Mobile Application Development

Session 7 – Activities

The final product(s) for this session can be found at:
http://www.dcs.bbk.ac.uk/lo/mad/madexamples/session7/classactivities/zedlandhotels/zedland-hotels.html

http://www.dcs.bbk.ac.uk/lo/mad/madexamples/session7/classactivities/zedlandhotelsdynamic/hotellist.html

1. Physical Events

In this activity, we will add a share facility to the Zedland Hotel finder, so that the user can share the app social media. The user will be able to access the sharing facility by performing a tap on any of several share icons located on the app home page.

Full social media sharing will not be implemented at this point. But this can be done later, in your own time, using a jQuery Mobile sharing plug-in (e.g. http://sharrre.com/).

HTML

We start by creating the jQM popup which will display when sharing is selected.

In *zedland-hotels.html*, in the content section of #home, create a <div> for the sharing popup with the following attributes and values:

Attribute	Value	Description
data-role	popup	define popup
id	sharepopup	element id
data-theme	b	changes theme
data-dismissible	false	popup cannot be closed by clicking outside
data-history	false	popup not added to navigation history
data-position-to	window	positions popup to centre of window

Inside the newly created popup <div>, create a header and a content section, as you would for a normal jQM page. In the header, add an <h1> heading with a value of *Share*. In the content section, create two buttons: *Share to* and *Cancel*. The buttons should be given a *data-rel* attribute with a value of *back*. Finally, add a <span> tag just after the text in the Share to link (This will allow us to present a custom share message to our user). Give this <span> an *id* of *sharechannel*.

```
<div data-role="popup" id="sharepopup" data-theme="b" data- →
dismissible="false"  data-history="false" data-position-to="window">
  <div data-role="header">
    <h1>Share</h1>
  </div>
  <div role="content">
    <a href="#" data-role="button" data-rel="back">Share to <span →
    id="sharechannel"></span></a>
    <a href="#" data-role="button" data-rel="back">Cancel</a>
  </div>
</div>
```

The next step is to create the share images and render them as links (These images are available in the session 7 resources folder).

Just before the newly created popup in *zedland-hotels.html*, add a new <a> element. Give this element the following attributes and values:

Attribute	Value	Description
href	sharepopup	references the popup to open
id	sharelinks	provides unique id
data-rel	popup	indicates to the framework the type of page to be opened

Inside this anchor element, include <img> tags for the Facebook, Instagram and Twitter images. Give each image a unique id as below, and a class attribute of *shareimage*.

```
<a href="#sharepopup" id="sharelinks" data-rel="popup" >
  <img src="images/facebook.png" id="facebook" →
  class="shareimage" alt="Share on Facebook">
  <img src="images/twitter.png" id="twitter" →
  class="shareimage" alt="Share on Twitter">
  <img src="images/linkedin.png" id="linkedin" →
  class="shareimage" alt="Share on LinkedIn">
</a>
```

Save *zedland-hotels.html*. test that your popup is working.

CSS

The share images need to be formatted so that they present properly across different screen widths. This will be done using responsive web design techniques.

Create a new CSS file. Save this file as *styles.css*. Reference this CSS file on each HTML page of your application.

In *styles.css*, create a class selector for the .*shareimage* buttons. In this selector, set the values for the image width, height, and max and min widths and heights.

Attribute	Value
width	5%
height	5%
max-width	50px
max-height	50px
min-width	30px
min-height	30px

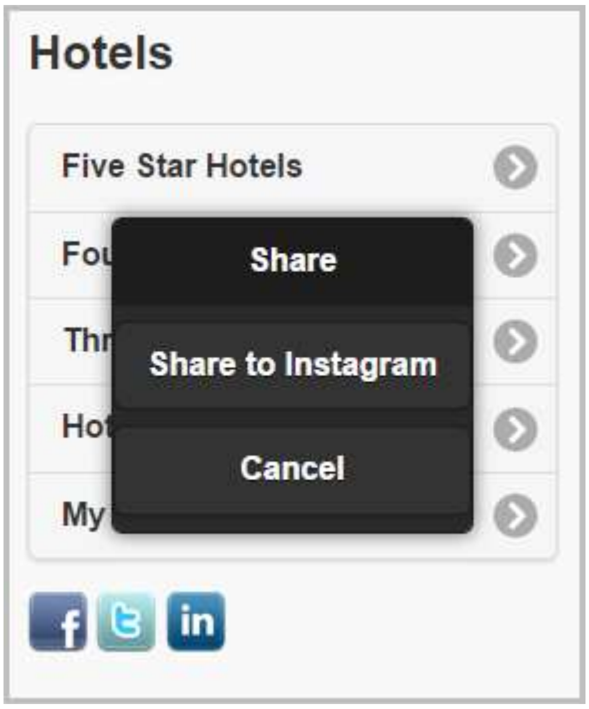
Next, create a selector for the anchor element that contains the images. Call this selector *#sharelinks*. Set the *text-decoration* for this element to *none*.

```
.shareimage {
width: 5%;
height: 5%;
min-width: 30px;
min-height: 30px;
max-width: 50px;
max-height: 50px;
}

#sharelinks {
text-decoration: none;
}
```

jQuery

The share functionality will work by presenting a custom share message to the user when he/she taps on any of the share images. We will use jQuery to create the custom message.



Create a new JavaScript file. Save this file as *share.js*. Link all of your application pages to this script.

In *share.js*, create a new *pagecreate* event handler and associate this handler with #*home, as shown below*.

```
$(document).on("pagecreate", "#home", function (e, ui) {
. . .
});
```

Inside this event handler, we will create a script to capture tap events that occur on any of our share images. If a tap event occurs, we will display a custom sharing message. This will be presented in the *#sharechannel* <span> element in the popup.

```
$(document).on("pagecreate", "#home", function (e, ui) {
   $("#facebook").bind("tap", function (e) {
       $("#sharechannel").html("Facebook");
   });
   $("#twitter").bind("tap", function (e) {
       $("#sharechannel").html("Twitter");
   });
   $("#linkedin").bind("tap", function (e) {
       $("#sharechannel").html("LinkedIn");
   });
});
```

2. Page Events

In this activity, we will rework the Zedland Hotel application, so that the list of hotels on the *#five* page is dynamically generated from external data. We will also rework the way hotel details are displayed. Instead of having them hardcoded on individual pages, we will use a single dynamically generated page, acquiring the content for that page from external data sources – in this case simple *.json* file (Though we will be using simple JSON files, the example will work just as well with PHP/MySQL, etc).

The Data

The JSON files are available in the *data* folder in Moodle, Session 7 Resources. This folder includes one file containing a full list of all the hotels (*hotels.json*), and fifteen separate files with the data for fifteen hotels (*hotelx.json*). Copy the *data* folder to the root directory of your application.

Dynamic List of Hotels

HTML

Open *zedland-hotels.html.* In the #five page, delete the existing hard-coded unordered list of hotel links. Then create a new unordered list. Give this list an *id* of *hotellist*. Give it a *data-role* of *listview*. Finally, set *data-inset* and data-filter to *true*.

```
<div data-role="content">
  <h2>Five Star Hotels</h2>
  <ul id="hotellist" data-role="listview" data-inset="true" →
  data-filter="true"></ul>
</div>
```

This list is where we will display our list of hotels. It will be dynamically populated from the JSON source file (*hotels.json*).

Save *zedland-hotels.html.*

jQuery
Create a new JavaScript file. Name the file *createhotellist.js.* Link ALL of your HTML pages to this file.

Start by creating a *pagecontainerbeforeshow* event handler.

```
$(document).on("pagecontainerbeforeshow", function (e, ui) {
 . . .
});
```

We will use *pagecontainerbeforeshow* rather than *pagecreate* because the list of hotels is going to be dynamically populated by an external data source each time the list page is visited. Because we are using a pagecontainer event, however, we have to find a way of running our script only when we are transitioning to the page holding the list of hotels (*#five*). We do this programmatically by finding the id of the *to* page in our transition using the *toPage* method of the *ui* object.

```
$(document).on("pagecontainerbeforeshow", function (e, ui) {
  var page = ui.toPage[0].id;
  if( page == 'five' ) {
    . . . .
  }
});
```

The next step is to create the code that will get the data from *hotels.json*, and use that data to populate our empty unordered list. To do this we will use the jQuery *get()* method.

The *get()* method loads data from an external data source. It takes three arguments (1) the path to the data source (2) a function to run, if the request for data succeeds (3) the type of data being retrieved.

```
if( page == 'five' ) {
  $.get("data/hotels.json", function(result, status) {
    . . . .
  }, "json");
}
```

Note that the function part of the *get* method itself takes two parameters (1) *result* (returned data) (2) status (e.g. whether the data request was successful). The data returned by the function is contained in the *result* object. To access data in *result*, we will use a *for* loop. As we traverse the collection, each time we encounter hotel details, we will extract those details and use them to build our list of hotels. Each <li> in our list will be a link which when clicked will take us to a hotel details page.

```
if( page == 'five' ) {
  $.get("data/hotels.json", function(result, status) {
    var hotel = "";
    for (var i = 0; i < result.length; i++) {
    hotel += "<li><a href='hotel.html?id=" + result[i].id + →
    "'>" + result[i].name + "</a></li>";
    }
  }, "json");
}
```

Though we have created a collection of <li> tags, they have not yet been appended to the <ul>, *#hotellist*. To do this we set the *html* value of the <ul> to the value of the string variable which contains our collection of list items. We then update the <ul> by running *the refresh()* method on it. Note that the *refresh()* method only affects new nodes appended to a list: not the existing list nodes.

```
$.get("data/hotels.json", function(result, status) {
    var hotel = "";
    for (var i = 0; i < result.length; i++) {
    hotel += "<li><a href='hotel.html?id=" + result[i].id + →
    "'>" + result[i].name + "</a></li>";
    }
    $("#hotellist").html(hotel).listview("refresh");
  }, "json");
```

Save *createhotellist.js.* Upload your HTML and JavaScript files to the DCSIS server. Make sure the hotel list is being generated correctly. Use a JavaScript console to identify and fix any errors. The dynamically generated URL for each hotel can be seen by placing the mouse over each link. You should see a querystring appended to the URL for each hotel (e.g. *hotel.html?=1*).

Displaying Hotels

HTML

We will use a single HTML page to display the hotels from the hotel list we created on *#five*.

Create a new HTML document. Name it *hotel.html*. Create a new jQM page in *hotel.html*. Give it an id of *#hotelpage*. In the content section of this page, create an <h2> element, two <div> elements and two anchor elements, with attributes and values as shown below.

```
div data-role="content">
  <h2 id="hoteltitle"></h2>
  <div id="hotelid"></div>
  <div id="contentArea"></div>
  <a href="#" data-role="button" data-icon="star" →
  id="addbtn">Favourites</a>
  <a href="login.html" data-role="button" data-icon="action" →
  id="resbtn">Reservation</a>
</div>
```

#hoteltitle is where we will display the hotel title. *#hotelid* is included, but it will not be displayed. It will be hidden using the CSS display property. It is there simply to allow us to add hotels to the favourites list.

Note that the favourites list as we created it last week was for hard-coded hotel pages. Here, we are using dynamic pages, and will need a different approach to adding favourites involving hotel ids.

Make sure you reference all of the site resources (e.g. CSS and JavaScript files) in the head of *hotel.html*, in exactly the same way you did for other HTML pages in the app.

Save *hotel.html*.

jQuery

We want *hotel.html* to dynamically load hotel data each time a user chooses a hotel from the list of hotels on *zedland-hotels.html#five*. Because of this, we again need to use an event that fires each time the page is visited, rather than only once when the page is first loaded or created. To satisfy this requirement, we will again use *pagecontainerbeforeshow*. Note that we use *pagecontainerbeforeshow* as opposed to *pagecontainershow*, because we want the hotel data to be in place before the transition is underway.

Create a new JavaScript file. Name the file, *showhotel.js*. Create the *pagecontainerbeforeshow* event handler and use the *toPage* method of the *ui* object to limit the scope of the script to *#hotelpage*.

```
$(document).on("pagecontainerbeforeshow", function (e, ui) {
  var page = ui.toPage[0].id;
  if( page == 'hotelpage' ) {
  . . . .
  }
});
```

To retrieve hotel data, we have to know the id of the hotel which was requested. This information comes from the querystring in the URL on *#five* (e.g*. hotel.html?id=1*). Our first task is to therefore extrapolate the id from the querystring. We do this by (1) finding out the active page, (2) finding the URL of that page, (3) extracting the querystring value from the URL. To get the active page, we use the *toPage* method of the ui object, as previously. To get the URL, we use the jQuery's $(location).attr('search'). To extract the querystring value, we use the jQuery *split()* method.

split () works by splitting a string into an array of substrings. The array will have two values: [0], text before the split point; [1], text after the split point. We can then access the value we require by using the array index.

```
if(page === 'hotelpage') {
  var thisPage = "#" + page;
  var thisUrl = $(location).attr('search');
  var thisId = thisUrl.split("=")[1];
 . . . .
}
```

Building the File Path
Once we have the hotel id, we can use it with the *get*() method to extract the hotel data from the relevant JSON file.

The *get()* method takes three parameters: the path to the data source, the function to retrieve the data, and the data type to be retrieved.

The path to the data source to be opened will be built using the retrieved querystring value. We simply take a partial JSON file name in string literal format and concatenate the querystring value to it, as shown below. This gives us a target filename to open and extract data from. So that if the querystring is *hotel.html?id=**1**,* we should end up with a filename of data/*hotel**1**.json*. This file name corresponds to one of the JSON files in our data folder.

```
if(page === 'hotelpage') {
  var thisPage = "#" + page;
  var thisUrl = $(location).attr('search');
  var thisId = thisUrl.split("=")[1];
  $.get("data/hotel" + thisId +".json", function(result, status) {
   . . .
  }, "json");
}
```

Building the Page
To process and display our data we will use the anonymous function that is the second parameter of the *get* method. This function will first retrieve a hotel *name* value from a relevant JSON file. This name value will be appended to the <h2> on *hotel.html*. The function will then append the id of the hotel to the <div> with the id *hotelid* (This will be used when we are adding hotels to our list of favourites). Finally, it will build the HTML that will contain the hotel details, and append this to the <div> with the id *contentArea* on *hotel.html*.

```
$.get("data/hotel" + thisId +".json", function(result, status) {
  $("#hoteltitle",thisPage).text(result.name);
  $("#hotelid", thisPage).html(thisId);
  var hotel =
  "<p>" + result.stars + "</p><p>" + result.description + "</p>";
  $("#contentArea", thisPage).html(hotel);
  }, "json");
```

Save *showhotel.js*. Make sure you reference it in ALL your HTML pages. Upload your work to the DCSIS server. Run the application. By clicking links to hotels in the listview on *#five*, you should now be able to view dynamically created hotel data on *hotel.html*. If your application is not working, make sure you use a JavaScript console to identify and fix any errors in your code.

CSS

When your application is working, you will notice that the hotel id is currently displaying on the page. The id serves no visual purpose and should be hidden. To hide it create a new declaration in *styles.css* and set the *display* value of *#hotelid* to *none*.