# COPMP 2772 and COMP 8772 Practical 3 – Part A

This practical is divided into 5 checkpoints. *Each checkpoint is worth 1% of your final grade. Ergo the entire lab is worth 5% of your total mark for the course.*

You are provided 3 weeks to complete the work in these practical's. **This practical is due before the end of your practical session in Week 11.**

## Lab purpose

The purpose of these checkpoints is to monitor progress and provide the necessary guidance to allow you to complete the practical portion of the assignment, which constitutes a majority of your marks for this course. As such it is strongly encouraged that you attempt and comprehend all checkpoints in all labs. Seeking a 50% pass rate for checkpoints in these labs will be setting yourself up for failure in the practical assignment. You have been warned!

While peer collaboration is a valuable and necessary part of the learning process there is a distinct difference between seeking assistance from fellow students and copying their work. Plagiarism and dishonest conduct have and will result in severe penalties (refer to Academic Dishonesty on FLO). You will be expected to explain and justify your checkpoint work to demonstrate it as your own, simply providing visual evidence is not sufficient to get checkpoints awarded.

## Environment

As stated in lectures while there are many options for both code creation and viewing HTML documents in the interests of consistency across students and to ensure a minimal amount of compatibility issues students are advised to write code using the VS code environment and all work will be presented in the Chrome browser.

- VSCode
- Chrome

These are also the environments that will be used for demonstration in workshops and for assessment of your assignment work.

**As a final note be aware that demonstrators will expect code to be formatted and readable with defined white space and indentation. This will be *fundamental* in later labs so get used to it now.**

## Task 1

Task one will involve adding in views and using ng-include to build a webpage out of templates. You will need to consider the different templates needed and how best to section your code so that the page displays as intended.

**Provided Solution (provided at the end of week 8)**

For this checkpoint you are encouraged, if you have it, to use your own solutions from Lab 02. If you are either dissatisfied with your own solution, or did not achieve the final checkpoints, an index.html, style.css and app.js files have been made available. This solution is on FLO in the "lab03_website.zip"

file. **You should copy the relevant files to your web_dev folder and ensure it all works before modifying anything!**

Should you use the provided solution it is up to you to ensure they are the correct files, in the correct locations, set up in a manner to work in the node.js environment. This will mean copying them to your working dev environment and replacing the existing barebones versions. For the bulk of this checkpoint you will be separating the current solution into one that is compiled out of templates. The final solution should have no visual dissimilarity from the starting version.

**Checkpoint Body**

The necessary templates for this checkpoint should include:

- title.html
- navbar.html
- news.html
- create.html
- about.html
- footer.html

There should be no need, for this checkpoint, to heavily modify either your style sheet, nor your working app.js (or equivalent) file. The created templates should be placed in a "views" folder and incorporated into a skeletal index.html file using the ng-include directive. Note that in order to display relative locations the file name must be inside single quotations:

ng-include="**'view.html'**"

Again, note the additional single quotes around view.html. **The index.html skeleton should include no displayable content.**

Careful consideration should be given to which div elements are used as ng-include containers, and how the use of div spacing may impact the eventual layout of your page. Ensure that your final solution mimics your own or the provided solutions original appearance.

***Checkpoint 11 – When you have completed the installation and are displaying the page at localhost contact a demonstrator to mark off the checkpoint.***

## Task 2

Task two and three will introduce the ngRoute service to allow for deep linking in your page. Before beginning this checkpoint ensure that you have a script reference to the angular ngRoute framework:

http://ajax.googleapis.com/ajax/libs/angularjs/1.6.8/angular-route.js

The above library was used to create the Lab templates and will work with this portion of the lab, and should be included as a <script src=… element in your main index along with your angularJS reference.

Completing this checkpoint will involve a number of steps, initially involving:

- Including the ngRoute library
- Ensuring your templates are sectioned off cleanly, with all the content for the news, content and about pages in separate html templates

- Specify your route pathways in your app.js (or whatever you called your script file)

As with previous checkpoints there should be no need to modify style / css code, nor the need to significantly modify the HTML document (assuming appropriate thought was given to templating).

**It is strongly recommended that you read the entire checkpoint through first, and take time to consider your implementation. This checkpoint WILL involve considerable changes to your current script file and it is suggested that you create a new copy to work on rather than directly editing your current script. Additionally, there is a significant likelihood you will delete or remove portions of the current implementation. Simply sitting down and starting to code is a recipe for disaster!**

Remember that in order to display ngRoute capable content, you must designate an element with the <ng-view></ng-view> tags, and that no content will appear within that tag (as it will be populated based on routing logic). Thus it is essential that each of your view templates is self-contained.

Once you are satisfied with your template structure you will need to modify your app.js file to make use of routing logic.

Ensure that you are making use of the ngRoute library within your app by injecting it into your angular module declaration:

```
app = angular.module("app", ["ngRoute"]);
```

Once this has been included you will need to establish the routes for your ng-view templates. In order to do this, you must make use of the `config` function. This is invoked the same way you make use of the `controller` function, except instead of injecting the $scope service, you inject the $routeProvider service:

```
app.config(function ($routeProvider){
```

Within this method you will use the $routeProvider service to set up your template routes. The $routeProvider has two primary functions, when and otherwise. For this you will need to use .when to declare the paths for each your view templates, and an otherwise statement to provide a brief 404 message to users that they have arrived at an unknown page.

The .when argument requires, at the minimum, a specification for which route to monitor, and which template to load when that route is accessed. It may also be useful now (and will be necessary later) to establish controllers for each of your content based routes. While we have so far relied on 1 controller to manage our page this is both poor modularisation implementation, as well as resulting in non functional operation when you need to pass variables around to different views (e.g. passing created news content to the news view).

Your routes should follow the format shown below (shown with a view declared controller):

```
$routeProvider
 .when("/", { templateUrl : "<viewdir>/news.html", controller : "newsCtrl" })
```

Each route will require an additional .when statement to specify the location (the shown example is when the root URL is targeted). In addition controllers declared in the routing script do NOT need to be declared in the view template itself.

Once all .when routes are provided, you should include an otherwise function to print a basic HTML message:

```
.otherwise({template : "<h1>404 I got no clue Fam</h1>"});
```

Once done this should provide routing logic for your page, and you should be able to change the view by going to the correct URL (localhost:8000/about). However, you will notice that there are still 2 significant issues with the SPA:

- Button clicking will not navigate. It is currently set to change page based on visibility, not based on route.
- We cannot show any created news (and will most likely get an error trying) because the single controller we were using cannot see the different view content in the news and create templates

To address the button issue you will need to change what the button does when it is clicked. Instead of showing or hiding a template, it will instead instruct the page to navigate to a URL.

This will mean making use of the $location service. As with other services to use this we must inject it into our controller. Note that it does NOT need to be added to the module like ngRoute:

```
app.controller("head", function($scope, $location){
```

It can simply be added to your controller along with $scope to be made available to that controller. Once the $location service has been added, you can use it to instruct the page to a URL using the .path function:

```
$scope.setCreate = () => {  $location.path('/create');…
```

This code would replace the ng-show / ng-hide directives, which become obsolete when using our routing to handle visibility.

The more complex issue to solve is allowing your page to share data across views. In order to achieve this, you must rethink how your page is communicating. As each controller cannot see other views, including the “primary” controller that you have probably used so far, this solution is no longer effective.

Instead you will need to define individual controllers for each of the routed views (and may find it beneficial to declare one for nav bar operation). Again when using route best practice is to declare controllers for each view within the route declaration.

This will also mean creating new controller function for each new controller, and copying the relevant code in to each one; the create controller needs to know how to add messages to an array, while the news posts controller needs to know what is in the array to us in the ng-repeat. This does not however solve the problem of accessing shared data. While we have previously discussed $rootScope as a way to universally share data it is not the correct solution; you want to avoid using $rootScope doing significant communication tasks.

The best option in AngularJS is to create your own Factory service. Factory services are utility objects that retain their state within an application, regardless of where they are called. In this instance we will create a Factory service and provide access to both the Create and News controllers. By doing this

the Create controller will update the Factory object, while the News controller will read from the service.

To save you time the code for the Factory service has been provided:

```
app.factory('newsData', function(){
    var newscontent = {};
    newscontent.arr = [];
    newscontent.add = (tit, cont, auth, tag) => newscontent.arr.push({
            "title" : tit, "content" : cont, "author" : auth, "tags" : tag});
    return newscontent;
});
```

This code creates a Factory service called 'newsData'. Within that service is a callback function where you can define behaviour. In this case the factory creates an object called newscontent, adds an array object to newscontent, and has an inner function called "add" that takes 4 arguments, and uses those arguments to assign values to the array.

Lastly the factory returns the newsontent object. This ensures that when called by controllers, the service returns the contents of the newscontent object. This is important to ensure the factory assigns the correct objects to our News controller that reads the Factory.

The last step is to provide your Factory service to your News and Create controllers so they can share data. This is done in the same way as any other service, injecting it into the controller callback function:

```
app.controller("newsCtrl", ['$scope', 'newsData', function($scope, newsData){…
```

**Note:** *When injecting Factory services into controllers the service names as well as the call back function are encapsulated in an array. This prevents some read access issues can occur within the AngularJS library. The format is shown below.*

```
app.controller("ctrlname", ['service', 'service2', 'service3', function(service,
service2, service3){

//inner data that can access service, service2 and service 3

}]);
```

By providing this factory service to both of the view controllers for news and create, the data contained within will be shared between them.

**For Create**

The create controller will use the newsData service to call the add function. The arguments passed to the add function are the $scope variables for your input filed ng-modal values.

**For News**

The news controller needs to get the array from the factory (newData.arr, because the Factory is already returning the newscontent object) and map it to a local $scope object that can then be used in the News view's ng-repeat attribute. After this, after ensuring that your objects are mapped correctly, the SPA should again transmit data between the separate views.

For the sake of correct Angular implementation, the final three steps are to

- Include an ng-controller attribute in the Nav bar content, and move the button code to that controller. Again this should be added as an attribute in the HTML template (as it is not routed content)
- Create a controller for the about page that maps the aboutContent and aboutHeader calues to that controller.
- Add the necessary code to implement the controllers to your script

To summarise:

- Include the ngRoute library and inject it into your module
- Ensure your templates are sufficiently encapsulated
- Create an ng-view element in your HTML where route content can be exposed
- Use the config function with the $routeProvider service to set up the necessary routes for news, create and about (you can ignore the Login page).
- Add controllers to each route for future handling.
- Inject the controller that manages your navbar with the $location service, and use its .path function to redirect the view to the correct template
- Add the Factory service code to your project.
- Modify your Create and News controller to gain access to the Factory service
- Move the remaining code from your obsolete body controller to controllers for your navbar and about templates.

Note that the final product should look no different than it did when this process started.

***Checkpoint 12 - 13 – When you have completed the routing and controller management contact a demonstrator to mark off the checkpoint.***