Comp 422 - Software Development for Wireless and Mobile Devices

- Semester: Fall 2016
- Dr Nick Hayward

An overview of jQuery Mobile

A brief overview and introduction to the jQuery Mobile UI library. Further documentation can be found at the following URL,

- jQuery Mobile API - https://api.jquerymobile.com/

Further project details can be found at the following URL,

- jQuery Mobile - https://jquerymobile.com/

Contents

Intro

jQuery mobile was designed as an off-shoot of the popular jQuery JavaScript library with specific focus upon mobile devices, specifications, and requirements. Its goal is to leverage HTML5 and CSS3 for simple, fast, and responsive development of mobile centric user interfaces. It has been designed with touch in mind, and responds to user generated events as expected for touch-enabled devices.

The key aspect of this library, and its particular relevance to Cordova development, is its support for cross-platform devices and operating systems. In effect, design once, and then deploy to all supported devices and browsers.

Browser support for the latest 1.4 version release is excellent, and further details can be found at the following URL

- jQuery Mobile 1.4 Browser Support - https://jQuerymobile.com/browser-support/1.4/

Customise UI design

With jQuery Mobile, we can customise our design, either manually or with **ThemeRoller** on the project's website.

- jQuery ThemeRoller - http://themeroller.jquerymobile.com/

We can also customise the JavaScript build to ensure we only download the required components for our given application. This builder is available at the following URL,

- jQuery Download Builder - http://jQuerymobile.com/download-builder/

In essence, anyone familiar with jQuery will feel at home with jQuery Mobile. It is based on jQuery, and has baked-in support for accessibility and universal access. jQuery claims that the project follows,

> > ...progressive enhancement and responsive web design (RWD) - http://demos.jquerymobile.com/1.4.5/rwd/ principles.

As noted, HTML5 is the technology underpinning this library, thereby making it easy to design and develop for applications. However, they also include a powerful API, which the project claims inherently makes is easy to customise and extend the library. Their API can be found at the following URL,

- jQuery Mobile API - http://api.jquerymobile.com/

jQuery Mobile guide

jQuery Mobile presents many different options for design and customising the UI of your mobile applications. We can use it as a component of responsive design, or add it as a baked-in component of a native mobile app. We can often consider jQuery Mobile as a mixture of the following,

- pages and the concept of dialogs
- general navigation
- working with content
- theming and UI design

Pages

Use of pages is a fundamental concept for structuring, rendering, and using an application. We can make them simple, including just a **basicpage** wrapper, or detailed and layered within many pages grouping content together within a **single multi-page** template. As an additional option for rendering this content, we can also consider pages as **native dialogs**, thereby offering the user a quick, responsive view in a styled modal overlay.

Then, we can further enhance and manipulate such rendering options with Ajax navigation and animated transitions. By default, jQuery Mobile offers a simple, yet powerful, Ajax-based system of navigation. In essence, it works by intercepting a standard anchor or link request, and instead processing it as an Ajax request.

This Ajax navigation supports the standard back button in browsers, thereby not resetting or breaking the application's navigation. It also supports a concept of pre-fetching pages into the DOM, and the ability to cache one, a few, or all pages within an application for quick, responsive rendering. As you might expect, if there is any perceptible delay in the return, this Ajax navigation will use a standard loader overlay to help inform the user in a timely manner.

As a requested page is loaded, it is effectively looking for just the content in the page, which is then inserted into the DOM. Any widgets in the incoming page will be updated to match defined styles and behaviour, and then the remainder of the page is ignored. It will not be included in the updated DOM of our page. However, the title of the page will be updated to reflect the loading and rendering of the new page into an active view.

As it's JS, and jQuery as well, we can apply animations and transitions for this loading and view. By default, jQuery Mobile sets this transition to fade, but we can modify this to another effect or simply remove. For example, we can use transitions such as fade, pop, flip, turn, slide, and so on.

We can also add standard HTML elements to our app's code. We can add headings, paragraphs, divs, links, images, and so on. We can use these elements to create our own custom layouts and designs, and style as required for our app's aesthetics. We simply add an additional stylesheet to the page's `<head>` element after the jQuery mobile stylesheet.

However, by default jQuery Mobile includes existing touch-friendly widgets for use within our UI. We can add forms, collapsible elements and sets, various popups and dialogs, responsive table designs, list views, and so on. We can also customise our JS code to ensure we only load and use those widgets and UI elements necessary for our application. Again, another simple way to speed up our application.

basic page structure

As we might expect with a HTML5 based framework, we need to define our pages with the standard HTML5 doctype,

```
<!DOCTYPE html>
```

This allows us to exploit and manipulate many of the new content categories available within HTML5. We can also add correct semantic organisation of our content, using the new `header`, `main`, and `footer` elements, for example.

Then, we need to define the `<head>` element, and associated metadata. For example, we can define the viewport for our devices, thereby customising for mobile devices.

The viewport tag basically allows us to set the default zoom level for page rendering, and its associated dimensions. If we don't set the value, most mobile web browsers will simply set a best guess, standardised width of approximately 900 pixels. This is normally set to help the page render well in both mobile and desktop browsers, which is, of course, little use for Cordova.

Therefore, by setting this meta tag to

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

we get a width that is set to the pixel width of the device's screen. Thankfully, this does not affect a user's ability to zoom a page.

Of course, we also need to add references to the jQuery Mobile libraries with the standard CSS `<link>` and JS `<script>` elements.

basic page template - initial page

```
<!DOCTYPE html>
<html>
<head>
    <title>Basic App</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/[version]/
  jquery.mobile-[version].min.css" />
    <script src="http://code.jquery.com/jquery-[version].min.js"></script>
    <script src="http://code.jquery.com/mobile/[version]/
  jquery.mobile-[version].min.js"></script>
</head>

<body>
    ...app content...
</body>
</html>
```

Identify a page within the `<body>` using an element such as a `<div>`, and a suitable identifying attribute `data-role="page"`.

```
<div data-role="page">
...
</div>
```

For jQuery mobile, there is a standard, defined pattern we can use for our app's pages.

Basic jQuery Mobile pattern for such page designs includes `div` elements with additional, defined `data-role` and `role` attributes to define semantic division of content. For example,

- `<div data-role="header">...</div>`
- `<div role="main" class="ui-content">...</div>`
- `<div data-role="footer"></div>`

This is quasi-HTML5, mimicking the semantic structure and elements for standard HTML5. In effect, the jQuery mobile attributes are replacing the standard names for HTML5 semantic elements.

An initial, sample page template for jQuery mobile is as follows,

```
<!DOCTYPE html>
<html>
<head>
    <title>Basic App</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="http://code.jquery.com/mobile/[version]/
  jquery.mobile-[version].min.css" />
    <script src="http://code.jquery.com/jquery-[version].min.js"></script>
    <script src="http://code.jquery.com/mobile/[version]/
  jquery.mobile-[version].min.js"></script>
</head>
<body>
  <!-- initial page -->
  <div data-role="page">
   <!-- header -->
     <div data-role="header">
    <h3>Welcome</h3>
     </div>
   <!-- main content -->
     <div role="main" class="ui-content">
    <p>first basic app content...</p>
     </div>
   <!-- footer -->
     <div data-role="footer">
    <h5>footer...</h5>
     </div>
  </div>
</body>
</html>
```

basic page template - multi-pages

To help identify each **page**, we add an `id` attribute with a unique value. These `id` attributes can then be used to easily create internal links between our so-called **pages**.

For example, a page block might be as follows,

```
<div data-role="page" id="page1">
...
</div>
```

which we can then reference as follows

```
<a href="#page1">page 1</a>
```

In effect, it's just like working with internal anchors from page to page. A template for multi-pages might be as follows,

```
<body>
  <!-- page1 -->
  <div data-role="page" id="page1">
    <div data-role="header">
      <h3>Welcome - page 1</h3>
    </div><!-- /header -->
    <div role="main" class="ui-content">
          <p>View internal page - <a href="#page2">page2</a></p>
    </div><!-- /content -->
    <div data-role="footer">
      <h5>footer - page 1</h5>
    </div><!-- /footer -->
  </div><!-- /page1 -->

  <!-- page2 -->
  <div data-role="page" id="page2">
    <div data-role="header">
      <h3>page 2</h3>
    </div><!-- /header -->
    <div role="main" class="ui-content">
          <p><a href="#page1">Return to page1</a></p>
    </div><!-- /content -->
    <div data-role="footer">
      <h5>footer - page 2</h5>
    </div><!-- /footer -->
  </div><!-- /page2 -->
</body>
```

basic page template - pre-fetching pages &c.

jQuery Mobile gives us the option to pre-fetch pages for use within our applications.

With this option, we are effectively telling jQuery Mobile to add the requested pages or resources to the DOM, thereby making this content instantly available if and when a user requests that page's link.

We can pre-fetch a page by adding the `data-prefetch` attribute to the required page link. We set the value for this attribute to a boolean value, either true or false. This tells jQuery Mobile to load this specified page within the current DOM, and after the current page has loaded. This content is fetched after the `pagecreate` event has been triggered by the current page.

```
View internal page - <a href="#page2" data-prefetch="true">page2</a>
```

We can also use this same concept within our JS code, thereby programmatically calling the required page's content as needed, independent of a given link. For example,

```
$( ":mobile-pagecontainer" ).pagecontainer( "load", pageUrl, { showLoadMsg: f
```

We're using the **Pagecontainer** widget's method, `load`, to fetch an external page, and then insert it into the DOM. We can also add some extra options to this fetched content, change how it's output, and so on. For example,

```
$( ":mobile-pagecontainer" ).pagecontainer( "load", "confirm.html", { role: "
```

In this example, we are fetching a page `confirm.html`, and then setting its output, its role, to a dialog box.

So, our selector is the default `:mobile-pagecontainer`, which equates to the default `<body>` element in HTML. It is the parent element for jQuery Mobile pages, both internal and external examples. We can also customise this page wrapper from `mobile-pagecontainer` to another preferred element. In essence, we can change the page's wrapper from the default `body` to a custom element elsewhere in the page.

Further details can be found in the

- API documentation - http://api.jquerymobile.com/pagecontainer/#method-load

basic page template - caching the DOM (multi-page apps)

One of the underlying issues with maintaining the DOM, and continually adding more and more pages, is that we can quickly fill a browser's memory. This can subsequently lead to the browser either dramatically slowing down or potentially crashing due to a lack of system resources.

jQuery Mobile has a simple way to handle this potential issue. As we load a page using Ajax, jQuery mobile adds a note that this new page needs to be removed from the DOM as soon as a user leaves or navigates away. Effectively, this clearing of the DOM cache forms part of the page events, and will be triggered as the given page is hidden.

Naturally, it doesn't clear the home page for the app. This also means that a single page template with many page containers will not be affected either. It simply remains in the cache by default.

If a user then tries to return to a cleared page, for example one we've just removed from the DOM cache, the browser will try its cache for an existing copy of the page. If it has, indeed, been removed then the browser will simply re-fetch the file from the server.

We can also customise such settings by using jQuery Mobile's option to programmatically cache pages, which then become instantly available if a user decides to return to a given page.

We can also set an attribute on a single page to ensure that it is cached and ready for use.

```
data-dom-cache="true"
```

Concept of dialogs

```
<a href="#page2" data-transition="pop">page2</a>
```

So, any page can be used to create these modal dialogs by adding an attribute for dialog,

```
<div data-role="page" data-dialog="true" id="page2">
```

By setting this attribute, we are informing jQuery Mobile that we would also like any appropriate styles and behaviours adding to this given page. For example, by default it adds styled corners, margins, sets the background properties of the underlying page, and adds any additional buttons or controls to help the user.

basic dialog template

```
<body>
<!-- page1 -->
<div data-role="page" id="page1">
  <div data-role="header">
    <h3>Welcome - page 1</h3>
  </div><!-- /header -->
  <div role="main" class="ui-content">
    <p>View internal page - <a href="#page2">page2</a></p>
  </div><!-- /content -->
  <div data-role="footer">
    <h5>footer - page 1</h5>
  </div><!-- /footer -->
</div><!-- /page1 -->

<!-- page2 -->
<div data-role="page" data-dialog="true" id="page2">
  <div data-role="header">
    <h3>page 2</h3>
  </div><!-- /header -->
  <div role="main" class="ui-content">
    <p><a href="#page1">Return to page1</a></p>
  </div><!-- /content -->
  <div data-role="footer">
    <h5>footer - page 2</h5>
  </div><!-- /footer -->
</div><!-- /page2 -->
</body>
```

The transition for this dialog will use the default **fade** option. We can also explicitly define a preferred transition effect, e.g.

```
<a href="#page2" data-transition="flow">page2</a>
```

This transition effect can be specified for accessing the dialog, and then exiting the dialog as well. e.g.

```
<a href="#page1" data-transition="flow">Return to page1</a>
```

So, an example template might look as follows,

```
<body>
    <!-- page1 -->
    <div data-role="page" id="page1">
        <div data-role="header">
        <h3>Welcome - page 1</h3>
        </div><!-- /header -->
      <div role="main" class="ui-content">
            <p>View internal page -
        <a href="#page2" data-transition="flow">page2</a>
      </p>
        </div><!-- /content -->
        <div data-role="footer">
        <h5>footer - page 1</h5>
        </div><!-- /footer -->
    </div><!-- /page1 -->
    <!-- page2 -->
    <div data-role="page" data-dialog="true" id="page2">
        <div data-role="header">
        <h3>page 2</h3>
        </div><!-- /header -->
        <div role="main" class="ui-content">
            <p>
        <a href="#page1" data-transition="flow">Return to page1</a>
      </p>
        </div><!-- /content -->
        <div data-role="footer">
        <h5>footer - page 2</h5>
        </div><!-- /footer -->
    </div><!-- /page2 -->
</body>
```

basic dialog template - close dialog

When we close an open modal dialog, the default is to return to the current page within the app. However, for a 'close' link we can also be explicit in the redirect reference/location. We might set the close link to the previous page, another link, and so on.

n.b. jQuery Mobile looks for a defined header element in the page. If present, a close button will be added to the left side of the header in the modal dialog. We can, of course, change the position of the button by adding the following attribute to the dialog container,

```
data-close-btn="right"
```

We can also remove this default button,

```
data-close-btn="none"
```

However, if we do need or want to add a **cancel** button, for example, jQuery Mobile offers the following attribute,

```
data-rel="back"
```

One of the nice features of using `back` is that it preserves the originating transition. It is not necessary to specify a transition for the cancel link.

We can also customise the text for our close button either programmatically or by simply setting an attribute and value pairing. The button itself will still be rendered as an icon, but the additional text will be readable by screen readers and other accessible devices. For example,

```
data-close-btn-text="closed text"
```

We can also chain dialogs from one to another, and so on. This chaining also works in reverse. One of the notable features of this chaining is that it will always navigate to the previous dialog until the originating root of

```
data-role="page"
```

is once more reached. This mechanism simply ensures logical, consistent navigation between dialogs within an app's navigation.

An example template might look as follows,

```
<body>
    <!-- page1 -->
    <div data-role="page" id="page1">
        <div data-role="header">
        <h3>Welcome - page 1</h3>
        </div><!-- /header -->
      <div role="main" class="ui-content">
            <p>View internal page -
        <a href="#page2" data-transition="slidedown">page2</a>
      </p>
        </div><!-- /content -->
        <div data-role="footer">
        <h5>footer - page 1</h5>
        </div><!-- /footer -->
    </div><!-- /page1 -->

    <!-- page2 -->
    <div data-role="page" data-dialog="true" id="page2">
        <div data-role="header">
        <h3>page 2</h3>
        </div><!-- /header -->
        <div role="main" class="ui-content">
            <p>
        <a href="#page1" data-rel="back">Cancel</a>
      </p>
        </div><!-- /content -->
        <div data-role="footer">
        <h5>footer - page 2</h5>
        </div><!-- /footer -->
    </div><!-- /page2 -->
</body>
```

basic dialog template - style dialog

As with other page containers and elements, we can customise the style and theming for our dialogs. We can set an attribute on the dialog's page container,

```
data-theme="b"
```

which then allows us to specify theme attributes to the header, content or footer containers. By default, jQuery Mobile include two default style themes, **a** or **b**. Basically, these come down to dark text on light background, and the reverse.

We can also change the design of the corners, which are rounded by default, or add a custom overlay for the dialog. It's this custom overlay swatch that really allows us to tailor our dialog designs as needed. We can manage this custom design with standard CSS, and set this using a class on the originating page container for the dialog.

For example,

```html
...
<!-- page2 -->
<div data-role="page" data-dialog="true" data-theme="b" id="page2">
  <div data-role="header">
    <h3>page 2</h3>
  </div><!-- /header -->
  <div role="main" class="ui-content">
    <p><a href="#page1" data-rel="back">Cancel</a></p>
  </div><!-- /content -->
  <div data-role="footer">
    <h5>footer - page 2</h5>
  </div><!-- /footer -->
</div><!-- /page2 -->
...
```

We can, naturally, also modify the actual size and dimensions of our dialog box. By default, it is set to a width of `92.5%` with a `max-width: 500px` of 500 pixels. They've also added a default top margin of 10%, and so on.

Therefore, the default CSS is as follows,

```css
.ui-dialog-contain {
    width: 92.5%;
    max-width: 500px;
    margin: 10% auto 15px auto;
    padding: 0;
    position: relative;
    top: -15px;
}
```

We can simply add the above ruleset to our CSS stylesheet and then override as required per design.

Navigation

For the purposes of mobile development, jQuery Mobile's navigation support thankfully follows an **asynchronous** pattern.

So, navigation in jQuery mobile, for example, is based upon loading pages into the DOM using AJAX. This will modify the page's content, and then re-render for display to the user. It will also include a set of aesthetically pleasing, and useful, animations to help inform the user of changes in state, and therefore appropriate updates in the content.

This navigation system effectively hijacks a link within a page's content container, and then routes it through an AJAX request. The benefit for developers is a particularly useful, and almost painless, approach to asynchronous navigation. Most of the time, we are not even aware of this updated request. In spite of hijacking the link request, it is still able to support standard concepts such as **anchors** and use of the **back** button without breaking the coherence and logic of the application.

Therefore, jQuery Mobile is able to load and view groups of disparate content in pages within our initial home document. In essence, the **many** combining to form the **one** coherent application.

Its support for core JavaScript event handling, in particular for URL fragment identifiers with `hashchange` and `popstate`, allows the application to persist, at least temporarily, a record of user navigation and paths through the content. We can also tap into this internal history of the application, and again hijack certain patterns to help us better inform the user about state changes, different paths, content, and so on.

example navigation

The following demo is an example of using the jQuery Mobile standard navigate method, `$.mobile.navigate`, which is used as a convenient way to track history and navigation events.

With this simple example, we can set our record information for the link, effectively any useful information for the link or affected change in state. We can then log the available direction for navigation, in this example the fact we can go **back**, the url for the nav state, and any available hash. In our example, the simple appended hash in the url, `#nav1`.

What this allows us to do is keep a clear record of user traversal through the application, log it as required by given state changes, and then use it to inform our application's logic, our user, and so on.

Using widgets

Within our app's webview container, we can add standard HTML elements for any required content containers.

For example, standard HTML and HTML5 elements such as `p`, `headings`, `lists`, `sections`, and so on.

Thankfully, we don't necessarily have to build the whole application from scratch.

jQuery Mobile includes a wide-range of pre-fabricated widgets we can add to our applications. These are naturally touch-friendly, and include collapsible elements, forms, responsive tables, dialogs, and many more.

using widgets - listview

A good example of this type of pre-fabricated widget is a **listview**.

jQuery Mobile helps us style, render and then manipulate standard data output and collections, including rendering of lists as interactive, animated views.

These lists are coded with a `data-role` attribute, as we saw earlier with a page value for a data role

```
data-role="listview"
```

This allows us to style and render our lists with additional options such as a dynamic search filter.

using widgets - listview - example

```html
<!-- listview example -->
<div>
  <ul data-role="listview">
    <li>Cannes</li>
    <li>Marseille</li>
    <li><a href="#page3" data-transition="slide">Monaco</a></li>
    <li>Nice</li>
  </ul>
</div>
```

- simple listview with slide transition

```html
<!-- page3 -->
<div data-role="page" id="page3">
  <div data-role="header">
    <h3>page 3</h3>
  </div><!-- /header -->
  <div role="main" class="ui-content">
    <p><a data-rel="back" class="ui-btn">Return</a></p>
  <section class="image-view">
    <img src="media/images/monaco1.jpg">
  <section>
  </div><!-- /content -->
  <div data-role="footer">
    <h5>footer - page 3</h5>
  </div><!-- /footer -->
```

```
</div><!-- /page3 -->
```

using widgets - listview - add filter

A **listview** can be a lot of fun, and very useful for easily organising our data.

However, we can also use a **listview** to add filtering and live search options to our lists.

We set a simple client-side filter by adding an attribute for `data-filter`, and then set the value to `true`

```
data-filter="true"
```

jQuery Mobile will then add a search query input field to the top of our list widget, and set a filter for the entered search query. Effectively, it's performing a pattern match using a partially entered string against strings in a designated list. It can match partial fragments of a list item, and then dynamically filter our list content.

We can also set some default, helpful text for the input field. Our way of prompting the user to interact with, and therefore use this feature correctly.

```
data-filter-placeholder="Search Cities"
```

To tidy up the presentation of our list, we can also add an inset using the attribute

```
data-inset="true"
```

using widgets - listview - adding some formatted content

One of the fun aspects of working with a framework such as jQuery Mobile, and others such as the excellent Ionic framework, is the simple way we can organise and format our data presentations and views.

For example, if we have a grouped dataset, we can still present it using lists. However, we can also add informative headings, links to different categories within this dataset, and simple styling to help differentiate components within the list interface.

Therefore, we structure the list as normal, with sub-headings, paragraphs, and so on. Then, jQuery Mobile gives us a simple option for setting certain list content as an aside. For example,

```
<p class="ui-li-aside">1 image</p>
```

There are many similar tweaks and additions we can add to help improve organisation and rendering of list data. Further details can, of course, be found in the jQuery Mobile

API.

using widgets - listview - example 2

```html
<ul data-role="listview" data-inset="true">
  <li data-role="list-divider" role="heading">French Cities</li>
  <li>
    <a href="#page3" data-transition="slide">
      <h3>Monaco</h3>
      <p><strong>Principality of Monaco</strong></p>
      <p>Monaco is a sovereign city-state...</p>
      <p class="ui-li-aside"><strong>1</strong> image</p>
    </a>
  </li>
  <li>
    <a href="#">
      <h3>Nice</h3>
      <p>Located in the south of France...</p>
    </a>
  </li>
</ul>
```