goal.ly:

An Exploration of Best Practices in Web Development

Daniel Watson

CIS 4914 Senior Project

Advisor: Dr. Douglas D. Dankel II
ddd@cise.ufl.edu
University of Florida
Department of CISE
Gainesville, FL 32611

Date of Final Presentation: April 23, 2012

Abstract

As the World Wide Web continues to mature, the capabilities and experiences that can be created through this medium continue to evolve. Web browsers today are more capable than was imaginable just a few short years ago. Organizations like Google and Mozilla have been showing what is possible using only the basic building blocks of HTML, CSS, and JavaScript. Once we had to install separate applications, like Adobe Acrobat Reader, to view PDF files. Now Mozilla has created a JavaScript application, PDF.js, that is capable of parsing and rendering a PDF completely in the web browser [1]. Many individuals and organizations have switched from using traditional desktop email clients to Google's web based GMail [2]. Advanced websites like these are beginning to be referred to as applications and have little in common with the websites of the past. Software Engineers must learn new techniques and become familiar with new tools and languages to keep up with these changing possibilities and expectations.

This project is an exploration of current trends and best practices for modern web application construction [3, 4]. Through the lens of a practical project, we will examine current trends and techniques and deliver a working website that can be used as both a learning experience and a tool.

Introduction

The purpose of this project is to create a useful web application while exploring modern Software Engineering techniques. As it was near the beginning of the year a goal tracking application seemed to be an appropriate application to me. One similar application that I particularly liked was AskMeEvery.com [5]. This website allows the user to input a question and phone number. It will then poll the user once a day via text message to answer the question. The site also provides historic graphs of the user's responses.

Problem Domain

One technique that I like for sticking with goals is Jerry Sienfeld's "Don't Break the Chain" method [6]. This is a motivational technique using past performance for motivation in the future (e.g., "I've studied every day for the last 67 days. If I don't study today I'll break the chain, so I better study.").

Inspired by this method, my web application, goal.ly, will allow the user to enter multiple questions to be polled on daily. It will poll the user via a Twitter direct message. The user will be able to select what time to have the question asked for each question and to specify what type of response is needed: boolean, numeric, or Likert scale. To provide these functionalities, goal.ly will use many Open Source technologies including Grails for server side support [7], Spring Security Core and Spring Security OAuth for user authentication [8, 9], Twitter Bootstrap for styling [10], Backbone.js for client-side Model-View-Controller (MVC) support [11], Twitter4j for communicating with Twitter [12], and the Google Chart API for graphing user responses [13].

Previous Work

One of the best resources for learning the newest HTML features is the HTML5 Rocks website [14]. It provides, or links to, documentation, references, and tools for learning and trying out the latest HTML features. In addition it has a blog that frequently links to impressive new JavaScript applications.

For best practices related to website construction, two of the best resources are the "Web Performance Best Practices" website from Google and the "Best Practices for Speeding Up Your Website" website from Yahoo [3, 4]. In general the Google site provides more information on features such as caching, request overhead, and minimizing payload size, while the Yahoo site provides more information on changes that should be made to the HTML itself to improve performance. In addition Google provides the PageSpeed tools for performance optimization [15]. For measuring the performance of websites, two good tools are the YSlow Firefox plugin [16] and the Developer Tools that come installed with Google Chrome [17]. The YSlow plugin provides a grade based on the performance of the loaded website. Google Chrome Developer Tools allow easy perform website performance audits and measure network utilization.

Solution

Every web based project duplicates certain functionality such as negotiating and handling incoming browser network connections and conforming to the HTTP specification. To abstract away these and other concerns many web frameworks have been created. Wikipedia lists 36 web frameworks for Java alone [18]. In 2004 David Heinemeier Hansson released the Ruby on Rails

(RoR) framework for the Ruby programming language. This framework uses a variation on the MVC pattern. It uses “convention over configuration”, whereby the framework is configured by default for the common case and the developer needs not provide any configuration or setup changes for the default settings. It embraces the “Don’t Repeat Yourself” philosophy and attempts to make as much user code as possible reusable. RoR also includes an object-relational mapping (ORM) library for handling the impedance mismatch between relational database systems (RDS) and object-oriented (OO) code. RoR was very successful and paved the way for modern web frameworks. These frameworks require much less code to produce the same output as their predecessors.

I choose to develop goal.ly using Grails. It is a RoR-inspired web framework that uses the Groovy programming language and runs on the Java Virtual Machine (JVM). Grails has a plugin system which provides many prebuilt libraries to provide additional functionality to a Grails web application. To speed up development goal.ly takes advantage of many of these plugins. By reusing Open Source software goal.ly benefits from a faster time to market and a reduced bug count. As Eric S. Raymond said, "given enough eyeballs, all bugs are shallow" [19].

As a live project, goal.ly needs to be served by a server somewhere on the Internet. Providers of a Platform as a Service (PaaS) perform server maintenance so developers can concentrate on core application functionality instead of having to divide attention between software enhancement and server support. Through automation and economies of scale, they are able to offer a much cheaper solution than traditional operation departments. I choose to use Heroku as my PaaS so that I could deploy goal.ly right from the version control system [20]. The website is also using a RDS provided by Heroku for server side storage.

A content delivery network (CDN) is a tool for delivering static content to the end user. A CDN has many servers located in various physical locations around the world. When a user requests a resource from the CDN, it will be delivered from a server that is geographically close to the user. This reduces user wait time. Additionally it is beneficial to use a CDN that has a different hostname from the main website application so that the browser does not have to upload user cookies with each request for a static resource. I choose to use Amazon CloudFront as the CDN for goal.ly based on its low cost and ease of use [21].

In developing a web application there is a tradeoff between ease of development and website performance. For example, JavaScript and CSS are easier to develop if each related bit of functionality is broken apart into its own file. However this can lead to the browser making many requests for static resources. For better performance the number of requests that the browser needs to make to display the web application should be kept to a minimum. To handle these conflicting goals, programs have been developed to combine the resources into one bundle. This way the files can be kept separate for development without sacrificing performance. In goal.ly I am using the Grails Resource plugin [22] to perform this purpose.

The Grails Resource plugin pipeline provides additional features beyond combining files. After combing the files a SHA1 hash is taken of the combined file. The file is then renamed to match this hash. This way whenever there are code or style changes the combined file will receive a new name. Next the resulting bundle is run through a minimization program. This removes whitespace, comments, and dead branches of code, reducing the client download size. After minimization the resulting text can be further compressed using gzip compression. When the final file is sent back to the user, a header is sent with it telling the browser that the static file can be safely cached for a long period of time, speeding up subsequent access to the web application.

To be a web application the website needs to perform as much processing as possible in the client browser to improve the user experience (UX). The UX is improved by giving faster feedback on user actions. For example, if a traditional website were collecting a string of text from the user, it might send the string to the server for validation. If the string did not pass validation, then an error would be rendered on the server side and sent back to the user. The user's browser would then reload and re-render the response. More modern websites should perform the validation, using JavaScript, inside the browser before sending the string to the server, thereby saving one request and a response (which might spawn many other requests if static resources were not cached properly).

Validation is not the only thing that can be performed in the browser. In goal.ly I allow the user to add, remove, and change questions asynchronously. For example when the user removes a question, goal.ly will send a request to the server in the background. The server will

respond to this request in JavaScript Object Notation (JSON) [23] format. This JSON response is orders of magnitude smaller than the HTML response that an older website might make. When the browser receives the response from the server, it will use JavaScript to remove just the one question from the page avoiding a full page re-render.

The first generation of websites to using client-side logic frequently relied on the Document Object Model (DOM) of the website to store state. This lead to confusing and unmaintainable code. To solve this problem, Software Engineers are starting to reuse a technique that was originally developed for the desktop. The MVC pattern has been established since 1979, but has only recently been used in the browser. With goal.ly I decided to use the Backbone.js MVC framework. Backbone.js provided the structure and separation of concerns necessary to provide a good UX while keeping the code maintainable. In goal.ly I used Backbone.js for storing the client-side model, validating it, and sending updates to the model to the server. Additionally I made use of Backbone.js views for rendering the questions from the model. Finally I used the Backbone.js router to allow the user to make full use of navigation buttons and bookmarks while remaining on one HTML page.

Unfortunately not all web browsers follow the HTML and CSS specifications exactly. This has led to numerous websites like “When can I use...” and “QuirksMode” [24, 25] for determining what works in each browser. Recently website designers have started developing CSS frameworks to ease these cross-browser issues. A CSS framework provides a set of CSS classes that have been tested across many browsers. The classes provide a base set of styles to build upon that provide the website with a consistent look and feel. For goal.ly I used the Twitter Bootstrap CSS framework. Released in 2011, it is a recent arrival in the CSS framework space, but it has quickly gained popularity. It is currently the most watched project on GitHub.

Since goal.ly uses Twitter as its delivery mechanism, it made sense for it to also use Twitter for authentication. The Spring Security OAuth plugin was used to provide secure authentication of goal.ly visitors. This allows the use of goal.ly without requiring the user to create a new account.

Twitter exposes an HTTP API to use their services programatically. In particular goal.ly needs to take advantage of the direct message feature. The Open Source Twitter4j library

provides a Java interface over the HTTP API allowing goal.ly to interact with Twitter in a more traditional Java-like fashion.

To communicate with the user, goal.ly creates two threads. One sends new questions to users and the other asks if there have been any user responses. I limited the questions to responses of three types that can be easily parsed. Internally goal.ly uses regular expressions to extract the user answer from the response. If the answer can't be determined, goal.ly currently ignores the user response. Future versions should alert the user that the response was not accepted.

To display the user's history in a chart, goal.ly could create an image of it on the server and deliver it to the user. However, in keeping with the client-side theme, goal.ly is using the JavaScript Google Charts API to render the user history in the browser. The charting is cross-browser and does not require any browser plugins. By doing the rendering on the client-side server load is reduced.

Results

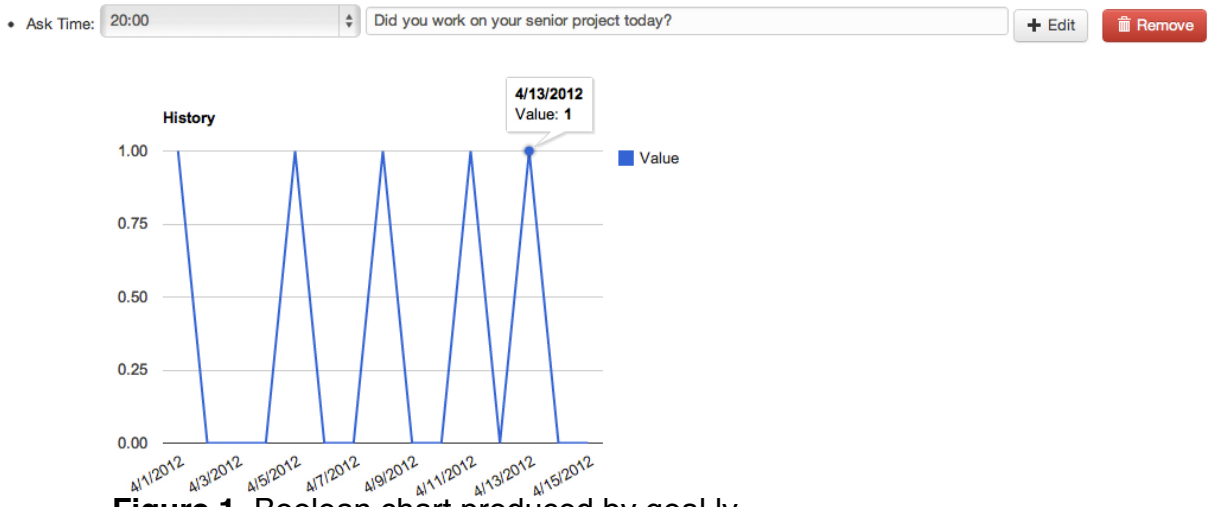

Figure 1. Boolean chart produced by goal.ly.

Figure 1 shows a chart of responses to a boolean question entered into goal.ly. The question will be asked at 8pm every day. The user can edit or remove the question, which will also remove the question history. Below the question is a graph of the response to this question for the past month, with 1 representing true and 0 representing false.

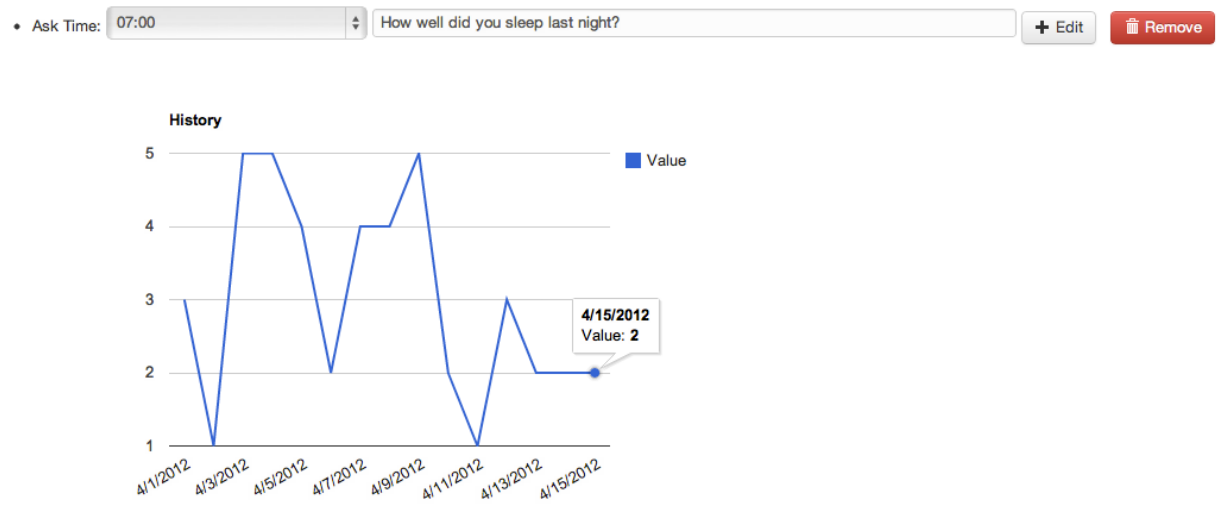

Figure 2. Likert scale chart produced by goal.ly.

Figure 2 shows what the graph looks like for a question that expects an answer provided in the Likert scale.

Summary

For this project I created a modern website using the current best practices. I improved my client-side coding skills, especially with JavaScript. A modern web application can be quickly and easily developed by taking advantage of the vast resources available on the Internet including large collections of Open Source software that are available to today's Software Engineer.

There were some problems with the described design. Twitter was not the best choice for question delivery. Twitter users are limited to 1000 direct messages a day, limiting the potential user base. The UX of the response needs to be improved. Currently the user has to copy and paste a code into the direct message response for goal.ly to be able to collate the response with the question. The login page needs to be changed to match the theme of the rest of the site. It should handle invalid input in a more user friendly manner. I think a better choice for goal.ly would be to revise my stance on user accounts and require the user to enter an email address to use goal.ly. With email as the delivery mechanism, the question length and the total number of

questions allowed could be increased. The collation code could be added to the subject of the email, which by default will be included in the response, improving the UX.

In addition to providing me with a learning experience, I believe that goal.ly is a useful tool in its own right. I plan to continue developing and using it. Given the design of goal.ly, it should be able to support many users concurrently without a high hosting cost.

Acknowledgements

I would like to thank my advisor, Dr. Douglas D. Dankel II, for his guidance and advice while I was completing this project, particularly for the insight that I could add parsing of answers using the Likert Scale with the same routine I was already using to parse numbers. Additionally I would like to thank my wife for her support, encouragement, and patience while I've worked on this project, and my cousin Steven Watson for proofreading this report.

References

[1] Anon, “mozilla/pdf.js”, https://github.com/mozilla/pdf.js, GitHub (as-of 15 Apr 2012).

[2] Anon, “Top 10 reasons to use Gmail”, https://mail.google.com/mail/help/intl/en/about.html, Google (as-of 15 Apr 2012).

[3] Anon, “Web Performance Best Practices”, https://developers.google.com/speed/docs/best-practices/rules_intro, Google (as-of 6 Apr 2012).

[4] Anon, “Best Practices for Speeding Up Your Website”, http://developer.yahoo.com/performance/rules.html, Yahoo (as-of 6 Apr 2012).

[5] Allsopp, C. “Simple, effortless collection of personal data with SMS | AskMeEvery”, http://askmeevery.com, AskMeEvery.com (as-of 6 Apr 2012).

[6] Isaac, B. “Jerry Seinfeld’s Productivity Secret”, http://lifehacker.com/281626/jerry-seinfelds-productivity-secret, lifehacker (as-of 6 Apr 2012).

[7] Anon, “Grails - The search is over.”, http://grails.org/, Grails (as-of 15 Apr 2012).

[8] Beckwith, B. “Grails - Plugin - Spring Security Core Plugin”, http://grails.org/plugin/spring-security-core,

Grails (as-of 15 Apr 2012).

[9] Artamonov, I. “Grails - Plugin - Twitter authentication for Spring Security plugin”, http://grails.org/plugin/spring-security-twitter,

Grails (as-of 15 Apr 2012).

[10] Anon, “Twitter Bootstrap”, http://twitter.github.com/bootstrap/, GitHub (as-of 15 Apr 2012).

[11] Anon, “Backbone.js”, http://backbonejs.org/, Backbone.js (as-of 15 Apr 2012).

[12] Yamamoto, Y. “Twitter4J - A Java library for the Twitter API”, http://twitter4j.org/en/index.html,

Twitter4j (as-of 15 Apr 2012).

[13] Anon, “Google Chart Tools - Google Developers”, https://developers.google.com/chart/,

Google (as-of 15 Apr 2012).

[14] Anon, “HTML5 Rocks - A resource for open web HTML5 developers”, http://www.html5rocks.com/en/,

HTML5 Rocks (as-of 15 Apr 2012).

[15] Anon, “Page Speed Online - Google Developers”, https://developers.google.com/pagespeed/,

Google (as-of 15 Apr 2012).

[16] Anon, “YSlow :: Add-ons for Firefox”, https://addons.mozilla.org/en-US/firefox/addon/yslow/,

Mozilla (as-of 15 Apr 2012).

[17] Anon, “Google Chrome Developer Tools - The Chromium Projects”, http://www.chromium.org/devtools

Chromium (as-of 15 Apr 2012).

[18] Anon, “Comparison of web application frameworks - Wikipedia, the free encyclopedia”, https://en.wikipedia.org/wiki/List_of_web_application_frameworks, Wikipedia (as-of 15 Apr 2012).

[19] Raymond, E. “The Cathedral and the Bazaar”, http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/,

Cathedral and the Bazaar (as-of 15 Apr 2012).

[20] Anon, “Heroku | How it Works”, http://www.heroku.com/how/deploy, Heroku (as-of 15 Apr 2012).

[21] Anon, “Amazon CloudFront”, https://aws.amazon.com/cloudfront/, Amazon (as-of 15 Apr 2012).

[22] Palmer, M. and L. Daley, “Grails - Plugin - Resources”, http://grails.org/plugin/resources, Grails (as-of 15 Apr 2012).

[23] Anon, “JSON”, http://json.org/, JSON (as-of 15 Apr 2012).

[24] Deveria, A. “When can I use... Support tables for HTML5, CSS3, etc”, http://caniuse.com/, CanIUse (as-of 15 Apr 2012).

[25] Koch, P. “QuirksMode - for all your browser quirks”, http://quirksmode.org/, QuirksMode (as-of 15 Apr 2012).

[26] Anon, “GNU Affero General Public License - GNU Project - Free Software Foundation (FSF)”, https://www.gnu.org/licenses/agpl.html, GNU (as-of 15 Apr 2012).

[27] Watson, D. “danielbwatson (Daniel Watson) · GitHub”, https://github.com/danielbwatson, GitHub (as-of 15 Apr 2012).

Appendix A - Technology Transfer Plan

I am making the source code of goal.ly available under the GNU AGPL [26]. It will be available through my GitHub account [27]. Additionally goal.ly itself is publicly available for use as a goal tracking tool.

Biography

Daniel Watson has been developing software professionally for 14 years. He is completing his baccalaureate degree in Computer Science at the University of Florida, Gainesville, FL. He expects to graduate in May 2012, a short 16 years after first starting at UF. His progress was delayed by the first "dot com bubble" and job responsibilities. For fun he enjoys cave diving and keeping up with this ever changing industry.