Ohmage – Progress Report 1 By Zorayr Khalapyan

**Table of Contents**

1. Overview 1
   1. Milestone 0 – Basic PhoneGap Integration 1
   2. Milestone 1 – Basic Form Filling 2
2. General Workflow Diagram 3
3. MWF JS Decorators 4
4. JavaScript Class Diagram 4

-- Last Updatd Wednesday, November 23, 2011 --

1. **Overview**

Please see section 1.1 and 1.2 for a general progress reports on milestones 0 and 1. The first milestone concentrates more on testing and PhoneGap technologies, while the later one focuses on Ohmage API integration with MWF and the mobile based web.

**1.1 Milestone 0 – Basic PhoneGap Integration**

Milestone 0 acted as a testing period for MWF integration with PhoneGap technologies on iOS and Android platforms. With online-storage and native-linking strategy, in which case the web pages would remain in an online storage server and the native containers would act as a web engine, wrapping the pages, several issues caused unwanted effects.

One of the main issues was visible blank pages on header redirects. So if a page issued redirect to another page that issued another redirect, then the user would have to go back through two blank pages before getting back to the original page. The cause of the problem was within PhoneGap’s internal architecture – every page load would start a new intent of the application without checking HTTP status codes and hence, treat redirect HTTP responses as actual web pages.

Another issue came up with orientation changes. Web pages wrapped within PhoneGap which depended on orientation change event listeners on JavaScript side did not function correctly as the events never triggered. PhoneGap might add additional functionality for uniform support for orientation change events across multiple devices sometimes in the near future.

There were several other issues involving security and user data protection. Given that PG allows access to the user’s address book, photo album etc. a single successful script injection attach could potentially harvest confidential user information.

The conclusion with these tests was that PhoneGap was architected for containing web pages that are locally stored on the phone and does not function well with the online-storage strategy.

**1.2 Milestone 1 – Basic Form Filling**

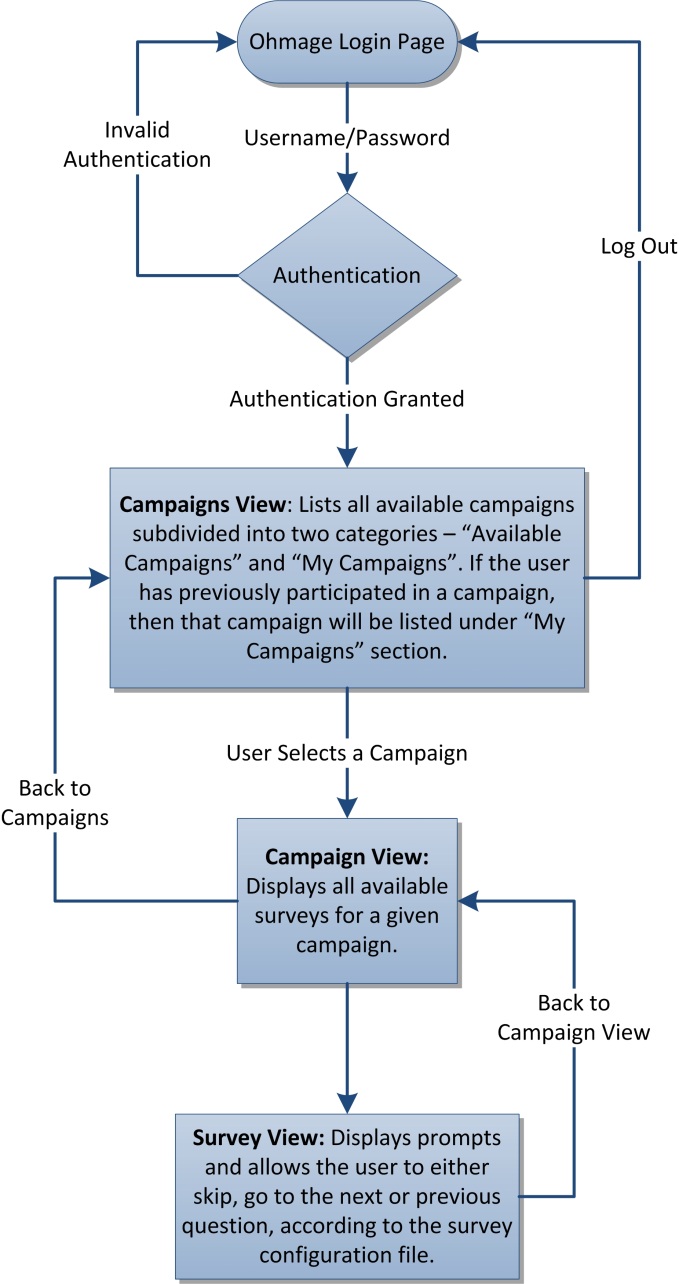
Milestone 1 integrated Ohamge API with MWF based solution. Current support includes user authentication via username and password, display of list of available campaigns, view of campaign surveys, and a survey view that allows users to answer prompts. A JavaScript based API library was created for ‘raw’ interaction with the server. In milestone 2, an extension to this library will be implemented for utilizing local storage as a caching mechanism for efficiency. With this feature, if the user has recently downloaded a configuration file, then the configuration file will be returned from the local storage instead of committing an AJAX request. This type of separation of responsibilities allowed the development to continue without the assumption of HTML5 support.

One of the challenges presented by this milestone was the state-less page loads. In other words, there is no way to pass to a new loading page an already created JavaScript object – hence, for displaying a specific survey within a page, the entire configuration file has to be loaded and parsed again, in an asynchronous fashion. To address this issue, an Event-Driven Object Construction pattern was chosen and implemented. In this pattern, the child object waits for the parent object to be asynchronously constructed and only then instantiates itself. If the loading of the configuration file is done via an AJAX call to the server, then this loading will take a significant amount of time. Nevertheless, if the user navigates to the survey view via the campaign view, which should be the case almost always, then the local storage capability will allow an efficient loading of the configuration file from the local memory.

Given the number of different types of prompts and the complex nature of navigation between different prompts, the mechanism for displaying the prompts and navigation from one prompt to another were put in two separate JavaScript classes. The Prompt class, which is responsible for generating the UI of a given prompt, includes a mapping between prompt type and the generating function – this will allow flexibility in adding new types of prompts in the future.

**2. General Workflow Diagram**

For implementing basic form filling, milestone 1, the below user workflow diagram was designed. The workflow allows easy navigation between campaigns, surveys, and prompts, as well as user authentication. The final version of the application will build upon this workflow to add more functionality such as settings and user information views.



**3.0 MWF JS Decorators**

Current implementation of Mobile Web Framework heavily relies on the server-based architecture of the application. User agent information parsed from the HTTP requests is utilized in order to serve the most compatible CSS and HTML content. Furthermore, MWF also supports PHP decorators which allow easy creation of MWF-styled elements if the pages are fetched from a server.

Moving towards offline support and server-less MWF, the need for greater JavaScript support for MWF was noted and hence, MWF JavaScript decorators were designed to allow simple creation and management of MWF elements. The implemented decorators are not reliant on JQuery and are optimized for MWF. Utilizing these decorators gained around 80% code reduction from the initial version. Current support includes menus, content, buttons, and also titles.

**4.0 JavaScript Class Diagram**

