Final Year Project Report

Full Unit - Final Report

OFFLINE HTML5 MAPS APPLICATION

Demo of Final Application

Michael Panlilio

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: DONGGYUN HAN

Department of Computer Science
Royal Holloway, University of London

April 12, 2024 *(Extension Applied)*

Table of Contents

Abstract

The number of internet applications and users alongside those applications is rapidly growing **[1]** leading to the wider improvement of web technology and standards, for instance, HTML5. This Hypertext Markup language has become very widespread across the modern web as well as the diverse set of recent technologies HTML5 provides. With the use of tools such as Service Workers, Cache, Web Storage and IndexedDB, a mapping application can be created with the capability of working offline as assets can be cached, and data can be stored locally. Many mapping applications are developed offering all kinds of functionality and interfaces, but for it to have offline functionality can benefit a lot of users that may need access to geographical data whilst in an area with very limited or no internet access at all. This project also focuses on creating a user-friendly application that provides various functionalities ranging from allowing a user to zoom in and out of the map to allowing a user to request for dynamic display of data. Therefore the use of HTML5 Canvas and jQuery allows implementing features to enhance user interaction and customisation. The development of this application will also have the motive to import large data sets from OpenStreetMap (OSM). OSM's core product includes a spatial database, which contains free geographical data and information from all over the world. **[2]** Therefore, with the support of HTML5 technologies, we can manage to retrieve the data from OSM and render it in an interactive map. This Offline HTML5 Application is also motivated by the way maps will be rendered. As the visualisation of vector data can be achieved using methods such as Canvas API **[3]**, we can manipulate this data to offer more functionality for the user in real-time. However, image tile maps offer an alternative with pre-rendered images indicating a quicker load time.

1 Introduction

Web development will keep on expanding, and the demand for all types of applications that bridge the gap between online and offline experiences has grown substantially. Utilising the power of HTML5 technologies, this project is outlined to develop an innovative offline mapping application with beneficial user functionality which will serve the purpose of being accessible in areas with limited or no internet connectivity. With the use of recent HTML5 advancements, the creation of a vector map application can be widely explored to render geographical data in real-time whilst offering dynamic control for users, especially in offline environments.

This final report will showcase the importance of web applications and the use of HTML5 components to create an optimal mapping application. Not only that, but it will also provide the significance on why mapping applications are essential tools in today's interconnected world. By utilising HTML5 technologies, this project aims to push boundaries of traditional mapping applications by enabling seamless offline access to crucial geographical data.

1.1 Aims and Goals of the Project

The Aims are as follows:

Develop an Offline Mapping Application
A mapping application that can still function without an internet connection can be achieved by utilising HTML5 technologies such as *Cache*, *Service Workers, Indexed DB, and Web Storage.*

Rendering Maps in Real-Time
Focusing on the implementation of Vector Maps, this provides a real-time rendering of location data and can result in enhanced user interaction and exploration.

The goals are as follows:

Concept Programs
Offline Application focusing on Service Workers and Cache.
Todo List Application focusing on IndexedDB or Web Storage.
Drawing Shape Application focusing on HTML5 Canvas.
Application to list OSM data in raw form.

OpenStreetMap Integration
Offline Mapping Application that renders OSM data and allows users to zoom and pan.

Performance Optimisation
Enhance rendering performance by focusing on IndexedDB and image caching.

Extended Features
Enhance the user experience by implementing search features and displaying dynamic data.
Support different OSM data formats.
Download map data dynamically when a connection is present.

1.2 Rationale

As mapping becomes more integral into our daily lives, the increasing reliance on mapping applications necessitates the development of an offline mapping application. Especially for the purposes such as navigation and seeking information about a place whether it's local or global, geographical data like this must have the capability to render in real-time whilst also in offline status.

These large data sets can be widely manipulated for users to have a better understanding of them. This also indicates the larger functionality that can also be implemented. Not just for a HTML5 application, but data like this can be used for further concepts such as virtual reality. Furthermore, it could even be used towards developing an offline mobile application with wider accessibility and limited configuration needs. **[4]**

2 Literature Review

2.1 Introduction

Enabling the innovation of enhanced offline applications that were once reliant on constant internet connectivity, the evolution of HTML5 has revolutionised the scope of web development. Amongst these advancements, the utilisation of HTML5 technologies such as Web Storage, Service Workers, Cache API and IndexedDB has given developers the opportunity to create web-based applications that have the seamless function to work in offline environments. Due to this, many possibilities for various fields have opened up including the realm of mapping applications.

Mapping applications have managed to become an integral part of every day navigation, providing users with many features such as real-time location information, route planning, and other exploration functionalities. In addition, although traditional mapping applications rely on image tile-based representations that require an internet connection for data retrieval and rendering, an alternative approach exists known as vector mapping.

While the use of vector tiles is becoming more widespread, **[5]** vector map applications are the effective utilisation of HTML5 offline technologies, it also enables dynamic data representation and customisation, providing users with a more interactive and versatile mapping experience.

Research Question:

How can recent advancements in HTML5 offline technologies be effectively utilised in the development of vector map applications, particularly focusing on the representation and manipulation of Open Street Map (OSM) data?

2.2 Background

Usually mapping applications have relied on tile-based representations, where map data becomes pre-rendered into a series of image tiles that are then fetched and displayed as needed. While effective, this approach has limitations, in significance when offline access of dynamic data manipulation is required. As mentioned, vector map applications offer a much better alternative as it stores location data in vector formats which can be therefore used for real-time rendering and dynamic interaction with map elements. This therefore enhances user experience and allows them to manipulate data in the way they want to.

As an open source database, Open Street Map (OSM) provides a collaborative platform which offers a rich dataset that includes a wide variety of detailed information ranging from roads, to buildings, to points of interests and much more. Utilisting OSM data in vector map applications creates the opportunity to create a versatile mapping solution that can be tailored to specific user needs and also be used in an environment with no internet connection.

Using both HTML5 technologies and vector map representations gives the opportunity of creating a robust mapping application that offers enhanced flexibility, performance, and usability. Ultimately, by harnessing the capabilities of Web Storage, Service Workers, Cache API and IndexedDB, mapping applications can overcome the restrictions of tile-based mapping solutions. Although, several challenges may arise when trying to achieve this. This includes efficient data storage and retrieval, rendering optimisation as well as data manipulation all within an offline environment.

2.3 Problem Areas

Developing a mapping application that uses vector tiles over the traditional image tiles whilst also implementing HTML5 technologies as well presents various significant challenges that need to be addressed in order to ensure that a successful implementation is carried out.

Handling Large Datasets

As vector map applications usually deal with large datasets, especially with detailed maps like OSM provides, storing and retrieving these large amounts of geographical data in an offline state poses challenges. For example, storing these large datasets can lead to a high memory consumption, which may potentially exceed browser memory limits which can therefore lead the mapping application to perform at a minimal state. However there are a few techniques to overcome this. For instance, by dividing map data into manageable chunks can facilitate faster data retrieval and rendering. In addition, other strategies that load map data incrementally based on the user's interaction can optimise memory usage and therefore minimise initial loading times.

Rendering Performance

As rendering vector maps in real-time requires efficient processing of spatial data in order to dynamically display map elements, this can significantly strain device resources resulting in both slow performance and map interaction. Therefore, it is crucial that smooth and responsive map rendering is present, especially when a user desires to zoom and pan across the map highlighting user

experience. A way to strategise this problem area is to pre-render and cache accessed map tiles in a frequent manner in order to minimise the rendering overhead and improve map responsiveness. Furthermore, we can prioritise rendering essential map elements first, in which data will be gradually added as the user interacts with the map.

Offline Data Synchronisation

One of the biggest challenges, especially when trying to achieve the offline functionality, is keeping locally stored map data synchronised with the latest updates from the OSM dataset. If not kept synchronised, then inconsistencies will arise between the local dataset and the original OSM data resulting in inaccuracies and outdated information in the mapping application. However, this can be tackled by queuing OSM update requests whilst offline and then synchronising them with the server when a network connection does become available.

Dynamic Feature Support

Multiple dynamic features like search functionality, data overlays, and interactive elements present challenges in data accessibility and real-time interaction when no internet connection is present. This is significant as offline mapping applications need to maintain the responsiveness and interactivity expected from online counterparts. Therefore, we can utilise HTML5 technologies such as IndexedDB or Web Storage to enable offline access and interaction. Not only that, but strategies like client-side processing can be utilised in which the application can offload processing tasks such as search queries to the client-side in order to minimise server dependence and enhance responsiveness allowing the user to still have a good experience.

Compatibility and Performance Variability

In conclusion, the integration of HTML5 technologies demands mapping applications to maintain uniform performance and functionality across diverse web browsers and network conditions. The challenge lies in the inconsistencies in browser capabilities, potentially resulting in compatibility glitches and performance disparities, thus impacting user usability and experience. Nonetheless, thorough testing across various web browsers can mitigate these concerns by detecting and resolving compatibility issues at the onset of the development phase.

2.4 Discussion Analysis

With one of the primary challenges being the efficient handling of large datasets, especially OSM data, researchers have explored various techniques to address this challenge. One method for handling large datasets is to compress the original data, so that the compressed form fits into the memory. **[6]** In addition, advancements in IndexedDB and Web Storage provide opportunities for efficient storage and retrieval of map data chunks, which enables a smoother user experience, even when facing limited connectivity. It may be more practical to use an IndexedDB approach as Web Storage in browsers is limited.

Thousands of applications exist that use the data of OSM, for example Osmand. **[7]** Demonstrating the implementation of HTML5 offline technologies in mapping applications, Osmand is an open-source offline map and navigation mobile application. As it utilises OSM data, it ensures that users have access to accurate and up-to-date map information including a large set of geographic

features. In addition, one of its biggest features is its extensive offline map coverage, where users can download maps for specific regions directly to their device and still have the capability to navigate and explore without the need for an active internet connection.

Lastly, OSM data can be obtained in various formats, including OSM-XML **[8]**, which provides a comprehensive representation of map features and attributes. This raw XML data can be stored locally and parsed dynamically to extract relevant information for rendering maps offline. However, due to the potentially large size of OSM datasets, preprocessing and optimization techniques are often employed to enhance storage efficiency and retrieval speed.

2.5 Conclusion

To conclude, integrating HTML5 technologies including offline functionality with vector map representations highlights a significant advancement in the development of mapping applications. From handling large datasets to optimise rendering performance as well as support dynamic features, developing a mapping application faces multiple obstacles in achieving a smooth and responsive offline mapping experience. However, this can be tackled by using innovative strategies such as data compression, incremental loading, and client-side processing. By overcoming these challenges, mapping applications can be developed providing an efficient and user-friendly experience as well as transcending the limitations of traditional tile-based mapping systems whilst offering an offline experience. Looking and planning ahead, leveraging the power of HTML5 technologies and OSM data, mapping applications have high capabilities to provide accurate and up-to-date information in real-time whilst also offering extensive offline coverage and dynamic interaction functionality.

3 Software Engineering

To meet the needs for user satisfaction, software engineering is significant as it will outline how this project will be structured and documented whilst dealing with the complexities that will come in.

3.1 Methodology

Treating this project like a puzzle, each concept program is a jigsaw piece where each deliverable has functionality that adds value towards the final application. In significance, we can use a more agile style approach and treat each concept program as an increment and deliver a functional subset of features in each iteration:

Increment 1: "Hello World" Offline HTML5 Application

Develop a simple offline app focusing on the use of Service Workers and Cache.
The technologies together serve as a foundation for offline functionality.

Increment 2: Todo List Application

Create a functional to-do list application focusing on IndexedDB or Web Storage.
Demonstrates the use of how different types of data can be stored within a browser.

Increment 3: Drawing Shapes with HTML5 Canvas

Create an application that allows users to draw shapes using HTML5 Canvas.
Explores HTML5 canvas functionality to benefit user experience.

Increment 4: OSM Data Listing App

Develop a web page that loads and lists OSM data in raw form.
Focuses on the use of importing data from external sources.

Following this strategy, after each increment; feedback can be documented back to the Revision Control System, necessary adjustments can be refactored, and the next set of features for the following iteration can be planned.

3.2 Testing

Testing creates a challenge towards this project. As Test Driven Development (TDD) will bring advantages such as improving code quality and the reliability of the applications, it was proposed at the start to conduct research on what testing frameworks should be used for the concept programs. Jumping from frameworks such as Jasmine to Mocha, I decided to finally use Jest as advised.

TDD with Jest was conducted for a minority of programs and managed to pass. However, as HTML5 is more frontend, TDD is not suitable so testing was executed differently. The other concept programs were still tested to ensure they served the correct functionality whilst no errors occurred, this was documented with test screenshots.

Towards the creation of the final application, Unit Testing like TDD will still be implemented to test certain features but we will also introduce other testing strategies such as regression testing to ensure when we implement new features no other functionality is declined. In addition, functional and performing testing will be conducted focusing on the performance of map rendering and the load of resources.

3.3 Revision Control System and Documentation

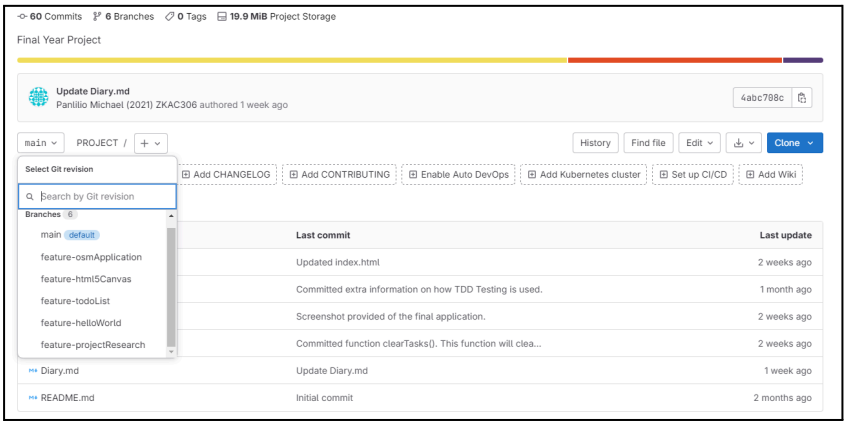


Figure 3.1: GitLab Repository of Offline HTML5 Maps ApplicationHTML

GitLab showcases this organised Offline HTML5 Mapping application. As you can see in ***Figure 3.1***, multiple feature branches are constructed to deal with each iteration (concept program). Frequent commits have been made on different branches indicating an organised repository.

No issues or conflicts have occurred when pushing or merging and although it has been a success using GitLab, it could've been more beneficial to release tag versions for each concept program. For improvement, this will be implemented in the final application.

4 Basic Web Page Development in HTML5

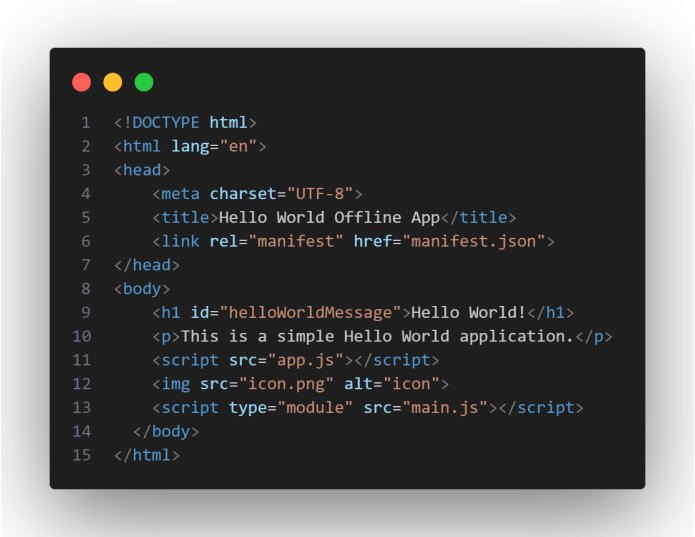
Web pages are what the World Wide Web is built upon. Web development involves creating and maintaining websites which use the three fundamental technologies to build those web pages:

HTML (Hypertext Markup Language)
JavaScript
CSS (Cascading Style Sheets)

These three technologies are crucial to use when building web pages as HTML provides the purpose of creating structured documents, JavaScript allows the implementation of complex features or functionality on web pages and CSS is used to define the design and layout of the webpage and the content within.

4.1 HTML

As HTML is the standard markup language to structure web pages, it consists of a wide series of elements that define the content. Elements such as headings, paragraphs, images and links etc. HTML brings several significant concepts towards web page development such as compatibility, being able to be universally supported by web browsers ensuring that web pages can be accessed and viewed by multiple users across different platforms as well as devices. This markup language also allows itself to be flexible and extensible whilst robust and so working very well along CSS, it can be customised and meet the needs of a wider branch of web development projects. As the population grows, more web pages amongst various categories are visited every day indicating HTML is an essential tool to use and a significant skill to learn.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello World Offline App</title>
    <link rel="manifest" href="manifest.json">
</head>
<body>
    <h1 id="helloWorldMessage">Hello World!</h1>
    <p>This is a simple Hello World application.</p>
    <script src="app.js"></script>
    <img src="icon.png" alt="icon">
    <script type="module" src="main.js"></script>
  </body>
</html>
```

Figure 4.1: Code Snippet from index.html to display Hello World

So required for this project was to create a concept program based on a '*Hello World* offline HTML5 application'. Focusing on the web page development side, ***Figure 4.1*** above displays a html file which when opened in the browser, outputs "Hello World" and a description followed by an image. As a web page can be much more complex, this simple file is briefly explained as follows:

<!DOCTYPE html>

Declares the document type and version of HTML which in this case is HTML5.

<html lang="en">

Opening tag for the document where the attribute *lang* indicates the document's language. This HTML tag contains the entire structure and content of a webpage and is followed with a closing tag </html>

<head>

Meta-information relating to the HTML document, including the character set, the title and links to external resources. CSS files would usually appear here.

<body>

Contains the main content of the document. Most elements such as headings, paragraphs, images and links would be mentioned inside the body. In addition, the *<script>* tag is used to implement JavaScript code or link your JavaScript file.

As mentioned in the title, we focus on HTML5 and more on the technologies provided. The main difference is that HTML5 is much more dynamic and includes these multimedia elements supporting a better visual and auditory. The HTML5 technologies: *Cache*, *Service Workers, Indexed DB, Web Storage* and *Canvas* are the building bricks to create this mapping application along with its' core functionalities ie. being offline, rendering and user interaction and customisation. HTML would also be backwards compatible **[9]**, allowing new features to be built on existing features but still allowing you to provide fallback content for older browser versions. In regards to browsers, although HTML5 is widely supported, there may be a difference in how browsers interpret and implement certain

features. As a security concern, HTML5 and the technologies it provides such as local storage can introduce potential security vulnerabilities. Furthermore, HTML5 may expand the Cross Site Scripting (XSS) attack surface **[10]** which could then lead to more malicious activities.

4.2 JavaScript

Programming Language, JavaScript is very significant as the majority of everything on the web would not be possible without it. From search engines to e-commerce to social media, JavaScript has become integral to Web Page Development to serve interaction and complexity in the applications. This is what is used to serve the main functionality of these documents, I'd call it the brains. In this project, it's crucial to use JavaScript towards the making of the application as the HTML5 technologies along with various functions are worked inside the script.
Therefore this is what will be used to achieve one of the main goals of the application; to access the document whilst in an offline status. In this case, we would implement a Service Worker inside the file so that JavaScript can register and control it to offer that offline functionality.
JavaScript is quite versatile as it can be used for both the front-end and back-end of web pages, seamlessly integrating with HTML and CSS. However, like HTML5, it can face challenges such as security attacks and vulnerability against attacks. In addition, performance and optimisation issues may be present too; one of the roots for these reasons is related to how the program uses or does not use an Application Programming Interface (API) **[11]**, which is quite significant to this project as it's in the decision if an API should be used or not to retrieve and render map data.

4.3 CSS

CSS is the style sheet language used to describe the presentation of the document. The importance of CSS is that what we see is a lot more user-friendly and more pleasing to the eye so that it's not hard to navigate across the content. Without it, web pages would just be plain text on white backgrounds with a chance of a horrible layout.

Unlike ***Figure 3.1***, a more complex HTML file would have classes inside their tags where CSS can recall those classes using a selector and then proceed to design those containers.

Figure 4.2: Simple CSS Code Snippet Example

As shown in ***Figure 4.2***, we can see that the selector *.mapping-container* is what will match the class name called in the HTML document. Inside this class, we have a declaration which is the property and value pair to apply the styles to the following container. In this case, the *background* is the property and *red* is the value. There is a large variety of different properties and values to design a webpage.

As far as the concept programs required within this project go, CSS won't be as significant as they focus more on the functionality; however until the final application is in development, CSS will be used to make the application look more user-friendly and achieve a nice design overall.

5 Advanced Technologies

As mentioned HTML5 provides these technologies for various functionality, we focus on the ones to achieve that offline functionality. However, there are a few other advanced technologies that are beneficial towards other purposes, for instance, user interaction and customisation.

5.1 jQuery

From jQuery themselves, they're known to be a '*fast, small, and feature-rich JavaScript library'*. Furthermore, this library is designed to simplify various tasks such as Document Object Model (DOM) Manipulation, event handling, animations etc. and so indicate that its main significance is allowing JavaScript to be easy to use on web pages.

jQuery may serve a few disadvantages. Even though jQuery provides several tasks within its library, functionality may be limited depending on the amount of customisation used on a webpage, indicating that using raw JavaScript may be inevitable. In addition, using jQuery and running commands requires importing the library, and although the size of it may be relatively small, it can still be a strain and require more time to open. This can impact the final application as jQuery will be used to integrate a mapping library/API, and so this is crucial to ensure the interactive map renders properly especially when it goes into an offline status.

Making an interactive map will potentially offer more functionality for the user so jQuery offers tools such as jQuery Cloud Zoom and Image Lens. This enables the final application to have a programmable interactive zooming of the map view in the window. **[12]**

5.2 HTML5 Canvas

Canvas is an API via JavaScript and the HTML *<canvas>* element providing the means of drawing graphics on a web page. These graphics can be manipulated and used for animation, game graphics, data visualisation and real-time video processing. It's very compatible across browsers and as it largely focuses more on 2D graphics, Canvas can be used to create more interactivity towards the following to benefit the user functionality of the mapping application:

Map Rendering

The *<canvas>* element can be used to create a drawing surface in order to render in the map. It can also be used to load map data such as coordinates and features and work along with JavaScript to draw those maps onto the canvas. In regards to the project, JavaScript will be suitably used to dynamically load and render the map tiles that OSM provides onto the canvas.

Handling User Interactions

Although jQuery provides tools to enable interactive zooming, canvas events can also be used to capture user interactions for panning and zooming across the canvas map.

Markers and Annotations

Canvas can also be used to draw markers and therefore can be used to mark the points of interest that a user may filter or other user-defined locations.

```
var canvas = document.getElementById('canvas');
var context = canvas.getContext('2d');

function drawRectangle(x, y) {
    context.beginPath();
    context.rect(x, y, 100, 250);
    context.fillStyle = 'yellow';
    context.fill();
    context.lineWidth = 7;
    context.strokeStyle = 'black';
    context.stroke();
}

function drawCircle(x, y) {
    context.beginPath();
    context.arc(x, y, 70, 0, 2 * Math.PI, false);
    context.fillStyle = 'green';
    context.fill();
    context.lineWidth = 7;
    context.strokeStyle = 'black';
    context.stroke();
}

function drawTriangle(x, y) {
    context.beginPath();
    context.moveTo(x, y);
    context.lineTo(x + 100, y + 250);
    context.lineTo(x - 100, y + 250);
    context.fillStyle = 'blue';
    context.fill();
    context.lineWidth = 7;
    context.strokeStyle = 'black';
    context.stroke();
}
```

Figure 5.1: Code Snippet from drawShapes.js

For a concept program to draw shapes using HTML5 canvas, we can see that from ***Figure 4.1***, the *'getContext'* method is used to obtain rendering context associated with the canvas and in this case, specifies that 2D rendering context is being requested. Therefore, the variable *context* can now be

used within functions to construct shapes. For example, the function *drawRectangle* defines different types of properties and executes different types of methods to construct a rectangle.

6 Developing an Offline HTML5 Application

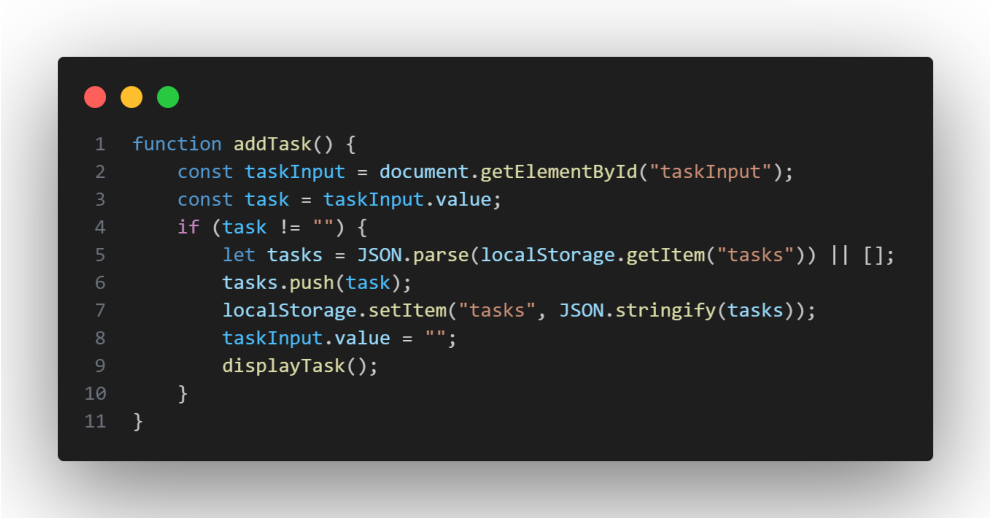
HTML5 providing offline technologies motivates the project. As navigation allows us to get from A to B, many of us would be lost without mapping applications and so it's quite crucial to develop one with the speciality of working offline as users can still access geographical data in an area with limited or no internet access.

6.1 Web Storage

Web Storage provides mechanisms to enable websites to store persistent data on users' devices with a large capacity and no information sent in Hypertext Transfer Protocol (HTTP) headers. These two mechanisms are as follows:

sessionStorage
It will maintain a separate storage area for each given origin that's available as long as the browser is open.

localStorage
Will serve the same function but is persistent as the data is still saved during the closing and opening of a browser.



```
function addTask() {
    const taskInput = document.getElementById("taskInput");
    const task = taskInput.value;
    if (task != "") {
        let tasks = JSON.parse(localStorage.getItem("tasks")) || [];
        tasks.push(task);
        localStorage.setItem("tasks", JSON.stringify(tasks));
        taskInput.value = "";
        displayTask();
    }
}
```

Figure 6.1: Code Snippet from todoList.js using localStorage

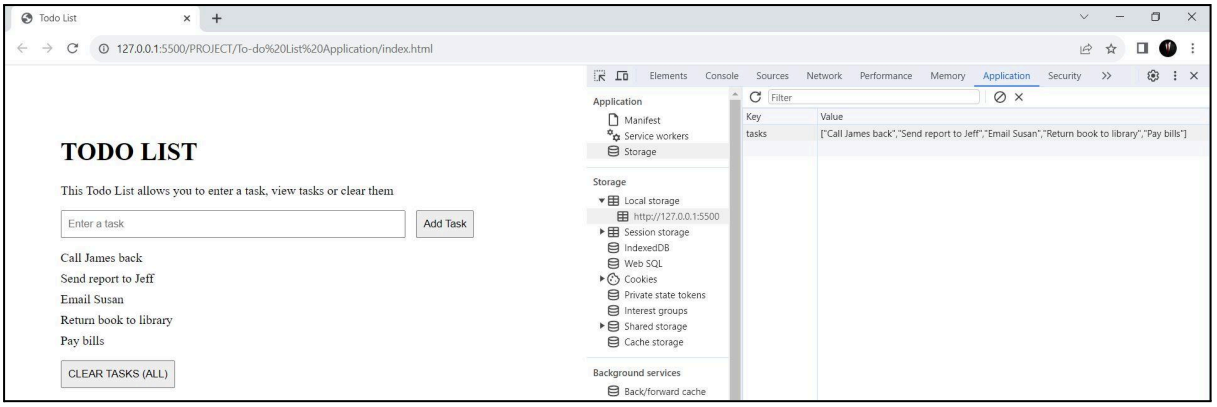


Figure 6.2: Todo List Application

Figure 6.1 was taken from the concept program required to create a *Todo List application*. It focuses on the use of Web Storage and storing the user's tasks within. Therefore, after checking if the user's input is not an empty string, a new array of *tasks* is declared where localStorage will retrieve any added tasks already stored. The user's input will be then pushed into the *tasks* array which will then be updated back into the localStorage of the browser. As the results are shown in ***Figure 6.2***, we can see the tasks are stored in the local storage of Google Chrome.

The concept of Web Storage brings several benefits towards the projected mapping application. For instance, it can save both user preferences and recent searches. Web Storage can provide storing map centre coordinates or zoom level so that users can go back to where they last looked when revisiting the application. In addition, it can also store a user's recent or saved search to build a personal history for them. Although ***Figure 6.1*** focuses on localStorage, in the final application sessionStorage can also be utilised to save temporary map data during user interactions.

However, if this application was storing a user's personal data, it may be wise to encrypt the data as it could pose a possible security risk since any user with access to that browser could see Web Storage data. **[13]** Moreover, Web Storage is also vulnerable to XSS attacks, and can steal all the data stored in a client's browser or change client settings **[14]**, therefore mitigation strategies such as validating and sanitising data must be introduced to ensure data is non-malicious.

In summary, Web Storage is a strong tool and can be beneficial in storing large amounts of key-value data which might be essential towards this project; however, data should be encouraged to be secure to prevent itself from attacks.

6.2 IndexedDB

A low-level API used for client-side storage of significant amounts of structured data, including files/blobs. **[15]** Similar to Web Storage, however, IndexedDB is useful for storing large amounts of structured data. Furthermore, IndexedDB will offer more complex querying capabilities when handling extensive data.

Where Web Storage will mostly likely store temporary user data, IndexedDB can be utilised by saving OSM data within its database, allowing efficient retrieval when the user loads the map. Highlighting

one of the main goals of this project, IndexedDB can store the map data locally providing access to the map even in an offline status. In addition, it can support user requests for instance. If a user wants to search for a certain area, IndexedDB can provide and use queries to retrieve that specific map data.

Using IndexedDB to achieve offline functionality may be challenging and so considering the complexity, it's best to ensure that the user experience is still understandable and efficient. Data can also become inconsistent if synchronisation is not handled carefully so the implementation of mechanisms may be useful to ensure accurate synchronisation for offline data. IndexedDB is also considered as a NoSQL database, which means that performing Structured Query Language (SQL) injections are not possible **[14]**, however like the other technologies, security attacks are still posed as a threat.

6.3 Service Workers

Service Workers are enabled by default in all modern browsers offering efficient compatibility. **[16]** I'd also agree that this technology plays a crucial role in enabling the offline purpose. Working well along with Cache, Service Workers are known as scriptable network proxies that run in the background to intercept and respond to network requests and will mostly deal with caching assets to achieve a loadable web page in offline status. Using Service Workers within this project is significant towards the fact it can cache the essential assets such as *HTML, CSS* and *JavaScript files*, (*also map tiles)* when the page is visited.

```
const CACHE_NAME = 'hello-world-cache-v1';
const urlsToCache = [
  '/',
  'index.html',
  'main.js',
];

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Cache opened');
        return cache.addAll(urlsToCache);
      })
  );
});

self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(response => {
        return response || fetch(event.request);
      })
  );
});
```

Figure 6.3: Code Snippet from service-worker.js to display Hello World program offline

Tracing back to the first concept program was a simple '*Hello World* HTML5 application', it was also required that the program has the ability to load without an internet connection, and that's why a Service Worker comes into play. So as you can see at the beginning of ***Figure 6.3***, it caches the assets *index.html* and *main .js*. Event listeners are also implemented to both install and fetch the cache: Once the Web Page is loaded, the Service Worker will install itself and then cache the data that are added. If the application is trying to be visited without an internet connection, then the Service Worker will intercept network requests and decide whether to fetch the response from the cache in which case, it will.

In conclusion, managing a Service Worker will contribute to the success of this offline mapping application and should be efficient to ensure a reliable user experience. However, cache management should still be overlooked as incorrect cache management can lead to displaying outdated content and introduce data inconsistencies towards the mapping data. In this case, it's best that thorough testing is conducted to make sure no conflicts are present to mitigate this risk.

6.4 Cache API

As the last contributor towards developing an Offline HTML5 Application, Cache API is a powerful tool to store and retrieve network requests and responses in a cache. As mentioned it's often used in conjunction with Service Workers, Cache API is used to ensure resources can be accessed even when they are offline. It also introduces the benefit that by caching frequently used resources, data can be loaded more quickly from the local cache towards the application, highlighting the reduction of repeat requests. This therefore improves the overall performance and responsiveness of the mapping application enhancing the user experience.

Concerning the Offline HTML5 application, Cache API can bring a few challenges. One of the main challenges is the storage limitation it imposes. Especially if map data is being imported which is relatively large, this dataset needs to be handled well so assets are rendered properly. Again, like Service Workers, it needs to be ensured that data is frequently updated and that the cache doesn't contain outdated data.

So even though this application offers access to the data when offline, both Service Workers and Cache should utilise themselves when it does have online access to ensure content is up to date and correct.

7 OpenStreetMap Data Representation

The map data that is used will be extracted from OSM and imported into the mapping application. Turning these complex data sets into maps gives us a better understanding of the information and so the visual representation presented can be manipulated to implement user interaction and customisation.

7.1 OpenStreetMap

OpenStreetMap(OSM) poses as a free and collaborative geographic database offering an editable map of the world. It's important to make aware that OSM is represented by four types of data: nodes, ways, relations and tags, all of which are constantly being edited by volunteers. **[17]** Therefore, whether it's decided that the map is imported through a library or its raw Extensive Markup Language (XML) data, the large data set needs to be updated regularly as it's constantly edited. OSM also brings versatility as it includes various detailed infrastructure regarding roads, building parks etc. Due to this, this increases the user's scope for map interaction.

The open-source software will also bring issues related to the quality, standardisation and completeness of the data used. **[17]** As said before, the retrieval of OSM data needs to be regularly updated and so whilst the application needs to work offline, accessing map data needs to be efficient. As it is a collaborative project, efforts by others can produce inaccuracy and lead to inconsistencies. In terms of standardisation, OSM data becomes available in different formats so this project needs to ensure that the application ensures compatibility with the different data formats that it may produce.

To conclude, maintaining OSM data may be challenging, although it produces a lot of opportunities for efficient user functionality as this data can be manipulated well as long as it's interpreted correctly.

7.2 Vector vs. Image Tile Maps

Vector Maps contain geographic data as a set of vectors which allow the dynamic rendering of map features. On the other hand, Image Tile Maps differ as they divide into small image tiles that have already been pre-rendered and loaded by the mapping application when needed.

Both concepts have their importance towards the application. Vector Maps are significant as they allow real-time rendering. Especially when user interaction such as zooming and panning will be implemented, it would be beneficial for the map to be dynamically redrawn based on the geographical data it owns. It also focuses on user customisation as vector maps offer flexibility in styling and design where elements can be styled differently on user preferences. For example, if a user wanted to highlight 'cafes only in a certain area', vector maps should be flexible to showcase cafes on the map. For Image Tile Maps, as they are divided into fixed images, they provide a good user experience as the visual experience will remain consistent. In similarity, both serve offline use, as their data can be stored locally and so whichever used the mapping application can still be accessed without an internet connection.

In summary, both Maps bring their benefits towards this application. In deciding on what type of map representation to use for the final application, there are two types of approaches I would pursue:

Vector Maps Approach:
Using vector maps allows real-time interactivity which is crucial as the Offline HTML5 Application requires both simple and extensive user functionality. Although it is more complex towards data processing and map rendering, I believe that it will enhance user experience more.

Hybrid Approach:
An approach to use both can also be considered. In this case, vector maps can be used for real-time interaction where image tiles will be used for offline access therefore in the offline status, data processing won't be as complex and pre-rendered tiles will be faster to load. However, the actual implementation to use both may be more time-consuming and challenging.

8 Code Development of Final Application

As we go through the development of the Offline HTML5 Maps Application, we focus on branching off all concept programs and using those principles towards the final application, especially with the HTML5 technologies that have been included. As significant as user readability is with a strong HTML structure and CSS design, what is majorly focused on in this final application is the use of JavaScript, where multiple script files have been developed to ensure a robust and efficient execution of various functions enhancing usability.

8.1 Leaflet Implementation

Quickly growing in popularity within the web development community due to its lightweight and wide open-source JavaScript library **[18]**, Leaflet heavily supports this application and implementing it in the mapping application involves integrating its functionality to display a rendered OSM map for the user to visually see on their web page and interact with the map data. Using Leaflet gave the opportunity to work with the features it provides such as tile layers, markers and popups which can all be customised allowing a better experience for the user.

Two ways were using Leaflet, either locally or using a hosted version. Initially, this application started using a hosted version as it did render faster and would always be updated in real-time. However, I ended up choosing to download Leaflet to be used locally so that when it is in an offline status, it is more efficient for the service worker to cache map tiles highlighting the technique of image caching which is a supportive method to improve rendering. Also to mention, this change did not majorly affect any other part code as only the stylesheet link and script src were altered.

Figure 8.1.1: Code Snippet from app.js which features the use of Leaflet

As shown in ***Figure 8.1.1*** we can see the instance of Leaflet being used where the map is declared and that Leaflet declares both a minimum and maximum zoom limit as well as sets the view at a certain zoom level when the application is first opened.

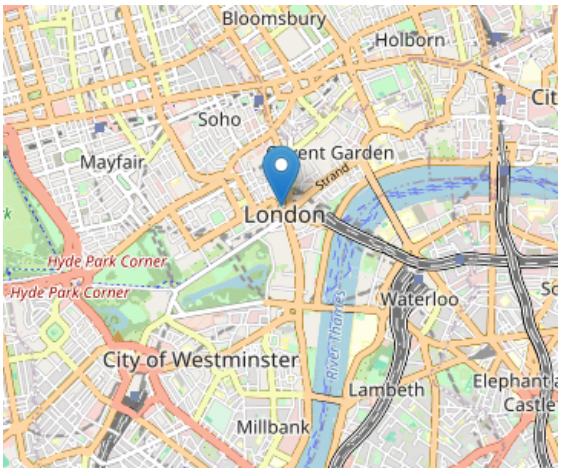


Figure 8.1.2: Leaflet Marker

As mentioned before, Leaflet also provides markers as shown in ***Figure 8.1.2***. As an advantage, these markers can be customised and therefore give the opportunity to easily present different types of data such as amenities. The benefit of using Leaflet is that the API is simple and intuitive to implement within the application and offers a wide range of extensions to extend functionality. Due to its flexibility and performance, Leaflet was an optimal choice to allow a visually appealing map that

8.2 Making it Offline (*Cache and Service Workers*)

One of the main functionalities is making this application work in an offline environment, and so we can branch off the 'Hello World' concept program that also uses a Service Worker to achieve the offline capability.

```
const CACHE_NAME = 'offline-cache-v1';
const urlsToCache = [
  '/',
  '/index.html',
  '/src/styles.css',
  '/src/app.js',
  '/src/dynamicSearch.js',
  '/src/dynamicRadius.js',
  '/src/savedLocation.js',
  '/leaflet/leaflet-src.js',
  '/leaflet/leaflet.css',
];

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => {
        console.log('Cache opened');
        return cache.addAll(urlsToCache);
      })
  );
});

self.addEventListener('fetch', event => {
  const { request } = event;

  if (request.url.startsWith('https://{s}.tile.openstreetmap.org')) {
    event.respondWith(
      caches.match(request).then(cachedResponse => {
        if (cachedResponse) {
          return cachedResponse;
        }

        return fetch(request).then(response => {
          if (!response || response.status !== 200 || response.type !== 'basic') {
            return response;
          }

          const responseToCache = response.clone();

          caches.open(CACHE_NAME).then(cache => {
            cache.put(request, responseToCache);
          });

          return response;
        });
      })
    );
  } else {
    event.respondWith(
      caches.match(request).then(response => {
        return response || fetch(request);
      })
    );
  }
});
```

Figure 8.2.1: Code Snippet of service-worker.js

Displayed in ***Figure 8.2.1***, there is a large array of URLs which will be cached when the service worker is installed in ***app.js***. The event listener is triggered when the service worker is first installed and then opens the cache and adds all the URLs to the cache.

Following the code, the event listener triggers whenever the web page makes a network request and then intercepts fetch requests and allows the service worker to respond with cached data or possible fetch data from the network and then proceed to cache it. In regards to OSM data, the fetch event listener checks if the request is for tiles from the OSM server. If it is, an attempt is made with a cached response if only available. If not, it fetches the request from the network, clones the response to then cache and return the response.

As dynamic data is queried and fetched through APIs, in an offline status, the application manages to achieve the capability of loading and displaying map data, as well as zoom and pan around the map.

8.3 Previous Searches (*Web Storage*)

As the Todo List explained in ***Figures 6.1 and 6.2***, the application also utilises web storage to specifically store and retrieve a user's previous search queries locally within the browser. This feature enables users to access their search history even when offline or when returning to the application at a later time. As a user performs a search for a location within the search bar, as long the location is known and the user gets directed to it within the map then their search then the local storage fetches that item and stores it in the list.

```
function addRecentSearch(query) {
  query = query.toUpperCase();
  var searches = JSON.parse(localStorage.getItem("recentSearches")) || [];
  var index = searches.indexOf(query);
  if (index !== -1) {
    searches.splice(index, 1);
  }
  searches.unshift(query);
  if (searches.length > 5) {
    searches.pop();
  }
  localStorage.setItem('recentSearches', JSON.stringify(searches));
}
```

Figure 8.3.1: Code Snippet from app.js using Web Storage to add a Previous search

```
function showSearchHistory() {
  var previousSearches = JSON.parse(localStorage.getItem("recentSearches")) || [];
  var dropdown = document.getElementById("previousSearches");
  dropdown.innerHTML = "";
  // Create a unique Set in order to remove duplicates
  var uniqueSearches = new Set();
  previousSearches.forEach(search => {
    var displayText = search.includes(',') ? search.split(',')[0] : search;
    uniqueSearches.add(displayText);
  });
  var uniqueSearchArray = Array.from(uniqueSearches);
  uniqueSearchArray.forEach(search => {
    var option = document.createElement("div");
    option.classList.add("previous-search");
    var searchElement = document.createElement("span");
    searchElement.textContent = search;
    searchElement.addEventListener("click", function () {
      document.getElementById('searchInput').value = search;
      searchLocation();
    });
    option.appendChild(searchElement);
    var removeButton = document.createElement("button");
    removeButton.textContent = "X";
    removeButton.id = "remove-button";
    removeButton.addEventListener("click", function (event) {
      event.stopPropagation();
      removePreviousSearch(search);
    });
    option.appendChild(removeButton);
    dropdown.appendChild(option);
  });
}
```

Figure 8.3.2: Code Snippet from app.js using Web Storage to show Previous Searches

As we can see in ***Figure 8.3.1***, all queries that are passed through the user get capitalised first before being added into the local storage in order to remove all duplicate previous searches. Additionally, ***Figure 8.3.2*** shows the function ***showSearchHistory()*** that when the search bar is clicked by the user, then the local storage retrieves the items within its array and displays them below the search bar. A remove button for each item that is stored in the local storage is also implemented so that the user can easily remove that previous search if desired.

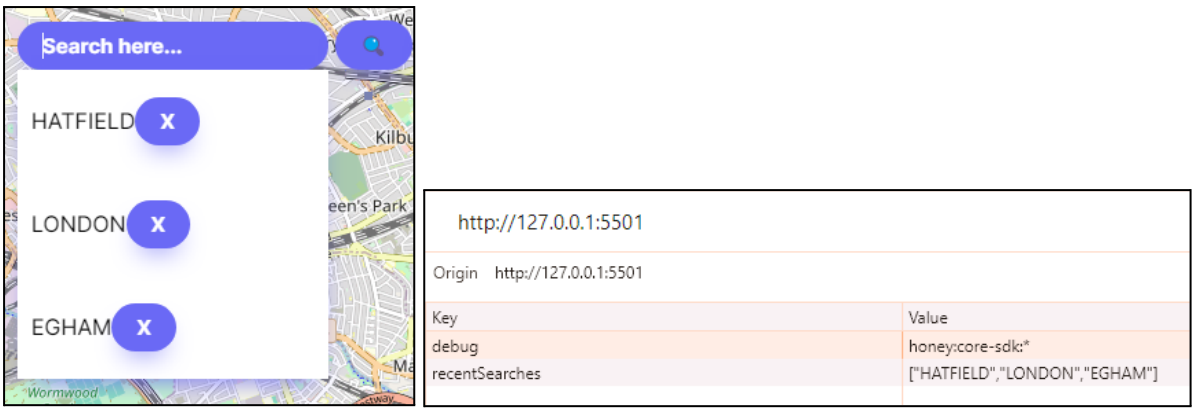


Figure 8.3.3: Previous Search Function shown in Maps Application and Local Storage

As displayed in ***Figure 8.3.3,*** we can see the execution of the function in which the previous search history is shown to the user as well as stored in the local web storage of the browser. Whether a user searches for a new location or a location that is already stored in the local storage, ***addRecentSearch()*** ensures that it prioritises the most recent search at the top of the list.

This feature provides a lot of convenience as users can easily access and revisit their past search queries. It also enhances the personalisation of the user experience by offering quick access to frequently visited locations. Due to localStorage having a limited storage capacity which potentially restricts the amount of previous search queries that can be stored, I decided to limit the amount of items that can fill the list to 5 to prevent overflow both visually and in storage. This however, can produce quite a drawback in significance to users with extensive search activity.

Overall, this demonstrates the maps application's ability to leverage Web Storage for storing user data and improving the user experience. In significance, it supports users to redirect themselves to past locations they may have potentially forgotten about. If this application was extended, it would be even better if this functionality could be synced to allow users to access their previous searches across different browsers.

8.4 Saved Locations (*IndexedDB*)

Although none of the concept programs used IndexedDB, its implementation in the mapping application allows users to store and manage their saved locations locally within the browser. As the low-level API provides a more robust and scalable solution compared to web storage, users are able to click anywhere on the map in which the application responds with a floating window displaying the location's address and the option to save that location as seen in ***Figure 8.4.1.***

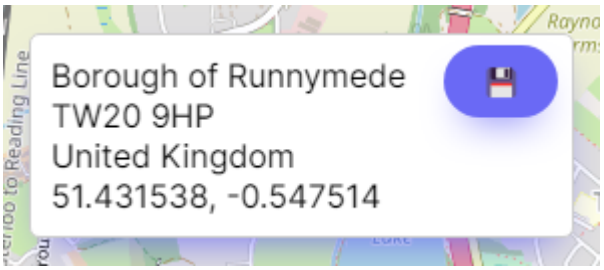
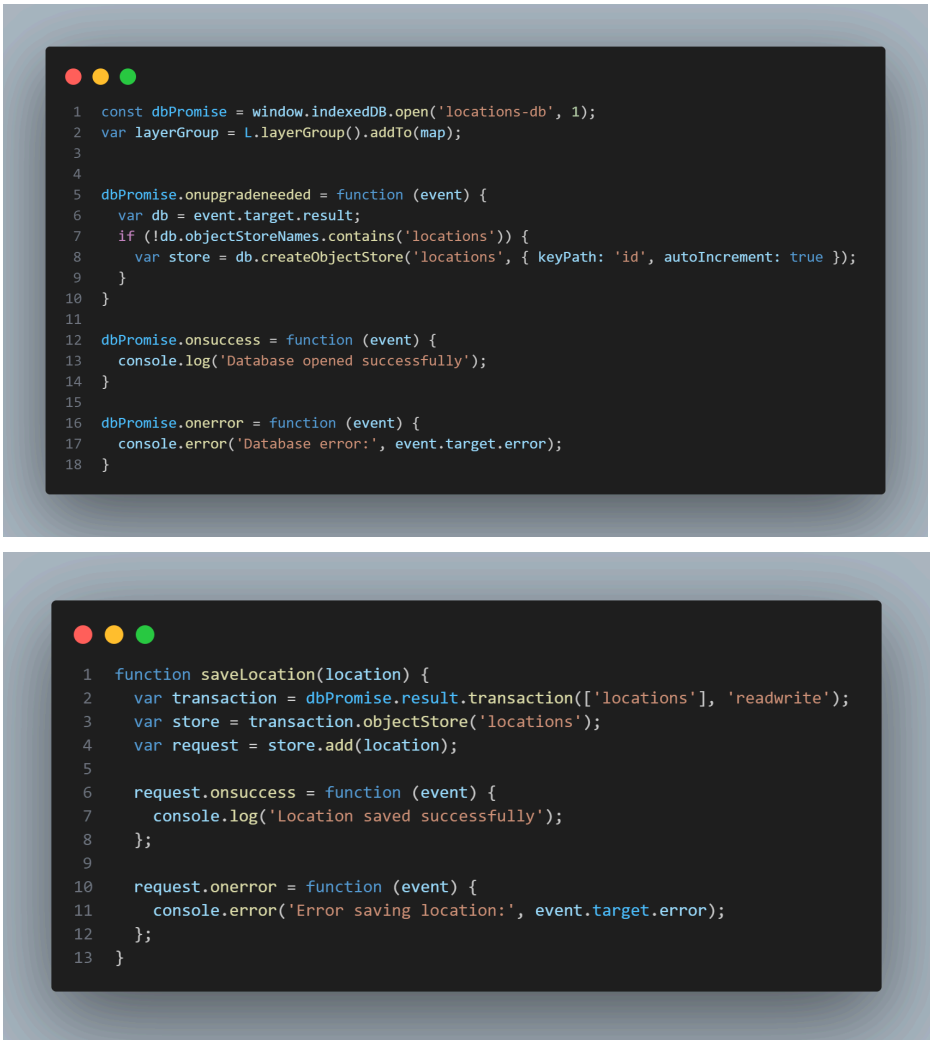


Figure 8.4.1: Floating Window to allow the user to save a location



```
const dbPromise = window.indexedDB.open('locations-db', 1);
var layerGroup = L.layerGroup().addTo(map);


dbPromise.onupgradeneeded = function (event) {
  var db = event.target.result;
  if (!db.objectStoreNames.contains('locations')) {
    var store = db.createObjectStore('locations', { keyPath: 'id', autoIncrement: true });
  }
}

dbPromise.onsuccess = function (event) {
  console.log('Database opened successfully');
}

dbPromise.onerror = function (event) {
  console.error('Database error:', event.target.error);
}
```

```
function saveLocation(location) {
  var transaction = dbPromise.result.transaction(['locations'], 'readwrite');
  var store = transaction.objectStore('locations');
  var request = store.add(location);

  request.onsuccess = function (event) {
    console.log('Location saved successfully');
  };

  request.onerror = function (event) {
    console.error('Error saving location:', event.target.error);
  };
}
```

Figure 8.4.2: Code Snippets using IndexedDB in savedLocation.js

As we see in ***Figure 8.4.2*** above, the program is responsible for opening an IndexedDB database named ***'locations-db'*** returning the promise that resolves when the database is successful when opening or updating. Furthermore, it initialises a new layer group from Leaflet within the database which will hold markers related to saved locations stored in ***'locations-db'***. The database also contains an event listener which becomes triggered when the database is being updated. This occurs by checking if the 'locations' object store exists in the database already and if not, it creates a new object store named 'locations' with an auto-incrementing primary key 'id'. This is then followed up by two more event listeners to alert the console if the location has been saved successfully or if there was an error when trying to save one.

saveLocation(location) is another function used by the button in ***Figure 8.4.1*** which is responsible for saving the user's desired location to the database. It proceeds to retrieve the object store from the transaction and adds the user's provided location data to that object store.

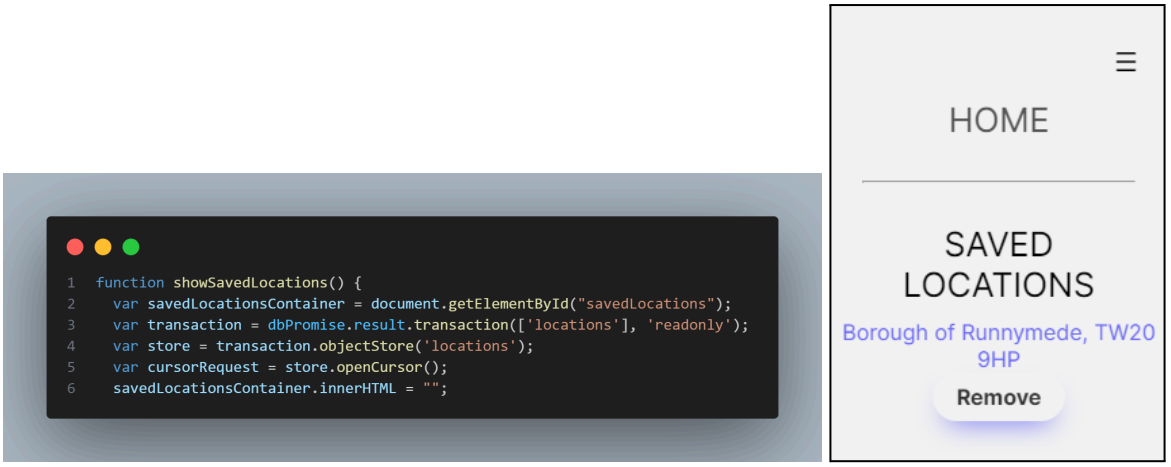


Figure 8.4.3: Code snippet of showSavedLocations() and Saved Location on Application

Figure 8.4.3 shows the beginning of the function which shows the users saved locations. It creates a container which can therefore be displayed in the application when a user desires to. It initiates a transaction on the IndexedDB database using ***'dbPromise'*** and continues to open the 'locations' object store to then request an iteration over the objects that are stored in the database using a cursor. Further down, ***showSavedLocations()*** manages to deal with the layer group that was created in the database to place the markers in the correct saved locations as well as display the corresponding address name as seen above. What also enhances user experience is that the addresses in the saved locations can be clicked to direct them to that location as well as remove the saved location which will give a request to the database for it to be deleted.

IndexedDB can handle larger amounts of structured data compared to Web Storage, and within this mapping application, it's more realistic that a user will save more locations than previous searches. In addition, its flexibility enables sufficient storage and retrieval of the user's saved locations. Although it offers a more robust and scalable solution within the browser, contributing to a more immersive mapping experience, managing errors and exceptions in IndexedDB transactions can be challenging therefore it requires more careful error handling to achieve smooth user experience as well as prevent significant data loss.

Extending this mapping application for an even more immersive application, it would be even better if multiple IndexedDB databases were initiated so that saved locations can be sorted into different categories. For instance, favourite locations, important locations, or even locations that a user wants to travel to.

8.5 Searching within a Radius (*HTML5 Canvas*)

Like in ***Figure 5.1*** drawing shapes onto the canvas, using HTML5 Canvas within this mapping application is very effective as it allows users to personalise their search by defining a circular area on the map in order to highlight different kinds of amenities within that specified radius. This feature initialises a canvas within the HTML body and utilises the canvas element for interacting with user-defined search areas.

```
function dynamicRadius(amenity, type, nameDefault, symbol) {
    markers.forEach(marker => map.removeLayer(marker));
    markers = [];

    var canvas = document.getElementById('radiusCanvas');
    var context = canvas.getContext('2d');
    canvas.width = map.getSize().x;
    canvas.height = map.getSize().y;
    canvas.style.display = 'block';
    map.getContainer().style.cursor = 'crosshair';
    var presetRadius = 3500;

    if (map.getZoom() != 13) {
        map.setZoom(13);
    }

    function handleClick(e) {
        var clickedPoint = map.latLngToContainerPoint(e.latlng);
        var circleBounds = getCircleBounds(clickedPoint, presetRadius / map.getZoom());
        canvas.style.display = 'none';
        map.getContainer().style.cursor = 'default';
        displayAmenities(circleBounds, amenity, type, nameDefault, symbol);

        map.off('click', handleClick);
        map.off('zoomend', handleZoomEnd);
        map.off('mousemove', handleMouseMove);
    }

    function handleZoomEnd() {
        if (map.getZoom() < 13) {
            map.setZoom(13);
        }
    }

    function handleMouseMove(e) {
        var cursorPoint = map.latLngToContainerPoint(e.latlng);
        context.clearRect(0, 0, canvas.width, canvas.height);
        context.fillStyle = 'rgba(94, 93, 240, 0.5)';
        context.beginPath();
        context.arc(cursorPoint.x, cursorPoint.y, presetRadius / map.getZoom(), 0, 2 * Math.PI);
        context.fill();
        context.strokeStyle = 'blue';
        context.beginPath();
        context.arc(cursorPoint.x, cursorPoint.y, presetRadius / map.getZoom(), 0, 2 * Math.PI);
        context.stroke();
    }

    map.on('click', handleClick);
    map.on('zoomend', handleZoomEnd);
    map.on('mousemove', handleMouseMove);
}
```

Figure 8.5.1: Code Snippet of dynamicRadius() which utilises HTML5 Canvas

As we can see, ***Figure 8.5.1*** shows a recursion of functions. As we focus more on the utilisation of HTML5 Canvas within ***dynamicRadius()***, we can see that it initialises a canvas element, retrieving it from ‘radiusCanvas’ from the HTML body and then proceeds to get the 2D rendering context of the context, which allows users to draw on it.

The function then retrieves the width and height of the map in order to set the canvas to the same size. After, the function ***handleMouseMove()*** is what allows the user to draw the radius circle onto the map. Similar to the concept program, the function draws a transparent circle on the canvas to therefore represent the radius around the cursor point as the mouse moves over the map.

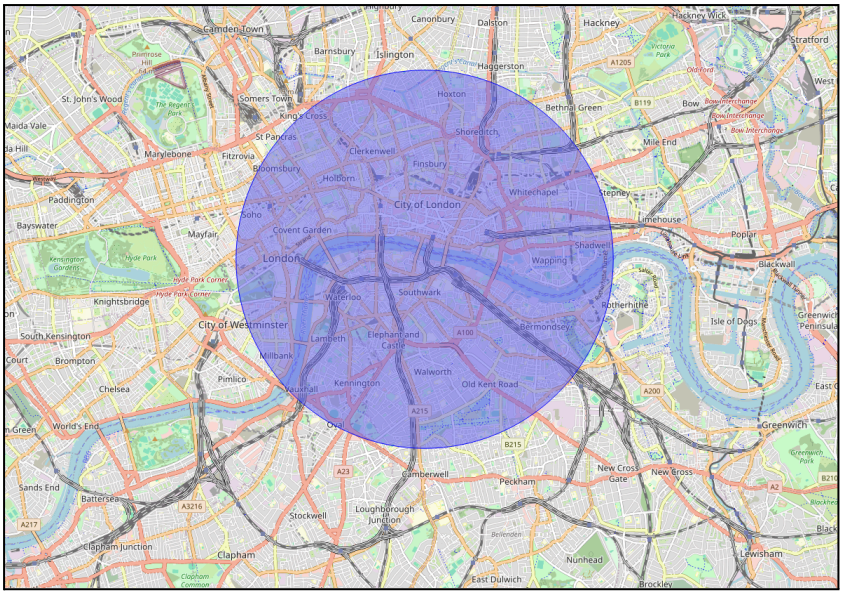


Figure 8.5.2: Radius Search on Map Application using HTML5 Canvas

Figure 8.5.2 shows this implementation of HTML5 Canvas as the user can decide in which area they want to search for their desired amenities. After clicking the area they want to cover, ***handleClick()*** will then manage to calculate the bounds of the circle around the clicked point, hide the canvas and then change the cursor back to default to call ***displayAmenities()*** which then uses Overpass querying for results.

The HTML5 technology of Canvas provides a flexible and customisable platform for a better visual representation of map data enhancing user understanding and engagement. Not only that, but users can dynamically search and see updates on the canvas in real-time underscoring the response of instant feedback. As HTML5 Canvas may require significant development effort, especially when handling complex search queries or large datasets, it is best to restrict users to zoom out past a certain zoom level to ensure a quicker and more defined search. Ultimately, enhancing the mapping application's functionality and usability leverages modern web technologies to deliver a visually engaging and interactive search experience directly within the map interface.

8.6 Nominatim and Overpass API

Nominatim and Overpass API are useful web services used for accessing OSM data, serving large benefits towards this mapping application. They both serve different purposes with distinct functionalities.

Firstly, Nominatim is an open-source search engine that consumes OSM data **[19]**, providing geocoding and reverse geocoding capabilities. This therefore allows users to search for places and addresses based on their textual descriptions. Nominatim is used in various instances in this mapping application but is mainly used within the search input and the saving locations.

```
function geoCode(lat, lon) {
  return fetch('https://nominatim.openstreetmap.org/reverse?format=json&lat=' + lat + '&lon=' + lon + '&addressdetails=1')
    .then(response => response.json())
    .catch(error => {
      console.error('Error fetching location details:', error);
    });
}
```

```
function searchLocation() {
  var query = document.getElementById('searchInput').value;
  if (query.length > 0) {
    fetch('https://nominatim.openstreetmap.org/search?format=json&q=' + encodeURIComponent(query))
      .then(response => response.json())
      .then(data => {
        if (data.length > 0) {
          var result = data[0];
          var lat = parseFloat(result.lat);
          var lon = parseFloat(result.lon);
          map.setView([lat, lon], 14);
          addRecentSearch(query);
          clearPreviousSearches();
        } else {
          alert('Location not found.');
        }
      })
      .catch(error => {
        console.error('Error:', error);
      });
  } else {
    alert('Please enter a location.');
  }
}
```

Figure 8.6.1: Code Snippet of functions geoCode() and searchLocation()

Starting with the first code snippet in ***Figure 8.6.1***, the function of ***geoCode()*** takes latitude and longitude coordinates as parameters to perform reverse geocoding by sending a GET request to the OSM Nominatim API. ***geoCode()*** is then used inside other functions across the application's script files. For instance, if a user clicks anywhere on the map, then the function will retrieve the coordinates for that position and display the address details including place name, postcode and coordinates as seen before in ***Figure 8.4.1***.

In terms of the search Input, ***searchLocation()*** will take the user-provided query string and then send a GET request to the Nominatim API endpoint for searching locations, passing the query string as a parameter. With this request, if a location is found, then it successfully directs the user to that location on the map. If no location is found then the console will catch an error and alert the user that a 'Location is not found'.

Nominatim provides a simple and straightforward API to perform geocoding and reverse geocoding queries, therefore making it easy to integrate into this mapping application. Also due to its open source, it provides access to a wide range of OSM data. However, it is quite limited as it only performs basic geocoding and reverse geocoding functionality and so lacks advanced querying capabilities for retrieving specific subsets of OSM data. For example, searching for something like 'McDonalds' would not direct you to or offer different locations of the fast food chain McDonald's but direct you to a random location called 'McDonalds'.

In contrast, Overpass API provides advanced querying capabilities for OSM data. Users are able to perform complex queries to the Overpass API with parameters such as location, object type or other desired properties and retrieve a response containing information for the features that satisfy the query. **[20]** Overpass API provides a range of advantages, where one is their custom querying capability, allowing users to define custom queries to extract precise data they need from the OSM database and enabling highly targeted data retrieval. As it's optimised for performance and can handle complex queries efficiently, the API also provides data in different formats such as XML, JSON and GeoJSON allowing flexibility and enabling a stronger compatibility across browsers.

However, due to its steeper complexity compared to Nominatim, Overpass API does require more understanding due to its Overpass Query Language (QL) and the structure of OSM data. In addition, it is also resource-intensive, as it can perform complex queries on large datasets. Therefore, performance issues can potentially arise hence why I implemented a limit on map zoom levels when using ***radiusSearch()*** for better search results.

To conclude, both web services are essential tools for accessing and querying OSM data offering users a better experience and enhancing usability when using the mapping application. However, one flaw did arise with the program in which it would be even better if autocomplete was implemented working alongside Nominatim within the search input. Due to the basic geocoding services Nominatim provides, it is quite challenging to achieve this feature.

8.7 Querying and Displaying OSM Data with Overpass Turbo

In regards to Overpass API, Overpass Turbo is also a web-based tool that works alongside Overpass API. Using this tool has resulted in the application managing to achieve more complex features such as allowing the user to search for interesting amenities as well as allowing the dynamic display of those different amenities.

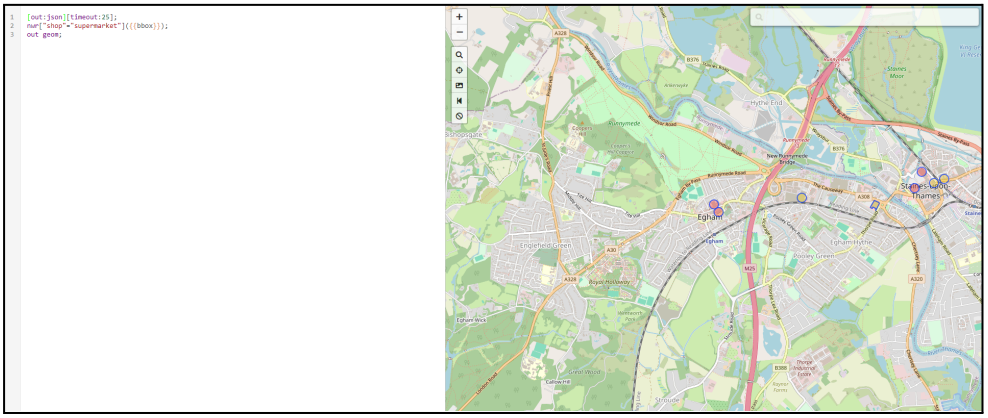


Figure 8.7.1: Overpass Turbo Querying Dynamic Data

As seen in ***Figure 8.7.1***, using different keys and values given by OSM, we can utilise this tool to support our mapping application. Where 'shop=supermarket' displays supermarkets in a given area, we can work with various amenities to allow the user to have a more extensive search.

```
function displayAmenities(bounds, amenity, type, nameDefault, symbol) {
    var southWest = bounds.getSouthWest();
    var northEast = bounds.getNorthEast();
    var overpassQuery = `[out:json];
            (
                node["${amenity}"="${type}"](${southWest.lat},${southWest.lng},${northEast.lat},${northEast.lng});
                way["${amenity}"="${type}"](${southWest.lat},${southWest.lng},${northEast.lat},${northEast.lng});
                relation["${amenity}"="${type}"](${southWest.lat},${southWest.lng},${northEast.lat},${northEast.lng});
            );
            out body;
            >;
            out skel qt;`;

    fetch('https://overpass-api.de/api/interpreter', {
        method: 'POST',
        body: overpassQuery
    })
        .then(response => response.json())
        .then(data => {
            markers.forEach(marker => map.removeLayer(marker));
            markers = [];
            data.elements.forEach(element => {
                var lat, lon, name;
                if (element.type === 'node') {
                    lat = element.lat;
                    lon = element.lon;
                    name = element.tags.name || nameDefault;
                } else if (element.type === 'way' || element.type === 'relation') {
                    lat = element.center.lat;
                    lon = element.center.lon;
                    name = element.tags.name || nameDefault;
                }

                var customIcon = L.divIcon({
                    className: 'custom-marker',
                    html: symbol,
                    iconSize: [30, 30],
                    iconAnchor: [15, 15]
                });
                var marker = L.marker([lat, lon], { icon: customIcon }).addTo(map)
                markers.push(marker);

                marker.on('click', function (e) {
                    var floatingWindow = document.getElementById('floating-window');
                    var content = `<div>${name}</div>`;
                    floatingWindow.innerHTML = content;
                    floatingWindow.style.display = 'block';
                });
            });
        })
        .catch(error => {
            console.error('Error fetching data:', error);
        });
}
```

Figure 8.7.2: Code Snippet of displayAmenities() in dynamicRadius.

```
function highlightATMs() {
    dynamicRadius('amenity', 'atm', 'ATMs', '🏧');
}

function highlightRepairs() {
    dynamicRadius('shop', 'car_repair', 'Auto Repair Shops', '🔧');
}

function highlightBakeries() {
    dynamicRadius('craft', 'bakery', 'Bakeries', '🥐');
}

function highlightBanks() {
    dynamicRadius('amenity', 'bank', 'Banks', '🏦');
}

function highlightCafes() {
    dynamicRadius('amenity', 'cafe', 'Cafes', '☕');
}

function highlightCinemas() {
    dynamicRadius('amenity', 'cinema', 'Cinemas', '🎦');
}

function highlightClothes() {
    dynamicRadius('shop', 'clothes', 'Clothing Store', '👕');
}
```

Figure 8.7.3: Code Snippet of different amenities that can be displayed on Map

Therefore, branching off Overpass Turbo has led to the creation of the function ***displayAmenities()***. As displayed in ***Figure 8.7.2***, the function creates an overpassQuery which takes 'amenity' as a parameter which corresponds to ***Key*** in OSM map features which may be *amenity, shop, tourism etc.* and also takes a type as a parameter which corresponds to ***Value*** in OSM map features. In this case, the type/value would be *supermarkets, hotels etc.*

As ***Figure 8.7.3*** shows just a few of the different amenities that can be searched within the mapping application, whichever one is chosen will get passed through by constructing the overpassQuery and then sending a POST request to the Overpass API endpoint with the constructed query string. After, it converts the response body to JSON format and processes that data returned from the API to iterate over the retrieved elements in which it ultimately extracts the coordinates and name information of that certain amenity and adds the corresponding marker to the map with a custom icon. As a result, this function will display amenities within a specified area on the map, in this case within the radius that the user chooses as the function ***dynamicRadius()*** is used before executing ***displayAmenities().***

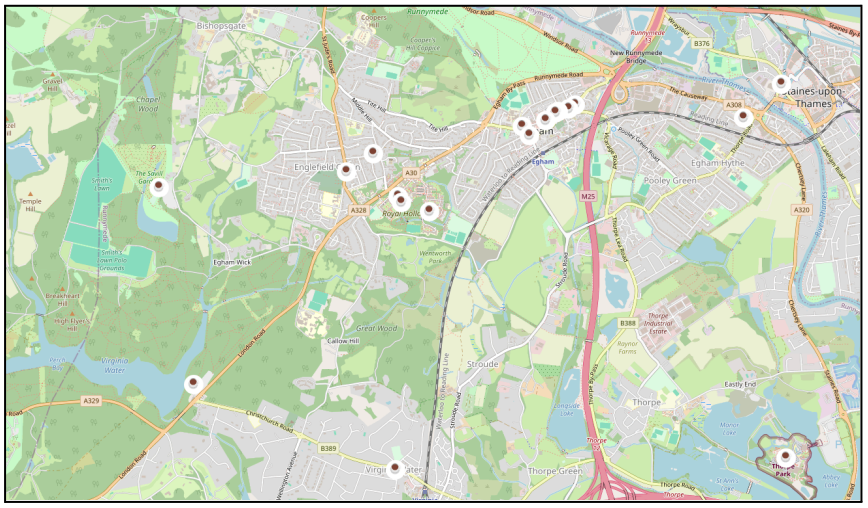


Figure 8.7.4: displayAmenities() in action highlighting cafes on screen with radius searching

Displayed in ***Figure 8.7.4***, the use of Overpass Turbo and Overpass API visualises the query results directly on an interactive map, allowing users to explore and analyse OSM data spatially in this case, exploring different cafes in Egham. ***dynamicSearch()*** also serves the same function as ***dynamicAmenities()*** using Overpass API but only serves as a shortcut function. Overall, allowing users to fetch and display amenities within a specified area on the map, provides an interactive and informative experience. As a maps application, it benefits many types of users from locals to tourists who may potentially want to view what type of services are nearby or what type of attractions there may be in a different location which allows them to plan ahead before visiting that location. Furthermore, the different types of amenities that the application offers can apply to various audiences. For example, someone who may be on the road may use the function to search for petrol services or even use the function to search for ATMs nearby.

8.8 Additional Features

As this mapping application already allows the user to load and display map data, zoom and move around the map, as well as focusing on the heavy use of HTML5 technologies, there are several other nifty features this application provides to give the user an overall, easier and better experience.

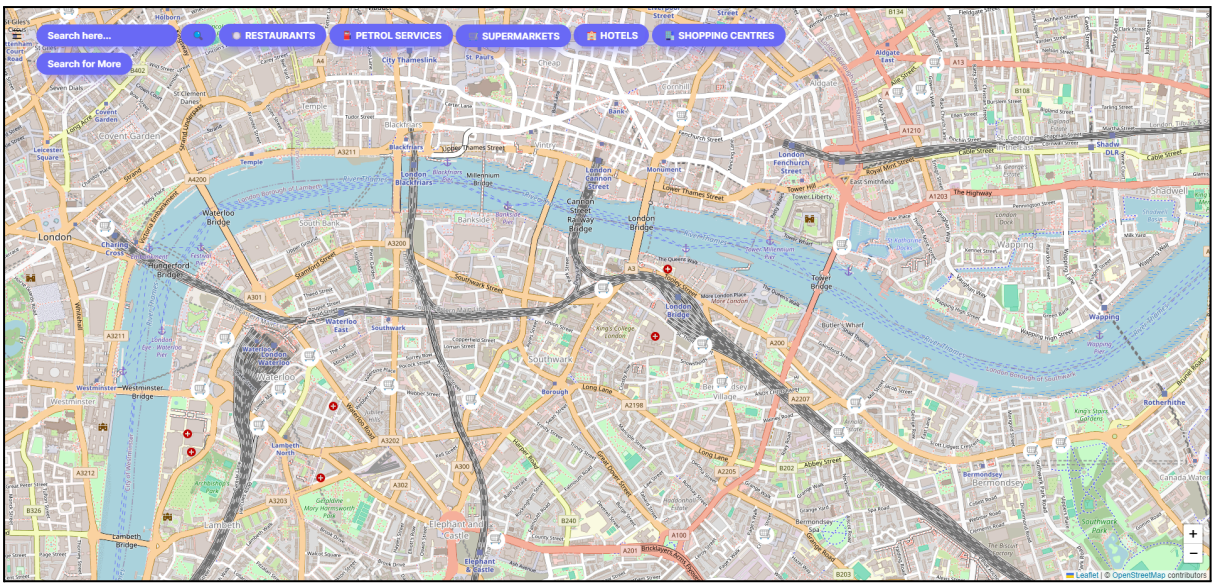


Figure 8.8.1: dynamicSearch() in the mapping application.

As mentioned, ***dynamicSearch()*** serves the same purpose as ***displayAmenities()*** however, ***dynamicSearch()*** is only used for the shortcuts above as shown in ***Figure 8.8.1.*** Wherever the map lies, pressing any of the shortcuts will set the map to a certain zoom level and search across the whole map canvas. Implementing this feature gives the user quick access to search for one of the popular amenities that would most likely be requested by the majority of the people.

Figure 8.8.2: dynamicSearch() in the mapping application.

Not only does providing custom markers support the user's readability, but also whether a user uses the shortcuts or radius search, they can proceed to click on the amenities that have been placed and the floating window appears to display the name of the corresponding amenity, enhancing a better user experience. In addition, if a user zooms out far enough or searches for another amenity, the markers are cleared so the map always looks clean and doesn't get flooded with markers.

```
function autocomplete(input, arr, searchFunctions) {
    let currentFocus;

    input.addEventListener("input", function (e) {
        const value = this.value;
        closeAllLists();
        if (!value) { return false; }
        currentFocus = -1;
        const autocompleteList = document.createElement("DIV");
        autocompleteList.setAttribute("id", input.id + "autocomplete-list");
        autocompleteList.setAttribute("class", "autocomplete-items");
        this.parentNode.appendChild(autocompleteList);
        arr.forEach(item => {
            if (item.substr(0, value.length).toUpperCase() === value.toUpperCase()) {
                const suggestion = document.createElement("DIV");
                suggestion.innerHTML = `<strong>${item.substr(0, value.length)}</strong>`;
                suggestion.innerHTML += item.substr(value.length);
                suggestion.innerHTML += `<input type='hidden' value='${item}'>`;
                suggestion.addEventListener("click", function () {
                    input.value = this.getElementsByTagName("input")[0].value;
                    closeAllLists();
                    const itemType = input.value.toLowerCase();
                    if (searchFunctions[itemType]) {
                        searchFunctions[itemType]();
                        document.getElementById('floating-searchWindow').style.display = 'none';
                        document.getElementById('map').style.display = 'block';
                        document.getElementById('overlay').style.display = 'none';
                        clearInput();
                    }
                });
                autocompleteList.appendChild(suggestion);
            }
        });
    });

    function closeAllLists(except) {
        const autocompleteItems = document.getElementsByClassName("autocomplete-items");
        for (let i = 0; i < autocompleteItems.length; i++) {
            if (except !== autocompleteItems[i] && except !== input) {
                autocompleteItems[i].parentNode.removeChild(autocompleteItems[i]);
            }
        }
    }

    document.addEventListener("click", function (e) {
        closeAllLists(e.target);
    });

    function clearInput() {
        input.value = '';
    }
}

autocomplete(document.getElementById("myInput"), amenities, searchFunctions);
```

Figure 8.8.3: Code Snippet of function autoComplete() in dynamicRadius.js

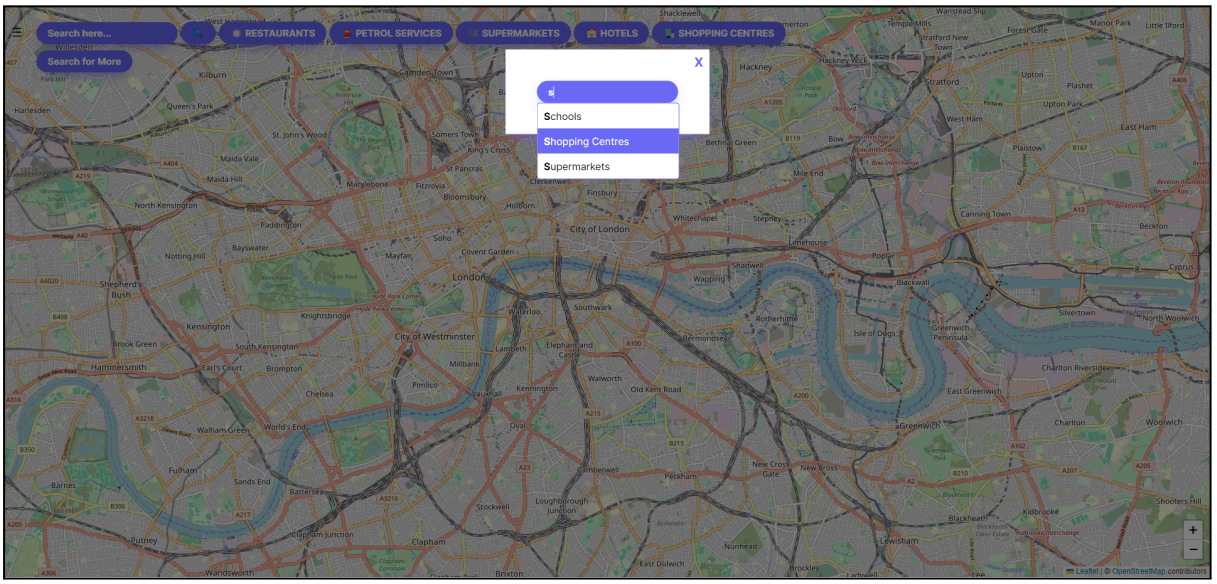


Figure 8.8.4: Search for More feature which contains autocomplete and radiusSearch()

The ***Search for more*** feature is where ***radiusSearch()*** can be used. The user can use this function to search various data, with more than 25 amenities provided. To benefit the user and reduce search time, I have implemented the function ***autoComplete()*** as seen in ***Figure 8.8.3*** to give suggestions on what they want to search for as soon as they start typing. After selecting an amenity, the user can proceed to use ***radiusSearch()*** to cover the area where they want to see the corresponding data.

8.9 Final HTML and CSS

As important functionality is, the way the website is structured and designed is crucial for providing an optimal user experience. Creating a visually attractive interface can captivate the user's attention, making them more likely to engage with the content. In addition, a good design will enhance user engagement by making it easier for users to navigate, find information, and complete tasks. By designing an application that shows clear navigation as well as an intuitive layout, the majority of users are capable of understanding the interface and interacting with it effectively.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Offline Mapping Application</title>
    <link rel="icon" type="image/map-icon" href="./src/favicon.ico">
    <link rel="stylesheet" href="./src/styles.css">
    <link rel="stylesheet" href="./leaflet/leaflet.css">
</head>

<body>
    <div id="map"></div>
    <canvas id="radiusCanvas"></canvas>
    <div id="floating-window"></div>
    <div class="searchContainer" id="searchContainer">
        <input type="text" id="searchInput" placeholder="Search here..." autocomplete="off">
        <button onclick="searchLocation()">🔍</button>
        <button onclick="searchRestaurants()">🍽 RESTAURANTS</button>
        <button onclick="searchServices()">⛽ PETROL SERVICES</button>
        <button onclick="searchSupermarkets()">🛒 SUPERMARKETS</button>
        <button onclick="searchHotels()">🏨 HOTELS</button>
        <button onclick="searchTowns()">🛍️ SHOPPING CENTRES</button>
        <div id="previousSearches" class="previous-searches"></div>
        <div class="extraSearchContainer">
            <button onclick="searchMore()">Search for More</button>
            <div id="overlay" class="overlay"></div>
            <div id="floating-searchWindow" class="floating-searchWindow">
                <form autocomplete="off">
                    <div class="autocomplete">
                        <input type="text" id="myInput" placeholder="Search for More...">
                    </div>
                </form>
                <button id="closeButton">X</button>
            </div>
        </div>
    </div>
    <div class="sidebarContainer">
        <div class="sidebar" id="sidebar">
            <div class="links">
                <a href="index.html">HOME</a>
            </div>
            <br><hr><br>
            <div class="links">
                <a href="#" onclick="showSavedLocations()">SAVED<br>LOCATIONS</a>
                <div id="savedLocations" class="savedLocationsContainer"></div>
            </div>
        </div>
        <div class="expand-btn" onclick="toggleSidebar()">☰</div>
    </div>
    <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"
        integrity="sha256-20nQCchB9co0qIjJZRGuk2/Z9VM+kNiyxNV1lvTlZBo=" crossorigin=""></script>
    <script src="./src/app.js"></script>
    <script src="./src/dynamicSearch.js"></script>
    <script src="./src/dynamicRadius.js"></script>
    <script src="./src/savedLocation.js"></script>
    <script src="./leaflet/leaflet-src.js"></script>
</body>

</html>
```

Figure 8.9.1: Full Code of index.html:

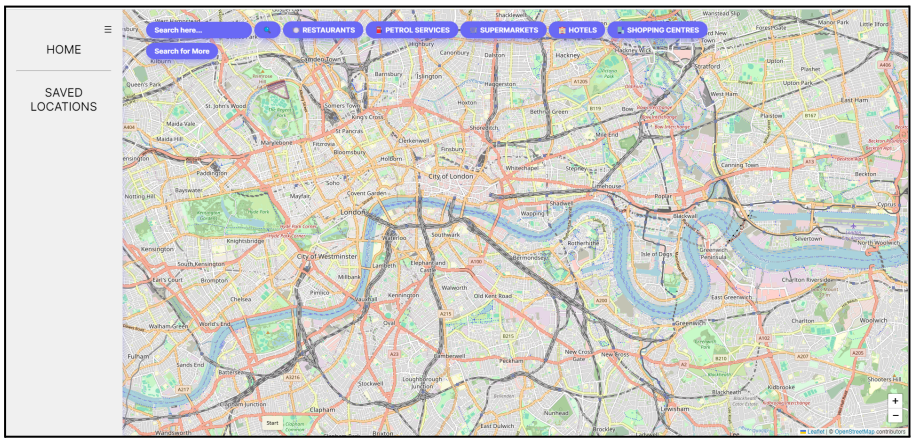


Figure 8.9.2: Final Offline HTML5 Maps Application

As we can see the full code of index.html, ***Figure 8.9.1.*** shows a simple structure where the body initialises the map, the canvas and multiple containers. With two main containers, ***searchContainer*** holds the search input and all the shortcuts whereas ***sidebarContainer*** holds the home button and the feature to show saved locations which can be toggled to be shown or hidden.

As a minimalist design aesthetic has rapidly gained popularity and respect in mainstream web audiences **[21]**, ***Figure 8.9.2*** shows what the final application looks like and supports this. Following the minimal design, implementing a sidebar is very beneficial for the user as if they wish to see their saved locations they can and if they wish to hide the sidebar, they are able to at any time. In addition, the implementation of the ***Search for More*** button is a better approach than implementing the search for dynamic data in the same search input as it may confuse users on what they're searching for. Overall, the minimalist design allows the user to have a better experience as their attention is majorly focused on the map as it fills the whole webpage and is hardly minimised throughout the use of this application.

9 Final Application Development Practices

9.1 Methodology and Source Control

Whilst developing the final application, the methodology that was used was still an agile style approach in which we broke the development down into a subset of phases. Especially where each concept program was capable of utilising some sort of feature in the final application, the approach allowed me to plan first, then design the functionality concerning the concept programs and after both develop and test the feature until I was satisfied. After a function was successfully programmed, then I would move on to the next phase and repeat the steps. If necessary adjustments needed to be refactored, whether a completed function was dependent on another incomplete function, then it would make sure that the code is properly refactored whilst still being robust.

An instance of this is the Todo List and the Previous Search feature. With the approach I took, it was made sure that during the design of the previous search feature, the code from the Todo List working with Web Storage can be thoroughly implemented in the final application and if it needs some sort of refactoring then it should be done until I can move onto the next feature.

In terms of source control, a final feature branch was constructed for the final application where frequent commits were made between different devices. The repository still maintained an organised structure with no issues or conflicts occurring when merging or fetching.

9.2 Code Issues

There were quite a few issues with certain parts of the code throughout development. Although there were a few major issues, most were minor issues that didn't take long to fix.

Duplicate Previous Search

As mentioned before, when implementing the previous search feature duplicates could easily pass through ways such as 'Egham' and 'EGHAM', although some recent search features may have this regardless of what a user types, I wanted to keep a clean interface therefore I made sure all inputs were capitalised before entering the list in the localStorage so that they can be compared to items already stored in the list in order to reduce the duplicate searches.

Removing Saved Locations Part 1

Although it was successful that a user could remove a saved location of the IndexedDB database, it would not update the markers in real-time and would need a page refresh to show the changes. To fix this, a layer was implemented for the markers to be added to and when once one was removed, ***showSavedLocations()*** would be called immediately after to remove that corresponding marker.

Removing Saved Locations Part 2

With the successful implementation of the layer for markers to be added and removed, it was inconsistent that markers would be removed. If only one location was saved, then it was easy to clear the marker in real-time. However, if there were multiple markers, once you try to remove one then it re-adds another marker to the same location making it seem it's still there. This was fixed by clearing the layer group every time ***showSavedLocations()*** is executed

Radius Search Still Being Used

Radius Search was a successful implementation after configuring the canvas in the HTML body and creating layers. However, using radius search with the circle disappearing after wherever you clicked would allow radius search to still be used just without the circle. By refactoring the code and adding event listeners to handle clicks, zooms and mouse movement, it was fixed where the behaviour to interact with the map stopped after using radius search once.

Slow Response When Searching for Dynamic Data

In populated locations such as London, searching for dynamic data like restaurants and cafes would respond in large coverage of data making the application either freeze or respond very slowly. To fix this, I implemented the feature of being zoomed in to a certain map level when using ***dynamicRadius()*** and restricted how far you can zoom out so the response is much quicker. With the shortcut buttons that use ***dynamicSearch()*** it will just zoom in to the specified level and display data.

CSS Styling

A few of the issues came from the way certain containers and items were designed which ruined the aesthetic of the page. With the majority of containers, I needed to configure the right z-index level, as some elements would not be able to be used as other elements had a higher priority and were overlapping. For instance, when clicking ***Search for More***, I was still able to use different functionality which was not intentional. Therefore, this was fixed by declaring the right z-index level for each element and container needed.

10 Bibliography

[1] Ijtihadie, R. M., Chisaki, Y., Usagawa, T., Cahyo, H. B., & Affandi, A. (2010, November). Offline web application and quiz synchronization for e-learning activity for mobile browser. In TENCON 2010-2010 IEEE Region 10 Conference (pp. 2402-2405). IEEE.

[2] Mooney, P., & Minghini, M. (2017). A review of OpenStreetMap data. Mapping and the citizen sensor, 37-59.

[3] Corcoran, P., Mooney, P., Winstanley, A. C., & Bertolotto, M. (2011). Effective vector data transmission and visualization using HTML5.

[4] Deb, P., Singh, N., Kumar, S., Rai, N., Naidu, D. P. S., & Iyengar, N. C. S. N. (2010). Offline navigation system for mobile devices. International Journal of Software Engineering & Application, 1(2).

[5] Netek, R., Masopust, J., Pavlicek, F., & Pechanec, V. (2020). Performance testing on vector vs. raster map tiles—comparative study on load metrics. *ISPRS International Journal of Geo-Information*, *9*(2), 101.

[6] Iváncsy, R., & Juhász, S. (2009). Approaches for Efficient Handling of Large Datasets. *Algarve, Portugal*, 143.

[7] Biagi, L., Brovelli, M. A., & Stucchi, L. (2020). Mapping the accessibility in openstreetmap: A comparison of different techniques. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *43*, 229-236.

[8] Mooney, P., & Corcoran, P. (2011, April). Accessing the history of objects in OpenStreetMap. In *Proceedings AGILE* (Vol. 353, No. 1, pp. 1-3).

[9] Bryan, H., & Anderson, M. (2014). Accessibility Issues in HTML5. International Journal on Recent and Innovation Trends in Computing and Communication, 2(11), 3415-3421.

[10] Dong, G., Zhang, Y., Wang, X., Wang, P., & Liu, L. (2014, May). Detecting cross site scripting vulnerabilities introduced by HTML5. In 2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE) (pp. 319-323). IEEE.

[11] Selakovic, M., & Pradel, M. (2016, May). Performance issues and optimizations in javascript: an empirical study. In Proceedings of the 38th International Conference on Software Engineering (pp. 61-72).

[12] Król, K., & Szomorova, L. (2015). The possibilities of using chosen jQuery JavaScript components in creating interactive maps. Geomatics, Landmanagement and Landscape, (2), 45-54.

[13] West, W., & Pulimood, S. M. (2012). Analysis of privacy and security in HTML5 web storage. Journal of Computing Sciences in Colleges, 27(3), 80-87.

[14] Kimak, S., Ellman, J., & Laing, C. (2012, June). An investigation into possible attacks on HTML5 indexedDB and their prevention. In 13th Annu. Postgrad. Symp. Converg. Telecommun. Netw. Broadcast.

[15] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[16] https://developer.moxilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers

[17] Vargas-Munoz, J. E., Srivastava, S., Tuia, D., & Falcao, A. X. (2020). OpenStreetMap: Challenges and opportunities in machine learning and remote sensing. IEEE Geoscience and Remote Sensing Magazine, 9(1), 184-199.

[18] Donohue, R. G., Sack, C. M., & Roth, R. E. (2013). Time series proportional symbol maps with Leaflet and JQuery. *Cartographic Perspectives*, (76), 43-66.

[19] Clemens, K. (2015). Geocoding with openstreetmap data. *GEOProcessing 2015*, *10*.

[20] Ehrig-Page, J. C. (2020). Evaluating methods for downloading OpenStreetMap data. *Cartographic Perspectives*, (95), 42-49.

[21] Meyer, K. (2015). Toward a Definition of Minimalism: Principles of Minimal Visual Design in Web Interfaces.

Appendix

Critical Analysis

Throughout the whole period from start to finish, I believe that the project was a success overall, however, there are significant aspects that I'm accountable for that potentially could've made my experience with the project even better. Firstly, as a responsibility, I'm pleased that I managed to book all my supervisor meetings in advance as well as attended and utilised every single one. I made sure that I had enough questions to ask and discuss with my supervisor to get an insight into the next steps in developing both the concept programs and my final application. In regards to professionalism, I also think that the use of the revision control system, GitLab, was a success as I managed to commit code in a frequent manner whilst also keeping an organised repository with feature branches. The early deliverables in which I had to construct concept programs were a success in each one, giving me the opportunity to learn about each HTML5 technology and how to use it to an efficient extent. Furthermore, it's very beneficial to learn how to utilise these technologies as these can be used in various ways for many different web applications and not just a mapping application.

Regarding the final application itself, I do believe that although the end product achieves the final deliverables as well as the suggested extensions, it could have been much better in the concept of the way the application was developed throughout as well as the way I implemented different features to offer a greater user experience. As a success, the final application managed to use HTML5 technologies: I managed to utilise cache and service workers to allow the application to work offline whilst still having the functionality that the final deliverables require. In an offline status, the mapping application still allows the user to load and display map data as well as zoom and move around the map. However, I think that using APIs such as Nominatim and Overpass may have not been the best implementation especially when going in an offline status as it fetches requests to their API server therefore the suggested extensions aren't capable of working well when there's no internet connection present. If I was able to go back, it would've been much better to offload data in the sense of either downloading the APIs to my repository or downloading JSON data to then query it later. In addition to offline functionality, it would've been a lot better to implement it first before everything rather than towards the end of development which has taken a toll through some stages of coding.

Moving on, I'm pleased to have successfully used both IndexedDB and Web Storage to allow users to store their data within their browsers. Learning how to use HTML5 technologies like these can be very beneficial towards different web applications that can store all types of data which can easily be retrieved. HTML5 Canvas was fun to use in both the concept program and the final application as having the ability to draw different types of shapes can result in offering a wide range of functionality for instance, using a radius search by drawing a circle to allow the user to search within that area. I do think that HTML5 canvas could have been used a bit more in the final application in terms of drawing stars for starred locations or circles to highlight dynamic data, however, the use of Leaflet and custom markers did offer a better alternative offering better readability for the user.

Apart from the use of HTML5 technologies, I enjoyed working on CSS styling and designing my web application. I believe that the minimal aesthetic was the best way to display this application making it easier for the user to read and navigate. As I aimed to show the map as much as possible, implementing a toggling sidebar was crucial as it allowed the user to use the functions within the sidebar only if they needed to. In addition to the floating windows, it was a significant feature to only show important information if a user wanted to.

It was crucial to keep a clean organised repository by creating directories where necessary, therefore making multiple javascript files was significant as well as I didn't want to overflow a single script page with tens of functions. However, I do see a flaw in which declaring containers and styling CSS should be within their own **.html** or **.css** file instead of initialising containers or even styling elements inside script files. For example:

```
content += ‘<button id=”saveLocationButton” onclick=”saveLocationButton()”>💾</button>;
```

In regards to future enhancements, if this project continued not only would I change the way the APIs have been implemented as said before but I would increase a larger catalogue of dynamic data for a user to search, a feature that would allow the user to search for a certain franchise and then respond to giving different locations of that corresponding franchise. Furthermore, I would increase user customisation and preferences by implementing a settings mode which can allow the user the ability to customise their markers, choose what they want to see or not, change font size and even choose between light mode and dark mode. Also mentioned before in ***8.4***, it would be beneficial to increase the amount of IndexedDB databases to allow the user to categorise their saved locations.

To conclude, I enjoyed developing this project very much and have realised the importance of both HTML5 technologies in web development as well as the significance of mapping applications, especially ones that can function offline. Mapping applications support almost everyone as we use navigation every single day, and if we are in a situation where there’s no internet connection present, an offline maps application will be very crucial. Although time has played a large factor where tasks are overdue by my own personal deadlines, I still managed to finish them through determination and perseverance. Despite the challenges throughout the project, I’m pleased to have maintained focus on achieving my goals.

Professional Issues

The rapid increase of digital technologies has revolutionised the way we navigate and interact with our surroundings. With that being said, a mapping application that uses extensive HTML5 technologies has the capability of being used in an offline status, when no internet connection is present. Utilising HTML5 technologies such as Web Storage, Service Workers, Cache and IndexedDB towards the development of an Offline HTML5 Maps Application stands as a testament to the seamless integration of functionality and convenience in the majority of all modern applications. However, in the development of creating a robust and efficient user application, developers find themselves at a dilemma where ethical considerations intersect with technical innovation. Therefore we can acknowledge the pressing concern that emerges: The fragile balance between privacy and functionality.

Represented as a significant advancement, an Offline Maps Application that utilises the key HTML5 components offers users the ability to access maps and navigation features without the reliance on a constant internet connection. Therefore, this application is designed to provide both convenience and reliability in being a crucial tool when navigating unfamiliar territories as well as exploring remote areas where internet connectivity may potentially be limited.

As mapping technology continues to evolve, the concerns surrounding user privacy have been caught to the front of public disclosure in which digital devices and the vast amounts of data that's been generated have raised apprehensions about the protection and privacy of personal information. This can make many users increasingly uneasy as they become more aware of the way in which their digital footprint is being monitored and utilised.

As one of the aims when developing an Offline HTML5 Maps application is to deliver enhanced functionality and customised experiences, we also face the ethical dilemma regarding the collection, storage and utilisation of user data. By trying to balance the goal of a high end mapping application with the significance to protect user privacy, it requires both careful consideration and diligent decision making.

As the HTML5 technologies are very significant components in making an offline mapping application improving user experience and performance, they also raise high privacy concerns and ethical dilemmas.

As Web Storage or Local Storage allows web applications to store data locally on a user's browser or device, this can enhance performance by reducing the need for repeated data retrieval. However, if sensitive data such as user preferences or location history is stored without adequate encryption or access controls, this leaves the data vulnerable to unauthorised access or even exploitation. Ethically, when developing this mapping application it must be considered whether the convenience of storing data locally outweighs the potential risks to user privacy and security.

In regards to Service Workers, which enables web applications to run scripts in the background allowing a very significant feature of offline functionality, it also raises concerns about potential misuse for tracking user behaviour or again collecting sensitive information without user consent

which can therefore lead to exploitation. Ensuring that Service Workers are used responsibly and transparently, it can potentially defeat the ethical dilemma and the risks of privacy infringement as long as users are informed about the purposes and implications of Service Worker usage.

As another privacy concern, caching mechanisms such as browser cache or image caching, very much improve application performance as it frequently stores accessed resources locally underscoring the offline functionality especially as it works alongside Service Workers. However, cached data may include sensitive information like user credentials or browsing history which can therefore pose a risk if this data is not secured or managed. As it benefits the performance of the web application it also balances the responsibility to protect user privacy and therefore a mapping application that uses Cache must implement a robust cache management technique to ensure user privacy and security.

Lastly, IndexedDB allows web applications to store large amounts (larger than web storage) of structured data locally, highlighting the functionality of offline access and enhanced performance. However, if IndexedDB is not implemented securely, then it can be vulnerable to attacks such as data breaches with the potential to expose sensitive user information. The ethical dilemma with IndexedDB centres on the responsibility of handling user data. A mapping application that may use IndexedDB must have robust security measures in place such as encryption and access controls in order to achieve the safeguard of data. Furthermore, it is also crucial to consider the implications of storing potentially sensitive data locally and ensure that users are informed and empowered to control their data.

The respect of user privacy is a grounded principle which asserts that users have the right to control their own personal information, and so through development it must be ensured that user privacy is upheld through the application's life cycle ranging from data collection and storage to usage and sharing. This therefore, underscored the importance of implementing robust privacy policies, obtaining explicit user consent for data collection and processing.

If user privacy is compromised for the sake of enhanced functionality, then significant ethical and practical implications may take place. For instance, if users acknowledge that their privacy is being compromised within the HTML5 components, they may lose trust in the application leading to negative publicity and damage to its reputation. In addition, collecting large amounts of user data that may be stored in an IndexedDB database can lead to sensitive data being exploited for malicious purposes such as identity theft or targeted advertising.

To conclude, as technology continues to advance, it is vital for developers of mapping applications to remain cautious and vigilant when upholding ethical standards as well as foster a culture of both privacy and responsibility in the environment of software development. If this is well achieved, it can be ensured that innovations such as Offline HTML5 Map applications serve to not only enhance functionality but also serve the fundamental rights and values of the users they serve.

Project Diary with Time Scale

DIARY FOR FINAL YEAR PROJECT -> Diary to allow me to note down and reflect throughout the final year project.

Friday 29/09/2023 - Supervisor Meeting: The first supervisor meeting consisted with a range of queries on how my Offline HTML5 APP should be executed along with other ideas that should be taken into consideration. Also spoke about time organisation and how to structure the final year project as a whole.

Thursday 05/10/2023 - Extensive Research Conducted research on HTML5 offline technologies, OSM data retrieval, and vector map rendering.

Friday 20/10/2023 - Additional Research In addition, I have researched different types of testing frameworks that will be suitable for HTML5 and JavaScript in order to use TDD when making the concept programs.

Tuesday 24/10/2023 - Start of TDD for Hello World Using Jasmine to apply TDD for the first concept program. I have not encountered any issues for this program yet although due to unexpected complexities I am behind the schedule that was made in the project plan.

Friday 03/11/2023 - Supervisor Meeting 2: Although only a few TDD tests have already been committed for one of the concept programs, it was suggested from the Supervisor to use Jest Testing framework instead of Jasmine/Mocha (Mocha was the most recent used). Mainly discussed about the overview and specification of the project as a whole. As a little behind schedule, it's appropriate to reflect, mitigate and evaluate towards the end. Supervisor also mentioned the two alternatives: Either implement OSM by myself or Install a library in which higher expectations are brought if that route is taken.

Friday 17/11/2023 - Concept Program 1 Done: Despite the delay, using Jest Testing framework we have successfully passed the two TDD tests for the Offline HelloWorld Program. Did encounter problem with service-worker and refactored the way it fetched the request. Should be able to load cache files in an offline status now.

Monday 20/11/2023 - Start Concept Program 2: Beginning of constructing a Todo list application using WebStorage. This application should contain several functions to add, display and clear tasks in which the tasks will be stored into the local storage of the browser. No TDD tests will be constructed as Jest framework doesn't have the ability to interact with the real browser environment especially local storage. However, formal testing will be documented and carried out to make sure all functions work.

Monday 20/11/2023 - Concept Program 2 Done: Todo list application has been implemented with functions that can successfully add tasks, remove tasks and clear all tasks whilst all on display. Test screenshots for each function have been documented to show that the tasks work efficiently on local storage and will be altered depending on which function has been used. This application is a simple design and can easily be modified for a better interface.

Tuesday 21/11/2023 - Start Concept Program 3: Begin to construct an application to draw shapes using HTML5 Canvas. is used to draw graphics and should be able to make all sort of different shapes. Also using Jest and TDD to test this application throughout.

Update: TDD will not be used for this specific concept program however, Test screenshots will be provided to prove that the program has the correct functionality and works properly.

Wednesday 22/11/2023 - Concept Program 3 Done: Managed to complete Shapes Drawing Application which allows you to select from a small range of shapes which can then be drawn onto the canvas wherever the mouse cursor has clicked on the canvas. Screenshots have been provided and documented to show the application has no errors but the correct functionality. This simple app can be more complex if wanted to by adding a larger range of shapes as well as different sizez or colours etc. Few minor issues whilst making this application such as constructing a triangle (especially with parameters) but no major issues were caused.

Wednesday 22/11/2023 - Start Concept Program 4: Starting the final concept program and aim to create some sort of web page that should load and list raw data from Open Street Map. It's most likely that a library will be used in order to retrieve this data.

Saturday 25/11/2023 - Concept Program 4 Done: Simple web page has been constructed to retrieve data from OpenStreetMap using Leaflet library in which this data can be displayed. This web page passes both TDD tests to make sure a map element is initalised as well as that the leaflet library loads properly. Although this code works completely fine, I may want to refactor this code and access OSM data using an API such as *Overpass API* along with Leaflet.

Wednesday 29/11/2023 - Supervisor Meeting 3: Discussed about the final concept program and interim. Although Concept Program 4 demonstrates well using leaflet library, need to refactor the program so it loads and lists raw data on the web page (may have to use xml files). Also was advised on how to conduct the interim report: All 4 reports all structure into one big report!

Friday 08/12/2023 - Concept Program 4 Updated: Concept Program 4 has been refactored to correctly serve its purpose as it lists and loads map.osm in raw.

Tuesday 23/01/2024 - Supervisor Meeting 4: As back from the holidays, ready to create the final application consisting of all concept programs created earlier in the project. Discussed that writing more information including final deliverables towards the interim will create the final report. Have two options to either implement everything myself by parsing xml and vector tiles or decide to go and use a library like leaflet and start to make it more complex with more complicated features. Features can include displaying the shortest destination between two points as well as filtering -> eg finding restaurants/cafes with specific info. Can use react but may be complex and time consuming but make sure to use some sort of database as well as test unit casing for the final application.

Tuesday 27/03/2024 - (FINAL) Supervisor Meeting 5: After developing the final application we discussed final implementations and ways of making this application efficient. The application looks good and was also discussed that the offline feature needs to be implemented ASAP. Other than that, the application will finish up by completing the extended suggestions and final CSS.

Friday 12/04/2024 - Final Offline HTML5 Maps Application developed: Offline HTML5 Maps Application has finally been developed and achieves the following:

Offline functionality which: Allows the user to load and display map data. Allows the user to zoom and move around the map.

Extended features: Allows the user to search for interesting features. Allows the dynamic display of different kinds of data (E.g. highlight cafes on screen). Downloading map data dynamically when a connection is present.