SEM2220 Assignment One
Report

Mobile Web - Creating the Conference Sessions List Dynamically

Submitted in partial fulfillment of the requirements for the award of the degree of

MEng
in
Software Engineering

Submitted by

crh13 Craig Heptinstall

Department of Computer Science
Aberystwyth University

29th February 2016

Chapter 1

1.1 Introduction

This report outlines findings and issues into developing an already existing mobile web application. Features and fixes have been performed on this application, with explanations of how each one was completed, with issues stated where necessary. The app, which provides basic information about a conference, is required to show a dynamic list of events, alongside a map showing the location of the conference area. While a map is already shown, the list of sessions was not functional when the template was provided. I have also chosen to add the feature of showing where sessions are located on the map as part of the flair marks.

1.2 Setting up the application

Upon first receiving the application template, I wanted to ensure that the project was capable of running as a standalone web application and as an android app. Running the application via a browser was completed simply using a local server (in my case, EasyPHP). After installing the local server, I then had the capability to run any server side or database code if required in this assignment. Without the use of a local server, I found that loading up the web files in the browser locally results in the spinning load icon would remain stuck due to Ajax requirements for the PhoneGap API.

For compiling the application as an android app meant using the PhoneGap service. Currently, there are two options for building via their service: via command line locally, and via their online UI. Because I had already experience in using the PhoneGap online UI, and its support for directly linking up with GitHub repositories, I chose this as the most efficient route. This meant on each commit of new features to the application, I was able to force a new build ready to be downloaded and run on a device.

When it came to performing any debugging of the application, I selected to use Chrome developer tools, because of past experience, and the good amount of debugging features available in the software. Further into this report the debugging under use will be shown, with the use of breakpoints, console logging, and screen options being used.

Before explaining how features were added and fixed, there are two components of the app code that should be explained:

- DataContext- This file holds all methods to do with gathering, inserting and editing data from the Web SQL database. It is called from Controller methods requesting data for use in the front end of the app.
- Controller- Controls the use of the front end by users, and gets any data via DataContext methods to be parsed and presented back to the user.

1.3 Dynamically loading sessions

The first task for the assignment was to get the sessions (stored in a WebSQL database) to load dynamically into the session list in the app. To do this, all that was required was a means of executing an SQL statement and return the results. The 'querySessions' method found in the DataContext file already had a provided statement to do this, whilst all I needed to do was execute it. After reading various tutorials on how to perform this, I was then able to get the relevant session objects passed back to the Controller. When it came to handling errors, the methods already present were useful, though simply needed making more specific for the session query.

Once the sessions were being returned to the Controller, it was then a case of producing the HTML to represent the session list. In the method 'renderSessionsList' I then went about looping over each session object, and creating the appropriate list tags. In order to know what HTML would be required to produce the required presentation layer, I ensured that the HMTL to be appended in the loop was reflecting previous examples of the app where the sessions had been hardcoded in.

Although this assignment asked the sessions to be appended to the HTML list, I instead used the Jquery .html() method to replace the entire section rather than add to it. The reason for this was caused when I noticed if a user changed tab, and then went back to the session page, the list became longer. This was due to the database being queried again, and then any sessions would be added to the already existing list. The method I used ensured the HTML was entirely replaced.

On top of creating the HTML dynamically for the list, I also had to add the code to represent the search functionality (See fig 1.1). Because the section content was being replaced, I decided that the search box would have to be generated on each refresh of the session list.

In order to replace the entire HTML, I decided to build up the content in a variable, and then add it after all sessions were added. This allowed me to easier debug individual elements and check the correct data was being placed in the presentation. Alongside this, I had to ensure that the various CSS classes, data-role tags and data-type tags were correctly placed on the generated HTML to ensure the appearance and functionality of the elements were correct.

Because I was refreshing the content dynamically, and JQuery mobile did not know of the changes directly to be able to allow list elements be found using the search box, I had to learn to call a method on the session list. The '.listview' method was used on the list HTML element, and ensured that the list was indexed ready for use by searching as shown in fig 1.2.

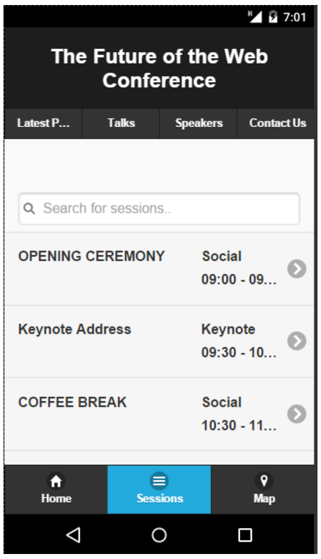


Figure 1.1: The session list dynamically pulled in, alongside the search bar

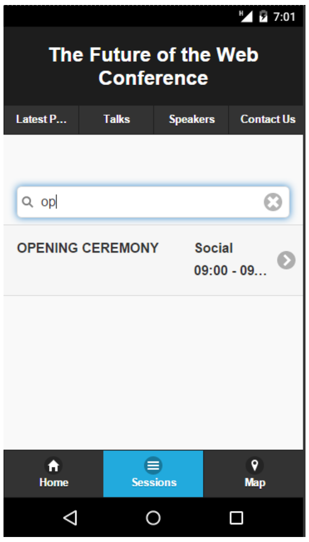


Figure 1.2: Results from the functioning search box using the JQuery mobile library

1.4 Adding session location map markers

As an extra task, and to implement some of the flair ideas in the assignment, I then moved onto changing the static map on the map page of the app to a more dynamic map which would incorporate markers for the locations of different sessions. Rather than the image that is created based on the location of the user, I decided to use the Google maps API to generate a controllable map.

Performing the first part of this task required me to incorporate the Google maps script into the app, use the required code to generate a map object, then attaching the map to a div (in this case, the same div used to hold the static image). See figures 1.3 and 1.4. By passing in the current location already gathered in the code, the map was easy to centre in on the location of the user. The API also allowed me to set more options such as map type, and zoom level.

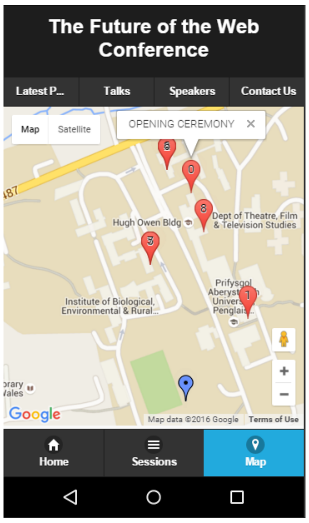

The Future of the Web Conference
Latest P...
Talks
Speakers
Contact Us
Map
Satellite
OPENING CEREMONY
Hugh Owen Bldg
Dept of Theatre, Film & Television Studies
Institute of Biological, Environmental & Rural...
Prifysgol Aberyst... Univers... Penglais
Google
Map data ©2016 Google
Terms of Use
Home
Sessions
Map

Figure 1.3: Map showing locations of sessions with markers. One marker has been clicked to show the event taking place there

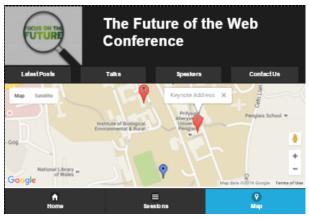


Figure 1.4: Landscape view of map

In order to add markers to the map correlating to the session locations, I had to perform a few steps:

1. Created a new table using similar code to what was already present in the DataContext, but for holding co-ordinates. A new column was added to the session table to reference the location rows. It was performed this way, as I was assuming certain sessions could be held in the same place therefore this avoided repeated data.

2. Created a method to clear and upgrade the Web SQL database assuming a user had already used the app before. This meant dropping tables and creating all those required again.

3. A join statement gathered all the sessions, and location coordinates for each session to add to each session object. This meant that when each session was returned to the Controller, that the map could read the longitude and latitude values directly.

4. Markers were generated, containing descriptions in the labels on click, and a number in the marker to identify it with each event.

1.5 Further bugs and issues

Alongside the additions to the code base, I also had to perform a few fixes to the functionality, and to the way the app was run. The first bug I found whilst testing my new features was that if I was to click on a tab at the bottom, and then press the current tab again, an index undefined error would occur and prevent the tabs working again. To fix this, I simply added a 'href' attribute to the current tab, in order for JQuery Mobile to be able to attempt to open something rather than null.

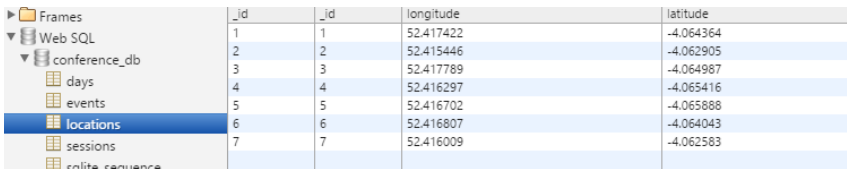
The next bug I went to fix was the map size. With that static map, I noticed that it had a set width, while the height stretched to the size of the screen. Instead, I used the code generated page sizes to ensure the map would fit the entire content area. I also chose to keep the navigation tabs always open rather than hiding, because if a user was to navigate the dynamic map, the tabs would keep hiding and showing on every click and this would become annoying.

Debugging the application was an issue I came across when using a USB cable, though this was due to multiple problems which I had to solve through online tutorials. By installing the latest Chrome Canary, and the latest drivers for my phone connection, I was able to progress with debugging.

1.6 Testing and running the app

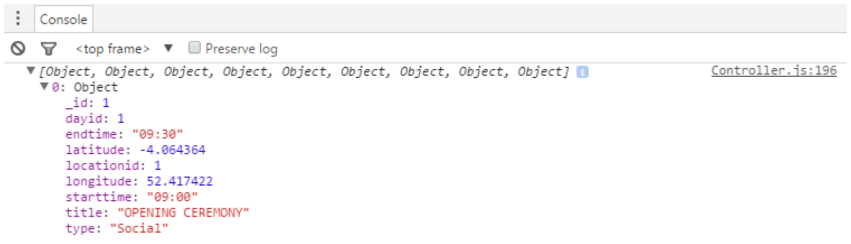
During the addition of features and fixes, I have ensured to debug and test the code in as many ways as possible. When I was initially developing, I used my local server to run the app on my local Chrome browser. The browser tools gave me plenty of options to check the WebSQL database (see fig 1.5) was holding correct data, and that the appearance of the app at a different range of window sizes worked well. Debugging and checking the console for errors (see figures 1.6 and 1.7) was also a major component of testing.

When I had done incremental changes I then tested the APK file from PhoneGap. To test quickly I decided to use an emulator (GenyMotion - shown in figure 1.8), and for bigger pushes I then went and installed the APK on my on phone to debug where necessary.



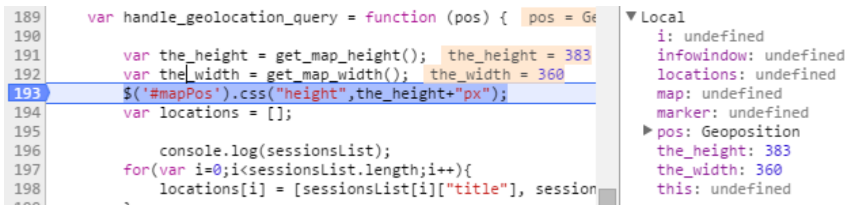
_id	_id	longitude	latitude
1	1	52.417422	-4.064364
2	2	52.415446	-4.062905
3	3	52.417789	-4.064987
4	4	52.416297	-4.065416
5	5	52.416702	-4.065888
6	6	52.416807	-4.064043
7	7	52.416009	-4.062583

Figure 1.5: The WebSQL table for the locations shown in the developer tools, useful for checking data on the presentation side and in the select statements



```
▼[Object, Object, Object, Object, Object, Object, Object, Object, Object]      Controller.js:196
  ▼0: Object
     _id: 1
     dayid: 1
     endtime: "09:30"
     latitude: -4.064364
     locationid: 1
     longitude: 52.417422
     starttime: "09:00"
     title: "OPENING CEREMONY"
     type: "Social"
```

Figure 1.6: Chrome developer tool showing console log of an array of session objects



```
var handle_geolocation_query = function (pos) {  pos = Ge

    var the_height = get_map_height();  the_height = 383
    var the_width = get_map_width();  the_width = 360
    $('#mapPos').css("height",the_height+"px");
    var locations = [];

        console.log(sessionsList);
    for(var i=0;i<sessionsList.length;i++){
        locations[i] = [sessionsList[i]["title"], session
```

```
▼ Local
    i: undefined
    infowindow: undefined
    locations: undefined
    map: undefined
    marker: undefined
  ▶ pos: Geoposition
    the_height: 383
    the_width: 360
    this: undefined
```

Figure 1.7: Use of the debug tool for checking the height and width of the screen is being received correctly

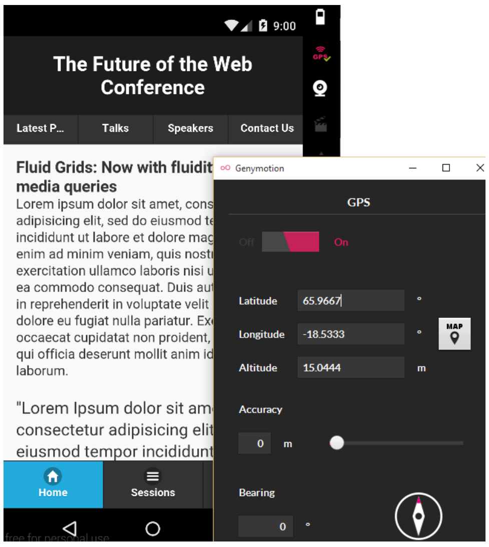


Figure 1.8: The GenyMotion emulator running the application, wih the window allowing the location to be manually set for testing in the app

1.7 Conclusion

Before this project I had no experience with WebSQL, especially not knowing that SQL could be called via client side code. In my opinion, this is something I would always keep server side, but in terms of a lightweight app it is useful. I have also learnt around the key JQuery Mobile functions, and how navigation works. Ensuring the app worked and looked the same in this app was a top priority throughout this task.

Following my work performed, I can now provide personal scores and reasons for the amounts:

Topic	**Score**	**Reason**
Documentation	14/20	Good description of what and how features were added, and good display of issues and fixes. Although report was above the 3 page limit, this was only over because of image content.
Implementation	35/50	Well commented code, with appropriate uses of the technologies on offer in order to create and edit various application features.
Flair	15/20	Addition of dynamic map and markers shows use of external API and more use of the WebSQL framework.
Testing	8/10	Shown use of the Chrome developer tool, use of an emulator, and use of a device to test code throughout the application.

Overall mark: 72%