Developing Google Reader client for Series 40 with Web Tools

This article explains how NewsFlow for Series 40 was developed with Web Tools. The article also covers how you can use 3rd party RESTful web services from your Web App and how to create compelling user interfaces with jQuery Mobile

Note: This is an entry in the Asha Touch Competition 2012Q3

Warning: JQuery is not fully supported on the browser; developers must select components very carefully to ensure only supported elements are used. Note that an app using unsupported elements may run on *Local Preview* but will fail on *Cloud Preview* (and on a real device).

This article needs to be updated: If you found this article useful, please fix the problems below then delete the {{ArticleNeedsUpdate}} template from the article to remove this warning.

Reasons: hamishwillee (11 Sep 2012)
This code does not run in Cloud Preview due to unsupported elements in JQuery (e.g. `data_role` does not work). It should be updated to use supported elements.

Introduction

NewsFlow is Google Reader client where user can read aggregated news feed with simple user interface. NewsFlow was originally an application written in Qt Quick which runs on MeeGo (N9), Symbian and Maemo devices. As Qt Quick also uses JavaScript it was easy to convert the whole application to full HTML5 web app which runs also nicely on Nokia Asha devices. NewsFlow uses Google Reader APIs so you can learn how to create RESTful API calls to 3rd party services.

Applications highlights the following features of Series 40 Web Tools:

- Password manager

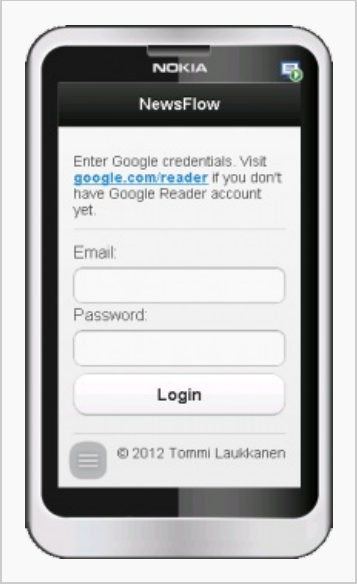
Application also shows on how to create user interface easily with jQuery Mobile, including:

- Pages with navigation
- Dynamic listview based on JSON data
- Displaying AJAX loader

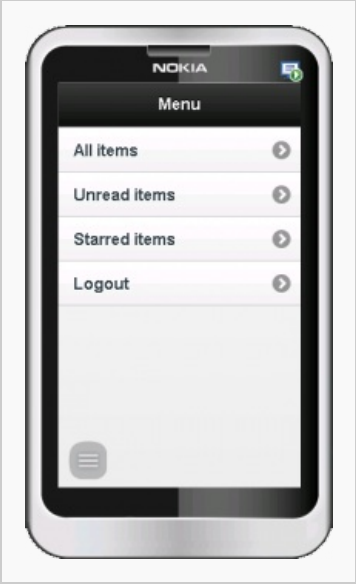
Application UI

The user interface is created with jQuery Mobile. Different views as modeled as Pages and navigation between pages is done part scripted and partly automated.

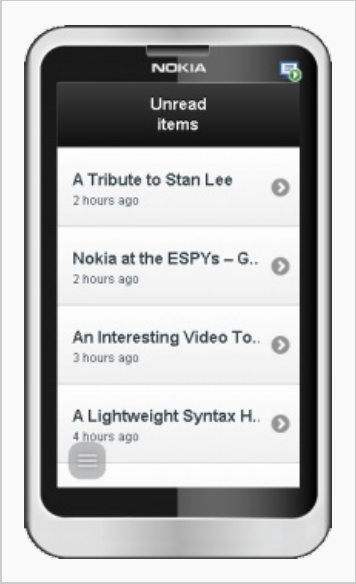
(C) Copyright Nokia 2014. All rights reserved.



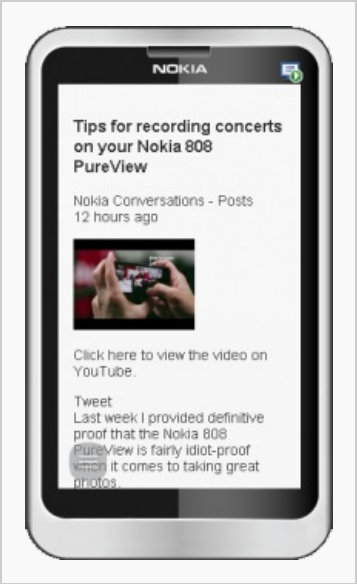
Login page



Menu page



Item list



Details page

The HTML part for the app is less than 80 lines. This **index.html** contains all four app pages:

```
<!DOCTYPE html>
<html>
 <head>
 <title>NewsFlow</title>
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="stylesheet" href="css/jquery.mobile-1.1.1.min.css" />
 <script src="js/jquery-1.7.2.min.js"></script>
 <script src="js/jquery.mobile-1.1.1.min.js"></script>
 <script src="js/reader.js"></script>
 <script src="js/storage.js"></script>
 <script src="js/common-global.js"></script>
 <script src="js/script.js"></script>
</head>
<body>

<div data-role="page" id="loginpage">
 <div data-role="header"><h1>NewsFlow</h1></div>
 <div data-role="content">
```


(C) Copyright Nokia 2014. All rights reserved.

```
  <div data-role="fieldcontain" class="ui-field-contain ui-body ui-br">
    Enter Google credentials. Visit
    <a href="http://www.google.com/reader">google.com/reader</a> if
    you don't have Google Reader account yet.
  </div>
  <div data-role="fieldcontain">
    <label for="email" class="ui-input-text">Email:</label>
   <input type="text" name="email" id="email" value="">
    <label for="password" class="ui-input-text">Password:</label>
   <input type="password" name="password" id="password" value="">
    <button id="loginbutton" data-role="button">Login</button>
  </div>
  <div data-role="fieldcontain" class="ui-field-contain ui-body ui-br" style="text-
align:right;">
    &copy; 2012 Tommi Laukkanen
  </div>
 </div><!-- /content -->
</div><!-- /page -->

<div data-role="page" id="menu">
 <div data-role="header"><h1>Menu</h1></div>
 <div data-role="content">
  <ul data-role="listview">
   <li><a id="allitemsbutton">All items</a></li>
   <li><a id="unreaditemsbutton">Unread items</a></li>
   <li><a id="starreditemsbutton">Starred items</a></li>
   <li><a id="logout-button" href="#loginpage" data-transition="slide">Logout</a></li>
  </ul>
 </div>
</div> <!-- /menu page -->

<div data-role="page" id="itemlistpage">
 <div data-role="header"><h1 style="white-space:normal;">Items</h1></div>
 <div data-role="content">
  <ul id="itemlist" data-role="listview">
   <li>Loading...</li>
  </ul>
  <button data-role="button" id="loadmorebutton">More</button>
  <a data-role="button" href="#menu">Back</a>
 </div>
</div> <!-- /item list page -->

<div data-role="page" id="detailspage">
 <div data-role="content">
  <h3 id="itemtitle" style="white-space:normal;">Loading...</h3>
  <div id="itemsource"></div>
  <div id="itempublished"></div>
  <div id="itemauthor"></div>
  <div id="itemcontent"></div>
  <button id="nextitembutton" data-role="button">Next</button>
  <a id="backtolistbutton" data-role="button" href="#itemlistpage" data-
transition="slide">Back</a>
 </div>
</div> <!-- /item details page -->

</body>
</html>
```


(C) Copyright Nokia 2014. All rights reserved.

There are few items that are good to highlight here:

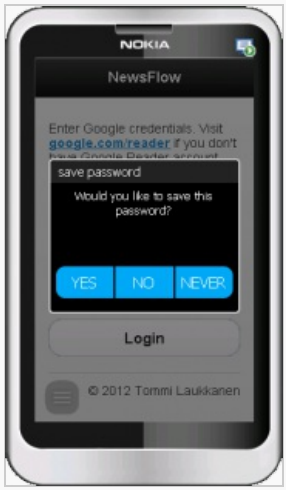
- By using the `data-role` attributes the jQuery Mobile handles the styling really well and you don't have write much additional CSS at all
- Basic navigation can be easily automated without JavaScript using the `href="#pageid"` references as shown for example in back button in details view:

```
<a data-role="button" href="#itemlistpage" data-transition="slide">Back</a>
```

When button is pressed the view changes to `#itemlistpage` with sliding transition animation.

Login and passwords

Small tip: Web Apps support automated password manager so you don't have to write code which would store the username or password into HTML5 storage.



Password manager in action

API Calls

API calls to Google Reader API are executed in **reader.js** JavaScript file. I usually write a single method that will take care of the HTTP requests along with authentication etc. In *NewsFlow* app the `doWebRequest()` function looks like this:

```
function doWebRequest(method, url, params, callback) {
    var doc = new XMLHttpRequest();
    $.mobile.showPageLoadingMsg();
    doc.onreadystatechange = function() {
        if (doc.readyState == XMLHttpRequest.HEADERS_RECEIVED) {
            var status = doc.status;
            if(status!=200) {
                showError("Google API returned " + status + " " + doc.statusText);
            }
        } else if (doc.readyState == XMLHttpRequest.DONE) {
         $.mobile.hidePageLoadingMsg();
            var data;
            var contentType = doc.getResponseHeader("Content-Type");
            data = doc.responseText;
            callback(data);
        }
    };
```


(C) Copyright Nokia 2014. All rights reserved.

```
    doc.open(method, url);
    doc.setRequestHeader("Cache-Control", "no-cache");
    doc.setRequestHeader("Pragma", "no-cache");
    if(sid.length>0) {
        doc.setRequestHeader("Authorization", "GoogleLogin auth=" + sid);
        doc.setRequestHeader("Cookie", "SID=" + sidToken);
    }
    if(params.length>0) {
        doc.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        doc.setRequestHeader("Content-Length", String(params.length));
        doc.send(params);
    } else {
        doc.send();
    }
}
```

The function uses `XMLHttpRequest` to execute the HTTP requests. jQuery Mobile is used here to show loading indicator when request starts:

```
$.mobile.showPageLoadingMsg();
```

...and also hiding the same indicator when request `readyState` is DONE:

```
$.mobile.hidePageLoadingMsg();
```

Authentication is handled with "Authorization" header as described in Google Data API documentation.

When authentication is complete and we have user's SID token then API calls are made simply as:

```
    itemsURL = "http://www.google.com/reader/api/0/stream/contents/user/-
/state/com.google/reading-list?xt=user/-/state/com.google/read";
    doWebRequest("GET", itemsURL, "", parseNews);

...

// Callback method which handles the HTTP response and parses the JSON data
function parseNews(data) {
    var doc = JSON.parse(data);
    for(var i in doc.items) {
        var item = doc.items[i];
        parseEntry(item); // Parse news entry to object and append to items array
    }
}
```

HTTP requests are executed in asynchronous fashion so you'll need to pass the callback function as parameter. Callback function is called when HTTP response is done and response data is passed to function as parameter. Usually 3rd party web services provide the data also in JSON format which is easy to parse in JavaScript with `JSON.parse(..)` command.

Dynamic list views

News items are loaded from the API and parsed from JSON to objects in a JavaScript array. When array is ready we'll redraw the list by looping through the model and appending the object to list as new list item elements. In the following code sample you also see that it is required to refresh the jQuery Mobile listview plugin so that it behaves as expected. When list is generated the click event function is binded to the added list items.

(C) Copyright Nokia 2014. All rights reserved.

```
function drawItems() {
 $("#itemlist").empty();
 for(var i in model) {
  var item = model[i];
  $("#itemlist").append("<li><a class='itembutton' href='#detailspage' id='item" + i +
"'><h3 class='ui-li-header'>" + item.title + "</h3><p class='ui-li-desc'>" +
item.published + "</p></a></li>");
 }
 $("#itemlist").listview("refresh");
 $(".itembutton").click(function(){
  var itemid = $(this).attr("id");
  console.log("Showing item " + itemid);
  var itemindex = itemid.replace("item", "");
  var item = model[itemindex];
  $("#itemtitle").text(item.title);
  $("#itempublished").text(item.published);
  $("#itemsource").text(item.source);
  $("#itemcontent").html(item.desc);
 });
}
```

Last words

Application is released as Open Source and is continually developed so actual source code in Nokia Project site can differ from snippets shown here. If you are interested in the application and how application improves over time then follow the project here.

(C) Copyright Nokia 2014. All rights reserved.