COPMP 2772 and COMP 8772 Practical 3 – Part B

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.

Task 4 / 5

This task is a single checkpoint for both tasks 4/5 and deals with making AJAX calls to a server. Before this task you will need to download the provided files:

- server.js
- default_news.json

It would also be a good idea to create a copy of default_news.json. This way if you make mistakes with the file you can reset it without worrying about formatting. **Both of these files should be placed in the root folder of your project.**

server.json

This contains a very basic Express.JS structure that will provide the contents of default_news.json to your Angular app, handle some simple get and post calls to facilitate updates / responses and server the core pages of your Angular app.

default_news.json

This is a simple JSON file with 2 default news posts. These will be called when your server starts to provide basic content. When posts are made, this file is updated with the new content so that restarting the server provided consistent data.

Express and FS

For part A of the lab the default Node.JS framework handled communication, serving our basic page on localhost:8000. However, in this lab Express will be handling the server aspects of your App. In order to do this any modules needed must be installed. While Node.JS has a number of modules by default it does not have the ones we need.

You will need to install the modules:

- Express
- FS

Express

Express is a framework that allows for greater and simpler interaction with the Node.JS environment. It is similar to how AngularJS provided extended functionality for HTML and JavaScript.

FS

FileSystem is a module that allows for file management and interaction. In this lab it will be used to read and write to the news json file. A note; FS is being used to manage a file read / write system. In a proper implementation this is a poor substitute for a proper database structure. It is being used here simply as a substitute.

Install and start the Express server

These modules are installed via the NPM in a similar way to how the you ran the server in Part A. As usual, it may be beneficial to copy your work from Part A into a new folder, and use this new folder for Part B.

Navigate to the root directory of your web folder. Once there you will execute two commands:

```
npm install Express –save

npm install FS --save
```

After each of these NPM will execute an install. Once both have been executed you can go to the package.json file and open it in VSCode. You should see the express and FS listed as dependencies.

```
  "dependencies": {
    "express": "^4.16.3",
    "fs": "0.0.1-security"
  }
}
```

At this stage you may wish to open server.js and familiarize yourself with the contents. It is **strongly** suggested that you do not modify this file for the lab work. The server file will handle GET and POST routes from your Angular app, taking an argument of the POSTed content in a JSON string, and GETting the contents of the json file when requests.

A key difference in the Express environment is that Node.JS must be instructed to execute a specific server file, so “npm start” will no longer be sufficient. To run the server instruct node to execute the server.js file:

```
node server.js
```

You should receive a message in the terminal:

```
Listening on port 3000
```

You should now be able to view your index.html page by going to the root localhost on port 3000

```
localhost:3000
```

Task Outline

This lab has 2 “simple” tasks, both making use of the $http service to GET and POST data to the server.

The key considerations in these task are:

- Calling the correct server route, with the correct AJAX request
- Processing the received data to the same format as your app uses
- Posting the data to the server in the correct format.

The exact implementations will depend largely on the methods you used to complete the previous lab tasks. The server will respond to both routes with consistent data and your app should make use of these responses. *You should not modify server.js to change the data types or responses.*

As a last note… This structure, for the sake of simplicity, only needs to call the GET route once, on page start-up. Outside of that the local structures you have already developed will handle updating data. Similarly, when POSTing data only that specific post should be updated, not the entire data array. While this may not be an optimal solution it has been chosen to have minimal impact on your existing implementations.

AJAX GET

A GET request to the http://localhost:3000/getnews will return response data in the pipelined .then function that follows the use of the $http function:

```
$http({
        //AJAX request
 }).then( function(response){
        //manage response
 });
```

$http is a service and as such should be invoked through a controller (where it should be declared in the same way $scope is)

The data in the response value is contained within the “data” variable. Note that the server stores the values in the same structure as it was suggested during lectures, within an object identifier. In this instance the object identifier is ‘arr’. So direct data entries can be extracted from:

```
response.data.arr
```

Before this data can be used it may be beneficial to ensure it is encoded correctly to an object following Angular’s preference. Angular provides two helper methods for working with JSON objects

- angular.toJson()
- angular.fromJson()

In the case of a GET request we are getting JSON data and converting it to a JS object:

```
var raw_data = angular.fromJson(response.data.arr)
```

Once you have access to the response.data.arr content you will need to update your news array with the retrieved content. How you do this will depend on your Angular implementation of the news data structure; you may need to add to the array, overwrite the contents, or some other system.

It is essential you consider where in your Angular app this execution will occur. It may also be beneficial to consider the $window.onload function. This will execute attached code once when the page is loaded…

```
$window.onload = () => {
```

AJAX POST

The POST request should work in a very similar manner to your GET request. It should be executed when the button to add a news post is pressed. Unlike the GET request, the server expects POST data on http://localhost:3000/writenews

While making a POST request requires a url and method, the same as GET, it also requires data and header information. This defines the data to be sent, and the format it should be sent as. The Express server expects data as a JSON and as such this should be the content type:

```
            data : //my data,
            headers: {'Content-Type': 'application/json'}
```

The server expects a JSON string that is a single entry. You may find it easier (conceptually) to create a single object first:

```
 = {"title" : $scope.in.title, "content" : $scope.in.conten, "author" :
$scope.in.author, "tags" : $scope.in.tags}
```

And then make use of angular.toJson to ensure it is correctly converted to a proper JSON string. That object can then be POST’ed as your data to the server. Again the exact implementation will strongly depend on your Angular app structure.

Checkpoint 14 / 15 – When you have completed the routing and controller management, and your server can be shut down and restarted while retaining data, contact a demonstrator to mark your work.