Mobile Application Development

Session 6 – Activities

The final product(s) for this session can be found at:
http://www.dcs.bbk.ac.uk/lo/mad/madexamples/session6/classactivities/zedlandhotels/zedland-hotels.html

1. Simple Form (login.html)

Our first form will be a simple login form with two simple text boxes and a submit button.

Create a new HTML page. Save the page as *login.html*. Make sure you add the following:

- links to the CDNs of jQuery and jQuery Mobile
- <meta> tag for viewport
- <script> and <link> tags for any scripts and/or CSS files we have used so far in the project (e.g. *hotelmap.js* and *hotelmap.css*)

Note that we need to include the script and link tags for all JavaScript and CSS files used in an application on every HTML page. This is due to the jQuery Mobile unique architecture. In this architecture, only the <head> section of the first page loaded is used in the creation of the DOM. As a user may bookmark a page and/or enter the site by a sub page, then we need to make sure all the scripts are available for every page. Full explanation.

Link to *login.html* from the application homepage and sub-pages from the *login* link in the footer navbar menu.

In *login.html*, create a new jQM page in the <body> section. Give the page an id of *login*.

Now create a header with a link back to the homepage, and a <h1>

with a heading of *login*. Also, create a footer. In the footer, include the app navbar with links to *about*, *contact us* and *login* (as in previous pages).

In the content section of the new page, create a new form using a <form> tag. Give the new form an id of *frm1*, a method of *post*, and an action of *booking.html*.

```
<form action="booking.html" method="post" id="frm1">
</form>
```

Note that in jQM all form elements must be given a unique id: not just in the same page, but across all pages. Full explanation.

Inside *frm1*, create three new <div> tags. Give each of these tags a *data-role* property with a value of *fieldcontain*. The tags will serve as containers for our form controls.

For accessibility purposes, we will give labels to the form controls. However, we will not display those labels in order to preserve screen real estate. Instead, we will use placeholder text inside the controls themselves. To hide the labels, we will use the jQM class *ui-hidden-accessible*. To create the placeholder text, we will use the HTML5 *placeholder* attribute. To make sure that our controls are displayed appropriately in smaller screen mobile devices, we will use the jQM *data-mini* property and set it to *true*.

The submit button in the form will be associated with the jQM stylesheet using the *ui-btn* class. By default, the button will display as block, and fill the width of the screen.

```
<form action="booking.html" method="post" id="frm1">
  <div data-role="fieldcontain">
    <label for="user" class="ui-hidden-accessible">User Name:</label>
    <input type="text" name="user" id="user" data-mini="true" →
    placeholder="Username"   required>
  </div>
  <div data-role="fieldcontain">
    <label for="pass" class="ui-hidden-accessible">Password:</label>
    <input type="password" name="pass" id="pass" data-mini="true" →
    placeholder="Password" required>
  </div>
  <div data-role="fieldcontain">
    <input type="submit" class="ui-btn" value="Submit" data-mini="true">
  </div>
</form>
```

To provide some basic validation, we could add the HTML 5 *required* attribute to each text input field. Note however, that this method of validation has notable limitations. Primarily, it would allow us to enter single characters, or numbers in name fields. Obviously, in a real-world application, we would require something more thorough than this, and to achieve it, we would probably use a jQuery validation plug-in.

Save your work and upload it to the DCSIS server. View it in your mobile device. Note that certain features may not work if you try to view your files locally, or in an emulator.

2. Complex Form (booking.html)

Our second, somewhat more complex, form will be the application booking form.

Create a new external page for the booking form following the guidelines from activity 1 above (e.g. <script> and <link> tags for site assets in the <head>, header and footer navigation, etc.). Give the jQM page an id of *booking*. Save the document as *booking.html*.

Create a new form in the content section. The form should have the following attributes and values:

id	frm2
action	bookingresults.html
method	post

This booking form will contain the following controls.

Label	Type	id	other attributes
First name	text	firstname	data-mini="true", required
Surname	text	surname	data-mini="true", required
Email	email	email	data-mini="true", required
From	date	from	data-mini="true", required
To	date	to	data-mini="true", required
No. of Persons	select (with four options: 1, 2, 3, 4)	persons	data-mini="true"
Breakfast? Dinner?	checkboxes	dinner breakfast	name=”breakfast” data-mini="true" name=”dinner” data-mini="true"
NA	submit	value=”submit”, class="ui-btn", data-inline="true", data-mini="true"	

Add the controls making sure that each control apart from the breakfast and dinner checkboxes are wrapped inside a <div> tag with a *data-role* of *fieldcontain*. For example:

```
<div data-role="fieldcontain">
 <label for="first">First Name:</label>
 <input type="text" id="first" data-mini="true" required>
</div>
```

Wrap the breakfast and dinner checkboxes inside a <div> with a data-role property set to *controlgroup*.

Finally, create the submission page for the booking form. This should be a simple external jQuery mobile page with a simple confirmation message. Save this page as *bookingresults.html*.

Save your work and upload it to the DCSIS server. View it in your mobile device. Note that certain features will not work if you try to view your files locally.

3. My Hotels Feature using Local Storage

In this activity, we will create a My Hotels feature. This feature will allow users to bookmark hotels while they are searching for them. It could also be used to bookmark favourite hotels that users have stayed at. To store details of bookmarked hotels, we will use HTML 5 local storage.

In summary, this part of the application will work as follows. A user will navigate to a hotel page. She will then have the option to click a button to add the hotel to her myhotels list. The user will be able to access her list of hotels from a link located on the home page.

Completed example.

Adding Items to Local Storage

HTML

The first thing we have do is to create the ability to add a hotel to the my hotels list in local storage.

Open *churchill-hotel.html*. In this page, add the following code in the content section after the <ul> containing the hotel details.

```
<a href="#" class="ui-btn" id="addbtn">Favourites</a>
<div id="nostorage"></div>
```

The first part of this code is the button that the user will click to add the hotel. The second part is an empty container that we will use to display a status message should the user's browser not support local storage.

We will need to make one final change in the HTML. When we add a hotel to local storage, we will add both its name and the URL of its page. Both of these can be derived from the hotel title on the hotel page. We will need, therefore, to access the hotel title from the script that will add hotels to local storage. To enable access, add an id of *hoteltitle* to the <h2> heading on the *churchill-hotel.html* page.

```
<h2 id="hoteltitle">Churchill Hotel</h2>
```

jQuery

The next task is to create the script that will add hotels to local storage. The script will be written in jQuery.

Create a new JavaScript file and save it as *addhotel.js*. Add a <script> tag for this file in the head of all the HTML pages in your application.

In *addhotel.js*, start by adding a jQM *pagecreate* event. This event will run each time a jQM page is created.

```
$(document).on("pagecreate", function() {
 . . .
});
```

Note that this event is not associated with any particular page in the application. The reason for this is that multiple (hotel) pages will include the my hotels functionality in the completed application. For the time-being, we are only including the functionality in one page.

Next, we need to capture the click event of *addbtn*, and set up a function to run when the button is clicked.

```
$(document).on("pagecreate", function() {
   $( "#addbtn" ).click(function() {
   });
});
```

This function will check if local storage is supported. If local storage is supported, the function will start the chain of events that will convert the hotel title into a partial URL, and save the title and URL to local storage. If local storage is not supported, it will display a *no storage* message to the user.

```
$( "#addbtn" ).click(function() {
      if (typeof(Storage) != "undefined") {
          setDetails(getTitle(), getUrl());
      } else {
        $("#nostorage").text("Local storage not supported");
      }
    });
```

The *setDetails() function* in the code above takes two arguments. Both arguments are in the form of calls to helper functions. The first helper function is *getTitle().* This function gets the hotel title by accessing the <h2> with the id *hoteltitle* on *churchill-hotel.html*. The second helper function is *getUrl()*. This function again gets the hotel title from *churchill-hotel.html*, only this time it converts it into a URL format by using a regular expression to replace spaces with dashes and remove capital letters.

```
function getTitle() {
  var title = $("#hoteltitle").text();
  return title;
}

function getUrl() {
  var title = $("#hoteltitle").text();
  var url = title.replace(/\s+/g, '-').toLowerCase();
  return url;
}
```

The *setDetails()* function then places the hotel title and partial URL inside a JavaScript object, and writes that object into local storage using *JSON.stringify*.

```
function setDetails(title, url) {
  var hotel = {name: title, hotelurl: url};
  localStorage.setItem("hotels",JSON.stringify(hotel));
}
```

The script is now complete. Save your work, and upload it to the Titan server. View it in your mobile device. Note that local storage will not work if you are viewing your files locally.

To check whether your code is working, and hotels are being added into local storage, you can use the Chrome or Firefox developer tools. In Chrome, go to *Tools/Developer Tools* and click the *Resources* tab. In the left column, click *local storage*. You should be able to see a list of domains and associated stored items.

Getting Items from Local Storage

The next stage is to create the feature that will allow the user to view any hotel he has saved into local storage. To do this we will need to do three things:

1. Create an external page (*myhotels.html*) to display saved hotels
2. Create a script to retrieve the hotels from local storage
3. Create a link from the homepage to the *myhotels.html* page.

HTML

Create *myhotels.html*. Create a new jQuery Mobile page inside the <body> section. Give the page an id of *myhotels*. Then create the requisite header, content and footer sections using the jQuery Mobile syntax. In the header, add the application title (Zedland Hotels) as an <h1>. Also create a link back to the homepage to the right of the <h1> heading. Make sure the link is displayed as a button (data-icon="home"). Finally, in the content section, create a <h2> with the text *My Hotels*. Also, create and <div> tag with an id of *hotel*, and a <div> tag with an id of *nostorage*.

```
<div data-role="page" id="myhotels">
  <div data-role="header">
    <a href="zedland-hotels.html" data-icon="home">Home</a>
    <h1>Zedland Hotels</h1>
  </div>
  <div data-role="content">
    <h2>Favourites</h2>
    <div id="hotel"></div>
    <div id="nostorage"></div>
  </div>
  <div data-role="footer" data-position="fixed">
    //navbar
  </div>
</div>
```

<div id="hotel"></div> is where we will display our hotel data.
<div id="nostorage"></div> is where we will display a status message to the user if local storage is not supported.

Next, add a link from the main navigation menu in *zedland-hotels.html* to *myhotels.html*.

JQuery
Create a new JavaScript file. Name the file *gethotel.js*. Add a link to the new file to all the pages in your application.

gethotel.js will do three things:

1. Check local storage is available
2. Get data from local storage
3. Display the data from local storage in my*hotels.html*.

my*hotels.html* will be dynamically populated with links to hotel pages. We will therefore need to use an event that triggers each time my*hotels.html* is shown, rather than just once when it is loaded or created. To this end, we will use the *pagecontainerbeforeshow* event.

Note that transition (show) events should always be used for dynamically populated pages.

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

The *pagecontainerbeforeshow* event works somewhat differently than other events we have seen so far in the module. This event, like all pagecontainer events is triggered on a page container (e.g. a document body) rather than an individual page. To associate the event with a particular page, we first need to programmatically identify that page. This can be done in a number of different ways. In our example, it is done in using the *toPage* method of the *ui* (user interface) object. If the *toPage* method returns *#myhotels*, then, and only then, will we run our script to check for local storage, and retrieve and display the hotel data.

```
$(document).on("pagecontainerbeforeshow", function (e, ui) {
  var page = ui.toPage[0].id;
      if( page == 'myhotels' ) {
    //Check for local storage
  }
});
```

Once again, if local storage is not available we will display a status message to the user. If local storage is available, we will call a function, *displayHotelDetails()*, that will get the hotel data from local storage and display it on *myhotels.html*. *displayHotelDetails()* takes a single argument in the form of the function *getHotelDetails()*. *getHotelDetails()* is responsible for retrieving hotel data from local storage.

```
$(document).on("pagecontainerbeforeshow", function (e, ui) {
  var page = ui.toPage[0].id;
      if( page == 'myhotels' ) {
        if (typeof(Storage) != "undefined") {
          displayHotelDetails(getHotelDetails());
        } else {
          $("#nostorage").text("Local storage not supported");
        }
      }
});
```

The g*etHotelDetails()* function uses *JSON.parse* to change the stored string format back to a JavaScript object. It assigns the JavaScript object to a variable, and returns that variable for use in *displayHotelDetails()*.

```
function getHotelDetails() {
  var hotelDetails =
  JSON.parse(localStorage.getItem('hotels'));
  return hotelDetails;
}
```

The *displayHotelDetails()* function takes the data passed to it by g*etHotelDetails()*, and uses it to build the HTML that will be displayed on the *myhotels.html* page.

```
function displayHotelDetails(details) {
  var hotel="<a href='" +
  details.hotelurl +
  ".html' class='ui-btn ui-corner-all ui-icon-arrow-r ui-btn-icon-right'>"+
  details.name + "</a>";
  $("#hotel").html(hotel);
}
```

Save your work and upload it to the DCSIS server. View it in your mobile device.