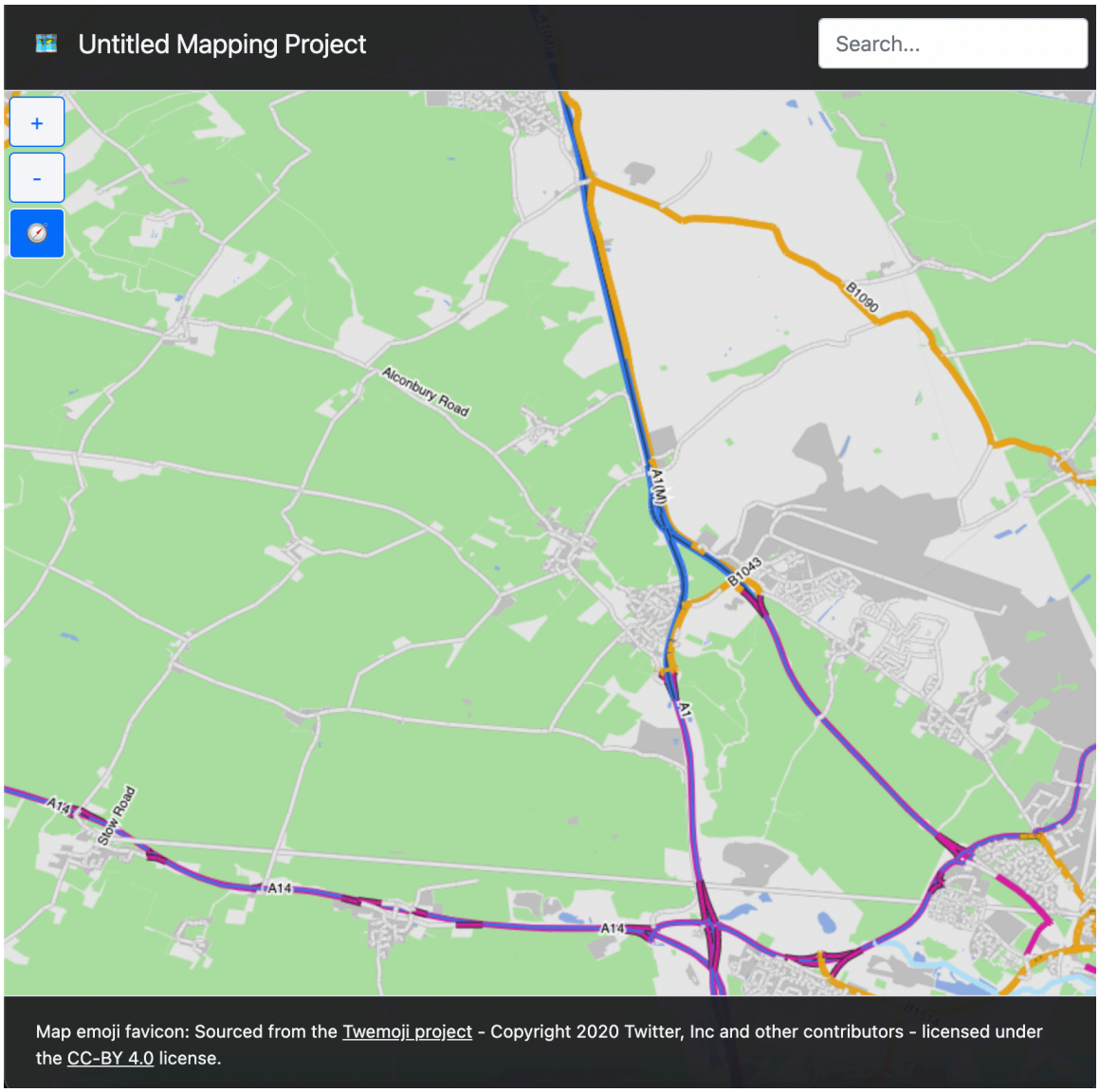
```
        // We have now examined all neighbours of this node, mark it as visited
        currentNode.visited = true
        // Remove from unvisitedNodes so we don't visit it again
        unvisitedNodes.delete(currentNode.ID)
        // Add to visited nodes so that we can back track later to find route
        visitedNodes[currentNode.ID] = currentNode

        // Set next current node to unvisited node with smallest distance
        nextNode = EMPTY
        FOR EACH node IN unvisitedNodes
            IF nextNode == EMPTY OR node.distance < nextNode.distance THEN
                nextNode = node
            END IF
        END FOR

        currentNode = nextNode
    END WHILE

    // The algorithm should have now found a route, now we back-track in order
    // to record this route
    currentNode = destinationPoint
    fastestPath = []
    WHILE currentNode != startingPoint
        fastestPath += currentNode
        currentNode = visitedNodes[currentNode.parentID]
    END WHILE

    return fastestPath.reverse()
END FUNCTION

START FUNCTION Search(searchTerm, database)
    searchResults = [];
    // Iterate through search index, adding a score for each map object
    // depending on its calculated relevance to search term.
    FOR EACH mapObject IN this.searchIndex
        searchConfidence = 0
        // If the search term matches the exact ID of the object
        // Rise this to the top of the results with a high
        // search confidence
        IF (mapObject.ID == searchTerm)
            searchConfidence += 20;
        ELSE
            // Otherwise, score by the amount of times the term appears in the
metadata
            searchConfidence += OCCURRENCES OF searchTerm IN apObject.metadata;
        END IF
```

```
        // If the map object explicitly defines a place such as a town and
        // It already matches the search term, increase its score so
        // it shows up higher. This is because if someone searches for
        // something like "Cambridge", we want to return the City, not
        // some random location 15 miles away that happens to have
        // "Cambridge" in its name.
        IF (mapObject.metadata DESCRIBES AS "place" AND searchConfidence > 0)
            searchConfidence += 15
        END IF

        ADD [mapObject, searchConfidence] to searchResults
    END FOR

    SORT searchResults by searchConfidence DESCENDING // Show highest scoring
results first

    result = FIRST entry IN searchResults WHERE entry.mapObject OF TYPE "POINT"
|| "PATH"

    IF result == EMPTY
        THROW ERROR
    ELSE
        IF result.mapObject OF TYPE "PATH"
            result.mapObject = getPathMidpoint(result.mapObject)
        END IF

        RETURN result.mapObject.ID
    END IF
END FUNCTION
```

HCI

This section will demonstrate how a user interacts with the application, by using screenshots and explanations.

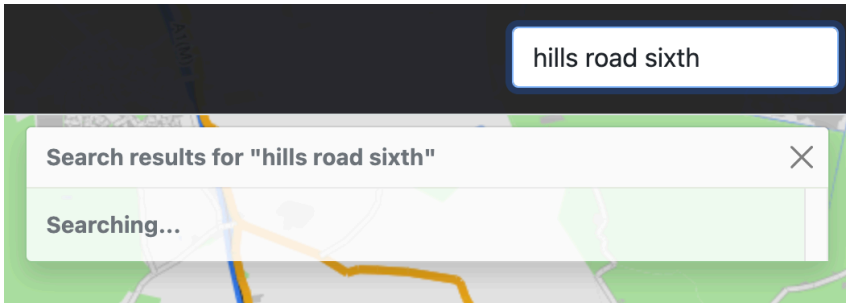
When the user first loads the website, the website will make its first request to the server for map data, which will then be displayed on screen, as shown below:



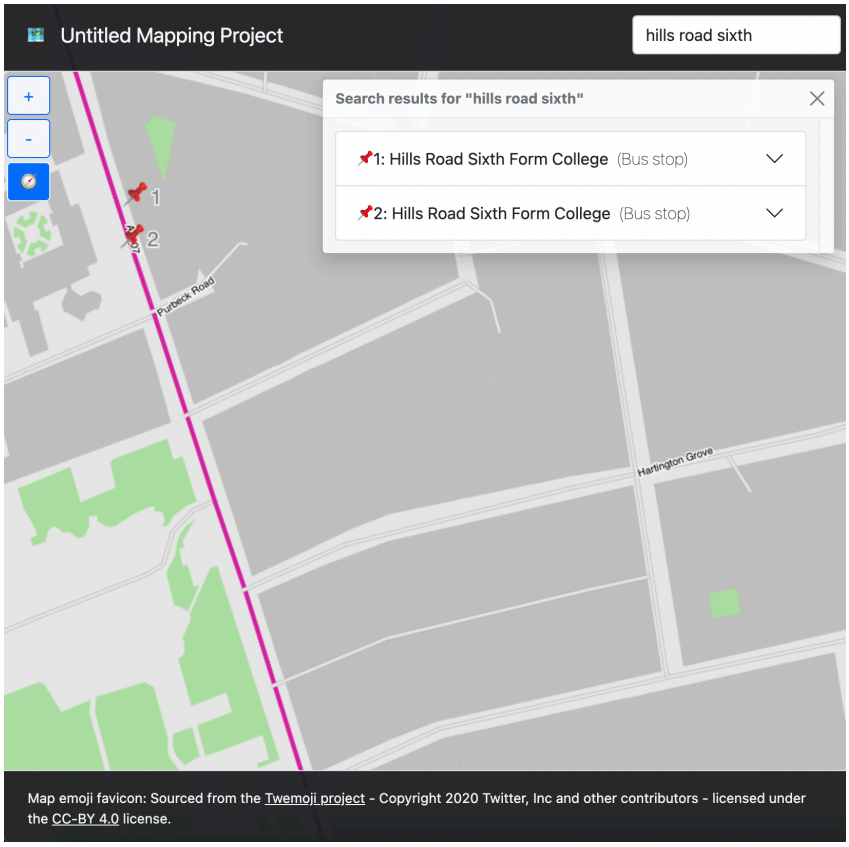
The user is free to pan and zoom this map using the +/- buttons to the left of the screen.

As the user moves about the map, new map data for this view is progressively loaded in from the server.

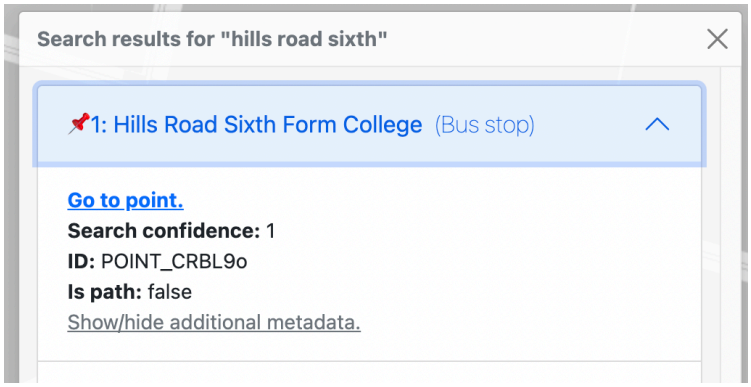
The user can use the search box at the top right of the screen to conduct a search of points and paths in the database, simply typing their search term and pressing enter, which will cause a draggable dialog box to appear, like that shown below:



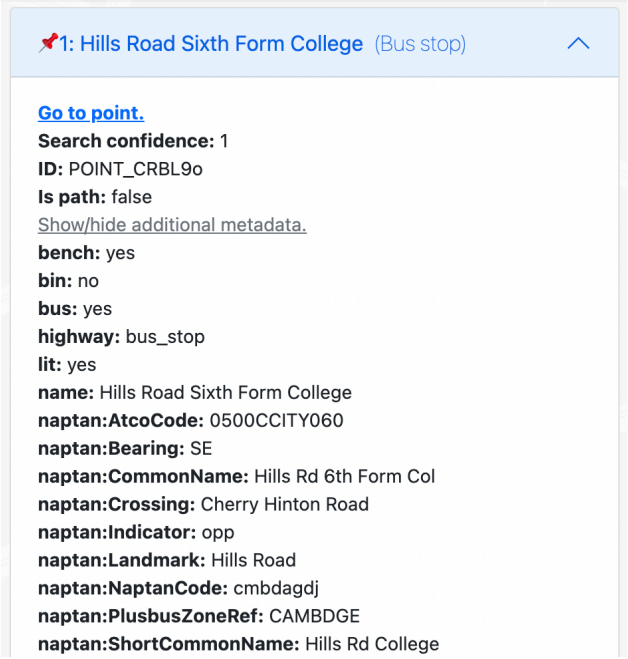
Results can then be displayed in the same dialog, translating the map to a view showing all the results:



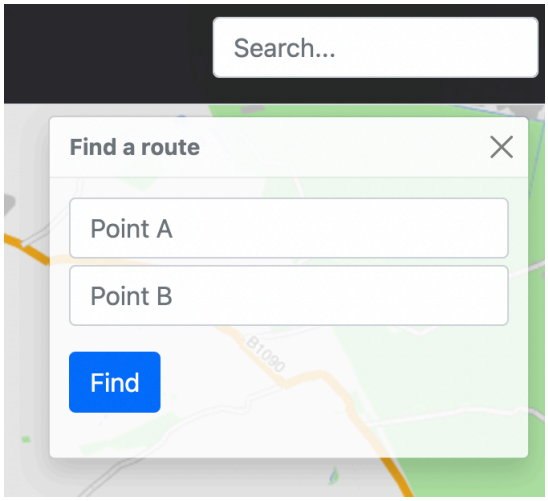
These results can then be selected to view additional information and translate the map to a specific point.



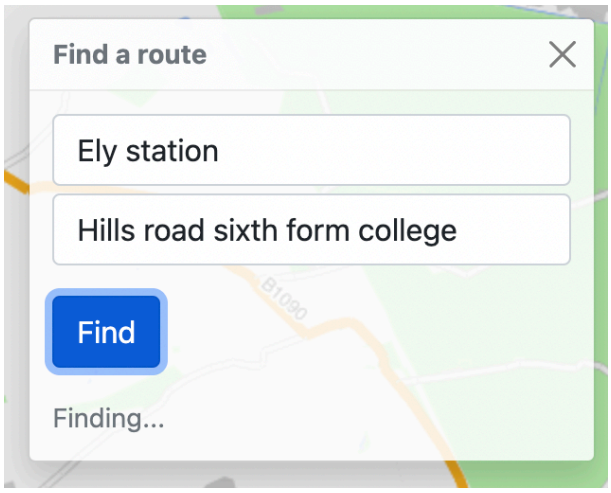
The “Show/hide additional metadata” button can be selected to show additional metadata, sourced from OSM.



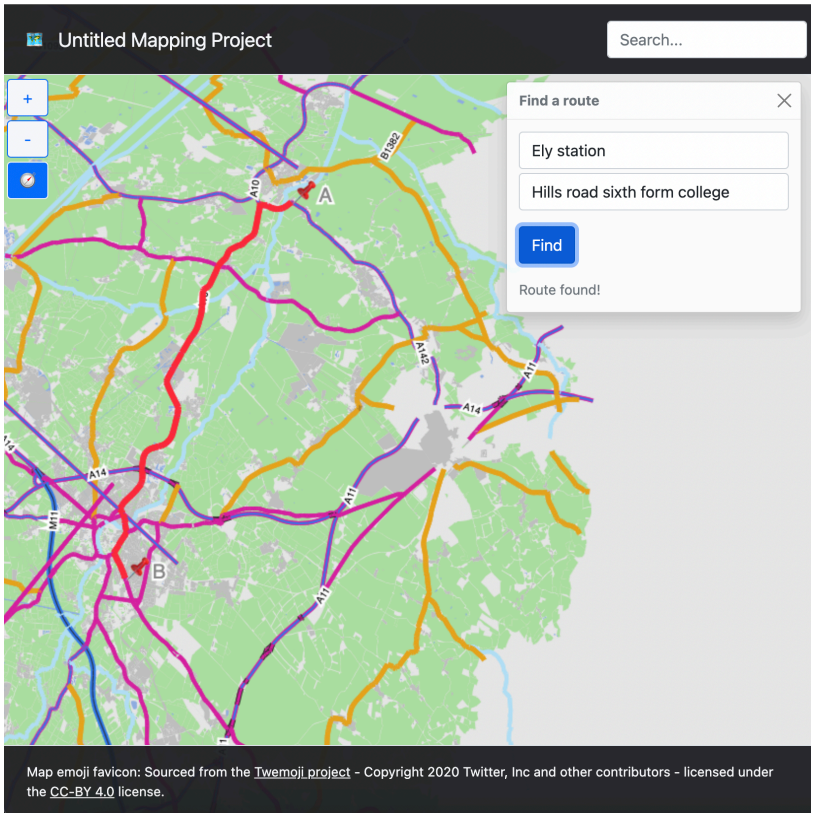
The compass button on the left of the display is used to show a dialog which allows the user to find a route between two points.



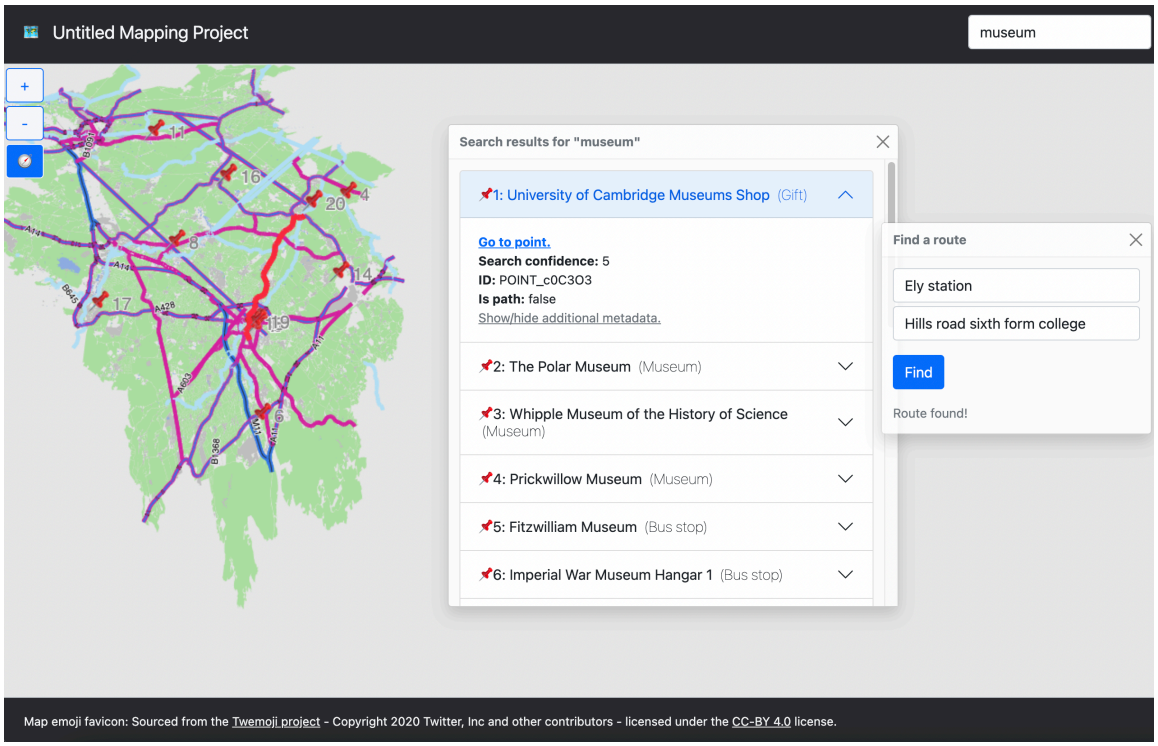
With search terms entered in both text boxes, the find button can be pressed and the status of the operation is displayed, while the server processes the request.



A route should then be displayed on the map.



Both the search and route-find draggable dialogs can be displayed at the same time.



Technical solution

Shared (used by both server and client)

src/shared/sharedStructures.js

```
shared = {};

// If running on backend, make accessible
try {
    module.exports.shared = shared;
} catch {}

// Readable debug_on setting across both client and server
shared.debug_on = true;

/** A database model used to containerise map data using an ID system
 * of the format [Type]_[UID]
 * Contained map objects are accessed using database.db[id],
 * map objects are added using database.addMapObject(mapObject).
 */
shared.MapDataObjectDB = class MapDataObjectDB {
    /** Object, where key is the ID of the MapObject */
    db = new Map();
    /** Caches point IDs */
    pointIDs = []
    /** Caches path IDs */
    pathIDs = []
    /** Caches part IDs */
    partIDs = []
    /** Caches area IDs */
    areaIDs = []
    /** Caches complex area IDs */
    complexAreaIDs = []
    /** Caches complex area part IDs */
    complexAreaPartIDs = []

    /**
     * Adds a map object to the database, generating a random ID.
     * @param {MapDataObject} mapObject Map object to add
     */
    addMapObject(mapObject) {
        let ID = "";

        if (mapObject.ID != null) {
            ID = mapObject.ID;
        } else {
```

```
            ID += shared.MapDataObjectDB.getMapObjectType(mapObject);

            ID += "_";

            // Generate random characters for ID
https://stackoverflow.com/a/1349426
            var characters       =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
            var charactersLength = characters.length;
            for ( var i = 0; i < 6; i++ ) {
                ID += characters.charAt(Math.floor(Math.random() *
charactersLength));
            }
        }

        // Add to DB
        mapObject.ID = ID;
        this.db.set(ID, mapObject);

        // Cache ID
        if (ID.indexOf("POINT_") == 0) this.pointIDs.push(ID);
        if (ID.indexOf("PART_") == 0) this.partIDs.push(ID);
        if (ID.indexOf("PATH_") == 0) this.pathIDs.push(ID);
        if (ID.indexOf("AREA_") == 0) this.areaIDs.push(ID);
        if (ID.indexOf("COMPLEX-AREA_") == 0) this.complexAreaIDs.push(ID);
        if (ID.indexOf("COMPLEX-AREA-PART_") == 0)
this.complexAreaPartIDs.push(ID);

        return mapObject;
    }

    getMapObjectsOfType(type) {
        let mapObjects = [];
        let objectIDs = this.getMapObjectIDsOfType(type);
        objectIDs.forEach((objectID) =>
mapObjects.push(this.db.get(objectID)));

        return mapObjects;
    }

    /**
     * Gets map object IDs of ID matching type.
     * @param {array or string} type
     * @returns
     */
    getMapObjectIDsOfType(type) {
        if (typeof type == "string") type = [type];
```

```
        let ids = [];
        type.forEach(type => {
            switch (type) {
                case "POINT":
                    ids = ids.concat(this.pointIDs);
                    break;
                case "PATH":
                    ids = ids.concat(this.pathIDs);
                    break;
                case "PART":
                    ids = ids.concat(this.partIDs);
                    break;
                case "AREA":
                    ids = ids.concat(this.areaIDs);
                    break;
                case "COMPLEX-AREA":
                    ids = ids.concat(this.complexAreaIDs);
                    break;
                case "COMPLEX-AREA-PART":
                    ids = ids.concat(this.complexAreaPartIDs);
                    break;
                default:
                    ids = ids.concat(this.db.keys().filter(id =>
id.indexOf(type + "_") == 0)); // Slow fallback
                    break;
            }
        });

        return ids;
    }

    /**
     * Converts a un-instanciated object into a database.
     * @param {*} object Un-instanciated object detailing a database
     * @returns {MapDataObjectDB}
     */
    static MapDataObjectDBFromObject(object) {
        var database = new shared.MapDataObjectDB();

        var db = object.db;
        var pointIDs = Object.keys(db).filter(id => id.indexOf("POINT_") == 0);
        var pathPartIDs = Object.keys(db).filter(id => id.indexOf("PART_") ==
0);
        var pathIDs = Object.keys(db).filter(id => id.indexOf("PATH_") == 0);
        var areaIDs = Object.keys(db).filter(id => id.indexOf("AREA_") == 0);
        var complexAreaIDs = Object.keys(db).filter(id => id.indexOf("COMPLEX-
AREA_") == 0);
```

```
        var complexAreaPartIDs = Object.keys(db).filter(id =>
id.indexOf("COMPLEX-AREA-PART_") == 0);

        pointIDs.forEach(pointID => {
            let point = shared.MapPoint.mapPointFromObject(db[pointID]);
            point.options = new shared.MapPoint().options; // Strip options,
replace with default
            database.addMapObject(point);
        });

        pathPartIDs.forEach(pathPartID => {
            let pathPart = shared.PathPart.pathPartFromObject(db[pathPartID]);
            database.addMapObject(pathPart);
        });

        pathIDs.forEach(pathID => {
            let path = shared.Path.pathFromObject(db[pathID]);
            database.addMapObject(path);
        });

        areaIDs.forEach(areaID => {
            let area = shared.Area.areaFromObject(db[areaID]);
            database.addMapObject(area);
        });

        complexAreaPartIDs.forEach(complexAreaPartID => {
            let complexAreaPart =
shared.ComplexAreaPart.complexAreaPartFromObject(db[complexAreaPartID]);
            database.addMapObject(complexAreaPart);
        });

        complexAreaIDs.forEach(complexAreaID => {
            let complexArea =
shared.ComplexArea.complexAreaFromObject(db[complexAreaID]);
            database.addMapObject(complexArea);
        })

        return database;
    }

    /**
     * Copy another database's items into this db.
     * @param {MapDataObjectDB} otherDB
     */
    mergeWithOtherDB(otherDB) {
        let otherDBitems = Array.from(otherDB.db.values());

        for (let i = 0; i < otherDBitems.length; i++) {
```

```
            const item = otherDBitems[i];

            if (this.db.get(item.ID) == undefined)
                this.addMapObject(item);
        }
    }

    static getMapObjectType(mapObject) {
        if (mapObject instanceof shared.MapPoint)
            return "POINT";
        else if (mapObject instanceof shared.PathPart)
            return "PART";
        else if (mapObject instanceof shared.Path)
            return "PATH";
        else if (mapObject instanceof shared.ComplexAreaPart)
            return "COMPLEX-AREA-PART"
        else if (mapObject instanceof shared.Area)
            return "AREA";
        else if (mapObject instanceof shared.ComplexArea)
            return "COMPLEX-AREA";
        else
            return "GENERIC";
    }
}

shared.MapGridCache = class MapGridCache {
    /** Maps objects onto a grid made of 10x10 squares so can be queried more
quickly */
    mapObjectsGridCache = new Map();
    /** Grid square size */
    gridSquareSize = 10;
    /** Pointer to database from which to cache map data */
    database

    constructor(database) {
        this.database = database;
    }

    get(squareRef) {
        return this.mapObjectsGridCache.get(squareRef);
    }

    xGridCoord(value) {
        let x;
        if (value.x != undefined) x = value.x;
        else x = value;

        return Math.floor(x/this.gridSquareSize) * this.gridSquareSize;
```

```
    }

    yGridCoord(value) {
        let y;
        if (value.y != undefined) y = value.y;
        else y = value;

        return Math.floor(y/this.gridSquareSize) * this.gridSquareSize;
    }

    getSquareRef(mapObj, xGridCoord=this.xGridCoord(mapObj),
yGridCoord=this.yGridCoord(mapObj)) {
        return `${xGridCoord}x${yGridCoord}`;
    }

    getSquare(mapObj, xGridCoord=this.xGridCoord(mapObj),
yGridCoord=this.yGridCoord(mapObj)) {
        return this.get(this.getSquareRef(mapObj, xGridCoord, yGridCoord));
    }

    cacheMapObjectToGrid(mapObject, xGridCoord=this.xGridCoord(mapObject),
yGridCoord=this.yGridCoord(mapObject)) {
        let square = this.get(this.getSquareRef(mapObject, xGridCoord,
yGridCoord));

        if (square == undefined)
            square = [];

        if (square.find(mapObjId => mapObjId == mapObject.ID) == undefined)
            square.push(mapObject.ID);

        this.mapObjectsGridCache.set(this.getSquareRef(mapObject, xGridCoord,
yGridCoord), square);
    }

    /**
     * Caches data to a hashmap grid, so that the queries for
     * which points are on screen can be conducted faster.
     * (JS Objects are typically implemented as hashmaps, but
     * aren't explicitly referred to as such.)
     */
    cacheDataToGrid() {
        this.mapObjectsGridCache.clear();

        let points = this.database.getMapObjectsOfType("POINT");
        for (let i = 0; i < points.length; i++) {
            const point = points[i];
            this.cacheMapObjectToGrid(point);
```

```
        }

        let pathParts = this.database.getMapObjectsOfType("PART");
        for (let i = 0; i < pathParts.length; i++) {
            const part = pathParts[i];
            this.cacheMapObjectToGrid(part,
                this.xGridCoord(part.getPoint(this.database)),
                this.yGridCoord(part.getPoint(this.database)));
        }

        let paths = this.database.getMapObjectsOfType("PATH");
        for (let i = 0; i < paths.length; i++) {
            const path = paths[i];
            path.getAllPointsOnPath(this.database).forEach(point => {
                this.cacheMapObjectToGrid(path, this.xGridCoord(point),
this.yGridCoord(point));
            });
        }

        let areas = this.database.getMapObjectsOfType(["AREA", "COMPLEX-AREA-
PART"]);
        for (let i = 0; i < areas.length; i++) {
            const area = areas[i];
            area.getAllPoints(this.database).forEach(point => {
                this.cacheMapObjectToGrid(area, this.xGridCoord(point),
this.yGridCoord(point));
            })
        }

        let complexAreas = this.database.getMapObjectsOfType("COMPLEX-AREA");
        for (let i = 0; i < complexAreas.length; i++) {
            const complexArea = complexAreas[i];
            let areas = complexArea.innerAreaIDs.map(id =>
this.database.db.get(id));
            areas.push(this.database.db.get(complexArea.outerAreaID));

            for (let j = 0; j < areas.length; j++) {
                const area = areas[j];
                if (area != undefined)
                area.getAllPoints(this.database).forEach(point => {
                    this.cacheMapObjectToGrid(complexArea,
this.xGridCoord(point), this.yGridCoord(point));
                })
            }
        }
    }

    getSurroundingSquareContent(mapObj, margin) {
```

```
        let xGridCoord = this.xGridCoord(mapObj);
        let yGridCoord = this.yGridCoord(mapObj);

        let returnSquare = [];

        let squares = [this.getSquare(mapObj, xGridCoord, yGridCoord)];
        let currentDistanceFromRootSquare = 1;
        while (currentDistanceFromRootSquare <= margin) {
            let offset = this.gridSquareSize * currentDistanceFromRootSquare;
            squares = squares.concat([
                this.getSquare(mapObj, xGridCoord, yGridCoord),
                this.getSquare(mapObj, xGridCoord, yGridCoord - offset),
                this.getSquare(mapObj, xGridCoord, yGridCoord + offset),
                this.getSquare(mapObj, xGridCoord - offset, yGridCoord),
                this.getSquare(mapObj, xGridCoord - offset, yGridCoord -
offset),
                this.getSquare(mapObj, xGridCoord - offset, yGridCoord +
offset),
                this.getSquare(mapObj, xGridCoord + offset, yGridCoord),
                this.getSquare(mapObj, xGridCoord + offset, yGridCoord -
offset),
                this.getSquare(mapObj, xGridCoord + offset, yGridCoord +
offset)
            ]);
            currentDistanceFromRootSquare++;
        }

        squares.forEach(square =>
            returnSquare = returnSquare.concat(square)
        );

        return returnSquare.filter(item => item != undefined);
    }

    static coordsFromRef(ref) {
        let xGridCoordString = ref.slice(0, ref.indexOf('x'));
        let yGridCoordString = ref.slice(ref.indexOf('x') + 1);
        let x = Number(xGridCoordString);
        let y = Number(yGridCoordString);

        return {x, y}
    }

    getSquareContentInBounds(x, y, width, height) {
        let currentCoords = {};
        let squareContent = [];
        currentCoords.y = this.yGridCoord(y);
        while (currentCoords.y < y + height) {
```

```
            currentCoords.x = this.xGridCoord(x);
            while (currentCoords.x < x + width) {
                let square = this.get(`${currentCoords.x}x${currentCoords.y}`);
                if (square != undefined)
                    square.forEach(id => squareContent.push(id));
                currentCoords.x += this.gridSquareSize;
            }
            currentCoords.y += this.gridSquareSize;
        }
        return squareContent;
    }
}

shared.MapDataObject = class MapDataObject {
    /** String for the ID of the data object */
    ID = null;
    /** Additional metadata, e.g. place name */
    metadata = {osm: {}}
}

shared.MapPoint = class MapPoint extends shared.MapDataObject {
    /** Fixed x position of point in relation to others */
    x
    /** Fixed y position of point in relation to others */
    y

    /**
     * Creates a point that can form part of a path and be displayed on a
canvas.
     * @param {int} x Fixed x position of point in relation to others
     * @param {int} y Fixed y position of point in relation to others
     * @param {object} metadata Optional metadata such as name
     */
    constructor (x, y, metadata={}) {
        super();

        this.x = x;
        this.y = y;

        this.metadata = metadata;
    }

    get label() {
        let metadata = this.metadata;
        if (metadata.path != undefined && metadata.path.osm != undefined)
metadata = metadata.path.osm;

        let label = `${this.ID}`;
```

```
        if (metadata.wikipedia != undefined &&
metadata.wikipedia.indexOf("en:") == 0)
            label =
metadata.wikipedia.slice(metadata.wikipedia.indexOf("en:")+3);
        else if (metadata.name != undefined)
            label = metadata.name;
        else if (metadata["addr:street"] != undefined &&
metadata["addr:housenumber"] != undefined)
            label = `${metadata["addr:housenumber"]}
${metadata["addr:street"]}`;
        else if (metadata.prow_ref != undefined)
            label = metadata.prow_ref;
        else if (metadata.ref != undefined)
            label = metadata.ref;

        return encodeHTML(label);
    }

    get metadataHTML() {
        let html = "";
        let metadata = this.metadata;
        if (metadata.path != undefined && metadata.path.osm != undefined)
metadata = metadata.path.osm;

        Object.keys(metadata).forEach(key => {
            html += `<strong>${encodeHTML(key)}:</strong>
${encodeHTML(metadata[key])}<br/>`;
        })

        return html;
    }

    get locationType() {
        let metadata = this.metadata;
        let type = "Location";
        if (metadata.path != undefined && metadata.path.osm != undefined)
metadata = metadata.path.osm;

        if (metadata.highway != undefined && this.metadata.path != undefined)
            type = `Road: ${metadata.highway}`;
        else if (metadata.highway != undefined && this.metadata.path ==
undefined)
            type = `${metadata.highway}`;
        else if (metadata.waterway != undefined)
            type = `Waterway: ${metadata.waterway}`
        else if (metadata.railway == "site")
            type = `Rail station`;
        else if (metadata.tourism != undefined)
```

```
            type = `${metadata.tourism}`;
        else if (metadata.leisure != undefined)
            type = `${metadata.leisure}`;
        else if (metadata.place != undefined)
            type = `${metadata.place}`;
        else if (metadata.shop != undefined)
            type = `${metadata.shop}`;
        else if (metadata.amenity != undefined)
            type = `${metadata.amenity}`;
        else if (metadata.public_transport != undefined)
            type = `Transport ${metadata.public_transport}`;
        else if (metadata.barrier)
            type = `${metadata.barrier}`;
        else if (metadata["addr:housenumber"] != undefined)
            type = `House`

        return (type[0].toUpperCase() + type.slice(1)).replaceAll("_", " ");
    }

    /**
     * Converts a simple object representing a MapPoint (such as that returned
from an API)
     * to a MapPoint.
     * @param {Object} object the MapPoint represented as a simple object to
convert
     * @returns MapPoint
     */
    static mapPointFromObject(object) {
        var mapPoint = new shared.MapPoint(object.x, object.y,
object.metadata);
        mapPoint.ID = object.ID;
        mapPoint.metadata = object.metadata;

        return mapPoint;
    }

    distanceToPoint(pointB) {
        let pointA = this;
        let distance = Math.sqrt( Math.abs(pointB.x - pointA.x)**2 +
Math.abs(pointB.y - pointA.y)**2);
        return distance;
    }
}

/**
 * Defines a "edge" between two nodes
 */
shared.PathPart = class PathPart extends shared.MapDataObject {
```

```
    /** The id of the {Point} referenced by this path part */
    pointID
    /** The IDs of the path parts this connects to */
    nextPathPartIDs = []

    /**
     * @param {string} pointID The ID of the point referenced by this path part
     * @param {string[]} nextPathPartIDs Array of next part IDs in the path
     */
    constructor (pointID=null, nextPathPartIDs=[], metadata={}) {
        super();

        this.pointID = pointID;
        this.nextPathPartIDs = nextPathPartIDs;
        this.metadata = metadata
    }

    /**
     * Converts a simple object representing a PathPart (such as that returned
from an API)
     * to a PathPart.
     * @param {Object} object the path part represented as a simple object to
convert
     * @returns PathPart
     */
    static pathPartFromObject (object) {
        var pathPart = new shared.PathPart(object.pointID,
object.nextPathPartIDs, object.metadata);
        pathPart.ID = object.ID;
        pathPart.metadata = object.metadata;
        return pathPart;
    }

    static getPartByStepsAway(database, pathPart, steps) {
        // Base case
        if (steps == 0 || pathPart.nextPathPartIDs.length == 0)
            return pathPart;
        else {
            var nextPathPart = database.db.get(pathPart.nextPathPartIDs[0]);
            return this.getPartByStepsAway(database, nextPathPart, steps-1);
        }
    }

    connectingTo(IDofPointConnectingTo, database) {
        var connectingPathPart = new PathPart(IDofPointConnectingTo);
        var pathPart = database.addMapObject(IDofPointConnectingTo);
        this.nextPathPartIDs.push(pathPart.ID);
```

```
        return pathPart;
    }

    getPoint(database) {
        return database.db.get(this.pointID);
    }

    getNextPart(database) {
        if (this.nextPathPartIDs.length == 0) return false;
        return database.db.get(this.nextPathPartIDs[0]);
    }

    getNextPoint(database) {
        let nextPathPart = this.getNextPart(database);
        if (!nextPathPart) return false;
        return nextPathPart.getPoint(database);
    }
}

shared.Path = class Path extends shared.MapDataObject {
    /** The {PathPart} object ID that begins the path */
    startingPathPartID

    /**
     * Creates a path using a starting points
     * @param {string} startingPathPartID The ID of {PathPart} object that
begins the path
     * @param {object} data Data, including options for the path when drawing
to screen
     */
    constructor (startingPathPartID) {
        super();

        this.startingPathPartID = startingPathPartID;
    }

    /**
     * Converts a simple object representing a Path (such as that returned from
an API)
     * to a Path.
     * @param {Object} object the path represented as a simple object to
convert
     * @returns Path
     */
    static pathFromObject(object) {
        var path = new shared.Path(object.startingPathPartID, object.data);
        path.ID = object.ID;
        path.metadata = object.metadata;
```

```
        return path;
    }

    /**
     * Converts a tree of connecting points to a array of all points (for
drawing individual points unconnectedly).
     * @returns {MapPoint[]}
     */
    getAllPointsOnPath(database) {
        let pointArray = [];
        let currentPathPartID = this.startingPathPartID;
        while (database.db.get(currentPathPartID) != undefined) {

pointArray.push(database.db.get(currentPathPartID).getPoint(database));
            currentPathPartID =
database.db.get(currentPathPartID).nextPathPartIDs[0];
        }
        return pointArray;
    }

    copyPathContentsToDB(fromDB, toDB,
currentPathPartID=this.startingPathPartID) {
        var currentPathPart = fromDB.db.get(currentPathPartID);

        toDB.addMapObject(currentPathPart);

toDB.addMapObject(fromDB.db.get(fromDB.db.get(currentPathPartID).pointID));

        if (currentPathPart.nextPathPartIDs != null &&
currentPathPart.nextPathPartIDs.length != 0)
            for (var i=0; i < currentPathPart.nextPathPartIDs.length; i++)
                this.copyPathContentsToDB(fromDB, toDB,
currentPathPartID=currentPathPart.nextPathPartIDs[i]);
    }

    /**
     * Converts a sequential array of points to a path
     * @param {MapPoint[]} pathArray A sequential array of {MapPoint}s
     * @returns {Path} {Path} of connecting points
     */
    static connectSequentialPoints(pathArray, database) {
        var pathIdArray = [];

        for (let i = 0; i < pathArray.length; i++) {
            const point = pathArray[i];
            database.addMapObject(point);
            pathIdArray.push(point.ID);
```

```
        }

        // Set up path parts
        var startingPathPart;
        var previousPathPart;

        for (let i = 0; i < pathArray.length; i++) {
            var currentPathPart = new shared.PathPart(pathIdArray[i]);

            database.addMapObject(currentPathPart);

            if (i == 0)
                startingPathPart = currentPathPart;
            if (previousPathPart != undefined)
                previousPathPart.nextPathPartIDs = [currentPathPart.ID];

            previousPathPart = currentPathPart;
        }

        var newPath = new shared.Path(startingPathPart.ID);
        return newPath;
    }
}

/**
 * Used to define polygons to drawn on map, such as bodies of water, and
 * buildings.
 */
shared.Area = class Area extends shared.MapDataObject {
    /** Sequential list of map point IDs that make up area */
    mapPointIDs

    /** Contains optional data
     * Such as borderStyle, fillStyle
    */
    data

    metadata = {
        areaType: {
            "first_level_descriptor": "land",
            "second_level_descriptor": ""
        }
    }

    constructor (mapPointIDs, data={}) {
        super();

        this.mapPointIDs = mapPointIDs;
```

```
        this.data = data;
    }

    getAllPoints(database) {
        return this.mapPointIDs.map(id => database.db.get(id));
    }

    static areaFromObject(object) {
        let area = new shared.Area(object.mapPointIDs, object.data);
        area.ID = object.ID;
        area.metadata = object.metadata;
        return area;
    }
}

shared.ComplexAreaPart = class ComplexAreaPart extends shared.Area {
    /**
     * {string} indicating whether the complex area part is an inner or outer
part
     * of a complex area.
     */
    outerOrInner = "unknown"

    constructor (mapPointIDs, outerOrInner, data={}) {
        super();

        this.mapPointIDs = mapPointIDs;
        this.outerOrInner = outerOrInner;

        if (this.outerOrInner == "inner")
            this.metadata.areaType["second_level_descriptor"] =  "none";

        this.data = data;
    }

    static complexAreaPartFromObject(object) {
        let complexAreaPart = new shared.ComplexAreaPart(object.mapPointIDs,
object.outerOrInner, object.data);
        complexAreaPart.ID = object.ID;
        complexAreaPart.metadata = object.metadata;
        return complexAreaPart;
    }
}

/**
 * Used to define area made of multiple areas
 */
shared.ComplexArea = class ComplexArea extends shared.MapDataObject {
```

```
    outerAreaID
    innerAreaIDs

    constructor(outerAreaID, innerAreaIDs) {
        super();

        this.outerAreaID = outerAreaID;
        this.innerAreaIDs = innerAreaIDs;
    }

    static complexAreaFromObject(object) {
        let complexArea = new shared.ComplexArea(object.outerAreaID,
object.innerAreaIDs);
        complexArea.ID = object.ID;
        complexArea.metadata = object.metadata;
        return complexArea;
    }
}

// Used to sanitize labels, etc for adding to DOM
https://stackoverflow.com/a/2794366
function encodeHTML(s) {
    return s.replace(/&/g, '&amp;').replace(/</g, '&lt;').replace(/"/g,
'&quot;');
}

shared.getPathMidpoint = (path, database) => {
    let pathPoints = path.getAllPointsOnPath(database);

    // Sum distance of path so we can find mid point
    let distanceTotal = 0;
    let lastPoint = pathPoints[0];
    pathPoints.forEach(pathPoint => {
        distanceTotal += Math.sqrt( Math.abs(pathPoint.x - lastPoint.x)**2 +
Math.abs(pathPoint.y - lastPoint.y)**2)
        lastPoint = pathPoint;
    });

    // Find point nearest to midpoint, use this as point
    let distanceTraversed = 0;
    lastPoint = pathPoints[0];
    for (let j = 0; j < pathPoints.length; j++) {
        const pathPoint = pathPoints[j];
        distanceTraversed += Math.sqrt( Math.abs(pathPoint.x - lastPoint.x)**2
+ Math.abs(pathPoint.y - lastPoint.y)**2);
        if (distanceTraversed >= distanceTotal/2) {
            return pathPoint;
        }
```

```
        lastPoint = pathPoint;
    }
}

shared.getBoundsOfPointArray = (pointArray) => {
    let minX;
    let minY;
    let maxX;
    let maxY;
    pointArray.forEach(point => {
        if (minY == undefined || point.y < minY) minY = point.y;
        if (minX == undefined || point.x < minX) minX = point.x;
        if (maxY == undefined || point.y > maxY) maxY = point.y;
        if (maxX == undefined || point.x > maxX) maxX = point.x;
    });
    let maxDistance = Math.sqrt( Math.abs(maxX - minX)**2 + Math.abs(maxY -
minY)**2 );
    let zoomLevel = (400/maxDistance < 10) ? 400/maxDistance : 10;
    return {minX, minY, maxX, maxY, maxDistance, zoomLevel};
}
```

Web client

Index.html and index.css are included in the Appendix instead of this section as they do not demonstrate algorithms or data structures but may be useful for reference.

src/web/index/index.js

```
canvasState = undefined;
debug_displayAreasDrawn = false;
debug_drawAllHighwayLabelsTest = true;
debug_drawHighwayLabels_smart = true;
debug_testDB = false;
globalPoint=null;

class CanvasState {
    /** The {CanvasRenderingContext2D} that is used on the canvas */
    ctx
    /** The canvas element that displays the map */
    canvas
    /** Indicates whether the mouse button is down */
    canvasMouseDown = false
    /** Indicates the last recorded mouse position relative to the page in X */
    lastPageX = -1
    /** Indicates the last recorded mouse position relative to the page in Y */
    lastPageY = -1
    /** The database containing map points */
    database = new shared.MapDataObjectDB()
```

```
    /** A {int} multiplier to represent the zoom level */
    zoomLevel = 1
    /** {int} representing how the map has been translated in x */
    xTranslation = 0
    /** {int} representing how the map has been translated in y */
    yTranslation = 0
    touchDevice = false
    /** An array of test nodes that should be drawn to screen, and nothing
else, when draw() is called if not null */
    testMapPoints = null
    /** Http request */
    httpReq
    /** Time of last map data update */
    timeOfLastMapDataUpdate = 0;
    /** Map data update ongoing flag */
    mapDataUpdateOngoing = false
    /** Map data update queued flag */
    mapDataUpdateQueued = false;
    /** Contains details about last/current area drawn to screen */
    area
    /** Indicates wether stroke is on */
    stokeOn = false;
    /** Stores the value of the last number of path types drawn */
    pathTypeCountLast = 0;
    /** Maps objects onto a grid made of 10x10 squares so can be queried more
quickly */
    mapObjectsGridCache;
    /** Array of points to draw (e.g, search markers are added to this) */
    pointsToDraw = []
    /** Array of paths to draw over other paths (e.g. routes) */
    pathsToDraw = []
    /** Stores Events for pinch gestures */
    pointEvents = []
    /** Store previous distance between digits */
    lastPinchDistance = 0;
    /** Indicates wether a pinch zoom is in progress */
    pinchZoomInProgress = false;

    /** Stores details of areas drawn to screen */
    areasDrawn = [];
    /** Drawn labels */
    labelsDrawn = [];

    constructor () {
        this.canvas = document.getElementById("mapCanvas");
        this.ctx = this.canvas.getContext('2d');
        this.canvas.style.cursor = "default";
```

```
        // Zoom functionality
        document.getElementById("zoom-in").onclick = (event) => {
            this.zoom(1);
        };

        document.getElementById("zoom-out").onclick = (event) => {
            this.zoom(-1);
        };

        // Translation functionality
        this.canvas.ontouchstart = (event) => {
            this.touchDevice = true;
            this.mapInteractionBegin(event.targetTouches[0].pageX,
event.targetTouches[0].pageY);
        }

        this.canvas.onmousedown = (event) => {
            if (!this.touchDevice) {
                this.mapInteractionBegin(event.pageX, event.pageY);
            }
        }

        this.canvas.ontouchend = () => this.mapInteractionEnd();
        this.canvas.onmouseup = () => this.mapInteractionEnd();

        this.canvas.ontouchmove = (event) => {
            if (event.targetTouches.length == 1 && !this.pinchZoomInProgress)
                this.mapDrag(event.targetTouches[0].pageX,
event.targetTouches[0].pageY);
        }

        this.canvas.onmousemove = (event) => {
            if (this.canvasMouseDown && !this.pinchZoomInProgress)
                this.mapDrag(event.pageX, event.pageY);
        }

        this.canvas.onmouseenter = (event) => {
            this.canvas.style.cursor = "grab";
        }

        this.canvas.onmouseleave = (event) => {
            this.canvas.style.cursor = "default";
        }

        // Code to support pinch zoom based on
https://developer.mozilla.org/en-
US/docs/Web/API/Pointer_events/Pinch_zoom_gestures by Mozilla Contributors,
licensed under CC-BY-SA 2.5
```

```
        this.canvas.onpointerdown = (event) => {
            canvasState.pointEvents.push(event)
        }

        this.canvas.onpointermove = (event) => {
            // Find this event in the cache and update its record with this
event
            for (var i = 0; i < canvasState.pointEvents.length; i++) {
                if (event.pointerId == canvasState.pointEvents[i].pointerId) {
                    canvasState.pointEvents[i] = event;
                    break;
                }
            }

            // If 2 fingers zooming
            if (canvasState.pointEvents.length == 2) {
                canvasState.pinchZoomInProgress = true;
                var curDiff = Math.abs(canvasState.pointEvents[0].clientX -
canvasState.pointEvents[1].clientX);

                if (this.lastPinchDistance != 0) {
                    canvasState.zoomLevel *=
(curDiff/canvasState.lastPinchDistance);
                    console.log((curDiff/canvasState.lastPinchDistance));
                    canvasState.draw();
                }

                canvasState.lastPinchDistance = curDiff;
            }
        }

        this.canvas.onpointerup = (eventA) => {
            canvasState.pointEvents = canvasState.pointEvents.filter(eventB =>
eventA.pointerId != eventB.pointerId);
            if (canvasState.pointEvents.length == 0) {
                canvasState.updateMapData();
                canvasState.pinchZoomInProgress = false;
            };
        }

        // Redraw on window resize to make sure canvas is right size
        window.addEventListener('resize', () => this.draw());

        // Set up map grid cache
        this.mapObjectsGridCache = new shared.MapGridCache(this.database)
    }

    mapInteractionEnd() {
```

```
        this.lastPageX = -1;
        this.lastPageY = -1;
        this.canvasMouseDown = false;
        if (this.canvas.style.cursor == "grabbing")
        this.canvas.style.cursor = "grab";

        this.updateMapData();
    }

    mapInteractionBegin(pageX, pageY) {
        this.lastPageX = pageX;
        this.lastPageY = pageY;
        this.canvasMouseDown = true;
        if (this.canvas.style.cursor == "grab")
            this.canvas.style.cursor = "grabbing";
    }

    mapDrag(pageX, pageY) {
        var relativeMouseX = pageX - this.lastPageX;
        var relativeMouseY = pageY - this.lastPageY;

        this.mapTranslate(relativeMouseX, relativeMouseY);

        this.lastPageX = pageX;
        this.lastPageY = pageY;

        this.draw();
    }

    mapTranslate(translateX, translateY) {
        this.xTranslation += (translateX / this.zoomLevel);
        this.yTranslation += (translateY / this.zoomLevel);
    }

    /**
     * Method to change the zoom level of the displayed map
     * @param {int} multiplier The multiplier to change the zoom by, normally
either -1 or 1
     */
    zoom(multiplier) {
        let zoomChange = 1 + (0.3 * multiplier);
        if (this.zoomLevel + zoomChange < 0)
            return;
        this.zoomLevel *= zoomChange;

        this.draw();
        this.updateMapData();
    }
```

```
    /**
     * Calls functions to draw path to screen
     */
    draw(drawBlankCanvasOnly=false) {
        // Update canvas width
        this.updateCanvasWidth();

        // Clear canvas
        this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height);

        // Fill background
        this.ctx.fillStyle = "#e6e6e6";
        this.ctx.fillRect(0, 0, this.canvas.width, this.canvas.height);

        if (drawBlankCanvasOnly) return;

        let objectsOnScreen = this.getObjectsOnScreen();

        let topLayerAreas = [];
        let areas = objectsOnScreen["AREA"];

        // Draw lower layer areas, adding lower layer areas to topLayerAreas to
draw later
        if (areas != undefined)
        for (let i = 0; i < areas.length; i++) {
            const area = areas[i];
            if (area.metadata.areaType["first_level_descriptor"] == "land") //
TODO: Create "layer" property and read from this instead
                area.draw(this);
            else
                topLayerAreas.push(area);
        }

        for (let i = 0; i < topLayerAreas.length; i++) {
            const area = topLayerAreas[i];
            area.draw(this);
        }

        let complexAreas = objectsOnScreen["COMPLEX-AREA"];

        if (complexAreas != undefined)
        for (let i = 0; i < complexAreas.length; i++) {
            const complexArea = complexAreas[i];
            complexArea.draw();
        }

        if (this.testMapPoints != null) {
```

```
            // Halt drawing, call test draw function instead, typical drawing
will not be executed
            this.#testDraw();
            return;
        }

        let paths = objectsOnScreen["PATH"];
        if (paths != undefined)
        for (let i = 0; i < paths.length; i++) {
            let path = paths[i];
            let acceptedPathTypes = this.getPathTypes();

            // Only plot line if is one of accepted path types for zoom level
            if
(acceptedPathTypes.includes(path.metadata.pathType["second_level_descriptor"])
            ||
acceptedPathTypes.includes(path.metadata.pathType["first_level_descriptor"]))
                path.plotLine(this);
        }
        // this.database.getMapObjectsOfType("POINT").forEach(point =>
point.drawPoint(this));

        // Clear drawn labels
        this.labelsDrawn = [];

        // Draw highway labels
        if (paths != undefined)
        for (let i = 0; i < paths.length; i++) {
            const path = paths[i];
            path.drawLabel();
        }

        // Draw points and paths to draw
        this.pointsToDraw.forEach(point => point.drawPoint());
        this.pathsToDraw.forEach(path => path.plotLine());

        if (debug_displayAreasDrawn) debug_displayAreasDrawnFunc();
    }

    #testDraw() {
        this.testMapPoints.forEach(mapPoint => {
            mapPoint.drawPoint(this);
        });
    }

    /**
     * Updates the width of the canvas displayed on screen
     */
```

```
    updateCanvasWidth() {
        // Resize to 100% (html decleration does not work)
        this.canvas.width = window.innerWidth;
        this.canvas.height = window.innerHeight;
    }

    updateMapData(forceUpdate=false) {
        // Assemble types of paths to query based on zoom level
        let pathTypes = this.getPathTypes();
        if (pathTypes.length > this.pathTypeCountLast) {
            this.pathTypeCountLast = pathTypes.length;
            // Force an update as more path types have been requested
            forceUpdate = true;
        }

        // Only continue if forceUpdate flag is set, or more than 500ms since
last map data update and mapDataUpdateOngoing flag is not set
        if (!forceUpdate && (this.mapDataUpdateOngoing || Date.now() -
this.timeOfLastMapDataUpdate < 500)) {
            if (!this.mapDataUpdateQueued) {
                // Try again in 1s time
                setTimeout(() => {
                    this.mapDataUpdateQueued = false;
                    this.updateMapData();
                }, 1000);
                this.mapDataUpdateQueued = true;
            }
            return;
        }

        this.mapDataUpdateOngoing = true;

        canvasState.area = {
            x: -canvasState.xTranslation - 500,
            y: -canvasState.yTranslation - 500,
            height:
(getAbsoluteHeight(canvasState.canvas)/canvasState.zoomLevel) + 500,
            width: (getAbsoluteWidth(canvasState.canvas)/canvasState.zoomLevel)
+ 500,
            pathTypeCount: pathTypes.length
        };

        // Check map area hasn't already been drawn
        let hasBeenDrawn = false;
        for (let i = 0; i < this.areasDrawn.length; i++) {
            const areaDrawn = this.areasDrawn[i];
            hasBeenDrawn = hasBeenDrawn || (areaDrawn.pathTypeCount >=
canvasState.area.pathTypeCount &&
```

```
                (canvasState.area.x >= areaDrawn.x && canvasState.area.x +
canvasState.area.width <= areaDrawn.x + areaDrawn.width
                    && canvasState.area.y >= areaDrawn.y && canvasState.area.y
+ canvasState.area.height <= areaDrawn.y + areaDrawn.height))
            if (hasBeenDrawn) break;
        }

        if (forceUpdate || !hasBeenDrawn) {
            // Make request
            let testingDBurl = `/api/GetTestDB`;
            let wholeDBurl = `/api/GetDBfromFile`;
            let testURLnoMapArea =
`/api/GetDBfromQuery?pathTypes=[%22motorway%22,%22primary%22,%22trunk%22,%22pri
mary_link%22,%22trunk_link%22,%22river%22]&&noMapAreaFilter=true`;
            let testURLlimitedArea =
`/api/GetDBfromQuery?pathTypes=[%22motorway%22,%22primary%22,%22trunk%22,%22pri
mary_link%22,%22trunk_link%22,%22river%22]&x=48.1699954728&y=9784.703958946639&
height=1317.4001900055023&width=1271.3921765555658&excludeAreas=[]`;
            let normalURL =
`/api/GetDBfromQuery?pathTypes=${JSON.stringify(pathTypes)}&area=${JSON.stringi
fy(canvasState.area)}`;

            let currentURL;
            if (debug_testDB) currentURL = testingDBurl;
            else currentURL = normalURL;

            this.httpReq.open("GET", currentURL);
            this.httpReq.send();

            // Display loading indicator
            document.getElementById("loading-indicator-
container").style.display = "block";
        } else {
            this.mapDataUpdateOngoing = false;
        }
    }

    mapDataReceiveFunc = () => {
        // Request returns db as uninstanciated object
        // we need to convert this
        var simpleDB = JSON.parse(canvasState.httpReq.response);

        if(shared.debug_on)
            console.log(`Received JSON has ${Object.keys(simpleDB.db).length}
items.`)

        var database =
shared.MapDataObjectDB.MapDataObjectDBFromObject(simpleDB);
```

```
        canvasState.database.mergeWithOtherDB(database);

        this.mapObjectsGridCache.cacheDataToGrid();

        if(shared.debug_on)
            console.log(`Computed database currently has
${canvasState.database.getMapObjectsOfType("PATH").length} paths.`);

        // Draw
        canvasState.draw();

        // Add area drawn to list of areas drawn
        canvasState.areasDrawn.push(Object.assign({}, canvasState.area));

        // Update time since last map data update
        canvasState.timeOfLastMapDataUpdate = Date.now();

        // Update map data update ongoing flag
        canvasState.mapDataUpdateOngoing = false;

        // Hide loading indicator
        document.getElementById("loading-indicator-container").style.display =
"none";
    }

    /**
     * Returns those map objects on screen as per mapObjectsGridCache.
     */
    getObjectsOnScreen() {
        var objectsOnScreen = {};
        let xTranslation = canvasState.xTranslation;
        let yTranslation = canvasState.yTranslation;
        let zoomLevel = canvasState.zoomLevel;

        let gridSquareSize = canvasState.mapObjectsGridCache.gridSquareSize;

        let xInitial = Math.floor((-
xTranslation)/gridSquareSize)*gridSquareSize - 2*gridSquareSize;
        let xIncrement = gridSquareSize;
        let xLimit = Math.floor(((canvasState.canvas.width/(zoomLevel*1.5)) -
xTranslation)/gridSquareSize)*gridSquareSize + 2*gridSquareSize;

        let yInitial = Math.floor((-
yTranslation)/gridSquareSize)*gridSquareSize - 2*gridSquareSize;
        let yIncrement = gridSquareSize;
        let yLimit = Math.floor(((canvasState.canvas.height/zoomLevel) -
yTranslation)/gridSquareSize)*gridSquareSize + 2*gridSquareSize;
```

```
        var objectIDsAdded = new Map();

        for (let x = xInitial;
        x < xLimit;
        x += xIncrement) {
            for (let y = yInitial;
            y < yLimit;
            y += yIncrement) {
                let square = canvasState.mapObjectsGridCache.get(`${x}x${y}`);
                if (square != undefined) {
                    for (let i = 0; i < square.length; i++) {
                        const mapObjectID = square[i];
                        if (!objectIDsAdded.get(mapObjectID)) {
                            let type = mapObjectID.slice(0,
mapObjectID.indexOf("_"));

                            if (objectsOnScreen[type] == undefined)
objectsOnScreen[type] = [];


objectsOnScreen[type].push(this.database.db.get(mapObjectID));
                            objectIDsAdded.set(mapObjectID, true);
                        }
                    }
                }
            }
        }

        return objectsOnScreen;
    }

    translateToCoords(x,y,zoom=true) {
        if (zoom) canvasState.zoomLevel = 4;

        canvasState.xTranslation = -x + (75 / canvasState.zoomLevel);
        canvasState.yTranslation = -y + (200 / canvasState.zoomLevel);

        canvasState.draw();
    }

    /**
     * Initiates a search
     * @param {string} input search term
     */
    search = (input) => {
        var resultsAccordion = document.getElementById("results-accordion");
```

```
        resultsAccordion.innerHTML = "<strong class='text-
muted'>Searching...</strong>";
        document.getElementById("results-title").innerText = `Search results
for "${input}"`;

        // Make search results draggable
        $("#results").draggable({cancel: "#results-accordion"});

        // Show results dialog
        let resultsEl = document.getElementById("results");
        (new bootstrap.Toast(resultsEl)).show();

        let http = new XMLHttpRequest();
        http.addEventListener("load", () => {
            if (http.responseText.indexOf("error") == 0) {
                resultsAccordion.innerHTML = "<strong class='text-muted'>An
error occurred :-(</strong>";
                return;
            };

            try {
                let responseArray = JSON.parse(http.responseText);

                canvasState.pointsToDraw = [];

                // Display search results
                let points = [];
                for (let i = 0; i < responseArray.length; i++) {
                    const mapObject = responseArray[i].mapObject;
                    let point;

                    if (mapObject.ID.indexOf("POINT") == 0)
                        point = MapPoint.mapPointFromObject(mapObject);
                    else if (mapObject.ID.indexOf("PATH") == 0) {
                        let path = Path.pathFromObject(mapObject);
                        path.midpoint =
MapPoint.mapPointFromObject(mapObject.midpoint);
                        point = path.midpoint;
                        // Add metadata of path to point
                        point.metadata.path = path.metadata;
                    }

                    point.searchScore = responseArray[i].score;

                    // // If no label available, don't display if search
confidence is below % of max
                    // if (points.length > 0 && point.label == point.ID &&
point.searchScore < points[0].searchScore * 0.6) continue;
```

```
                    points.push(point);
                }

                resultsAccordion.innerHTML = "";

                if (points.length == 0) resultsAccordion.innerHTML = "<strong
class='text-muted'>No results found :-(</strong>";
                else resultsAccordion.innerHTML = "";

                for (let i = 0; i < points.length; i++) {
                    const point = points[i];

                    let pointToDraw = MapPoint.generatePointWithLabel(point.x,
point.y, `${i+1}`);

                    if (!canvasState.pointsToDraw.includes(pointToDraw));
                        canvasState.pointsToDraw.push(pointToDraw);

                    resultsAccordion.innerHTML += `
                        <div class="accordion-item">
                        <h2 class="accordion-header">
                            <button class="accordion-button collapsed"
type="button" data-bs-toggle="collapse" data-bs-target="#collapse${i}">
                            <div>📌${i+1}: ${point.label} <span class="fw-
lighter"> (${point.locationType})</span></div>
                            </button>
                        </h2>
                        <div id="collapse${i}" class="accordion-collapse
collapse">
                            <div class="accordion-body">
                            <a
href="javascript:canvasState.translateToCoords(${point.x},${point.y})">
                            <strong>Go to point.</strong></a>
                            <br/><strong>Search confidence:</strong>
${point.searchScore}
                            <br/><strong>ID:</strong> ${point.ID}
                            <br/><strong>Is path:</strong>
${point.metadata.path != undefined}
                            <br/>
                            <a class="link-secondary" data-bs-toggle="collapse"
href="#metadata-collapse-${i}" role="button">
                                Show/hide additional metadata.
                            </a>
                            <div class="collapse" id="metadata-collapse-${i}">
                                ${point.metadataHTML}
                            </div>
                            </div>
```

```
                        </div>
                        </div>`;

                    if (point.metadata.path != undefined) point.metadata.path =
undefined;
                }

                if (points.length > 0) {
                    let bounds = shared.getBoundsOfPointArray(points);
                    canvasState.zoomLevel = bounds.zoomLevel;
                    canvasState.translateToCoords(bounds.minX, bounds.minY,
false);
                    canvasState.draw();
                    canvasState.updateMapData();
                }
            } catch (err) {console.log(err)}
        });
        http.open("GET", `/api/PointSearch?searchTerm="${input}"`);
        http.send();
    }

    route(pointA, pointB) {
        let http = new XMLHttpRequest();
        http.addEventListener("load", () => {
            if (http.responseText.indexOf("error") == 0 || http.responseText ==
"[]") {
                let errorText = "An error occurred.";
                if (http.responseText == "error: input")
                    errorText = "Could not find points.";
                if (http.responseText.indexOf("error: highway point
undefined.") == 0)
                    errorText = "One or more of the points is not near enough
to an accepted highway to be able to find a route.";
                if (http.responseText == "[]")
                    errorText = "Unable to find route.";

                document.getElementById("route-status").innerText = errorText +
" :-(";
                return
            };

            let responseArray = JSON.parse(http.responseText);

            let pointArray = responseArray.map(simplePoint =>
shared.MapPoint.mapPointFromObject(simplePoint));

            let path = Path.connectSequentialPoints(pointArray,
canvasState.database);
```

```
            path.metadata.pathType = {};
            path.metadata.pathType["second_level_descriptor"] =
"returned_route";
            canvasState.pathsToDraw = [path];

            let bounds =
shared.getBoundsOfPointArray(path.getAllPointsOnPath(canvasState.database));
            canvasState.zoomLevel = bounds.zoomLevel;
            canvasState.translateToCoords(bounds.minX, bounds.minY, false);

            let pointA = MapPoint.generatePointWithLabel(pointArray[0].x,
pointArray[0].y, `A`);
            let pointB =
MapPoint.generatePointWithLabel(pointArray[pointArray.length - 1].x,
pointArray[pointArray.length - 1].y, `B`);

            canvasState.pointsToDraw = [];

            if (!canvasState.pointsToDraw.includes(pointA));
                canvasState.pointsToDraw.push(pointA);
            if (!canvasState.pointsToDraw.includes(pointB));
                canvasState.pointsToDraw.push(pointB);

            canvasState.updateMapData();
            canvasState.draw();

            document.getElementById("route-status").innerText = "Route found!";
        });
        http.open("GET",
`/api/FindRoute?startingPointTerm="${pointA}"&destinationPointTerm="${pointB}"`
);
        http.send();
        document.getElementById("route-status").innerText = "Finding...";
    }

    /**
     * Returns an array of the path types to display based on zoom level
     */
    getPathTypes() {
        let pathsToDisplay = ["motorway", "primary", "trunk", "primary_link",
"trunk_link", "river", "returned_route"];
        if (this.zoomLevel > 0.05)
            pathsToDisplay = pathsToDisplay.concat(["secondary",
"secondary_link"]);
        if (this.zoomLevel > 0.2)
            pathsToDisplay = pathsToDisplay.concat(["tertiary",
"tertiary_link"]);
        if (this.zoomLevel > 0.4)
```

```
            pathsToDisplay = pathsToDisplay.concat(["unclassified",
"residential"]);

        return pathsToDisplay;
    }
}

class Path extends shared.Path {
    /** Data, including options for the path when drawing to screen */
    data = {
        borderWidth: 0,
        lineWidth: 0,
        borderStyle: "none",
        fillStyle: "none"
    }

    /**
     *
     * @param {string} startingPathPartID The ID of the path part that starts
the path
     * @param {CanvasState} canvasState
     * @param {object} data Options, etc
     */
    constructor (startingPathPartID=null, data={}) {
        super();
        this.startingPathPartID = startingPathPartID;
        this.data = {...this.data, ...data};
    }

    /**
     * Plots line by traversing tree.
     */
     plotLine() {
        if (canvasState == null) {
            console.warn("Canvas state not defined, unable to plot path.");
            return;
        }

        this.getPathStyle();

        // Initialize array containing the  initial PathParts to draw
        let startingPathPartsToDraw =
[canvasState.database.db.get(this.startingPathPartID)];

        const skipOneInEveryNo = 1;

        // Iterate through startingPathPartsToDraw
        for (let i = 0; i < startingPathPartsToDraw.length; i++) {
```

```
            const startingPathPart = startingPathPartsToDraw[i];
            canvasState.database.db.get(startingPathPart.pointID).canvasState =
canvasState;

            // Move to the starting point
            let startX =
canvasState.database.db.get(startingPathPart.pointID).displayedX;
            let startY =
canvasState.database.db.get(startingPathPart.pointID).displayedY;

            // If we get to a branch, push the other branches to
startingPathPartsToDraw to iterate through later
            let currentPathPart = startingPathPart;
            while (currentPathPart.nextPathPartIDs.length >= 0) {

canvasState.database.db.get(currentPathPart.pointID).canvasState = canvasState;

                // Advance pointer to next connecting point in the closest
branch
                // Or if skipping, skip to the one after that if not at the end
of the path
                let nextPointer =
shared.PathPart.getPartByStepsAway(canvasState.database, currentPathPart, 3);

                let endX =
canvasState.database.db.get(currentPathPart.pointID).displayedX;
                let endY =
canvasState.database.db.get(currentPathPart.pointID).displayedY;

                // Plot a line from the last plotted point to the point at
currentPathPart
                canvasState.ctx.beginPath();

                canvasState.ctx.lineWidth = this.data.borderWidth +
this.data.lineWidth;
                canvasState.ctx.strokeStyle = this.data.borderStyle;

                canvasState.ctx.moveTo(startX, startY);
                canvasState.ctx.lineTo(endX, endY);

                canvasState.ctx.stroke();

                canvasState.ctx.beginPath();

                canvasState.ctx.lineWidth = this.data.lineWidth;
                canvasState.ctx.strokeStyle = this.data.fillStyle;

                canvasState.ctx.moveTo(startX, startY);
```

```
                canvasState.ctx.lineTo(endX, endY);

                canvasState.ctx.stroke();

                startX = endX;
                startY = endY;

                for (let j = 1; j < currentPathPart.nextPathPartIDs.length;
j++) {

startingPathPartsToDraw.push(canvasState.database.db.get(currentPathPart).nextP
athPartIds[j]);
                }

                if (currentPathPart.nextPathPartIDs.length == 0) break;

                currentPathPart = nextPointer;
            }
        }
    }

    /**
     * Plots points contained in tree.
     */
    plotPoints(canvasState=canvasState) {
        this.getAllPointsOnPath().forEach(point =>
point.drawPoint(canvasState));
    }

    /**
     * Apply a key to the style of the path. Called while drawing.
     */
    getPathStyle() {
        switch (this.metadata.pathType["first_level_descriptor"]) {
            case "water_way":
                this.data.borderWidth = 0;
                this.data.lineWidth = 4;
                this.data.borderStyle = "none";
                this.data.fillStyle = "#b0e1f7";
                break;
            default:
                break;
        }
        switch (this.metadata.pathType["second_level_descriptor"]) {
            case "returned_route":
                this.data.borderWidth = 0;
                this.data.lineWidth = 6;
                this.data.borderStyle = "none";
```

```
    this.data.fillStyle = "#ff293b"; // Red
    break;
case "motorway":
  this.data.borderWidth = 4;
  this.data.lineWidth = 1;
  this.data.borderStyle = "#347aeb"; // Blue
  this.data.fillStyle = "#2b2b2b";
  break;
case "motorway_link":
  this.data.borderStyle = "#347aeb";
  this.data.fillStyle = "#2b2b2b";
  break;
case "trunk":
  this.data.borderWidth = 2;
  this.data.lineWidth = 2;
  this.data.borderStyle = "#d622a0"; // Pink
  this.data.fillStyle = "#347aeb"; // Blue
  break;
case "trunk_link":
  this.data.borderWidth = 2;
  this.data.lineWidth = 3;
  this.data.borderStyle = "#2b2b2b";
  this.data.fillStyle = "#d622a0";
  break;
case "primary":
  this.data.borderWidth = 4;
  this.data.lineWidth = 0;
  this.data.borderStyle = "#d622a0"; // Pink
  break;
case "primary_link":
  this.data.lineWidth = 0;
  this.data.borderStyle = "#d622a0";
  break;
case "secondary":
  this.data.borderWidth = 0;
  this.data.lineWidth = 5;
  this.data.fillStyle = "#e6a31e"; // Orange
  break;
case "secondary_link":
  this.data.borderWidth = 1;
  this.data.lineWidth = 4;
  this.data.borderStyle = "#2b2b2b";
  this.data.fillStyle = "#e6a31e";
  break;
case "tertiary_link":
case "tertiary":
case "unclassified":
case "residential":
```

```
                this.data.borderWidth = 1,
                this.data.lineWidth = 3,
                this.data.borderStyle = "#adadad",
                this.data.fillStyle = "#e6e6e6" // Gray
              break;
            default:
              break;
          }

          if (this.data.borderWidth == 0 && this.data.lineWidth == 0) {
            this.data.borderWidth = 1,
            this.data.lineWidth = 3,
            this.data.borderStyle = "#adadad",
            this.data.fillStyle = "#e6e6e6" // Gray
          }
    }

    drawLabel() {
        canvasState.ctx.font = '10px sans-serif';
        canvasState.ctx.fillStyle = "black";
        canvasState.ctx.strokeStyle = "white";

        if (!debug_drawHighwayLabels_smart && debug_drawAllHighwayLabelsTest &&
this.metadata.osm.ref != undefined) {
            let startingPoint =
canvasState.database.db.get(canvasState.database.db.get(this.startingPathPartID
).pointID);
            canvasState.ctx.fillText(this.metadata.osm.ref,
startingPoint.displayedX, startingPoint.displayedY);
        }

        if (debug_drawHighwayLabels_smart) {
            let startingPathPart =
canvasState.database.db.get(this.startingPathPartID);
            let startingPoint =
startingPathPart.getPoint(canvasState.database);

            // Get label text
            let labelText;
            if (this.metadata.osm.ref != undefined) labelText =
this.metadata.osm.ref;
            else if (this.metadata.osm.name != undefined) labelText =
this.metadata.osm.name;

            if (labelText == undefined) return;

            let textWidth = labelText.length * 6.5;
```

```
            let nextPart =
startingPathPart.getPartByDistanceAway(canvasState.database, textWidth);
            if (!nextPart) return; // Don't draw if there isn't a suitable
point to measure angle for
            let nextPoint = nextPart.getPoint(canvasState.database);

            // Get angle between startingPoint and nextPoint
            let changeInX = startingPoint.x - nextPoint.x;
            let changeInY = startingPoint.y - nextPoint.y;
            let angle = Math.atan(changeInY/changeInX);

            let newLabel = {text: labelText,
                        x: startingPoint.displayedX + 3,
                        y: startingPoint.displayedY + 3,
                        angle: angle};

            // Only draw if there isn't another label close by and the coords
are on screen.
            // Check by iterating through labels drawn
            for (let i = 0; i < canvasState.labelsDrawn.length; i++) {
                const oldLabel = canvasState.labelsDrawn[i];
                let minDistanceAwayFromOtherLabels = textWidth;
                if (Math.abs(newLabel.x - oldLabel.x) <
minDistanceAwayFromOtherLabels || Math.abs(newLabel.y - oldLabel.y) <
minDistanceAwayFromOtherLabels)
                    return;
            }

            // Move origin to starting point (rotation is done about origin)
            canvasState.ctx.translate(newLabel.x, newLabel.y);
            // Set angle of label drawing
            canvasState.ctx.rotate(newLabel.angle);

            // Draw label
            canvasState.ctx.strokeText(newLabel.text, 0, 0);
            canvasState.ctx.fillText(newLabel.text, 0, 0);

            // Reset angle and translation
            canvasState.ctx.setTransform(1, 0, 0, 1, 0, 0);

            // Add label to labels drawn
            if (!canvasState.labelsDrawn.includes(newLabel))
                canvasState.labelsDrawn.push(newLabel);
        }
    }
}
```

```
// This is needed so that references to Paths in shared.js reference Paths with
the client functionality defined here
shared.Path = Path;

class PathPart extends shared.PathPart {
    /**
     * @param {string} pointID The ID of the point referenced by this path part
     * @param {string} nextPathPartIDs Array of IDs of the next part(s) in the
path
     */
     constructor (pointID=null, nextPathPartIDs=[], metadata={}) {
        super();

        this.pointID = pointID;
        this.nextPathPartIDs = nextPathPartIDs;
        this.metadata = metadata
    }

    /**
     * Get the nearest part at least distance away
     * @param {MapDataObjectDB} database
     * @param {Number} distance
     * @returns {PathPart} or false if not found.
     */
     getPartByDistanceAway(database, distance) {
        let possiblePart = this;
        let getDistance = () =>
            MapPoint.displayedDistanceBetweenPoints(
            possiblePart.getPoint(database),
            this.getPoint(database));

        while (getDistance() < distance) {
            let nextPart = possiblePart.getNextPart(database);
            if (!nextPart) return false;
            else possiblePart = nextPart;
        }

        return possiblePart;
    }
}

shared.PathPart = PathPart;

class MapPoint extends shared.MapPoint {
    canvasState

    /** Options for the point when drawing to screen */
    options = {
```

```
        pointDrawMethod: "none",
        pointText: "📍",
        pointFont: "sans-serif",
        pointFontWidth: 16,
        pointFillStyle: "#878787",
        pathDrawPointX: 3,
        pathDrawPointY: -6
    }

    /**
     * Creates a point that can form part of a path and be displayed on a
canvas.
     * @param {int} x Fixed x position of point in relation to others
     * @param {int} y Fixed y position of point in relation to others
     * @param {object} options Options for the point when drawing to screen
     * @param {object} metadata Optional metadata
     */
     constructor (x, y, metadata={}) {
        super();

        this.x = x;
        this.y = y;

        this.metadata = metadata;
    }

    /** Gets the x position relative to the canvas */
    get displayedX() {
        return (this.x + canvasState.xTranslation) * 1.5 *
canvasState.zoomLevel;
    }

    /** Gets the y position relative to the canvas */
    get displayedY() {
        return (this.y + canvasState.yTranslation) * canvasState.zoomLevel;
    }

    /**
     * Function to draw the point to the screen
     */
     drawPoint() {
        if (canvasState == null) {
            console.warn("Canvas state not defined, unable to draw point.");
            return;
        }

        if (this.options.pointDrawMethod == "text") {
            // Draw point
```

```
            canvasState.ctx.fillStyle = this.options.pointFillStyle;
            canvasState.ctx.font = `${this.options.pointFontWidth}px
${this.options.pointFont}`;
            canvasState.ctx.fillText(this.options.pointText, this.displayedX,
this.displayedY);

            // Draw label
            let xOffset =
canvasState.ctx.measureText(this.options.pointText).width + 3;
            canvasState.ctx.fillStyle = this.options.pointFillStyle;
            canvasState.ctx.strokeStyle = this.options.labelBorderStyle;
            canvasState.ctx.font = `${this.options.labelFontWidth}px
${this.options.pointFont}`;
            canvasState.ctx.strokeText(this.options.labelText, this.displayedX
+ xOffset, this.displayedY);
            canvasState.ctx.fillText(this.options.labelText, this.displayedX +
xOffset, this.displayedY);
        }
    }

    static displayedDistanceBetweenPoints(a, b) {
        return Math.sqrt((a.displayedX - b.displayedX)**2 + (a.displayedY -
b.displayedY)**2);
    }

    static generatePointWithLabel(x, y, label) {
        let point = new MapPoint(x, y);
        point.options = {
            pointDrawMethod: "text",
            pointText: `📌`,
            labelText: `${label}`,
            labelFontWidth: 20,
            pointFont: "sans-serif",
            pointFontWidth: 25,
            pointFillStyle: "#878787",
            labelBorderStyle: "white",
            pathDrawPointX: 3,
            pathDrawPointY: -6
        }
        return point;
    }
}

// This is needed so that references to MapPoints in shared.js references
MapPoints with the client functionality defined here
shared.MapPoint = MapPoint;

class Area extends shared.Area {
```

```
    constructor (mapPointIDs, data={}) {
        super();

        this.mapPointIDs = mapPointIDs;
        this.data = data;
    }

    draw() {
        this.setAreaDrawStyle(canvasState);
        canvasState.ctx.beginPath();

        this.plotPath();

        Area.finishFillArea();
    }

    plotPath(reverse=false) {
        /** Might need to reverse in order to plot hole.
         * This is because JS canvas only draws a hole
         * if one polygon is drawn in opposite direction
         * to the other (e.g. anticlockwise and clockwise).
        */
       let mapPointIDs = this.mapPointIDs.slice(0); // Clone array
       if (reverse) mapPointIDs = mapPointIDs.reverse();

        for (let i = 0; i < mapPointIDs.length; i++) {
            const mapPointID = mapPointIDs[i];
            const mapPoint = canvasState.database.db.get(mapPointID);

            if (i==0) canvasState.ctx.moveTo(mapPoint.displayedX,
mapPoint.displayedY);
            else canvasState.ctx.lineTo(mapPoint.displayedX,
mapPoint.displayedY);
        }

        canvasState.ctx.closePath();
    }

    static finishFillArea() {
        canvasState.ctx.fill();
        if (canvasState.strokeOn) {
            canvasState.ctx.stroke();
            canvasState.strokeOn = false;
        }
    }

    static isClockwise(pointIDs) {
        // https://stackoverflow.com/a/1165943
```

```
        let areaSum = 0;
        for (let i = 0; i < pointIDs.length - 1; i++) {
            const point = canvasState.database.db.get(pointIDs[i]);
            const pointAfter = canvasState.database.db.get(pointIDs[i+1]);
            areaSum += (pointAfter.x - point.x) * (pointAfter.y +
pointAfter.y);
        }
        return areaSum >= 0;
    }

    isClockwise() {
        return Area.isClockwise(this.mapPointIDs);
    }

    isAreaOnScreen() {
        let isApointOnScreen = false;
        for (let i = 0; !isApointOnScreen && i < this.mapPointIDs.length; i++)
{
            const mapPoint = canvasState.database.db.get(this.mapPointIDs[i]);
            if (!isApointOnScreen) isApointOnScreen =
areCoordsOnScreen(mapPoint.displayedX, mapPoint.displayedY, canvasState);
        }

        return isApointOnScreen;
    }

    /**
     * Sets properties of how an area should be drawn to screen (such as fill
colour)
     * to be called before drawing this area.
     * @param {2D canvas context thing from HTML standard} ctx
     */
    setAreaDrawStyle(canvasState) {
        let ctx = canvasState.ctx;

        if (this.metadata.areaType != undefined &&
this.metadata.areaType["first_level_descriptor"] != undefined)
        switch (this.metadata.areaType["first_level_descriptor"]) {
            case "water_area":
                ctx.fillStyle = "#8fafe3";
                break;
            case "land":
                ctx.fillStyle = "#bfbfbf";
                break;
            default:
                ctx.fillStyle = "#ffffff";
                break;
        }
```

```
        if (this.metadata.areaType != undefined &&
this.metadata.areaType["second_level_descriptor"] != undefined)
        switch (this.metadata.areaType["second_level_descriptor"]) {
            case "allotments":
            case "farmland":
            case "farmyard":
            case "flowerbed":
            case "forest":
            case "meadow":
            case "orchard":
            case "vineyard":
            case "grass":
            case "recreation_ground":
            case "village_green":
            case "plant_nursery":
                ctx.fillStyle = "#ace0a1";
                break;
            case "basin":
            case "reservoir":
            case "salt_pond":
                ctx.fillStyle = "#8fafe3";
                break;
            case "none":
                ctx.fillStyle = "rgba(255, 255, 255, 0)";
                break;
            default:
                break;
        }
    }
}

shared.Area = Area;

class ComplexArea extends shared.ComplexArea {
    constructor(outerAreaID, innerAreaIDs) {
        super();

        this.outerAreaID = outerAreaID;
        this.innerAreaIDs = innerAreaIDs;
    }

    get outerArea() {
        return canvasState.database.db.get(this.outerAreaID);
    }

    get innerAreas() {
        return this.innerAreaIDs.map(id => canvasState.database.db.get(id));
```

```
    }

    draw() {
        if (this.outerArea == undefined) return;

        this.outerArea.setAreaDrawStyle(canvasState);
        canvasState.ctx.beginPath();

        this.outerArea.plotPath();
        let isOuterAreaClockwise = this.outerArea.isClockwise();

        // Draw inner areas that require empty hole, electing those
        // that are filled to simply be draw over the larger shape
        let innerAreasFilled = [];
        this.innerAreas.forEach(innerArea => {
            if (innerArea != undefined)
            if (innerArea.metadata.areaType["second_level_descriptor"] !=
"none")
                innerAreasFilled.push(innerArea);
            else {
                let drawInReverse = isOuterAreaClockwise ==
innerArea.isClockwise();
                innerArea.plotPath(drawInReverse);
            }
        });

        Area.finishFillArea();

        // Draw filled inner areas
        innerAreasFilled.forEach(area => area.draw());
    }
}

shared.ComplexArea = ComplexArea;

/** Unfortunately this class has to be fully duplicated from
 * sharedStructures unlike the other classes, otherwise it
 * inherits shared.Area (without drawing functions) not Area
 * (with drawing functions).
 */
 ComplexAreaPart = class ComplexAreaPart extends Area {
    /**
     * {string} indicating whether the complex area part is an inner or outer
part
     * of a complex area.
     */
    outerOrInner = "unknown"
```

```
    constructor (mapPointIDs, outerOrInner, data={}) {
        super();

        this.mapPointIDs = mapPointIDs;
        this.outerOrInner = outerOrInner;
        this.data = data;
    }

    static complexAreaPartFromObject(object) {
        let complexAreaPart = new shared.ComplexAreaPart(object.mapPointIDs,
object.outerOrInner, object.data);
        complexAreaPart.ID = object.ID;
        complexAreaPart.metadata = object.metadata;
        return complexAreaPart;
    }
}

shared.ComplexAreaPart = ComplexAreaPart;

/**
 * Function called at page load
 */
function main() {
    // Create canvas state
    canvasState = new CanvasState();

    // Set up search event listener
    document.getElementById("search_input").addEventListener("keypress",
function (e) {
        if (e.key == "Enter") {
            canvasState.search(document.getElementById("search_input").value);
        }
    });

    // Import test nodes
    // Translate graph so does not overlap header
    canvasState.mapTranslate(15,
getAbsoluteHeight(document.getElementById("header")) + 15);

    // Drawing test overrides
    canvasState.xTranslation =  2292.886051499995;
    canvasState.yTranslation = -7349.380475070653;
    canvasState.zoomLevel =  0.5000000000000001;

    if (debug_testDB == true) debug_func_viewOrigin();

    // Load correct canvas width
    canvasState.updateCanvasWidth();
```

```
    // Display blank canvas
    canvasState.draw(true);

    // Get test db from server
    canvasState.httpReq = new XMLHttpRequest();
    canvasState.httpReq.addEventListener("load",
canvasState.mapDataReceiveFunc);

    // if(shared.debug_on) debug_viewWholeMap(canvasState);

    canvasState.updateMapData();

    routeSubmitFunction = () => {
        let pointA_input = document.getElementById("pointA_input").value;
        let pointB_input = document.getElementById("pointB_input").value;
        canvasState.route(pointA_input, pointB_input);
    }

    document.getElementById("pointB_input").addEventListener("keypress",
function (e) {
        if (e.key == "Enter") {
            routeSubmitFunction();
        }
    });

    $("#route-search-toast").draggable({cancel: "#route-search-body"});
    document.getElementById("route-ui-show-btn").onclick = () =>  (new
bootstrap.Toast($("#route-search-toast"))).show();
    document.getElementById("route-submit-btn").onclick = () =>
{routeSubmitFunction()};
}

/**
 * Function that can be called for debug purposes to view the whole map.
 * Useful to verify that map data is being loaded correctly.
 * @param {*} canvasState
 */
function debug_viewWholeMap(canvasState) {
    canvasState.xTranslation = -4021.6666666666615;
    canvasState.yTranslation = -2433.8333333333303;
    canvasState.zoomLevel = 0.10000000000000014;
}

function debug_func_viewOrigin() {
    canvasState.xTranslation =  0;
    canvasState.yTranslation = 0;
    canvasState.zoomLevel =  1;
```

```
    canvasState.draw()
}

/**
 * Display canvasState.areasDrawn on map for debug purposes.
 */
function debug_displayAreasDrawnFunc() {
    canvasState.areasDrawn.forEach(areaDrawn => {
        canvasState.ctx.beginPath();
        canvasState.ctx.lineWidth = "6";
        canvasState.ctx.strokeStyle = "red";
        canvasState.ctx.rect(
            (areaDrawn.x + canvasState.xTranslation) * canvasState.zoomLevel,
            (areaDrawn.y + canvasState.yTranslation) * canvasState.zoomLevel,
            areaDrawn.width * canvasState.zoomLevel,
            areaDrawn.height * canvasState.zoomLevel
        );
        canvasState.ctx.stroke();
    })
}

function areCoordsOnScreen(x, y) {
    return x > 0 && x < canvasState.canvas.width
            && y > 0 && y < canvasState.canvas.height;
}

// Once the page has fully loaded, call main
document.addEventListener('DOMContentLoaded', main, false);

// Libraries

// https://stackoverflow.com/a/23749355/
function getAbsoluteHeight(el) {
    // Get the DOM Node if you pass in a string
    el = (typeof el === 'string') ? document.querySelector(el) : el;

    var styles = window.getComputedStyle(el);
    var margin = parseFloat(styles['marginTop']) +
                    parseFloat(styles['marginBottom']);

    return Math.ceil(el.offsetHeight + margin);
}

function getAbsoluteWidth(el) {
    // Get the DOM Node if you pass in a string
    el = (typeof el === 'string') ? document.querySelector(el) : el;

    var styles = window.getComputedStyle(el);
```

```
    var margin = parseFloat(styles['marginRight']) +
                   parseFloat(styles['marginLeft']);

    return Math.ceil(el.offsetWidth + margin);
}
```

Server

src/index.js

```
//Load HTTP module

var express = require('express');
var fs = require('fs');
var app = express();
const hostname = 'localhost';
const port = 80;
const options = {index: "index/index.html"};
const path = require("path");
const JSONStream = require('JSONStream');
var shared = require('./shared/sharedStructures.js').shared;

var logger = new (require('./logging.js').Logger)();

logger.log("Server is starting...");

// Using https://stackoverflow.com/a/29371929
app.use(function (req, res, next) {
    var filename = "/" + path.basename(req.url);
    logger.log(filename + " was requested.");
    next();
});

logger.log("Loading db.");
getData(() => {
    logger.log("Loaded db.");

    app.get("/api/GetTestDB", (req, res) =>
require("./api/getTestDB.js").getTestDB(shared, req, res));
    app.get("/api/GetTestOSMpoints", (req, res) =>
require("./api/getTestOSMpoints.js").getTestOSMpoints(shared, req, res));
    app.get("/api/GetDBfromFile", (req, res) =>
require("./api/getDBfromFile.js").getDBfromFile(shared, req, res));
    app.get("/api/GetDBfromQuery", (req, res) =>
require("./api/dbFromQuery.js").getDBfromQuery(shared, req, res));
    app.get("/api/PointSearch", (req, res) =>
require("./api/pointSearch.js").pointSearchAPI(shared, req, res));
    app.get("/api/FindRoute", (req, res) =>
require("./api/findRoute.js").findRoute(shared, req, res));

    app.use(express.static('web', options));
    app.use(express.static('shared', options));
```

```
    // Error handler
    app.use((err, req, res, next) => {
        logger.log(err.stack)
        res.status(500).send("error: misc");
    })

    app.listen(port);

    logger.log(`Server has started at http://${hostname}:${port}`);
});

function getData(callback) {
    // Read file
    data = fs.readFileSync('db.json', 'utf8');

    // Convert from JSON to MapDataObjectDB
    let simpleDB = JSON.parse(data);
    shared.database =
shared.MapDataObjectDB.MapDataObjectDBFromObject(simpleDB);

    // Cache database data to grid
    shared.mapObjectsGridCache = new shared.MapGridCache(shared.database);
    shared.mapObjectsGridCache.gridSquareSize = 10;
    shared.mapObjectsGridCache.cacheDataToGrid();

    // Create search index
    let SearchIndex = require("./api/pointSearch.js").SearchIndex;
    shared.searchIndex = new SearchIndex(shared.database);

    callback();
}

// // Read JSON (based on https://stackoverflow.com/a/49298376)
// function getData(callback) {
//     let readStream = fs.createReadStream('db.json', 'utf8');
//     let parser = JSONStream.parse(readStream);
//     readStream.pipe(parser);

//     parser.on('data', (simpleDB) => {
//         shared.database =
shared.MapDataObjectDB.MapDataObjectDBFromObject(simpleDB);
//         // shared.highwayPaths =
require("./api/dbFromQuery.js").getHighwayPaths(shared);

//         // Cache database data to grid
//         shared.mapObjectsGridCache = new
shared.MapGridCache(shared.database);
//         shared.mapObjectsGridCache.gridSquareSize = 10;
```

```
//          shared.mapObjectsGridCache.cacheDataToGrid();
//          callback();
//      });
// }
```

src/convertFromOSM.js

```
const fs = require('fs');
var convert = require('xml-js');
var shared = require('./shared/sharedStructures.js').shared;

var mapDatabase = new shared.MapDataObjectDB();

var osmData;
var cachedOSMdataAvailable = process.argv.includes("--cacheRead") &&
fs.existsSync(".osmToJS_osmData.json");
if (cachedOSMdataAvailable) {
  console.log("Reading OSM-JS cache...")
  osmData = JSON.parse(fs.readFileSync(".osmToJS_osmData.json"));
  console.log("\tOSM-JS cache read complete");
} else {
  console.log("Beginning conversion from OSM to OSM-JS - This might take a
while...");
  let data = fs.readFileSync('../docs/cambridgeshire-latest.osm', 'utf8');
  osmData = convert.xml2js(data, {compact: false});
  console.log("\tOSM to OSM-JS conversion complete.");
}

// Cache osmData
try {
  if (!process.argv.includes("--noCacheWrite") && !cachedOSMdataAvailable)
  fs.writeFile(".osmToJS_osmData.json", JSON.stringify(osmData), function(err)
{
    if(err) {
        return console.log(err);
    }
    console.log("Background process: OS-JS data been cached");
  });
} catch (e) {
  // Error most likely occurs because osmData is too large for JSON.stringify
  console.log(`Unable to cache OS-JS data. ${e.name}: ${e.message}. Continuing
anyway...`);
}

console.log("Extracting ways, nodes and relations...");
```

```
var cachedNodedataAvailable = process.argv.includes("--cacheRead") &&
fs.existsSync(".osmToJS_nodes.json");

var allWays = [];
var nodes = {};
var relations = [];

for (let i = 0; i < osmData.elements[0].elements.length; i++) {
  const element = osmData.elements[0].elements[i];
  if (element.name == "node" && !cachedNodedataAvailable)
nodes[element.attributes.id] = element;
  if (element.name == "way") allWays.push(element);
  if (element.name == "relation") relations.push(element);
}

if (cachedNodedataAvailable) {
  console.log("\tReading cached node data:");
  nodes = JSON.parse(fs.readFileSync(".osmToJS_nodes.json"));
  console.log("\t\tCached node data read completed.");
}

console.log("\tWay, node and relation extraction complete.");

// Cache nodes
try {
  if (!process.argv.includes("--noCacheWrite") && !cachedNodedataAvailable)
  fs.writeFile(".osmToJS_nodes.json", JSON.stringify(nodes), function(err) {
    if(err) {
        return console.log(err);
    }
    console.log("Background process: Nodes have been cached");
  });
} catch (e) {
  // Error most likely occurs because osmData is too large for JSON.stringify
  console.log(`Unable to cache nodes. ${e.name}: ${e.message}. Continuing
anyway...`);
}

// Extract ways with <tag k="highway" ...>

var filteredWays = [];

var cachedwaysAvailable = process.argv.includes("--cacheRead") &&
fs.existsSync(".osmToJS_ways.json");
if (cachedwaysAvailable) {
  console.log("Reading cached ways...");
  allWays = JSON.parse(fs.readFileSync(".osmToJS_ways.json"));
  console.log("\tCached way read complete.")
```

```
} else {
  console.log("Extracting ways...");
  for (let w = 0; w < allWays.length; w++) {
    const way = allWays[w];
    const wayType = getWayType(way);
    if (wayType != "no_way")
          filteredWays.push(way);
  }

  console.log("\tWay extraction complete.");
}

// Cache ways
try {
  if (!process.argv.includes("--noCacheWrite") && !cachedwaysAvailable)
  fs.writeFile(".osmToJS_ways.json", JSON.stringify(filteredWays),
function(err) {
    if(err) {
        return console.log(err);
    }
    console.log("Background process: Ways have been cached");
  });
} catch (e) {
  // Error most likely occurs because osmData is too large for JSON.stringify
  console.log(`Unable to cache ways. ${e.name}: ${e.message}. Continuing
anyway...`);
}

// Extract multi-polygons, identifying way areas that make up mutli-polygons
class Multipolygon {
  outerWayId
  innerWayIds
  relation
  constructor(outerWayId, innerWayIds, relation) {
    this.outerWayId = outerWayId;
    this.innerWayIds = innerWayIds;
    this.relation = relation;
  }
}

var multipolygons = [];
relations.forEach(relation => {
  let isMultipolygon = false;
  let outerWayId;
  let innerWayIds = [];
    relation.elements.forEach(element => {
      if (element.name == "tag" && element.attributes.k == "type" &&
element.attributes.v == "multipolygon")
```

```
        isMultipolygon = true;
      else if (element.name == "member" && element.attributes.type == "way")
        if (element.attributes.role == "outer") outerWayId =
element.attributes.ref;
        else if (element.attributes.role == "inner")
innerWayIds.push(element.attributes.ref);
    });

    if (isMultipolygon && outerWayId != undefined && innerWayIds.length > 0)
      multipolygons.push(new Multipolygon(outerWayId, innerWayIds, relation));
});

// Identify those ways that compose multipolygons so can be efficiently
// queried later.
var wayIDsComposingMultipolygons = [];
multipolygons.forEach(multipolygon => {
  wayIDsComposingMultipolygons.push(multipolygon.outerWayId);
  wayIDsComposingMultipolygons =
wayIDsComposingMultipolygons.concat(multipolygon.innerWayIds);

  let complexArea = new shared.ComplexArea();
  Object.assign(complexArea.metadata, {"osm":
extractMetadata(multipolygon.relation)});
  complexArea.metadata.osm.outerWayId = multipolygon.outerWayId;
  complexArea.metadata.osm.innerWayIds = multipolygon.innerWayIds;
  mapDatabase.addMapObject(complexArea);
})

let complexAreas = mapDatabase.getMapObjectsOfType("COMPLEX-AREA");

// Extract nodes for each way and creating paths
console.log("Generating DB...");

var wLength = filteredWays.length;
var refsAddedToDb = [];

// var wLength = 100;
for (let w = 0; w < wLength; w++) {
  const way = filteredWays[w];
  const wayType = getWayType(way, wayIDsComposingMultipolygons);

  if (wayType == "other") continue;

  // Extract node references to add
  var nodeRefsOfWay = [];
  for (let e = 0; e < way.elements.length; e++) {
    const element = way.elements[e];
    if (element.name == "nd") nodeRefsOfWay.push(element.attributes.ref);
```

```
  }

  // Extract nodes from node references to add
  var mapPointsOfWay = [];

  for (let r = 0; r < nodeRefsOfWay.length; r++) {
    const ref = nodeRefsOfWay[r];
    let node = nodes[ref];
    let mapPoint = addNodeToDb(node, mapDatabase);

    // Add map point to array
    mapPointsOfWay.push(mapPoint);
    refsAddedToDb.push(ref);
  }

  if (wayType == "highway" || wayType == "water_way") {
    // Generate path
    let path = shared.Path.connectSequentialPoints(mapPointsOfWay,
mapDatabase);

    // Extract all metadata of path
    Object.assign(path.metadata, {"osm": extractMetadata(way)});

    // Add path type
    if (wayType == "highway")
      path.metadata.pathType = {"first_level_descriptor": "highway",
"second_level_descriptor": path.metadata.osm.highway};
    else if (wayType == "water_way")
      path.metadata.pathType = {"first_level_descriptor": "water_way",
"second_level_descriptor": path.metadata.osm.waterway};

    mapDatabase.addMapObject(path);
  } else if (["water_area", "land", "inner", "outer"].includes(wayType)) {
    // Add mapPoints to DB and extract IDs
    let mapPointIDs = [];
    mapPointsOfWay.forEach(mapPoint => {
      mapPoint = mapDatabase.addMapObject(mapPoint);
      mapPointIDs.push(mapPoint.ID);
    });

    // Construct area and add to db
    let area;
    if (wayType == "inner" || wayType == "outer")
      area = new shared.ComplexAreaPart(mapPointIDs, wayType);
    else
      area = new shared.Area(mapPointIDs);

    // Add metadata to area mapObject
```

```
    Object.assign(area.metadata, {"osm": extractMetadata(way)});
    if (wayType != "inner" && wayType != "outer")
      area.metadata.areaType = {"first_level_descriptor": wayType};
    if (area.metadata.osm.waterway != undefined)
      area.metadata.areaType["second_level_descriptor"] =
area.metadata.osm.waterway;
    else if (area.metadata.osm.landuse != undefined)
      area.metadata.areaType["second_level_descriptor"] =
area.metadata.osm.landuse;
    else if (wayType == "outer") {
      area.metadata.areaType["first_level_descriptor"] = "land";
    } else if (wayType == "inner")
      area.metadata.areaType["second_level_descriptor"] = "none";

    mapDatabase.addMapObject(area);
  }

  process.stdout.cursorTo(0);
  process.stdout.write(`${Math.round(((w+1)/wLength)*10000)/100}%
(${w+1}/${wLength}) of ways added to db.`);
}

// Associate complex area parts with complex areas
let complexAreaParts = mapDatabase.getMapObjectsOfType("COMPLEX-AREA-PART");

console.log("\nProcessing complex areas...");
let complexAreasLength = complexAreas.length;
for (let i = 0; i < complexAreasLength; i++) {
  const complexArea = complexAreas[i];
  complexArea.outerAreaID = complexAreaParts.find(part => part.metadata.osm.id
== complexArea.metadata.osm.outerWayId).ID;
  complexArea.innerAreaIDs = complexAreaParts.filter(part =>
complexArea.metadata.osm.innerWayIds.includes(part.metadata.osm.id)).map(part
=> part.ID);

  process.stdout.cursorTo(0);
  process.stdout.write(`${Math.round(((i+1)/complexAreasLength)*10000)/100}%
(${i+1}/${complexAreasLength}) of complex areas processed.`);
}

console.log("\nComplex areas processed.");

// Add all nodes to db
console.log("Adding all nodes to db...");
let nodesFullArray = Object.values(nodes);
let nodesLength = nodesFullArray.length;
for (let i = 0; i < nodesLength; i++) {
  const node = nodesFullArray[i];
```

```
  addNodeToDb(node, mapDatabase);

  process.stdout.cursorTo(0);
  process.stdout.write(`${Math.round(((i+1)/nodesLength)*10000)/100}%
(${i+1}/${nodesLength}) of nodes added to db.`);
}
console.log("\nAdded all nodes to db...");

console.log("Database generated. Writing to file...");

// Write database to file
var dbFilename = `db-${Date.now()}.json`;
try {
  let dbKeys = Array.from(mapDatabase.db.keys());
  let writeStream = fs.createWriteStream(dbFilename);

  writeStream.write(`{"db":{`);
  for (let i = 0; i < dbKeys.length; i++) {
    const dbKey = dbKeys[i];

writeStream.write(`"${dbKey}":${JSON.stringify(mapDatabase.db.get(dbKey))}`);
    if (i+1 < dbKeys.length) writeStream.write(`,`);

    process.stdout.cursorTo(0);
    process.stdout.write(`${Math.round(((i+1)/dbKeys.length)*10000)/100}%
(${i+1}/${dbKeys.length}) database items written to file.`);
  }
  writeStream.write("}}");
  writeStream.end();
  console.log(`\nMap database has been saved to ${dbFilename}!`);
} catch (err) {
  console.log(err);
}

function convertNodeToMapPoint(node) {
  if (node.attributes == undefined || node.attributes.lon == undefined ||
node.attributes.lat == undefined)
    return null;

  // We have to convert latitude and longitude to x,y coordinates
  // Meaning we have to use triganometry, as lat and lon are
  // defined in terms of angle
  // This uses the mercator projection as defined at
https://gis.stackexchange.com/a/314225 CC BY-SA 4.0

  // Radius of earth
  var radius = 6371;
```

```
  // Convert longitude to x
  var x = radius * (node.attributes.lon);

  // Convert latitude to y
  var y = radius * Math.log(Math.abs(Math.tan((Math.PI/4) +
(node.attributes.lat / 2))));

  var MapPoint = new shared.MapPoint(x, y);

  return MapPoint;
}

/**
 * @returns {string} the type of way
 */
function getWayType(way, wayIDsComposingMultipolygons=[]) {
  let wayId = "";
  if (way.attributes.id != undefined) wayId = way.attributes.id;
  else if (way.attributes.ref != undefined) wayId = way.attributes.ref;

  if (wayIDsComposingMultipolygons.includes(wayId)) {
    for (let i = 0; i < multipolygons.length; i++) {
      const multipolygon = multipolygons[i];
      if (multipolygon.outerWayId == wayId) return "outer";
      else if (multipolygon.innerWayIds.includes(wayId)) return "inner";
    }
  }

  for (let i = 0; i < way.elements.length; i++) {
    const element = way.elements[i];
    if (element.name == "tag" && element.attributes.k == "highway") return
"highway";
    if (element.name == "tag" && element.attributes.k == "waterway") return
"water_way";
    if (element.name == "tag" && element.attributes.k == "natural" &&
element.attributes.v == "water") return "water_area";
    if (element.name == "tag" && element.attributes.k == "landuse") return
"land";
  }

  if (way.name != "way") return "no_way";

  return "other";
}

/**
 * Extracts the metadata for a simple db way.
```

```
 * @param {object} way way from simple db
 * @returns object containing tag key:value pairs
 */
function extractMetadata(way) {
  let metadata = {};

  for (let e = 0; e < way.elements.length; e++) {
    const element = way.elements[e];
    if (element.name == "tag")
      metadata[element.attributes.k] = element.attributes.v;
  }

  if (way.attributes.id != undefined) metadata.id = way.attributes.id;
  else if (way.attributes.ref != undefined) metadata.id = way.attributes.ref;

  return metadata;
}

function addNodeToDb(node, mapDatabase) {
  let mapPoint = convertNodeToMapPoint(node);

  // Extract all metadata of node
  if (node.elements != undefined)
    for (let e = 0; e < node.elements.length; e++) {
      const element = node.elements[e];
      if (element.name == "tag")
        mapPoint.metadata[element.attributes.k] = element.attributes.v;
    }

  mapDatabase.addMapObject(mapPoint);

  return mapPoint;
}
```

src/logging.js

```
const fs = require('fs');
const consts = require('./consts.js')

/**
 * Logger class, instanciated once to provide logging functionality
 */
module.exports.Logger = class Logger {
    /**
     * Logfile location
     */
    logfile
```

```
    /**
     * Instanciates the logger, creating a log file if it does not exist
     */
    constructor() {
        this.logfile = `logs/map-${Date.now()}.log`;
        var message = `Logger has started.`;

        // Create 'logs' folder if it doesn't exist
        if (!fs.existsSync('logs')){
            fs.mkdirSync('logs');
        }

        this.log(message);
    }

    /**
     * Generates a log stamp that is added at the beginning of log messages, to
indicate project name and timestamp
     * @param {bool} logError Optional flag, if true, adds a note to log stamps
to indicate that only console logging is being used.
     * @returns {string} A log stamp that is added at the beginning of log
messages
     */
    generateLogStamp = () => `[${consts.projectName}, ${Date.now()}]`;

    /**
     * Used to write a log message to console and file.
     * @param {string} message Message to write to log
     */
    log(message) {
        var stampedMessage = `${this.generateLogStamp()} ${message}`;

        console.log(stampedMessage);

        fs.appendFile(this.logfile, stampedMessage + "\n", err => {
            if (err) console.log("Logging error: " + err.name);
        });
    }
}
```

src/dbFromQuery.js

```
fs = require('fs');
logger = new (require('../logging.js').Logger)();

// Used to count number of items filtered via pathTypes for debugging
```

```
var debug_pathTypeFilterCount = 0;
// Used to count number of items filtered via search term for debugging
var debug_searchFilterCount = 0;

/**
 * Querys the database stored in a file based on parameters passed to API.
 * @param {Object} shared The shared structures
 * @param {*} req
 * @param {*} res
 */
 module.exports.getDBfromQuery = function (shared, req, res) {
    var databaseToReturn = new shared.MapDataObjectDB();

    let startTime = Date.now();

    // Get map db for area
    var areaDB = filterByMapArea(req, res, shared.database, shared);

    // Filter paths
    paths = [];
    rootDBpaths = areaDB.getMapObjectsOfType("PATH");
    for (let i = 0; i < rootDBpaths.length; i++) {
        const path = rootDBpaths[i];
        var meetsCriteria = true;

        // These filter function calls return undefined if unsuccesful, where
bool && undefined == undefined
        // Filter by pathTypes
        meetsCriteria = meetsCriteria && filterByPathType(req, res, path,
shared);

        if (meetsCriteria) {
            paths.push(path)
        } else if (meetsCriteria == undefined) return;
    }

    paths.forEach(path => {
        path.copyPathContentsToDB(areaDB, databaseToReturn);
        databaseToReturn.addMapObject(path);
    });

    areaDB.getMapObjectsOfType(["AREA", "COMPLEX-AREA-PART"]).forEach(area => {
        databaseToReturn.addMapObject(area);
        area.mapPointIDs.forEach(mapPointID => {
            let point = shared.database.db.get(mapPointID);
            if (point != undefined)
            databaseToReturn.addMapObject(point);
        });
```

```
    });

    areaDB.getMapObjectsOfType("COMPLEX-AREA").forEach(area => {
        databaseToReturn.addMapObject(area);
    });
    let returnString = "";

    let dbKeys = Array.from(databaseToReturn.db.keys());

    returnString += `{"db":{`;
    for (let i = 0; i < dbKeys.length; i++) {
        const dbKey = dbKeys[i];
        returnString +=
`"${dbKey}":${JSON.stringify(databaseToReturn.db.get(dbKey))}`;
        if (i+1 < dbKeys.length) returnString += `,`;
    }

    returnString += "}}";
    res.send(returnString);
 }

 function filterByPathType(req, res, path, shared) {
    var pathTypeError = false;
    // var acceptedPathTypes = ["motorway","primary","trunk"];
    var acceptedPathTypes = [];

    try {
        if (req.query.pathTypes != undefined)
        acceptedPathTypes = JSON.parse(req.query.pathTypes);
    } catch {
        pathTypeError = true;
    }

    if (pathTypeError || !(acceptedPathTypes instanceof Array))
        return throwParamError("INVALID_PARAM: pathTypes (dbFromQuery)", res);

    containsSecondLevelDescriptor = path.metadata.pathType != undefined &&
path.metadata.pathType["second_level_descriptor"] != undefined &&
acceptedPathTypes.includes(path.metadata.pathType["second_level_descriptor"]);
    containsFirstLevelDescriptor = path.metadata.pathType != undefined &&
path.metadata.pathType["first_level_descriptor"] != undefined &&
acceptedPathTypes.includes(path.metadata.pathType["first_level_descriptor"]);

    let result = (acceptedPathTypes.length == 0
        || containsSecondLevelDescriptor
        || containsFirstLevelDescriptor);

    if (shared.debug_on && result) debug_pathTypeFilterCount++;
```

```
    return result;
}

function filterByMapArea(req, res, db, shared) {
    if (req.query.noMapAreaFilter) return db;

    // Filter by map area
    var x;
    var y;
    var height;
    var width;

    var mapAreaError = false;

    try {
        let area = req.query.area;
        if (area == undefined) mapAreaError = true;
        area = JSON.parse(area);

        if (area.x != undefined)
            x = parseInt(area.x);
        else mapAreaError = true;

        if (area.y != undefined)
            y = parseInt(area.y);
        else mapAreaError = true;

        if (area.height != undefined)
            height = parseInt(area.height);
        else mapAreaError = true;

        if (area.width != undefined)
            width = parseInt(area.width);
        else mapAreaError = true;
    } catch (err) {
        mapAreaError = true;
    }

    if (mapAreaError) return throwParamError("INVALID_PARAMS: mapArea
(dbFromQuery)", res);

    var filteredDB = new shared.MapDataObjectDB();

    // Find objects in map area
    var idsInArea = shared.mapObjectsGridCache.getSquareContentInBounds(x, y,
width, height);
```

```
    for (let i = 0; i < idsInArea.length; i++) {
       const objectID = idsInArea[i];
       let mapObject = db.db.get(objectID);

       if (mapObject == undefined)
           continue;

       filteredDB.addMapObject(mapObject);

       // If path, add whole path, its parts and points to db
       if (mapObject instanceof shared.Path) {
           let path = mapObject;
           path.copyPathContentsToDB(db, filteredDB);
       }
    }

    return filteredDB;
}

function throwParamError(errorDesc, res) {
   res.status(400).send(errorDesc);
   return undefined;
}
```

src/findRoute.js

```
logger = new (require('../logging.js').Logger)();
PointSearch = require("./pointSearch.js");

/**
 * Uses path finding to find the shortest route from a starting point
 * to a destination, returning this to the client.
 * @param {*} shared
 * @param {*} req
 * @param {*} res
 */
module.exports.findRoute = (shared, req, res) => {
    var startingPointTerm;
    var destinationPointTerm;

    let inputError = false;
    if (req.query.startingPointTerm != undefined &&
req.query.destinationPointTerm != undefined) {
        // Convert search terms from json
        try {
            startingPointTerm =
JSON.parse(decodeURI(req.query.startingPointTerm));
            destinationPointTerm =
JSON.parse(decodeURI(req.query.destinationPointTerm));
        } catch {inputError = true}
    } else inputError = true;

    if (inputError) {
        res.end("error: input");
        return;
    }

    let startingPointID = quickSearch(startingPointTerm, shared);
    let destinationPointID = quickSearch(destinationPointTerm, shared);

    if (startingPointID == undefined || destinationPointID == undefined) {
        res.end("error: input");
        return;
    }

    let startingPoint = shared.database.db.get(startingPointID);
    let destinationPoint = shared.database.db.get(destinationPointID);

    // Find point on highway nearest to starting point. We only route on major
    // highways to decrease nodes and therefore improve performance.
    let startingHighway = nearestHighwayPoint(startingPoint, shared);
```

```
    let destinationHighway = nearestHighwayPoint(destinationPoint, shared);

    if (startingHighway.error || destinationHighway.error) {
        res.end(`error: highway point undefined. error:
{a:${startingHighway.error}, b:${destinationHighway.error}}`);
        return;
    }

    let route = dijkstras(startingHighway, destinationHighway, shared);

    res.send(route);
};

function quickSearch(term, shared) {
    let database = shared.database;

    let result = PointSearch.pointSearch(term, shared).find(result =>
result.mapObject.ID.indexOf("POINT") == 0 ||
result.mapObject.ID.indexOf("PATH") == 0);

    if (result == undefined) return undefined;

    if (result.mapObject.ID.indexOf("PATH") == 0) {
        let path = database.db.get(result.mapObject.ID)
        return shared.getPathMidpoint(path, database).ID;
    }

    return result.mapObject.ID;
}

function isAcceptedPath(path) {
    return ["motorway", "motorway_link", "trunk", "trunk_link", "primary",
"primary_link", "secondary", "secondary_link"]
    .includes(path.metadata.pathType["second_level_descriptor"]);
}

function nearestHighwayPoint(pointA, shared) {
    let squaresToExamine =
[shared.mapObjectsGridCache.getSurroundingSquareContent(pointA, 10)];
    let closestPoint;
    let closestPath;

    for (let i = 0; i < squaresToExamine.length; i++) {
        const square = squaresToExamine[i];
        // Find paths in square
        for (let j = 0; j < square.length; j++) {
            const mapObjA = shared.database.db.get(square[j]);
```

```
            if (!(mapObjA instanceof shared.Path && isAcceptedPath(mapObjA)))
continue;

            let path = mapObjA;
            // Get points on path
            let pathPoints = path.getAllPointsOnPath(shared.database);
            for (let k = 0; k < pathPoints.length; k++) {
                const pointB = pathPoints[k];
                if (closestPoint == undefined || pointA.distanceToPoint(pointB)
< pointA.distanceToPoint(closestPoint)) {
                    closestPoint = pointB;
                    closestPath = path;
                }
            }
        }
    }

    if (closestPoint == undefined) return {error: true};

    return {point: closestPoint, path: closestPath};
}

function getNeighbours(shared, currentNode, nodeList) {
    let square = shared.mapObjectsGridCache.getSquare(currentNode);
    let neighbours = square.map(id => nodeList.get(id)).filter(node => node !=
undefined);
    let nearbyPaths = square.filter(id => id.indexOf("PATH_") == 0).map(id =>
shared.database.db.get(id));
    nearbyPaths.forEach(path => {
        let pointsOnPath = path.getAllPointsOnPath(shared.database);
        let currentNodeOnPathIndex = pointsOnPath.findIndex(point => point.ID
== currentNode.ID);
        if (currentNodeOnPathIndex != -1 && pointsOnPath[currentNodeOnPathIndex
+ 1] != undefined && nodeList.get(pointsOnPath[currentNodeOnPathIndex + 1].ID)
!= undefined)
            neighbours.push(pointsOnPath[currentNodeOnPathIndex + 1]);
        if (currentNodeOnPathIndex != -1 && pointsOnPath[currentNodeOnPathIndex
- 1] != undefined && nodeList.get(pointsOnPath[currentNodeOnPathIndex - 1].ID)
!= undefined)
            neighbours.push(pointsOnPath[currentNodeOnPathIndex - 1]);
    });

    return neighbours;
}

function dijkstras(startingHighway, destinationHighway, shared) {
    class NodeProps {
        visited = false;
```

```
        distance = Infinity;
        parentID = null;
    }

    // Define sets
    let visitedNodes = new Map();
    let unvisitedNodes = new Map();
    let nodeProps = new Map();

    shared.database.getMapObjectsOfType("PATH").filter(path =>
        isAcceptedPath(path)
    ).forEach(path =>
        path.getAllPointsOnPath(shared.database).forEach(node =>{
            unvisitedNodes.set(node.ID, node);
            nodeProps.set(node.ID, new NodeProps());
        }));

    nodeProps.set(startingHighway.point.ID, new NodeProps());
    nodeProps.set(destinationHighway.point.ID, new NodeProps());
    nodeProps.get(startingHighway.point.ID).distance = 0;

    let currentNode = startingHighway.point;

    while (!nodeProps.get(destinationHighway.point.ID).visited) {
        let neighbours = getNeighbours(shared, currentNode, unvisitedNodes);

        if (neighbours.every(node => nodeProps.get(node.ID).distance ==
Infinity))
            return [];

        neighbours.forEach(node => {
            let tentativeDistance = nodeProps.get(currentNode.ID).distance +
currentNode.distanceToPoint(node);
            if (tentativeDistance < nodeProps.get(node.ID).distance)
nodeProps.get(node.ID).distance = tentativeDistance;
            if (node.ID != currentNode.ID) nodeProps.get(node.ID).parentID =
currentNode.ID;
        });

        nodeProps.get(currentNode.ID).visited = true;
        unvisitedNodes.delete(currentNode.ID);
        visitedNodes.set(currentNode.ID, currentNode);

        let nextNode;

        for (let node of unvisitedNodes) {
            if ((nextNode == undefined || nodeProps.get(node[0]).distance <
nodeProps.get(nextNode.ID).distance)) {
```

```
                nextNode = node[1];
            };
        }

        if (nextNode == undefined) return [];

        currentNode = nextNode;
    }

    // Backtrack to find fastest path
    currentNode = destinationHighway.point;
    let fastestPath = [];
    while (currentNode.ID != startingHighway.point.ID) {
        let parentID = nodeProps.get(currentNode.ID).parentID;
        fastestPath.push(currentNode);
        currentNode = visitedNodes.get(parentID);
    }

    return fastestPath.reverse();
}
```

src/pointSearch.js

```
logger = new (require('../logging.js').Logger)();

class SearchResult {
    mapObject
    score

    constructor (mapObject, score) {
        this.mapObject = mapObject;
        this.score = score;
    }
}

/**
 * Conducts a search on points in root db, returning those that match
 * search params.
 * @param {*} shared
 * @param {*} req
 * @param {*} res
 */
module.exports.pointSearchAPI = (shared, req, res) => {
    // Get search params
    var searchTerm;
    let inputError = false;
    if (req.query.searchTerm != undefined) {
```

```
        // Convert search terms from json
        try {
            searchTerm = JSON.parse(decodeURI(req.query.searchTerm));
            if (!(typeof searchTerm == "string")) inputError = true
        } catch {inputError = true}
    } else inputError = true;

    if (inputError) {
        res.end("error");
        return;
    }

    let searchResults = module.exports.pointSearch(searchTerm, shared);

    res.send(searchResults);
};

module.exports.pointSearch = (searchTerm, shared) => {
    // Returned database will just contain points, should not just be merged
into client db
    var searchResults = [];
    let searchIndex = shared.searchIndex.map;

    for (const [id, field] of searchIndex) {
        // let regexWholeWord = new
RegExp(`(\s|^|{|")+(${searchTerm})(\s|$|")+`, "gi");
        // let matchesWholeWord =
[...metadataAsString.matchAll(regexWholeWord)];
        // let regexMatchNoWhitespace = new RegExp(`${searchTerm}`, "gi");
        // let matchesNoWhitespace =
[...metadataAsString.matchAll(regexMatchNoWhitespace)];
        let regexMatchSimple = new RegExp(`${searchTerm}`, "gi");
        let matchesSimple =
[...field.metadataAsString.matchAll(regexMatchSimple)];
        let matchesIDscore = (id == searchTerm) ? 20 : 0;
        let isPlaceScore = (field.mapObject.metadata.place != undefined &&
matchesSimple.length != 0) ? 15 : 0;
        let score = isPlaceScore + matchesIDscore + matchesSimple.length;
        if (score > 0) {
            let result = new SearchResult(field.mapObject, score);
            searchResults.push(result);
        }
    }

    searchResults.sort((a,b) => {
        return b.score - a.score;
    });
```

```
    searchResults = searchResults.slice(0, 20);

    return searchResults;
}

/**
 * This creates a search index which is created at start of runtime that
 * enables stringified metadata to be searched and read from fast.
 */
module.exports.SearchIndex = class SearchIndex {
    /** Maps a map object ID onto its stringified map data */
    map = new Map();

    /** Initializes the index */
    constructor (database) {
        let rootDBmapObjects = database.getMapObjectsOfType(["POINT", "PATH"]);
        rootDBmapObjects.forEach(mapObject => {
            let metadataAsString = JSON.stringify(mapObject.metadata);
            if (mapObject instanceof shared.Path) mapObject.midpoint =
shared.getPathMidpoint(mapObject, shared.database);
            this.map.set(mapObject.ID, {metadataAsString, mapObject});
        });
    }
}
```

src/getDBfromFile.js

```
fs = require('fs');

/**
 * Sends test points from JSON as JSON data back to the client.
 * @param {Object} shared The shared structures
 * @param {*} req
 * @param {*} res
 */
 module.exports.getDBfromFile = function (shared, req, res) {
    // Read file
    fs.readFile('db.json', 'utf8' , (err, data) => {
        if (err) {
            const error='FILE_READ_ERROR (getDBfromFile)';
            res.status(500).send(error);
            module.exports.Logger.log(error + "\n" + err.toString());
            return;
        }

        res.send(data);
    });
```

```
 }
```

src/getTestDB.js

```
/**
 * Sends test database containing test paths as JSON data back to the client.
 * @param {Object} shared The shared structures
 * @param {*} req
 * @param {*} res
 */
module.exports.getTestDB = function (shared, req, res) {
    var database = new shared.MapDataObjectDB();

    // Create path 1
    var pointsPath1 = [
        new shared.MapPoint(50, 50),
        new shared.MapPoint(60, 55),
        new shared.MapPoint(100, 70),
        new shared.MapPoint(110, 100)
    ]

    var path1 = shared.Path.connectSequentialPoints(pointsPath1, database);
    path1.metadata.pathType = {"first_level_descriptor": "highway",
"second_level_descriptor": "primary"};


    database.addMapObject(path1);

    // Create path 2
    var pointsPath2 = [
        new shared.MapPoint(30, 200),
        new shared.MapPoint(400, 170),
        new shared.MapPoint(300, 250),
        new shared.MapPoint(270, 20),
        new shared.MapPoint(120, 100),
        new shared.MapPoint(160, 160)
    ];

    var path2 = shared.Path.connectSequentialPoints(pointsPath2, database);
    path2.metadata.pathType = {"first_level_descriptor": "highway",
"second_level_descriptor": "primary"};

    database.addMapObject(path2);

    var polygonOuterPoints = [
```

```
        new shared.MapPoint(30, 210),
        new shared.MapPoint(100, 210),
        new shared.MapPoint(100, 240),
        new shared.MapPoint(30, 240)
    ]

    var polygonInnerPoints = [
        new shared.MapPoint(35, 215),
        new shared.MapPoint(95, 215),
        new shared.MapPoint(95, 235),
        new shared.MapPoint(35, 235)
    ]

    var combinedComplexPoints = polygonInnerPoints.concat(polygonOuterPoints);

    for (let i = 0; i < combinedComplexPoints.length; i++) {
        const point = combinedComplexPoints[i];
        database.addMapObject(point);
    }

    var polygonOuterIDs = polygonOuterPoints.map(point => point.ID);
    var polygonOuterArea = new shared.ComplexAreaPart(polygonOuterIDs,
"outer");
    polygonOuterArea.metadata.areaType = {"first_level_descriptor": "land",
"second_level_descriptor": "grass"};

    database.addMapObject(polygonOuterArea);

    var polygonInnerIDs = polygonInnerPoints.map(point => point.ID);
    var polygonInnerArea = new shared.ComplexAreaPart(polygonInnerIDs,
"inner");
    polygonInnerArea.metadata.areaType = {"first_level_descriptor": "land",
"second_level_descriptor": "none"};

    database.addMapObject(polygonInnerArea);

    var testComplexArea = new shared.ComplexArea(polygonOuterArea.ID,
[polygonInnerArea.ID]);

    database.addMapObject(testComplexArea);

    res.send(database);
}
```

Abandoned – Tile mapping

The following code was abandoned and should not be considered as fully working. It is not utilised as part of the final codebase.

It was a partially completed attempt at implementing tile mapping, which produced accurate map tiles, but where not all map tiles of the map were written to file by the program, and which while could be displayed on the client, were not aligned correctly. It was abandoned as I did not believe it to realistically be completed in time.

This is not a complete listing of all associated tile mapping files, but identifies the main algorithms and data structures.

src/MapAreaToImage.js [excerpt]

This is only an excerpt, as the rest of the file just copies canvas data structures from the web index.js.

```
// Initialize CanvasState
canvasState = new CanvasState();

// Read file
console.log("Reading from db.json...");
data = fs.readFileSync('db.json', 'utf8');

// Convert from JSON to MapDataObjectDB
let simpleDB = JSON.parse(data);
canvasState.database =
shared.MapDataObjectDB.MapDataObjectDBFromObject(simpleDB);

console.log("DB read complete.");

// Set up canvas on canvasState

let squareLength = 1000;
canvasState.canvas = canvasLib.createCanvas(squareLength, squareLength);
canvasState.ctx = canvasState.canvas.getContext('2d');

// let zoomLevels = [0.05, 0.1, 0.2, 0.5, 1, 2, 3];

let zoomLevels = [0.5];

for (let i = 0; i < zoomLevels.length; i++) {
    canvasState.zoomLevel = zoomLevels[i];

    // Find maximum and minimum coords in database to set canvas bounds
    var minY;
    var minX;
```

```
    var maxY;
    var maxX;

    canvasState.database.getMapObjectsOfType("POINT").forEach(point => {
        if (minY == undefined || point.y < minY) {minY = point.y}
        if (minX == undefined || point.x < minX) {minX = point.x}
        if (maxY == undefined || point.y > maxY) {maxY = point.y}
        if (maxX == undefined || point.x > maxX) {maxX = point.x}
    });

    canvasState.yTranslation = -(minY * canvasState.zoomLevel);

    var currentSquareY = minY;

    while (-canvasState.yTranslation < maxY * canvasState.zoomLevel) {
        var currentSquareX = minX;
        canvasState.xTranslation = -(minX * 1.5 * canvasState.zoomLevel);
        while (-canvasState.xTranslation < maxX * canvasState.zoomLevel * 1.5)
{
            // Draw objects
            canvasState.ctx.clearRect(0, 0, squareLength, squareLength);
            canvasState.database.getMapObjectsOfType("AREA").forEach(area =>
area.draw());

            let acceptedPathTypes = canvasState.getPathTypes();

            canvasState.database.getMapObjectsOfType("PATH").forEach(path => {
                // Only plot line if is one of accepted path types for zoom
level
                if
(acceptedPathTypes.includes(path.metadata.pathType["second_level_descriptor"])
                ||
acceptedPathTypes.includes(path.metadata.pathType["first_level_descriptor"]))
                    path.plotLine(this);
            });

            // Write canvas to png
            let fileName =
`./mapAreaImages/${currentSquareX}x${currentSquareY}_${canvasState.zoomLevel}x.
png`;
            let buffer = canvasState.canvas.toBuffer();
            fs.writeFileSync(fileName, buffer);
            console.log(`Canvas has been written to "${fileName}".`);

            currentSquareX += squareLength * 1.5 * canvasState.zoomLevel;
            canvasState.xTranslation -= squareLength;
        }
        currentSquareY += squareLength * canvasState.zoomLevel;
```

```
        canvasState.yTranslation -= squareLength;
    }
}
```

src/shared/sharedStructures.js [excerpt]

```
/**
 * Used to define tiles to be drawn to screen.
 */
 shared.Tile = class Tile extends this.shared.MapDataObject {
    x
    y
    lengthX
    lengthY
    zoomLevel
    element

    constructor(x, y, lengthX, lengthY, zoomLevel) {
        super();
        this.x = x
        this.y = y
        this.lengthX = lengthX
        this.lengthY = lengthY
        this.zoomLevel = zoomLevel
    }

    get fileName() {
        return `${this.x}x${this.y}_${this.zoomLevel}x.png`;
    }

    static tileFromObject(object) {
        let tile = new shared.Tile(object.x, object.y, object.lengthX,
object.lengthY, object.zoomLevel);
        tile.ID = object.ID;
        tile.metadata = object.metadata;
        return tile;
    }
}
```

Testing

This section outlines tests carried out on the final version of my project to assess how it meets my planned objectives.

Testing video

Testing from the web app can be seen on YouTube at https://www.youtube.com/watch?v=HuN0bmUCLIY, from the perspective of an end user interacting with the application.

00:00 Server started from terminal

> Expected outcome: Server starts successfully.
>
> Remarks: This outcome is achieved, and takes 40 seconds to achieve, which does not result in delays for the end-user in a normal set-up, where the server is constantly available.

00:40 Map web app is started in browser, map data is then loaded, panned and zoomed to show Cambridge

> Expected outcome: Map data is loaded and can be easily panned and zoomed to show the city.
>
> Remarks: This outcome is successfully achieved, though does face noticeable latency.

01:30 Search for "museum" is initiated

> Expected outcome: Up to 20 results for Cambridgeshire museums are returned, where all points are shown in the resultant map view.
>
> Remarks: The outcome is successfully achieved, with search results appearing quickly.

01:50 Result is selected and "go to point" is pressed to automatically translate and zoom map to point - Map is then panned and zoomed again.

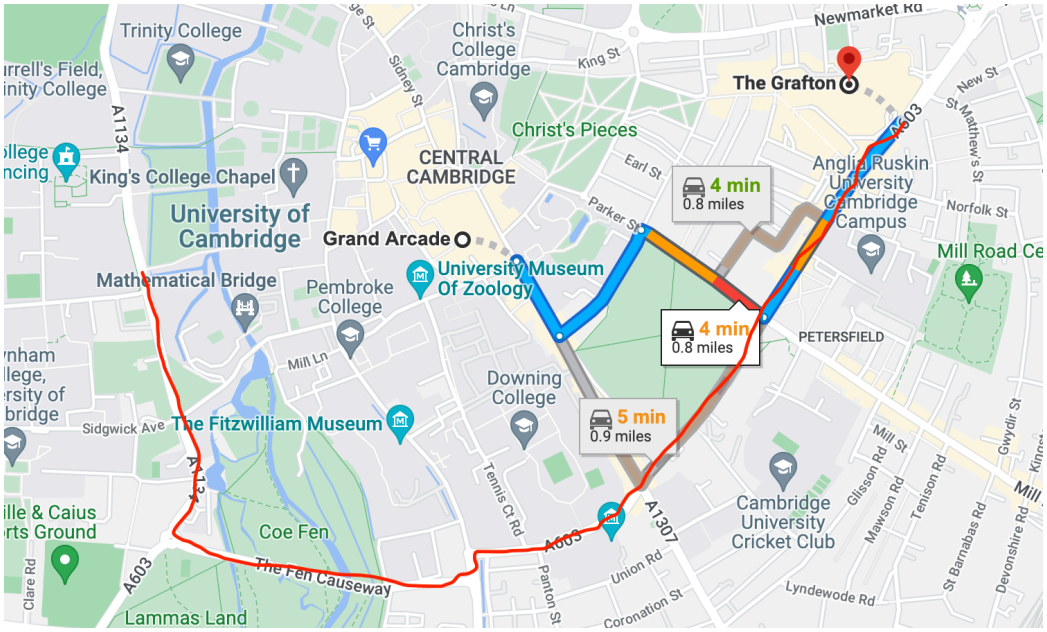
> Expected outcome: Result shows corresponding metadata and is successfully translated and zoomed to.
>
> Remarks: This is successfully achieved, where at this closer zoom level there is no noticeable latency and appears very smooth.

02:00 Route is then requested between "Grand arcade" and "the grafton"

02:30 Route is displayed, and the resultant view is panned and zoomed

> Expected outcome: The route is successfully displayed, translated and zoomed automatically to an appropriate zoom level, and can be panned and zoomed manually as expected.

Remarks: The found route is not particularly useful, as the starting point is by residential streets yet can only route via major roadways. A comparison between the generated route and that from Google Maps is displayed below, with my project's generated route shown in red.



02:45 Route find between two gibberish locations is initiated

Expected outcome: An error is displayed in the "find route" dialog.

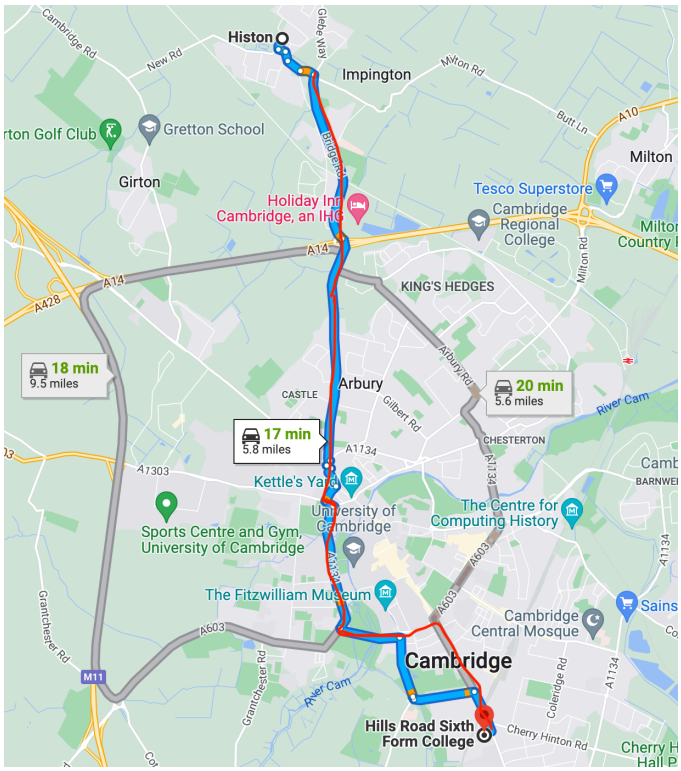
Remarks: This outcome is successfully achieved.

02:55 Route find between "Histon" and "Hills road sixth form college"

03:30 Route is displayed

Expected outcome: Route is successfully displayed.

Remarks: This outcome is successfully achieved, with the project displaying a useful and efficient route, due to the nearby main roads for both points. A comparison with Google Maps is again included below:



03:35 Search request for gibberish is initiated

Expected outcome: Error is displayed in search dialog.

Remarks: This outcome is successfully achieved.

Objective 1 (“Web client is able to send requests and receive appropriate data back to/from backend”)

Server successfully starts up

Action: Start server with command “node --max-old-space-size=16384 index.js” within src directory.

Expected output: The server outputs “Server has started at http://localhost:80” after loading database into memory and conducting indexing.

Actual output:

```
[Mapping project, 1646226818950] Logger has started.

[Mapping project, 1646226818953] Server is starting...

[Mapping project, 1646226818954] Loading db.

[Mapping project, 1646226860547] Logger has started.

[Mapping project, 1646226866401] Loaded db.

[Mapping project, 1646226866420] Server has started at http://localhost:80
```

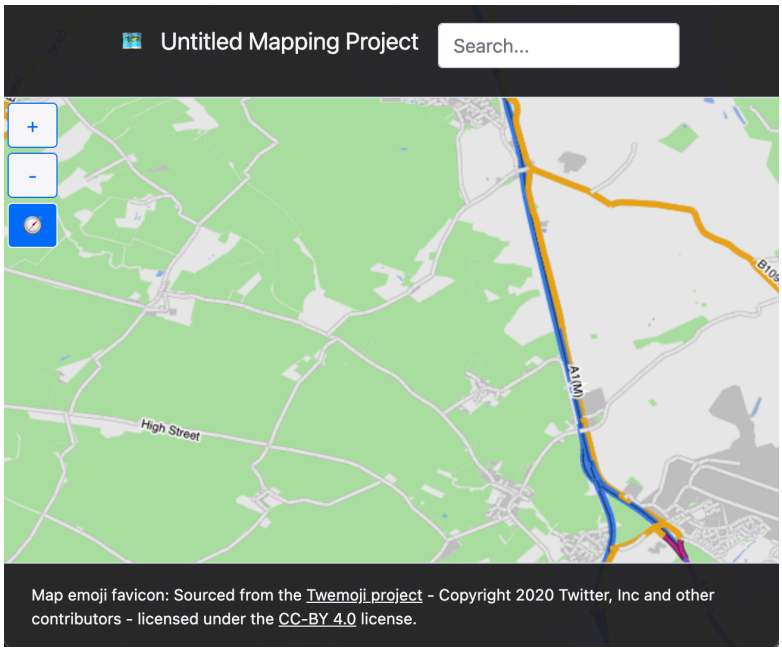
Web application successfully loads

Action: Load http://localhost in browser.

Expected output: Web application successfully loads a view of the map, and requests are logged in the server log.

Actual output:

Successfully loads, with no errors shown in developer console.




```
[Mapping project, 1646227062188] / was requested.

[Mapping project, 1646227062376] /index.css was requested.

[Mapping project, 1646227062399] /sharedStructures.js was requested.

[Mapping project, 1646227062401] /index.js was requested.

[Mapping project, 1646227062419] /spinner.gif was requested.

[Mapping project, 1646227062642]
/GetDBfromQuery?pathTypes=[%22motorway%22,%22primary%22,%22trunk%22,%22primary_
link%22,%22trunk_link%22,%22river%22,%22returned_route%22,%22secondary%22,%22se
condary_link%22,%22tertiary%22,%22tertiary_link%22,%22unclassified%22,%22reside
ntial%22]&area={%22x%22:-
2792.886051499995,%22y%22:6849.380475070653,%22height%22:2103.9999999999995,%22
width%22:3859.999999999999,%22pathTypeCount%22:13} was requested.

[Mapping project, 1646227062646] Logger has started.

[Mapping project, 1646227067916] /favicon.ico was requested.
```

Success.

Corresponding JSON data is returned for requests to GET /api/GetDBfromQuery

(Large map view containing all map data for specified highways)

Action: CURL initiated GET request to /api/GetDBfromQuery?pathTypes=[%22motorway%22,%22primary%22,%22trunk%22,%22primary_link%22,%22trunk_link%22,%22river%22]&&noMapAreaFilter=true

Expected output: Corresponding JSON map data in custom database format is returned to the client.

Actual output:

```
[peterwarrington@Peters-MacBook-Air ~ % curl "http://localhost/api/GetDBfromQuery]
?pathTypes=[%22motorway%22,%22primary%22,%22trunk%22,%22primary_link%22,%22trunk
_link%22,%22river%22]&&noMapAreaFilter=true" -g -o test1.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 82.6M  100 82.6M    0     0  25.6M      0  0:00:03  0:00:03 --:--:-- 25.7M
```

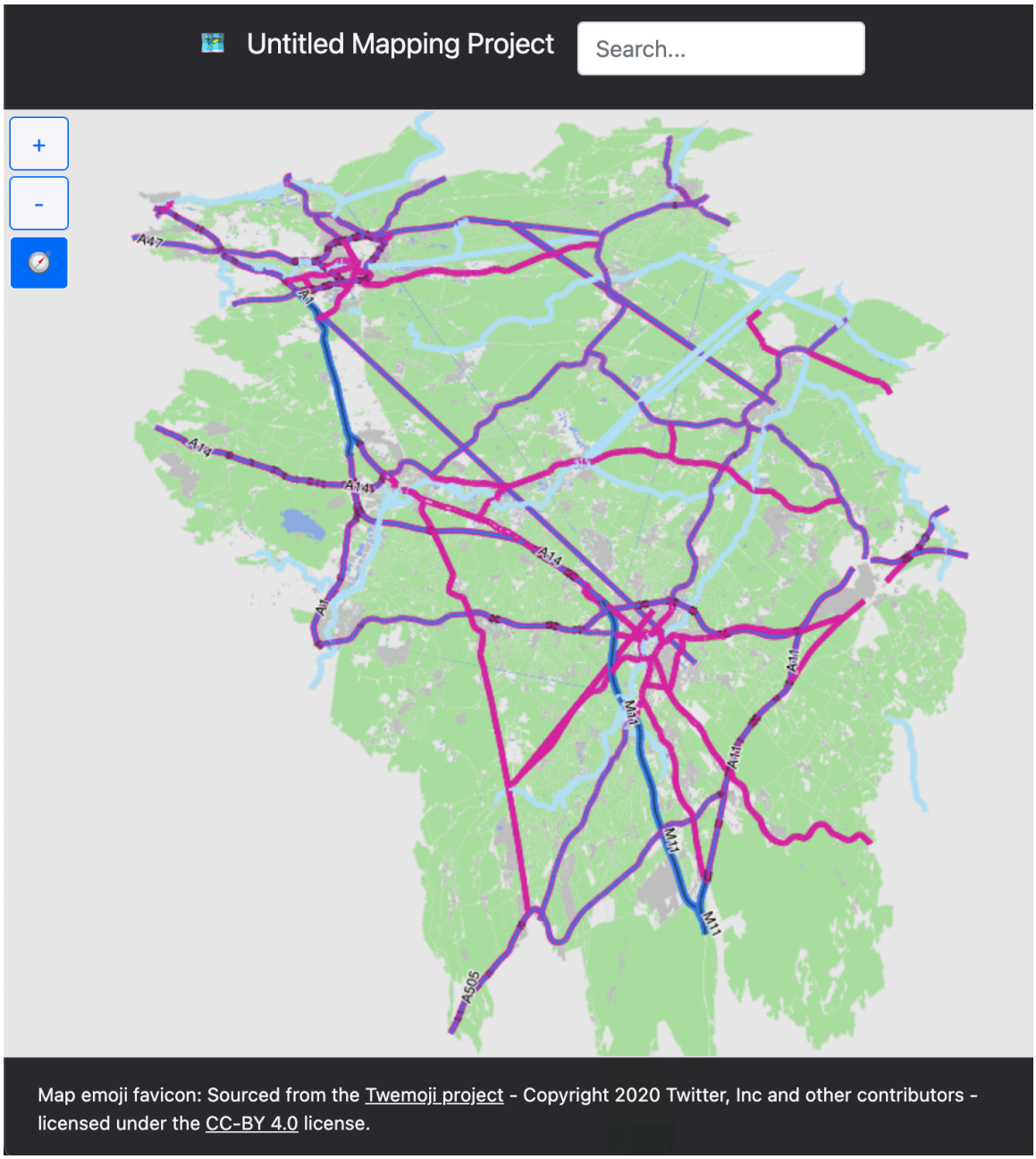
Success.

Action: Force the client to download from this URL by hardcoding it to request it.

Expected output: All map data is loaded and displayed on the client

Actual output:

(After zoomed out)



Success.

(Small map view containing map data specified by bounds)

Action: CURL initiated GET request to /api/GetDBfromQuery?pathTypes=[%22motorway%22,%22primary%22,%22trunk%22,%22primary_link%22,%22trunk_link%22,%22river%22,%22returned_route%22,%22secondary%22,%22secondary_link%22,%22tertiary%22,%22tertiary_link%22,%22unclassified%22,%22residential%22]&area={%22x%22:-2792.886051499995,%22y%22:6849.380475070653,%22height%22:2135.999999999995,%22width%22:1961.9999999999998,%22pathTypeCount%22:13}

Expected output: Corresponding JSON data.

Actual output:

```
peterwarrington@Peters-MacBook-Air ~ % curl "http://localhost/api/GetDBfromQuery
?pathTypes=[%22motorway%22,%22primary%22,%22trunk%22,%22primary_link%22,%22trunk
_link%22,%22river%22,%22returned_route%22,%22secondary%22,%22secondary_link%22,%
22tertiary%22,%22tertiary_link%22,%22unclassified%22,%22residential%22]&area={%2
2x%22:-2792.886051499995,%22y%22:6849.380475070653,%22height%22:2135.99999999999
95,%22width%22:1961.9999999999998,%22pathTypeCount%22:13}" -g -o test2.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 11.3M  100 11.3M    0     0  5460k      0  0:00:02  0:00:02 --:--:-- 5473k
```

Success.

Action: Map initiated GET request to same URL.

Expected output: Map data displayed on map.

Actual output:

Same screenshot as in first test – Success.

Objective 2 ("The backend is able to read from OpenStreetMap data and return this to clients via this API")

convertFromOSM.js on OSM data for Cambridgeshire

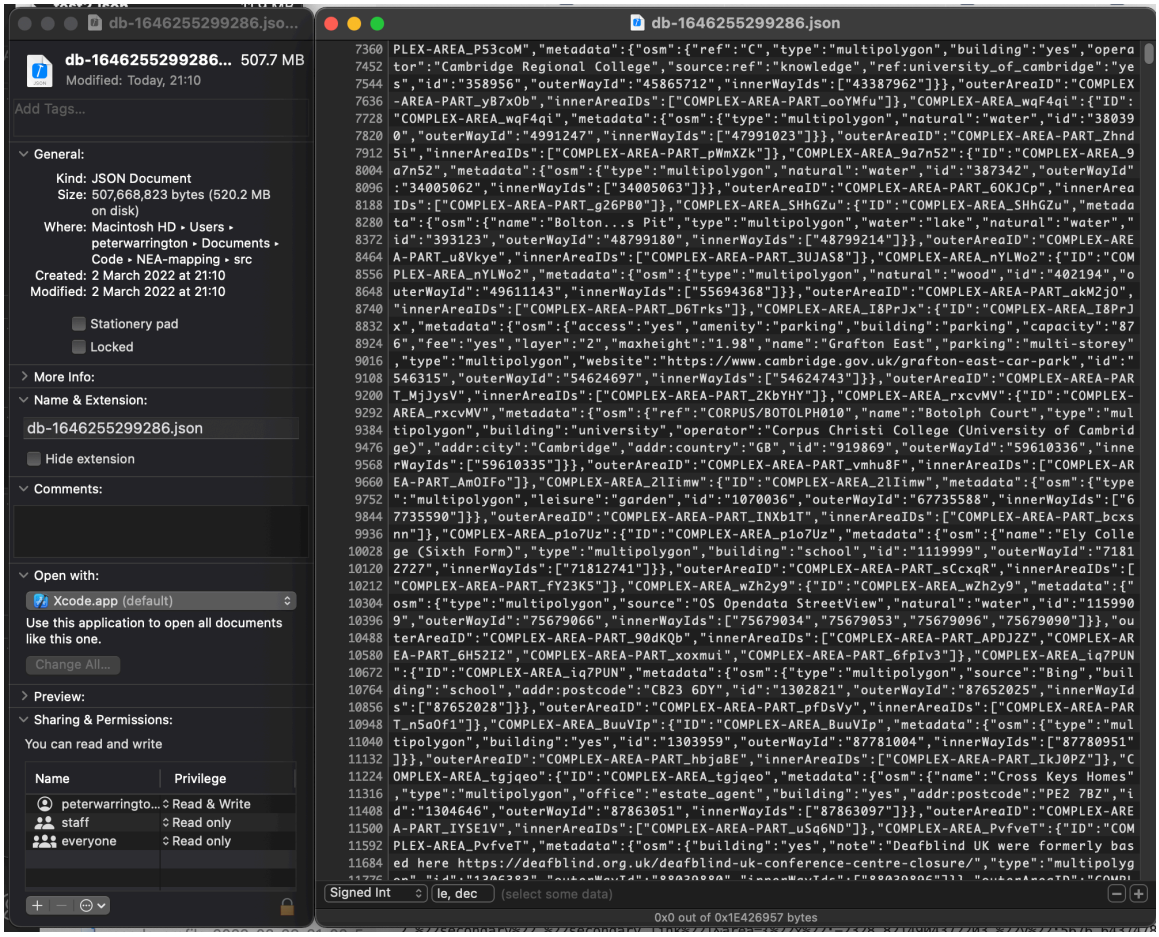
Action: Initiate convertFromOSM.js from command line using "node --max-old-space-size=16384 convertFromOSM.js", and with data contained in docs/cambridgeshire-latest.osm (the path to this file is hardcoded into convertFromOSM.js).

Cambridgeshire OSM data is sourced from https://download.geofabrik.de/europe/great-britain/england/cambridgeshire.html#:~:text=cambridgeshire%2Dlatest.osm.bz2%2C%20yields%20OSM%20XML%20when%20decompressed

Expected output: The process outputs a success message, producing a JSON database file that can then be read by the server.

Actual output:

```
Beginning conversion from OSM to OSM-JS - This might take a while...
     OSM to OSM-JS conversion complete.
Unable to cache OS-JS data. RangeError: Invalid string length. Continuing
anyway...
Extracting ways, nodes and relations...
     Way, node and relation extraction complete.
Extracting ways...
     Way extraction complete.
Generating DB...
100% (444278/444278) of ways added to db.b.
Processing complex areas...
100% (653/653) of complex areas processed.d.
Complex areas processed.
Adding all nodes to db...
100% (2516313/2516313) of nodes added to db.b.
Added all nodes to db...
Database generated. Writing to file...
100% (4911617/4911617) database items written to file.e.
Map database has been saved to db-1646255299286.json!
Background process: Nodes have been cached
Background process: Ways have been cached
```



The resultant map database was loaded in for use in the other tests, as shown in their respective screenshots.

The “unable to cache OSM.js data” error means that the generated JS object representation of the raw XML data was too large to be cached to file, something that is not crucial to the script, but can make run time faster when re-processing map data, for example when adding features during development. My objectives are therefore met.

Success

Objective 3 (“The web client is then able to display and represent data returned from the server on a map that can be panned and zoomed”)

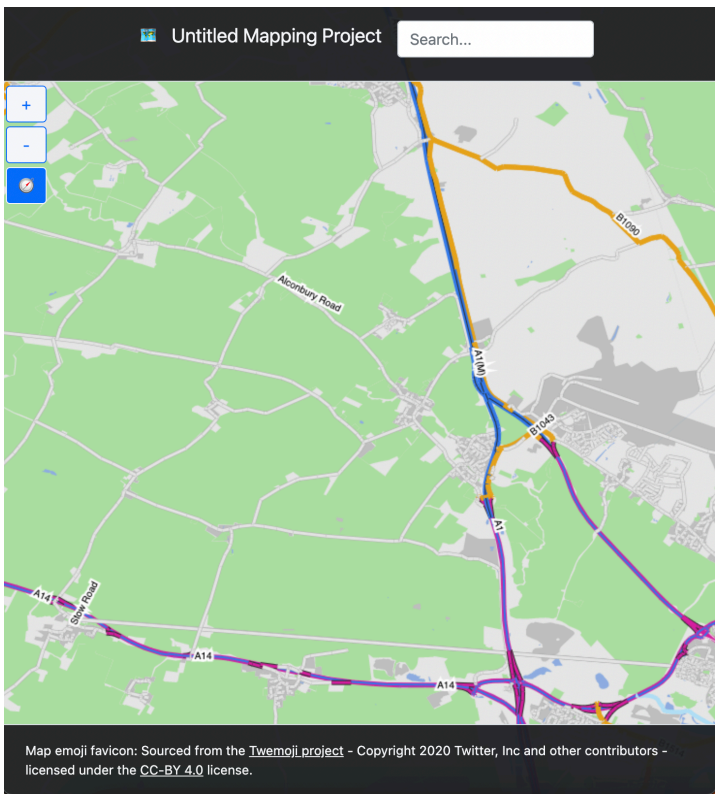
Pan without zoom to map extreme

Action: Pan map from top left initial position to bottom right without zooming, using Cambridgeshire map data.

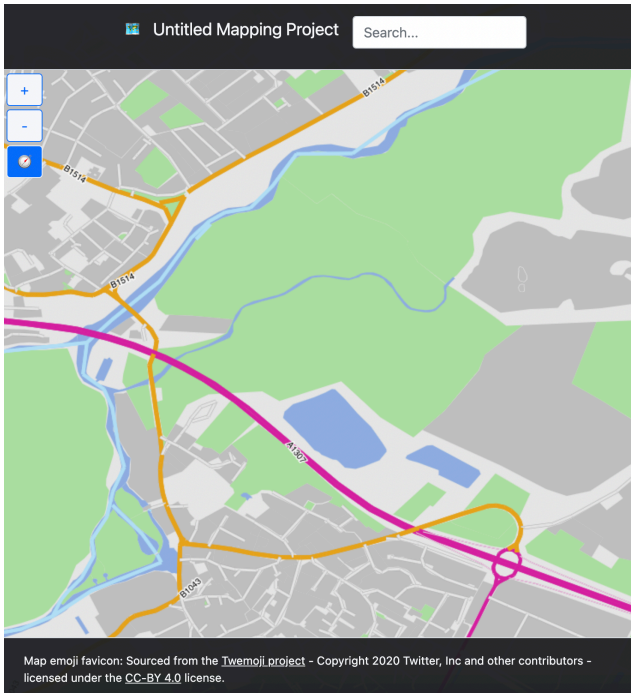
Expected output: Map is successfully panned, progressively loading map data which is displayed on screen.

Actual output:

Initial position



Mid-position



Final position

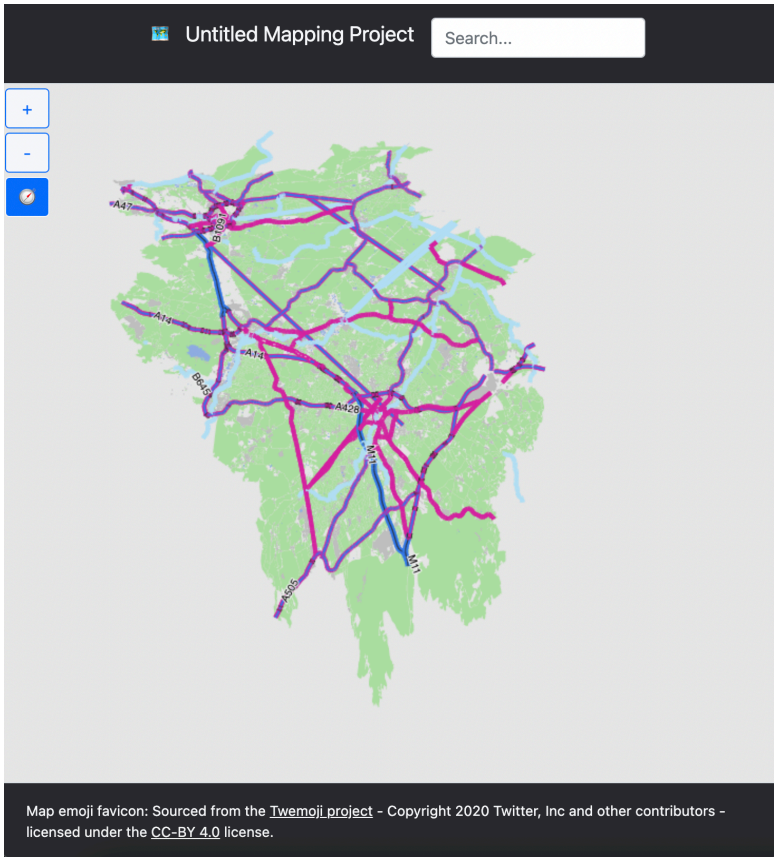
Success.

Zoom out to show whole of map

Action: From initial position, zoom out so that the entire map is displayed in the view.

Expected output: Map is successfully displayed, with only the most major of highways displayed.

Actual output:



Success

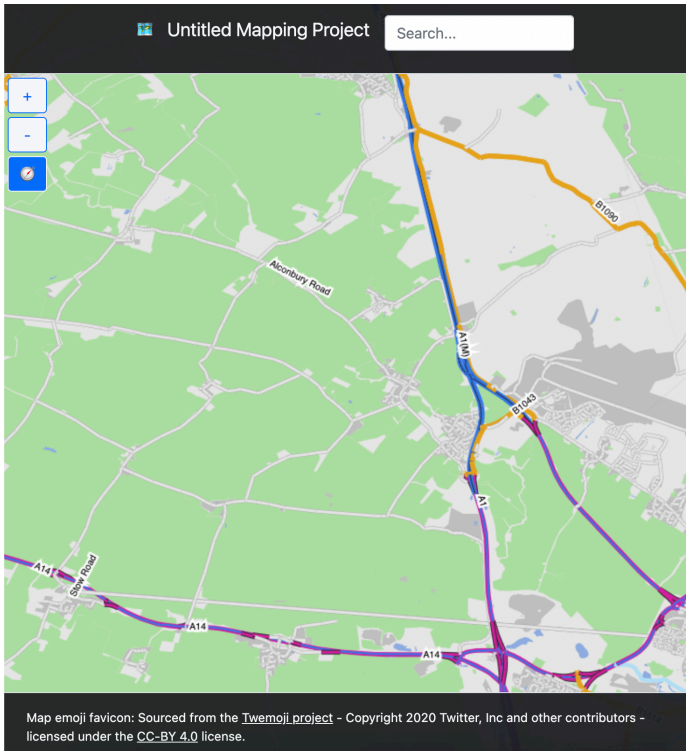
Zoom in from initial position

Action: From initial position, zoom in until no discernible map features can be found. Then zoom out back to initial zoom level.

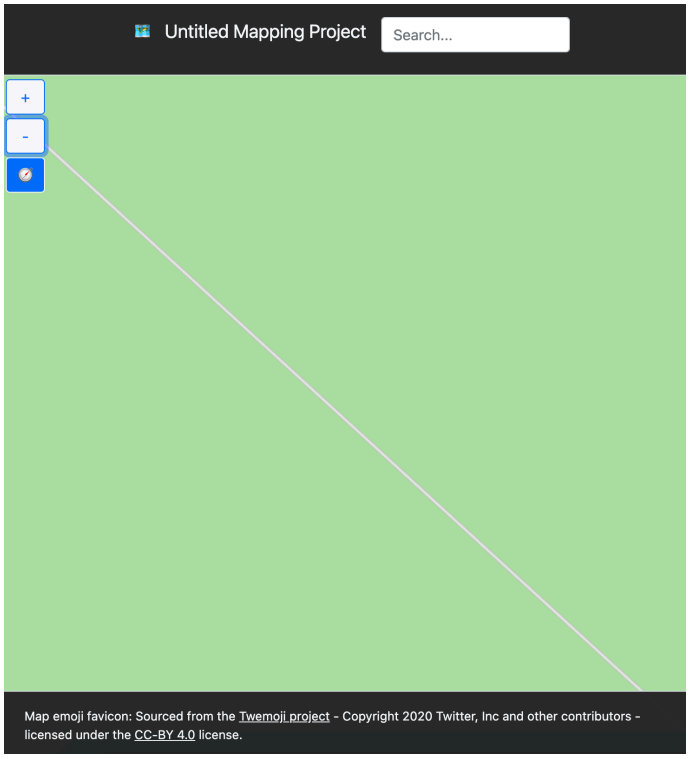
Expected output: Map successfully displays, though only the background of the map area zoomed into.

Actual output:

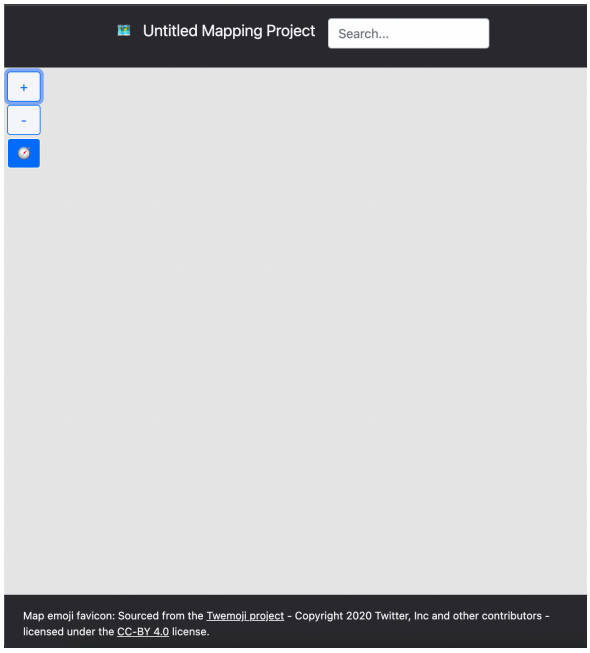
Initial



Partial zoom in

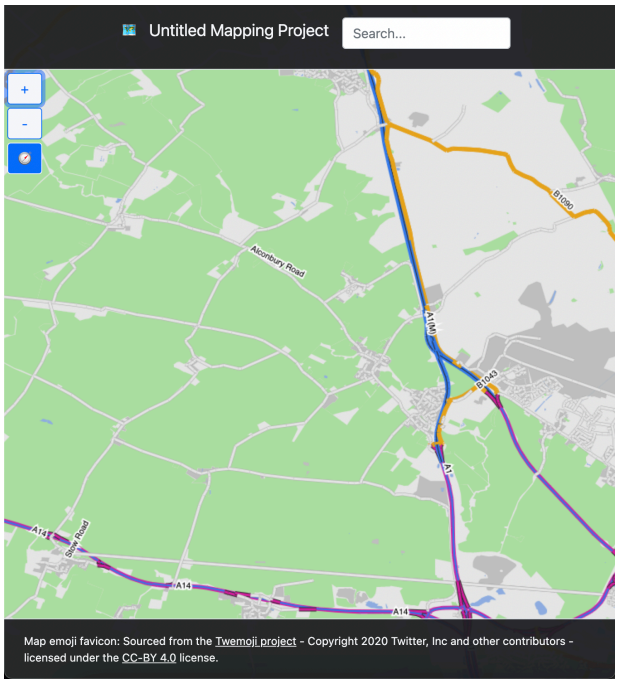


Zoom in until no map features



Map area colour is lost, but this is only a minor error.

Zoom back out to initial zoom level



Partial success.

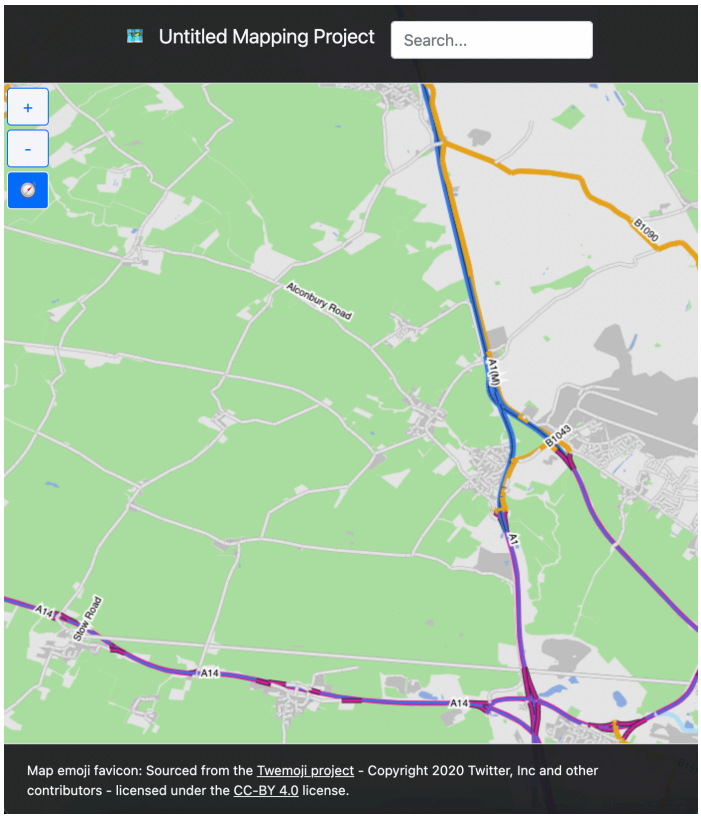
Zoom in, pan, zoom out

Action: Zoom in so that map features are still displayed, pan into Cambridge, then zoom out to show the entire city.

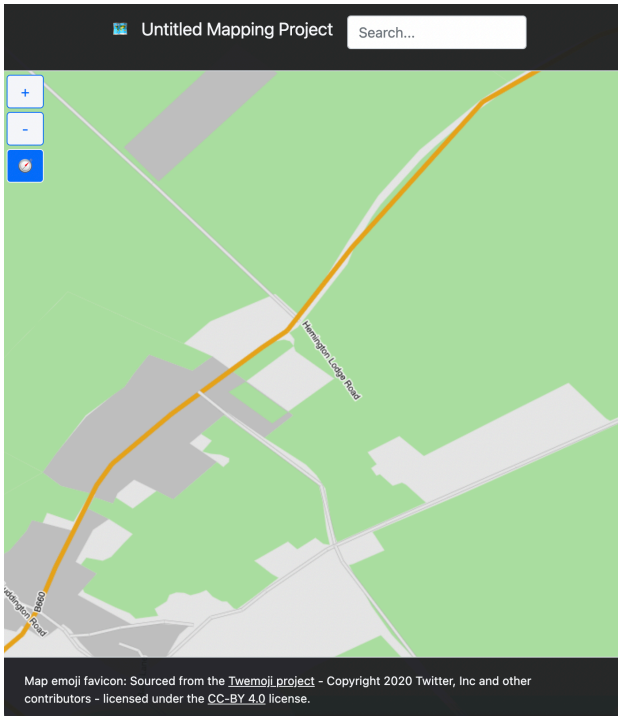
Expected output: Map is successfully displayed at all points

Actual output:

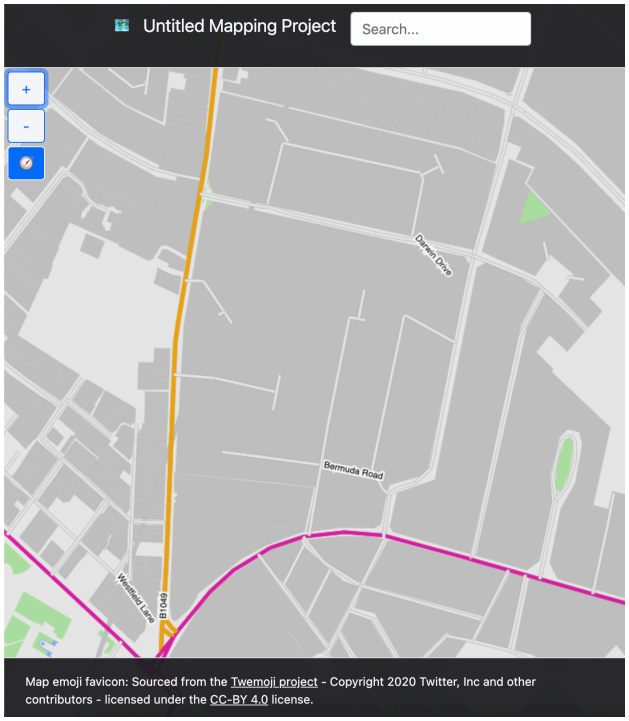
Initial



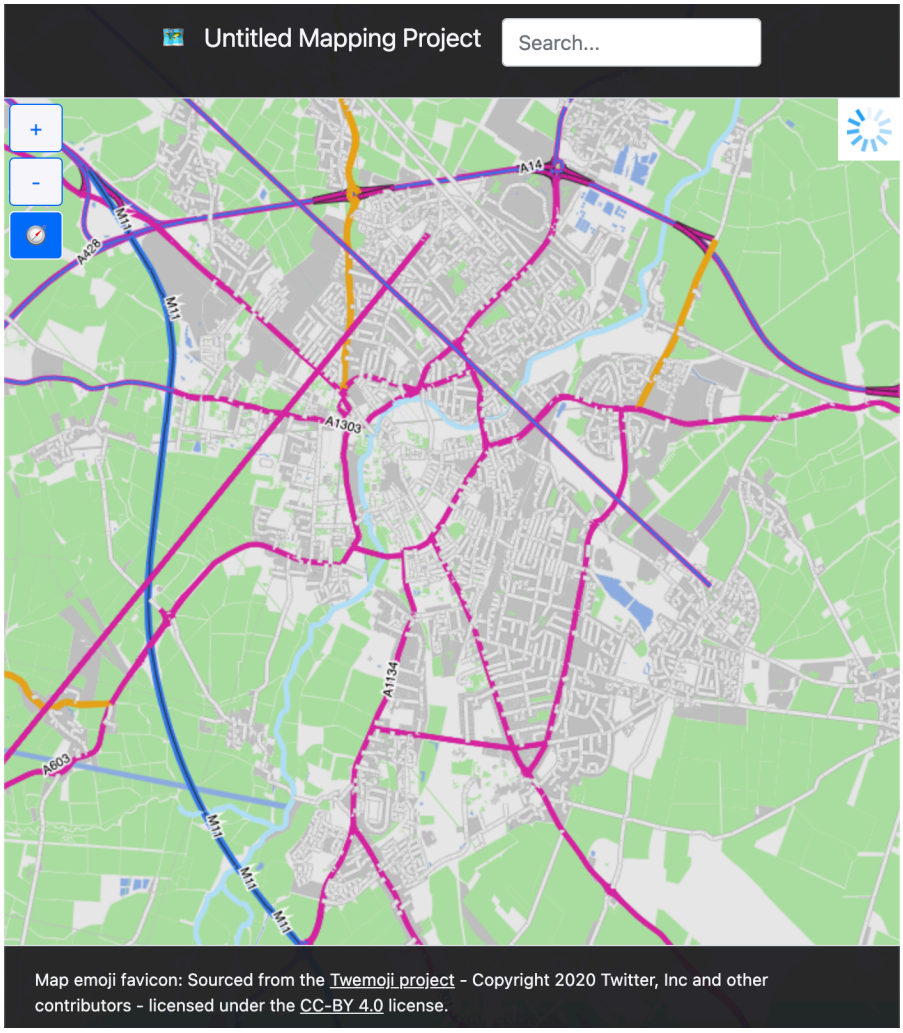
Initial with zoom



Panned from same zoom level into Central Cambridge



Zoomed out from central Cambridge



Success.

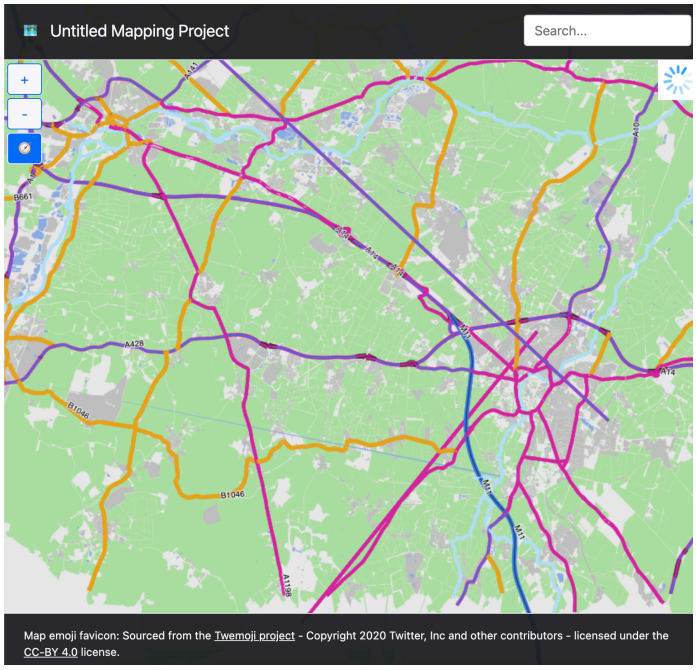
Zoom out, pan, zoom in

Action: Zoom out so that whole of Cambridge is displayed, pan to Cambridge, zoom in

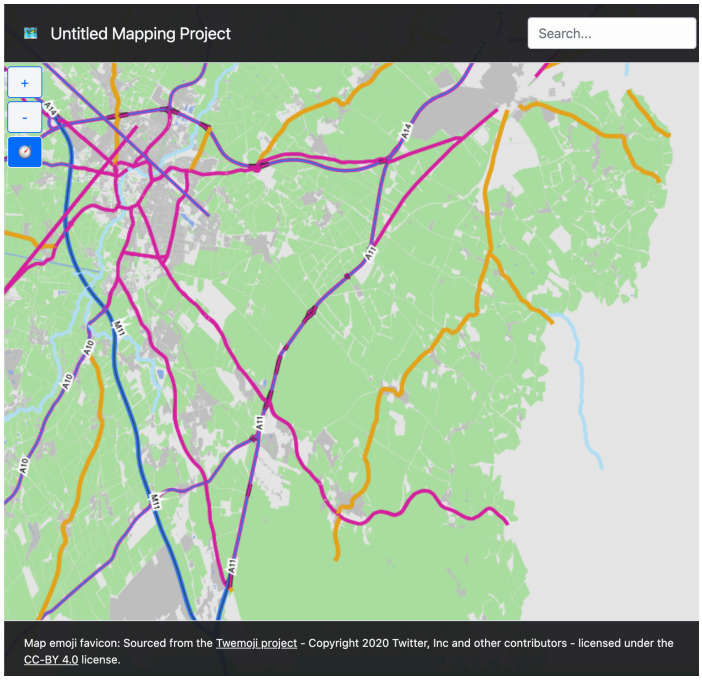
Expected output: Map data is displayed correct

Actual output:

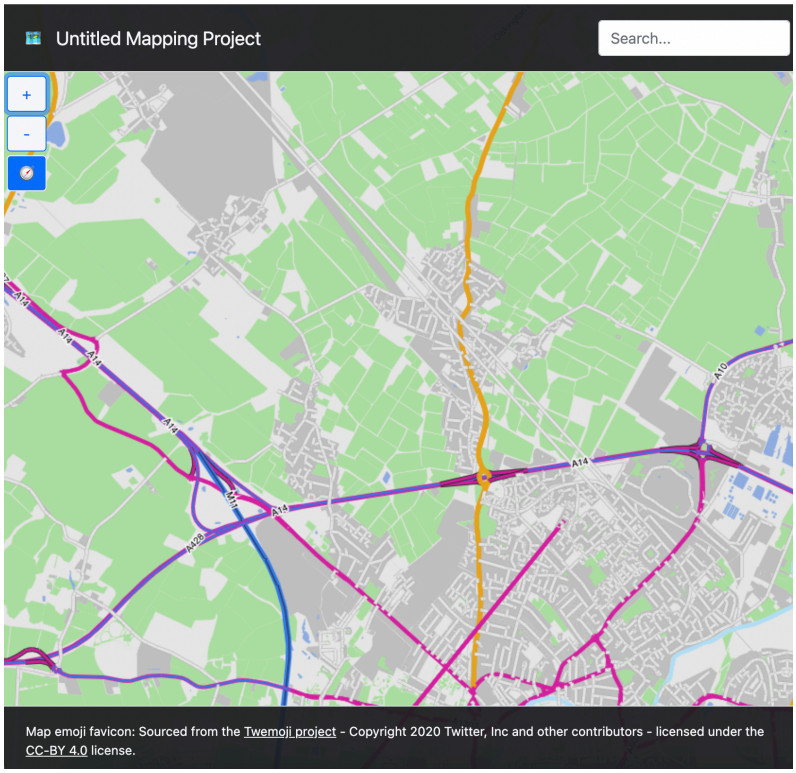
Zoom out


Untitled Mapping Project
Search...
Map emoji favicon: Sourced from the Twemoji project - Copyright 2020 Twitter, Inc and other contributors - licensed under the CC-BY 4.0 license.

Pan


Untitled Mapping Project
Search...
Map emoji favicon: Sourced from the Twemoji project - Copyright 2020 Twitter, Inc and other contributors - licensed under the CC-BY 4.0 license.

Zoom in



Success

Objective 4 (“A search function is available on this web client to be able to find locations using the API”)

API request to /api//PointSearch, searching for “museum”

Action: CURL initiated GET request to /api/PointSearch?searchTerm=%22museum%22

Expected output: An array of up to 20 corresponding points and their associated search confidences.

Actual output:

```
peterwarrington@Peters-MacBook-Air Downloads % curl "http://localhost/api/PointS
earch?searchTerm=%22museum%22" -g -o test3.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  6856  100  6856    0     0   2627      0  0:00:02  0:00:02 --:--:--  2634
```

Success

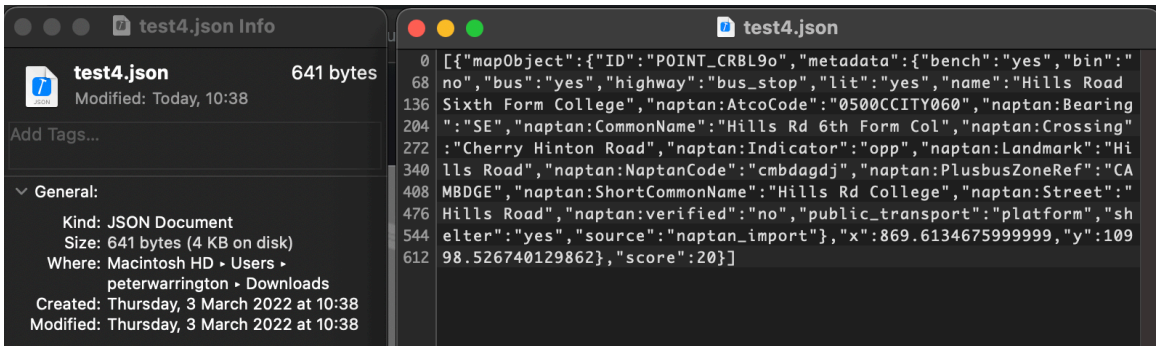
API request to /api/PointSearch for a specific point ID

Action: CURL initiated GET request to
“/api/PointSearch?searchTerm=%22POINT_CRBL9o%22”

Expected output: An array containing one map object and its associated search confidence will be returned. The map object should be the point with ID “POINT_CRBL9o” (Hills Road Sixth Form College).

Actual output:

```
[peterwarrington@Peters-MacBook-Air Downloads % curl "http://localhost/api/PointS
earch?searchTerm=%22POINT_CRBL9o%22" -g -o test4.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   641  100   641    0     0    251      0  0:00:02  0:00:02 --:--:--   251
```



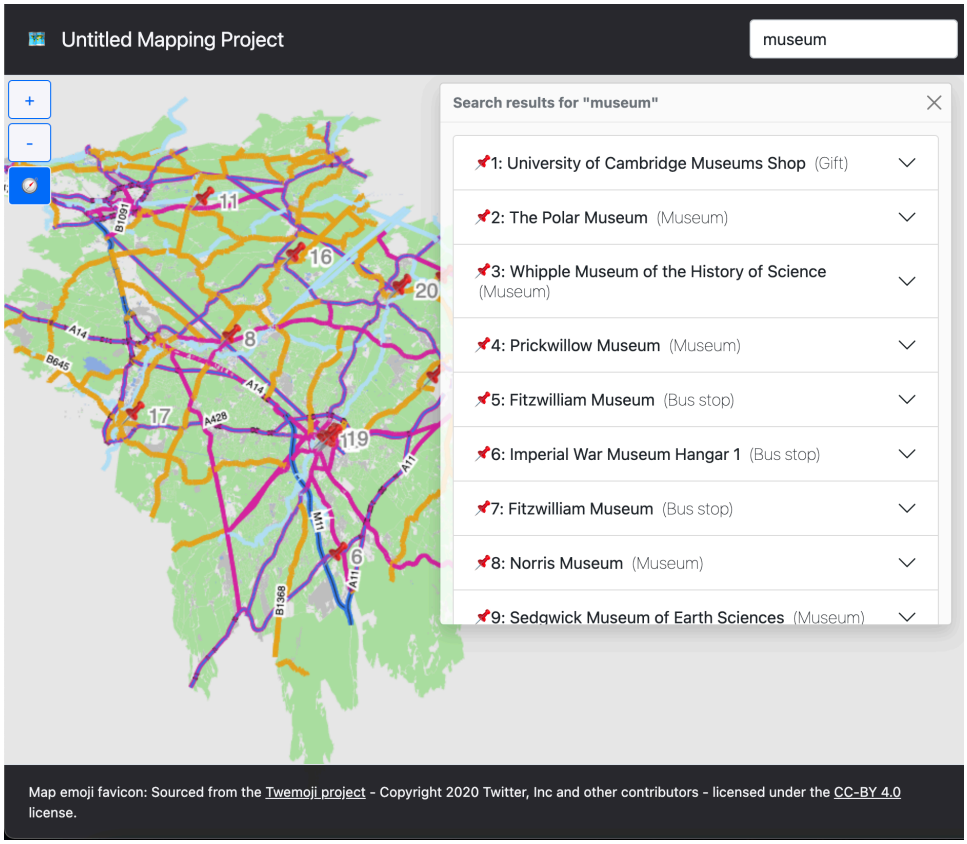
Success.

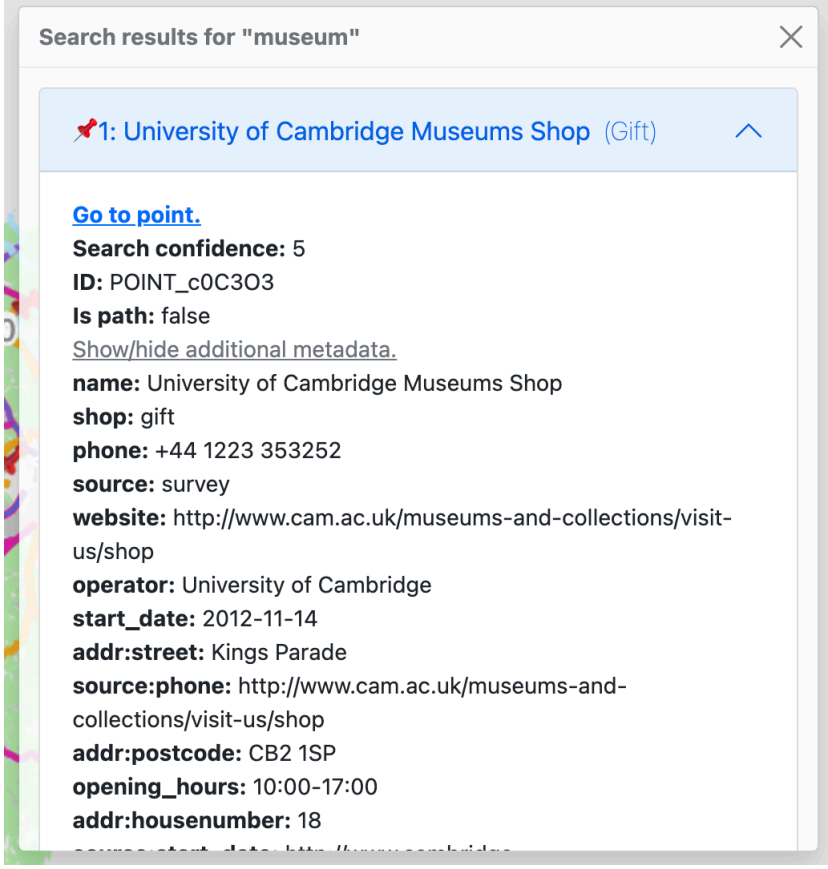
Searching for “museum” via web app

Action: Search for “museum” using web app search function.

Expected output: The map translates and zooms to show a view showing all points, with a displayed list of results and associated metadata.

Actual output:




Search results for "museum"
1: University of Cambridge Museums Shop (Gift)
Go to point.
Search confidence: 5
ID: POINT_c0C3O3
Is path: false
Show/hide additional metadata.
name: University of Cambridge Museums Shop
shop: gift
phone: +44 1223 353252
source: survey
website: http://www.cam.ac.uk/museums-and-collections/visit-us/shop
operator: University of Cambridge
start_date: 2012-11-14
addr:street: Kings Parade
source:phone: http://www.cam.ac.uk/museums-and-collections/visit-us/shop
addr:postcode: CB2 1SP
opening_hours: 10:00-17:00
addr:housenumber: 18

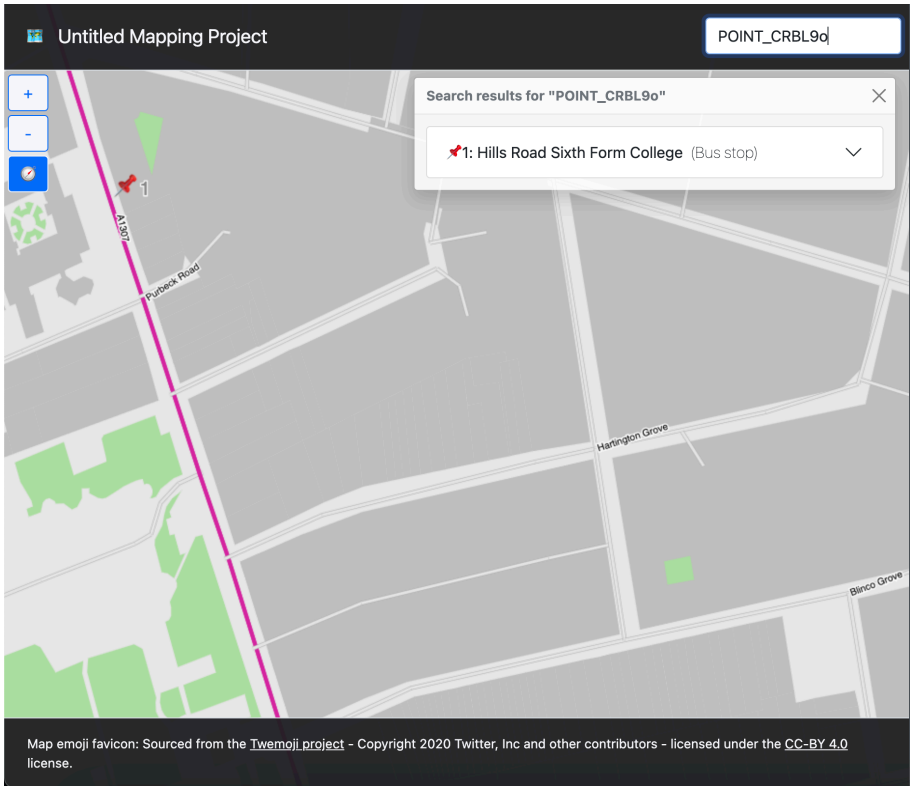
Success.

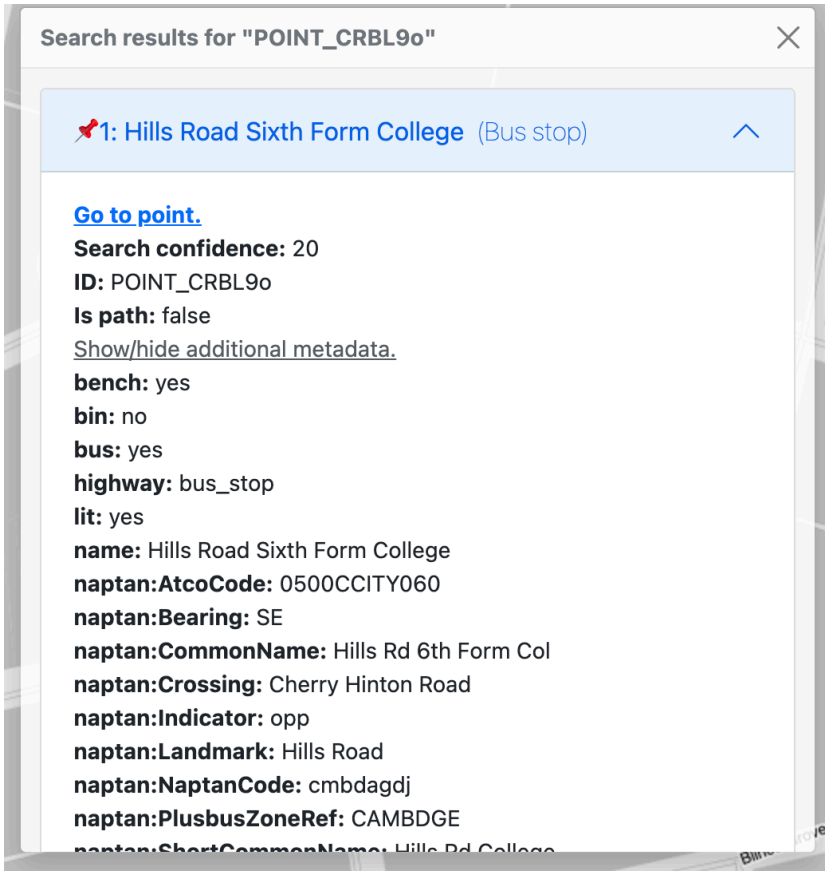
Search for specific point via web app

Action: Search for “POINT_CRBL9o” using web app search function

Expected output: Single point displayed and zoomed onto on map, with single result detail displayed.

Actual output:




Search results for "POINT_CRBL9o"
1: Hills Road Sixth Form College (Bus stop)
Go to point.
Search confidence: 20
ID: POINT_CRBL9o
Is path: false
Show/hide additional metadata.
bench: yes
bin: no
bus: yes
highway: bus_stop
lit: yes
name: Hills Road Sixth Form College
naptan:AtcoCode: 0500CCITY060
naptan:Bearing: SE
naptan:CommonName: Hills Rd 6th Form Col
naptan:Crossing: Cherry Hinton Road
naptan:Indicator: opp
naptan:Landmark: Hills Road
naptan:NaptanCode: cmbdagdj
naptan:PlusbusZoneRef: CAMBDGE

Success.

Objective 5 (“A function will be available to find an efficient route from one location on the map to another using path theory. This will take into account distance and this route must be traversable in the real world.”)

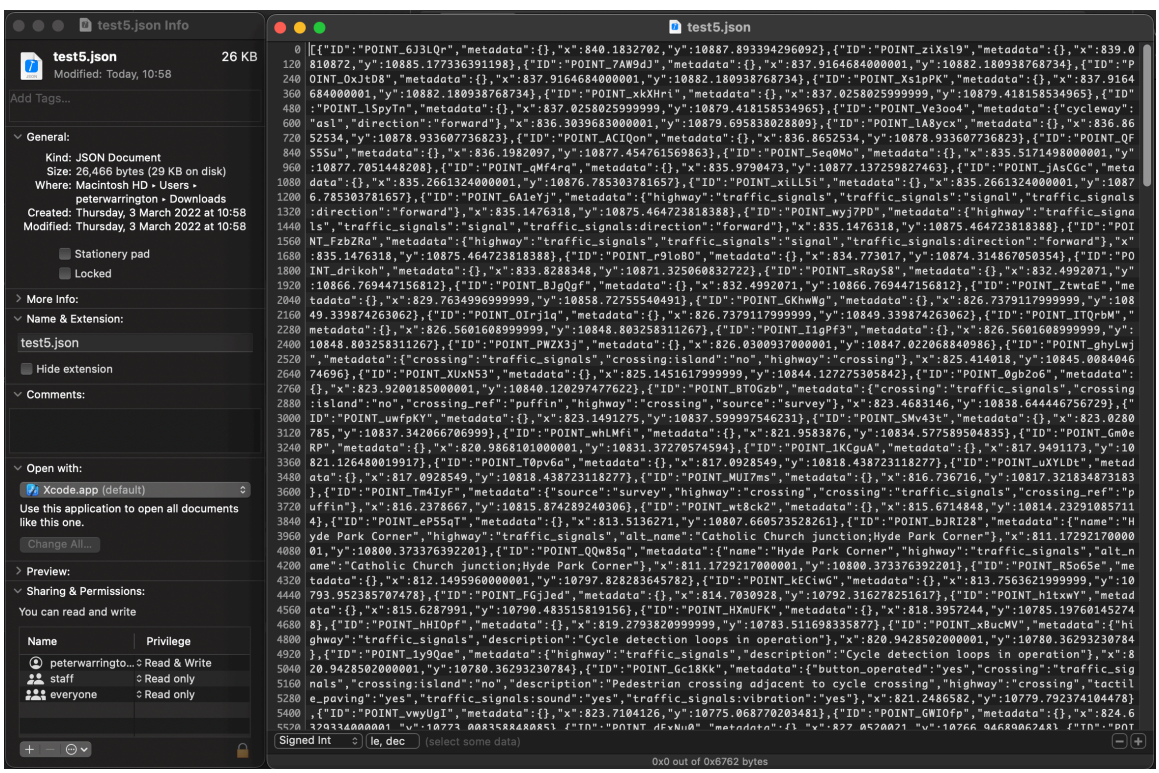
API request to /api/FindRoute searching between “Cambridge station” and “Cambridge north”

Action: CURL initiated get request to /api/FindRoute?startingPointTerm=%22Cambridge%20station%22&destinationPointTerm=%22Cambridge%20north%22

Expected output: An array of points that make up the route.

Actual output:

```
peterwarrington@Peters-MacBook-Air Downloads % curl "http://localhost/api/FindRoute?startingPointTerm=%22Cambridge%20station%22&destinationPointTerm=%22Cambridge%20north%22" -g -o test5.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 26466  100 26466    0     0   1555      0  0:00:17  0:00:17 --:--:--  7087
```



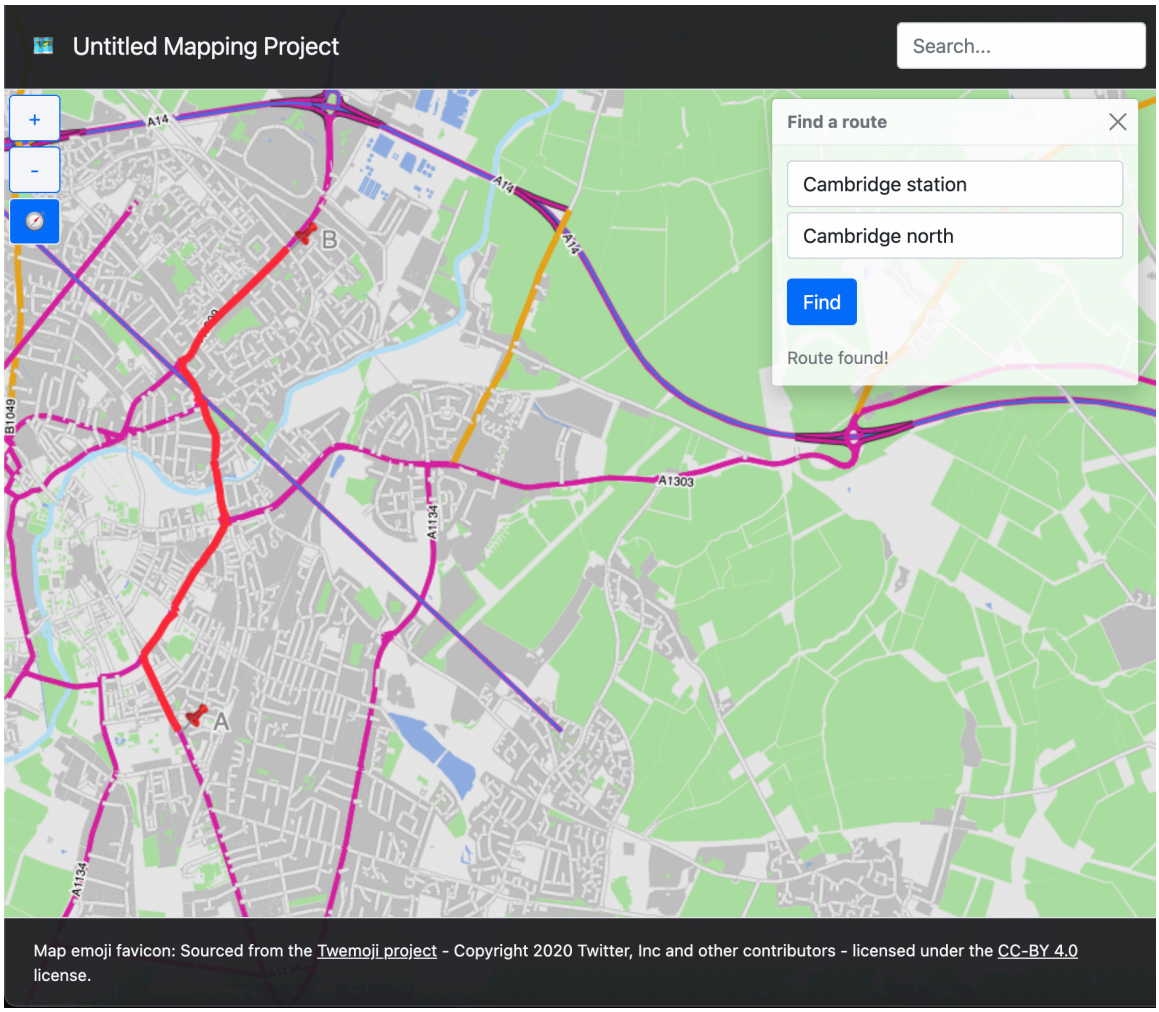
Success, though slow as finding a route from a set of thousands of nodes.

Search for route between “Cambridge station” and “Cambridge north” via web app

Action: Search for route between “Cambridge station” and “Cambridge north” via web app routing function.

Expected output: Map translated and zoomed to display route styled on map in red, success status message displayed.

Actual output:



Success, though spent same amount of time finding route as CURL request.

Objective 6 (“Function serviced by database on the backend to allow a user to sign-in/register and save places/routes”)

Objective was not implemented due to time restraint, and to focus on the complexity of the rest of the project that was perhaps underestimated.

Objective 7a (“Testers reports that they are able to easily find their home using the search function”)

Action: With an explanation that searching only takes place at street level (no house numbers), I tasked the testers with finding their house given the initial map view of the app.

Expected outcome: Users are successfully able to quickly locate their street and their house after.

Actual outcome:

Tester A

User was not able to find their street as out of the 20 results for their road, none were their own. User reported that the map was “laggy” and that map makers could be stylised.

Tester B

User was able to find their street from the list of results and identify it on the map. They reported that the application was intuitive and easy to use, and at their zoom level did not identify any lag.

Tester C

User was easily able to find their street and identify their house on the map, stating that it was convenient to do so in an accessibly presented way.

Objective 7b (“Testers reports that they are able to easily generate a route”)

Action: Testers were tasked with finding a route between their home and a point of their choosing, with the same disclaimer about searching only at street level.

Expected outcome: Users are successfully able to generate a route.

Actual outcome:

Tester A

User was able to easily find a route between their house and their college, reporting that it was accurate and the route they in fact take to college.

Tester B

This user was able to easily find a route between their house and their college, reporting that it was easy, though they noted it only routed via the main road nearest to their house, not via their residential road.

Tester C

The user was able to plot a route between two nearby points, but this was not routed through residential roads like was appropriate for this purpose, instead the nearby primary roads to both points, and therefore was not useful for this application.

Objectives 7c and 7d ("Testers report that they are able to easily register and log in, save their location and route to their account")

These features were not implemented in the final project as explained previously.

Other objectives are related to the technical design of the project therefore cannot be assessed through testing.

Evaluation

Overall, I believe I have created a mapping project that successfully meets the main objectives of a web-based mapping application, providing much of the map display, search, and routing features that users generally expect from a mapping service, in an intuitive and simple interface. Compared to mainstream services like Google maps however, it does not offer the same level of functionality, with routing being slow and only being able to be conducted on major roads, while map translation and zooming at higher zoom levels generally faces high latency due to the lack of map tiling. More generally however, I believe my project has made effective and wide-ranging use of object-oriented data structures throughout the whole project and has benefitted from the use of a custom database and caching implementation and design. The use of these data structures in this way made building on code iterations over the course of development easier, with many of the main data structures having been established in the first prototype.

Evaluating against set objectives

1. **Web client is able to send requests and receive appropriate data back to/from backend**

An API has been created to allow transfer of data between client and server over HTTP.

2. **The backend is able to read from OpenStreetMap (OSM xml) data and return this to clients via this API**

The convertFromOSM.js script handles this process and is able to import reliable and comprehensive map data for Cambridgeshire to the project's custom database format, though struggles with some other OSM data sets, unable to handle those above 512mb.

3. **The web client is then able to display and represent data returned from the server on a map that can be panned and zoomed**

This objective has been achieved, with colour coded map features such as land usage, water bodies and roads successfully displayed, but while panning and zooming can be slow and laggy at higher zoom levels containing lots of map data.

4. **A search function is available on this web client to be able to find locations using the API**

A search function is provided accessible via the web app, which can search through all points and paths using their associated OSM metadata. The search algorithm I have designed isn't perfect, and isn't available at house level, but is generally reliable and fast, with relevant results sorted via an index ranking system.

5. **A function will be available to find a efficient route from one location on the map to another using path theory. This will take into account distance and this route must be traversable in the real world.**

This was achieved, where the application can return a route via main roads between two points using Dijkstra's algorithm. This is however slow and is not suitable for small distances best connected by residential and other minor roads, or where points are too far away from a main road for output to be produced or be useful.

    5. **a. Extension: Take into account traffic, whether that be based on a custom specified metric or the location of traffic lights along a route**

    This was not achieved due to the lack of available traffic data and time constraints

6. **Function serviced by database on the backend to allow a user to sign-in/register and save places/routes**

This was not achieved due to time constraints and underestimated complexity of working on other functionality.

7. **Testers reports that they are able to easily:**

    a. Find their home using the search function

    b. Generate a route

    c. Register and log-in

    d. Save this location and route to their account

    See testing section.

8. **Data structures and important algorithms are well-documented using comments**

Data structures and algorithms are generally commented, with the most documentation given to those structures and algorithms that are most confusing to follow without comments. Commenting therefore can vary throughout the code base.

9. API is well documented

API is not documented other than in the code, with it generally being simple to understand given the list of possible API endpoints in /src/index.js, by referencing the web app implementation, and by examining specific API implementation files to understand parameters.

10. Effective use of object-oriented programming to be able to create effective data structures shared across the project, including the appropriate use of inheritance.

Object-oriented objects and classes are used extensively throughout my project, with inheritance widely used, most notably to implement different map objects. Map objects all inherit from a base map object class which can be added to the database, where the parts that make up complex areas additionally inherit from the Area class. These are clearly organised

11. Normalised database structure to store the data mentioned above.

The database system used by my project has been custom implemented using hard coded tables, but which offers fast and efficient access, including for iteration, with the use of primary and foreign keys much like in a traditional database. I took care to make sure I met the 3rd normal form when do this, to ensure a lack of data redundancy.

Extension:

12. Cache the most frequently accessed places directly on the database

13. Function to be able to plot new data on map that is then made available

14. Ability to add notes to the map

15. Ability to add other annotations to the map

Extension objectives were not met due to time limitations and underestimation of the complexity of other objectives as explained previously.

Possible improvements to project

From the feedback I have received during testing, it is clear that the routing function would be the focus of my attention in building any improvements. In its current implementation it is slow (up to a minute at some distances), and limited in the routes that can be generated, only using main roads at present in order to reduce the number of nodes and therefore run time.

In order to improve this, I would try two main things. One of these is to use an algorithm to reduce the number of nodes on the paths used to determine routes, for example using the Ramer–Douglas–Peucker algorithm, via a process known as cartographic generalization.[12] This would also decrease latency when translating across the map where applied to the client-side, given less processing for every frame has to take place as a result. The other method I could use is to use an alternative path-finding algorithm, such as A*, which uses heuristics to make path finding more efficient.

Another issue has been latency when translating and panning the map, where the client must redraw every single vertex on screen, something that takes a noticeable amount of time when more zoomed out and more are displayed. A solution to this would be to use map tiling so that all views of the map are generated in advance and stored on the server, to be quickly downloaded by the client, without the extensive mathematical calculations that would normally be needed, simply displaying images to the screen. I began an attempt at this during the project as explained previously but was not able to complete this.

In addition to these improvements, I would also work to meet my extension goals and implement user authentication as described in my objectives, to build those features that allow users to easily exercise the control that a self-hosted mapping solution uniquely offers, with the ability to add annotations and draw map features to the map.

[12] https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm

Apendix

Analysis

Graph theory research from start of project:

```
# Graph theory research
This document will look at how graph theory can be utilised with details on how
this can be implemented, making notes from sources.

# Notes
- Graph theory models paths in terms of pairs between nodes (also called
vertices) which are connected by links (also called edges) [1]
- Pathfinding is largely based on Dijkstra's algorithm, and the use of weighted
graphs [2]
- Weighting refers to the "cost" it takes to traverse a link, e.g. how long it
takes, it is a number [2,3]
- For Dijkstra's algorithm:
    "This algorithm begins with a start node and an "open set" of candidate
nodes. At each step, the node in the open set with the lowest distance from the
start is examined. The node is marked "closed", and all nodes adjacent to it
are added to the open set if they have not already been examined. This process
repeats until a path to the destination has been found. Since the lowest
distance nodes are examined first, the first time the destination is found, the
path to it will be the shortest path."

# References
[1] https://en.wikipedia.org/wiki/Graph_theory

[2] https://en.wikipedia.org/wiki/Pathfinding

[3]
https://en.wikipedia.org/wiki/Glossary_of_graph_theory#Weighted_graphs_and_netw
orks
```

Planned database/object structure

```
# Database/object structure
This document will detail a potential model for my database, and the
requirements for this. This will additionally be used to update my existing
data structures.

## Requirements
- Store map data
    - Points
    - Paths
```

    - PathParts
- Store user data
    - User name
    - User email
    - Hashed & salted user password
    - Saved map data
    - Saved routes
- Store cached routes

Structure
In order to be normalised:
    *"*Every non-key attribute is a fact about the key, the whole key, and nothing but the key*"*
    - https://hrsfc.sharepoint.com/Sites/CompSci-Stu/Essentials/AQA%20Computer%20Science%20Unit%202.pdf

Map data
Point
PointID
    string
lat
    decimal
lon
    decimal
metadata
    object
    Serving as a collection of optional metadata such as name and description.
options
    object
    Contains options for drawing to the screen, and potentially other things.

Path
PathID
    string
BeginningPathPartID
    string
metadata
    object
options
    object

PathPart
PathPartID
    string
PointID
    string
NextPathPartIDs
    string[] (nullable)

```
Metadata
    object

### User data
#### User
UserID
    string
UserName
    string
UserEmail
    string
UserPassword
    string
    Salted hash

#### UserMapData
UserID
    string
MapDataObject
    string
Params
    Object
```

Version of analysis documentation during early stages of project

```
# NEA Analysis

# The problem:
User wants access to a self-hosted mapping platform for their organisation,
from which they can import real-world map data and modify this. They want to be
able to add things like notes, rough drawings, and would like the ability to
calculate paths. This should be accessible on a variety of platforms -
Including phones, tablets and desktop computers.

# Proposed solution:
A web-based mapping app that operates on a client-server model, which can
import map data in the OpenStreetMap XML format, and return this to web clients
from a custom database so that it can be displayed to the user. This can then
be used to find paths between points, and where this data can be modified, able
to add things like annotations. This will use a authentication system.

# Objectives:
See attached "Project objectives" document.

# Research:
```

- I have conducted research on the OSM XML file format and the useful information that can be extracted from this, proposing a written alogrithm for how to do this (OSM Conversion algorithm document).
- I have been researching the Bootstrap framework and implementing it so that I can generate UI without spending lots of time worrying about implementation of UI elements. For this I have been making use of the Bootstrap reference from their website.
- I have done some research on path theory, including specifically on Dijkstra's algorithm which I have included in my "Graph theory research" document. This will be used to generate paths given connecting points on a map.

 # Proposed solution detail:
- See High level project structure document and diagram for an outline of my initial plan, and how different project elements will interact.
- TODO: Create database layout.

 # Current implementation
At the moment, I have a web client serviced by a basic web server that can display data returned from the backend on a scaleable and translateable map. The server and client use shared classes in order to represent this map data. Large parts of my OSM conversion algorithm have also been implemented, reading OSM points from a ~400MB file (Cambridgeshire) and converting them into instances of my classes, which can then be shown on the client. Displaying this OSM map data is currently buggy however and has some issues, for example some roads seem to be missing and scaling is conducted about map origin making navigating the map difficult.

Early project purpose documentation

I am working to create a project that is capable of parsing, representing and displaying real-world map data, which can then be used to calculate the most efficient paths between points. This would work in conjunction with a backend API that returns map data to a client, where the backend is responsible for calculating paths and caching these so that they can be returned to clients more quickly.

I believe that a web app built using standard web technology (HTML5, CSS, JS) would be the most effective way to achieve this, as it enables use of the service using one client codebase regardless of device specifics and is well supported by mobiles in particular. This is what I have started to create as part of my mini-project, leaving behind the backend for now so I can build the data structures and build a more simple base from which the rest of the project can be effectively built on. The codebase I have created as it stands is capable of displaying map data to the user given a set of connecting points, where this map is scalable and translatable.

The backend I have built at the moment simply serves the static site that acts as the client, but is and will be built using Node.JS, a backend JavaScript runtime environment that is suited to building APIs and is well used in professional environments. It means that in addition I can reuse the same data structures for both client and server, making transfer and development between the two faster and more efficient such that I do not have to make two separate implementations of two data structures for example.

I will be using OpenStreetMap data as part of my project, which provides comprehensive, worldwide community built map-data under a freely-usable license given attribution.

OSM Conversion algorithm:

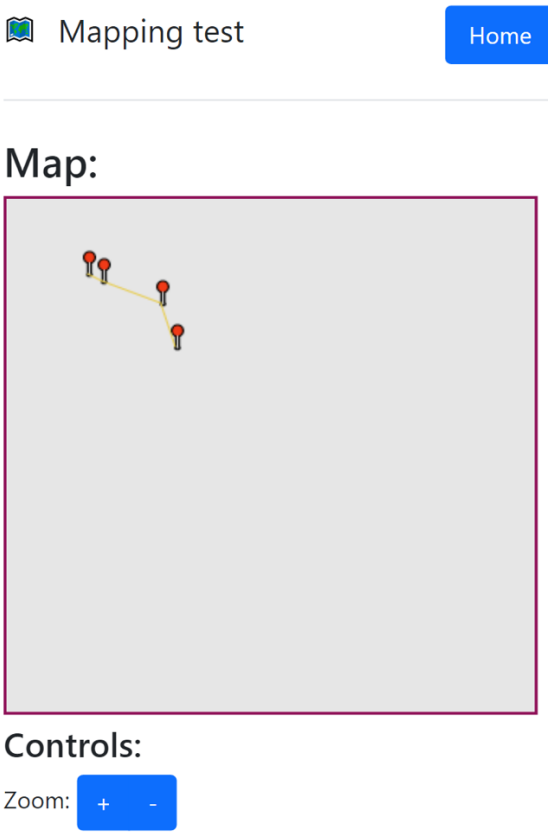
```
# OSM Conversion algorithm

## Objective
To convert OSM data into the format required by the client.

## Formats
### Custom JS format (Desired)

* JSON
* Each road is a Path object.
* Each road is given a unique path ID
* A Path is defined in terms of its initial PathPart
* Each Point of a Path is connected in a chain of PathParts
* Each PathPart contains its point and the next PathParts in the chain

### OSM (To be converted)
* XML
* Each road is defined in terms of a list of nodes
* Road metadata is given using <tag> elements in the format

    `<tag k="key" v="value"/>`
* Each road is a <way> and has <tag>s with k="highway"
* Each <way> contains component nodes expressed as <nd>s in the format:

    `<nd ref="idOfTheNode"/>`
* Each <node> has node in the format:

    `<node id="25496583" lat="51.5173639" lon="-0.140043" version="1"
changeset="203496" user="80n" uid="1238" visible="true" timestamp="2007-01-
28T11:40:26Z"></node>`

## Objectives
```

```
* Convert OSM data from XML to JSON
* Find ways with "highway" tag
* For each of these ways, find the component nodes
* For each of these nodes, convert the latitude and longitude to x,y
coordinates
* Convert each node to a MapPoint
* Create path for each way using connecting MapParts of MapPoints
```

Project development screenshots over time

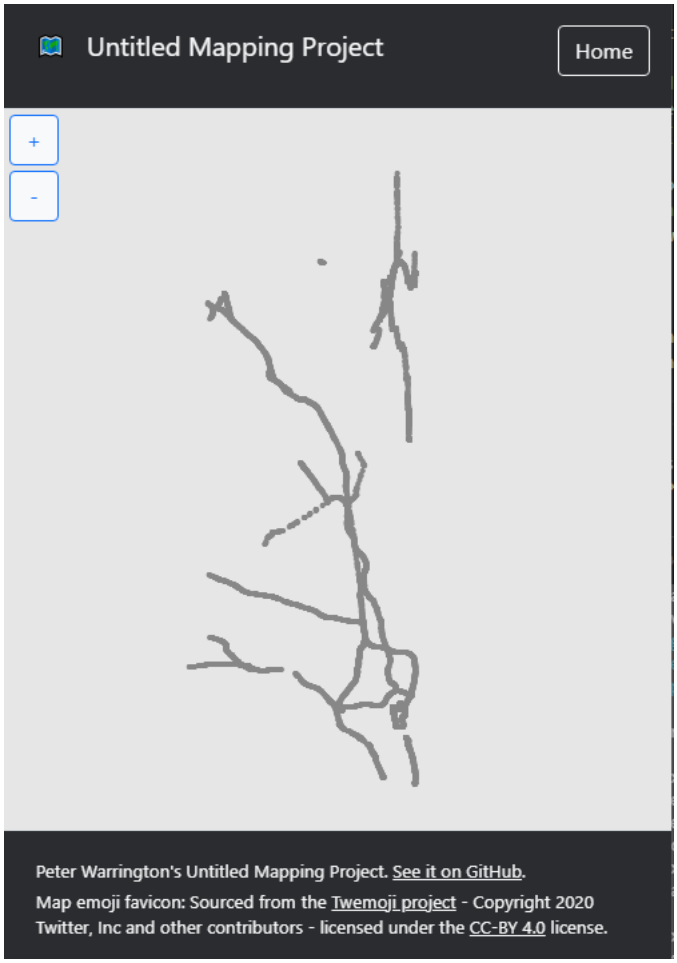
02/07/2021

Early version of prototype, simply demonstrating some points joined by a path.



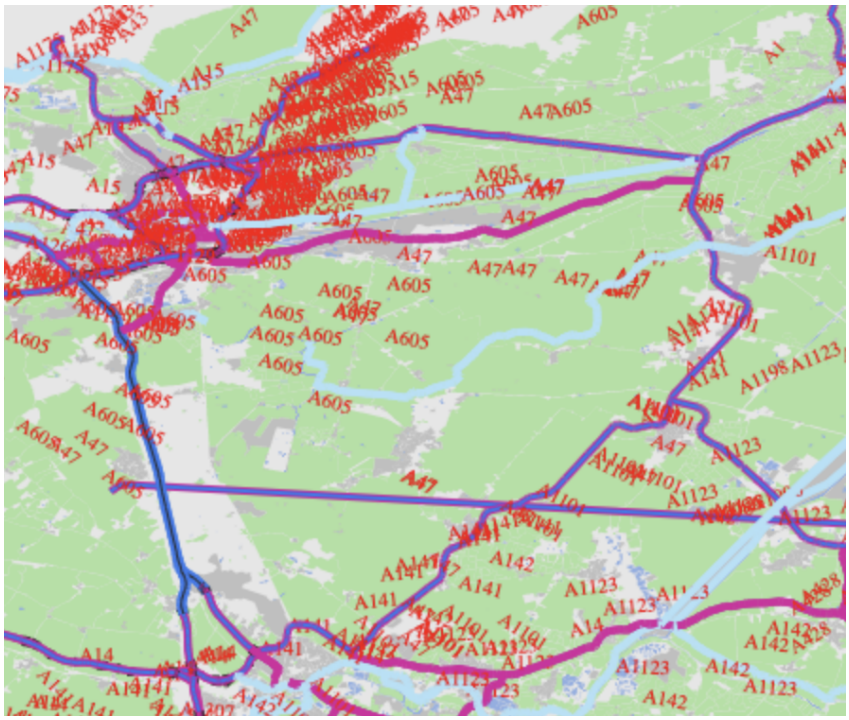
18/11/2022

Screenshot showing some of the first OSM-imported map data returned to the client.



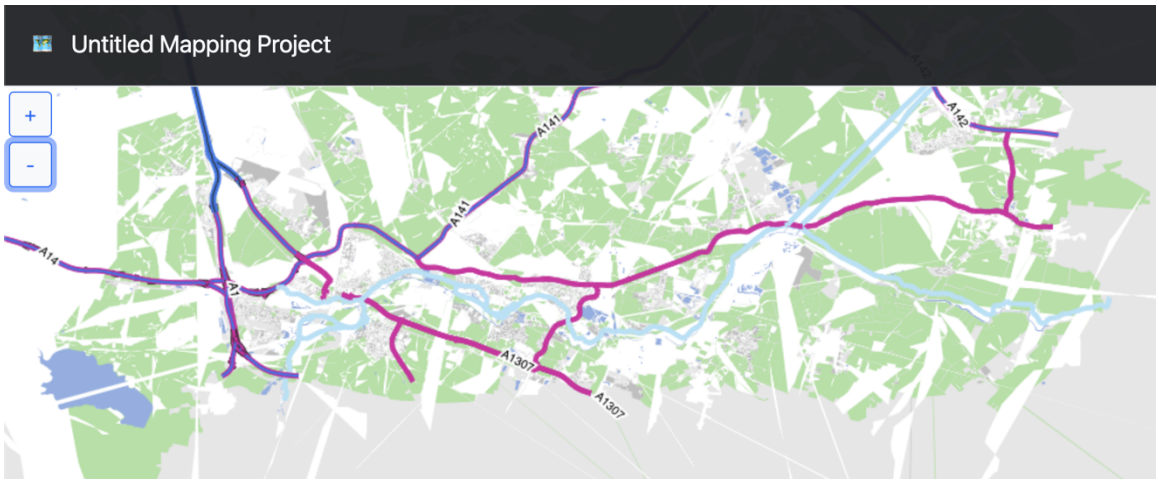
17/02/2022

Trying to implement road labels, the issue seen here was due to incorrectly assigning rotation of labels relative to origin rather than from the path.



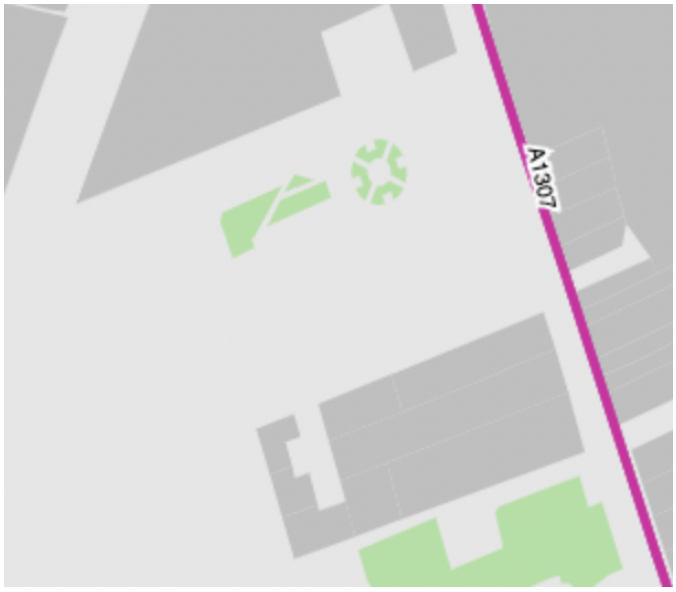
20/02/2022

During initial work to implement local caching based on map grid positions.



21/02/2022

Screenshot showing how complex areas (shapes with holes in them) were not yet displayed at this point.



23/02/2022

Screenshot showing earlier work in implementing complex areas, with simply the outer sections of these areas being implemented, with no styling yet.

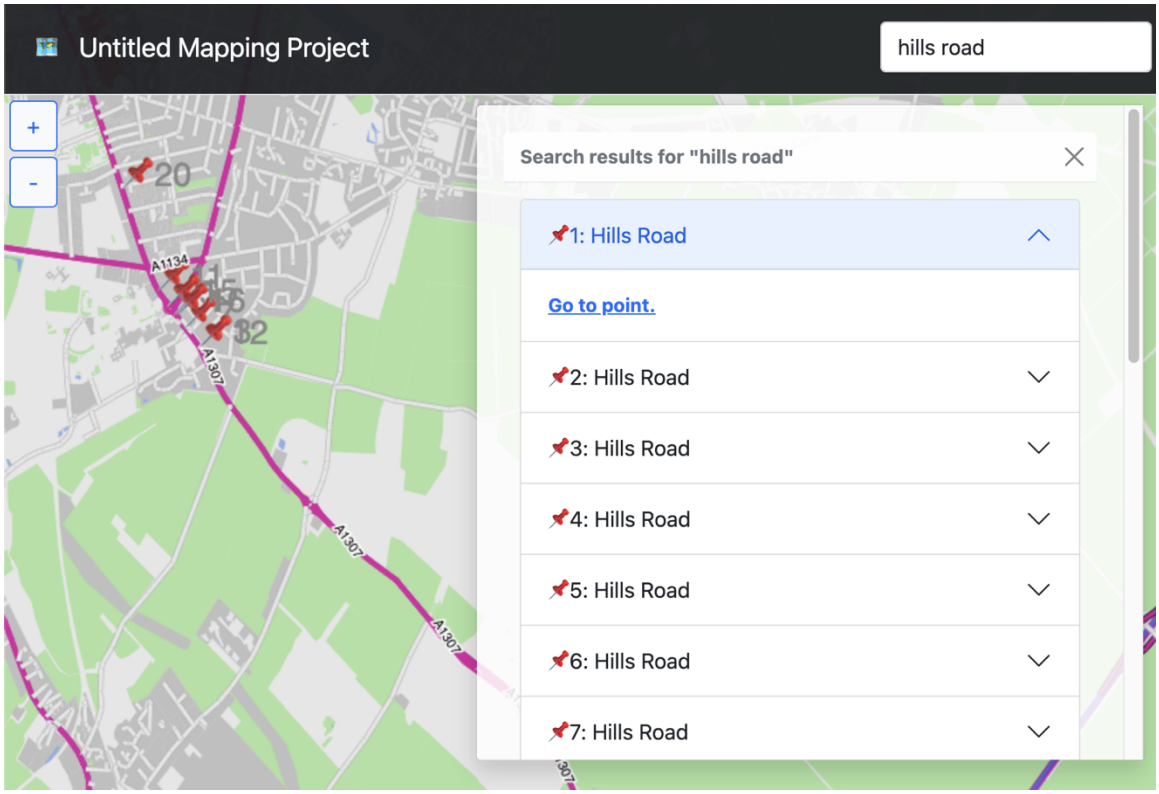
24/02/2022

Screenshot showing complex areas having been implemented

Screenshot showing more trouble with caching based on grid positions

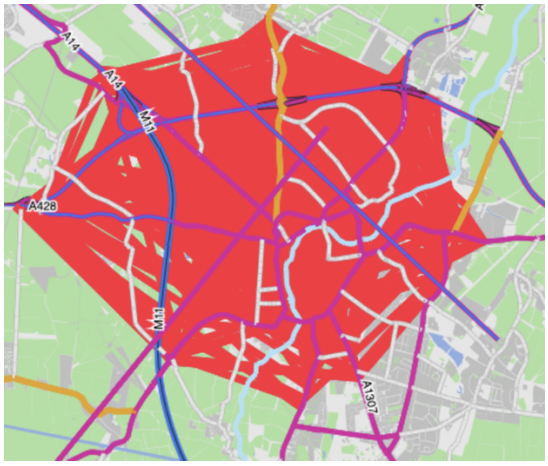
26/02/2022

Screenshot showing lack of distinguishment of search results with same name, due to the lack of location type descriptors being displayed, and the lack of a search ranking system.



27/02/2022

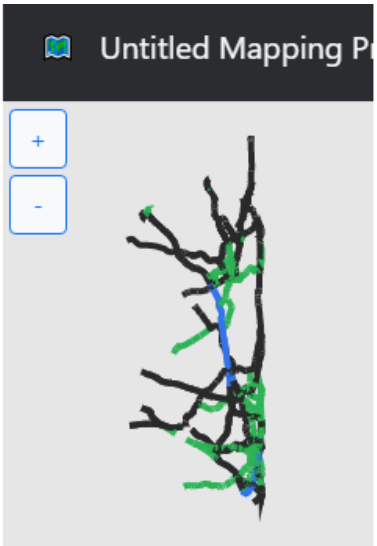
Screenshot showing early route finding code returning path of all nodes visited in determining route, rather than the route itself.



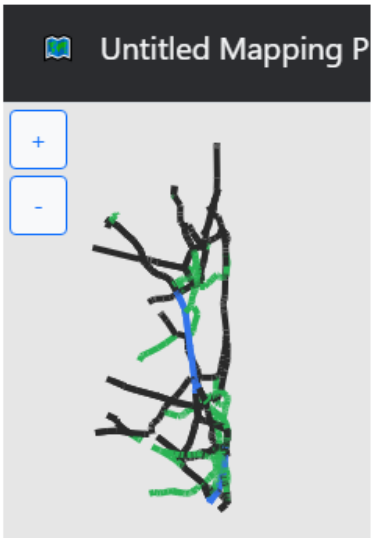
Early testing

27/12/2021

Testing was conducted to verify that there was no difference in displayed map data after a commit to only display points that were on screen.



Before commit



After commit "Only draw those points on screen"

Verifies same map is drawn to screen

Code

Final project - index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, user-scalable=no">
    <title>Untitled Mapping Project</title>
    <script src="sharedStructures.js"></script>
    <script src="index/index.js"></script>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
    <link rel="stylesheet" href="index/index.css">
```

```
</head>
<body>
    <!-- header -->
    <header class="d-flex flex-wrap justify-content-center py-3 border-bottom
px-2" id="header">
        <a href="/" class="d-flex align-items-center mb-3 mb-md-0 me-3 me-md-
auto text-light text-decoration-none">
          <span class="header-icon mx-3">🗺️</span>
          <span class="fs-5">Untitled Mapping Project</span>
        </a>

        <ul class="nav nav-pills d-flex">
          <li class="nav-item">
            <div class="d-flex">
              <input type="text" class="form-control" id="search_input"
placeholder="Search...">
            </div>
          </li>
        </ul>
      </header>

    <!-- Contains page content, mainly map and map controls-->
    <div id="mapContainer">
        <canvas id="mapCanvas" height="350px" width="100%"></canvas>
    </div>

    <!-- controls -->
    <div id="controls" class="d-grid gap-1 m-1">
      <button type="button" class="btn btn-outline-primary bg-light text-
primary" id="zoom-in">+</button>
      <button type="button" class="btn btn-outline-primary bg-light text-
primary" id="zoom-out">-</button>
      <button type="button" class="btn btn-outline-light bg-primary text-light"
id="route-ui-show-btn">🕐</button>
    </div>

    <!-- Loading indicator -->
    <div id="loading-indicator-container" style="display:none">
      <img src="/spinner.gif" id="loading-indicator">
    </div>

    <!-- Search results display -->
    <div id="results" class="container toast" role="alert" aria-
live="assertive" aria-atomic="true" data-bs-autohide="false">
      <div class="toast-header">
        <strong class="me-auto" id="results-title">Search results</strong>
        <button type="button" class="btn-close" data-bs-dismiss="toast" aria-
label="Close"></button>
```

```
      </div>
      <div class="accordion toast-body" id="results-accordion">

        </div>
      </div>
    </div>

    <!-- Route search display -->
    <div class="container toast" id="route-search-toast" role="alert" aria-
live="assertive" aria-atomic="true" data-bs-autohide="false">
      <div class="toast-header">
        <strong class="me-auto">Find a route</strong>
        <button type="button" class="btn-close" data-bs-dismiss="toast" aria-
label="Close"></button>
      </div>
      <div class="accordion toast-body" id="route-search-body">
        <input type="text" class="form-control mb-1" id="pointA_input"
placeholder="Point A">
        <input type="text" class="form-control mb-3" id="pointB_input"
placeholder="Point B">
        <button type="submit" class="btn btn-primary mb-3" id="route-submit-
btn">Find</button>
        <br/>
        <span id="route-status" class="text-secondary"></span>
      </div>
    </div>

    <footer class="text-light mt-3 border-top pt-3 px-4" id="footer">
      <p class="mb-3" style="font-size: 10pt;">
        Map emoji favicon: Sourced from the <a
href="https://twemoji.twitter.com/" class="link-light">Twemoji project</a> -
        Copyright 2020 Twitter, Inc and other contributors -
        licensed under the <a
href="https://creativecommons.org/licenses/by/4.0/" class="link-light">CC-BY
4.0</a> license.
      </p>
    </footer>



    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.
js" integrity="sha384-
ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

    <!-- Used to make search results draggable -->
```

```
    <script src="https://code.jquery.com/jquery-3.6.0.js" integrity="sha256-
H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk=" crossorigin="anonymous"></script>
    <script
      src="https://code.jquery.com/ui/1.13.1/jquery-ui.js"
      integrity="sha256-6XMVI0zB8cRzfZjqKcD01PBsAy3FlDASrlC8SxCpInY="
      crossorigin="anonymous"></script>

</body>
</html>
```

Final project - Index.css

```
.header-icon {
    font-size: 12pt;
    color: #fff;
}
#mapCanvas {
    position: absolute;
    top: 0;
    left: 0;
    z-index: -1000;
}
html, body {
    overflow-x: hidden;
    overflow-y: hidden;
    touch-action: none;
}
#header {
    background-color: #202124f1;
}
#footer {
    position: absolute;
    bottom: 0;
    width: 100%;
    background-color: #202124f1;
}
#controls {
    width: 15px;
    position: absolute;
}
#loading-indicator-container {
    position: absolute;
    padding: 5pt;
    background: #fff;
    right: 0px;
}
.toast {
```

```
    position: absolute;
    padding: 0;
    right: 5pt;
    margin: 5pt;
    width: 500px;
    max-width: 90vw;
}
#results-accordion {
    overflow-y: scroll;
    max-height: 60vh;
}
.toast-header {
    cursor: grab;
}
.toast-header button {
    cursor: auto;
}
#route-search-toast {
    width: 300px;
}
#loading-indicator {
    width: 35px;
}
```

Prototype - src/web/index/index.js:

```
class CanvasState {
    /** The {CanvasRenderingContext2D} that is used on the canvas */
    ctx
    /** The canvas element that displays the map */
    canvas
    /** Indicates whether the mouse button is down */
    canvasMouseDown = false
    /** Indicates the last recorded mouse position relative to the page in X */
    lastPageX = -1
    /** Indicates the last recorded mouse position relative to the page in Y */
    lastPageY = -1
    /** The array of {Path}s to be drawn to screen */
    paths = []
    /** A {int} multiplier to represent the zoom level */
    zoomLevel = 1
    /** {int} representing how the map has been translated in x */
    xTranslation = 0
    /** {int} representing how the map has been translated in y */
    yTranslation = 0

    constructor () {
```

```
this.canvas = document.getElementById("mapCanvas");
this.ctx = this.canvas.getContext('2d');

// Zoom functionality
document.getElementById("zoom-in").onclick = (event) => {
    this.zoom(1);
};

document.getElementById("zoom-out").onclick = (event) => {
    this.zoom(-1);
};

// Translation functionality
this.canvas.onmousedown = (event) => {
    this.lastPageX = event.pageX;
    this.lastPageY = event.pageY;
    this.canvasMouseDown = true;
    if (this.canvas.style.cursor == "grab")
        this.canvas.style.cursor = "grabbing";
}

this.canvas.onmouseup = (event) => {
    this.lastPageX = -1;
    this.lastPageY = -1;
    this.canvasMouseDown = false;
    if (this.canvas.style.cursor == "grabbing")
    this.canvas.style.cursor = "grab";
}

this.canvas.onmousemove = (event) => {
    if (this.canvasMouseDown) {
        var relativeMouseX = event.pageX - this.lastPageX;
        var relativeMouseY = event.pageY - this.lastPageY;

        this.xTranslation += relativeMouseX;
        this.yTranslation += relativeMouseY;

        this.lastPageX = event.pageX;
        this.lastPageY = event.pageY;

        this.draw();
    }
}

this.canvas.onmouseenter = (event) => {
    this.canvas.style.cursor = "grab";
}
```

```
        this.canvas.onmouseleave = (event) => {
            this.canvas.style.cursor = "default";
        }

        // Redraw on window resize to make sure canvas is right size
        window.addEventListener('resize', () => this.draw());
    }

    /**
     * Method to change the zoom level of the displayed map
     * @param {int} multiplier The multiplier to change the zoom by, normally
either -1 or 1
     */
    zoom(multiplier) {
        let zoomChange = 0.1 * multiplier;
        if (this.zoomLevel + zoomChange < 0.1)
            return;
        this.zoomLevel += zoomChange;
        this.draw();
    }

    /**
     * Calls functions to draw path to screen
     */
    draw() {
        // Update canvas width
        this.updateCanvasWidth();

        // Clear canvas
        this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height);

        // Fill background
        this.ctx.fillStyle = "#e6e6e6";
        this.ctx.fillRect(0, 0, this.canvas.width, this.canvas.height);

        this.paths.forEach(path => path.plotPoints());
        this.paths.forEach(path => path.plotLine());
    }

    /**
     * Updates the width of the canvas displayed on screen
     */
    updateCanvasWidth() {
        let mapContainerStyles =
window.getComputedStyle(document.getElementById("mapContainer"));

        // Resize to 100% (html decleration does not work)
```

```
        let paddingLeft =
parseInt(mapContainerStyles.getPropertyValue('padding-left'));
        let paddingRight =
parseInt(mapContainerStyles.getPropertyValue('padding-right'));
        let spacing = paddingLeft + paddingRight + 5;

        this.canvas.width = document.getElementById("mapContainer").clientWidth
- spacing;
    }
}

class MapPoint {
    /** Fixed x position of point in relation to others */
    x
    /** Fixed y position of point in relation to others */
    y
    /** The {canvasState} instance */
    canvasState

    /** Gets the x position relative to the canvas */
    get displayedX() {
        return (this.x * this.canvasState.zoomLevel) +
this.canvasState.xTranslation;
    }

    /** Gets the y position relative to the canvas */
    get displayedY() {
        return (this.y * this.canvasState.zoomLevel) +
this.canvasState.yTranslation;
    }

    /** Gets the x position of where the path should be drawn relative to
canvas */
    get pathPointDisplayX() {
        return this.displayedX + this.options.pathDrawPointX;
    }

    /** Gets the y position of where the path should be drawn relative to
canvas */
    get pathPointDisplayY() {
        return this.displayedY + this.options.pathDrawPointY;
    }

    /** Options for the point when drawing to screen */
    options = {
        pointDrawMethod: "text",
        pointText: "📍",
        pointFont: "sans-serif",
```

```
        pointFontWidth: 16,
        pointFillStyle: "#878787",
        pathDrawPointX: 5,
        pathDrawPointY: 1
    }

    /** {MapPoint[]} array containing the {MapPoint}s that this point connects
to. Can be undefined. */
    pointsConnectingTo
    /**
     * Creates a point that can form part of a path and be displayed on a
canvas.
     * @param {int} x Fixed x position of point in relation to others
     * @param {int} y Fixed y position of point in relation to others
     * @param {CanvasRenderingContext2D} ctx The {CanvasRenderingContext2D}
that is used on the canvas
     * @param {MapPoint} pointsConnectingTo The sequential {MapPoint} that
follows this one. Can be undefined.
     * @param {object} options Options for the point when drawing to screen
     */
    constructor (x, y, canvasState, options={}, pointsConnectingTo=undefined) {
        this.x = x;
        this.y = y;
        this.canvasState = canvasState;
        this.pointsConnectingTo = pointsConnectingTo;

        this.options = {...this.options, ...options};
    }

    /**
     * Function to draw the point to the screen
     */
    drawPoint() {
        this.canvasState.ctx.fillStyle = this.options.pointFillStyle;
        this.canvasState.ctx.font = `${this.options.pointFontWidth}px
${this.options.pointFont}`;
        this.canvasState.ctx.fillText(this.options.pointText, this.displayedX,
this.displayedY);
    }

    /** Returns wether this is the last point of a path */
    get isEndOfPath() {
        return this.pointsConnectingTo == undefined;
    }

    /**
     * Connects a point to this point
```

```
     * @param {Point} pointConnectingTo The point that you want to connect this
point to
     * @returns The point that you are connecting to this point
     */
    connectingTo(pointConnectingTo) {
        if (this.pointsConnectingTo == undefined)
            this.pointsConnectingTo = [];
        this.pointsConnectingTo.push(pointConnectingTo);
        return pointConnectingTo;
    }
}

class Path {
    /** The {string} used to identify the path */
    pathId
    /** The {MapPoint} object that starts the path */
    startingPoint
    /** The {canvasState} instance */
    canvasState
    /** Options for the path when drawing to screen */
    options = {
        pathFillStyle: "#e8cc4a",
        pathLineWidth: 4,
        recalculatePathFlag: false
    }

    /** Returns the line width to be displayed on the canvas */
    get lineWidth() {
        return this.options.pathLineWidth;
    }

    /**
     * Creates a path using a starting points
     * @param {MapPoint} startingPoint The {MapPoint} object that starts the
path
     * @param {CanvasRenderingContext2D} ctx The {CanvasRenderingContext2D}
that is used on the canvas
     * @param {object} options Options for the path when drawing to screen
     */
    constructor (startingPoint, canvasState, pathId, options={}) {
        this.startingPoint = startingPoint;
        this.canvasState = canvasState;
        this.options = {...this.options, ...options};
        this.pathId = pathId;
    }

    /**
```

```
     * Converts a tree of connecting points to a array of all points (for
drawing individual points unconnectedly)
     * @param {MapPoint} startingPoints The {MapPoint[]} object containing a
starting point on a path
     * @param {MapPoint[]} pathArray A sequential array of {MapPoint}s
     * @returns {MapPoint[]}
     */
    static getAllPointsOnPath(startingPoints, pathArray=[]) {
        startingPoints.forEach(startingPoint => {
            pathArray.push(startingPoint);
            if (startingPoint.pointsConnectingTo != undefined)
                Path.getAllPointsOnPath(startingPoint.pointsConnectingTo,
pathArray);
        });
        return pathArray;
    }

    /**
     * Converts a sequential array of points to a tree of connecting points
     * @param {MapPoint[]} pathArray A sequential array of {MapPoint}s
     * @returns {MapPoint} The {MapPoint} object that starts the path
     */
    static connectSequentialPoints(pathArray) {
        var pathArrayCopy = Object.values(Object.assign({},
pathArray)).reverse();
        for (let i = 1; i < pathArrayCopy.length; i++) {
            pathArrayCopy[i].pointsConnectingTo = [pathArrayCopy[i-1]];
        }
        return pathArrayCopy[pathArrayCopy.length - 1];
    }

    /**
     * Plots line connecting points in this.startingPoint to screen.
     */
    plotLine() {
        // Set properties
        this.canvasState.ctx.lineWidth = this.lineWidth;
        this.canvasState.ctx.strokeStyle = this.options.pathFillStyle;

        this.canvasState.ctx.beginPath();

        // Initialize array containing the  initial points to draw
        let startingPointsToDraw = [...this.startingPoint.pointsConnectingTo];

        // Iterate through startingPointsToDraw
        for (let i = 0; i < startingPointsToDraw.length; i++) {
            const startingPoint = startingPointsToDraw[i];
```

```
            // Move to the starting point
            this.canvasState.ctx.moveTo(this.startingPoint.pathPointDisplayX,
this.startingPoint.pathPointDisplayY);

            // If we get to a branch, push the other branches to
startingPointsToDraw to iterate through later
            let currentPoint = startingPoint;
            while (!currentPoint.isEndOfPath) {
                // Plot a line from the last plotted point to the point at
currentPoint
                this.canvasState.ctx.lineTo(currentPoint.pathPointDisplayX,
currentPoint.pathPointDisplayY);

                if (currentPoint.pointsConnectingTo.length > 1) {
                    for (let j = 1; j < currentPoint.pointsConnectingTo.length;
j++) {

startingPointsToDraw.push(currentPoint.pointsConnectingTo[j]);
                    }
                }

                // Advance pointer to next connecting point in the closest
branch
                currentPoint = currentPoint.pointsConnectingTo[0];
            }
            this.canvasState.ctx.lineTo(currentPoint.pathPointDisplayX,
currentPoint.pathPointDisplayY);

            // Draw line to canvas
            this.canvasState.ctx.stroke();
        }
    }

    /**
     * Plots markers for points in this.startingPoint
     */
    plotPoints() {
        Path.getAllPointsOnPath([this.startingPoint]).forEach(point =>
point.drawPoint());
    }
}

/**
 * Function called at page load to test some points on canvas
 */
function MapTest() {
    // Create canvas state
    canvasState = new CanvasState();
```

```
    // Create path 1
    var startingPointPath1 = new MapPoint(50, 50, canvasState, {pointText:
"📍\t\tPath 1"});
    startingPointPath1.connectingTo(new MapPoint(60, 55, canvasState))
    .connectingTo(new MapPoint(100, 70, canvasState))
    .connectingTo(new MapPoint(110, 100, canvasState));

    var path1 = new Path(startingPointPath1, canvasState, "Path1");

    // Create path 2
    var startingPointPath2 = new MapPoint(30, 200, canvasState, {pointText:
"📍\t\tPath 2"});
    startingPointPath2.connectingTo(new MapPoint(400, 170, canvasState))
    .connectingTo(new MapPoint(300, 250, canvasState))
    .connectingTo(new MapPoint(270, 20, canvasState))
    .connectingTo(new MapPoint(120, 100, canvasState))
    .connectingTo(new MapPoint(160, 160, canvasState));

    var path2 = new Path(startingPointPath2, canvasState, "Path2");

    canvasState.paths = [path1, path2];

    // Draw points
    canvasState.draw();
}

// Once the page has fully loaded, call MapTest
document.addEventListener('DOMContentLoaded', MapTest, false);
```

Git commit log

2021-06-25:Peter Warrington::Initial plan

2021-06-25:Peter Warrington::Fix plan checkboxes

2021-06-25:Peter Warrington::Fix plan checkboxes again

2021-07-02:Peter Warrington::Create static map testing site, able to display array of points

2021-07-09:Peter Warrington::Add zoom foundations

2021-07-09:Peter Warrington::Add zoom functionality

2021-07-12:Peter Warrington::Add vs code launch.json

2021-07-12:Peter Warrington::Add consts

2021-07-12:Peter Warrington::Add comments to web/index.js

2021-07-12:Peter Warrington::Add logging functionality

2021-07-12:Peter Warrington::Amend gitignore

2021-07-12:Peter Warrington::Fix request logging

2021-07-12:Peter Warrington::Set vscode current dir in launch.json

2021-07-12:Peter Warrington::Fix width of map canvas

2021-07-12:Peter Warrington::Add map translation functionality

2021-07-12:Peter Warrington::Prevent negative scaling

2021-07-12:Peter Warrington::Dynamic resizing of map canvas

2021-07-12:Peter Warrington::Move zoom into seperate function

2021-07-16:Peter Warrington::Create project purpose document

2021-07-16:Peter Warrington::Mutli-path support, CanvasState to store globals

2021-09-16:Peter Warrington::Create initial project objectives

2021-09-16:Peter Warrington::Delete README.md

2021-09-16:Peter Warrington::Change objectives to check boxes

2021-09-16:Peter Warrington::Objectives update

2021-09-16:Peter Warrington::Objectives update

2021-09-16:Peter Warrington::Objectives update

2021-09-23:Peter Warrington::Seperate points and paths

2021-09-23:Peter Warrington::Add favicon and footer

2021-09-23:Peter Warrington::Add basic touch support

2021-09-23:Peter Warrington::Full-screen map redesign

2021-09-23:Peter Warrington::Add specific map translation function

2021-09-23:Peter Warrington::disable phone zoom

2021-09-23:Peter Warrington::Adjust text sizes in line with static demo

2021-09-27:Peter Warrington::Move map structures to back/frontend shared file

2021-09-27:Peter Warrington::Rename page title from "Map test page"

2021-09-30:Peter Warrington::High level project structure

2021-09-30:Peter Warrington::move shared structures into single shared object

2021-09-30:Peter Warrington::Client access paths from backend

2021-09-30:Peter Warrington::Add route finding to high level project diagram

2021-09-30:Peter Warrington::Fix typo

2021-10-15:Peter Warrington::Seperate drawing methods/properties from shared

2021-10-15:Peter Warrington::Merge branch 'main' of https://github.com/PeterWarrington/NEA-mapping into main

2021-10-18:Peter Warrington::Add commentst to index.js

2021-10-18:Peter Warrington::Fix point options not being passed to client

2021-11-13:Peter Warrington::Detail OSM conversion algorithm plan

2021-11-13:Peter Warrington::Script to create project format map data from OSM

2021-11-13:Peter Warrington::Add libraries neccessary for conversion from osm

2021-11-13:Peter Warrington::Config VS code to be able to launch OSM conversion

2021-11-13:Peter Warrington::Add ability to display OSM test points in browser

2021-11-13:Peter Warrington::Gitignore output files

2021-11-15:Peter Warrington::Create initial analysis report, containing TODOs

2021-11-15:Peter Warrington::Begin graph theory research notes

2021-11-18:Peter Warrington::Move analysis, implementation details, export pdf

2021-11-18:Peter Warrington::Switch from 📍 point marker to ●

2021-11-18:Peter Warrington::Switch from 📍 point marker to ●

2021-11-18:Peter Warrington::Merge branch 'main' of https://github.com/PeterWarrington/NEA-mapping into main

2021-11-18:Peter Warrington::Plot points over lines so visible

2021-11-18:Peter Warrington::Update screenshot

2021-11-18:Peter Warrington::Add pandoc generation example

2021-11-18:Peter Warrington::Add newline to saved log messages

2021-11-18:Peter Warrington::Add newline to saved log messages

2021-11-18:Peter Warrington::Merge branch 'main' of https://github.com/PeterWarrington/NEA-mapping into main

2021-11-19:Peter Warrington::Merge branch 'main' of https://github.com/PeterWarrington/NEA-mapping into main

2021-11-19:Peter Warrington::Merge branch 'main' of https://github.com/PeterWarrington/NEA-mapping into main

2021-11-19:Peter Warrington::Remove extra new lines from terminal logs

2021-11-19:Peter Warrington::Create initial database strucutre file, map data

2021-11-29:Peter Warrington::Write up initial database and object structure

2021-12-02:Peter Warrington::Update database structure

2021-12-02:Peter Warrington::Convert data structures to a normalised db format

2021-12-06:Peter Warrington::Improve zoom control relative to translation

2021-12-06:Peter Warrington::fix symbolic link

2021-12-06:Peter Warrington::Revert "fix symbolic link"

This reverts commit 14ee950e5f24c9bb6dcb67d40fb4aa331cfc56e8.

2021-12-06:Peter Warrington::fix symbolic link

2021-12-06:Peter Warrington::Number objectives

2021-12-13:Peter Warrington::Amend OSM conversion algorithm detail

2021-12-13:Peter Warrington::WIP OSM conversion changes undergoing testing

2021-12-16:Peter Warrington::Improve convertFromOSM iteration

2021-12-16:Peter Warrington::unsuccesful work on threading convertFromOSM

2021-12-16:Peter Warrington::Make threaded convert from osm run for testing

2021-12-16:Peter Warrington::Fix import

2021-12-21:Peter Warrington::gitignore cpuprofile files

2021-12-21:Peter Warrington::Add '--cacheRead' to convertOSM launch.json config

2021-12-21:Peter Warrington::gitignore src/db.json

2021-12-21:Peter Warrington::Decrease convertOSM time from 5 hours to 2 mins

... and add metadata for convertOSM map objects

2021-12-25:Peter Warrington::allocate more ram to server for db cacheing

2021-12-25:Peter Warrington::colour code roads

2021-12-25:Peter Warrington::GetDBfromQuery endpoint add

2021-12-27:Peter Warrington::Only draw those points on screen

2021-12-27:Peter Warrington::Test 'Only draw those points on screen'

2021-12-27:Peter Warrington::Fix logging not being instantaneous

2021-12-27:Peter Warrington::Log that server has started

2022-01-01:Peter Warrington::Add getMapObjectIDsOfType to db implementation

2022-01-01:Peter Warrington::Request specific map area from server

2022-01-01:Peter Warrington::Fix dbFromQuery map area

2022-01-01:Peter Warrington::Update web DB query

2022-01-02:Peter Warrington::Add express dependency

2022-01-02:Peter Warrington::Add .DS_store to gitignore

2022-01-02:Peter Warrington::Change backslash to forwardslash in launch.json

2022-01-06:Peter Warrington::Only request points that are not already on screen

2022-01-06:Peter Warrington::Remove debug console.log

2022-01-22:Peter Warrington::Revert timered map load, add debug_viewWholeMap

2022-01-22:Peter Warrington::Add debug mode, with some debug logging

In order to try and fix map area filtering

2022-01-22:Peter Warrington::Fix map data dispaly issues

Issue arrose from plotLine incorrectly reading from a global database

object defined in MapTest that was only for intermediary conversion

purposes.

Made sure it read from canvasState.database instead to fix this.

Also added database merging.

2022-01-22:Peter Warrington::Fix reference to database not canvasState.database

2022-01-22:Peter Warrington::Add map data update on map translation

2022-01-22:Peter Warrington::Move http req receive data func into canvasState

2022-01-22:Peter Warrington::Fix excludeAreas not working

2022-01-24:Peter Warrington::Fix db returned when no highway or search filter

2022-01-24:Peter Warrington::Add more roads colours and catch cache error

2022-01-24:Peter Warrington::.gitignore changes

2022-01-24:Peter Warrington::Apply borders to roads

2022-01-28:Peter Warrington::Add Area map object definition

2022-01-28:Peter Warrington::osmConvert: Convert water bodies

2022-01-28:Peter Warrington::change requested highways

2022-02-03:Peter Warrington::add metadata to areas

2022-02-03:Peter Warrington::add basic code to return map areas on query

2022-02-03:Peter Warrington::basic code to draw areas to screen

2022-02-03:Peter Warrington::Filter areas by map area

2022-02-04:Peter Warrington::only draw areas on screen to improve perfromance

2022-02-04:Peter Warrington::add search button (no functionality)

2022-02-11:Peter Warrington::Basic search API improve

Should have more data on raw map before start searching

2022-02-11:Peter Warrington::Seperate OSM and non-OSM metadata

My philosophy is that metadata should be queried from the custom

defined metadata keys, which sources from OSM metadata,

but can still fall back on raw OSM metadata

2022-02-11:Peter Warrington::Add river to accepted "highways"

2022-02-11:Peter Warrington::"highway" -> "pathType"

2022-02-12:Peter Warrington::convertFromOSM: convert land

2022-02-12:Peter Warrington::allocate server more memory

2022-02-12:Peter Warrington::custom JSON stringify to workaround data too large

2022-02-12:Peter Warrington::dbFromQuery: Rtn whole Area not just Area onscreen

2022-02-12:Peter Warrington::add ID and metadata in areaFromObject

2022-02-12:Peter Warrington::draw areas on different layers, custom area style

2022-02-12:Peter Warrington::fix colours for areas

2022-02-12:Peter Warrington::add more cache error handling

2022-02-15:Peter Warrington::Fix lat-lng projection to x-y by using mercator

2022-02-15:Peter Warrington::specify URLs to request map data as vars

2022-02-15:Peter Warrington::Only request map data when not already downloaded

2022-02-15:Peter Warrington::Add load spinner, use multiplier based zoom level

2022-02-15:Peter Warrington::Load whole paths at a time

2022-02-17:Peter Warrington::optimise getMapObjectIDsOfType

decreasing query time substantially (measured from ~5.5s -> ~3s)

2022-02-17:Peter Warrington::optimise query by doing custom json conversion

for some reason using a return string takes less time than using

res.write(), not quite sure why

2022-02-17:Peter Warrington::Test draw highway labels to screen

2022-02-17:Peter Warrington::part: add getPoint(),getNextPart(),getNextPoint()

2022-02-17:Peter Warrington::Add better label drawing (angles, spaced out)

2022-02-17:Peter Warrington::sharedStructures: fix error during getNextPoint

2022-02-17:Peter Warrington::Translate highway labels slightly

2022-02-17:Peter Warrington::Actually examine width during query

2022-02-17:Peter Warrington::Add another test url

2022-02-18:Peter Warrington::server-side db: remove point display options

Decreases db file size to ~54% original size

decreases query load (~2.5s -> ~2.25s)

2022-02-19:Peter Warrington::labels: Calculate angle using distance, add border

Calculating angle using distance rather than points away, as points

away is not a reliable indicator of distance

2022-02-19:Peter Warrington::Define more area colours

2022-02-21:Peter Warrington::Add complex area definition (revert)

2022-02-21:Peter Warrington::Change paths displayed based on zoom level

2022-02-21:Peter Warrington::Shared structs to one file, not use symbolic links

2022-02-21:Peter Warrington::Load new roads despite exclude area

2022-02-21:Peter Warrington::Fix water way colour

2022-02-21:Peter Warrington::Fix road styles being overriden wrong

2022-02-23:Peter Warrington::ComplexAreas made of ComplexAreaParts, add drawing

2022-02-23:Peter Warrington::osmConvert: Extract complex areas

2022-02-24:Peter Warrington::Proper complex area drawing

2022-02-24:Peter Warrington::Improved performance and complex area drawing

2022-02-24:Peter Warrington::Merge branch 'main' of https://github.com/PeterWarrington/NEA-mapping

2022-02-25:Peter Warrington::Use map instead of object for grid

2022-02-25:Peter Warrington::Use map for db

2022-02-25:Peter Warrington::Code to write map areas images to file for clients

2022-02-26:Peter Warrington::Draw full map (not labels) to map images

2022-02-26:Peter Warrington: (origin/tile-code-broken, tile-code-broken):Not working map area tile code

2022-02-26:Peter Warrington::git ignore left over map area images

2022-02-26:Peter Warrington::Add search functionality

2022-02-26:Peter Warrington::Also enable to search for paths

2022-02-27:Peter Warrington::Improve search

2022-02-27:Peter Warrington::Don't expand search results by default

2022-02-27:Peter Warrington::Add pinch zoom mobile

2022-02-27:Peter Warrington::Add all nodes to db.json for searching

2022-02-27:Peter Warrington::Search improvements

2022-02-27:Peter Warrington::Move grid square cache into shared structures

This is to make path finding easier later, and just helps with

organisation

2022-02-28:Peter Warrington::Add code to find route via dijkstra

2022-02-28:Peter Warrington::Better error handling

2022-02-28:Peter Warrington::Improve search, route experience: status print etc

2022-02-28:Peter Warrington::Increase bounds route finder finds nearby highway

2022-02-28:Peter Warrington::Toggle to show search result metadata

2022-02-28:Peter Warrington::Can use search terms to find route

2022-02-28:Peter Warrington::Send route request on enter key

2022-02-28:Peter Warrington::Rank places first in search results

Helps with routing

2022-02-28:Peter Warrington::Fix search scoring

2022-02-28:Peter Warrington::Road names as route source/destination support

2022-02-28:Peter Warrington::Improve db from query time, and queries made

2022-02-28:Peter Warrington::Optimise route finding

2022-02-28:Peter Warrington::Optimise search and therefore routing using index

2022-03-01:Peter Warrington::Add title page for full doc

2022-03-02:Peter Warrington: (HEAD -> main, origin/main, origin/HEAD):Work on documentation