Grange Mobile App

Social Media Applications

Final Assignment

Table of Contents

1.Introduction

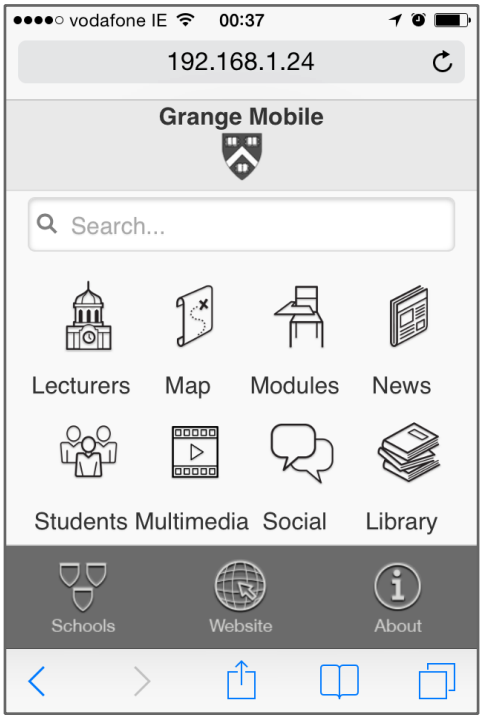
This report, prepared by Vanya Mutafchieva, is part of the final assignment for the Social Media Applications Module and aims to document the development process of the artefact part of the assignment (the Grange Mobile App).

The Grange Mobile App itself is a mobile web application for a single user (student) of the fictitious Grange Institute of Technology. It's built using the MAMP webserver solution package (Mac OS, Apache, MySQL, PHP) and is programmed using a combination of HTML5 and jQuery Mobile, JavaScript, jQuery, SQL and PHP, and using JSON for exchanging information between browser and server.

2.Requirements

2.1 Concept of Operations

All major functionality of the app is to be accessed through the home screen where 11 tiled icons are representative of the 11 main actions a user can take when using the app:



1. **Lecturers** - access a list of current lecturers for the semester's modules and view contact details for each lecturer

2. **Map** - view the user's current location on a Google map, as well as the locations of the four schools of the institute

3. **Modules** - access a list of the user's current modules for the semester and view detailed information for each module

4. **News** - access a list of the Institute's RSS news feed, view each post in a separate screen and follow a link to the original article online

5. **Students** - access a list of other students in the user's class and view detailed information for each student

6. **Multimedia** - access a thumbnail list of recent videos released by the Institute and view each video on a separate page

7. **Social** - access a list of the institute's 20 most recent tweets

8. **Library** - access the Library screen containing opening hours information for the four libraries, as well as a search box for library books availability, which leads to a detailed screen for in-depth information such as number of copies available of a particular book and the user's current distance to each library

9. **Schools** - access a list of the Institute's four colleges, for each college this links to another list of the college's schools, which then links to that school's website

10. **Website** - links to the Institute's website

11. **About** - view 'About' information about the Institute

2.1 User Scenario

[Report Requirement No1 - user scenario for a single user]

This user scenario demonstrates some of the more interesting user needs which the app's functionality addresses:

Gonzo is a student in GIT (Grange Institute of Technology). He has some time to kill during a gap between lectures. He decides to read the GIT news. He launches the Grange Mobile App and is presented with the home screen. The home screen shows a tiled view of icons for different actions. Gonzo taps on the 'News' icon and is taken to the 'News' screen, which is a list view of GIT's RSS feed. Gonzo taps on an article about technology. This bring him to a 'Story' screen where he is presented with a summary article and a 'Read full story in browser' button which links to the original article online. After quickly glancing through the text Gonzo taps on the 'Read full story in browser' button and is taken to the original article online.
(Use case 1 - View news feed/ Read individual news articles)

While reading the article about web technology Gonzo is reminded about his Dynamic Web Development assignment. He wants to ask his lecturer Charlie Cullen a question, he decides to send him a quick email. He resumes the Grange Mobile App and when presented with the 'Story' screen, which was the last open screen from his previous session, he taps the top left 'home' button on the breadcrumb navigation menu and is taken to the home screen. Gonzo now taps on the 'Lecturers' icon and is taken to the 'Lecturers' screen which lists all of Gonzo's lecturers for the semester. Gonzo taps on the 'Charlie Cullen' item from the list view and is taken to a 'Lecturer' screen with detailed contact information about Charlie. Gonzo locates the email section and taps on it. This opens a 'New Message' email screen. The 'To:' and 'From:' fields are already pre-populated with Charlie's and Gonzo's email addresses respectively, and so Gonzo enters a subject title in the 'Subject' field, types his question underneath in the email body field and presses the 'Send' button.
(Use case 2 - View lecturer's contact details/ Send email)

[Extra Credit Requirement No1 - Second Context]

Later that day on his commute home Gonzo receives an email back from Charlie suggesting a solution to Gonzo's issue. Charlie recommends a book for Gonzo to read for more in-depth explanation. Gonzo now wants to check if the book is available for loan from the college library. He opens the Grange Mobile App again and in the home screen taps on the 'Library' icon. This takes him to the 'Library' screen where in the search box Gonzo types the title of the book 'Clean Code' to check if it's available. As he types two hints appear underneath the search box in a drop down menu. Gonzo chooses 'Clean Code' and is taken to the 'Title' screen displaying details about the book. It looks like the book is available in all four libraries. Gonzo notices that the app has also calculated for him his current distance from each library - the closest to him is at Aungier Street.
(Use case 3 - Check library books' availability and current distance to library)

Gonzo now wants to check exactly where the Aungier street library is on the map as he has never been in that building before. He taps on the top left 'home' button on the breadcrumb navigation menu, and this brings him back to the home screen. Gonzo now taps on the 'Map' icon and this opens the 'Map' screen. On the map Gonzo can see four markers indicating the locations of each of the four school's libraries, as well as his current location. Gonzo makes up his mind about which library to go to.
(Use case 4 - View schools locations on the map/ View user's current location)

3. Design

3.1 Use Cases

All the pages of the app including all data from the server will be loaded at the app's launch, with the exception of the Map page. However, in order to demonstrate the logical sequence of actions taken by the user and the software for each use case, in this section I will list all the queries to the server and 3rd party data APIs as if they are triggered at the time of the action. These will be enclosed in brackets for clarity.

[Report Requirement No2 - at least 2 use cases for the user]

3.1.1 Use case 1 - View news feed/ Read individual news articles

Action No.	User	Software
1	launch app	
2		load all pages of the app (except Google map)
3		display 'Home' screen
4	tap on the 'News' icon	
5		(fetch rss data from news link and generate list of 'News')
6		display 'News' screen
7	tap on the 'Innovative State: How New Technologies Can Transform Government' item on the 'News' list	
8		(generate data for the 'Innovative State: How New Technologies Can Transform Government' story screen)
9		display 'Story' screen for the 'Innovative State: How New Technologies Can Transform Government' article
10	tap on the 'Read full story in browser' button	
11		load linked web page

3.1.2.Use case 2 - View lecturer's contact details/ Send email

Action No.	User	Software
1	resume app	
2		display ‘Story' screen for the ‘Innovative State: How New Technologies Can Transform Government’ article
3	tap on the 'home' icon on the breadcrumb navigation menu	
4		display 'Home' screen
5	tap on the 'Lecturers' icon	
6		(check if ‘Lecturers’ data is stored in local storage and retrieve data if stored locally and if recent)
7		(query server and fetch ‘Lecturers’ data if no data is stored in local storage or if data is old)
8		display list of lecturers on ‘Lecturers’ screen
9	tap on the ‘Charlie Cullen' item on the ‘Lecturers’ list	
10		(generate ‘Lecturer’ screen for Charlie Cullen)
11		display ‘Lecturer’ screen for Charlie Cullen
12	tap on the ‘Email’ section	
13		launch the email phone app
14		email app populates the ‘To’ and ‘From’ fields with Charlie’s and Gonzo’s email addresses
15		email app displays ‘New Message’ email screen
16	type in a title in the ‘Subject’ field	
17	type in text in the ‘Body’ field	
18	press the ‘Send’ button	
19		email app sends the email

3.1.3 Use Case 3 - Check library books' availability/ current distance to library

Action No.	User	Software
1	launch app	
2		display 'Home' screen
3	tap on the 'Library' icon	
4		(check if 'Library' data is stored in local storage and retrieve data if stored locally and if recent)
5		(query server and fetch 'Library' data if no data is stored in local storage or if data is old)
6		retrieve locations for the colleges from the server or the local storage (if previously stored)
7		calculate the distance to each college from current position
8		generate a list of book titles
9		display 'Library' screen
10	type 'Clean Code' in search box	
11		display matching titles in a dropdown menu underneath search box
12	tap on the 'Clean Code' item from the search results dropdown list	
13		generate the 'Title' screen for the 'Clean Code' book including availability and current distance to each library
14		display details for the 'Title' screen for the 'Clean Code' book including availability and current distance to each library

3.1.4 Use case 4 - View schools locations on map / View user's current location

Action No.	User	Software
1	tap on the 'home' icon on the breadcrumb navigation menu	
2		display 'Home' screen
3	tap on the 'Map' icon	
4		generate Google map
5		display Google map
6		fetch colleges location data from local storage or the server (if data not stored in local storage)
7		create markers for each college on the map, add info window for each marker
8		display markers for the colleges on the map
9		calculate current location of user
10		create a marker for user’s current location on the map, add info window to marker
11		display marker for user’s current location on the map

3.2 Simple Data Model

[Report Requirement No3 - a simple data model for the user]

3.2.1. Data Model for Use Case 1

Action No.	User	Software	Input/ Process/ Output	Data
1	launch app			
2		load all pages of the app (except Google map)	process	
3		display 'Home' screen	output	HTML + jQuery Mobile
4	tap on the 'News' icon		input	
5		(fetch rss data from news link and generate list of 'News')	process	feed feed.items item item.title
6		display 'News' screen	output	HTML + jQuery Mobile
7	tap on the 'Innovative State: How New Technologies Can Transform Government' item on the 'News' list		input	
8		(generate data for the 'Innovative State: How New Technologies Can Transform Government' story screen)	process	feed feed.items item item.title item.description item.link
9		display 'Story' screen for the 'Innovative State: How New Technologies Can Transform Government' article	output	HTML + jQuery Mobile
10	tap on the 'Read full story in browser' button		input	
11		load linked web page	process	item.link

3.2.2. Data Model for Use Case 2

Action No.	User	Software	Input/ Process/ Output	Data
1	resume app			
2		display ‘Story' screen for the ‘Innovative State: How New Technologies Can Transform Government’ article	output	HTML + jQuery Mobile
3	tap on the 'home' icon on the breadcrumb navigation menu		input	
4		display 'Home' screen	output	HTML + jQuery Mobile
5	tap on the 'Lecturers' icon		input	
6		(check if ‘Lecturers’ data is stored in local storage and retrieve data if stored locally and if recent)	process	localStorage localStorageName JSONString JSONObject
7		(query server and fetch ‘Lecturers’ data if no data is stored in local storage or if data is old)	process	data dataIsRecent currentTimestamp objectToStore localStorage lecturers lecturer lecturer.firstName lecturer.lastName
8		display list of lecturers on ‘Lecturers’ screen	output	HTML + jQuery Mobile
9	tap on the ‘Charlie Cullen' item on the ‘Lecturers’ list		input	
10		(generate ‘Lecturer’ screen for Charlie Cullen)	process	lecturer lecturer.firstName lecturer.lastName lecturer.email lecturer.staffNumber lecturer.phoneNumber
11		display ‘Lecturer’ screen for Charlie Cullen	output	HTML + jQuery Mobile
12	tap on the ‘Email’ section		input	

Action No.	User	Software	Input/ Process/ Output	Data
13		launch the email phone app	process (by another app)	n/a (data by another app)
14		email app populates the ‘To’ and ‘From’ fields with Charlie’s and Gonzo’s email addresses	process (by another app)	n/a (data by another app)
15		email app displays ‘New Message’ email screen	process (by another app)	n/a (data by another app)
16	type in a title in the ‘Subject’ field		process (by another app)	n/a (data by another app)
17	type in text in the ‘Body’ field		process (by another app)	n/a (data by another app)
18	press the ‘Send’ button		process (by another app)	n/a (data by another app)
19		email app sends the email	process (by another app)	n/a (data by another app)

3.2.3. Data Model for Use Case 3

Action No.	User	Software	Input/ Process/ Output	Data
1	launch app			
2		display 'Home' screen	output	HTML + jQuery Mobile
3	tap on the 'Library' icon		input	
4		(check if ‘Library’ data is stored in local storage and retrieve data if stored locally and if recent)	process	localStorage localStorageName JSONString JSONObject
5		(query server and fetch ‘Library’ data if no data is stored in local storage or if data is old)	process	data dataIsRecent currentTimestamp objectToStore localStorage books book book.catNo book.title book.year book.author
6		retrieve locations for the colleges from the server or the local storage (if previously stored)	process	data dataIsRecent currentTimestamp objectToStore localStorage collegeLatLng collegesLocation
7		calculate the distance to each college from current position	process	position.latitude position.longitude collegesLocation distanceToCollege
8		generate a list of book titles	process	books book book.catNo book.title book.year book.author
9		display ‘Library’ screen	output	HTML + jQuery Mobile
10	type ‘Clean Code’ in search box		input	

Action No.	User	Software	Input/ Process/ Output	Data
11		display matching titles in a dropdown menu underneath search box	output	HTML + jQuery Mobile
12	tap on the ‘Clean Code’ item from the search results dropdown list		input	
13		generate details for the ‘Title’ screen for the ‘Clean Code’ book including availability and current distance to each library	process	books book book.catNo book.title book.year book.author book.availability.KevinStr book.availability.AungieStr book.availability.BoltonStr book.availability.CathalBrughaStr collegesLocation distanceToCollege
14		display the ‘Title’ screen for the ‘Clean Code’ book including availability and current distance to each library	output	HTML + jQuery Mobile

3.2.4. Data Model for Use Case 4

Action No.	User	Software	Input/ Process/ Output	Data
1	tap on the 'home' icon on the breadcrumb navigation menu		input	
2		display 'Home' screen	output	HTML + jQuery Mobile
3	tap on the 'Map' icon		input	
4		generate Google map	process	dublinLatLng myOptions map data
5		display Google map	output	HTML + jQuery Mobile
6		fetch colleges location data from local storage or the server (if data not stored in local storage)	process	data dataIsRecent currentTimestamp objectToStore localStorage collegeLatLng collegesLocation
7		create markers for each college on the map, add info window for each marker	process	colleges college college.lat college.lon college.address collegeLatLng collegeMarker collegeInfoWindow
8		display markers for the colleges on the map	output	HTML + jQuery Mobile
9		calculate current location of user	process	position.latitude position.longitude MyLatLng
10		create a marker for user's current location on the map, add info window to marker	process	MyLatLng MyMarker myInfoWindow
11		display marker for user's current location on the map	output	HTML + jQuery Mobile

3.3 Prioritised Function List

[Report Requirement No4 - prioritised function list]

3.3.1. Function List for Use Case 1

Action No.	Software	Input/ Process/ Output	Data	Functions
1				
2				
3	display 'Home' screen	output	HTML + jQuery Mobile	
4		input		
5	(fetch rss data from news link and generate list of 'News')	process	feed feed.items item item.title	
6	display 'News' screen	output	HTML + jQuery Mobile	**generateNewsPage();** *//renders the 'News' page and the 'Story' pages*
7		input		
8	(generate data for the 'Innovative State: How New Technologies Can Transform Government' story screen)	process	feed feed.items item item.title item.description item.link	**getFeed(**url**)** -> feed getFeed(*String*) -> *Object* *// returns an Object containing an array of Feed Objects* **Priority 6 (low)**
9	display 'Story' screen for the 'Innovative State: How New Technologies Can Transform Government' article	output	HTML + jQuery Mobile	
10		input		
11	load linked web page	process	item.link	

3.3.2. Function List for Use Case 2

Action No.	Software	Input/ Process/ Output	Data	Functions
6	(check if 'Lecturers' data is stored in local storage and retrieve data if stored locally and if recent)	process	localStorage localStorageName JSONString JSONObject	**saveToLocalStorage(**objectToStore, localStorageName**)** saveToLocalStorage(*Object, String*) *// stores the JSON Object in string format in local storage* **retrieveFromLocalStorage(**localStorageName**)** -> JSONObject retrieveFromLocalStorage(*String*) *-> Object* *// retrieves JSON String from local storage and returns it as an Object* **Priority 2 (high)**
7	(query server and fetch 'Lecturers' data if no data is stored in local storage or if data is old)	process	data dataIsRecent currentTimestamp objectToStore localStorage lecturers lecturer lecturer.firstName lecturer.lastName	**generateLecturersPages()** *// queries the server or local storage for Lecturers data and once it fetches the data calls the renderLecturersPages function passing the data to it*
8	display list of lecturers on 'Lecturers' screen	output	HTML + jQuery Mobile	
9		input		**renderLecturersPages(**data**)** renderLecturersPages(*Object*) *// takes the Lecturers data Object as an argument and renders the 'Lecturers' page and all the 'Lecturer' pages* **Priority 1 (high)**
10	(generate 'Lecturer' screen for Charlie Cullen)	process	lecturer lecturer.firstName lecturer.lastName lecturer.email lecturer.staffNumber lecturer.phoneNumber	
11	display 'Lecturer' screen for Charlie Cullen	output	HTML + jQuery Mobile	
12		input		

3.3.3. Function List for Use Case 3

Action No.	Software	Input/ Process/ Output	Data	Functions
1				
2	display 'Home' screen	output	HTML + jQuery Mobile	
3		input		
4	(check if 'Library' data is stored in local storage and retrieve data if stored locally and if recent)	process	localStorage localStorageName JSONString JSONObject	same as above
5	(query server and fetch 'Library' data if no data is stored in local storage or if data is old)	process	data dataIsRecent currentTimestamp objectToStore localStorage books book book.catNo book.title book.year book.author	**generateLibraryPages()** *// queries the server or local storage for Library data and once it fetches the data calls the renderLibraryPages function passing the data to it* **renderLibraryPages(**data**)** renderLibraryPages(*Object*) *// takes the Library data Object as an argument and renders the 'Library' page and all the 'Title' pages* **Priority 3 (high)**
6	retrieve locations for the colleges from the server or the local storage (if previously stored)	process	data dataIsRecent currentTimestamp objectToStore localStorage collegeLatLng collegesLocation	**fetchCollegesLocations()** -> data fetchCollegesLocations() -> *Object* *// fetches Colleges coordinates data from the server or local storage and returns an Object containing the data* **Priority 3 (high)**

Action No.	Software	Input/ Process/ Output	Data	Functions
7	calculate the distance to each college from current position	process	position.latitude position.longitude collegesLocation distanceToCollege	**calculateDistanceToColleges()** *// uses geolocation to fetch current position and calls fetchCollegesLocations and distance functions to calculate the current distance to each college* **distance(**lat1, lon1, lat2, lon2**)** -> Float distance(*lat1, lon1, lat2, lon2*) -> *Float* *// calculates distance in km between two points by taking these points lat and long coordinates as arguments* **Priority 4 (medium)**
8	generate a list of book titles	process	books book book.catNo book.title book.year book.author	**generateLibraryPages()** **renderLibraryPages(**data**)**
9	display 'Library' screen	output	HTML + jQuery Mobile	same as above **Priority 3 (high)**
10		input		
11	display matching titles in a dropdown menu underneath search box	output	HTML + jQuery Mobile	
12		input		

Action No.	Software	Input/ Process/ Output	Data	Functions
13	generate details for the ‘Title’ screen for the ‘Clean Code’ book including availability and current distance to each library	process	books book book.catNo book.title book.year book.author book.availability.KevinStr book.availability.AungieStr book.availability.BoltonStr book.availability.CathalBrughaStr collegesLocation distanceToCollege	
14	display the ‘Title’ screen for the ‘Clean Code’ book including availability and current distance to each library	output	HTML + jQuery Mobile	

3.3.4. Function List for Use Case 4

Action No.	Software	Input/ Process/ Output	Data	Functions
1		input		
2	display 'Home' screen	output	HTML + jQuery Mobile	
3		input		
4	generate Google map	process	dublinLatLng myOptions map data	$(document).**on('pageinit',** '#mapPage', **drawMap);** **drawMap()** *// draws Google map and adds Markers for all colleges* **Priority 5 (medium)**
5	display Google map	output	HTML + jQuery Mobile	
6	fetch colleges location data from local storage or the server (if data not stored in local storage)	process	data dataIsRecent currentTimestamp objectToStore localStorage collegeLatLng collegesLocation	**fetchCollegesLocations()** -> data fetchCollegesLocations() -> *Object* *// fetches Colleges coordinates data from the server or local storage and returns an Object containing the data* **Priority 3 (high)**
7	create markers for each college on the map, add info window for each marker	process	colleges college college.lat college.lon college.address collegeLatLng collegeMarker collegeInfoWindow	**createCollegeMarker(**data**)** // draws four markers on the map, one for each college, as well as Info Window for each **Priority 5 (medium)**
8	display markers for the colleges on the map	output	HTML + jQuery Mobile	
9	calculate current location of user	process	position.latitude position.longitude MyLatLng	**geoFindMe()** *// uses geolocation to fetch current position of user and creates a Marker on the map with info window*
10	create a marker for user's current location on the map, add info window to marker	process	MyLatLng MyMarker myInfoWindow	

Action No.	Software	Input/ Process/ Output	Data	Functions
11	display marker for user's current location on the map	output	HTML + jQuery Mobile	**Priority 5 (medium)**

4. Implementation

[Report Requirement No5 - coding documentation]

4.1 Implementation of Home Page

[Artefact Requirement No3 - using jQuery Mobile to provide effective UI for the app]
First, I built the home page using a jQuery Mobile 3-column Grid layout *(see Grange_Mobile_App/ public_html/index.html, lines 111 - 135).*

4.2 Implementation of Lecturers, Modules and Student Pages

[Artefact Requirement No1 - at least 3 separate JSON calls]
After that I implemented the Lecturers, Modules and Students pages, making sure to satisfy the requirement for **3 separate JSON calls** to the database. The implementation of all of these followed a similar pattern.

For example, the Modules screen is a jQuery Mobile page with a listview, the <ul> tag having its own **id** attribute called **"modulesPageList"** *(see Grange_Mobile_App/ public_html/ index.html, line 205)*, which is populated when a JavaScript function **generateModulesPages()** is called *(see Grange_Mobile_App/ public_html/ javascript/ pagesGeneratingFunctions.js, lines 89 - 122)*

```
88  // generate Modules pages
89  function generateModulesPages() {
90
91      // try to retrieve the data from local storage
92      var data = retrieveFromLocalStorage('modules');
93
94      // initialise variable DataIsRecent to false
95      var dataIsRecent = false;
96
97      // get seconds since epoch (1 jan 1970)
98      var currentTimestamp = new Date()/1000;
99
100     // if there is data stored in local storage
101     if (data) {
102         // check if that data is not older than 1 day
103         if (currentTimestamp - data.timestamp < 24*60*60){
104             // if it is set DataIsRecent to true
105             dataIsRecent = true;
106         }
107     }
108     //  if there is data stored in local storage and that data is not older than 1 day
109     if (data && dataIsRecent) {
110         // render it
111         renderModulesPages(data);
112     }
113     else { // if no data is stored in local storage or data is older than 1 day
114         // make MODULES json call
115         $.getJSON('php/json-data-modules.php', function(returnedData) { // because we
116             // save data to local storage
117             saveToLocalStorage(returnedData, 'modules');
118             // render data
119             renderModulesPages(returnedData);
120         });
121     }
122 }
123
```

The **generateModulesPages()** function works by first checking if there is data stored in the local storage, and if there is it checks if the data is older than 1 day - if it is, it makes a JSON call to the server and once it receives the new data saves it to the local storage for later use and calls a function called **renderModulesPages(data)**. However, if the data is not older than 1 day it calls the **renderModulesPages(data)** function straight away *(see Grange_Mobile_App/ public_html/ javascript/ pagesGeneratingFunctions.js, lines 125- 168)*.

```
function renderModulesPages(data){

    // start modules loop
    $.each(data.modules, function(index, module) {

        //build modules' list page
        $("#modulesPageList").append('<li><a href="#' + module.moduleNo + '">' + module.moduleName + '</a></li>');

        //build module's details page
        var modulePageHtml = '';

        // page (open div)
        modulePageHtml += '<div data-role="page" id="' + module.moduleNo + '">';

            // header (open div)
            modulePageHtml += '<div style="background-color:#474747; height:60px;" data-role="header">';

                // breadcrumb navigation
                modulePageHtml += '<div class="ui-grid-d">';
                modulePageHtml += '<div class="ui-block-a breadcrumbHome"><a href="#homePage"><img src="images/menu_home_small.png" class="ui-
                modulePageHtml += '<div class="ui-block-b breadcrumbRecent"><a href="#modulesPage"><img src="images/menu_modules.png" class="u
                modulePageHtml += '<div class="ui-block-c breadcrumbCurrent"><h4 class="breadcrumbText">Module</h4></div></div>';

            // header (close div)
            modulePageHtml += '</div>';

            // content (open div)
            modulePageHtml += '<div data-role="main" class="ui-content">';
                // content sections
                modulePageHtml += '<div class="ui-body ui-body-a ui-corner-all">module<h4>' + module.moduleName + '</h4></div><br/>';
                modulePageHtml += '<div class="ui-body ui-body-a ui-corner-all">website<h4><a href="'+ module.website +'">'+ module.website +'
                modulePageHtml += '<div class="ui-body ui-body-a ui-corner-all">location<h4>'+ module.location +' </h4>room<h4>' + module.room
                modulePageHtml += '<div class="ui-body ui-body-a ui-corner-all">credits<h4>' + module.credits + '</h4></div><br/>';
            // content (close div)
            modulePageHtml += '</div>';
            // footer
            //modulePageHtml += '<div data-role="footer"><h5>Grange Mobile 2014</h5></div>';
        // page (close div)
        modulePageHtml += '</div>';

        $('body').append(modulePageHtml);

    }); // end modules loop
}
```

The **renderModulesPages** function then loops through the 'modules' JSON object and appends a clickable list item in the Modules page listview for each module *(line 131)*. It then builds a module details page for each module page *(lines 133 - 165)*.

4.3 Structuring of Library, Social, News and Schools Pages

Throughout the project I re-used the above methods from section 4.2 to structure the Library page, Social page, News page and the various Schools pages (Art College, Engineering College, Business College, Science College) *(see Grange_Mobile_App/ public_html/ javascript/ pagesGeneratingFunctions.js)*.

In order to implement all those pages I extended the provided SQL database *(see Grange_Mobile_App/ public_html/ database/ collegeDatabase.sql)*.

I also added new PHP scripts to query and return JSON formatted data for each new database table *(see Grange_Mobile_App/ public_html/ php/)*.

4.4 Implementation of Multimedia, Website and About Pages

[Extra Credit Requirement No2 - adding rich media handling]

Next, I implemented all the hard-coded pages in the app - the Multimedia page, **using HTML5 rich media handling** *(see an example at Grange_Mobile_App/ public_html/ index.html, lines 296 - 299)*, the Website page, and the About page. These did not require the use of any interesting functions.

4.5 Functionality of News and Social Pages

[Extra Credit Requirement No3 - using AJAX and JSON calls to fetch RSS and JSON data respectively from third-party data APIs]

After that I implemented the functionality for the News and Social pages, the first being a RSS news feed and the second being a Twitter feed ().

(*See functions* **generateNewsPage()** *and* **generateSocialPage()** *in Grange_Mobile_App/ public_html/ javascript/ pagesGeneratingFunctions.js, lines 552 - 612; and 616 - 634).*

The **generateNewsPage()** function *(source https://github.com/jfhovinne/jFeed/blob/master/example.html)* works by making an AJAX call to fetch RSS formatted data and then loops through that data to build the News page first and then build each individual 'Story' page.

To be able to implement this I was forced to save the formatted data locally beforehand in an xml file *(See Grange_Mobile_App/ public_html/ rss/ rss.xml)* rather than accessing it directly through the url link because (as I discovered later on) javascript code in a web page is allowed only to make requests to the same IP address where it itself (the page plus the JavaScript) came from for security reasons. However, if this app was not for a fictional institution but was commissioned and deployed in reality, it would be served from the server that had the RSS on it (i.e. the Grange server) and this issue won't happen.

I ran into one more issue with the implementation of the RSS feed. Originally I wanted to use the DIT News Link from http://www.dit.ie/news/rss/index.xml but this wasn't very well formatted and instead I used a Harvard News Link http://news.harvard.edu/gazette/rss-feeds/

For the implementation of the **generateSocialPage()** function I re-used the example code from lecture 10 and the lab but modified the app_tokens.php file to use my own OAuth tokens and in the timeline_response.php file changed the 'screen_name' to 'ditofficial' so I that I could populate the page with DIT's official tweets. I also modified the php code slightly to accumulate the tweets into an array *(See Grange_Mobile_App/ public_html/ php/ timeline_response.php, lines 24 - 30)*

4.6 Implementation of the Map Page

[Extra Credit Requirement No4 - adding locationing functionality]

After a lot of trial and error I implemented the Map page as follows:

First of all, this is the only page in the entire app that doesn't load fully at the launch of the application. It waits for the page to initialise itself before even starting the Google map code *(hence the **on ('pageinit')** function, See Grange_Mobile_App/ public_html/ index.html, line 67)* but then the Google map waits for 1/2 a second before triggering a resize event. This delay gives the page enough time to render itself so that by the time the Google resizing function works, the page is fully drawn and the map can draw itself at the correct size.

```
96                  // initialise Google Map on Map page
97                  $(document).on('pageinit', '#mapPage', drawMap);
```

This is necessary because of the fact that the jQuery Mobile "pages" aren't proper HTML pages as such - they are really just styled s. As it turns out when these virtual pages are created, when the app loads, they don't yet have a proper set of pixel dimensions because they aren't properly rendered on screen yet. This means that the JavaScript code that Google maps uses to work out how to draw itself gets confused. So, when the page really is drawn on the screen it gets refreshed by resizing itself now that it has some real dimensions to work with.

```
5   var map;
6
7   // draw google map, add markers for all colleges
8   function drawMap() {
9
10      var dublinLatLng = new google.maps.LatLng(53.338545, -6.26607); // position set to Dublin coordinates
11      var myOptions = {
12              zoom: 14,
13              center: dublinLatLng,
14              mapTypeId: google.maps.MapTypeId.ROADMAP,
15              disableDefaultUI: true
16              //styles: styles
17          };
18
19      //map = new google.maps.Map($("#map-canvas"), myOptions);
20      map = new google.maps.Map(document.getElementById('map-canvas'), myOptions);
21
22      var data = fetchCollegesLocations();
23      //console.log(data);
24
25      createCollegeMarker(data);
26
27      geoFindMe();
28
29      setTimeout(function(){
30          google.maps.event.trigger(map, "resize");
31          map.setCenter(dublinLatLng);
32       }, 500); // wait for page to render, then resize
33
34  }
```

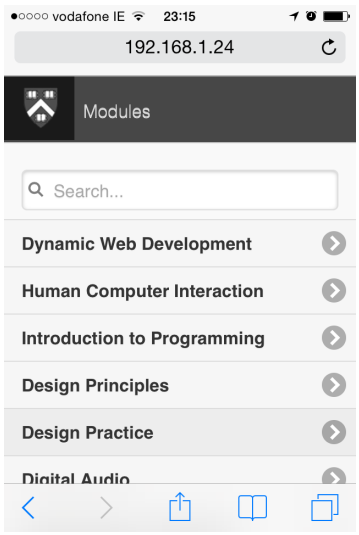
The **drawMap()** function *(See Grange_Mobile_App/ public_html/ javascript/ GoogleMapsScript.js, lines 7 - 34)* works by drawing a map first *(source https://developers.google.com/maps/documentation/javascript/examples/map-simple)* and then calling the **fetchCollegesLocations()** function *(See Grange_Mobile_App/ public_html/ php/ timeline_response.php, lines 36 - 70)*, which retrieves the coordinates of the four colleges from the database or from the local storage (logic is the same as with the generateModulesPages() function from 4.2) and returns them as a JSON object.

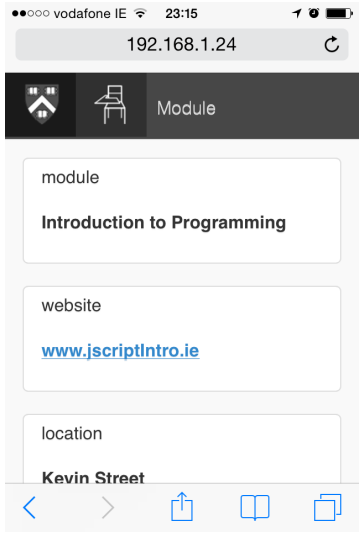
```
72  // create markers for the colleges on the map
73  function createCollegeMarker(data){
74
75      var collegeLatLng;
76      var collegeMarker;
77      var collegeInfoWindow;
78
79      // start colleges loop
80      $.each(data.colleges, function(index, college) {
81
82              var collegeLatLng = new google.maps.LatLng(college.lat, college.lon);
83              //create a marker for for each college
84              var collegeMarker = new google.maps.Marker({
85                  position: collegeLatLng,
86                  map: map,
87                  title: college.address
88              });
89
90              var collegeInfoWindow = new google.maps.InfoWindow({
91                content: college.address
92              });
93              collegeInfoWindow.open(map,collegeMarker);
94
95              google.maps.event.addListener(collegeMarker, 'click', function() {
96                  collegeInfoWindow.open(map,collegeMarker);
97              });
98      }); // end colleges loop
99  }
```

Then the **createCollegeMarker(data)** function is called, taking as an argument that JSON object, and generating markers for each of the four colleges on the map featuring popped-up info windows *(See Grange_Mobile_App/ public_html/ javascript/ GoogleMapsScript.js, lines 72 - 99)*.

```
101 // geolocation
102 function geoFindMe() {
103   var output = $("#out");
104
105   if (!navigator.geolocation){
106     output.text("Geolocation is not supported by your browser");
107     return;
108   }
109
110   function success(position) {
111     var latitude  = position.coords.latitude;
112     var longitude = position.coords.longitude;
113
114     output.text("Map");
115
116
117     //create a marker for current position
118     var myLatLng = new google.maps.LatLng(latitude, longitude);
119     var myMarker = new google.maps.Marker({
120         position: myLatLng,
121         map: map,
122         title: 'current position'
123     });
124     var myInfowindow = new google.maps.InfoWindow({
125           content: 'current location'
126         });
127         myInfowindow.open(map,myMarker);
128
129         google.maps.event.addListener(myMarker, 'click', function() {
130             myInfowindow.open(map,myMarker);
131         });
132   };
133
134   function error() {
135     output.text("Unable to retrieve your location");
136   };
137
138   output.text("Locating…");
139
140   navigator.geolocation.getCurrentPosition(success, error);
141 }
```

After that the **geoFindMe()** function gets called *(source https://developers.google.com/maps/documentation/javascript/examples/map-geolocation, See Grange_Mobile_App/ public_html/ javascript/ GoogleMapsScript.js, lines 101 - 141)* fetches the current position and creates a marker with a popped-up info window. Finally the resize event is triggered with half a second delay.





4.7 Implementation of the Breadcrumb Navigation Menu

[Artefact Requirement No3 - using jQuery Mobile to provide effective UI for the app]

Next, I implemented the Breadcrumb navigation menu. This was fairly straightforward. For an example of the code: (*See Grange_Mobile_App/ public_html/ javascript/ pagesGeneratingFunctions.js, lines 139 - 149)*

```
<!-- Modules Page -->
<div data-role="page" id="modulesPage">
    <div style="background-color:#474747; height:60px;" data-role="header">
        <!-- Breadcrumb Navigation -->
        <div data-role="listview" class="ui-grid-d">
            <div class="ui-block-a breadcrumbHome"><a href="#homePage"><img src="images/menu_home_small.png" class="ui-btn breadcrumbImage"
            <div class="ui-block-c breadcrumbCurrent"><h4 class="breadcrumbText">Modules</h4></div>
        </div>
        <!-- End of Breadcrumb Navigation -->
    </div>
    <div data-role="main" class="ui-content">
        <ul data-role="listview" data-filter="true" data-filter-placeholder="Search..." id='modulesPageList'>
        </ul>
    </div>
    <div data-role="footer">
        <h5>Grange Mobile 2014</h5>
    </div>
</div>
<!-- End of Modules Page -->
```

```
// header (open div)
modulePageHtml += '<div style="background-color:#474747; height:60px;" data-role="header">';

    // breadcrumb navigation
    modulePageHtml += '<div class="ui-grid-d">';
    modulePageHtml += '<div class="ui-block-a breadcrumbHome"><a href="#homePage"><img src="images/menu_home_small.png" class="ui-btn breadcrumbImage"></
    modulePageHtml += '<div class="ui-block-b breadcrumbRecent"><a href="#modulesPage"><img src="images/menu_modules.png" class="ui-btn breadcrumbImage">
    modulePageHtml += '<div class="ui-block-c breadcrumbCurrent"><h4 class="breadcrumbText">Module</h4></div></div>';

// header (close div)
modulePageHtml += '</div>';
```

4.8 Implementation of the Use of Local Storage (for offline use of the app)

[Artefact Requirement No4 - using HTML5 functionality]
[Extra Credit Requirement No5 - any other extension beyond the taught material]

After that I implemented the use of local storage by creating two functions: **saveToLocalStorage** and **retrieveFromLocalStorage** to save all the fetched JSON formatted data and re-use later in order to be able to use the app offline, as well as to help the app perform faster by not having to fetch data from the server so often.
(See Grange_Mobile_App/ public_html/ javascript/ useLocalStorageFunctions.js, lines 6 - 36)

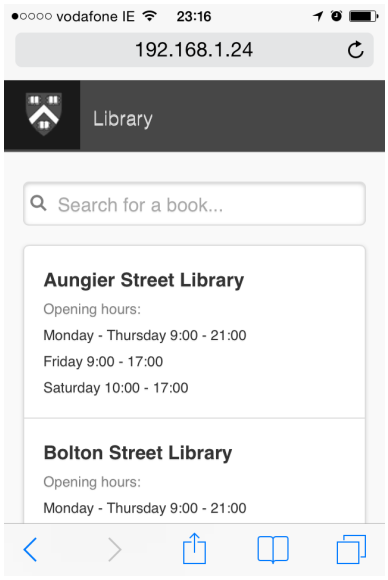
```
6   // save json string to local storage
7   function saveToLocalStorage(objectToStore, localStorageName){
8
9       // first check if HTML5 storage is available
10      if (typeof(Storage) !== 'undefined') {
11
12          // save seconds since epoch (1 jan 1970)
13          objectToStore.timestamp = new Date()/ 1000;
14
15          localStorage[localStorageName] = JSON.stringify(objectToStore);
16      }
17
18  }
19
20  // retrieve json string from local storage
21  function retrieveFromLocalStorage(localStorageName){
22
23      // check if json string is already stored
24          if (typeof localStorage[localStorageName] !== 'undefined') {
25
26              var JSONString = localStorage[localStorageName];
27              // parse JSON String and convert to JavaScript Object
28              JSONObject = JSON.parse(JSONString);
29
30              return JSONObject;
31          }
32          else{
33              return false;
34          }
35
36  }
```

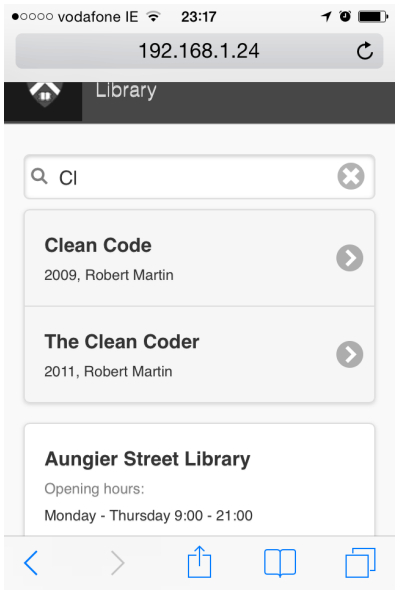
The **saveToLocalStorage(objectToStore, localStorageName)** function takes two arguments - **objectToStore** is the formatted JSON data fetched from the server, while **localStorageName** will be the name of the new object to be stored in the **localStorage** object. Before saving the data it first checks if HTML5 storage is available in the browser and if it is, it *stringifies* the **objectToStore** because the only format we could keep it in local storage as, is a String.

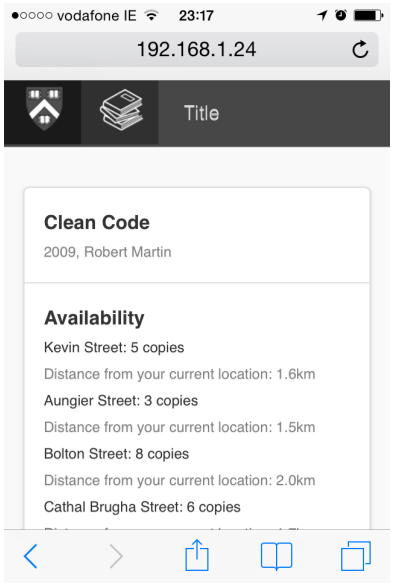
The **retrieveFromLocalStorage(localStorageName)** function first checks if anything is stored in **localStorage**, and if it is, parses the JSON string and converts it to a JavaScript object, and finally returns the object.

4.9 Implementation of the functionality of the Library Page

Finally I implemented the Library page functionality, which uses the jQuery Mobile filter reveal feature. This feature makes is easy to build a simple autocomplete using local data. When the filterable list has the data-filter-reveal="true" attribute, it auto-hides all the list items when the search field is blank.







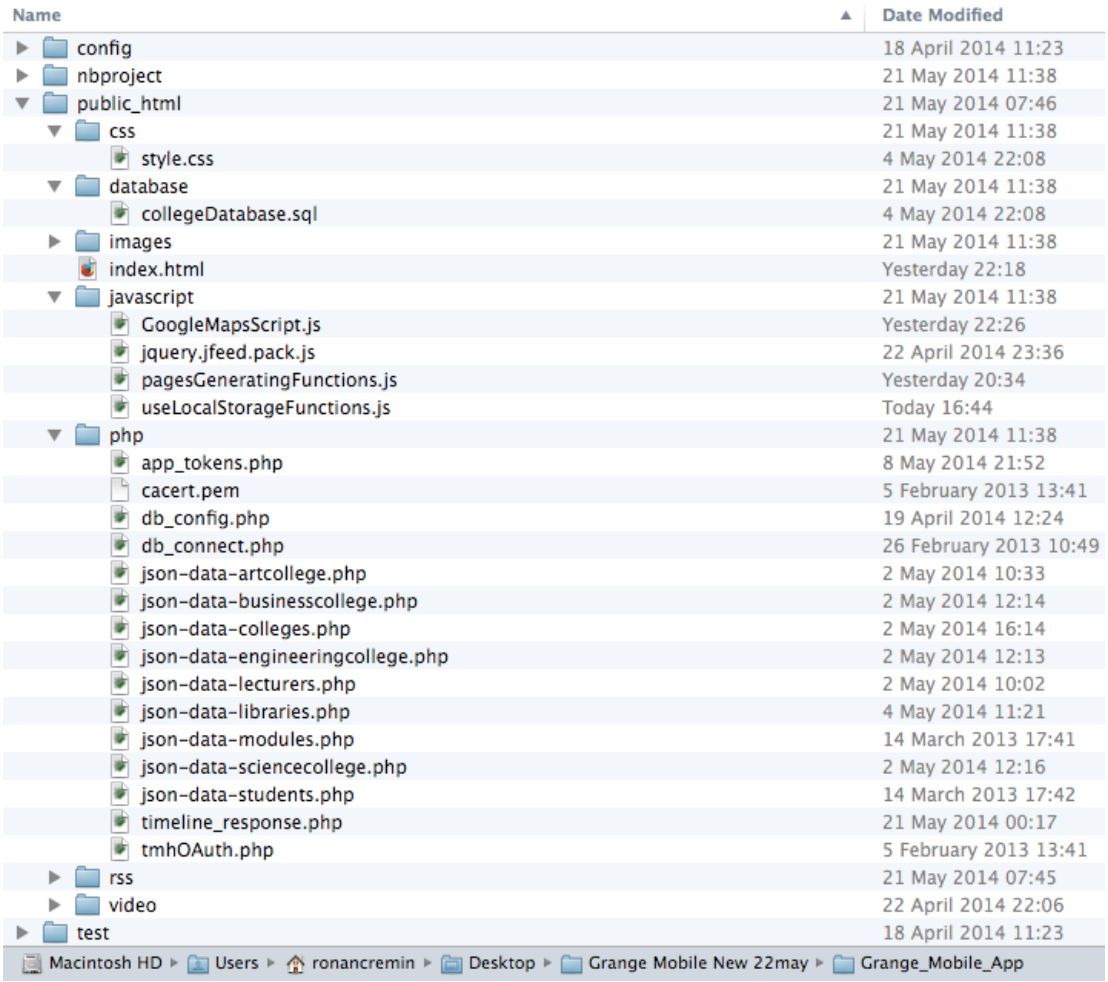
[Artefact Requirement No2 - using jQuery to handle and display results of JSON calls]
But the more interesting part of this was implementing the calculation for the user's current distance to each library and displaying it on the page. I implemented this by having the **renderLibraryPages(data)** function *(See Grange_Mobile_App/ public_html/ javascript/ pagesGeneratingFunctions.js, lines 295 - 356)* first retrieve the colleges locations by calling the **fetchCollegesLocations()** function *(see section 4.6)* and then while looping the books fetched data, calculating the distance to each college from the current position by calling the **calculateDistanceToColleges()** function *(See Grange_Mobile_App/ public_html/ javascript/ GoogleMapsScript.js, lines 144 - 177)*, which uses the **distance** function to calculate the distance between two points *(source http://www.geodatasource.com/developers/javascript, (See Grange_Mobile_App/ public_html/ javascript/ GoogleMapsScript.js, lines 180 - 197))*

```
144  // calculate distance to colleges
145  function calculateDistanceToColleges() {
146    var output = $(".distance");
147
148    if (!navigator.geolocation){
149      output.text("Geolocation is not supported by your browser");
150      return;
151    }
152
153    function success(position) {
154      var myLat  = position.coords.latitude;
155      var myLon = position.coords.longitude;
156
157      //console.log("successfully fetches current coordinates");
158      // retrieve object containing colleges' coordinates
159      var collegesLocation = fetchCollegesLocations();
160      // calculate distance
161      var distToCollege;
162      for (var i=0; i<4; i++){
163          var output = $('.' + collegesLocation.colleges[i].address.replace(/ /g, ''));
164          //console.log('Replacing' + collegesLocation.colleges[i].address.replace(/ /g, ''))
165          distToCollege = distance(myLat, myLon, collegesLocation.colleges[i].lat, collegesLocation.colleges[i].lon);
166          output.text("Distance from your current location: " + distToCollege + "km");
167      }
168    };
169
170    function error() {
171      output.text("Unable to retrieve your location");
172    };
173
174    output.text("Locating…");
175
176    navigator.geolocation.getCurrentPosition(success, error);
177  }
```

```
180  // calculate distance between two points
181  // source http://www.geodatasource.com/developers/javascript
182  function distance(lat1, lon1, lat2, lon2) {
183
184      var radlat1 = Math.PI * lat1/180;
185      var radlat2 = Math.PI * lat2/180;
186      var radlon1 = Math.PI * lon1/180;
187      var radlon2 = Math.PI * lon2/180;
188      var theta = lon1-lon2;
189      var radtheta = Math.PI * theta/180;
190
191      var dist = Math.sin(radlat1) * Math.sin(radlat2) + Math.cos(radlat1) * Math.cos(radlat2) * Math.cos(radtheta);
192      dist = Math.acos(dist);
193      dist = dist * 180/Math.PI;
194      dist = dist * 60 * 1.1515 * 1.609344;
195
196      return (dist).toFixed(1);
197  }
198
```

4.10 Implementation of appropriate folder structure

[Artefact Requirement No5 - appropriate folder structure for all assets and code]
Appropriate folder structure is demonstrated in the image below:



5. Testing

No	Page	Functionality	Test successful
1	Home page	displays correctly, all links work	yes
2	Lecturers page	data displays correctly, navigation and all links work	yes
3	Lecturer pages	data displays correctly, navigation and all links work	yes
4	Map page	map displays correctly and markers are positioned in the correct locations, detects current position correctly, loads quickly	yes
5	Modules page	data displays correctly, navigation and all links work	yes
6	News page	data displays correctly, navigation and all links work	yes
7	Story page	data displays correctly, navigation and all links work	yes
8	Students page	data displays correctly, navigation and all links work	yes
9	Student pages	data displays correctly, navigation and all links work	yes
10	Multimedia page	data displays correctly, navigation and all links work	yes
11	Video pages	page displays correctly, video plays correctly, navigation and all link work	yes
12	Social page	displays full text of DIT tweets correctly	yes
13	Library page	reveal filter functionality performs correctly	yes
14	Title pages	data displays correctly, navigation and all links work, distance is calculated correctly	yes
15	Schools page	data displays correctly, navigation and all links work	yes
16	Colleges pages	data displays correctly, navigation and all links work	yes
17	Website link	link to website works	yes
18	About page	displays correctly	yes
19	Breadcrumb Navigation	functions correctly (firefox has a small issue with images)	yes
20	Local Storage	stores and retrieves data successfully	yes

6. Critical Analysis

No	Page/ Functionality	Worked	Didn't Work
1	Home page	Easy, learned about implementing Grid Layout with images in jQuery Mobile	
2	Lecturers pages	No issues	
3	Map page	This was one of the more challenging pages to implement. As I have explained above there were issues with the map not displaying properly. In the end I managed to solve the issue and I am happy with the result.	My idea in the beginning was that this page would have more functionality, for example I wanted the info window for the markers to display more information about the schools and include a photo and a web link. Also I wanted to link the map with the Library page. However, I didn't have time to do this.
4	Modules pages	No issues	
5	News pages	I am happy with my implementation for the News page. I learned a lot while trying to solve the source issue (as explained in detail in the implementation section)	As explained in the implementation section, I wanted to use DIT's news feed but because it included photos and it wasn't well structured I had to use news feed from somewhere else. Another problem I ran into was the fact that I couldn't fetch the RSS data directly from the URL and resolved to saving an xml file locally.
6	Students pages	I had some choppiness while scrolling up and down, which took a while to debug. It was due to orphaned tags. Very happy with solving this one!	
7	Multimedia page	Overall happy with the implementation.	Ended up hardcoding all the video pages, maybe could have come up with a more elegant solution. Also didn't have time in the end to download the respective DIT video from YouTube, so used the same copy of the video from lectures for all videos. Didn't also have time to investigate for more HTML5 specific controls.
8	Social page	Very happy with managing to fetch the DIT tweets and also fitting the whole text in each box. This page took quite a while to research, debug etc.	

No	Page/ Functionality	Worked	Didn't Work
9	Library page	This was very challenging, as it is packed with a lot of functionality. The main issue was the fetchCollegeLocations function - because is used both by the map and the library pages, it was difficult to coordinate between the two. I am very happy with how this worked out in the end.	I wanted to dynamically display the opening hours and have a function which tell the user if a given library is closed or open at the moment but had no time to implement it.
10	Schools page	No issues	
11	Website link	No issues	
12	About page	No issues	
13	Breadcrumb Navigation	Works well in Chrome and Safari. Happy with how it turned out!	Firefox displays the icon too big, had no time to fix this.
14	Local Storage	Very proud with coming up with this functionality - I wanted to demonstrate the use of HTML5 specific features. Some head wrecking if-else statements as well here, which took a lot of time to get right.	
15	Phone Gap		Wanted to package my app in PhoneGap but didn't leave enough time and when I finally did it turns out it's not as straightforward as just dragging your html/javascript etc. It seemed like it needed a lot of debugging. Lesson learned for the future!
16	Factoring out the JavaScript and CSS	It worked in the end but I left it to the end - big mistake as it took so much longer than if I had been doing it all along. Another lesson learned here!	

7. Conclusion

I am happy with my work for this assignment. Overall I managed to implement almost everything I set out to do, while learning a lot along the way. I believe this experience will be very beneficial later on for the implementation of the major project.