CSCE 120: Learning To Code

Module 13.0: Organizing Code II
Organizing a Project & Using Git

Using JavaScript Libraries

As developers, we stand on the shoulders of giants. Various libraries and code frameworks have been developed that include the collective talent, experience, wisdom, and hard work of thousands of the best developers. Good libraries contain some of the best, most optimized and rigorously tested code to be found. most beautiful code to be found. Using libraries is integral to software development. Rarely do we ever start from scratch, using libraries speeds development, makes software more reliable, and utilizes the thousands of man-hours of development, testing, and refinement that have gone into the libraries.

jQuery, Bootstrap

Throughout this course we've been using jQuery, jQuery UI, and Bootstrap. These JavaScript (and CSS) libraries that provide functionality, style, and even the ability to customize and define your own themes.

Though the core libraries are very powerful, there are also hundreds of additional *plugins* that extend the functionality of these libraries. For example, jQuery and jQuery UI plugins can be found at http://plugins.jquery.com/. Many more plugins can be found on the web. Some are so useful and sophisticated that they have their own website and support; http://datatables.net for example.

Bootstrap also has its own community of developers and contributors. There are many plugins, themes and templates to be found:

- http://tutorialzine.com/2013/07/50-must-have-plugins-for-extending-twitter-boots
- https://wrapbootstrap.com/
- http://bootswatch.com/

- http://startbootstrap.com/
- http://bootstrapzero.com/

Upgrading & Semantic Versioning

One concern when using a library is designing your code to grow with future versions. As libraries are developed, they may fix bugs, add new features, be made to be more efficient, and other improvements. Upgrading to the latest version of a library can improve your application too. Typically, new versions of a library are *backward* compatible, meaning that upgrading to a new version shouldn't negatively affect your code.

There are exceptions, however. Modern software typically uses *semantic versioning* (http://semver.org/ which uses three numbers to indicate a version. These three numbers correspond to a *Major.Minor.Patch* convention:

- Major – a major version can indicate a major change in the way the library functions and has potential issues with backwards compatibility. A major version can remove support for *deprecated* features, features which were previously announced to be obsolete and will no longer be supported (and which users can expect to be removed in future versions). Alternatively, a library may remove features entirely (such as legacy browser support in the case of jQuery).
- Minor – a minor version indicates improved or added features that are backwards compatible
- Patch – indicates an updated library that includes bug fixes and should be backwards compatible. Note that it may not effectively be backwards compatible if your application was written to work around the bug or if it contained bugs itself that relied on the library's bug. Even with patch upgrades, *regression* testing is necessary (testing to find new bugs or *regressions* caused by changes or enhancements)

When deciding on whether to upgrade a library, it is important to be aware of all the issues and potential drawbacks and to plan accordingly. A well-designed application should be able to grow and accommodate such updates.

HTML5 Boilerplate

HTML5 Boilerplate (http://html5boilerplate.com/) is a template is designed to help developers get started building a web site or application. It offers several features that get you up and running with HTML5 including a "normalization" (or reset) of CSS rules as well as basic styles and styles to accommodate multiple platforms (small screen phones to large screen desktops).

In addition, HTML5 Boilerplate can package popular libraries like jQuery and Bootstrap. It also incorporates Modernizr (http://modernizr.com/, a tool that can detect whether

or not legacy browsers support certain features (such as local storage). In the even that browsers do not support an essential feature of your app, HTML5 Boilerplate can provide *polyfills*. This allows you to build robust applications without having to worry about cross-platform compatibility.

Data Driven Documents (d3js)

D3.js (see http://d3js.org/) is a JavaScript library that allows you to manipulate documents with a *data-first* emphasis. It allows you to bind data to the DOM, then apply data-driven transformations on the DOM. This library has many similarities to jQuery in that it supports selectors, but the selectors can act on bound data as well.

The real power if D3.js is the extensive library of examples and extensions for visualizing data. It has been used extensively by many websites in infographics, journalistic articles, etc.

Others

There are many other popular and emerging libraries and frameworks. Web technologies evolve daily; some examples:

- jQuery Charts (http://canvasjs.com/jquery-charts/) – a jQuery plugin for generating charts and graphs
- three.js (http://threejs.org/) – a library that allows you to build 3D graphics for games and animation
- node.js (http://nodejs.org/) – a server-side platform for JavaScript that allows you to develop in JavaScript on the "backend" of a multi-tiered application instead of inside a browser with a DOM.

Frameworks

There are *many* JavaScript-based *frameworks* (such as Bakcone.js, AngularJS, KnockoutJS, Ember, React, et. al.). Some provide an MVC (model-view-controller) architecture or facilitate developing an SPA (Single-Page Application). Each has their own advantages and disadvantages (as well as critics and evangelists) and development is continually in flux.

The difference between a library and a framework is often referred to as the *inversion of control*. A library provides several useful features that you can use and functions that you can call. However, a framework provides all of the control. With a framework, there is far less programming and far more *configuration*. You configure the framework and end up filling in the pieces that it doesn't provide (in a sense, the framework "calls you" rather

than you calling the framework). Frameworks can greatly reduce the time and effort required to develop a project. However, the inversion of control comes at a cost. Though many standard features and operations become easy, any customization or corner-cases become much more difficult.

Organizing a Project

In many of the activities, we *hotlinked* to various external resources and libraries. Hotlinking is when you load a JavaScript, CSS, or other resource by referencing it with an external URL. For example, if our page is hosted on `http://example.com` and we reference a script hosted on Google's Content Delivery Network (CDN) at `https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js`, then we were using Google's bandwidth and resources to host our script.

Hotlinking has an advantage in that we're not using our own server's resources (bandwidth, etc.). However, there are several drawbacks to hotlinking. We have no control over the quality, security or reliability of the external resource. If the external resource's server goes down, then our application is also unusable. If the external host decides to change the resource or no longer support it, then our application breaks. If the external resource is compromised (say malicious code is injected into it), then the security of our application is also compromised.

Instead, it is best practice to store all resources (scripts, CSS, images, etc.) within your project and host them from the same server. These resources are usually called *assets* and should be well-organized in different folders within your project. Libraries such as jQuery and Bootstrap should be downloaded and stored in the assets folder.

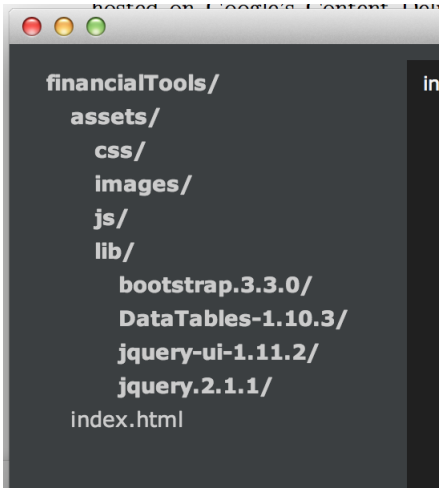

Figure 1: Organizing Assets and Resources

Minifiers

Many programming languages are *compiled* into native machine code that runs directly on the underlying hardware of the computer. JavaScript, however, is an *interpreted* language that executes in a browser (note: many browsers do compile JavaScript to make it execute more efficiently). When a language is compiled all of the high-level language structures that make the code readable and organized are removed and only the necessary machine code (a bunch of binary 0s and 1s) is left.

We can do something similar with JavaScript. For example, comments and whitespace have no effect on the execution of our code and so they do not need to be included in the actual script. If they were to be removed, it could greatly decrease the size of our script and thus save on bandwidth and could be downloaded by the browser faster.

JavaScript *minifiers* do exactly this for us and more. For example, long variable names can be replaced with shorter ones, unnecessary characters can be removed, etc. One of the most powerful minimizers is Google's Closure Compiler (see http://closure-compiler.appspot.com/home). Minifying JavaScript can result in scripts that are 30–50% smaller. Minified code is not only more compact, but has the potential to be faster to interpret and compile. Minifiers also exist for CSS (for example, http://www.cssdrive.com/index.php/main/csscompressor/). There are also entire frameworks for CSS (http://lesscss.org/, http://cssnext.io/) that allow you to use future features and make CSS more programmatic.

Transpilers

A related concept are *transpilers*. A compiler compiles code written in a high-level language into low-level machine code (binary). However, a transpiler *tranlsates* code in one language to another. For example, there are several emerging web programming languages such as TypeScript, CoffeeScript, Dart, etc. Each language is intended to offer features that are not found in JavaScript (such as classes, interfaces, and strong typing of variables). However, widespread browser support for these languages is nonexistent. Instead, a programmer can write their application in these languages, using their features, and then transpile it into version-compliant JavaScript code. Transpilers also exist to allow you to write code in ES6 or later versions and then transpile it to a previous version to ensure wider browser support.

There are other, similar technologies currently in development. For example, ASM (http://asmjs.org/) and WebAssembly are low-level assembly languages for the web which have the potential to take a program written in *any* language and compile/transpile it into a standard assembly language that is recognized by web browsers. This may be the beginning of the end for languages like JavaScript.

Licenses and Legal Concerns

Another major concern when using third-party libraries are legal considerations. Libraries may come with different legal licenses that may limit how you use the library. Some licenses allow you to use the library for non-profit purposes only. Others may allow you to use it as long as you attribute the original source. Yet others allow you unrestricted usage. Some allow you to make modifications, others done. Yet others are completely commercial and require that you pay for the right to integrate or use their code.

When deciding which libraries to use, you want to make sure that you are not violating any copyright or intellectual property laws or you may be subject to civil legal actions. Despite this, most widely used libraries can be used without much concern. It is important to at least be familiar with some of these common licenses.

- MIT License (http://choosealicense.com/licenses/mit/) – essentially an unrestricted license that allows you to do whatever you want. However, it does indemnify the license issuer from any liability. jQuery, jQuery UI, and Bootstrap use the MIT License.
- GNU General Public License (GPL, http://www.gnu.org/copyleft/gpl.html) – a more complex and complete legal license that requires the continuation of the license to any derived products. This its he most common license in the free and open source software (FOSS) community.
- Apache 2.0 License (http://www.apache.org/licenses/LICENSE-2.0.html) – A less restrictive license that allows permission to restrict rights of derived works.

Version Control

Another important concern when developing software is *version control*. As you develop software and make changes, add features, fix bugs, etc. it is often useful to have a means to keep track of changes and to ensure that your code base and artifacts are well-protected by being stored on a reliable server (or multiple servers). This allows you access to historic versions of your application's code in case something breaks.

The solution is to use a *revision control system* that allows you to "check-in" changes to a code base. It keeps track of all changes and allows you to "branch" a code base into a separate copy so that you can develop features or enhancements in isolation of the main code base (often called the "trunk" in keeping with the tree metaphor). Once a branch is completed (and well-tested and reviewed), it can then be *merged* back into the main trunk and it becomes part of the project.

These systems are not only used for organizational and backup purposes, but are absolutely essential when developing software as part of a team. Each team member can have their own working copy of the project code without interfering with other developer's copies or the main trunk. Only when separate branches have to be merged into the trunk

do conflicting changes have to be addressed. Otherwise, such a system allows multiple developers to work on a very complex project in an organized manner.

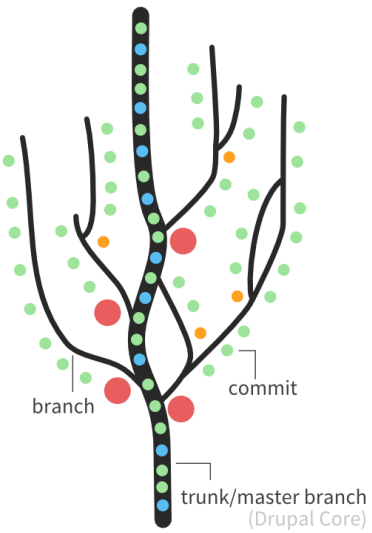

Figure 2: Trunk, branches, and merging visualization of the Drupal project

There are several widely used revision control systems including CVS (Concurrent Versions System), SVN (Apache Subversion), and Git. SVN is a *centralized* system: there is a single server that acts as the main code repository. Individual developers can check out copies and branch copies (which are also stored in the main repository). They also check all changes into the main repository.

Git is a *decentralized* system; multiple servers can act as repositories, but each copy on each developer's own machine is *also* a complete revision copy. Code commits are committed to the local repository. Merging a branch into another requires a push/pull request. Decentralizing the system means that anyone's machine can act as a code repository and can lead to wider collaboration and independence since different parties are no longer dependent on one master repository.

Git has become increasingly popular, but can be difficult to use for a beginner. Fortunately, there are many tutorials and examples available:

- https://guides.github.com/activities/hello-world/
- http://lifehacker.com/5983680/how-the-heck-do-i-use-github
- https://help.github.com/articles/set-up-git/
- http://rogerdudler.github.io/git-guide/

Git itself is free and open source software (FOSS). In general, you would need to setup your own Git server to be able to check code in and share it with others. However, there are many services that offer Git hosting, the biggest and most popular being GitHub.com which offers Git hosting as well as a variety of features. Basic services are offered for free

with full access features coming at a monthly premium (many are free for students).