Developing a prototype library WebApp for mobile devices

This item was submitted to Loughborough University’s Institutional Repository by the/an author.

Citation: COOPER, J. and BREWERTON, G., 2013. Developing a prototype library WebApp for mobile devices. Ariadne, 71.

Additional Information:

- This article was published in Ariadne [© UKOLN] and is also available at: http://www.ariadne.ac.uk/

Metadata Record: https://dspace.lboro.ac.uk/2134/12832

Version: Published

Publisher: © UKOLN

Please cite the published version.
This item was submitted to Loughborough’s Institutional Repository (https://dspace.lboro.ac.uk/) by the author and is made available under the following Creative Commons Licence conditions.

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
Developing a Prototype Library WebApp for Mobile Devices

Jason Cooper and Gary Brewerton describe the development of a prototype WebApp to improve access to Library systems at Loughborough University for mobile devices.

Introduction

Reviewing Loughborough University Library’s Web site statistics over a 12-month period (October 2011 – September 2012) showed a monthly average of 1,200 visits via mobile devices (eg smart phones and tablet computers). These visits account for 4% of the total monthly average visits; but plotting the percentage of visits per month from such mobile devices demonstrated over the period a steady increase, rising from 2% to 8%. These figures were supported by comparison with statistics from the Library’s blog, where, over the same period, there was also a steady increase in the percentage of visits from mobile devices. This increase was on a smaller scale than the Web site, rising from 0.5% up to 4%.

Having identified this increase in the usage of mobile devices, it was decided to investigate ways to support mobile access more effectively. As part of this investigation, the Library's Systems Team undertook the development of a prototype mobile app.

Deciding on the Prototype's Features

The first task undertaken was to produce a list of functionality that could be included in the Library WebApp. The list was based upon current Library services and consisted of the following:

- Support library catalogue searching
- Display opening hours (pulled from Library web-site so data can be maintained in one location)
- Display current item loans, requests and holds
  - Indicate overdue items
  - Indicate recalled items
  - Offer option to renew loaned items
  - Offer option to cancel requests for items
- Reading lists
  - Ensure module list displays all modules for which the user is registered
  - Should handle multiple levels of reading lists
  - Include thumbnails
  - Include library holding information
- Display current room/PC bookings
  - Display list of bookings including resource name, start time and end time for each booking.
  - Offer option to cancel a room/PC booking
  - Offer option to make a room/PC booking
- Display upcoming library events (pulled from Library web-site)
  - Include both upcoming workshops and events
- Display library news (taken as a feed from our Library blog)
- Offer feedback option

After reviewing this list, it was decided to leave out the searching of the Library Catalogue feature as the Library's discovery tool (Ex Libris's Primo [1]) was scheduled for a number of updates that would improve the support of mobile devices. Therefore it was decided to wait and see how the improved mobile interface performed before deciding how to integrate it into the mobile app.

Additionally it was decided not to implement a number of the other features, those that would either require new APIs to be created for other systems or those that would alter the information stored in the other systems. These features would be carried forward for implementation in a future version of the mobile app. Consequently features excluded from the pilot version were:

- library catalogue searching
- the option to renew loaned items and cancel requested items
- the option to make or cancel a room/PC booking

WebApp versus Native Apps

An important early decision was whether to create the Mobile App as a WebApp or as a number of native apps? A native app is one that is developed in the native language for the platform (Objective-C for iPhone/iPad devices, Java for Android devices, etc) and usually delivered via an app-store (iTunes for Apple, Google Play for Android, etc). A WebApp is developed in HTML5 and JavaScript, being delivered to the mobile device via the World Wide Web.

There are pros and cons to developing a mobile app as a native app or as a WebApp. Native apps have full access to a mobile device's resources but need to be developed as a separate app for each platform on which they are to be made available. Conversely developing a mobile app as a WebApp restricts the resources that can be accessed to those available to the device's Web browser, although a single developed WebApp can work on multiple platforms.

There are also pros and cons with the different publishing methods of native apps and WebApps. To be available to the majority of devices using a specific platform, native apps need to be made available through that platform’s app-store. The process to get an app included in an app-store varies from app-store to app-store. It usually involves an additional delay, during which the app must be checked to ensure it does not infringe the app-store's terms and conditions. However, one advantage of publishing through a platform's app-store is that it is the first place a user will look to see if an app is available.

WebApps are a lot easier to publish as they are simply hosted on a Web server which users just access via their browser and then add a link to the WebApp on their home screen/desktop. As there is no app-store submission procedure to navigate, there is no additional delay between finishing a version of the WebApp and releasing it. More
complex arrangements are required to advertise a WebApp, otherwise potential users may fail to realise that it has actually become available.

There is a third hybrid path available which involves the use of Adobe PhoneGap [2]/Apache Cordova [3] to convert a WebApp into a native app. This method gives you a lot of the advantages of WebApp development (e.g. one set of WebApp files that will be available on multiple platforms) while also allowing you to access more of the device's resources than a pure WebApp would. The resulting native apps produced via PhoneGap/Cordova can usually be made available by the relevant platform app-stores.

It was decided to develop the Library mobile app as a WebApp as there was no need to access any of the device's resources other than those available in the Web browser. Moreover, developing a single WebApp would require less time and fewer resources.

Technologies

The Library mobile app was developed in HTML5 [4] and makes use of the jQuery Mobile framework [5] to provide the majority of the user interface. A mobile app using the jQuery Mobile framework consists of one or more pages. Each page can have its own content including a header and/or footer.

A page may also have a number of pop-up container types attached to it. These are hidden till they are triggered, at which point they pop-up and present their content to the user. Pop-ups can be positioned either over specific pieces of content or in the centre of the window.

Both pages and pop-ups can be populated with standard HTML content, jQuery Mobile widgets or a combination of the two. The jQuery Mobile widget types are styled to give a consistent feel to the WebApp; these styles can be overridden by either attaching specific attributes to the widget or through customising the CSS. The widgets used by the mobile app are list views, collapsible content blocks and buttons. The list view widget produces a list in a format that is more suitable for use on a mobile device than traditional HTML lists.

Collapsible content blocks provide a header bar with an attached container for other content. When the user clicks/taps the block's title bar, the attached container will be collapsed or expanded, depending on its current state. This state can also be pre-defined as expanded or collapsed when creating the widget.

jQuery Mobile automatically converts anchor elements in the content of a page to its button widget. These are a lot easier for mobile device users to select than a traditional HTML link and again can be styled through either customising CSS or by defining a number of attributes on the source anchor element.

jQuery Mobile is built on top of the jQuery framework [6] and as such this is also required by the mobile app. Due to it already being present and its AJAX [7] functionality being very powerful, it was decided to use jQuery for communicating with the mobile app's back-end APIs.

Back-end APIs

The active functionality for the mobile app is provided through a set of back-end APIs. These APIs are Perl CGI scripts hosted on a standard Web server and accessed
via a RESTful interface. Their output is returned in XML [8], JSON [9] or JSONP [10] format, depending on the information being returned.

Most of the back-end APIs call APIs for other Library systems and then process and filter the results before returning them to the mobile app.

The server used to host the APIs is an existing Web server that already hosts a number of other small Library projects. Almost all the APIs require a valid security hash that identifies and authenticates the user. As the security hash has to be passed in most API calls, all the APIs are accessed over HTTPS connections to secure them. Each security hash is only valid for a period of time that is specified on the server.

**The Pilot Mobile App**

As most features of the mobile app are personalised, the first APIs required by the mobile app were those to log users in and to obtain their details. If the mobile app does not hold a current security hash, it presents users with a login form requesting their username and password. These details are then passed to the login API for authentication. The login API authenticates the users’ credentials against the University's Active Directory and, if they are valid, returns a security hash.

Once the mobile app has a valid security hash it is used to call the getUserDetails API which returns the following information about the user:

- name
- email address
- department
- school
- modules on which they are registered

In addition, it also generates a new security hash and returns that to the mobile app. As the security hash is refreshed every time the mobile app is started, users will only have to login again if they do not use the app before their current security hash expires.

Having obtained users’ details, the mobile app presents users with its starting page, which consists of a menu with the following options:

- opening hours
- my loans
- my reading lists
- my bookings
- my librarian
- events
- library news

Beneath this menu there is also a feedback button which allows users to send feedback to the Library.
Selecting the Opening Hours option presents the user with a number of collapsible tables. Each table contains the opening hours for one week, starting with the current week. By default, only the first table is expanded, and the rest remain collapsed; this makes it obvious to the users which week is the current one.

The opening hours are loaded dynamically from a file on the server which is regularly generated from opening hours data contained in the content management system hosting the Library's Web site. The opening hours data are processed and converted into JSON format then saved into a file on the mobile app's Web server. The resulting file is ideally suited to being loaded dynamically by the mobile app.
My Loans, the next option on the main menu displays three collapsible tables (Loans, Holds and Requests). They are populated with data retrieved from the getLoans API. The getLoans API accesses the user's current loans, holds and requests details directly from the Library Management System (LMS), which in this case is Aleph [11] from ExLibris. The requests to the LMS are made via its X-Server protocol. The getLoans API returns the following details for each loan currently held by the user:

- title
- authors
- due date and time
- if it has been recalled
- if it is overdue

The getLoans API returns just the title and authors for items being held or that have been requested, since they have no due date, recall status or overdue status.

As well as populating these tables the mobile app also populates a counter in the title bar of each collapsible table showing the number of entries contained. These counters allow the mobile app to collapse the holds and requests tables by default, but still warn the user when there is content in there.

The third option in the main menu, My Reading Lists, uses the module information for the user that was retrieved when the mobile app was started. The module information is used to retrieve the reading lists attached to those modules from the University's reading list management system (RLMS) via its APIs. The University uses the Loughborough Online Reading List System (LORLS) [12] as its RLMS which, since its redevelopment in version 6 [13], makes a very strong set of APIs available.
The mobile app presents users with a list of their reading lists along with module codes and names of tutors. If there is no reading list available for a module the module code is annotated with the message ‘No reading list for module’.

When the user selects an available reading list, the content of that list is loaded from the RLMS and displayed. Each entry in the reading list is shown in a similar format to that used by the standard RLMS Web interface, so that the user sees the information in a consistent way.

In a reading list any entry the user can select will either be a link to another reading list or an actual item. If it is a link to another reading list, the mobile app will repeat the process to load and display the new reading list. If it is an item then a new page is shown with more details about the item, including metadata, library holdings and a cover image, provided one is available from Google Books [14].

![Reading list item page](image)

When the user selects the My Bookings option from the main menu a table lists their current and upcoming bookings of library resources (Rooms, PC, etc.). Each booking entry shows the resource booked along with the start and end dates and times of the booking. The booking information is retrieved from the Library's resource booking system (WUBS) [15] using an API that had been developed previously for extracting a specific user's future bookings.

The last personalised feature is the My Librarian feature, where the user is presented with the details for the Library contact responsible for their school/department. The mobile app obtains a list of Library contacts from an existing system that is used to provide Library contact information to a number of online systems. Each entry in the list returned contains the contacts’ name, job title, email address, URL of an image and, if relevant, the school for which they are responsible.

The mobile app uses this information to select the Academic Librarian who is responsible for that user’s school or department and present his or her details at the top of the page. If there is no Academic Librarian so designated, then a predefined fall-back option is used, currently the Library's Customer Services Manager.
Underneath the main contact's details is a collapsed list of all other Library contacts which users can expand to look through if they are looking for a specific Academic Librarian to contact.

The Events and Library News options in the main menu operate in the same way with the information of these being obtained from RSS feeds [16]. The Events information is taken from an RSS feed on the Library Web site and the Library news from an RSS feed on the Library's blog. As RSS feeds are in a consistent XML format the mobile app just needs one function to process them and populate a collapsible list with their content. If the RSS entry has a link attached, then the entry in the list will link through to that in case the user wishes to find out more details.

When the feedback option is selected by the user it presents a form in a pop-up. The form contains two text fields for the user to complete, subject and message. There is also a button that, when selected, will pass the subject and message along with the user’s name and email address to the feedback API. The feedback API uses this information to create and send an email to a predefined email address.

With the exception of the main menu of the mobile app, all pages have a back button located in the left-hand side of the header bar. This provides the user with a consistent means of returning to the previous page.

**Benefits of HTML5**

Two key features of HTML5 for WebApp development are caching and local storage. Local storage allows the mobile app to store data locally in the browser in key/value pairs which can only be accessed by the Web site that created it. Local storage is similar in many ways to HTTP cookies except they can store larger quantities of data and are not sent to the server with the HTTP requests. They can also only store primitive JavaScript data types and arrays. To store more complex objects in local storage, the mobile app has first to convert them into a JSON string, and, when retrieving them, it has to parse that JSON string to recreate the object.

Every time the mobile app retrieves and processes data from an API, a copy of the resulting value/object is stored in local storage. If a call to an API fails, then the mobile app checks to see if there is a suitable entry in the local storage and, if so, it retrieves the value/object from local storage and uses that. This provides users with access to the majority of their data even when they do not have an Internet connection.

The caching aspect of HTML lets the mobile app specify which resources should be cached locally by the browser and which URLs should never be cached. When a browser loads a WebApp which specifies a cache manifest file, the browser will download it and compare it to the previous version it has cached. If the contents of the two files differ, then the browser will use the latest version and reload its cache by downloading all the listed cacheable resources. From then on, when the browser is asked to retrieve a resource, it will check to see if it has a cached version and, if so, will use that.

During the development it was discovered that trying to cache too much data would cause problems for some browsers and devices due to upper limits of the quantity of data they can cache. Caching HTML, JavaScript, Style Sheets and icon files worked well, but trying to cache larger images quickly filled up the cache.
Development Timescale

The time taken actually to produce the pilot version of the mobile app was approximately two weeks and the development was undertaken by a single developer, working around his usual tasks. The quick development time was mainly thanks to the initial choices made in the project. Developing a single WebApp instead of multiple native apps significantly reduced the time and breadth of knowledge required. Moreover, the decision to limit the initial content for the mobile app to that which was easily available via existing APIs removed the need to develop APIs for other systems.

The developer was already skilled in Web development and had a very good understanding of the programming languages involved. Using jQuery Mobile allowed him to concentrate on the content of the mobile app and still feel confident that the final look and feel would be suitable for a mobile app. Furthermore, the use of jQuery Mobile allowed local branding to be easily applied through the mobile app's style sheet.

Piloting the Mobile App and Future Enhancements

Given the plethora of mobile devices available, it was impractical to test the mobile app on a majority of devices. Instead, testing took place on a small number of devices covering the supported platforms (Android and Apple IOS). This testing showed the mobile app to work in the standard browsers used on the Android and Apple platforms. Thereafter, the mobile app was made available to users in a pilot phase to discover any device-specific bugs.

The pilot version of the mobile app was made available to members of the University during November 2012. Over the next few weeks, 65 people made use of the app and a number of feedback emails were received from users. The main content of these emails were requests for additional functionality rather than bug reports. The lack of device-specific bugs has been attributed to the use of the jQuery and jQuery Mobile frameworks, as they already take a lot of device-specific discrepancies into account.

After reviewing the new feature requests a list of functionality for future releases of the mobile app was drawn up. The features in development for the next version of the mobile app are:

- Renewing loans
- Cancelling loan requests
- Booking resources
- Cancelling booked resources

There are already APIs available for renewing loans and cancelling loan requests in the local LMS which should allow easy addition of those functions to the mobile app. On the other hand, the making and cancelling bookings functions will require more in-depth development as there are currently no APIs for these operations in the local resource-booking system.
Conclusion

The jQuery Mobile framework with the jQuery framework is a very powerful and effective tool for developing WebApps and, provided suitable APIs are available, can be used to produce a usable mobile app in a very short period of time.

Also by using existing resources, the development of a mobile app can be achieved with very little financial outlay.

References

2. PhoneGap http://phonegap.com/
5. jQuery Mobile http://jquerymobile.com/
10. JSON-P: Safer cross-domain Ajax with JSON-P/JSONP http://json-p.org/
12. LORLS: Loughborough Online Reading List System http://blog.lboro.ac.uk/lorls
16. RSS Advisory Board. RSS 2.0 Specification http://www.rssboard.org/rss-specification