*Dylan Maryk (2015)*

*Supervisor: Gregory Chockler*

# *FINAL YEAR FULL UNIT PROJECT - INTERIM REVIEW*

Offline HTML5 Maps Application

# Contents

# Introduction and Objectives

## Introduction

I am doing the Offline HTML5 Maps Application project because I have a keen interest in software, particularly on mobile platforms, that utilises map technology to provide a user with helpful information, especially contextually relevant information based on their current location. It is worthwhile to apply this interest using web technologies instead, experimenting with the combination of HTML5 mapping and local storage, while still offering functionality traditionally only offered with online map services, such as providing directions and information about places of interest. The final project will be an offline HTML5 application that is capable of storing and displaying richly detailed maps, using both tile images and vectors, with a wealth of information and functionality a user would expect from a native application.

I believe this would be a useful project that currently faces minimal competition. For example, Google's Maps mobile application is a native application, hence only usable on certain platforms, and currently only allows downloading and viewing offline certain areas. Furthermore, it does not currently support search or directions functionality offline [1].

The project reports I wrote throughout the first term revolve around the sorts of topics I have been researching and how I have used them when developing either proof of concept programs or the final project. These will include key background concepts and technologies relevant to web development and mapping tools.

## Objectives

- Complete early deliverables
    - Proof of concept programs
        - "Hello world" offline HTML5 application
        - To-do list application using an IndexedDB
        - Webpage that loads and lists raw OpenStreetMap data
    - Reports
        - HTML5 Caching and Storage
        - OSM Data Representation
- Complete basic functionality of final deliverables
    - Program fully works offline
    - Program loads and displays map
    - Program allows zooming and panning map
    - Program supports basic search functionality
- Complete extensions
    - Advanced search functionality
    - Cover more areas
    - Basic directions between two points

# Planning and Timescale

## Term 1

I have developed three proof of concept programs during the first term, accounting for the early deliverables:

1. "Hello world" offline HTML5 application: A simple application built using HTML5 that is viewable offline.
2. To-do list application using an IndexedDB: A simple application to demonstrate the use of an IndexedDB for storing data offline.
3. Webpage that loads and lists raw OpenStreetMap data: A demonstration of retrieving and displaying OpenStreetMap data in a raw format.

| Delivery date | Development | Report | Notes |
|---|---|---|---|
| 9th October | "Hello world" offline HTML5 application | | I am familiar with using these languages, so I expect developing a basic offline application to take relatively little time. |
| 16th October | To-do list application using an IndexedDB | | I have not used IndexedDBs before and so need a week to understand it and implement it. |
| 23rd October | | HTML5 offline technologies: Basics of caching and IndexedDB | I will write a little on what I have learnt about IndexedDBs and other methods of caching data. If this is finished quickly I can move onto HTML5 canvases sooner. |
| 30th October | Webpage that loads and lists raw OSM data | | |
| 6th November | | OSM data representation: Comparing and contrasting vector and tile image maps | After playing with downloading and displaying raw OSM data, I want to spend a week researching how to transform |

| | | | |
|---|---|---|---|
| | | | this data into visual maps. |
| 13th November | Load and display map data online and offline, probably limited to a specific area at first like London | | Using what I have learnt from the early deliverables, I will mainly focus on completing basic versions of the final deliverables for the remainder of the term. I may write short reports on what I have worked on when I reach key milestones, such as completing one of the deliverables. |
| 20th November | Extra information on request | | Play with supporting basic uses of the user tapping on certain areas of the map and relevant information being displayed, possibly retrieving extra data from other sources. |
| 27th November | Polish existing functionality i.e. Improve performance, smooth-line experience of downloading data with the user having to do less | | |
| 4th December | Basic search functionality | | Allow the user to enter a search term and have the focus of the map move to the location best matching the term. Will probably have a very crude UI at first. |

I have now completed the early deliverables and also the final deliverables, at least with basic functionality and not yet fully optimised in terms of performance and user experience, as I have successfully implemented a program that works offline, loads and displays map data and allows the user to zoom and move around the map. Basic search functionality however has not been implemented yet, although I originally planned to have at least made a start on it by now.

## Term 2

I have a rough idea of some of the more advanced features I want to implement during the second term.

- Advanced search functionality, including listing various possible matches to a search term and allowing for filtering by certain types of locations.
- Vector data covering more landmass, as opposed to only a certain region.
- Basic directions from one point to another, calculated offline.

# Interim Reports

## HTML5 Caching and Storage

### Introduction

I have investigated the basics of storing information offline using modern web browser technologies, in order to develop two of the proof of concept programs, and later on the final deliverables.

HTML application caching is vital to allow the user to view the program in their web browser of choice even when they are lacking an Internet connection, and a form of data storage must be used to store the OpenStreetMap data itself while the user is online, so that it can be accessed later even when the user is offline.

### Application Caching

The HTML5 application cache provides a simple means by which to locally store the files required to display a website offline automatically, whenever the user loads the page while they are online, ensuring the cached files are updated as frequently as possible. However, it is important to note for the further development of the project that browsers can have different size limits for cached data [2], hence it is preferable to cache only the minimum necessary to display what it makes sense to be viewable offline. It must be noted that the browser's cache for a website will not be updated unless the cache manifest, the file which lists all the files on the server to cache, is updated. Therefore, changes to the online version of the website will not be cached while the user is online unless the developer updates the cache manifest. This can be done as simply as changing a comment e.g. "# v1" to "# v2".

I have written my .appcache file so that every file that is unlikely to be modified frequently is cached, which includes "index.html" and every CSS and JavaScript file, as well as the .osm file representing OpenStreetMap vector data (I will cover this in detail later). Below the list of files to cache, I have included:

*NETWORK:*
*

This ensures that for any files not cached, the app will attempt to retrieve them from the relevant remote locations, meaning the app will always try to get the remote resources before using a fallback option. In the case of getting tile images from OpenStreetMap, I have implemented some timeout functionality which will fall back to using cached data if it is available.

### Data Storage

The bulk of data storage is done using technologies such as the IndexedDB API, which is designed to store larger amounts of data locally. It has a global limit, the maximum amount of storage space that can be used by the browser for all origins (top level domains), of 50%

of free space on the device [3]. The group limit, the limit for each group of origins, is 20% of the global limit. This must be handled when considering how much map data can be downloaded onto the user's device. The API is interacted with via JavaScript.

I am caching data using an IndexedDB database whereby string representations of OpenStreetMap tile images are stored. This is the cached data that will be used should an attempt to retrieve the resource remotely times out, as mentioned above.

Conclusion

When implemented properly, ensuring the correct resources are cached, HTML5 application caching is a simple and effective method of storing key resources of the program offline. The fact it is a standard technology in HTML5 is useful in that it can be used in most modern browsers.

IndexedDB is also a standard web technology, and so is a good choice of data storage because it works in most browsers, is well documented and is capable of storing large quantities of data.

## OSM Data Representation

### Introduction

There are two formats of data the program must be able to store offline: tiles as images and additional information as vector data. In summary, images retrieved from OpenStreetMap via their API represent sections of a map at specified zoom levels, and vector data retrieved represents additional details that overlay the images to provide the user with extra information.

### Caching and Rendering Vector Data

The first step to representing OpenStreetMap data in a visual format is to actually acquire that data. The OpenStreetMap API provides an endpoint for retrieving this map data by defining a bounding box as input, consisting of the longitude of the left, latitude of the bottom, longitude of the right and latitude of the top. This returns raw map data for the specified region in the form of a .osm file [4]. For example, we can retrieve data for Greater London using the bounding box [5] "-0.221272,51.4801,-0.071754,51.533309":

*http://api06.dev.openstreetmap.org/api/0.6/map?bbox=-0.221272,51.4801,-0.071754,51.533309*

The raw data can be retrieved from this URL using JavaScript in the browser, meaning it can then be stored locally on the user's device, so that the same data does not need to be requested every time the page is loaded. However, because this data may be uploaded frequently, and I believe it best to rely on other organisation's servers as little as possible, due to possible downtime for example, I have decided to write a bash script that can be made to run periodically. This bash script can frequently download files required by the project, such as not only .osm files but also jQuery JavaScript files. The user's browser can then download and cache these files directly from my own server. This is the line in the bash script responsible for downloading OpenStreetMap data into a .osm file:

*wget -O london.osm "http://api06.dev.openstreetmap.org/api/0.6/map?bbox=-0.221272,51.4801,-0.071754,51.533309"*

In terms of rendering raw OpenStreetMap data, I wanted to find a method for parsing and displaying it as manually as possible i.e. finding a balance between producing my own code and making use of existing libraries. A starting point was to convert the OpenStreetMap data from an XML format into a GeoJSON format. This would be useful as many JavaScript mapping libraries work with GeoJSON formatted data. I have chosen to do this using the osmtogeojson open-source library [6]. Here is how I parse and convert the XML data from the .osm file using jQuery:

```
$.get("london.osm", function(data) {
        var dataParsed = $.parseXML(data);
        var dataGeoJson = osmtogeojson(dataParsed);
        L.geoJson(dataGeoJson).addTo(map);
```

*});*

I opted to experiment with using the open-source JavaScript library Leaflet [7], as it is designed with being mobile-friendly in mind, is lightweight (beneficial in terms of being quick to cache and taking up minimal storage space on the user's device) and because it is open-source can be more easily modified. There are therefore also many other open-source projects that build upon Leaflet already in existence.

The above accounts for the basics of how to get and display useful information for the user without an Internet connection, the .osm, .js etc. files required being cached when the user first visits the website while they are online. However, it is also vital to retrieve and cache map tiles from OpenStreetMap, these tiles representing the map itself underneath the additional information. This project therefore combines image tiles and vectors: images tiles for the essential map itself and vectors for the additional details that, later on in development, will allow the user to view and search for specific information.

Displaying and Storing Tile Images

Leaflet handles the basics of loading and displaying tiles. It allows for initialising a tile layer with a URL template that is used to generate the URL for each tile to display, along with a number of optional parameters [8].

*var osmUrl = "http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png";*
*var osmLayer = new CustomTileLayer(osmUrl);*

I am using my own extended version of the tile layer class, "CustomTileLayer", which I will go into detail about later. The URL template has four parameters: "{s}" for one of the available subdomains to aid possible parallel requests, "{z}" for the map's current zoom level, "{x}" and "{y}" for the coordinates of the tile for which an image is being requested. Leaflet handles adding these parameters and adding the resulting URL as the source of the relevant HTML object that represents the tile. This covers displaying the correct map tiles on request, but the next step is to add functionality on top of Leaflet to handle caching these images to an IndexedDB database, so they can be loaded instead if the user is offline.

*var request = window.IndexedDB.open("TileStorage", 3);*

This line opens the "TileStorage" database so it can be interacted with. There are three event handlers for this that I make use of [9]. When "onsuccess" is called, the database exists, so the database is initialised.

*db = event.target.result;*

When "onerror" is called, it is most likely the user did not give permission for the browser to use IndexedDB, so an error is printed in the console.

*console.log("Did not allow local data storage.");*

When “onupgradeneeded” is called, the database does not yet contain an object store, so a new one is created.

```
db = event.target.result;
db.createObjectStore("tile");
console.log("Created new object store.");
```

This handles the structuring of the database, ready for adding data to and retrieving data from the “tile” object store.

Probably the most important part of the process of caching images however is done when certain events are fired by the tile layer instance. I respond to three events. The first is “tileloadstart”, fired before an image source is added to a tile.

```
event.tile.crossOrigin = "Anonymous";
```

This line sets the “crossOrigin” parameter of the tile to “Anonymous”, which is necessary to allow the image, once it has been downloaded, to be retrieved from a canvas once that canvas has been used to transform the image into a Base64 formatted string. Otherwise, when attempting to retrieve the formatted string from the canvas, an error will be raised saying that the canvas has been “tainted”. This is a security measure in place for when, as in this case, images defined by an “img” element are loaded from a foreign source, which “protects users from having private data exposed by using images to pull information from remote web sites without permission” [10].

The second event responded to is “tileload”, fired when an image has been successfully loaded.

```
var tileImageString = getBase64Image(event.tile);
var tileImagePoint = event.tile.point;

if (db) {
        storeTileImage(tileImageString, tileImagePoint);
} else {
        tilesToStore.push({image: tileImageString, point: tileImagePoint});
}
```

If the database has been initialised, the tile’s image, in Base64 format, and the tile’s position and zoom level are added to the “tile” object store. Otherwise, they are pushed to an array. Once the database has been initialised, the images and positions stored in this array will be added to the object store.

Finally, the “tileerror” event is responded to, fired when the image loading process times out.

```
if (db) {
        var tile = event.tile;
```

```
	var tileImagePointString = getPointString(tile.point);
	var request = getObjectStore().get(tileImagePointString);
	request.onsuccess = function(event) {
		var tileImageString = request.result;

		if (tileImageString) {
			tile.src = tileImageString;
		} else {
			console.log("No tile image stored for point " + tileImagePointString +
".");
		}
	};
}
```

Tile images and positions are stored in the object store using a key-value relationship: the position and zoom level is the unique key and the image as a Base64 string is the value. If the database has been initialised, the value for the tile's position is attempted to be retrieved from the database. If successful, this value is set as the tile's image source.

Now that these images and positions are being stored, the program must determine when to load them from the local database instead of retrieving the relevant images remotely. To achieve this, I have opted to implement timeout functionality. To do this, I have overridden the "_loadTile" function of Leaflet's "TileLayer" class in order to give each tile a number of seconds after which a function is called that checks if the image was successfully loaded.

```
tilesToCheck.push(t);
setTimeout(checkImageLoaded, 5000, this);
```

If the program has failed to get the tile's image from OpenStreetMap's API, the "checkImageLoaded" function fires the "tileerror" event.

```
function checkImageLoaded(layer) {
	var tile = tilesToCheck[tilesToCheckCounter];

	if (!tile.complete) {
		layer.fire("tileerror", {
			tile: tile
		});
	}

	tilesToCheckCounter++;
}
```

Conclusion

The combination of the Leaflet library, the IndexedDB API and some additional custom implementations allows for a functional program suited for the purpose of downloading,

storing, loading and displaying map data from the OpenStreetMap API. While I have so far implemented the core functionality of the program, there remain inefficiencies. For example, when the user is online, even tile images that have already been downloaded and stored will be re-stored in the local database.

## Diary

1:36 pm on September 29, 2015

I have completed the majority of my project plan. I have to finish off the timeline and risk assessment and add a bibliography. I am meeting my project supervisor at 6pm today.

5:47 pm on September 30, 2015

My project supervisor yesterday advised me on how I should go about completing my plan. Today I worked some more on the timeline and risk assessment.

10:59 am on October 4, 2015

On Friday I submitted my final plan.

4:05 pm on October 10, 2015

I have set up a private Git repository in which to keep all my work, including code and reports. I have so far committed my project plan.

4:46 pm on October 10, 2015

I have set up my development environment. I am using Sublime Text 3 to write my HTML5 deliverables, and using the SFTP plugin for it to map the files on my local machine to my remote web server. This means that I can edit files locally and they will automatically be uploaded to the server and available online via a web address. This speeds up testing. I am using SourceTree to manage my Git repository. Commits are pushed to a private repository on Bitbucket.

5:44 pm on October 10, 2015

I have created a basic Hello World offline application using application cache and started a report on the basics of HTML5 application caching.

8:44 pm on October 14, 2015

I have started on a simple to-do list application to get to understand the indexedDB API used to store large amounts of data offline. I am simultaneously gradually working on my first report.

4:06 pm on October 18, 2015

Functionality for adding tasks to the to-do list is complete, now to add the ability to mark tasks as complete.

10:48 pm on October 18, 2015

Started adding functionality for marking tasks as complete. Need to fix a bug where when tasks are marked as complete and hence should be updated, new tasks are added instead.

11:50 pm on October 25, 2015

Successfully fixed bug. Marking a task as complete now updates the existing entry. The to-do list application early deliverable has been completed.

11:54 pm on October 25, 2015

Update to previous update: It turned out that using auto-incrementing IDs would not allow existing records to be updated. As a workaround, I used the current UNIX timestamp as a unique ID for each new record.

4:11 pm on October 27, 2015

Started on the early deliverable of displaying raw OpenStreetMap data in a browser. Successfully getting raw map data for central London to display in the console. Also started writing a new report: OSM Data Representation.

4:49 pm on October 27, 2015

The raw map data is now being successfully displayed in the browser.

3:39 pm on November 4, 2015

Started on the main project. Started on the core structure of a project for visualising a map from raw data.

6:32 pm on November 7, 2015

I have made use of the open-source "osmtogeojson" library to convert OpenStreetMap data into GeoJSON formatted data, as this is what many JavaScript-based mapping libraries use to visualise data.

6:49 pm on November 8, 2015

As far as I understand at the moment, I was mistaken in thinking the raw data from OpenStreetMap represents map data itself. Instead, having mapped the data using http://www.geojson.io, after converting it to GeoJSON format, it appears only to represent significant points on a map and information about those points, but these points still require a tiled map to be placed on top of. Hence, I will look into how to get map tiles from OpenStreetMap separately.

8:53 pm on November 8, 2015

I created a map using the open-source Leaflet JavaScript library. I am now downloading map tiles from OpenStreetMap as and when they need to be displayed to the user. The additional information from the .osm file, which is cached, is now displayed over the tiles after being converted to GeoJSON format. Displaying this additional information currently works offline, displaying map tiles does not yet. Currently, the additional information is only represented as points on the map. The user cannot yet click on these points to see further details. This will be implemented later, as caching map tiles is higher priority.

11:47 pm on November 8, 2015

I have started experimenting with locally storing tile images using IndexedDB. I am currently getting the element holding each tile image and source URL of each tile image by handling the "tileload" event, fired by Leaflet's TileLayer class, each time a tile has finished loading.

4:22 pm on November 15, 2015

By overriding the "_loadTile" function of Leaflet's TileLayer class, I have rewritten the function so that setting the image source of the tile is done only after the "tileloadstart" event is fired, so that when the event is fired I can set the tile's "crossOrigin" parameter to "Anonymous". This allows me to get the image in a data URL format after it has been drawn on a canvas. For security reasons, unless this parameter is set accordingly, the canvas is considered tainted and data cannot be retrieved from it. Now therefore, the tile's image can be converted to a Base64 string and stored in an IndexedDB database. After I ensure these strings correctly represent the images, I can work on loading these images from the database and displaying them in the case that the user is offline.

11:32 pm on November 15, 2015

I am now storing image tiles using the positions of these titles on the map as keys: "x" and "y" coordinates and a "z" value representing zoom level. This will now allow for getting the correct images from the database when offline based on the positions currently visible on the map.

2:45 pm on November 18, 2015

Yesterday I started implementing loading tile images from the database when the "tileerror" event is fired, as this means Leaflet has failed to download the image from OpenStreetMap.

7:18 pm on November 19, 2015

I have implemented getting image tiles from the database, using the currently observable sections of the map and the zoom level as keys. E.g. For a position of "1021, 680" and a zoom level of "11", the key will be "1021,680,11". Currently, the website only caches image tiles for exactly which positions and zoom levels loaded for the user when their device was online. I may therefore next look into downloading more data while the user is online, such as tiles for nearby positions or additional zoom levels.

3:58 pm on November 21, 2015

I have ensured the app attempts to get resources stored online, even if some resources should always be retrieved from the cache. Before, if application caching was being used, the app would not attempt to get OpenStreetMap tile images.

4:27 pm on November 21, 2015

I made the map fill the browser, with help fromhttp://stackoverflow.com/a/21652935 on the CSS required.

4:54 pm on November 21, 2015

I added a timestamp comment to the application cache, as whenever I update this, files in the cache that have been updated will be re-cached by the browser.

9:46 pm on November 21, 2015

I added a timeout for loading tiles from OpenStreetMap. If after 5 seconds a tile has not been retrieved from OpenStreetMap, it will be attempted to load the tile from the database. This ensures the user will receive up-to-date map content if they are online, but if they are offline they will only wait a few seconds before the cached content is displayed instead.

9:46 pm on November 22, 2015

Removed pointless removing and re-adding of the Base64 string prefix. Added significantly to "OSM Data Representation" report.

11:50 pm on December 1, 2015

I fixed the map's tiles not filling the full height of the map vertically. This issue was caused by "offlinemaps.css" not being run before the JavaScript files, as this CSS file sets the map to fill the height and width of the browser window. Originally, the tiles were being loaded before the map's size was set.

# Bibliography


[1] Maps for mobile: Download a map and use it offline
https://support.google.com/gmm/answer/3273567?hl=en-GB

[2] W3Schools: HTML5 Application Cache
http://www.w3schools.com/html/html5_app_cache.asp

[3] Mozilla IndexedDB documentation: Browser storage limits and eviction criteria
https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Browser_storage_limits_and_eviction_criteria#Storage_limits

[4] OpenStreetMap Wiki: Retrieving map data by bounding box
http://wiki.openstreetmap.org/wiki/API_v0.6#Retrieving_map_data_by_bounding_box:_GET_.2Fapi.2F0.6.2Fmap

[5] OpenStreetMap Wiki: Bounding Box
http://wiki.openstreetmap.org/wiki/Bounding_Box

[6] osmtogeojson: Convert OSM to GeoJSON
https://github.com/tyrasd/osmtogeojson

[7] Leaflet: Open-source JavaScript library for mobile-friendly interactive maps
http://leafletjs.com/

[8] Leaflet Documentation: TileLayer
http://leafletjs.com/reference.html#tilelayer

[9] Mozilla IndexedDB documentation: IDBOpenDBRequest
https://developer.mozilla.org/en-US/docs/Web/API/IDBOpenDBRequest

[10] Mozilla HTML documentation: CORS enabled image
https://developer.mozilla.org/en-US/docs/Web/HTML/CORS_enabled_image