COM S 319

Portfolio II

Hope Scheffert & Jacob Neyens

TABLE OF CONTENTS

OVERVIEW

Note: The goal of this portfolio was to experiment with AngularJS and PHP. We have been extremely behind on the labs all semester, so we knew for this portfolio we needed to step it up so that we can make up for our not-so-great labs. Unfortunately, we were unable to work on the AngularJS lab before submitting this. Therefore, this project was very difficult because we had no introduction with the lab to kind of warm us up to the framework. Jacob and I spent hours on end watching and reading tutorials on AngularJS to familiarize ourselves in order to put together a good portfolio.

First we started off experimenting with AngularJS and how it works by making simple custom directives and using the provided directives such as ng-repeat, ng-cloak, ng-click, etc. Then we decided to take it up a notch and develop a SPA which was based on the app Trello that both Jacob and I use daily. Here we were able to use Angular routing and store data in local storage to keep a list of tasks "to do" for a user with a "priority". Our goal was to make a custom directive that had a template which showed a new empty list (just like Trello), but after hours of research and a discussion with Mitra, we decided that it was too complex for this moment in time, and that we were better off sticking to something we were actually able to do. One of the things we wanted to do with our last portfolio but didn't have a chance to was implement some CSS styling to make our page prettier. This time, we used a W3Schools CSS script to do that for us. It definitely makes our SPA look professional and clean, which we are proud of.

After these shorter experiments, we were looking for something more complex and new to do with AngularJS. After yet some more

research, we decided to do AngularJS form validation. But what kind of form? Well, with the election in full force, we thought it would be cool to make a form that simulated a voting ballot for our Presidential Election. Then, Jacob discovered an AngularJS graph that we used to display the results of the election to our user.

We had some issues while using these graphing APIs. The first one we tried to implement was called "FusionCharts". It is a widely used graphing software that is used by major companies like Apple, LinkedIn, Facebook, and even by the United States Government for their Federal IT Dashboard. We thought this was a great choice to produce the graphs we wanted, but after testing we realized we had to switch to something that would not require a commercial license, such as Google Charts.

Because we had just recently gone over regular expressions during lectures, we decided we could use ng-pattern with our regular expressions instead of, for example, the "email", "tel", or "date" input types which are already taken care of for you if specified. We actually started off using those, but changed it in order to get some practice working with regular expressions.

NEW & COMPLEX ISSUES

1. Custom Directives

a. To start out with, we did some simple experiments using a separate template file and displaying an array of json objects in both a list and a table with ng-repeat. We also used the link function to add event listeners that manipulate the DOM and grab parts of the scope object to

show a message. These experiments are in *directives.html, directives.js, firstDirectiveTemplate.html,* and *secondDirectiveTemplate.html*.

b. (Note: Below I will explain in more detail the the work I did with making the Trello-like SPA). In this particular experiment, I wanted to take advantage of the fact that a directive is a one-time template that can be injected and used whenever needed, which prevents duplicate code. My goal (like Trello) was to allow a user to click a button to create a new list and name it whatever they wanted, and add to it. This is where the custom directive came in. I googled and messed with it for hours, but I was unable to figure out how to make the template so that we could implement a button that made a new list. Jacob and I talked with Mitra and he said that what we were trying to do was very complex. After being stuck for so long, we decided to move ahead with something else due to time constraints and frustration. However, I included my experimentation code in *listDirectiveExperiment.html.* Though incomplete, maybe you can look at what I did or see where I went wrong.

2. Routing

a. After making the Trello-like list and failing to make a directive to dynamically create lists on a single page, I decided to take up Angular Routing to give the functionality of having multiple To Do lists. So instead I made a SPA that represents a task manager for 3 classes. Each tab is a class with a To Do list that the user is able to

add a task and grant the task a priority level. Priority 1 is a task that must be done as soon as possible, when a user adds a task and it is priority 1, I give that div `class="priority1"` which is defined in our CSS to have a background color of red, signifying that is is most important. The same applies for priority 2, which appears orange, and finally priority 3 which is yellow. It actually took me a really long time to get this working correctly because if I added tasks to one class to do list, then navigated to another class, upon return the data was deleted. After some research, I decided that storing a "task" array in localstorage was the best way to do this. Now, each time you go to a new class, the list is repopulated automatically from the local storage object, which stores json stringified "task" objects. The code for this experiment can be found in *routingExperiment.html, moreRoutingExperiments.html, main.html, first.html, second.html,* and *third.html* with the most complex code (using task objects instead of strings) in *moreRoutingExperiments.html*.

3. Form Validation/Regular Expressions

a. This was by far the most time consuming of our experiments. I decided to use angular form validation to create a simulated presidential election. I have actually never voted before so I'm not sure what information is needed to do so, but for the sake of practice I made several fields to validate. The most important fields, of course, are the voter's social security number and their

choice of candidate (as well as their age to make sure they are at least 18 years old). To begin with, I used input types such as email, tel, and date--but after going over regular expressions in lecture, I decided to get some practice writing those. I used ng-pattern for most fields to match regular expressions. First and last name must be alphabetical only which is matched by "`/^[a-zA-Z]*/`".Social security number and phone number are matched as "`/^(\d{3})[-]?(\d{2})[-]?(\d{4})$/`" and "`/^(\d{3})[-]?(\d{3})[-]?(\d{4})$/`" respectively. The `[-]` matches optional dashes for formatting. Email is matched by "`/^[a-zA-Z]+[a-zA-Z0-9._]+@[a-z]+\.[a-z.]{2,5}$/`". This one was a little more difficult. We didn't want to be too picky, so we allowed all but symbols and for the domain name and such we disallow capital letters because I've never seen those in email addresses. Finally, we wanted to keep date uniform so we only match the format MM/DD/YYYY which is matched by "`/^\d{2}([/])\d{2}([/])\d{4}$/`" which is very similar to the phone number and ssn. If we wanted to match spaces, dots, or dashes in between the digits, we could just substitute `[/]` with `[ /-.]` instead. The last input is an angular select drop down which uses `ng-switch-when` similar to a switch/case block. When a candidate is chosen, a div appears with their name, party, and picture. Finally, the voter clicks on the vote button and the form is checked for valid inputs. If it is not, it will notify the voter what they need to fix. If the form is valid, our custom service

will update our json file with the new vote count. The form validation can be found in *angularFormView.html*.

4. Custom Services/PHP

a. On the back end of the form, we needed to account for voters and votes. We decided to make two json files and use PHP to send and retrieve information from the server. We made three PHP files: *addVoters.php*, *sendVotes.php*, and *getVotes.php*. In order to use these, we made two custom services: votes and voters. Each time a form was submitted and valid, we had to first check the *voters.json* file for another voter with the same ssn (since everyone has different social security numbers). If they were not found in the file, we added them and returned so that we could make a call to *sendVotes.php* to increment the votes object by 1 vote to the specified candidate. The reason we used the json files is because we wanted to be able to easily get the votes and voters in order to display the results of our simulated election in a graph. The custom services and controller can be found in *angularjsForm.js*.

5. Graphing Results

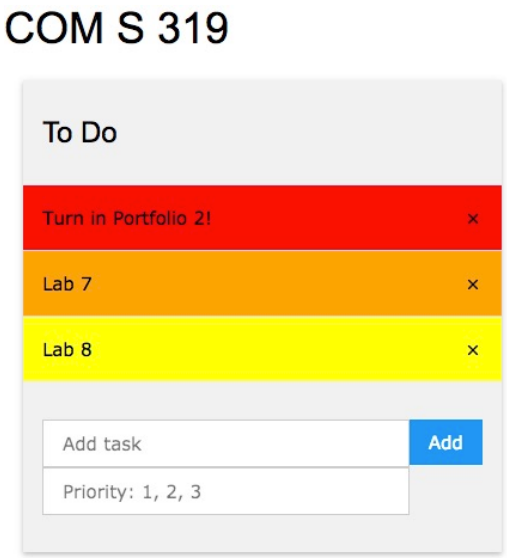
a. This may have been some of the most frustrating part of our portfolio. We researched different kinds of graphing API's to find something that would work with AngularJS. The first one that caught our attention was FusionCharts and we decided to roll with it. It is a widely used API and seemed somewhat simple to learn since neither of us had ever used it. After figuring it out and having a nice looking

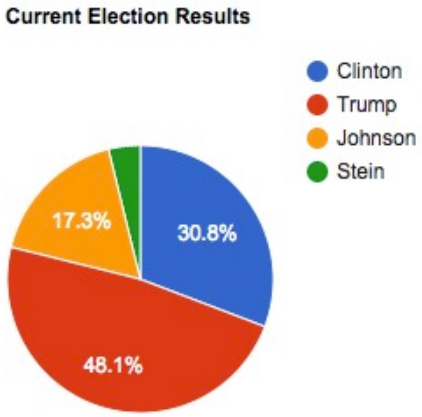
graph with some dummy data, we decided to run it on our XAMPP servers and try to pull stored data from a JSON file. And that's where it started to get frustrating. For some reason the graphs would not generate when running the server. After troubleshooting for a while we found that FusionCharts has a free service (which we we're using) and they also have a commercial service that can be purchased. The free service can't be used on a server, so we had to find another method. We decided to transition to using Google Charts instead, which is a completely free service.

BLOOM'S TAXONOMY

1. *Creation*
    a. *Develop:* The creation of these experiments gradually got more difficult as we started adding on more features to simpler code the more familiar we got with angular. We were looking to create something that would challenge us. We decided that using AngularJS would be difficult and give us the challenge we were looking for, since we had not done the lab for it yet, and we were able to expand on what we had learned in lecture.
    b. *Design:* As stated earlier, one of the things we wanted to do after working on our first portfolio was make our webpages prettier with CSS. We know that this isn't an important factor when it comes down to it, but you have to

admit, the list's look pretty fancy with some CSS instead of just plain bulleted list in an ugly standard html page. This makes it more fun to show someone what we're doing in the class, since real websites are always pretty. The graphs are also generated with nice looking colors to give some pop and not look ugly. We did some research to figure out the color of each of the candidate's respective parties (Libertarian Party = Gold, Green Party = Green, Democratic Party = Blue, and Republican Party = Red) and used those for their slice in the pie graph.





2. *Analysis*

   a. *Compare:* In our opinion, the difficulty level of this portfolio is much higher than our first. We feel that we have learned beyond what we had in class, and hopefully lab 7 should be easy after finishing this. We were also able to integrate regular expressions into our form validation, which was very last minute, but we saw it as a good opportunity for some variety.

   b. *Criticism:* There were a couple of times that we got in over our head when trying to do something too difficult too fast. Mitra explained it as we were "jumping too far." An example of these times were right at the beginning when we were trying to make a custom directive to display a to do list or at the end when we were using google-charts to display graphs of the votes. We would try to follow examples and build on them instead of actually learning what was going on. After learning how AngularJS functions with custom directives it was much easier to design what we intended to do.

3. *Evaluation*

   a. *Judge:* Overall, our Portfolio II is much better than our Portfolio I in many ways. We dove deeper into the coding and really researched about AngularJS. I would say that it is the most complex topic/language that we have learned thus far and while we have learned a lot, we still didn't accomplish what we set out to do.

   b. *Argue:* Because we haven't even touched lab 7 (AngularJS lab) yet, we had little introduction or warm up

to the framework which forced us to learn on our own and research what exactly can be done with AngularJS. In a lot of ways, this limited us. But given the time we had and the fact that we had to learn something as complex as angular, we are very satisfied with our set of experiments. We do not consider ourselves wizards at programming, but we try really hard. We practically live in either Mitra's office, or the Pearson help room and office hours!

c. *Evaluate:*

   i. The hardest part was definitely working between services, controllers, and the DOM. We had a very difficult time trying to figure out how to communicate between all three of these. There are a lot of rules that we didn't grasp about AngularJS, such as services not having access to the scope, therefore it makes it tricky to use the service for data but then getting it back to the controller to update the DOM.

   ii. We also had trouble when graphing because we needed to make an asynchronous http request to obtain the votes object and populate our pie chart. We ended up having to go see Mitra for help and after taking it home and debugging it, he was able to help us tremendously by showing us a different way to do this by using "promises". This allows the request to wait for the data to come back and then save it to draw the graph.

iii. An Example of Using Promises:

```
118
119  //Make a request to get data and then wait using a "promise"
120  var myPromise = new Promise(function(resolve, reject) {
121      $http.get("getVotes.php",{cache: true})
122      .success(function(data) {
123          resolve(data);
124      });
125
126  });
127  //Once get the data, save it for the graph
128  myPromise.then(function(data) {
129      votesObj = data;
130      clintonVotes = votesObj.clinton;
131      trumpVotes = votesObj.trump;
132      johnsonVotes = votesObj.johnson;
133      steinVotes = votesObj.stein;
134
135      modifyGraph();
136      // this function makes $watch work
137      $scope.$apply(function(){});
138  })
```

CHECKLIST

- I included code that I had tried to make work but couldn't figure out how to make the custom directive. I receive an injector module error, for example, and I wasn't able to figure out what caused it. This is in *listDirectiveExperiment.html* and *my-list.html.*
- Take a look at *angularjsForm.js* which contains a couple of custom services explained above, and also using a promise in order to wait for our http request for data and finally graphing it on a pie chart. These were the most difficult concepts and we needed Mitra's help for some of it.
- Note in first.html (for example) when displaying the list, using `ng-click="removeTask($index)` which means that when that specific task is clicked, it calls the remove function in *moreRoutingExperiments.html* to delete that index in the array of tasks.
- Note at the very end of *portfolio2.html* we use "bootstrap" to display two different angular modules in one page (we found this technique with some research and had to use it again when displaying the pie chart with google charts)

**Note that in the php files, you may need to change

```
$path =
"/Applications/XAMPP/xamppfiles/htdocs/portfolio2/voters.json";
```

To whatever path the files are when grading.