Lab 5 (DH2642 only): Angular framework

Goals

- use local web server to run the application
- use Angular framework
- use different routes (address) for different screens even though it will remain a single-page application
- persist the information between page reloads

Assignment

The goal of this lab session is to adapt your application to use Angular framework. Angular is a framework that supports MVC-like separation and offers many other features that makes building web applications faster and less error prone. In ideal projects you would chose a framework in the beginning and work on it from the start. However, as frameworks constantly change and new ones come out, our goal with previous labs was to give you the understanding of the basics. Once you know that, learning any new framework should be easier.

Step 0. Setup

In this lab we are going to use local web server to run the code. Even though we are still going to work on the client side code (no server side programming) there are certain limitations when you open HTML files directly in the browser. The main issue is that you are not allowed to load other files through JavaScript. Since Angular uses this method to separate your view code in several files (more on this later), we need this feature. Therefore we are going to setup a local web server and some other useful tools.

1. Get and install node (if you do not already have it). The computers in the lab rooms should already have it, you will just need to do "module add node" to activate it (every time you start the terminal).
2. Get the fresh copy of the base code and checkout lab5 branch
    - git clone https://github.com/kth-csc-iprog/dinnerplanner-html.git dinnerplanner-lab5
    - cd dinnerplanner-lab5
    - git checkout lab5
3. Once in the code directory, you should run **npm install** in your command line which will install the web server and other needed components
4. After the install has finished you can run the web server and base code with **npm**

start

5. You should now be able to go to http://localhost:8000/ and see the base code running

And this is it. You are now ready to start implementing the dinner planner in Angular. To understand how the lab5 code is organized here is a quick overview:

- app - contains the application code
- app/index.html - the main layout file that contains the code shared among all the screens
- app/partials/ - here you can find the partial HTML files that contain view specific layout code
- app/js/ - contains all the JavaScript files that contain the application code (controllers and model)
- bower.json, package.json, etc. - the files in the root of the folder serve for configuring your setup. In other words, the "npm install" and "npm start" commands use the information in those files to install the web server and other dependencies (like bootstrap, angular, etc.) and run the web server. For this lab you will need to slightly modify one of these files only in Step 4. After running the install command there will be two other folders created (bower_components and node_modules) that will contain the dependency libraries. Feel free to explore to see what is there.

Step 1. Understanding the Angular framework through the code example

The base code provides the start of the dinner planner application. If you run it, you will get the home screen with "Create new dinner" button which will than lead you to the second screen which will just show number of guests and allow you to edit it. This basic example already demonstrates the main concepts:

- model (number of guest)
- views (e.g. home and search, defined through templates)
- controller
- the startup code

The start of your application is index.html. Notice how **ng-app** directive is added to the <html> tag. This tells Angular where the application starts. It also defines which module (dinnerPlanner in our case) contains the main application code. If you look in app.js you will notice how the dinnerPlanner module is created in the first line of code. The other addition to index.html is the **ng-view** that tells Angular where to load each view. As noted earlier, index.html server only for containing the overall shared layout of your application, while the individual views will be loaded from other files.

Since we are using Angular we are going to take the advantage of it's **routing** capabilities to have a fully modern Single-Page application. In the previous labs you were just showing and hiding the DOM elements as the view was supposed to change, but the address in the browser stayed the same. With Angular each of our views is going to have a different

address, which will give us the possibility to use back and forward buttons in the browser to navigate through the application. For instance, you can notice if you go to http://localhost:8000/ that Angular changes the address to http://localhost:8000/#/home. Similar if you click the 'Create new dinner' button it will change to http://localhost:8000/#/search

This functionality, or in other words the views (html templates) and condition under which each of them is loaded, are defined in the rest of the code in app.js.

The **views** are defined in HTML **templates** (also called partials). You can find them in the partials/ folder. If you check the home.html you will notice the simple link (styled as button with bootstrap) that links to the "#/search". Nothing Angular specific there, however Angular will interpret the address change once the button was pressed and load the search.html view according to what was defined in app.js. The search.html at the moment has only **ng-include** directive which loads the dinner-sidebar.html template. Since the sidebar is present on two screens in our prototype (search and single dish) we put it in a separate template. Looking at dinner-sidebar.html we can see it contains the number of guests input box and text and some Angular elements, which we will address shortly. Back in the search.html we can also see the **ng-controller** directive that associates the DinnerCtrl (controller) with the included sidebar view.

The DinnerCtrl **controller** code is located in dinnerCtrl.js file. As in previous labs, the controller in Angular is the link between the view and the model. However, the difference here is twofold. First, we do not create it as a normal JavaScript object, but instead use Angular controller method on our dinnerPlanner module. The second difference is that we do not access directly the DOM elements in the controller (you should **not have** any $("something") calls in the Angular controllers). Instead we use the $scope service that we pass to the controller constructor method. Any object or method that defined on the $scope object we can then use in our views. If you look back a the dinner-sidebar.html you will notice we use **ng-model** directive to create two-directional binding to $scope.numberOfGuests object in our controller (when using objects and methods in the view, the '$scope.' is omitted, i.e. implied by Angular). If we would in our controller code change the value of that object, the view would update automatically. Furthermore, since we used the ng-model directive (that is typically used on input or select HTML elements) our if we change the value in the input box, the $scope.numberOfGuests will change as well. To just print the values from the $scope objects or methods you can also use the '{{ expression }}' as it is done with {{getNumberOfGuests()}} methods.

Another service that we passed to our DinnerCtrl constructor is the **Dinner** service. This service represents our **model**. We define the model in the dinnerService.js. The model is the core of your application. It is the application logic. Therefore it makes sense that the model should not change much depending on which framework you use. In fact you will copy most of the model code from previous labs. The main difference between the previous lab (where model was just a JavaScript object) is that Angular has a special concept for wrapping your model code (and other similar code) called services. Angular

service can easily be used in other parts of Angular code (like we do with Dinner in the DinnerCtrl) and Angular insures that is is created only once (when it is first time used) and reused the rest of the times. You can check the Dinner service code and notice the familiar get/setNumberOfGuests methods that were used in the controller.

Step 2. Model and REST APIs the Angular way

The dinnerService.js already contains the code that creates the service and the example methods for number of guests. You can now copy your model code from the previous lab and insert it into the dinnerService.

The first difference we will need to make in our model is to remove the Observable pattern. Since we use Angular it takes care of watching for changes the elements we bind (through $scope) in our views, so we do not need the pattern anymore.

Task: remove all the Observable code (don't forget the calls to notifyObserver in other methods)

Another important difference will be how we access the BigOven APIs. Instead of using low-level jQuery ajax method, we will use Angular's $resource service. This service simplifies the access of REST APIs. The $resource service will give us an object which we will then use to make the API calls.

We will create $resource object in our model for 2 API calls we need to make:

```
this.DishSearch = $resource('http://api.bigoven.com/recipes',
{pg:1,rpp:25,api_key:'YOUR_API_KEY'});
this.Dish = $resource('http://api.bigoven.com/recipe/:id',
{api_key:'YOUR_API_KEY'});
```

Now, in the controller, if we want to search for dishes we can call Dinner.DishSearch.get({title_kw:'chicken'}) or to get a single dish we would do Dinner.Dish.get({id:12345}).

Since we have this new way of getting the data from the API, you should remove the getAllDishes and getDish methods as they are not needed anymore.

Step 3. Implementing the views and controller

In this step we need to implement the views and controllers. For both of these you will probably be able to reuse bits of your code from the previous labs, but some modifications will be needed.

Let us start by shown an example of how to get the dishes in the search controller. Let us say we want to have a *search* method on our scope that we will call when the Find button is pressed. The simplest thing we could do in that method is to say:

```
$scope.dishes = Dinner.DishSearch.get({title_kw:query})
```

however that would not give us the best result as we would not take care of the case when there is error or the moments while the data is being retrieved (as we did in Lab 4).

Therefore the full search method can be defined as:

```
 $scope.search = function(query) {
   $scope.status = "Searching...";
   Dinner.DishSearch.get({title_kw:query},function(data){
     $scope.dishes=data.Results;
     $scope.status = "Showing " + data.Results.length + "
results";
   },function(data){
     $scope.status = "There was an error";
   });
 }
```

The *get* method of our DishSearch $resource can have as arguments two optional methods. First one is called if there is success, while the second one is called in case of error. Instead of modifying the view directly to show and hide the status information as in Lab 4, here we just set the scope variable. We could have also have two different variables, one for success message, other for error message and then in our view show them differently (i.e. show the error message in red).

Let us now implement the search view (search.html template). We need the search input box (we will ignore the type of dish drop down as this is not supported by BigOven) a search button and a list of all the results. You should try to use, as much as possible, the view code from the previous labs, however we will show here a very simple example:

```
<input ng-model="query"/>

<button ng-click="search(query)">Find</button>
{{status}}
<ul>
 <li ng-repeat="dish in dishes">
 <img ng-src="{{dish.ImageURL}}"/>
 {{dish.Title}}
 <a ng-href="#/dish/{{dish.RecipeID}}">View more</a>
 {{dish.Category}} {{dish.Subcategory}}
 </li>
</ul>
```

Note: if your page is not changing after the refresh even though you changed (and saved) the HTML partial your browser might be caching the partials. In the developer tools in Chrome, under the Network tab you can disable caching.

What we needed to add to our view code to make it work with our controller is:

- ng-model to our **input** so we can use the value of the input in our click method
- ng-click to our **button** to call the search method on the $scope
- {{status}} to display our status message depending if the search is in progress. If

you want to have a more complex notification here (for instance showing some animation while search is in progress or red error message in case of error) you can for example use ng-show/ng-hide and create boolean $scope objects to control which one is shown/hidden.

- ng-repeat to iterate over all the elements in the $scope.dishes
- ng-src with {{dish.ImageURL}} to show the dish image
- ng-href with {{dish.ReciepeID}} to link to the dish details view
- other {{dish.SOMETHING}} expressions to show dish information

If you test your code now on http://localhost:8000/#/search, you should be able to get the search results from BigOven API.

You should now have enough of building blocks and understanding of angular framework to implement the remaining views and controllers. We have created the JavaScript files for all the needed controllers, however you will need to create (at least) two more partials for the last two screens (dinner overview and preparation instructions). Hint: you can reuse the same controller on different views if you need similar functionality on them.

Also, remember that you can have different controllers on several DOM elements in the same view and they can be organized hierarchically.

Step 4. Store the information on page reload

As the final improvement of our Dinner Planner app we will add a bit of persistence. We want to achieve that when a page is reloaded the data previously entered is not lost. To achieve this we will use the cookies to store the information that the user selected. This will, of course, mean that it's only limited persistence because on different computer (or even just different browser) the cookies will not be the same. However, cookies are important part of dynamic web (that's how your gmail, Facebook etc. stay logged in when you open and close your browser).

To implement the functionaliy you will need to save the number of guests and selected menu in the cookies. Luckily Angular provides a convenient way to do this by using $cookieStore service. However we have not preloaded this model in the Lab configuration so that you can learn how to add other modules (and other JavaScript libraries) to your project. In Lab 5 project we use Bower package manager for managing JavaScript files. To add the needed JavaScript files you will need to:

1. Modify the bower.json and add "angular-cookies":"1.3.x" to the list of dependencies
2. Stop the web server you have run in step 0. and re-run **npm install** - this should download the needed javascript and add it in your bower_components folder
3. Include the JavaScript in index.html
4. Rerun the server **npm start**

You can now reload your application in the browser to make sure that there JavaScript file was loaded correctly (check the console).

Now that you have the JavaScript it's time to tell Angular you want to use it. You need to

1. Modify the app.js to include the "ngCookies" module in the dinnerPlannerApp creation
2. Add the $cookieStore in the dinnerService.js as the dependency in the constructor

Now you are ready to implement the feature. The implementation should be rather easy. In your model (dinnerService.js) you need to make sure that:

- whenever you are changing the number of guests or menu you also store those values to the $cookieStore
- you load the values from the cookie store the first time you create the variables that store your number of guest and menu

The only tricky part here is that there is limit to how much information can be stored in a one cookie. Therefore, if you your menu variable contains the full dish object (instead of just RecipeID) you will not be able to directly save that variable in the cookie, but you will instead need to save only the IDs.

What to deliver on 11th March

When you have finished with developing, just push your changes to GitHub and present your results to assistants in one of the Lab sessions.

Visa tidigare händelser (5) ›

Elizaveta Sigova kommenterade | 26 februari 12:22

Hi Filip!

Thanks for the help! It works now :)

I misinterpreted this line: <ng-include src="'partials/dinner-sidebar.html'" ng-controller="DinnerCtrl"></ng-include>

I guess I need to find out what ng-include does :)

/Elizaveta

... eller skriv ett nytt inlägg

Alla användare med KTH-konto får läsa.

Senast ändrad: 2015-02-19 11:47. Visa versioner

Taggar: Saknas än så länge.

Följ denna sida Anmäl missbruk

http://www.kth.se/