Ilia Benson

Professor Mushkatblat

Comp 467

4 December 2017

**Draw Something Authoring Tool**

**GOAL**

The goal of this project is to create a simple authoring tool while utilizing 2 frameworks, one for the frontend and one for the backend, while also utilizing technologies such as Dependency Injection, Object Relation Model, Unit Testing, Model View Controller, Single Page Application and RESTful API.

**DESCRIPTION**

This application has a very simple list of features. It can draw a line by point and clicking on the draw button to activate, subsequently a second click deactivates the draw, and then clicking on the white area and drawing any pattern. A palette for color picking. Functional buttons to Undo and Redo actions. A Clear button to clear the work space. A Save button to save your art work as png onto the computer. And finally, an option to change the thickness of the line. These requirements are fairly basic in functionality but the requirements for usability are far more challenging.

The hardest usability requirement was responsiveness to screen sizes. This has been an issue that programmers have suffered through for many years. Luckily Angular solves this very easily by constantly polling the viewable screen resolution and adjusting the work space dynamically based on that. Another changeling requirement was to log every event that transpired between the frontend and the backend. These two entities are completely separated and decoupled. I created a messaging service that would log every API request and call and then display it to a messaging component.

**USER SPEC**

Because this is a web based project and to help in our endeavor to quickly accomplish our goals and minimize time spent doing or building functionality that is not related to the actual content of our project, I will utilize libraries and frameworks. Specifically, one framework for the frontend and one for the backend. I have decided to use Laravel 5.5 as my backend framework because it is written in PHP, has some of the best documentation I have ever seen, has a virtual environment that is an optional download called Homestead and it is the framework I am most familiar with. The virtual environment is a prepared Ubuntu linux box that already comes with everything you need to start building a new Laravel project with 0 configuration. It also comes packaged with many other commonly used package managers and tools such as npm, which we will be using as our frontend package manager, and composer, which is our backend package manager. Aside from Homestead, Laravel is very easy to use and can stand up a simple website very quickly. It is a MVC framework with support for DI, incorporates a great ORM and has many other features such as PHP Unit Test. Additionally it provides many packages developed by the Laravel team to enhance functionality with regards to caching (Redis), authentication (passport with Oauth), and even ecommerce functionality (cashier). All in all it is a one stop shop feature and support wise. Laravel also has its own frontend templating language called Blade, this allows you to generate a fully functional simple website without the need of a frontend framework. Laravel provides support for many other frontend frameworks and also ships with bootstrap to help with all of your CSS needs.

For the frontend framework, I chose to go with Angular 5. Angular is written in typescript so transpiling it into any version of JavaScript to support all kinds of browser hell is fairly easy. Angular is also an MVC framework and thus clashes a little with Laravel. To solve this problem, we will be using Angular for the View almost entirely and Laravel for the Model. Both frameworks will share the Controller portion. Essentially Laravel will route everything to a single point, this point will interact with the rest of the backend, while Angular will utilize all the front facing routing. This is a very nice balance as each framework has a single point of contact between each other, the main route, and are completely separate from each other in all other instances. This approach allows for easy code separation as well as easy feature and concern separation. Additionally, the entire application can be built on the basis of data transportation, the backend can be built without interfering with the frontend as long as both agree on a singular data format and type, which is JSON for this project. Angular has a lot of built in features for manipulating the frontend. The google team has supplied plenty of plugins for the developer to use that make mundane tasks simple, such as taking in a data object and displaying it to the user as an interactive table, this can be achieved with the Material plugin for angular.

For the CSS portion of this project I have decided to make the use of twitter’s bootstrap 4 and SASS. Bootstrap is a great library that does much of the heavy lifting of getting all of your CSS in order to make the webpage look more appealing. It was written in SASS, a preprocessing CSS language, and thus if I need to extend any functionality I can just write it into bootstrap with SASS. All of these frontend libraries will be linked to a CDN at first but then eventually I think I will cache them with Larvel’s Redis support. Once the site is fully built I will migrate the entire stack to a AWS prebuilt Ubuntu container for Laravel/Angular websites. This will minimize all the hassle that usually comes with deployment. The entire project will be revisioned using GIT and it will live on GitHub. All documentation will be done via this document and the supplied Readme.md.

Because this is a drawing application I have opted to go for an SPA. This will allow me to split the concerns of frontend and backend even further as all of the user interaction features will happen on one page, thus limiting the amount of routing I need to do with the frontend to one page, and the backend is already set to a one page route since the MVC is split across 2 frameworks for the same application. Additionally, I have chosen to go with Canvas as my image format, simply because it offers a lot of low level manipulation which usually results in efficiency. Furthermore, there is a great package, ng2-canvas-whiteboard, that does a lot of the heavy lifting of setting up the drawing wire frame for us.

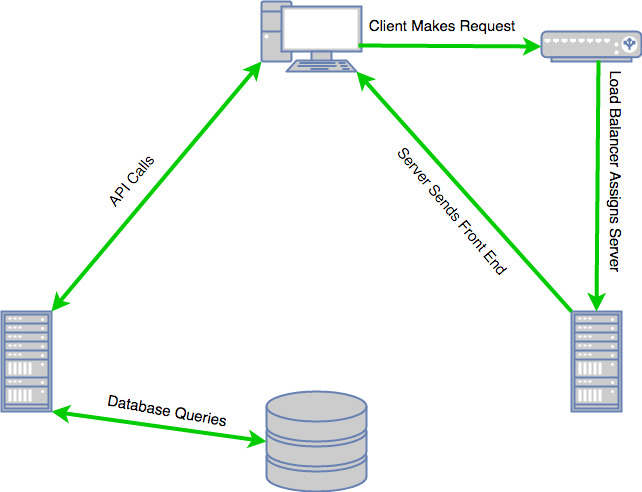
This entire application will then be deployed on AWS using one of their premade Ubuntu linux boxes as well as the Ubuntu one line server installation, lamp-server^. Once installed the apache server will need to be configured and the application placed in the DocumentRoot location. Due to the nature of the free tier of EC2 VM’s it will be impossible to follow the deployment procedure outlined in the readme. There is simply not enough resources to download all the packaged needed for booth frameworks and then compile the Laravel dependencies and transpile Angular into JavaScript. Because of this a second deployment branch has been created that has everything set up for deployment, all that is need is to clone that branch and place the files into a configured DocumentRoot location.

**ARCHITECTURE**

****

**Fig. 1 Client Server Model**

Fig. 1 displays a normal client-server relation. In my project the line width property is stored in the MySQL database and is retrieved asynchronously when the application is launched and when that value is changed. The frontend makes an API call to the backend, which then queries the database via the ORM and returns the result. Similarly, the frontend sends a new value to the backend via an API call to update the value in the database, this too is done via the ORM. Laravel functions as a RESTful API that’s sole responsibility is to query the database and return or store information asked or given to it by the frontend. The communication between the two is passed via JSON. Additionally, middleware can be added to the backend to do more functionality such as authentication and CORS support. Authentication has not been added to this project but CORS support has been.



**Fig. 2 Client Served by Backend**

Fig. 2 displays the Laravel backend is hit first by a client trying to access the website. Laravel serves the JavaScript Angular files to the client. All other communication is then made by the frontend as API calls to the backend. On my hosted website the frontend and backend code are located on the same server in the same root directory. This is not always the case with enterprise applications. It is very typical to have your database, frontend and backend all on different servers and have load balancers direct to servers with the least load that are hosting copies of the same code. This produces a CORS issue where the requesting website address, the frontend, is requesting something on a different website, the backend, and thus the Cross-Origin Resource Sharing error arises. There are 2 popular ways to solve this and still maintain your code spread across multiple origins. The first is to just hard code the PHP to ignore these types of errors which is considered sloppy. The better solution is to have a middleware component accept the traffic and pass it to the backend, shielding the fact that it came from a different origin. This approach allows you to have more flexibility with screening the traffic as middleware can have executable functionality whereas PHP variables do not. The location of the frontend and backend code can be completely separated because neither cares about the other as long as the JSON objects that are passed between the two can be properly interpreted.

**CODE GUIDELINES**

Angular and Laravel both have predefined, code separation enforcing, and naming convention enforcing patterns already built into the frameworks. They both come Command Line interface tools that allow you to easily generate the appropriate components for each in order to create functionality. The templates are generated and assigned predefined functions based on the name given at creation time. For example if I want to create model called Hero and also create a factory, controller and migration for that model I can specify this in the command line tool. When the CLI is building the files it auto assigns names based on type. So, the migrations, which holds multiple instances of Hero, is appropriately named Heroes. Similarly, the factory and controller are aptly named. The functions inside the template files are also generated and labeled with the typical CRUD names. Angular does a very similar thing. Both frameworks enforce good conventions while at the same time saving you time in grunt work. With regards to code separation, both use a component based model that separates each component based on functionality and concern, writing clean code is easy with Angular and Laravel as long as you pay attention to what is you are trying to do where.

For custom functions, I opted to go with camel case as both frameworks already use it. The one irritation between the two was that Angular used 2 spaces as its indentation length and Laravel opted to use 4. I decided to keep both of these conventions as converting one to the other would be pointless since both are supposed to be treated as separate entities as is. When it comes to conventions and coding guidelines the most important convention is consistency, choose one thing and stick with it, this is what I have done.

As a separate tracking tool I included with my application a Log that displays all the interaction events that take place between the frontend and the backend. All the API calls are recorded along with a message that describes whether an update or a fetch is taking place. This tool can be cleared via the clear button or via browser page reload.

**WORKS CITED**

"Amazon Web Services (AWS) - Cloud Computing Services." *Amazon Web Services, Inc.* N.p., n.d. Web.

"Angular Docs." *Angular Docs*. N.p., n.d. Web.

"Build Software Better, Together." *GitHub*. N.p., n.d. Web.

"CakePHP - Build Fast, Grow Solid | PHP Framework | Home." *CakePHP - The Rapid Development Php Framework.* N.p., n.d. Web.

"Canvas API." *Mozilla Developer Network*. N.p., n.d. Web.

"CSS." *Mozilla Developer Network*. N.p., n.d. Web.

"Dependency Injection." *Wikipedia*. Wikimedia Foundation, 03 Oct. 2017. Web.

"Django." *The Web Framework for Perfectionists with Deadlines | Django*. N.p., n.d. Web.

*Doxygen: Main Page*. N.p., n.d. Web.

Foundation, Node.js. *Node.js*. N.p., n.d. Web.

*Git*. N.p., n.d. Web.

Gskinner.com, Inc. "A Suite of JavaScript Libraries and Tools Designed for Working with HTML5." *CreateJS*. N.p., n.d. Web.

"Hypertext Preprocessor." *Php*. N.p., n.d. Web.

"Identity Infrastructure, Built for Developers." *Auth0*. N.p., n.d. Web.

"Introduction to the DOM." *Mozilla Developer Network*. N.p., n.d. Web.

"JavaScript." *Mozilla Developer Network*. N.p., n.d. Web.

"LAMP (software Bundle)." *Wikipedia*. Wikimedia Foundation, 07 Oct. 2017. Web.

Mark Otto, Jacob Thornton, and Bootstrap Contributors. "Bootstrap." *Bootstrap · The Most Popular HTML, CSS, and JS Framework in the World.* N.p., n.d. Web.

"Model–view–controller." *Wikipedia*. Wikimedia Foundation, 08 Oct. 2017. Web.

*MySQL*. N.p., n.d. Web.

"OAuth.net." *OAuth Community Site*. N.p., n.d. Web.

"Object-relational Mapping." *Wikipedia*. Wikimedia Foundation, 06 Oct. 2017. Web.

"The Official Microsoft ASP.NET Site." *The Official Microsoft ASP.NET Site*. N.p., n.d. Web.

Otwell, Taylor. "Laravel Homestead." *Laravel Homestead - Laravel - The PHP Framework For Web Artisans*. N.p., n.d. Web.

Otwell, Taylor. "Love Beautiful Code? We Do Too." *Laravel - The PHP Framework For Web Artisans*. N.p., n.d. Web.

"React." *A JavaScript Library for Building User Interfaces*. N.p., n.d. Web.

"Single-page Application." *Wikipedia*. Wikimedia Foundation, 05 Oct. 2017. Web.

"Syntactically Awesome Style Sheets." *Sass*. N.p., n.d. Web.

Team, The Core Less. "Getting Started." *Getting Started | Less.js*. N.p., n.d. Web.

"Vue.js." *Vue.js*. N.p., n.d. Web.

"Vue.js." *Vue.js*. N.p., n.d. Web.

*W3C HTML*. N.p., n.d. Web.

"W3C." *W3C SVG Working Group*. N.p., n.d. Web.

"Welcome to PHPUnit!" *PHPUnit – The PHP Testing Framework*. N.p., n.d. Web.

"Welcome to Python.org." *Python.org*. N.p., n.d. Web.

"Zero Downtime PHP Deployment." *Envoyer - Zero Downtime PHP Deployment*. N.p., n.d. Web.