QlikView

Doctor Finder Extension Demo:

A case study in the use of QlikView extensions on the web

Brian Munz - Last Updated 1/29/2011

QlikView

Contents

Introduction 3

Assumptions 3

Disclaimer 3

Purpose 3

UX Overview 3

Technical Overview 5

HTML & CSS 6

Workbench 6

Extension 7

SQL / AJAX 8

Conclusion 9

QlikView

Introduction

The purpose of these case study was to push the limits of QlikView extensions with a particular focus on use within a web application. The extension was meant to be built for exclusive use on a web site, fully utilizing the capabilities of Javascript. Overall, the purpose of this document is to increase awareness as to what is possible and serve as an inspiration for others to push the limits of the use of extensions on the web. By combining Javascript, QlikView, and the web, the possibilities are endless.

Assumptions

It is assumed that the reader of this document is knowledgeable in the basics of setting up extensions, what they are and how they work. Also, the reader should be familiar with basic QlikView Workbench use and how to add objects to a web page. Lastly, a general and basic understanding of CSS, Javascript, and HTML is needed to understand how they work together.

Disclaimer

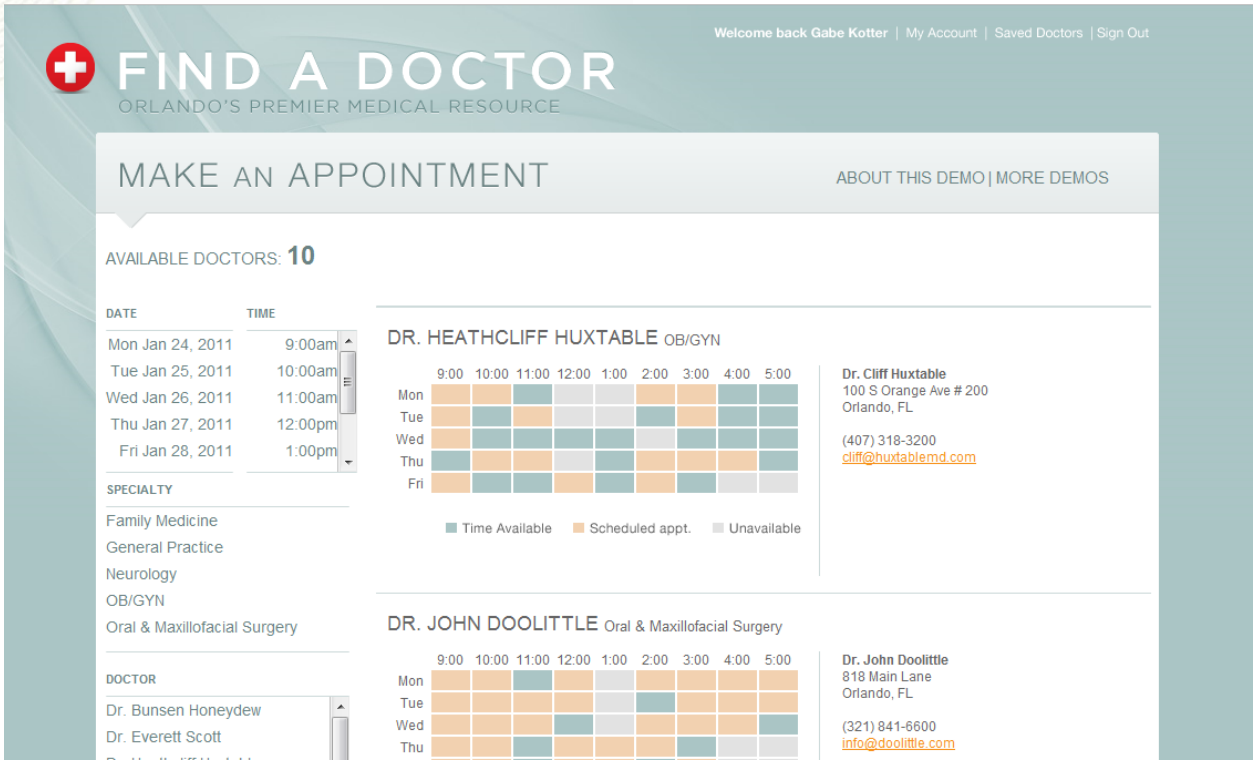
A short disclaimer is needed to point out that this case study is a highly customized solution. The extension that was built was not meant to be a portable and flexible extension that could be easily moved and installed into other QlikView machines and used for the same purpose.

Purpose

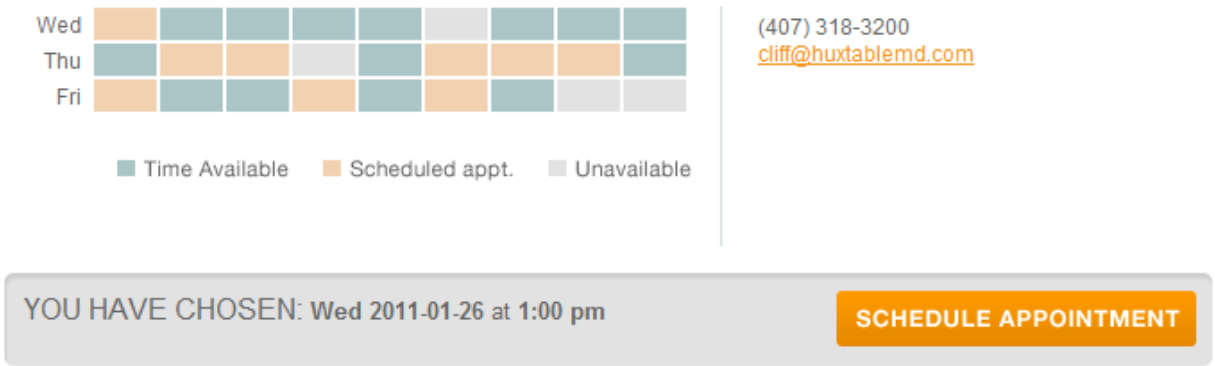
Beyond the technical reasons stated above, the overall purpose of this demo was to create a fully functioning website which seamlessly integrates QlikView on the backend in a less overt way than simply displaying an application in AJAX mode. Along with this, another goal was to use and feature some of QlikView's more webby technologies, namely extensions, workbench, and dynamic update. There will be some technical aspects to this document, but it is more meant to explain how this demo was achieved and increase awareness of the possibilities.

UX Overview

The user experience for this demo is fairly typical on the web for finding and setting an appointment. A general idea of the UI and design created for this demo is below:



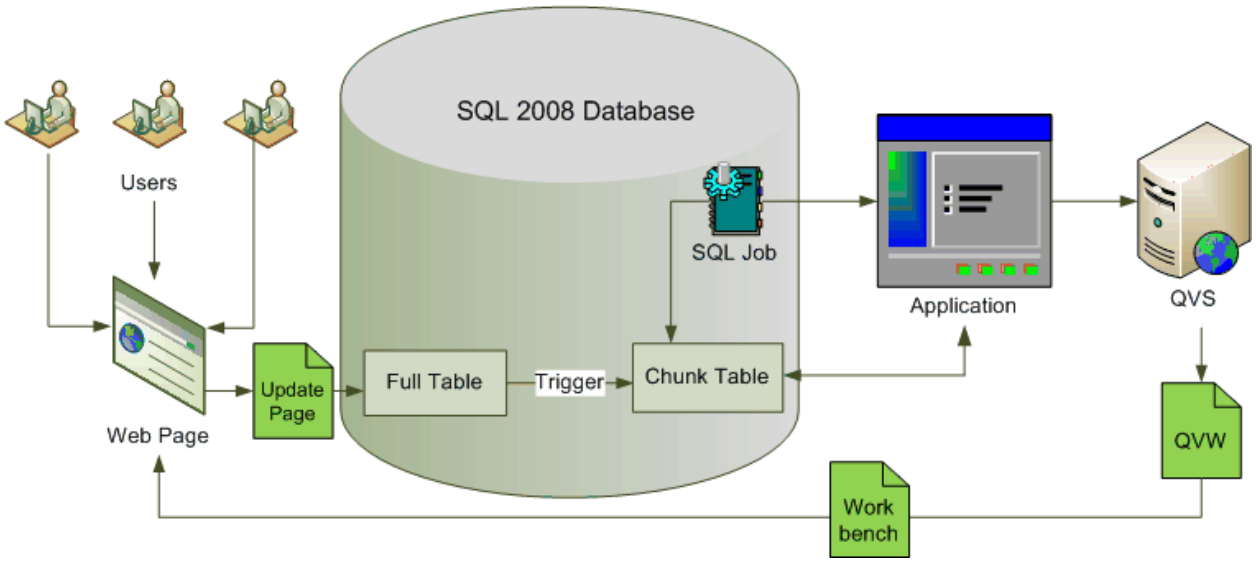
A user accesses the web page, looking for a doctor with whom they want an appointment. After filtering through the various choices based on specialty, availability, etc., the user clicks on an open spot in the doctor's calendar, revealing a confirmation area with a button to click to confirm their choice:



Once the button is clicked, a confirmation message appears, and that time block with then appear as taken on the website.

Technical Overview

The following diagram generally explains the way the web page, QVS, Qlikview, and SQL all function together in this site:



The web page uses Workbench to display the QlikView components on the webpage. These components are styled and modified using javascript (jquery) and CSS with the rest of the page being HTML.

Once the user makes a choice and submits it, another page is called via AJAX which inserts the new data into a SQL database. From here, the data is also updated in QVS which is driving the webpage. The documentation for this has been included in this package as far as how to set up SQL database updates in having them reflected in QlikView in real-time. Since real-time does not automatically refresh QlikView's ajax page, once the user closes out the confirmation box, the page is refreshed and the new data is reflected on the page.

HTML & CSS

Nothing overly extraordinary was needed for this demo as far as HTML markup and CSS is concerned. Most of the trickery here was done using other technologies and the page layout and styling was fairly straightforward.

One thing to note, however, is that all of the CSS stylesheets (as well as JS documents) were stored relative to the webpage rather than the QlikView document. This was done to better emulate a real website and how these things would normally be done. It's true that it is possible to include files from inside an extension, but a more web-friendly way is to include the files from the page root so the styles and functions can be shared within the entire page rather than simply within the extension.

Another word of caution is that the way workbench writes the objects out to the page in AJAX has it containing some fairly deep and complicated table and div structures, which can make targeting the different list boxes and objects difficult sometimes. With some working and trial and error, and by looking through the code used in this demo, it shouldn't be too difficult.

Workbench

In this demo, workbench was used as an easy way to add the QlikView objects to the web page. The styling on the workbench objects was completely done through CSS included on the webpage.

One point of interest is that workbench objects require that a width and height be set on the object itself. This means that for the list of doctors, for example, if you had to set a height on the object, the list could be longer or shorter than its containing box, creating either an unnecessary and awkward amount of white space, or scrollbars for that object box. In general, having multiple scrollbars on a web page is a practice which is avoided on the web for usability purposes, especially in main content areas. It is a poor user experience to give the user the ability to scroll the entire page as well as components on the page with many different scroll bars.

So, this was avoided in this demo with some jquery coding:

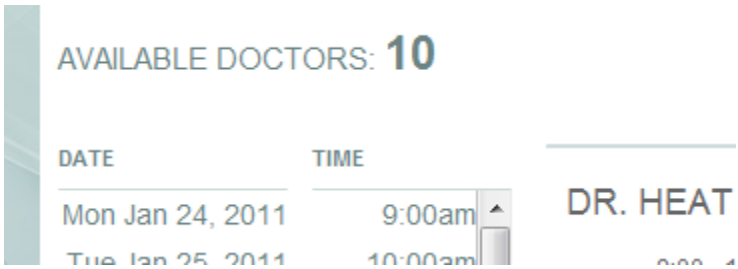
```
$('#results .QvFrame').height((doctorList.length * 260) + 45);
```

All this code really does is take the length of the doctor list (the number of doctors currently being displayed) and multiply it by the height of each of those listings, then set the height of the frame displaying the content to that number. This avoids the need for scrollbars and allows users to scroll up and down the page rather than the object, giving a more webby user experience.

Extension

The extension was the most complicated part of this demo. In order for it to work and look seamlessly with the webpage, it had to communicate with it. This means that there were pieces of Javascript code communicating with the parent page, and thus this extension would fail if opened in QlikView itself. So, essentially this is an extension that is built solely for the web.

An example of this is in the setting of the "Available Doctors" number at the top of the page.



This section of the page actually sites outside of the extension, but the number itself can't be known until the extension is loaded, and if selections are made which change that number, it will need to be updated.

Using jQuery, making this update is relatively simple. Even though the JS file is being loaded from the QlikView extension directory, it is able to target and modify page level elements. So, in this case, the HTML code for the Available doctors number is:

```
<h3>Available Doctors: <span id="docNum"></span></h3>
```

The span which contains the number has an id of "docNum." When the page is first loaded (prior to the extension) this span is empty. In order to add the number of doctors to this span, simple use jquery to update this span from inside the extension like this:

```
$('#docNum').html(doctorList.length);
```

doctorList.length is simply the length of the list that was pulled in from QlikView and that number is being entered into the HTML of the "ducNum" span. Since the script of an extension is run every time any clicks or changes are made in QlikView, the number will be updated.

This demo also customized the data being sent into the extension in order to make the website creation even easier. For example, each doctor has information about his or herself and their practice. In QlikView this information is usually separate. A doctor's address might be comprised of street data, state data, zip code data, etc.

To pull this into an extension, each individual dimension would need to be sent in one by one. This creates a lot more coding inside the extension and make it more of a hassle for the person to set up the extension itself. So, what was done for this demo was to have the aggregation of that data happen within QlikView itself to create a web-facing address. In the Qlikview script all of the different info was put together to create an address similar to this:

Dr. Cliff Huxtable
100 S Orange Ave # 200
Orlando, FL

(407) 318-3200
cliff@huxtablemd.com

That way, only one dimension containing the full address can be fed in rather than each component.

Something to note here, however, is that this information will also need to be styled. So some html code itself (such as line breaks and email links) were added into the dimension also. This allowed for the different lines to be targeted by the stylesheet. Again, this is another good example of QlikView and extensions working together to create a better web-facing solution.

SQL / AJAX

As stated earlier, the dynamic update component is too complex for the scope of this document, but the insertion of the data into SQL was also done from the extension. The insertion itself is a very simple page that any .NET developer would be able to easily create, however, the interesting piece here is that the page was called using ajax, and that it was called from inside the extension.

Once the selection is made and the database needs to be updated, the javascript code is triggered from within the extension to send the data to the insertion page. This makes inserting data into a database from an extension fairly easy in a web solution. Rather than attempting to connect to a database from within an extension, simply use AJAX to send the data to a page that does whatever is needed. This makes it possible to do most web-related tasks while still using an extension to handle the data display and association. This can be a very powerful way to communicate to other web applications from within an extension.

Conclusion

Overall, by just scratching the surface of the code used for this demo, it is clear to see that it was not easy to build. The point here, however, is not to show how flexible and portable extensions can be, but rather how powerful QlikView can be in terms of a web solution. In summary, using QlikView, this demo achieves a web solution without making any concessions. By pushing QlikView and its web components to their limits, professional uncompromising solutions are very achievable.