CREATING ANGULAR APPLICATION

By

Stefan Popov

Contents

[1 Angular application 4](#_Toc460416746)

[1.1 First steps 4](#_Toc460416747)

[1.2 Starting with AngularJS 4](#_Toc460416748)

[1.2.1 Modules 4](#_Toc460416749)

[1.2.2 Bootstraping Angular 5](#_Toc460416750)

[1.2.3 Components 5](#_Toc460416751)

[1.2.4 Controllers 7](#_Toc460416752)

[1.3 Getting the data from the server ($http.get) 9](#_Toc460416753)

[1.4 Initializing data in component(Lifecycle hooks, $onInit) 10](#_Toc460416754)

[1.5 Displaying data in the template 10](#_Toc460416755)

[1.6 Displaying menu items in template(ng-repeat) 12](#_Toc460416756)

[1.7 Adding buttons for the Breakfast/Lunch/Dinner (ng-class, ng-mousedown) 14](#_Toc460416757)

[1.8 Adding and removing items from the menu (ng-click, $http.post, ng-show) 15](#_Toc460416758)

[1.9 Routing (ng-link) 20](#_Toc460416759)

[1.10 Creating order view (Using parent’s routes) 26](#_Toc460416760)

[1.11 Creating order overview component (Route parameters, Router lifecycle hooks) 28](#_Toc460416761)

[1.12 Creating add item to the order component 29](#_Toc460416762)

[1.13 Adding item to order (Raising an event on child) 30](#_Toc460416763)

[1.14 Some final changes 31](#_Toc460416764)

[1.15 Tasks 32](#_Toc460416765)

[2 Angular services 33](#_Toc460416766)

[2.1 Creating Angular service 33](#_Toc460416767)

[2.1.1 Provider function 33](#_Toc460416768)

[2.1.2 Factory function 37](#_Toc460416769)

[2.1.3 Service function 38](#_Toc460416770)

[2.1.4 Value function 41](#_Toc460416771)

[2.1.5 Constant function 41](#_Toc460416772)

[2.1.6 Dependency Annotations 43](#_Toc460416773)

[2.1.7 Summary 45](#_Toc460416774)

[2.2 Built-in Angular services 45](#_Toc460416775)

[2.2.1 Promises, $q Service and $http Service 45](#_Toc460416776)

[2.2.2 $cookies, $cookieStore and $log services 53](#_Toc460416777)

[2.2.3 $cacheFactory 56](#_Toc460416778)

[2.3 Tasks 58](#_Toc460416779)

# Angular application

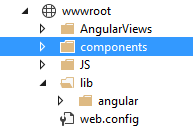
## First steps

First we need to include angular in project by adding reference to in in bower.json file.

If bower.json file doesn’t exist, we need to create it by adding new Bower Configuration File from the list of shown files.

Bower is package manager for frontend packages, all it does is install required packages with their dependencies.

After we’ve saved bower.json file, in wwwroot folder, lib subfolder there should be angular folder, as seen on image.

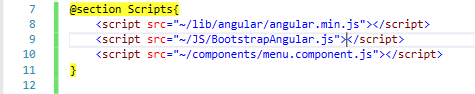


## Starting with AngularJS

On the page we plan to use angular, we need to add angular.js or angular.min.js file. It is first script on the image below.

The other two scripts represent bootstrap of Angular module and Angular component which will be shown on page.

We should include this in in Views/Home/Index.cshtml

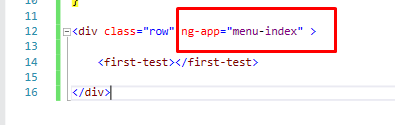


### Modules

In order to use Angular on the page, we need to tell the application on what part of the page we want to use the Angular. We do it by using ng-app attribute. Ng-app let us define the element in which we are going to use logic from Angular module. Name of the module is defined as the value of the attribute.

On the image below you can see that I’ve set the ng-app attribute on the div and inside of that div we will have “menu-index” module logic loaded.

Module is container for different parts of angular application, something like div in html.



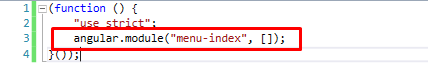
### Bootstraping Angular

We need to create new folder inside wwwroot called JS and inside of it file BootstrapAngular.js.

Bootstrap represents Angular initialization process.

We initialize Angular model by calling angular.module(*nameOfTheModule*, []);

Empty array [] in the method represents the list of modules initialized module depends on.



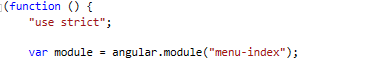
### Components

Once we have initialized Angular module, we can get reference to it.

We should create new folder inside wwwroot folder called components and inside of it create file menu.component.js.

Inside of it we create immediately-invoked function expression (type iife and press tab) and get reference to the “menu-index”.

As you can see, difference between initializing module and getting one is the second parameter, empty array, which we don’t have when getting existing module.

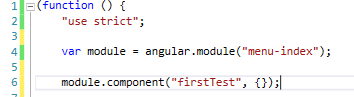


Once we get the reference of the module we can create our first component.

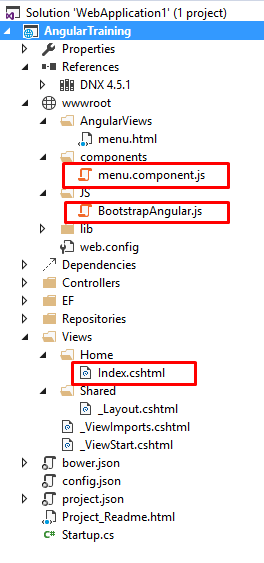
Components were first introduced in AngularJS 1.5. They are special kind of directives that uses a simpler configuration which is suitable for a component-based application structure.

Directives are markers on DOM elements which tell Angular that it should attach behavior specified inside directive to elements.

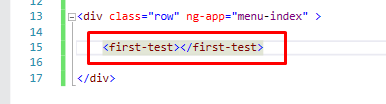
We create component by calling .component() on a module. It takes two parameters, first one is the name of the component, and the second one is object with options. As you can see, components names should be CamelCase.



So far, the project’s file structure should look like the one on the image.

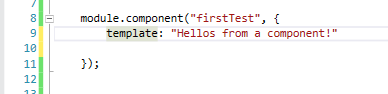


In the file Views/Home/Index.cshtml where we’ve set up our menu-index we should add our custom element with the name of our component. Notice that component name is in camelcase, and our element name words should be separated with “-“.

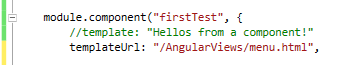


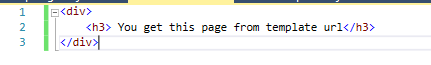
Custom elements represents part of the html that is controlled by componenet.

In our wwwroot/components/menu.component.js we can add template, it can be any valid html. If we add line from the image, save and start the project we should see “Hellos from a component!” on the page



Instead of inserting html directly in our js file, can insert whole html pages using templateUrl. We should create new file in wwwroot/AngularViews/menu.html add some html as shown on image, comment out template, and add templateUrl in our component.





After saving and starting project we should see “You get this page from template url” on the page.

### Controllers

Controller is defined by a JavaScript **constructor function** that is used to augment the [Angular Scope](https://docs.angularjs.org/guide/scope). Everything inside of a controller is available to be data-bind in a component. We add controller to a component as shown below.

Any properties we define in controller will be available for data binding from a view.





We have defined message property inside controller using this.message (we are “this” to tell angular that the property should be visible in component scope), and we can access it in the view by calling $ctrl.message inside of {{}}.

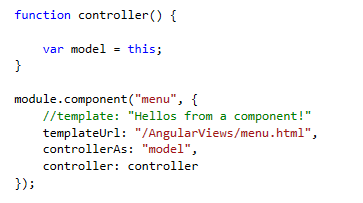
**{{}} Represent interpolation – one way data binding, it goes from component to the template because it shows data from components controller but you can’t change the data in controller with interpolation.**

Instead of using $ctrl to access controllers in our views, we can rename it to whatever we want using controllerAs in our component. As shown below we have renamed it to model and accessed the message from controller using model.message. Save the project and run it to see the changes.





Because controller is just a function, we can put it outside of our component, and just reference it inside component.



## Getting the data from the server ($http.get)

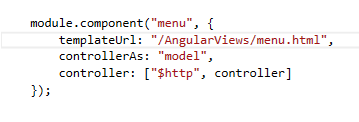
In order to show anything in our templates, first we need to get data from our web api. On the page ‘/menu/menufortoday’ we can find all meals that are on menu for today.

To get a data from server we will use $http service.

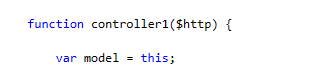
The $http service is core Angular service that facilitates communication with the remote HTTP servers via the browser’s XMLHttpRequests object or via JSONP.

In order to use $http service inside of our controller, we need to tell our component that it needs to be injected.

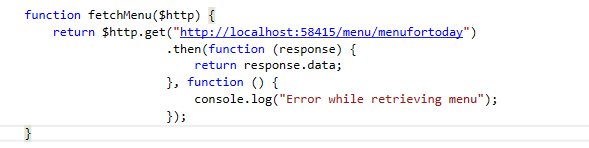
We do that by adding it in array in the component controller option



and by injecting it in controller function.



After that we create a function that will take $http service as parameter and that will fetch menu for today.



.get method represents the GET requests towards the url specified in parameter.

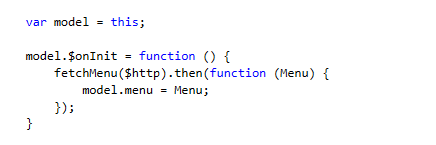
.then method returns a promise, and takes two arguments, callback functions for success and error.

As you can see on success we return response data, and on error we log it in console.

## Initializing data in component(Lifecycle hooks, $onInit)

Lifecycle hooks represent certain points in lifecycle of a component. For now we will talk about $onInit lifecycle hook. It is called on each controller after all the controllers on an element have been constructed and had their bindings initialized.

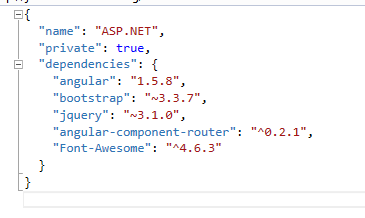
This is the place where you should initialize the starting values of parameters in controller.



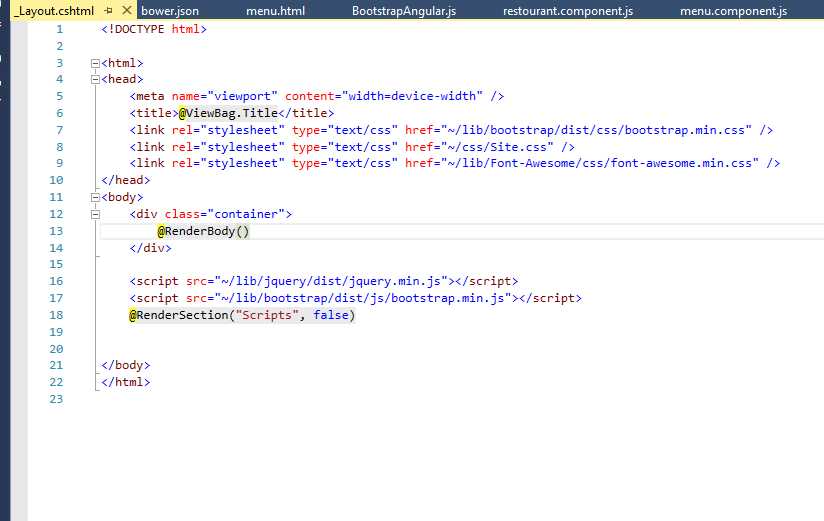
## Displaying data in the template

So far we have fetched the data from the server and set it in the controller. Now we are going to show them in our template.

But first we should add some dependencies that we are going to require at some point in this tutorial. In bower.json file we should add bootstrap, jquery, angular-component-router and Font-Awesome.



Don’t forget to include all the the new files in your project. Also wrap @RenderBody() in the div and set the class of container, we are going to need it for Bootstrap.

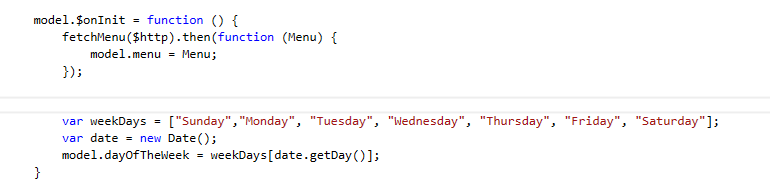


First thing I want on template is to set heading that will show for what day is the current menu.

In the template it should look like this:



As we said {{}} represents one way data binding, so that means we should set the date in our controller and this is the way we are going to do it



Because we want it initialized at the moment component initializes we are going to put the code in the $onInit lifecycle hook.

All we did here is that we created array with the names of the days, got the current date and then set the value of model.dayOfTheWeek.

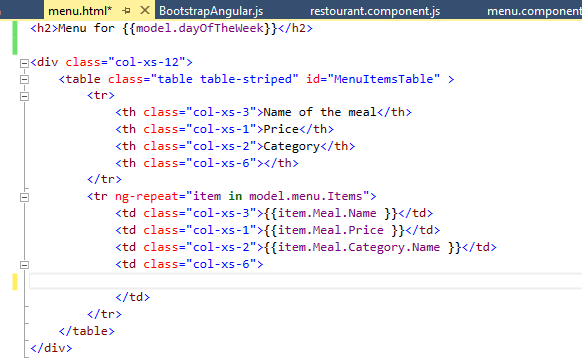
date.getDay() returns numbers 0-6 for Sunday-Saturday.

After we have saved that and got back to the template, we should see the header showing the correct day.



## Displaying menu items in template(ng-repeat)

All I want to do here is to display items in the table, here is the html markup I’ve used for that



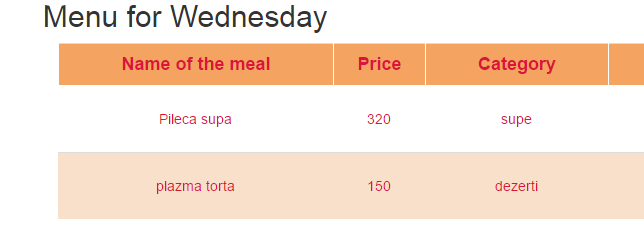
Col-xs-xx, table and table-stripped represent bootstrap classes that are going to style up our page. Col-xs-xx represent grid on the page, each row has 12 cells, those cells are represented by the number on the end of the class and it tell bootstrap how much cells the element is going to take up.

We will need 4 columns, for now we are going to use first 3 columns, where we are going to show name, price and category of the meal in the menu.

We are going to use ng-repeat built-in directive. It instantiates a template once per item from a collection. So in this case, <tr> element is going to be repeated as many times as there are elements inside of model.menu.Items array.

Item represents local variable and the current element in the loop. So all we did here is that we made a table which shows the data of the menu items array.

Styles are going to be applied from provided .css file.

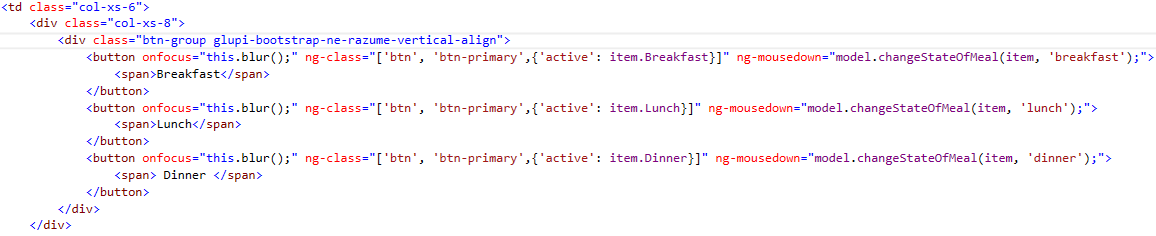


After we save the project and start it we should see something like this on the screen.

## Adding buttons for the Breakfast/Lunch/Dinner (ng-class, ng-mousedown)

In the last column I want to have 3 buttons. Each should represent one of the meals Breakfast/Lunch/Dinner and it should be active if the dish should be served for that meal, or inactive if it should not.

The markup on the page looks like this:

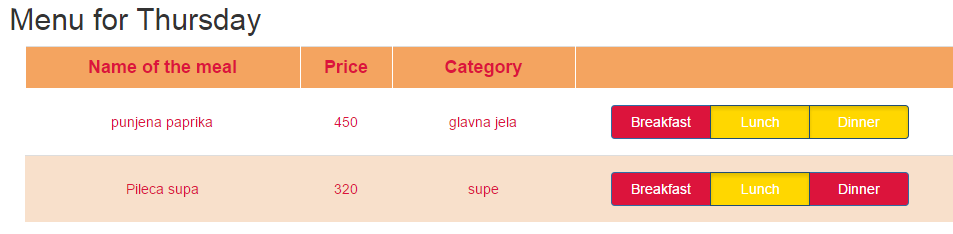


First of all, onfocus=”this.blur()” is there to remove the focus of the element, so the buttons don’t look stuck when clicked (this comes with bootstrap, and this could be done with css and classes, but this is just the easy way).

Another built-in directive is ng-class. It tells angular that it should attach given classes to the element, also you can set some condition weather the class should be applied or not.

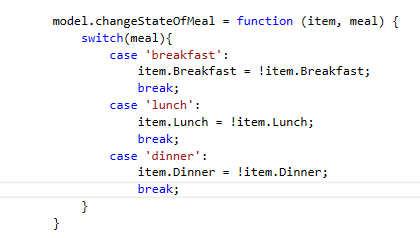
As you can see on the picture, we have given element’s classes btn and btn-primary. Those are bootstrap classes and they should style our buttons. The third element in the array is our condition which says that if the dish should be served for that meal (has for example dinner set to true) it should have class active.

When we start the application the page we get should look something like this:



But when we click on buttons nothing will happen until we implement model.changeStateOfTheMeal() function in our controller. Ng-mousedown tell Angular that when the mousedown event is raised, it should apply given behavior. In our case it will call model.changeStateOfTheMeal() function. This function should change the state of boolean values breakfast, lunch or dinner.

We should define the function in the component’s controller.

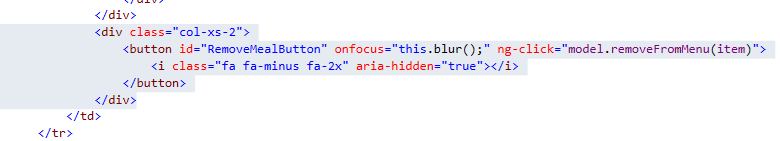


Now when we refresh the page, when we click on the button it should toggle the ‘active’ class on the element.

## Adding and removing items from the menu (ng-click, $http.post, ng-show)

I want to add button in each row which will allow me to remove the not needed items from the menu.

In the last <td> element of the row we should just add

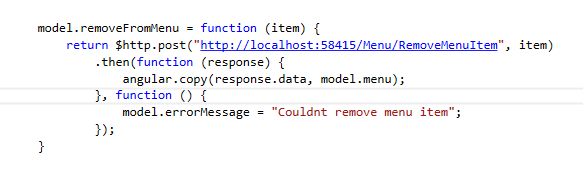


Same as the ng-mousedown directive, ng-click adds behavior for click events.

This line of code adds icons from Font-Awesome, you can find more about it on <http://fontawesome.io/icons/>

<i class=”fa fa-minus fa-2x” aria-hidden=”true”></i>

Now everything that is left is to implement the function in our controller which will remove the selected meal.



We want for changes to happen on server, not just on client. We have *‘/Menu/RemoveMenuItem’* in our web api that will remove given item from the today’s menu and return changed menu back.

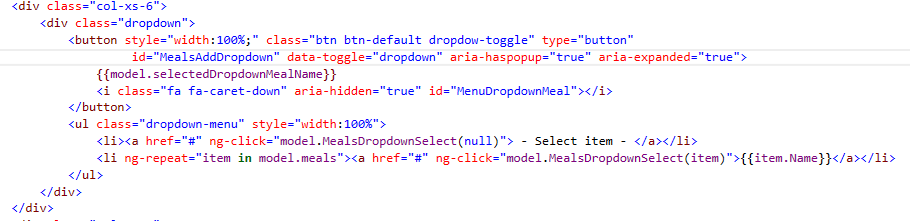
As you can see we are again using $http, but this time post method and we are setting the item (argument in the method) to the body that is the second argument in the post method. Again we define then method with two callback functions for success and error.

As we know, from api we will get changed menu back, so all we need in success function is to change menu from the controller to the menu we got back from api. We do that with angular.copy(), it takes two arguments, first is the data that is going to replace old data, and the second is the data that is going to be replace. You can see that we take data from the response we get from api, and replace model.menu with the data we got from api.

You can refresh the page and see that now we have rows deleted when we press on the – button.

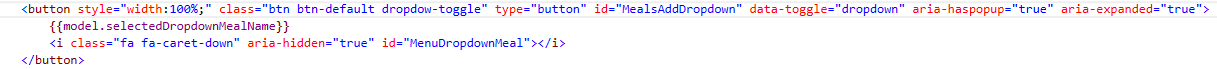
Below the table I want dropdown which will contain meals that are not in the menu and a button that will add the selected meal from the dropdown to the menu.

The markup for dropdown should look like this:



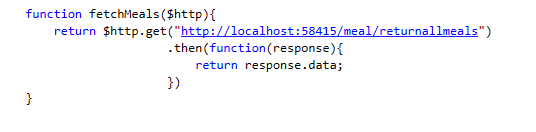
We have used bootstrap classes to make a dropdown button, as you can see we have wrapped button and unordered list into div that has class ‘dropdown’.

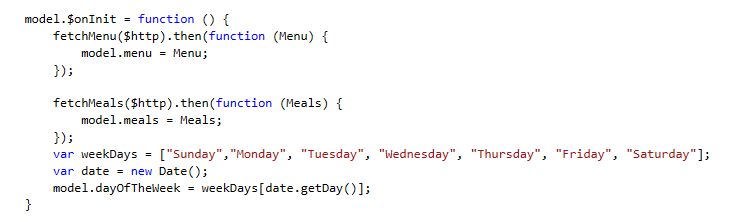
We have set the text of the button to controller variable, so whenever variable changes so will the text and added caret icon to it {{model.selectedDropdownMealName}}. We have also set data-toggle, aria-haspopup, aria-expended which are needed for dropdown to work.



As far as dropdown goes, we need to set the items in the dropdown list. For that we are using unordered list with class ‘dropdown-menu’ which is another bootstrap class. We also want dish to be removed from dropdown list when added to the menu and dish to be added to dropdown list when removed from menu.

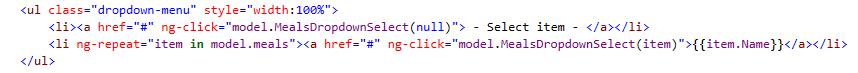
But first we need to fetch all dishes that are not already in menu with.





First list item is our default empty item when we don’t have any dish selected. After that we have ng-repeat for every meal in model.meals which contains dishes that are not included into menu.

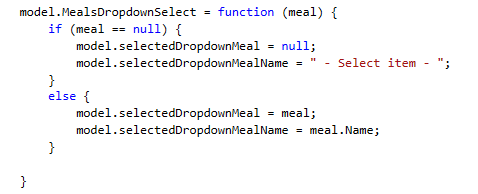
Also as you can see we need to implement model.MealsDropdownSelect() method which will set the selected item in the dropdown so we have reference to it when we want to add it to menu.



First we need to create variable model.selectedDropdownMeal and set it to null so we know we don’t have anything selected and we will need it later for some checks. Also we have model.selectedDropdownMealName which we use to show currently selected item name. The default value is “ – Select item - ”



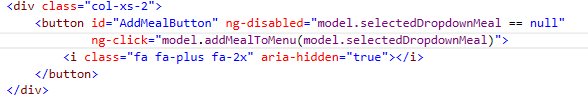
The method that should be called when we click on item from dropdown select looks like this.



It should set the selectedDropdownMeal and selectedDropdownMealName if we click any buttone that is not default and set default values if we click on our default item.

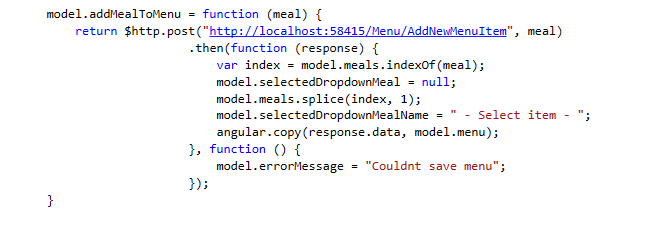
Now that we have set up the dropdown, now we need button that will add the selected item to the menu. And I want that button to be disabled if there is no selected dish in the dropdown.

This is the markup for the button:



We use ng-disabled to disable the element when the condition is met. In this case we will disable element if selectedDropdownMeal is null. Whenever the selected element is different than null it will be enabled.

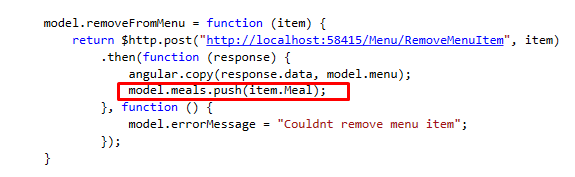
Now we need to impelemnt model.addMealToMenu method:



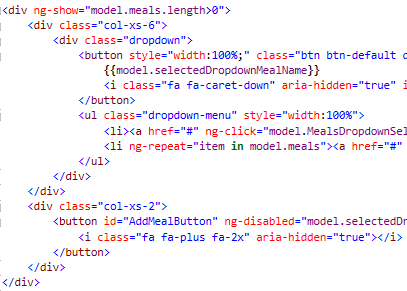
We already have ‘/Menu/AddNewMenuItem’ in our api, we just need to pass a meal and we will get back menu with added item, which put instead of current menu with angular.copy().

We also need to delete given meal from model.meals so we can’t add same meal multiple times to the menu. We do that by searching index of given meal in the array with indexOf() and removing it from array with splice() using index. We also need to set the selectedDropdownMeal to default value by setting it to null.

Now only thing left to do is to also add a meal to dropdown when we remove it from menu, for that we need to add one more line in removeFromMenu so whenever we remove a meal, it is returned to the dropdown.



As a last thing here I want to hide the dropdown if there are no dishes in it. It is pretty easy, we just wrap the dropdown in diw which will have ng-show directive (it is opposite of ng-hide), and tell it that when there are items in model.meals, we should hide it.



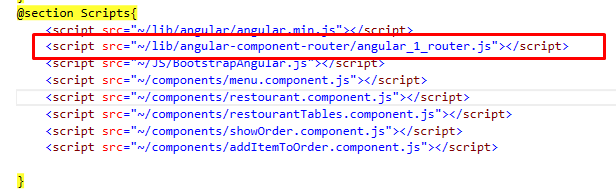
## Routing (ng-link)

First of all, we have renamed firstTest component to menu. And changed the component in the Views/Home/index.cshtml to restaurant which we will create pretty soon.

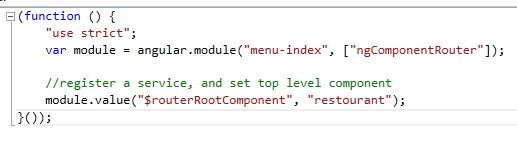


AngularJS is famous for its single page applications. That means that we can load different page on the current view without reloading page. This is where routing in Angular kicks in, it allows us to go from one component to the other.

In order to use component rooter we need to include angular-component-rooter to our application. We have already downloaded it with bower, now we just need to include it in Views/Home/index.cshtml.

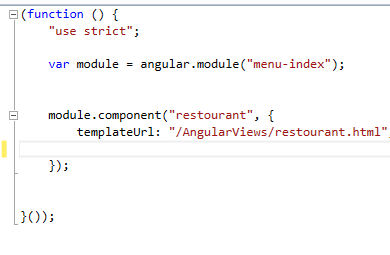


After we have done it we need to include that module in our application. In wwwroot/JS/BootstrapAngular.js we should include ngComponentRouter as dependency and after that we need to register router service and set top level component.



As you can see we have set restaurant component as our top level component.

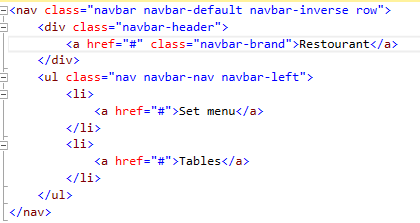
Now to create restaurant component.



Two pictures above you can see that we have already added reference to the component in Views/Home/index.cshtml.

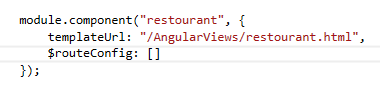
We also need to create restaurant.html in wwwroot/AngularViews.

Since this is our top level component, I want menu here which will be visible in every page of the application and which we will use for navigation.



It is simple navigation bar we created using bootstrap. There are three links which don’t lead anywhere (have href=”#”). This will be changed with Angular routes. For now I want that first and second link take us to the manu page we just created.

We need to register routes in our restaurant component. We do that by adding $routeConfig option which is array.

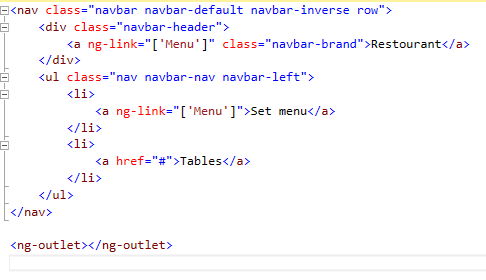


To add a route is simple, we just need to specify path, component and name of the route like this:



useAsDefault: true just tells angular that if it gets route that doesn’t match any existing route, it will reroute to that one.

Now we need to add routes in our template. We do that by using ng-link directive and specify the name of the route to which we want to go.



As you can see, we pass name of the route which we specified in $routeConfig inside of ng-link in [].

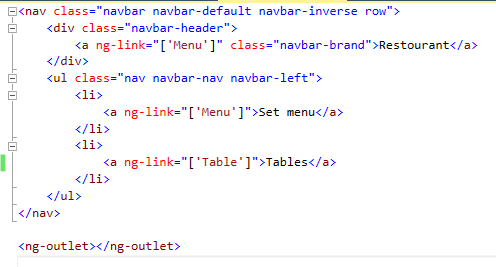
If you can see in the picture, we have added <ng-outlet> tag. This is the container in which router will display pages which we specified in routes. Since we have defined navbar outside of the outlet, it will be available in the whole application.

Now if we go to our application you will see we will be redirected to /menu, and if we try to change the route to anything else we will be redirected back to /menu.

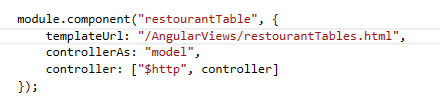
I also want to add another route which we lead to restourantTable component. We will create that component in few moments too.



And also we need to add ng-link to template.



Now to create restourantTable component inside wwwroot/components/restaurant.component.js

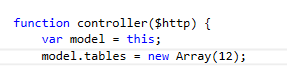


And we should also create restourantTables.html inside of wwwroot/AngularViews.

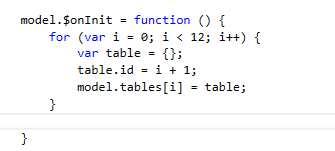
Don’t forget to add component.js to Views/Home/index.cshtml

On this page I want to show 12 tables and display if they are free or are there orders for them.

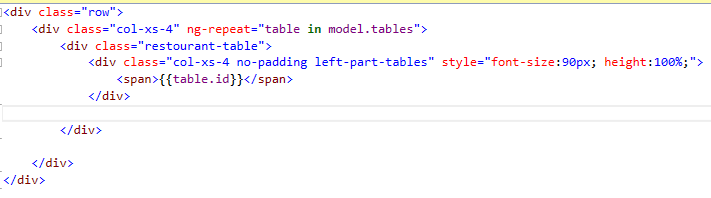
So first things first, I need to create new array of 12 elements in controller.



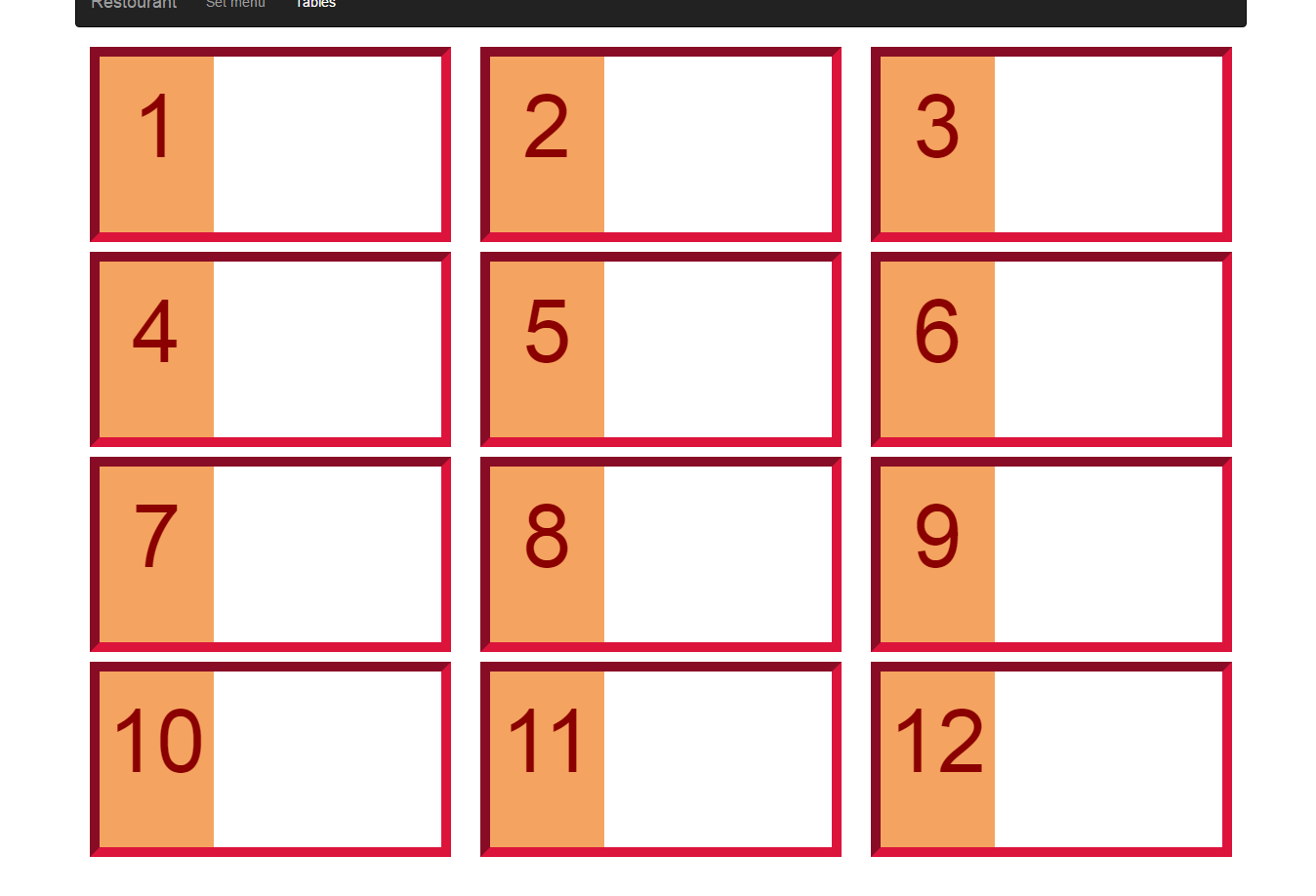
And when the component is initialized I want to create 12 table objects and set their ids.



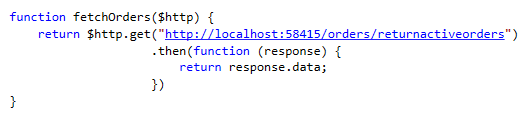
In restourantTables.html I am going to create 12 objects and display ids of the tables.



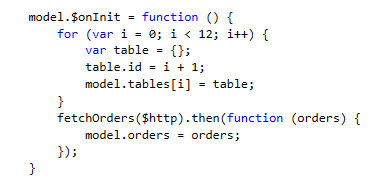
It should look like this



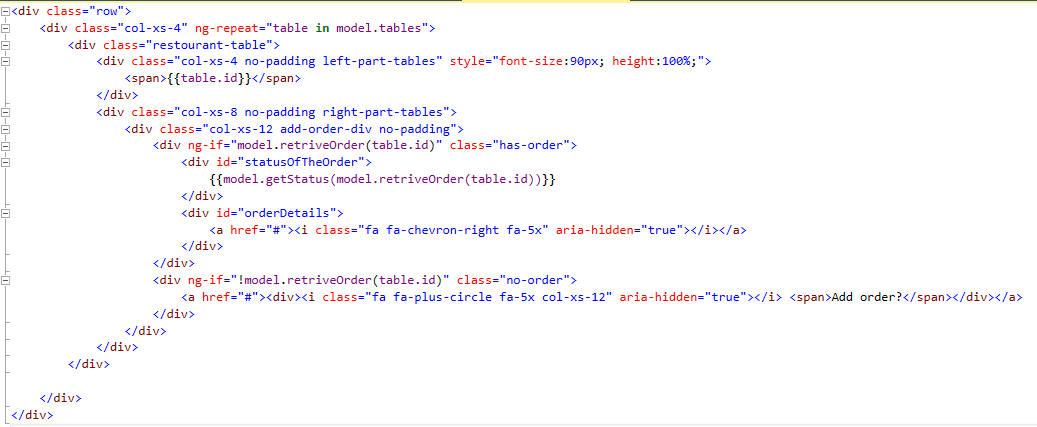
Now I need to fetch all active orders from server so I can bind them to tables, so we know which tables are taken and which ones are free.   
We can find all active orders on ‘/orders/returnactiveorders’ and we will use this method to fetch them.



In our model.$onInit method we will add call to the above method and set model.orders in controller.



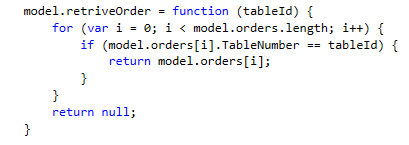
I want to show a link for adding new order if there is no order for that table and if there is an order for that table I want to show the status for that order and link for showing details of that order.

As you can see here we use ng-if directive. It displays the element if the condition is true, and hides it if it is false. It might seem to you that it is same directive as ng-hide or ng-show, but the difference between those is that ng-if removes elements from DOM when it is not displaying them and adding them back when the condition is true, while ng-hide and ng-show just set the display: none of the element.

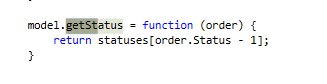
As you can see we have 2 ng-ifs here, first one is displaying div when there is an order for that table, and the second one is displaying when there is no order for the table.

To determine whether the table has order or not we are using model.retriveorder() method.

All that method does it iterates through model.orders and checks if there is any order for given table id, if there is it returns it.



We have one more method in there, model.getStatus(), we get the type of the status when using it. Depending on the statusid of the order, we return the string with status name from the array.



## Creating order view (Using parent’s routes)

The empty link on the table should lead us to the order overview page. That means we need to set up routes for it. There is just one little problem. If I would define routes for it in restourantTable component that would mean that I would need to put <ng-outlet> in restourantTables component because $routeConfig requires <ng-outlet> in the component it can’t use parent’s ng-outlet. If we would do that, then the whole content of the restourantTable.html would stay, but just additional content from the new component would be loaded in ng-outlet.

Because I don’t want that, I want that my orders overview page is loaded inside of parent’s ng-outlet. To do that, I need to give reference of parent’s component (restaurant) to the child’s component (restourantTable).

We need to pass $router from parent’s component to child’s component, we do that through binding. ‘<’ next to $router tells angular that parameter $router is input in the child component which means that parent component will pass that parameter to the child.

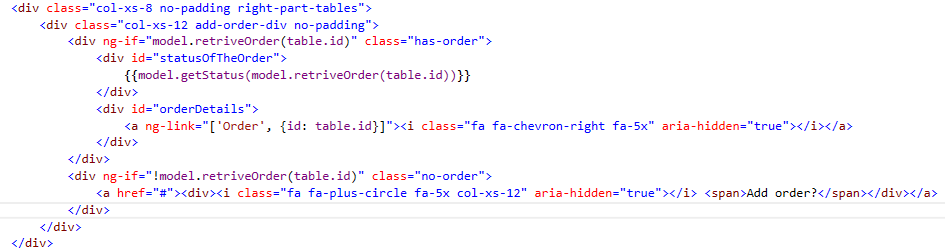
One more thing, we need to define $routeConfig for restourantTable component, because at this point Angular will throw error if there is no $routeConfig and we try to use ng-link in template. Also we need to add one fake route, because $routeConfig will not work if there are no routes defined.

Route for order overview page will not be defined in restourantTable component, but in restaurant component, because we want to use restaurant component ng-outlet.



When we’ve defined the path for order overview page we have added ‘:id’ on the end of the path. That represent route parameter and through it we will pass table id.

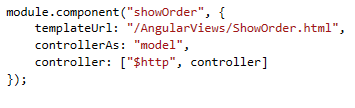
Now when we add ng-link to restourantTables.html we need to pass that parameter along with the route name. We do it in object literal like it is shown in picture.



## Creating order overview component (Route parameters, Router lifecycle hooks)

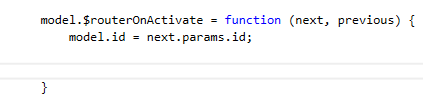
We need to create showOrder.component.js in wwwroot/components and showOrder.html in wwwroot/AngularViews. Don’t forget to add showOrder.component.js in Views/Home/index.cshtml.

First we create a component.

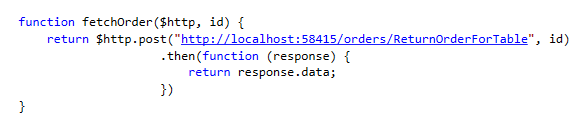


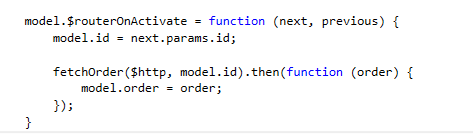
We know that we will get tableid from route parameters, but how do we access route parameters? Through $routerOnActivate – it tells angular what to do whenever router calls this component. It takes two arguments first one is the route to where angular is redirected and the second is the route from which request originated.

We get id parameter by accessing next route, its params.id and we should set it to model.id.

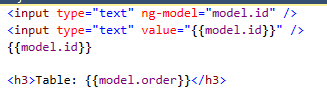


We also need to get the order for the given table from our web api and set it to model.order.



  
In showOrder.html I want to show you two-way binding using ng-model. Unlike one way binding where we get data from controller and show it in our template, with two way binding we can change data in template and send it back to controller. Those 2 inputs should clarify you the differences between one way and two way bindings. Try changing values in both inputs and see what will happen.

Also I want to display Order in json format on the page, we will need it for the next part.

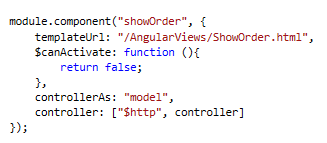


Another lifecycle hook that I want to mention is $canActivate. It allows us to define weather the component is active or not, meaning if we can access it or not.

This lifecycle hook needs to be invoked before the controller is instantiated so that means it can’t be on our controller, it needs to be defined before it. It should be a function that returns true or false.

As you can see on picture we have defined it before controller and just return false, so the component is inaccessible. If we save the component and refresh the page and try to got to show order page, you will see that we are not getting redirected.

You can use this lifecycle hook to check some condition before accessing the component.

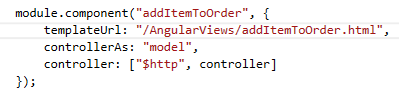


You should now remove $canActivate so that you don’t have any issues in the next part.

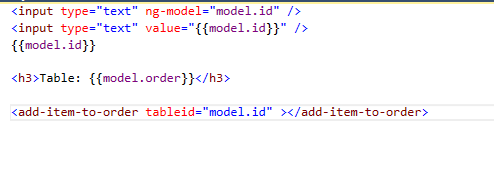
## Creating add item to the order component

Now I want to create a child component which will not be called from the router, it will be added directly on the page. For that we will create addItemToOrder.component.js and addItemToOrder.html.

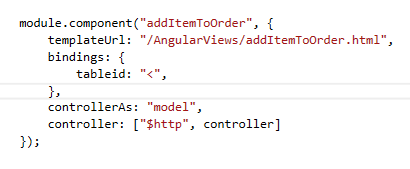
And we should create component



Just to show you how we can pass data from parent component to child I am going to pass tableId to the child component like this.



Now we need to tell child component (additemToOrder) that we have input from parent component. We do that through bindings. Again we will set it to ‘<’ because it is input into component.

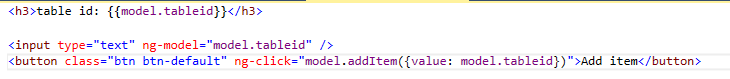


Now in our view, we can access tableid from binding through model.tableid as shown in picture.

## Adding item to order (Raising an event on child)

For now all we did is that we have sent input in child component, but is it possible to send data from child component to parent component? Of course it is possible. We can raise events in child component which will trigger events in parent component.

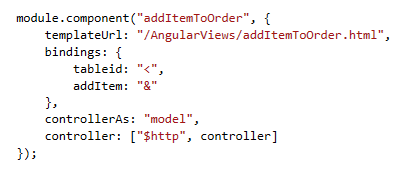
In addItemToOrder.html we should add a button with ng-click, it should call a method model.addItem and we are going to pass a parameter value through object literal.



We will not define method addItem() in addItemToOrder component will just set the binding of it to ‘&’.

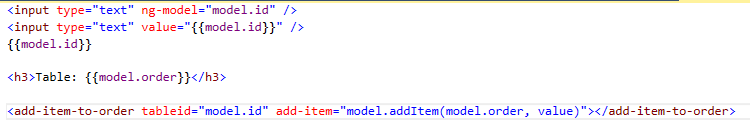
The ‘&’ operator allows you to invoke or evaluate an expression on the parent scope.

You can think of bindings as public api for component.



Using ‘&’ operator will be enough for child to raise an event on parent, but now we need to tell parent that there is going to be event raised on child.

We do that by setting the attribute on our child component inside parent component(showOrder) html. Once again we lowercase everything and put – before the capital letters.

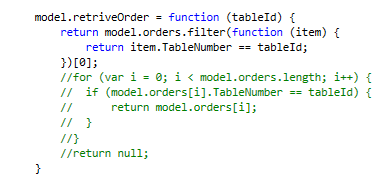
- add-item represents the binding

- model.addItem represents the method in showOrder component controller

- model.order is the order from showOrder component controller

- value is from object literal inside of child component.

## Some final changes



I have changed the model.retriveOrder method in restourantTable component. Instead of using for loop, I am using filter method. For me it is a little easier to use it in this particular instance.

The filter() method creates new array with all elements that pass the test implemented by provided function.

We will always want to return first element of the array, since we know there will always be 0 or 1 elements in the arrays, since there can’t be more than 1 active order on any table.

## Tasks

1. You need to implement functionality on menu component that every time you click on a button Breakfast/Lunch/Dinner everything is saved on server.
2. On order overview page you need to remove json and inputs and you need to show Order with items:
   1. Create a form in child component which will add new item to order
   2. Form should pass meal and quantity to parent component (order item)
   3. Only meals that are not in the order can be added to order
   4. If there are no meals to add to order, child component should not be shown
   5. Total price of order should be automatically updated whenever new item is added
   6. Breakfast is from 8AM – 11AM, Lunch is from 11AM – 5PM, Dinner is from 5PM – 10PM, make sure that only meals that are served for lunch on that day can be put in order depending on the time of the day
   7. Add a button to save order (changes should be saved on server when this button is pressed)
   8. Add a button to return back to the tables overview
3. Create new order page
   1. When adding orders only meals that are not in the new order can be added to it
   2. Breakfast is from 8AM – 11AM, Lunch is from 11AM – 5PM, Dinner is from 5PM – 10PM, make sure that only meals that are served for lunch on that day can be put in order depending on the time of the day
   3. Adding an option can be done in any way you want
   4. Price should be automatically calculated
   5. Add a button to save order
   6. Add a button to return back to tables overview page

# Angular services

Services are designed to be injected into the other components in you application therefore, their creation is very much a part of Angular’s dependency injection system. You can use services to organize and share code across application.

Services are:

* Lazily instantiated – service is instantiated only when an application component needs it
* Singletons – each application component gets reference to single instance generated by the service factory.

Built-in angular services always start with $ ($http, $q, $log…).

## Creating Angular service

There are five functions you can use to create Angular services. Those five functions are:

* **provider()** – Creating services with provider() function allows you to explicitly create a configurable provider object. The provider knows how to create the resulting service. All functions except **constant()** internally call the provider function
* **factory()** – wrapper around provider
* **service()** – wrapper around factory function. It will internally call factory function which will then call provider function.
* **value() –** thin wrapper around the factory function
* **constant()**

In order for a service to be created, there must be a provider which knows how to create that service.

**Objects that know how to create injectable services are providers.**

We use $provide service to call provide function. It has several methods that register components with the Angular injector.

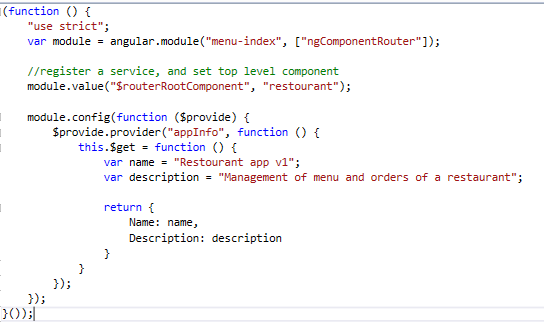
Basic process is that provide service ($provide) creates a provider which contains a function that is used to create a service.

### Provider function

Usage: $provide.provider()

Provider must define property $get, this is the function that will be called by angular to create a service.

Service is configurable via underlying provider.



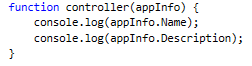
Angular injector will automatically inject $provide build in service. As you can see we call provider() method on $provide service and pass name of the service (appInfo) as first argument and function which will create service as a second argument. As we said, provider() method must contain $get property which will actually create a service.

We have just set two variables name and description and returned object literal which represents the service.

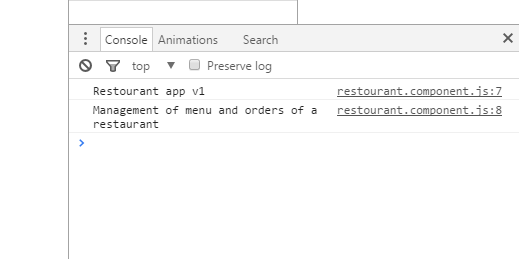
In restaurant’s component controller we have included appInfo service as dependency.



Passed it to controller and logged the name and description to the console.

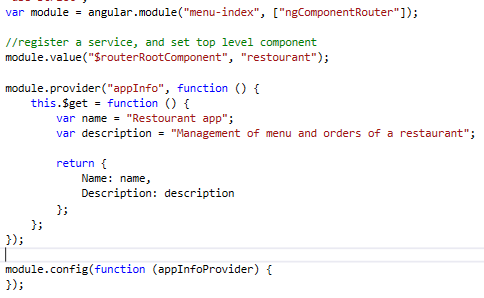


If we start the application now, when it loads we should see the following output in console.



We are using provider() function when we need some of its properties configured during module configuration phase.

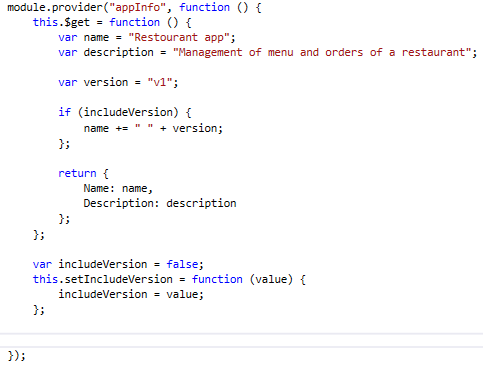
Provider function is also exposed to module object, so you don’t need to inject the $provide service.



We are now going to add version info variable inside of $get function and if bool includeVersion is true we should append version to the name.

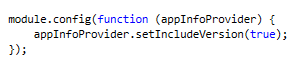
We should also make a setter function on this object for includeVersion property.

As you can see like $get, setIncludeVersion is also defined on this object. But unlike $get function which will create service, we use setIncludeVersion to configure service and what is returned by $get function.



In module.config instead of injecting $provide service we are providing appInfoProvider (underlying provider) which angular automatically creates when we call provide() function, it will always be the name of the service (appInfo) following with the word Provider.

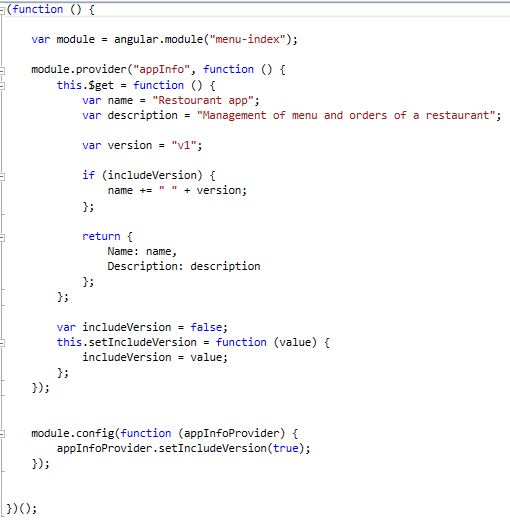
We can now set includeVersion field in module configuration phase by just calling setIncludeVersion on appInfoProvider.



If you refresh the page and check the console you will see version appended to the name of the name. Same if you change setIncludeVersion to false, version won’t be included.

Basically you use provider() function when you need to configure service in module configuration function, if you don’t need to do that you will probably use some of the other functions we already mentioned.

For clearer project structure, I have moved provider service to new file wwwroot/services/common.service.js. Don’t forget to include the new file in Index.cshtml.



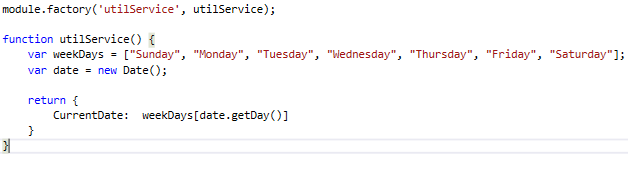
### Factory function

It is simpler version of provider function when additional configuration is not necessary



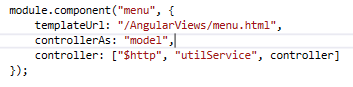
We create it by calling factory method on our module inside of wwwroot/services/common.service.js and passing the name of the service (utilService) as first parameter and function (utilService) that will be called to create and return our service as second parameter. That function will be assigned to get property when the provider function will be called behind the scene (because factory is just wrapper around the provider function).

Now we need to create utilService function that will create our service and return current day of the week. We do it like this:



We just copied first two lines from menu.component.js and created object literal which represents our service. Inside of it we have set the value of CurrentDate to weekDays[date.getDay()] same way we did in menu.component.js.

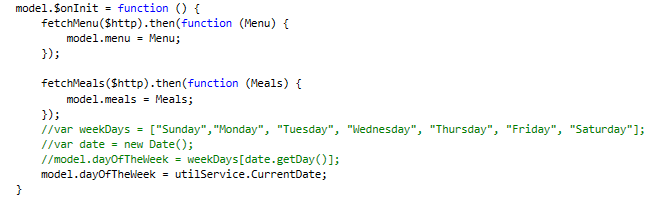
Now that our service is ready we need to include it in menu.component.js.



And in controller.



After that we just need to replace the code.



And we have included the service in our controller.

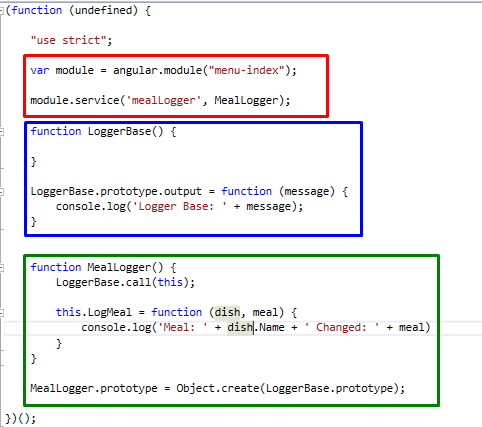
### Service function

Service function calls factory function which calls provider function.

Function you pass to the service method will be treated as a constructor and called with Javascript “new” operator.

So you should use service method instead of factory method when you need your function to be treated as constructor and called with “new” operator. One of the reasons you would want to use service method is when you have inheritance.

Here is the way we are going to implement service using service method.



Here we can see how inheritance work, we have base class LoggerBase (blue rectangle) and set prototype property of output to simple function that logs in console.

Few things you should know about prototypes:

* Every Javascript object has a prototype. The prototype is also an object.
* All Javascript objects inherit their properties and methods from their prototype.

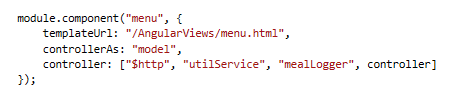
After that we have made MealLogger (green rectangle) which calls base constructor function, LoggerBase.call(this) and adds another method that logs in console dish name and changed meal. On the end we have set the MealLogger prototype to the prototype of LoggerBase.

After we have set up inheritance we should register our service using service method(red rectangle).

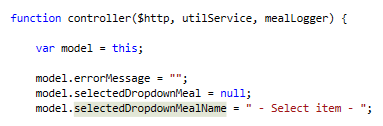
Using service method guarantees us that resulting service object has all the methods from base class as well as the child class.

Include new service file in index.cshtml.

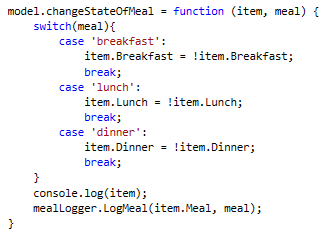
Now we need to include our service in menu component.



And inject it into controller.



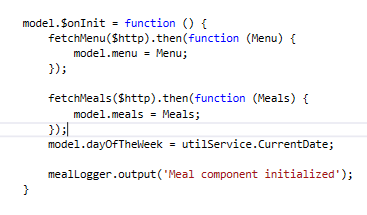
We are going to log every time someone clicks on meal button on menu page, we just need to make a call for a method inside of mealLogger service.



And now we can check the console to see that every time a button is pressed there will be new log in console.

Also if you remember we have defined output method on our base class. Just to show you that we can use that function, we are going to call it inside model.$onInit property.

We are going to log simple message “Meal component initialized” every time a component is initialized. If you save file and navigate to menu page and open up console you should see the message inside of it.



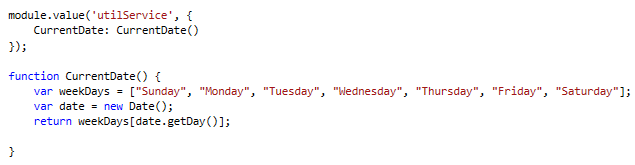
### Value function

Value function is shorthand for calling factory function without parameters. So when you don’t need to inject anything in factory function you can use value function instead.

It can’t be injected into a module configuration function and it can be overridden by and decorator.

Value function is wrapper around factory function and you can use it instead of factory when you don’t need to inject anything into service.

Since we are not injecting anything into utilService, we are going to change it from factory to value.



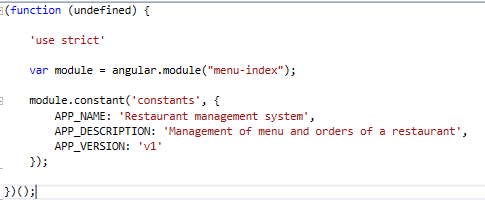
### Constant function

Constant function allows you to register an object literal function or some other constant value with the injector, but it doesn’t call any of the other service creation functions.

Can be injected into a module configuration function and it can’t be overridden by decorators.

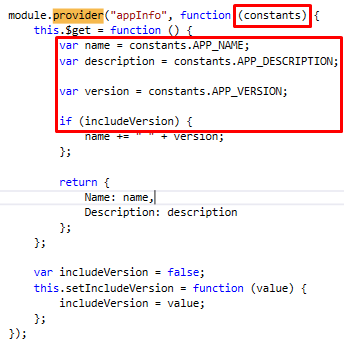
We need to create new file in wwwroot/Services names constant.service.js. Don’t forget to add the file in Index.cshtml.

We are going to call constant method on our module, name the service ‘constants’ and return object literal with constants as seen below.



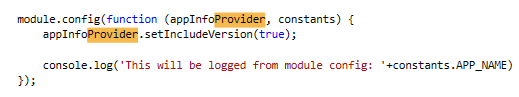
In this way if use APP\_NAME in more than one place we don’t need to change it on every page, we just need to change the value of constant.

In common.service.js we are going to chage appInfo service.



We have injected constants service in anonymous function and we are now getting name, description and version from constants function. If you save the files and refresh the page you should now see values from constants service in console.

Just to show that constant function is injectable in module configuration function we are going to add it there and log message for it.



If you refresh the page you should see new message in console.

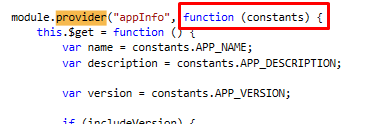
### Dependency Annotations

We use dependency annotations for couple of reasons:

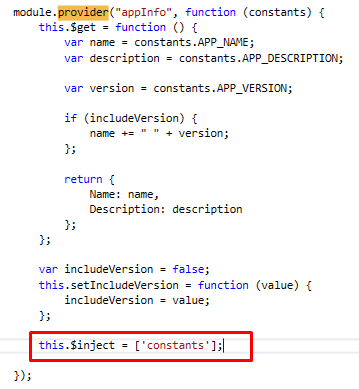
* To inform injector what services should be injected into component
* To support minimization of javascript code. If you try to inject services into a component without using proper annotation technique javascript will result in mangled parameter names that cause code to break.

There are three ways to annotate dependencies:

1. First is to specify the names of the dependencies using the parameters you passed to the component. This code is not safe to minify. We have used this technique when we have passed our constants service into appInfo service.



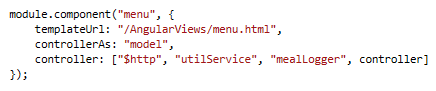
1. You can use $inject property annotation – assign inject property of a service an array of the service names that should be injected. With this technique it is save to minify the code.

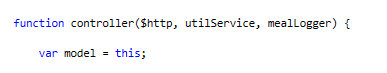


Since we are using anonymous function we are accessing $inject property by this.$inject, in case of the named function we would use nameOfTheFunction.$inject.

1. Use inline array annotations – you pass an array of strings, which are the dependencies, as the second parameter to your service creation function. The last parameter of the array is the function that will be called to create a service

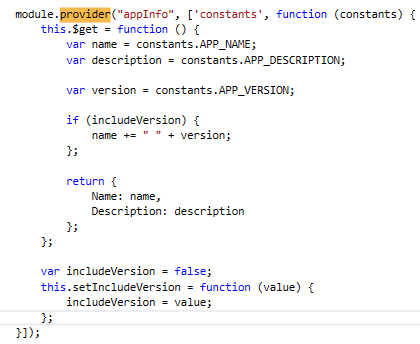
We have already used this technique when we were adding $http and other services function to our controllers.



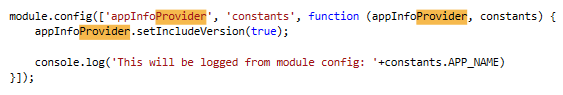


Also order of the dependencies matter, if you try to change the order of the dependencies in controller and you don’t change it in the array in component definition you will get an error.

We are going to change the appInfo service and annotate the dependencies.



And also module configuration function.



### Summary

You should use provider function when you need to configure service inside of module configuration function. If you don’t need to configure service inside of module configuration function you should use factory function (wrapper around provider function). If you need your function to be treated as constructor and called with new operator you should use service function (wrapper around factory function). If you don’t inject anything into factory function you can use value function (wrapper around factory function). Constant function doesn’t call any of the previous functions and it allows you to register object literal function in the injector.

## Built-in Angular services

### Promises, $q Service and $http Service

In Javascript, promises are object which represent the pending result of an asynchronous operation (Definition by Martin Fowler).

$q service provides an API for working with promises, but it also provides an API for the deferred objects that return promises to the calling code and signal them with results when the asynchronous operation is complete.

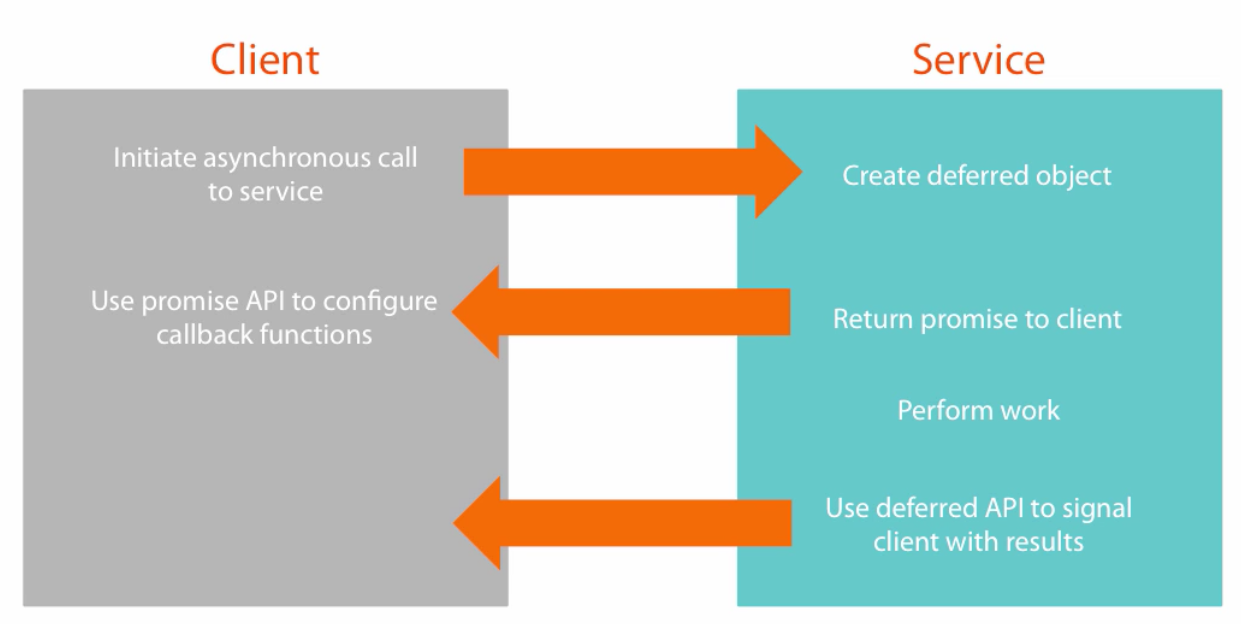
The next picture will demonstrate how the $q service works. You can think of client as a component or a function that wants to perform some work asynchronously, and service is the component that will perform the work and notify the client when the work is complete.

Client first initiates asynchronous call to the service, after the service receives the call it will use $q service to create a new deferred object. The deferred object will serve as the conduit through which the service can communicate the status of the asynchronous work back to the client.

The deferred object will immediately return a promise back to the client. This will prevent the call to the service from blocking execution while the work is performed.

Once the promise is received by the client, the client can use the promise API to configure callback functions to execute when the work and the service is complete.

Meanwhile, the service is performing the work the client requested. When the work is complete, either because it finished successfully or an error was encounters, the service uses the deferred API to signal the status of the work to the client, at that point the client can execute the appropriate callback function depending was the work successful or not.



$http service is the primary service available in Angular for sending and receiving data to and from a server. It is a function to which you pass configuration object that specifies the details about the type of request and how it should be performed. The function returns a promise.

$http configuration object contains all of the details about how the request should be executed. It is object literal you pass to the $http service. Common properties of configuration object are:

* method – HTTP verb that should be used (GET, POST…)
* url – where the request should be sent
* params – key value pairs that will be appended to the url
* data – send data to the server (body of $http request)
* headers – let you specify additional http headers
* cache – boolean value that determines whether the request should be cached or not

Response object of $http requests:

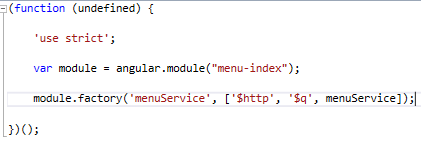
* it is a promise and the response object will be passed as a parameter to the callback functions that process the promise.
* Properties :
  + Data – body of response (for example JSON data)
  + Status – http status code
  + statusText – corresponding status message
  + headers – $http response headers
  + config – contains configuration object that was passed to the $http service when the request was made

#### Get data from server

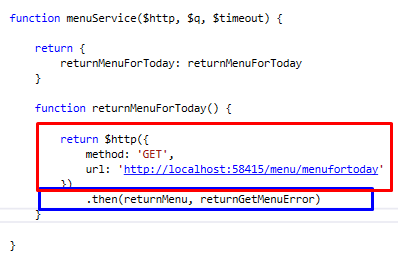
Now we are going to remove fetch Menu function from menu.component.js and we are going to rewrite it into service.

Under wwwroot/Services we need to create new Javascript file called data.service.js.

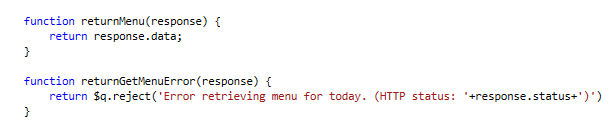
Inside of it in immediately invoked function expression we are going to make menuService service and inject into service creation function $http and $q services.



After that we are going to create our service creation function. It should look like below.

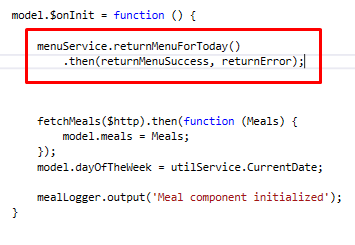


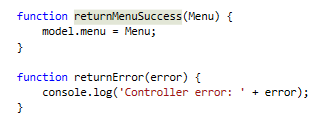
Red rectangle represents $http configuration object. As you can see we have set the method to ‘GET’ since we want to pull the data from the server and we have set the url to which the requests should be sent. The $http service returns back a promise. To handle promises we use .then method. It takes two arguments, function for success and function for error. You could use inline functions instead of named functions also.



If successful we are just returning data of the response, else we are using $q.reject() to handle an error. $q.reject() is just a shortcut to create a deferred object and reject it immediately.

We are handling the promise directly in the service, since we don’t want our controllers to handle $http response object, they just need to get the data or the reason why the data wasn’t available.

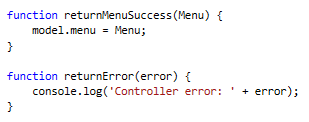




In our menu.component.js controller we have commented out the old way of retrieving the menu and injected menuService (don’t forget to add it as dependency annotation in array and as parameter of controller function). Since services also return a promise, we need to handle it with .then function. We know we will either get menu if successful or error message if error happened. Don’t forget to include new js file in Index.cshtml.

If you refresh the page you will see nothing changed, but we are now using service for retrieving the menu, and we can reuse that service any number of times in different controllers.

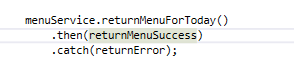
You can further chain .catch method after .then method. For example if error was thrown inside of success function nothing will handle it.



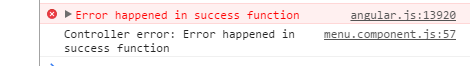
If we throw an error in returnMenuSuccess function, returnError won’t handle it since it handles only errors from promise. If you check the console of the browser you will see thrown error.



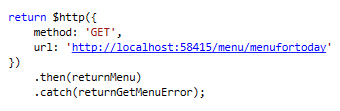
If we remove the error function from .then method and chain the .catch method, we will handle all errors that happen.



And the console will log message.



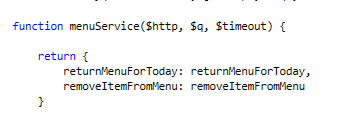
We have also changed the handling of promise in our service.



#### Remove data from server

We have changed our web api, we have changed the method from ‘POST’ to ‘DELETE’ and you can find new action on this url ‘http://localhost:58415/Menu/Items/:id’ where :id represents the id of menu item that is going to be deleted. You need to uncomment it in MenuController and comment the old actions if it hasn’t already be done.

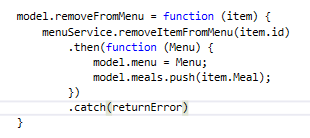
First we are going to add new function in our menuService service in data.service.js.





It is pretty similar to our ‘GET’ method, as you can see here we have set the method to ‘DELETE’ in our $http configuration object. You need to pass the id of the menu item to this function, after which it is appended to the url. As already said $http configuration objects returns promise, if successful we are calling the removeItemFomMenuSuccess, which returns menu from response body, or calls removeItemFromMenuError which calls $q.reject().

In menu.component.js we are going to change model.removeFromMenu function.

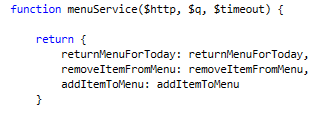


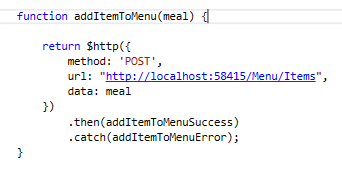
As you can see on picture, we are using inline function when promise returns success result and already defined returnError function in our catch method.

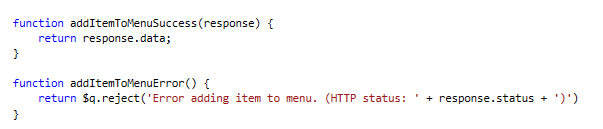
#### Posting data to server

We have also changed our api, so now to add item to menu we need to post data to ‘http://localhost:58415/Menu/Items’. Uncomment the new code, and comment the old one in MenuController.

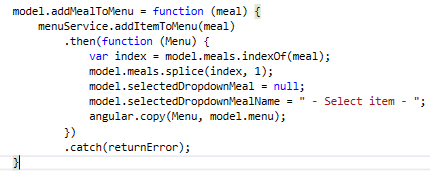
Again first we need to add some code in our data.service.js.







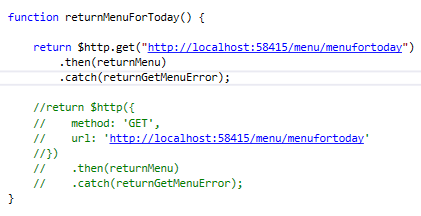
Same like previous two methods we are creating $http configuration object. Just this time we are setting the data property of configuration object which represents the data we are posting to server. In this case it is the meal we want to add to menu.

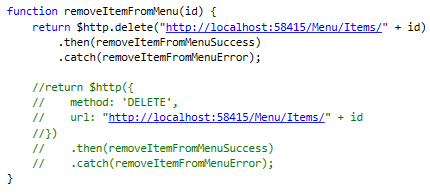


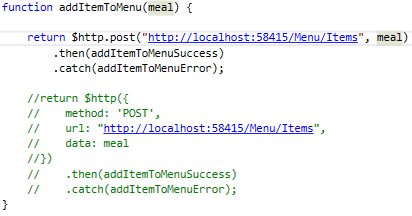
In menu.component.js we have changed the function addMealToMenu so it now uses service to add new menu item.

#### Shortcut functions

We can also use shortcut functions for communicating with the server, here are the examples.







### $cookies, $cookieStore and $log services

To read/write simple string values in cookies, you can use $cookies service.

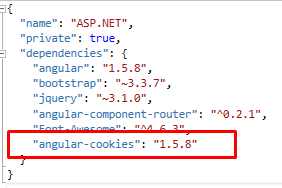
If you need to store an object, you can use $cookieStore service. It serializes and deserializes the object to and from JSON.

$log service includes five functions for showing messages in console:

* log()
* info()
* warn()
* error()
* debug()

You can configure whether or not calls to debug will be shown in console using the logProvider.

In order to be able to use $cookies and $cookieStore we need to get angular-cookies package and include it in our module.

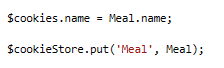




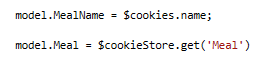
After that we need to inject service into controller and we are ready to use cookies.

To store values in cookies as simple strings we just use $cookies service as shown on picture.

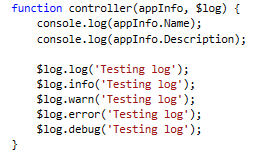
If we need to store objects we use $cookieStore service. Put method of $cookieStore takes two arguments. First one is the name of the key by which the object will be identified and second is the value that will be stored with that key.



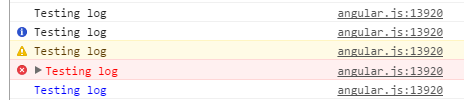
Retrieving data from cookies is also easy, as shown on picture below all you need to do is to call $cookies.keyName or $cookieStore.get(‘keyName).



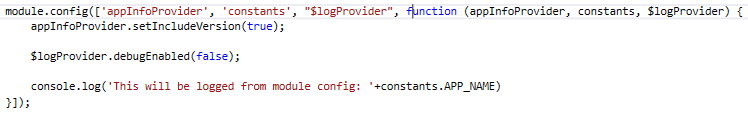
I have injected $log function in restaurant component and wrote the following code.



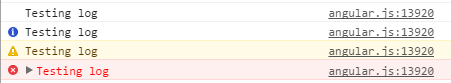
Now when we refresh the page we should see messages in console.



We can enable or disable debug messages in module configuration function.



We need to inject $logProvider into configuration function and set debugEnabled to false. If we now reload the page we won’t see debug output.



### $cacheFactory

$cacheFactory creates cache objects from which you can add and remove items. Once a cache object has been created by this service it can be accessed in other part of application using the service.

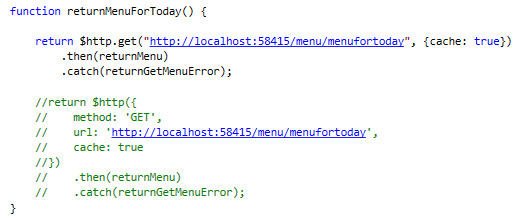
In order to enable caching of $http requests we need to set cache property of $http configuration object to true. Angular will then cache the $http response object in the default $http cache. If you want to cache the response in your own cache object you could specify that object as the value of the cache property instead of specifying true.

In cache objects items are stores as key value pairs. When you cache an $http request, the URL specified on the $http configuration object is used as the k ey for the cache value. The actual value that is cached is the $http response object.

Before we enable caching go on menu overview page then change to tables page then go back to menu page, you will notice on network tab of your browser that every time you go back to the menu overview page menufortoday call is made.



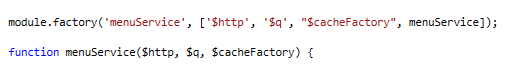
We are going to enable caching for returnMenuForToday method in data.service.js.



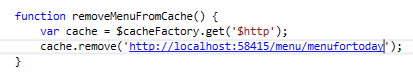
If you refresh the page and check again your browser console you will see that returnMenuForToday is called only once when the page is loaded, if you change pages and go back to menu overview page you will see that no call was made.

We know need to remove menu from cache when new item is added to the menu since then the cached object and object we would get from the method would not be the same.

First we need to inject $cacheFactory service into our controller.



After that we need to create a function that will remove menu from cache.



$cacheFactory.get(‘$http’) will get cache object with the name ‘$http’ which represents default cache object for $http requests. If you want this object to be named differently you need to specify the name in the cache parameter of the $http configuration object.

Rmove method removes the item from the cache object with the provided key, if you remember for the $http requests key is the URL.

Now we need to call this function every time we add or remove item from menu.



If you now add or remove item from the page, change pages and come back to the page you will see that returnMenuForToday is called once to fill up the cache.

## Tasks

1. You need to refactor the code so all the communication to the server goes through services
2. Cache all GET HTTP requests
3. Implement PUT requests for editing the objects inside of application (you will need to add appropriate methods inside of repositories)