Lab 5.1: Introduction to Angular.js
IN705 Databases Three

Introduction

Now that we have a REST/JSON service available to handle our data, it is extremely easy to create clientsthat interact with it. One way to create such a client is to use one of the many imodern JavaScript frameworks available. Angular.js is one example of such a framework and it is the one we will introduce in this lab.

We will use Angular.js to handle the HTTP calls between our web client and our REST service, but before we do that we will familiarise ourselves with the basics of the framework.

1 Clone the sample code from GitHub

An repository containing a basic code skeleton is available of GitHub. You can download the code with the Git command

```
git clone git@github.com:tclark/splatter-client.git
```

You will see that the repository contains the following files

index.html An HTML page including the basic directives to connect our Angular.js app to our HTML.

app.js A JaveScript file in which we will place our client application code. It also includes some sample data.

angular.min.js, angular-resource.js Angular libraries on which our application depends.

2 Review the sample code

First, look at the app.js file. You will see that we create a *module* called "splatter-web" and that we attach a *controller* called "UserController" to our module. We will add code to this controller later. You will also see that we have some sample data in basic JavaScript data structures.

Next, look at the index.html file. Notice how we have associated our Angular.js module with the page using the `ng-app` directive at the top of the file. We also have three script tags to load the required JavaScript code files.

Finally, observe how we have connected our `UserController` with a `div` element on the page. This means that properties of the controller will be accessible only within the associated `div`. We have also created a local label, `user`, to refer to our controller within this scope.

3 Hello, Angular world

We will begin with a simple example. Add the following line to your controller:

```
this.hello = "Hello, world";
```

Now your controller exposes its `hello` property.

Add the following line to controller's scope in the `div` in `index.html`:

```
<p>{{user.hello}}</p>
```

Save both files and view your web page in a browser. Now you can see how properties of the controller can be passed through to an HTML page, which is most of what you need to know.

4 Displaying user information

For the time being we have a sample user available in our JavaScript file. Associate this user data with our controller by adding the line

```
this.u = u1;
```

Now we can access the user's properties in our HTML file with the variables `user.u.name`, `user.u.email`, and `user.u.blurb`. Add HTML and Angular expressions (the code in double curly braces) to your index.html file to show our user's properties.

5 Displaying the splatts feed

We also have a sample news feed in the form of an array of splatts. Assign the value of this array to you `UserController`'s `feed` property. We can display the feed on our HTML page, but we need a way to loop over an array.

In your `index.html` file, add a new `div` like the one below.

```
<div ng-repeat="splatt in user.feed">

</div>
```

Angular will create one div for each member of the `user.feed` array. Inside the div you can access the individual splatt attributes with the local variable `splatt`. So, for example, you can display the splatt body with the expression `{{splatt.body}}`.

The splatts have a `created_at` attribute, so it's worthwhile to look at how we can use a *filter* to controll its display. Angular has a `date` filter that can read in timestamps is a variety of formats and then display them in another format. For example, we can apply it to our splatt timestamps like this:

```
{{splatt.created_at | date: 'dd/MM/yyyy h:mma'}}
```

There are filters available for many kinds of data.

6 Adding a form

So far we have dealt with displaying information. Now let's see how to enter information and work with it using Angular.

Add a simple form to your `index.html` file like this:

```
<form name="myform">
  <input type="text" />
  <input type="submit" value="Update user" />
</form>
```

It's a simple form that doesn't do much, but we want to make it update the user's name that we display on our page. To do this we need to

1. Bind the form to a controller;
2. Direct the form to a method of this controller;
3. In the controller method, take the form data and use it to update the user.

To bind the form to a controller, add the following attributes to the `form` tag.

`ng-controller="UpdateFormController as form"`

`ng-submit="form.updateUser(user)"`

Then bind the text input to a controller attribute with the directive

`ng-model="form.data.name"`

in the text `input` tag.

Now the form is ready to work with a controller. We just need to add the controller to the indicated spot in our `app.js` file. Here is the code for the controller.

```
app.controller("UpdateFormController", function() {
    this.data = {};
    this.updateUser = function(user) {
        user.u.name = this.data.name;
        this.data = {}; // clears the form
    }
 });
```

Verify that your form works and that it updates the user's name on the page.

7 Conclusion

Angular.js provides a lot of power to build a dynamic and responsive web application in a pretty manageable way. You should take some time to play around with Angular. Documentation and tutorials are available at `http://angularjs.org`.

Next week we will see how to connect our web page to our REST service using Angular.