Lesson Notes/Plan

First lets create an Angular project this will require installing Node, Git and then Angular with NPM install angular cli.

Node is a package manager, and a package is essentially a collection of pre written code that is intended for a particular purpose in this case Angular is what is known as a Javascript framework that makes developing easier but providing a variety of tools out the box, and a good architectural framework to follow in regards to how to write our code.

Git is a tool that allows us to pull and push (upload/download) our code changes into a repository(Online storage), there is more to git and git flow but there is no need to cover that right now.

Once the Angular node package has been installed then we can create a new Angular project with the Angular CLI (Command Line Interface), the CLI runs on top of the windows built in command prompt except it will know recognize any Angular specific commands that are typed in.

So to create a new Angular project we type ng new “project name”, this will prompt us with a couple of options such as do we want to set up Angular Routing and what form of CSS styling the project should be set up with, I cant think of a use case where you wouldn’t want Angular Routing setup out of the box as it’s what allows you to navigate through your app unless you wanted to build your own version in house, the CSS styling aspect is in reference to whether you want to use CSS, SCSS, or SASS to style you’re your HTML elements it’s literally just a different syntax on how your write your CSS, while SASS, and SCSS provide some extra benefits but when you build/run your project everything is converted in CSS at the end of the day.

Once this project has been created it’s important to run NPM install as this will install any dependencies the project requires, explain what dependencies are where the NPM install command looks to find out which ones to install, then we can navigate into it, and run the command ng serve, which does at the name implies it serves our application up on a local port which is 4200, so go to localhost:4200 on your browser to see the Angular application with it’s template code.

An Angular application is powerful due to its architectural setup of modules, and components, and module is a collection of a primary piece of functionality, so for example you’d have a module for all administration tasks on your websites, and then a module for all mapping functionality, then within these modules there would components which are just more granular/specific pieces of functionality i.e within the administration module, you’d have a component that handles all user account page, and then a component that handles the user login page, within each of these components there will be a HTML, TS, and SCSS files and the idea is to encapsulate(contain) everything relating to that functionality within that component, it makes the code considerably easier to manage.

The primary starting point for an Angular application, i.e where to start implementing code is the app.component.html as this is your primary view you could alter this but I see no reason to, this is where all the global components (components such as the header and footer that will always be within view) reside and then the router-outlet.

The router-outlet is linked up to the app.module.routing.ts which I will delve into later, but for now all you need to know that any components that you navigate to within the website will be displayed within this router-outlet html tag.

Angular has this strange bug that even though we specified to the CSS style to be SCSS in the ng new command that it still defaults to SASS, navigate to the angular.json to change this then explain how the angular.json is responsible for the NG G C and so forth.

Excellent next we can look to start coding, first lets remove margin and padding from every element on the site within the styles.scss with the \*{} syntax, as each browser has margin and padding by default.

Next lets create a globalComponents folder, then run the ng g c header command to create a header component, once that’s been created it’s important to note that Angular will automatically import this newly created component into the app.module.ts, the app.module.ts is essentially the manifest for our Angular application so when we make reference to header.component within the HTML it will know where to find all the relevant files via the import statement at top of the app.module.ts, this is also true for services, and any other modules and so forth.

Lets navigate to the header.component.html where we can start to type up the HTML

* Check prewritten HTML for reference for this.

Consider what we would need in our header?

* Navigation items, both general and then admin based ones, a title, a logo, and then a mobile menu icon when in mobile view.

Before we start writing up any CSS lets get the data structure in place for the \*ngFor directive, a directive is an Angular term but it’s essentially an attribute that you can attach to a HTML element that will alter the way it behaves in one way or another, much like the way you can style a HTML element by typing style=”color: red” that will set all the text within that element red, a directive is simply that but often providing more functionality that a style attribute would.

In the case of ngFor this is a directive that will loop/iterate/go through every element/item within the linked array and for each element/item it will create an exact copy of the HTML element it’s attached to, in this case it’s an li>a tag, the \* signifies that this is a structurally altering directive, that simply means that this directive will alter the HTML structure as it’s creating new HTML elements, we also have another common structural directive known as \*ngIf=”condition” and this means that if the condition within the parenthesis is true or false it will hide or show the HTML element it’s attached to.

Check header.component.ts for reference to the arrays/object that will we loop over, objects are what make up everything within JS but more on that later, for now all we need to know is that an object is a collection of “key value pairs”, so a key is the reference you make to gain access to a value.

For example

* Var testObject = { nameKey: “sam” }

So the object is called testObject to gain access to the value of “sam” I need a “key” to unlock that value/”door”, at least that’s a way I like to think about key value pairs. This can be done through what is known as dot notation which looks like this testObject.nameKey.

Or to actually see that you’ve unlocked the value with the key console.log(testObject.nameKey)

In our example code there are two arrays within the object and those are the arrays that we will loop over with \*ngFor, probably best to do a more visual explanation of this works in regards to the HTML, it’s actually exactly the same as how you access the values in the above example with the keys.

The [] brackets on the <a> tags are used when we need to pass a dynamic variable into that HTML attribute, so for instance in this case for each iteration of \*ngFor we are passing in the routerLink key value from our object, whereas if we were hardcoding that value it would simply be routerLink=”x”, however this would mean that every <a> tag will have the same routerLink and therefore all navigate to the same component/route within our application.

String interpolation {{}} exactly the same concept as the [] brackets notation except this time for passing in variables into the HTML elements text content.

The CSS now 😉

Refer to the CSS within the component and talk through that.

Also don’t forget to mention the importing of the ‘Roboto’ font, and how you have to link the style sheet, and then set the font family of the global CSS selector.

Make the mobile menu next, normally I’d design mobile first but hey I forgot and as the mobile component and header component are both going to use the same data object for the \*ngFor elements, we can create a service.

A service is somewhat as the name suggests, it’s a typescript file that provides a service often they provide data to components across the application or business logic, the reason they are particular useful is in this case where we have two components that require the same data, so rather than having to rewrite the data out twice we can use this service to provide the data, and then link/wire up our components to the service.

This is done via a process known as dependency injection which I wont cover in detail right now, but it’s basically importing the service into our component, in somewhat of a similar fashion that we import components into the app.module.ts to allow the application to have access to this components by knowing where to reference them in relation to the file structure.

We create a service in a similar fashion to creating a component with ng g s serviceName.

Our service will need to return the headerOpts object, so we require a method/function to do just that, and name is appropriately and clearly, in this case getHeaderOpts seems like an appropriate name.

And this function has the return keyword in it, and as the name implies it will return whatever comes after it.

So in order to wire/link this up to our components we have to use the constructors in our components, a constructor is just another method/function that is called before anything else on the component is when it’s loaded on our application and it will construct any dependencies the component needs, it’s some what similar to how NPM install will grab any relevant dependencies required for our app this will grab any services that are required by our components.

It’s somewhat similar to assigning a variable to a value, except in the constructor in this instance we use the syntax private/public readonly variableNameService : DataService which means that we have created a variable that is of type DataService, readonly also means that we can only use the methods on the DataService and not change them it’s just an extra level of control to prevent you accidently altering a service some way through a component, relatively unlikely that you’d do it by accident anyway.

Once that has been wired up, we need to call the getHeaderOpts method and then assign that response which will be the object within our service into a local variable on the component, in order to do that Angular has what is known as lifecycle hooks, and these are methods/functions that are called during specific times during the components construction.

What this means is that when we navigate to component for example before it gets rendered into our browser view, and then after it’s rendered there are several processes that take place from starting that process to the end of that process that we can hook into to carry out functions.

The most common lifecycle hook is known as ngOnInit which as the name implies is a function that is triggered as the component is initialised(loaded into the browser/view) and it’s here that we can call our getHeaderOpts method and assign the data to headerOpts, if we didn’t do this then when we try to run \*ngFor on the headerOpts variable in our component it won’t do anything as that variable wont have anything assigned to it.

There are a couple of ways we can invoke the getHeaderOpts with our newly created service variable, one of them is to call the service variable getHeaderOpts directly within the ngOnInit method like so

ngOnInit() {

this.headerOpts = this.headerOptionsService.getHeaderOpts();

}

Which isn’t inherently wrong but It’s better to wrap this line of code in it’s own function

this.headerOpts = this.headerOptionsService.getHeaderOpts();

And then call that function within ngOnInit like so

ngOnInit() {

this.getHeaderOpts();

}

getHeaderOpts(){

this.headerOpts = this.headerOptionsService.getHeaderOpts();

}

The reasoning behind this is this is easier to test which I’ll talk about later (Testing is boring and lame no one wants to do it 😊 ) but we know have a function that clearly explains what it’s doing just based on it’s name, so again legibility/transparency is an important factor at play here.

Once we have wired up both the mobile menu and header component to this new service we can create the mobile menu component html, and in exactly the same way we created the ul>li tags for the header component.html with \*ngFor we will do the same thing in the mobile menu component.

Check prewritten CSS for reference to styling.

Next is to setup a @Input() Boolean to trigger the visibility of the mobileMenu an @Input is a piece of functionality that allows the parent component to pass along a variable into the child component, via an input tag on the childs unique HTML tag.

The parent component in this case is the header.component.html as it contains the app-mobile-menu html tag within it, making the app-mobile-menu the child element to pass a variable into with the use of [] brackets.

So on our mobile-menu.ts we have an @Input property like so.

@Input() showMenu = false;

Which we default to false as we want to hide the mobile menu by default, then within the header.component.html & header.component.ts

showMenuVariable = false;

<app-mobile-menu [showMenu]="showMenuVariable"></app-mobile-menu>

The text within the [ ] must match the name of the @Input field on mobile-menu.component.ts, and then within the parenthesis we pass in the variable from the header.component.ts which is called showMenuVariable.

Once we have that variable being passed along we can write a method to toggle the value of this between true and false like so.

toggleMobileMenu(){

this.showMenuVariable = !this.showMenuVariable;

}

The syntax of = ! basically means that we are going to set the this.showMenuVariable to the exact opposite of it’s current state, so false will go to true and vise versa.

Then to bind this to the html we can use a (click)=”functionName()” attribute like so.

<i class="fas fa-bars header\_logo\_menu" (click)="toggleMobileMenu()"></i>

Next lets look at animating the mobile menu, with a closed class and transform: translate-y, there are a couple of CSS aspects to remember here and that’s to make both the header, and mobile menu fixed and then set the z-index of each respectively so the mobilemenu sits below the header – Check css for example.

To add openlayers into our application we first need to install the openlayers node package, “npm i ol –save” the –save will save this package reference in our package.json so when we install on a new machine it will install this package.

* Note – I’m going to skip the openlayers stuff for now at least in terms of explaining it, makes sense to cover the other aspects of JS and HTML first, also I need do more research into it to be able to explain proficiently.

The next part of the application is setting up the dive site information panel on the right hand side, there are a couple of aspects to consider here, first is how are we going to send data through to the component based on what point was clicked on the map?

Secondly how will this panel behave on a mobile device?

Then there are the aspects such as getting the weather, and surface temperature of the water from an external service that we can plug into our application.

I’ve started to create modules know for the remaining functionality as it’s a good way to keep things further organised, but it also has some performance benefits such as lazy loading which I’ll demonstrate later, but lazy loading means that unless that module is navigated to via a routerLink it wont be loaded in the initial application when we first visit our site, which means it will load quicker, this is really significant if you have an absolute massive website the last thing you want is to make the user wait longer than they need to, to be greeted with your site.

In order to use the components within our module throughout the site, we will need to export those components – Reference the module export and explain how that works in relation to app.module.ts

Answering the above questions, there are two ways that you could send the data the first is to use an @Input property which would work and is perfectly acceptable, however for this use case I’m going to use a service and the reasoning for this is to help keep the map-view.component.ts lean and clean of all logic that doesn’t relate directly to the view of that component, so we can abstract out the large data structure containing all the relevant information about the dive sites into a service, and it also means that any other components throughout the site that might need access to that data can easily grab it without the need to have it piped through an @Input tag, which would mean there unique component html tag would also need to be nested within the map-view.component.html.

The general rule of thumb Is that a component.ts file should only contain logic that controls the presentation of data, html, anything else should be abstracted out into services/classes accordingly.

In regards to the behaviour on mobile device, my proposed solution would be to have it behave in a very similar fashion to the mobile menu and have the pane slide out from the right once a dive site has been selected, using a Boolean to control that behaviour.

I refactored the code at this point so talk through what I’ve done in the headerOptionsService and how I’ve wired up the diveSites data object with the map-view.component.ts

Next we wire up the dive-site-info-panel.component.ts with the diveSiteInfoService, which introduces the use of subscriptions and observables.

An observable is a stream of data, so lets say we have a backend server somewhere that sends us through random numbers every 10s like so

5 > 8 > 10 > 7 ect this is a continuous stream of data if that makes sense?

There is quite a lot to observables they themselves could be an entire topic, but for our use case here, we simply need to know that they return a stream of data or more precisely in our case a piece of information containing our current dive site information.

In order to grab this data you need to subscribe to the observable and in much the same way you subscribe to a newsletter you will get a new one each time it’s published so we get new data each time it’s published through the Observable data stream.

So looking at the service code –

diveSiteInfo = new BehaviorSubject<any>(null)

get getDiveSiteInfo(): BehaviorSubject<any>{

return this.diveSiteInfo;

}

The BehaviorSubject is our Observable that we can subscribe to grab the latest piece of data sent to it, so the get getDiveSiteInfo function returns that observable, the “get” keyword is a special keyword here which I actually can’t explain fully at the moment so I’ll come back to that once I’ve researched an adequate explanation of it.

Notice how the getDiveSiteInfo() is followed by : BehaviorSubject<any> this piece of syntax following the : states what this function will return, as it’s returning our BehaviorSubject that we declared above, if it returns nothing we would type : void, void meaning it’s void of any value.

Next lets look at how this is subscribed to in our component.

Notice the difference between the first and second code block in regards to the (response) part, one has a function keyword prior and the other => after, they both behave in a similar way but it’s more common to see the first example as it’s a shorter syntax, and known as an arrow function, allowing us to emit the need for the function keyword.

getCurrentDiveInfo(){

this.diveSiteInfoService.getDiveSiteInfo.subscribe((response)=>{

if(response !== null){

this.diveSiteInformation = response;

}

})

}

getCurrrentDiveInfoFullSyntax(){

this.diveSiteInfoService.getDiveSiteInfo.subscribe(function (response){

if(response !== null){

this.diveSiteInformation = response;

}

})

}

Looking at the current code relating to the Observable though so that function within the service is returning the observable which we can then subscribe to with the .subscribe syntax, which means that there is a connection between this component and behaviour subject as long as it’s being subscribed to, and every time a new response comes through we will assign that to the diveSiteInformation variable.

Remember that we need to call this function in ngOnInit!

Should note that we could call this response anything it’s simply a function argument/parameter, but it’s good to name your function parameters appropriately again for legibility reasons.

Delve into some simply function examples and how we use function parameters – Strings, numbers, objects, switch statement ect.

Next lets look at how we set a new value in our BehaviourSubject, which requires setting some properties on the feature as we create the feature, and I can talk through a forEach function and actually write out the vanilla code to show how it works under the hood.

Which also looks into how we can access arrays with the index, so demonstrate indexOf method as well.