

ORACLE HTML5 RICH WEB APPLICATION

A Major Qualifying Project Report:

submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by:

Anyansi, Onyedikachi Jeffrey

Lonergan, Ian Patrick

Date: April 23, 2012

Approved:

Professor Gary F. Pollice, Major Advisor

1. HTML5
2. jQuery
3. Web Development

Abstract

When a data-intensive Web application has no connection to the server, usability becomes impossible or, at a minimum, very problematic. No matter how a connection becomes unavailable, using the application or between accesses, users can suddenly find they are unable to get any work done until a connection is reestablished. Oracle™ approached us with this problem, and our project is aimed at providing a possible solution to this issue, allowing a local user to continue their work regardless of the status of the connection. We developed a proof-of-concept framework that would show it is possible to allow developers to build Web applications capable of this.

Acknowledgements

We would like to give thanks to everyone who has guided and assisted us in throughout our project.

First and foremost we wish to thank Gary Pollice, our project advisor, for helping bring the project to fruition at every step of the way and always pointing us to the best resources we needed to accomplish our research and development.

We also give our thanks to Abhijit Kumar, our main contact at Oracle, who set us on the right path for developing a framework capable of satisfying the needs of a real business.

Thanks also go to Oracle for providing us with this opportunity to work on and learn from an excellent and pioneering project.

Table of Contents

Abstract	i
Acknowledgements	ii
List of Illustrations	iii
1. Introduction	1
2. Background	2
2.1 The Growing Mobile Ecosystem	2
2.2 How to build the client	4
2.3 JavaScript Tools and Libraries	10
2.4 Client-side Storage	21
2.4.1 Native solutions -- MySQL™	22
2.4.2 Browser-based Storage	22
3. Methodology	24
3.1 Server and Client Application	24
3.1.1 Requirements	24
3.1.2 Application Proposal	25
3.1.3 Building the Server	26
3.1.4 Building the Client	28
3.2 The Framework	29
3.2.1 Abstracting the Request to the Framework	30
3.2.2 Supplying API for Communication with the Server	31
3.2.3 Queuing Requests	32
3.2.4 Implementing Local Storage	33
3.2.4 Various Bug Fixes and Improvements	34
4. Results and Analysis	35
5. Future Work and Conclusions	36
5.1 Future Work	36
5.2 Conclusion	38
References	40

List of Illustrations

Figure 1 - Mobile Web Usage Growing. from: "Internet Trends" (2012)	3
Figure 2 - Web Storage Browser Support. courtesy: html5rocks.com	24
Figure 3 - Standard Server/Client Representation	26
Figure 4 - Server API.....	27
Figure 5 - DataTables Plugin With Data.....	28
Figure 6 - Editing Data with Jeditable	29
Figure 7 – Specifying the API	32

1. Introduction

For our project, we were approached by Oracle with a problem: data-driven Web applications, when the connection to the server disappears, become unable to function. They requested that we develop a proof-of-concept framework that would allow Web-developers to, when working with a RESTful server application, continue modifying data locally even when the connection to the server is broken. Mobile devices can have issues working when a connection is no longer available, which can occur because of general interference, travelling, and any other situations where access to the Internet is prohibited or impossible.

Solving the issue of being able to continuously modify data when a connection to the server is no longer available, and once connected again, have all local changes reflected to the server data, would allow more rapid development of Web applications that heavily rely on modification of data. As mobile devices become more common, it becomes more important to have ways of dealing with temporary or extended connection downtime. With a framework capable of handling this, the weight of the work would shift off the shoulders of the developer.

We first researched JavaScript frameworks and methods of persistent storage on HTML5 client applications. Once our research was settled, and in order to demonstrate and test our proof-of-concept, we developed a simple Web application as might exist today. The application we eventually settled on developing in concert with our framework was a tool for allowing teachers to update grades and comment on an

assignment. This application was suitable to the needs of Oracle and showing how our framework could work.

We then began building the framework and converted our test application to use it instead of the normal methods of communicating with the server. As the framework grew we implemented more features, ranging from the modest ability to perform multiple types of HTTP requests to fully implementing persistent storage on the client that lasted even if the application was closed before all changes were sent to the server.

We organize this paper as follows. The “Background” section reviews the current state of the field of HTML 5 and Web development. The “Methodology” section discusses our work in developing the simple server and clients, along with our framework. “Results and Analysis” discusses the outcome of the project, our accomplishments, and Oracle’s response to what we developed. In “Future Work and Conclusions” we outline possible future changes to the framework and how it might be implemented, how it might impact Web development, and what we have learned and gained from its development and our studies.

2. Background

In this section, we will describe the research that we did for this project.

2.1 The Growing Mobile Ecosystem

Oracle wanted their application to be supported by all devices. They specifically emphasized the need for the application to be on mobile devices. In 2007, Apple®

released the iPhone®.¹ Shortly after, Google™ released their own smartphone, the Android™. With the advent of smart phones and tablet devices, mobile computing quickly became a new platform for software developers to target. One could now run applications on a phone or tablet device and is no longer confined to a desktop or laptop computer.

This was important for the software development industry. New applications were born in this platform and new companies that specifically targeted tablet devices rose to prominence. The mobile movement is continuing to gain steam and it quickly became clear that mobile computing was here to stay. It is estimated that by 2014 there will be more people connected to the Internet via mobile devices than from desktop computers.²

Mobile Web Usage Growing

Forward Projection: Mobile Web Browsing vs. Desktop Web Browsing
(2007-2015)

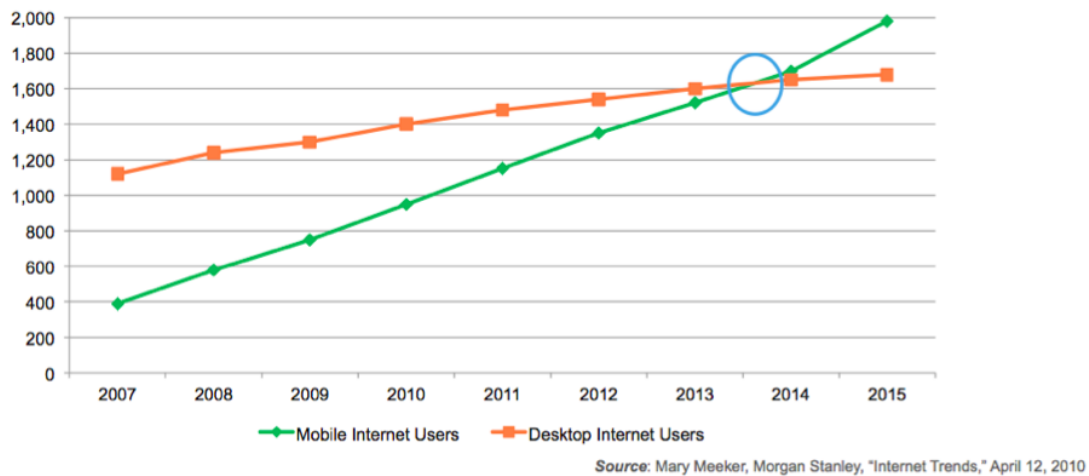


Figure 1 - Mobile Web Usage Growing. from: *Internet Trends*"(2012)

¹ (Honan, 2007)

² (Meeker, Devitt & Wu, 2010)

The mobile movement has shown no signs of slowing down. Companies that were once dedicated to desktop devices, such as Microsoft®, quickly began to push into this new mobile arena. Applications which software mainly ran on Internet-connected desktop devices such as the social network Facebook™ and mapping service Google Maps™ started porting their applications to mobile devices. Just recently, a struggling game company OMGPOP™, who created over 50 games for the desktop-Web, reluctantly started creating mobile games. Their most recent mobile game Draw Something™ was runoff success. A company who was once nearing collapse instantly started bringing in large amounts of revenue daily. OMGPOP quickly got snatched up by game company Zynga™, another large company trying to push into mobile, in a 210 million dollar acquisition.³

As more people are connecting to the Internet via mobile devices, more companies are hopping on the mobile bandwagon and creating applications for the mobile ecosystem. When creating a new application, developers now need to strongly consider whether or not to support the growing mobile platform.

2.2 How to build the client

Because Oracle's application needed to support all these devices, we had to determine the best way to build a client. We considered using Java™. Java is supported by all desktop computers and is the language used to build Android applications. Java's

³ (Cutler, 2012)

guiding principle is “Write once, run anywhere”.⁴ However, one cannot use Java to develop native applications on the iPhone. The iPhone is a major mobile platform, which made it impossible to just ignore it. Due to this, developing a native application seemed like it was a lost cause. There is no single language that you can use to develop native applications on all desktop operating systems as well as mobile operating systems. This is because applications for the iPhone are coded using Objective-C, whereas on Android applications are developed in Java.

Next, we considered building a Web client application that would be run in a browser. Any desktop computer would be able to run the application as long as there was a working browser on the system and most desktop operating systems come with at least one browser installed. Mobile devices come with a browser as well, so building a Web application using JavaScript could be a potential solution. A JavaScript-based application runs through the browser so it would require no extra work to have the application supported by all these various devices.

2.2.1 State of the Art in Web Application Development

Before we could finalize our design decisions, we had to make sure that JavaScript in the browser was capable of being used to build full-featured, high performance applications. When browsers first appeared, their initial function was to render static Web pages retrieved from the Web. Creating dynamic applications was quite more complicated than that.

⁴ (Bacon, Vechev, Cheng, Grove, Hind, Rajan & Yahav. 2005)

The present is quite different. We have been exposed to Facebook, Twitter, and other sites that housed complex Web applications. They proved that building such applications can be done, and that sites no longer need to be a collection of static pages. We delved a bit deeper in order to see how such feats were accomplished.

In April of 2004, Google released Gmail™, a full-featured dynamic email application on the Web⁵. Then in the beginning of 2005, Google released Google Maps, a Web mapping service application. Both of these applications were revolutionary at the time as they ditched the conventional static page-fetching methodology that many other sites were using. Gmail and Google Maps were highly complex and sophisticated applications.

In 2000, a technique called XMLHTTP was used in Microsoft's Outlook Web Access. XMLHTTP was a way of dynamically retrieving XML files over the Internet (HTTP).⁶ Google started to use this technique in Google Maps and Gmail for querying and fetching data from the server in the background, without requiring a page refresh from the browser. It was not until these applications were released in 2005 that the term 'Ajax' (Asynchronous JavaScript and XMLHTTP) was coined to describe this methodology.⁷ With their liberal usage of Ajax, Google popularized this technique.⁸ They showed Web developers that it was possible to build highly dynamic Web applications. This led to a new era of Web applications and now dynamic features can be found on sites all over the Internet.

⁵ (Google, 2004)

⁶ (Hopmann, 2007)

⁷ (Swartz, 2005)

⁸ (Asleson & Schutta, 2005)

Another new emergent technology that proved to us that the Web was easily capable of supporting complex applications was the advent of HTML5. HTML5 is technically the fifth revision of HTML. However, when people refer to HTML5, they generally use it to mean the new HTML tags introduced in HTML5, the new stylesheet techniques introduced in CSS3, and the new JavaScript APIs. HTML5 is an umbrella term for these new advancements in Web development.

The new technologies in HTML5 are specifically created to facilitate the development of complex applications. It contains new abilities for Web developers such as 2D graphics rendering and vector graphics, offline Web application, timed media playback, local Web storage, a drag and drop interface between the Web page and your local computer, geolocation, document editing, WebSockets and more.

These new tools make creating Web applications simpler and easier for the Web developer as well as giving the developer power to do more complex actions through the Web page. HTML5 is particular useful for mobile devices. Recently Web developers have started using HTML5 to target mobile devices and creating HTML5 mobile applications. These are applications that use the Web as a platform but aim to rival the power, ability and features, of a native mobile application. Many developers find this method of development to be enticing since HTML5 applications automatically target all mobile devices and a developer no longer needs to code different applications for each mobile operating system. It is also useful because the application does not need to be approved by Apple in order to appear on their App Store™. The application is no longer bound by the closed application ecosystem that Apple has on their iPhone and iPad®. Building mobile applications this way also has the advantage of being able to continuously update

the application. When publishing a native application through the Apple App Store or the Google Android Market anytime the application developer needs to update their application, they need to submit a new version to the App Store. And in Apple's case, the new update will need to get approved which can take some time and prevent developers from releasing quick bug fixes. However, HTML5 applications can update whenever they want simply by changing the JavaScript code that the server sends to the mobile device.

2.2.2 Restful Architectures

We decided on creating a Web-based application using JavaScript and some new HTML5 technologies. We next researched on server-side design. Oracle had a requirement that the server we communicated with use a RESTful architecture.

REST stands for Representational State Transfer. It is a style of software architecture used for distributed systems. In software an architectural style generally refers to a set of design rules that identify the kinds of components and connectors that may be used to compose a system or subsystem.⁹ A service can be considered 'RESTful' if it adheres to certain REST principles. The REST architectural style was developed alongside HTTP 1.1. As such, the World Wide Web's guiding principles are what the REST architectural style is about. The World Wide Web itself can be considered the largest implementation of a REST architectural style. RESTful architectures are generally seen on the Internet.

⁹ (Shaw & Clements, 1996)

REST is a key design idiom that embraces a stateless client-server architecture in which the Web services are viewed as resources and can be identified by their URLs. REST is an analytical description of the existing Web architecture, and thus the interplay between the style and the underlying HTTP protocol appears seamless.¹⁰

A RESTful API design makes use of 4 HTTP protocol verbs: GET, POST, PUT, and DELETE. GET is used to retrieve a resource from the server. This resource is most commonly Web pages, however it can be used to get JSON objects or xml representations of data. POST is commonly used to send data or information to the server. It is commonly used in input forms to send user-input information, such as the user's contact information. PUT modifies data that already exists in the server. And DELETE simply tells the server to delete some specific data. These last two are generally used in conjunction with a database located on the server. However each of these verbs can be mapped to database actions on a server.

For example, in the teacher application, the application would make a GET request in order to retrieve all the grades for a particular application. It would use a POST request whenever the teacher registers a new student or adds a new assignment. The POST request would cause a new student/assignment row to be created in the database. The PUT request would be used whenever the teacher would try to modify a grade they already recorded for a student. And the DELETE request of course would be used to remove any of the information that the teacher was in charge of.

¹⁰ (Tyagi, 2006)

A RESTful API maps URLs for each of these verbs to specific actions on the server. For example, doing a GET request on this URL `http://exampleapp.com/student/8` would send back all the data of the student who has an id of 8 in the database. Likewise a DELETE request to that URL would simply delete all of that information.

Dynamic applications tend to use Ajax to query a server that uses a RESTful API. The application would query the server in the background and the server API would be dedicated to serving data back to the client. It would also maintain the database and keep handle any of the database queries the client requested of it.

Using a RESTful architecture simplifies development. By restricting the actions to a defined set of verbs and by creating a set of guidelines through this architectural style, Web developers can spend less time thinking about how they want their client and server to communicate. They could just use the verbs dictated by conventional RESTful practices to model their application logic. RESTful applications also facilitate simple and easy to understand APIs. This way, when using a third-party API in your application, you no longer need to spend time trying to understand the API style for each different third party. Instead, the RESTful pattern makes the various APIs more homogenous and easy to understand.

2.3 JavaScript Tools and Libraries

After we completed our research on RESTful architectures, we started thinking about how we were going to actually build the application. Once we decided on using JavaScript, Oracle told us that they wanted us to do research on JavaScript libraries, tools and frameworks. A library is a set of programs built on top of a language in order to give

the programmer the power to do more things. Libraries can be created for a plethora of reasons. They can be used to extend the functionality of a language, make tasks using a programming language easier to do, or even provide a framework for structuring your programs application logic.

Oracle wanted us to determine which libraries would be the best for helping us solve this problem. In order to properly ascertain which tools might useful, we looked at any JavaScript resources we could get our hands on.

2.3.1 jQuery

jQuery was one of the first libraries we looked at. At first glance, jQuery is a free and open-source library dedicated to making cross-browser DOM manipulation simple. However, there are many other useful things contained under jQuery's hood that make JavaScript development easier. jQuery is described as the “Write less, do more, JavaScript library.”¹¹

Many programmers would agree that jQuery truly is a wonderful resource for JavaScript development. jQuery is ubiquitous on the Web. Most of the sites that you see using JavaScript are being supported by jQuery under the hood. jQuery is being used by most of large entities on the Web such as Google, Netflix™, Mozilla.org™, Wordpress™, etc. It has become so prevalent that some newcomers to JavaScript development confuse function calls from jQuery's libraries with actual native JavaScript code. It's used by over

¹¹ (The jQuery Project, 2011)

55% of the top 10000 Web sites and is known as the most popular JavaScript library on the Internet¹².

jQuery was initially released on August 26, 2006. It was developed by John Resig because he wanted to fix a problem that he was having with developing on the Web. A decade ago, JavaScript was widely regarded as simply a toy scripting language. There were not many people actively working on making JavaScript a better language, and there also was not an accepted way to do many common tasks in JavaScript. However as the Web grew in popularity, and more groups and companies started using the Internet in more legitimate ways, JavaScript development grew more popular. Since more developers were going to spend more time dedicated to that platform, JavaScript development needed to be stronger and easier.

One frustrating problem with JavaScript development is that different browsers used JavaScript in different ways. If a developer writes JavaScript code that works in one browser, there was a good chance that it did not work in a different browser. This made JavaScript development somewhat unattractive as the programmer is forced to write different code for each browser he wanted to support. This is problem that John Resig set out to solve by creating a JavaScript library that would provide one consistent API. An API that works no matter which browser was being used.

jQuery's API was focused on DOM manipulation. DOM stands for 'document object model'. The DOM a cross platform convention for representing and interacting with objects in HTML, XHTML, and XML documents. It represents all the different sections that are displayed on a Web page. When you're developing for the Web, there are

¹² (Builtwith.com, 2012)

many times when you'd want to manipulate some of the objects displayed on the Web page. That is where jQuery comes in. It provides a well-thought out library for selecting and manipulating those DOM objects.

jQuery allows you to 'query' the DOM for different objects. You can use any attribute of an object for grabbing an object (or objects) out of the DOM. For example, say this html element existed on the Web page:

```
<a href="contact.html" name="contact" id="contact" class="navigation-link">Contact</a>
```

In order to get jQuery to 'select' this object from the DOM, you could do something like:

```
jQuery('[name="contact"]')
```

This returns a DOM element (or a list of DOM elements) where the name attribute equals "contact". The jQuery function takes a selector string that represents the query you are giving it. That function will return a jQuery object, which wraps around the HTML DOM element. With this jQuery object, you can perform many jQuery library functions on that DOM element.

There are some shorthand selector strings that you can use for the common queries that occur. You can use the '.' to specify the class attribute. The selector string '.navigation-link' could also be used to select the link element above. You can also use the '#' symbol to specify ids. The jQuery function is aliased to '\$'. The dollar sign symbol is commonly seen in jQuery code used in place of the jQuery function.

```
var contact_link = $('#contact')
```

In this example, the link element is selected by querying against its ID. The jQuery object containing the link DOM element is stored in a JavaScript variable called

contact_link. jQuery allows the programmer to perform different actions on the element. For example, you could change the site that the link points to.

```
contact_link.attr('href', 'http://Oracle.com')
```

This line of code changes the 'href' attribute of the link element in the DOM. Now whenever someone clicks that link, it would take them to Oracle.com instead of the contact page of the current Web site.

That's just the tip of the iceberg however. jQuery's DOM manipulation capabilities extend far beyond that. You can delete elements from the page, change the styling on specific elements, move elements, animate elements and much more. jQuery is not only for DOM manipulation however. Over the years functionality for doing other common things has crept its way into the jQuery library.

Another major problem that jQuery solves is cross-browser Ajax requests. Ajax stands for Asynchronous JavaScript and XML. It is a Web technique that facilitates asynchronous Web-development.

Using Ajax, a Web site can send and retrieve information to and from a server asynchronously (in the background). This way, the site and server can communicate without ever having to refresh the Web page. Google Gmail popularized this technique by showing developers that it can help you build full-fledged Web applications online by using asynchronous requests to the server.

jQuery provides a simple interface for using Ajax. It contains a function properly called 'ajax' that you can call from jQuery. Using the .ajax() function, a developer must simply specify the action that they want the browser to perform in the background, the information that they need to be sent, and also the URL they want the browser to send the

information to. Ajax() can also take a callback function as an argument, which will be ran whenever the request to the server comes back.

The Ajax function handled all the complex JavaScript code for doing Ajax under the hood. It works properly for all the major browsers so the Web developer doesn't have to worry about whether or not his Ajax implementation would for a different browser. Ajax will be a major technique we're going to make use of in this Oracle project, so a library that makes it easier to perform use this tech will be very useful.

A major factor that attracted us to jQuery was just how ubiquitous it was. There is a massive number of plugins built on top of jQuery that gives developers even more power. Making plugins for jQuery was also incredibly simple, so we would be able to use jQuery for the amazing features it gives us as well as have it be a hosting platform for the code we write to solve the problem Oracle gave us. The ubiquity of jQuery had another huge benefit; if we ever got stuck on a problem using jQuery, all we need to do is Google the problem and it was highly likely that someone else stumbled upon the same problem. We would not have trouble finding help on the Internet for our issues. All these beneficial factors made the decision to use jQuery an easy one to make. It was simply the best tool to help up build out the solution to Oracle's problem.

2.3.2 Prototype

Prototype is a JavaScript framework that aims to ease development of dynamic Web applications.¹³ Prototype is a suite of JavaScript functionality that makes it easier

¹³ (Prototype.org 2012)

for developers to do a wide array of different tasks such as DOM manipulation and Ajax handling. Sam Stephenson created Prototype in February of 2005 as part of the foundation for Ajax support in Ruby on Rails, a server side framework for the programming language ruby that tries to make developing Web applications as easy as possible.¹⁴

It is quite similar to jQuery. However, unlike jQuery, the way prototype handles DOM manipulation is by extending the DOM objects¹⁵. This means that the prototype framework directly gives the DOM objects new methods instead of creating a wrapper around the object like jQuery does. Many JavaScript experts discourage extending the prototypes of the base JavaScript objects, so Prototype has received much criticism due to their implementation. These issues, among other reasons directly led to Prototype not being as widely adopted as jQuery. We chose jQuery over prototype because of Prototype lacks wide scale adoption and because we felt that jQuery's API for DOM manipulation and Ajax handling was much more intuitive and user-friendly.

2.3.3 Google Closure Tools

Google closure tools are an assortment of various JavaScript libraries that serve a wide-array of different uses. These are a collection of tools that Google uses in-house that they decided to open-source for the shared benefit of the Web development community.

¹⁴ (Prototype.org 2012)

¹⁵ (Kangax, 2012)

Closure tools was used to build full-features applications like Google Docs™, Gmail, the Google+™ social network, Google Maps, and more¹⁶.

Google **Closure Library** is a large assortment of various libraries that each provides the developer with useful functionality. It consists of a large set of reusable UI widgets and controls, and from lower-level utilities for DOM manipulation, server communication, animation, data structures, unit testing, rich-text editing, and more.

The **closure compiler** is a JavaScript optimizer that parses JavaScript code and produces a heavily compressed code for use in production. The compiler works by:

- Removing white space
- Replacing variable/function names with smaller variable names. (for example: var username would get changed to var b)
- It completely removes any functions that don't ever get called.

The closure compiler is designed to work in conjunction with the massive Closure library. Since the Closure libraries contain libraries for facilitating a very wide assortment of functionality, the libraries in the Closure library suite all together take up way too much space. However, the value of all being able to choose functions and objects from all those libraries is very important. Google Closure compiler makes the space no longer an issue. Because of the compilers minimization ability, developers can just simply use whichever function/object from any one of the Closure libraries without worrying about how much space the entire language will take up. The compiler would just completely get rid of any

¹⁶ (Google, 2012)

of the code in those libraries that is not being used in the application. This way Google can keep adding more useful libraries to the Closure library without remorse.

Closure templates is a tool for easily generating HTML and UI elements from server side code. It can be used with both Java and JavaScript. Templates can be generated both server-side and on the client. The **Closure Linter** is a program that enforces the guidelines enforced by Google's JavaScript style guide. A linter is utility that parses code and points out any syntactical or semantic issues that the linter finds in the code. **Closure Stylesheets** is a system that supports a number of Google extensions to the standard CSS language. It is a platform that makes writing CSS much easier and more programmatic.

All of these tools are used extensively in Google applications so it is clear that they are battle tested and reliable. Google is also one of the most successful Web companies and they build some of the most innovative Web applications in terms of pushing Web technology forward. Because of this, the closure tools are bound to be full of some of the most useful and state-of-the-art JavaScript development tools. Despite all this, we chose not to use it for our project. We had trouble finding examples of people outside of Google using it so we did not know how non-Google developers felt about the tools. We also had no prior experience using the Closure system so it would have been a somewhat steep learning curve to figure out how all the different parts work together.

2.3.4 Dojo Toolkit

The Dojo toolkit is an open source JavaScript library that was created to make cross-platform JavaScript development easier and quicker. Like jQuery and Prototype, Dojo includes functionality for performing asynchronous requests. Dojo comes with Dijit, a widget system. Dijit contains a collection of widgets that provide useful features such as map creating, 2d vector drawing, sortable tables, dynamic chars, and more. The widgets seemed to be centered on displaying views in the browser. Making graphical applications with Dijit seem to be its primary use case. Dijit is perhaps the most enticing aspect of the Dojo toolkit. Dojo also includes mechanisms for facilitating data storage on both the client and the server. Dojo seems to be a useful tool, however, it did not fit the needs of this project. There was no real graphical focus in our app and Dojo's asynchronous API was not as clean and intuitive as jQuery's.

2.3.5 Ext JS

ExtJS is another Web framework whose goal is to facilitate the creation of interactive Web applications. ExtJS was initially created as an add-on the Yahoo User Interface library, but it quickly spun off into a standalone framework. In the beginning, ExtJS would make use of either jQuery or prototype under the hood to perform actions like DOM manipulation and Ajax requests, but in further versions, ExtJS removed these dependencies. Like Dojo, ExtJS's primary usecase is for building highly graphical Web applications. ExtJS seems to be incredibly useful for Web applications that have many

moving graphical elements. Also like Dojo, ExtJS didn't seem to be a good fit for our project.

2.3.6 Underscore.js

Underscore is a library unlike the ones mentioned in this paper thus far. Underscore is a utility-belt library for JavaScript that provides a lot of the functional programming support that you would expect in Prototype.js (or Ruby), but without extending any of the built-in JavaScript objects.¹⁷ Underscore is a small open-source JavaScript library that was designed to complement jQuery. It provides functions such as map, filter, invoke and many 60+ more functions that are typical in functional languages. It also provides list comprehensions similar to those seen in Python. Underscore also includes a lightweight testing suite as well as a lightweight templating library. The functions that Underscore provides would be incredibly useful if we were building a full-featured application as opposed to a proof of concept, so we did not find need for it in our project.

2.3.7 Node.js

Oracle specifically asked us to look into Node.js as it was being talked about quite a bit in the Web developer community. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-

¹⁷ (DocumentCloud, 2012)

intensive real-time applications that run across distributed devices¹⁸. Node.js was released very recently and since then it amassed a large dedicated following of coders.

Node.js is actually not a client-side JavaScript framework that is interesting because JavaScript has only been known to run on browsers in the client. Node.js actually lets the developer write JavaScript code on the server. It takes the functional callback style that JavaScript is well known for and creates a platform that facilitates the development of Web servers that can take in multiple requests concurrently. JavaScript code in Node.js is all asynchronous using callbacks, so it is unlike developing normal JavaScript.

This method has gained quite a bit of steam in the Web development community. It allows the developer to write JavaScript both on the client and the server. Front end engineers who were already proficient with JavaScript can now use their JavaScript ability to make servers that can wither the storm of high trafficked Web applications.

The work we did for this project was going to be client-side and Oracle had already started working on the server portion of their application so node.js didn't have a place in our tool kit.

2.4 Client-side Storage

The solution we came up with required the client to be able to store data locally. In order to get this to work, we needed to research the different methods that existed for doing that storage.

¹⁸ (NodeJS.org, 2012)

2.4.1 Native solutions -- MySQL™

Our first option was using a conventional database locally. Oracle is the creator of the world's most popular open source database, MySQL.¹⁹ MySQL is a widely used and battle tested relational database that's gained adoption by a large chunk of software developers everywhere. It also had another advantage in that it was built by Oracle so they would have no issue implementing the system into their application.

The problem with using this native storage system however is that the database would need to be stored somewhere on the user's local system. The application would also need to have read write access to the file system, specifically the location where the database is stored. And the user's local machine must be able to run C and C++ in order to run the MySQL application. These requirements would not have been an issue on desktop machines. However, on mobile devices, we run into all sorts of problems. Firstly, the iOS (iphone and ipad operating system) does not allow read/write access to the file system for app developers. This is presumably for security reasons. It would also not be possible to run the application on either android or iOS. Mobile devices were Oracle's most needed platform for this application, so these problems discounted the use of native storage solutions. We had to look instead towards storage solutions built into the browser.

2.4.2 Browser-based Storage

Browser-based storage solutions would work well since they can be supported by any device that has a browser that adopts the storage system. Some new developments in HTML5 brought developers some new storage capabilities. HTML5 **Local Storage** is a

¹⁹ (MySQL.com, 2012)

simple key-value pair based store. Web Storage is a W3C specification that provides functionality for storing data on the client side until the end of a session (Session Storage), or beyond (Local Storage). It is much more powerful than traditional cookies, and easier to work with.²⁰ A developer would use a simple native JavaScript API to interface with html5 Local Storage. While useful, a simple key-value store would make it difficult for the client to represent the state of a database since databases have more complicated data storage formats.

WebSQL and **IndexedDB** are two more new storage solutions HTML5 provides with us. They are both built on top of HTML5 local storage and provide a system much more akin to traditional databases. WebSQL is a relational and gives the developer all the features (and strain) expected from conventional relational database systems such as MySQL. IndexedDB fits in somewhere between simple Local Storage and full-featured WebSQL. It supports indexes like those of relational databases, so searching objects matching a particular field is fast; you don't have to manually iterate through every object in the store.²¹










Both of these higher level solutions would fit perfectly for our application since we needed to replicate the data in a database into the client storage. Using something that is already similar to conventional databases would have made the process of developing the application go much smoother.

However, there was one major issue. Like most of the problems that existed with some of the other solutions we researched, it had to do with support. Both IndexedDB

²⁰ (Opera.com, 2012)

²¹ (HTML5rocks.com, 2012)

and WebSQL are not supported by most browsers and devices.

BROWSER SUPPORT									
									
<i>Web Storage - name/value pairs</i> <small>↗</small>	4+	4+	4+	11+	8+	5+	3+	—	11+
<i>IndexedDB</i> <small>↗</small>	11+	4+	—	—	10	—	—	—	—
<i>Web SQL Database</i> <small>↗</small>	4+	—	4+	11+	—	5+	3+	—	11+

Data courtesy of [caniuse.com](#) ↗ and [Chrome Platform Status](#) ↗.

Figure 2 - Web Storage Browser Support. courtesy: [html5rocks.com](#)

Simple HTML5 Local Storage was however supported by nearly all browsers and is supported on all devices (include mobile). HTML5 turned out to be the only storage solution that we could move forward with. In order to accomplish the task of replicating the database locally, we would need to build a custom solution on top of Local Storage.

3. Methodology

Our work was divided into three primary sections: research (the results of which are covered in Section 2), generation of the client and server application, and building of the eventual framework for interaction.

3.1 Server and Client Application

3.1.1 Requirements

Oracle had several requirements before we began building our application in order to test our eventual framework with:

- Keyed towards business applications, managing life-cycles of several entities
 - Related to each other (1:1, 1:n)
 - Read and written from a database
- Application relies on RESTful data services
- Client should be able to modify data regardless of connection to server
 - Data should be modified locally when unable to send to server
 - Changes to data should persist even if application is closed
- Application should run on mobile devices
 - Tablets (iPad)
 - Laptops

3.1.2 Application Proposal

To begin our work on the framework we first had to propose to Oracle an application to design and develop, with which we would eventually build the framework capable of handling modifying data regardless of connection to the server. Our original proposal was to build a computer process metric monitoring application that would continuously update a chart of data as new data became available (which would be randomly generated on the server at a specific interval). This proposal was rejected because the standard business case that was being aimed for involved updating and creating of new records on the client side, which would then be sent to the server once a connection was available.

Our second proposal was to develop an application that would allow teachers to update grades and comments on an assignment, with a standard connection between the server and the client (See Figure 3.1). This allowed a teacher to begin grading assignments at school on a laptop, but continue grading them if they leave the office and no longer have a connection to the server at the school. This would allow them to

continue inputting grades during travel or from home (even if no VPN is available into the school). This proposal was accepted by Abhijit, our main contact at Oracle.

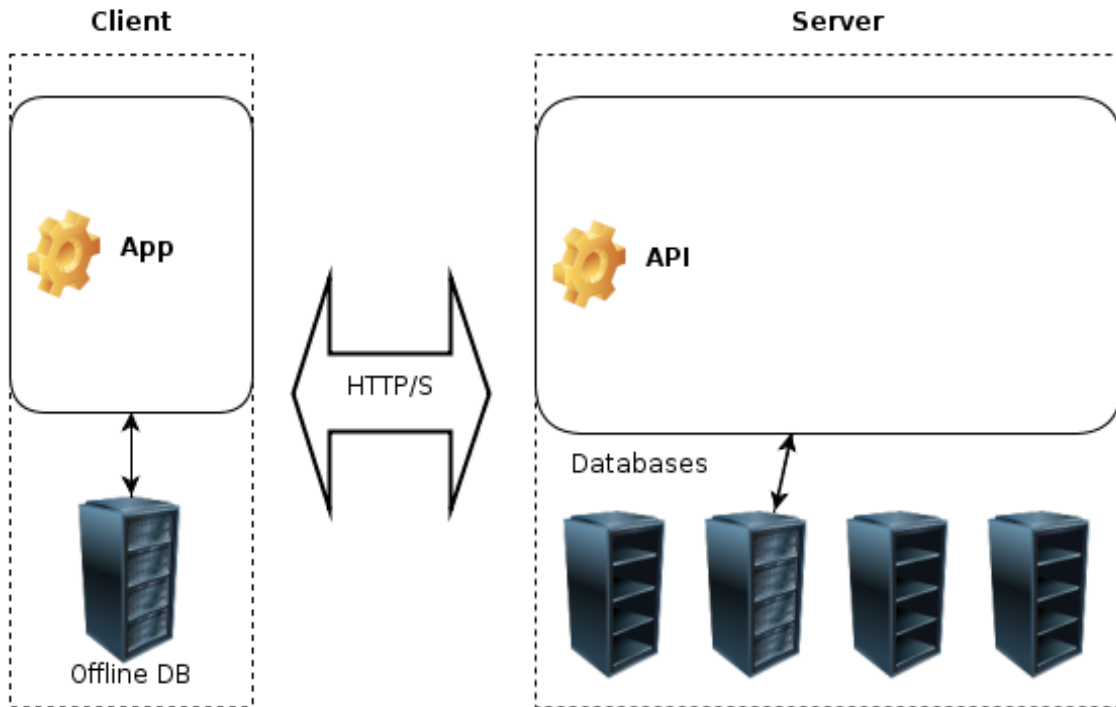


Figure 3 - Standard Server/Client Representation

3.1.3 Building the Server

Since the goal of the project was the generation of the framework and not the application, we wanted to finish the sample application as quickly as possible. We then built a simple server that would communicate as any other HTML5 application would work, with which we could then go forward and implement with our framework instead of the standard Ajax queries. We designed an API for the server that would allow a teacher to give grades and comments to assignments in their classes (See Figure 2). The

server was written in Python using Flask, a micro-framework for Web development.²² This allowed us to have a functioning server within two days of beginning the writing. A database was put into place containing a small number of classes, students, and grades, with which we could begin working with the server.

```
/classes
    GET: gets the list of classes
/assignments/<class_id>
    GET: gets the list of assignments for the given class
/grades/<assignment_id>
    GET: gets the list of grades for the given assignment
    PUT: adds a grade record to the database for the given assignment
    requires: student_id, grade
    POST: edits a grade record
    requires: student_id, grade
/grades/<student_id>/<assignment_id>
    DELETE: deletes the given students grade record for the assignment
/students
    GET: gets the list of students
```

Figure 4 - Server API

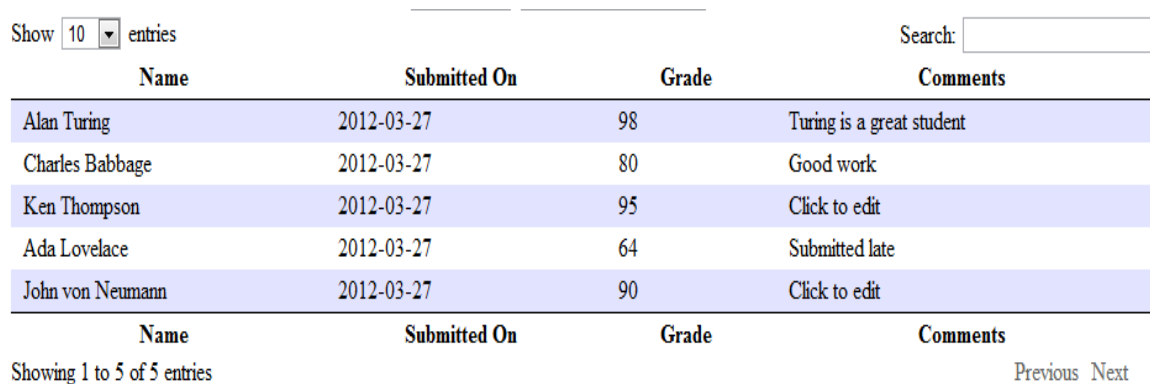
Requests to the server are passed by a combination of data embedded directly into the URL and sent as part of the parameter. The server responded to all GET requests with a JSON array, containing a list of JSON objects representing the array of results from the database.

²²(Ronacher, 2012)

3.1.4 Building the Client

Once the server was completed, we began work on a client to interact with the server, as any other modern HTML5 application might today. Using our research of HTML5 and the requirements set forth by Oracle, we decided that the best JavaScript framework to use in building this application would be jQuery, due to its already extensive usage and support (Detailed in Background).

In order to display the grade information, we decided upon the usage of a jQuery plugin called DataTables, which allowed for easy implementation of many features for viewing a list of data (See Figure 3.3).²³ This allowed for ease of showing the data once it was retrieved from the server. The data, in the first iteration of the application, was retrieved through a simple Ajax GET request to the server.



The screenshot shows a web interface for a DataTables plugin. At the top left, there is a 'Show 10 entries' control with a dropdown arrow. To the right is a search box labeled 'Search:'. Below these is a table with four columns: 'Name', 'Submitted On', 'Grade', and 'Comments'. The table contains five rows of data, each with a light blue background. Below the table, there is a 'Showing 1 to 5 of 5 entries' label and 'Previous Next' navigation links.

Name	Submitted On	Grade	Comments
Alan Turing	2012-03-27	98	Turing is a great student
Charles Babbage	2012-03-27	80	Good work
Ken Thompson	2012-03-27	95	Click to edit
Ada Lovelace	2012-03-27	64	Submitted late
John von Neumann	2012-03-27	90	Click to edit

Figure 5 - DataTables Plugin With Data

In order to modify the data once it was shown, we decided to use another jQuery plugin, called Jeditable, which allowed editing of cells in the table in place (See Figure

²³(Jardine, 2012)

3.3).²⁴ Once the change was made and the user hit enter, an event was triggered, calling a function that in turn performed an Ajax PUT request, updating the specified data.

Show entries Search:

Name	Submitted On	Grade	Comments
Alan Turing	2012-03-27	<input type="text" value="98"/>	Turing is a great student
Charles Babbage	2012-03-27	80	Good work
Ken Thompson	2012-03-27	95	Click to edit
Ada Lovelace	2012-03-27	64	Submitted late
John von Neumann	2012-03-27	90	Click to edit

Showing 1 to 5 of 5 entries Previous Next

Figure 6 - Editing Data with Jeditable

This base application worked fine while the application had a connection to the Internet (and the server), but once a connection was lost, attempted updates to the server returned error messages. With the basic application in place, we began working on the framework.

3.2 The Framework

The framework was developed piece by piece, adding in additional code to solve new problems. The coding was all done as part of a jQuery plugin, which we expanded to solve problems one at a time. First the four main request types were separated into functions in the framework that could handle performing the actual Ajax requests. The next step was to come up with a method of specifying the API of the server to the framework. After this a queue was implemented to handle all requests so if the connection to the server was lost the requests could be performed when it was re-

²⁴(Tuupola, 2012)

established. The final step was to implement the persistent storage and modification of local data, even when the application is closed and the connection to the server is missing.

Every step of the framework generation was tested individually using the JavaScript test suite QUnit. Our code was also ran against JSHint, which is used to ensure that JavaScript code follows a set of standard conventions.²⁵

3.2.1 Abstracting the Request to the Framework

The first implementation of the framework had four functions, `doPut`, `doGet`, `doPost`, `doDelete`. Each function took two arguments, the URL and an associative array of parameters to pass to the URL. GET requests were performed synchronously as to return the result of the requests to the application, whereas all other requests were performed asynchronously. This solved the problem of abstracting most of the request away from the client, but did nothing to solve the issue of what to do when the connection was lost.

At this point we essentially had the same application as when we began coding the framework, but the client no longer had to manually perform the Ajax requests. However, the framework relied completely on the developer manually supplying the full URL and all relevant data as an associative array.

²⁵(Kovalyov)

3.2.2 Supplying API for Communication with the Server

At this point, with the ability of the framework to perform the four critical requests, we began work on determining how to specify the API for communicating between the framework and the server. Our initial reasoning brought us to two possibilities for specifying this: through a GET request to the server that would return the API or by specifying the API to the framework in the client. The API in either case would specify all necessary URLs and parameters for communication between the client and the server.

We settled on specifying the API through the client, because this provided the necessary DOM element that all jQuery functions had to be called on (jQuery acts on DOM objects). Custom tags were used to specify the communication protocol for a specific URL (See Figure 3.5). The server was modified to include a new URL called “test” which was used solely to tell if the server was connected, and so long as it returns a 200 OK the client knows it is connected. At this point, performing any request required now that you just specify the type of request to perform, the item to retrieve, and the parameters. The framework built the URL with regex replacement of any missing parameters and passed the rest of them in as the normal data for the request.

```

<oracleApp id="tapp" name="tapp">
  <url test="/app">http://localhost:5000/</url>
  <api>
    <method type="GET">
      <mapping item="grades" location="/grades/$assignmentID"></mapping>
      <mapping item="assignments" location="/assignments/$classID"></mapping>
      <mapping item="classes" location="/classes"></mapping>
    </method>
    <method type="POST">
      <mapping item="grades" location="/grades/$assignmentID">
        <parameter name="student_id"></parameter>
        <parameter name="grade"></parameter>
      </mapping>
    </method>
    <method type="PUT">
      <mapping item="grades" location="/grades/$assignmentID">
        <parameter name="student_id"></parameter>
        <parameter name="grade"></parameter>
      </mapping>
    </method>
    <method type="DELETE">
      <mapping item="grades" location="/grades/$assignmentID">
        <parameter name="student_id"></parameter>
        <parameter name="grade"></parameter>
      </mapping>
    </method>
  </api>
</oracleApp>

```

Figure 7 – Specifying the API

3.2.3 Queuing Requests

Brainstorming methods to queue the changes being made to the server, it was decided to store all required information for every request when there was no connection to the server such that the application could perform the requests once a connection to the server was re-established. Bringing up our thoughts to Professor Pollice, he confirmed this method of storing all required data to be performed later was a variation of a design pattern called the command pattern, and is what we used in this iteration (and the final version) of the application. We implemented a simple FIFO stack to hold the list of requests. Every single request is added to the stack and, if a connection to the server exists, every request is processed (until the connection no longer exists).

This solved the first major problem with the application. Now, if the connection to the server went down in the middle of the client interaction, so long as the page was not refreshed, when the connection came back every change was sent over the server successfully. However, if the application were closed, the changes would be lost.

3.2.4 Implementing Local Storage

Our research led us to believe that the best solution for storing information locally for when the server connection could disappear was Local Storage, due to its steady adoption by all major browsers (including tablet browsers). Our work first focused on ensuring that our GET requests stored the information they received on the client in a persistent manner.

We changed the functions to now include a parameter for `tableID` that we would use to identify what subset of the data we wanted to access, specifically so we could access a particular item again later by referencing the same `tableID`. This meant that we stored the GET request results under “`appName.item.tableID`”. To store the data we used the `stringify` method of JSON to convert the array of JSON objects retrieved from the server into a string that we could store in Local Storage. When a GET request was performed on the same data (identified by the `tableID`) and a connection was unavailable, we instead use to `parseJSON` method of jQuery to return the original JSON array.

With this implemented, after every change to a local table, we simply had to perform another GET request on that subset to update the local table. However, this was inefficient, especially with larger data sets. Instead, we needed a way to modify the local

data whenever we performed a POST, PUT, and DELETE request, so that it would match the source data. This was first done by implementing three functions, `localPut`, `localPost`, and `localDelete`. Each function worked by emulating their purpose in a RESTful environment. This, however, for PUT and DELETE requests, required that we created a unique identifier, which named `aaRecordID`, for each row within a subset, which could be anything so long as it was still unique to that row within that subject. We modified the server to return a unique `aaRecordID` for each row.

We also added the queue of local changes to what was being stored in Local Storage, because we noticed that, while the local data was being locally changed correctly, our changes were not being sent to the server if we reloaded the application. At this point we also fixed an issue with the client believing it was connected to the server even when no connection existed. This was due to the client keeping a cache of the test URL response, which it would default to when the connection was not detected. By specifying not to use a cache in the Ajax request, we averted this issue.

3.2.4 Various Bug Fixes and Improvements

We also added the queue of local changes to what was being stored in Local Storage, because we noticed that, while the local data was being locally changed correctly, our changes were not being sent to the server if we reloaded the application. At this point we also fixed an issue with the client believing it was connected to the server even when no connection existed. This was due to the client keeping a cache of the test URL response, which it would default to when the connection was not detected. By specifying not to use a cache in the Ajax request, we averted this issue.

Based on feedback from Abhijit, we worked on combining repeated code sections that were common to several functions, putting them in separate functions and calling them as needed. Our `doPut`, `doGet`, `doPost`, and `doDelete` were renamed to `PUT`, `GET`, `POST`, and `DELETE`, respectively, for aesthetic purposes and ease of use. We also worked on modularizing the code, ensuring that all access to `Local Storage` was done through a set of two functions, one for setting and one for getting. This allows for the code to be easily modified in the future for different methods or storing data to be handled depending on the capabilities of the current browser.

A similar approach was taken for handling the results of `GET` requests; a single function is now in place which can be modified to determine what format the data is in and perform the conversion to `JSON` (in case the server sends data in `XML`, `CSV`, or some other format). We also changed the framework to no longer be dependent on the server to set the `aaRecordID` parameter on each row retrieved from a `GET` request. These changes were put in place to minimize the amount of requirements on a server that would be used with the framework, making the framework almost completely independent from the server. With this done, our work on the framework was completed.

4. Results and Analysis

The framework that has been developed acts as a proof-of-concept for an advanced system of handling communications between a server and a client that allows data and changes made to data to persist even when the connection between the server and the client does not. This allows for developers of client-side Web applications to rapidly build data-intensive applications that can continue to modify data even when

offline without having to create their own methods for handling data when there is no server connection available.

This is specifically useful for programmers of applications that will be used on mobile platforms with wireless connections, such as laptops, tablet computers, and smart phones, where the Web connection may be especially poor, where traveling prohibits access to the Internet, or where there is no VPN available to access the needed resources. With the framework in place, all of these issues are solved by providing a new API for communication with the server that handles most of these issues. The code is also built in a manner such that it can easily be modularized, and changes that would most likely be implemented in the future would not require any significant rewrites to the code, only the addition of new code in the form of a handler.

The original response from Oracle on the framework was positive, though they had some suggested changes to the framework. These changes were either implemented or documented as to how they might be put in place (under Future Work). The application met our goals set at the start of the project, and we were able to program an ability to generalize the code for use with a variety of server beyond what we had originally intended.

5. Future Work and Conclusions

5.1 Future Work

Going forward, there is additional work that can be done to expand the usage of the framework to handle inter-table dependencies and triggers, storing of local data when `Local Storage` is not available on a client, dealing with different communication

protocols, and enabling better handling of conflicting changes in the database. These additions could be implemented by Oracle or even as part of a further MQP. With these changes to the proof-of-concept framework it would be suited to be deployed into more general usage.

The first piece of future work to implement would be an emulation of triggers as they are used in modern databases. This could be implemented by the client during the `init` function querying a URL specific to this framework on the server which specifies a list of existing triggers in the database. The application could then parse the list of triggers and generate an array of functions simulating the triggers that can be called as appropriate on the local data stores. These solutions most likely would involve the storage of additional information on the client.

When `Local Storage` is unavailable, the current code defaults to storing the local data in a simple array, which as mentioned previously means that the application can continue running when a connection is unavailable, but reloading it means all changes are lost. The code is, however, written in a fashion that it could be easily modularized, meaning that once the client browser's capabilities are determined, the appropriate handler function given the capabilities of the browser and browser version can be specified. All of this can be determined and set during the `init` function.

Since the application was developed in such a modularized fashion, different communication protocols with the server can also be handled in an easy fashion. Once the client knows the format of the data it is receiving from the client, the framework could choose an appropriate conversion method, depending on if the data received is in XML, CSV, XHTML, HTML, or any other format, to convert the data received into the

JSON form used by the framework. There are several possible methods for determining what form the data is being sent to the client in; this could be done through the client requesting the communication protocol at the start during the `init` function or simply through reading the MIME-type of each GET response.

There is also the issue of two clients changing the same information. A common current method of solving this is to keep a time-stamp on every record and, when one record is changed in two locations, the second client to perform the change receives a message asking it whether it wishes to overwrite the other client's changes. This would also involve keeping a record of every request that has been sent but has not yet received a response from. There is, however, a potential issue in a single client making multiple changes to the same data, which could also trigger the same warning when it should not be necessary. This could be solved by combining multiple posts to the same data into a single request when possible.

5.2 Conclusion

The framework proof-of-concept we have created has the potential to be a powerful and widely used tool in the hands of developers, and once further developed it could be a popular tool for the development of data-intensive HTML5 Web applications. The development of the application has also been a great opportunity to work with Oracle and expand our experiences and knowledge in cutting-edge Web-development, especially dealing with HTML5 application development and jQuery.

During the process of researching the current field of Web-development and building our own jQuery framework we have gained a great deal of practical skills and

technical expertise that will surely be useful in the future. With the ever-growing importance of Internet applications in the current market, we have gained many marketable skills and solid proof that we cannot just work in the field, but even create innovative models in Web-development.

We have demonstrated that it is possible to build an application that is fully capable of keeping records of transactions when there is no connection to the server available, and then sending the transactions to the server once the connection is re-established. It also is able to modify the local data at the same time, ensuring that the local data reflects the server data should be once the changes are actually pushed through (in the case that the connection is unavailable). Any RESTful server capable of sending data in the form of a JSON array can be used with our framework, and with a few small modifications to our framework other data forms could be used as well.

Overall, the project was a success. While we were not able to implement every possible feature, we were able to build and test a sample Web application as might exist today, build a framework that abstracted communication with the server and allowed the client to continue working regardless of the connection to the server, and modify our test application so that it would use our framework. GET, PUT, POST, and DELETE requests were fully implemented for the communications specified, and it worked during all testing. For those features we weren't able to develop, we did implement modular functions that would make any changes easy to perform. Our code base will surely be useful for further development, we have learned a great deal, and are very thankful for the opportunity we were given to work on such a ground-breaking project.

References

- Asleson, R., & Schutta, N. T. (2005). *Foundations of Ajax. Foundations*. Apress.
Retrieved from <http://www.amazon.com/dp/1590595823>
- Bacon, D. F., Vechev, M. T., Cheng, P., Grove, D., Hind, M., Rajan, V. T., Yahav, E., et al. (2005). High-level real-time programming in Java. *Proceedings of the 5th ACM international conference on Embedded software EMSOFT 05*, 35(10), 68-78. ACM Press. Retrieved from <http://portal.acm.org/citation.cfm?doid=1086228.1086242>
- The jQuery Project. (2011). jQuery: The Write Less, Do More, JavaScript Library. *Physical Review E*. jQuery. Retrieved from <http://jquery.com/>
- Comparison of JavaScript frameworks. (n.d.). Retrieved from http://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks
- Cutler, K.-M. (2012). Done Deal: Zynga Gets “Draw Something” Phenom By Acquiring OMGPOP. (We’re Hearing \$210M.) | TechCrunch. Retrieved April 25, 2012, from <http://techcrunch.com/2012/03/21/done-deal-zynga-gets-draw-something-phenom-by-acquiring-omgpop-were-hearing-210m/>
- Dixit, S. (2012). Web Storage: easier, more powerful client-side data storage - Dev.Opera. Retrieved April 25, 2012, from <http://dev.opera.com/articles/view/Web-storage/>
- DocumentCloud. (n.d.). Underscore.js. Retrieved April 25, 2012, from <http://documentcloud.github.com/underscore/>
- Google. (2004). Google Gets the Message, Launches Gmail. Retrieved April 25, 2012, from <http://www.google.com/press/pressrel/gmail.html>
- Google. (n.d.). Closure Tools — Google Developers. Retrieved April 25, 2012, from <https://developers.google.com/closure/>
- Honan, M. (n.d.). apple unveils iphone. 2007. Retrieved April 25, 2012, from <http://www.macworld.com/article/1054769/iphone.html>
- Hopmann, A. (n.d.). Story of XMLHttpRequest » Alex Hopmann’s Blog. 2007. Retrieved April 26, 2012, from <http://www.alexhopmann.com/story-of-xmlhttp/>
- Jardine, A. (2012). *DataTables*. Retrieved April 20, 2012, from <http://datatables.net/>
- jQuery Usage Statistics. (n.d.). Retrieved April 25, 2012, from <http://trends.builtwith.com/javascript/jquery>

- Kangax. (n.d.). Perfection kills » What's wrong with extending the DOM. Retrieved April 25, 2012, from <http://perfectionkills.com/whats-wrong-with-extending-the-dom/>
- Kovalyov, A. *About JSHint*. Retrieved April 14 from <http://www.jshint.com/about/>
- Meeker, M., Devitt, S., & Wu, L. (2010). Internet Trends. Retrieved April 25, 2012, from http://www.morganstanley.com/institutional/techresearch/pdfs/Internet_Trends_041210.pdf
- node.js. (n.d.). Retrieved April 25, 2012, from <http://nodejs.org/>
- Oracle. (n.d.). MySQL :: The world's most popular open source database. Retrieved April 25, 2012, from <http://www.mysql.com/>
- Prototype JavaScript framework: Easy Ajax and DOM manipulation for dynamic Web applications. (n.d.-a). Retrieved April 25, 2012, a from <http://prototypejs.org/>
- Prototype JavaScript framework: The Prototype Core Team. (n.d.-a). Retrieved April 25, 2012, a from <http://prototypejs.org/core>
- Ronacher, A. (2012). *Flask*. Retrieved April 20, 2012, from <http://flask.pocoo.org/>
- Shaw, M., & Clements, P. (1996a). Toward boxology. *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops* - (pp. 50-54). New York, New York, USA: ACM Press. Retrieved from <http://dl.acm.org/citation.cfm?id=243327.243352>
- Swartz, A. (n.d.). a brief history of ajax (aaron swartz's raw thought). 2005. Retrieved April 26, 2012, from <http://www.aaronsw.com/weblog/ajaxhistory>
- Tuupola, M. (2012). *Jeditable - edit in place plugin for jQuery*. Retrieved April 20, 2012, from <http://www.appelsiini.net/projects/jeditable>
- Tyagi, S. (2006). restful Web services. Retrieved April 25, 2012, from <http://www.oracle.com/technetwork/articles/javase/index-137171.html>