Angular Report

By: Matthew Skea

Introduction to project

This report has been written to describe my learning outcomes during the development of my of KEAs Web Development elective covering Angular, while using a project developed in class to demonstrate use of Angular in a running application, the uses data from an online API.

Introduction to Angular

Angular

Angular is an Open-Source front-end web application framework developed and maintained by Google. Angular has been developed to address challenges encountered in developing Single-Page Applications(SPA)[1] across modern device platforms.

Single-Page Applications

Single-Page Applications(SPA) are websites that make use of AJAX to update sections of the DOM with new data retrieved from a server. SPAs give a mobile like feel to web applications making the website more responsive to user interaction.

Angular CLI

In order to start developing with Angular, Angular CLI(Command Line Interface) has to be installed on the development machine, along with a lightweight code editor of choice, I used Visual Code. Visual Code is an ideal code editor for for Angular as it recognizes Angular's syntax and has a welly integrated command line interpreter , that can be used to send off Node Package Manager (npm) and Angular CLI(ng) commands.

Angular CLI gives the developer a quick CLI to generate components, routes, services, testing and pipes[2] that are used in Angular.

Angular.io

Angular.io[3] is the main archive for information about how to implement an Angular project. The website covers a guide to get started, tutorials to introduce developers to Angular and a fundamentals guide that gives developers further insight into developing with Angular, and more.

Components

Components control calls to their views, and declare reusable UI building blocks for an application[4]. Components are used used to do the following:

- Use a HTML template and CSS file that is specified in the @Component decoration, the decoration specifies Angular metadata for the component[5].
- Pass data between parent and children components[6]
- Work with data sent from the client, by initializing functions setup within the component. These functions can work with data sent from an online API.

[1] https://en.wikipedia.org/wiki/AngularJS
[2] https://cli.angular.io/
[3] https://angular.io/docs
[4] https://angular-2-training-book.rangle.io/handout/components/
[5] https://angular.io/tutorial/toh-pt1
[6] https://angular.io/guide/component-interaction

```
import { Component }   from '@angular/core';
import { Router }      from '@angular/router';
import { AuthService } from '../auth.service';
import { ViewEncapsulation } from '@angular/compiler/src/core';

@Component({
templateUrl: './login.component.html',
styleUrls: ['./login.component.css']
})
export class LoginComponent {
  message: string;

  constructor(public authService: AuthService, public router: Router) {
    this.setMessage();
  }

  setMessage() {
    this.message = 'Logged ' + (this.authService.isLoggedIn ? 'in' : 'out');
  }

  login() {
    this.message = 'Trying to log in ...';

    this.authService.login().subscribe(() => {
      this.setMessage();
      if (this.authService.isLoggedIn) {
        // Get the redirect URL from our auth service
        // If no redirect has been set, use the default
        let redirect = this.authService.redirectUrl ? this.authService.redirectUrl : '/admin';

        // Redirect the user
        this.router.navigate([redirect]);
      }
    });
  }

  logout() {
    this.authService.logout();
    this.setMessage();
  }
}
```

Figure 1 - demonstrates my login component making use of component functionality mentioned above

Templates

Templates are used to define the mockup HTML of an Angular component. Once the template has been specified in the component, developers can add HTML elements and CSS styling or make use of a downloaded package like Bootstrap or Angular Material to the template.

During or after the HTML element additions, the developer can add Angular functionality to the elements using Angular directives like ngFor and ngIf along with data binding to data models setup in components, to develop an Angular SPA template.

Angular Material

I used Angular Material and some custom css to style my website once loaded. Angular Material[7] is an easy to use UI component library used to style Angular websites, Angular Material has really styling animations and spinners that are easy to use.

app.component.html

```
<!--The content below is only a placeholder and can be replaced.-->
<mat-toolbar color="primary"  class="mat-elevation-z4">
  <mat-toolbar-row>
    <button mat-button routerLink="/beanies" routerLinkActive="active">Beanies</button>
    <!-- TODO: Change hardcoded id  -->
    <button mat-button routerLink="/beanie" routerLinkActive="active" id="create-beanie-e2e">Create beanie</button>
    <button mat-button routerLink="/admin" routerLinkActive="active">Admin</button>
  </mat-toolbar-row>
</mat-toolbar>

<router-outlet></router-outlet>
```

Figure 2 - demonstrates Angular Material applied in the app.component.html view

[7] https://material.angular.io/

Figure 3 - demonstrates Angular Material design the beanie project

Data binding

Data binding is used in Angular to update elements that are bound to specific pieces of data, the elements are updated on the fly by Angular[8] using structural derivatives[9]. Binding DOM is done by surrounding the DOM event name in parentheses and assign a quoted template statement to it[10].

Model binding was used in the project to perform live client side validation, when creating or updating beanie information.

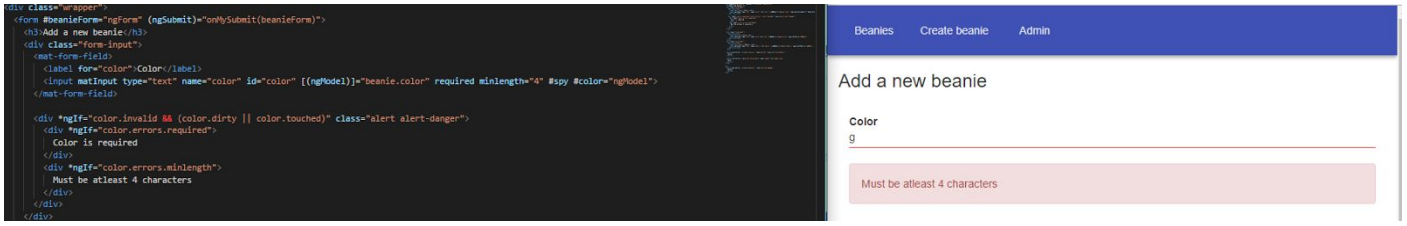

Figure 4 - demonstrates Angular Material applied in the app.component.html view

Forms

Angular handles processing form data, based on the statement that the form is bound to. Validation of the form is handled by Angular in the view, as opposed to setting up client side validation in Javascript when building traditional websites. If the data is valid, it gets forwarded to the component to handle the data being posted.

In the beanie project, a form is used to handle user input when creating, updating or deleting beanies. To see the form mockup and output please refer to figure 3, the forms submission behaviour to the server is handled inside the beanie component.

Routing

Routes tell the website where to go when users perform actions like clicking on links, buttons or pasting urls into their browser that redirect them to another page within a website. In Angular routes are setup with paths, paired to components, the path forwards the user to the components view.

Routing was used throughout the website to direct users between pages and limit users from accessing pages that they do not have the authority to access, that is handled by AuthoGuard.

Authguard

As mentioned before, AuthGuard is used to limit user access based on authorization. The AuthGuard flat is set up in the applications routes.

Authorization is handled by a login component that only permits access to the system once a user clicks on the login button.

[8] https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview
[9] https://angular.io/guide/structural-directives
[10] https://angular.io/guide/user-input

```
const appRoutes: Routes = [
  { path: 'beanies', component: BeanieListComponent, pathMatch: 'full' },
  {
    path: 'beanie/:_id', component: BeanieComponent, pathMatch: 'full',
    canActivate: [AuthGuard]
  },
  {
    path: 'beanie', component: BeanieComponent, pathMatch: 'full',
    canActivate: [AuthGuard]
  },
  {
    path: 'admin', component: AdminComponent, pathMatch: 'full',
    canActivate: [AuthGuard]
  },
  { path: 'admin/a', component: A , pathMatch: 'full'},
  { path: 'admin/b', component: B , pathMatch: 'full'},
  { path: 'login', component: LoginComponent , pathMatch: 'full'},

  {
    path: '',
    redirectTo: '/beanies', // Where to go when no route is specified
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

Figure 5 - demonstrates Angular routing and authorization

HTTP

Most web applications use HTTP to send data across the internet, and Angular handles sending data the same way. When getting or posting data to HTTP Angular handles things differently, with HttpClient API, http can be used within components to perform CRUD functionality with a server.

The HttpClient was used in the project to perform CRUD functionality on beanies.

```
// Save data to the server
this.data.temp.push(this.beanie);

//Setup a custom id for my items
this.beanie.customerId = "100001";
//Push the new beanie to the server
// Create a placeholder for the beanie
const body = this.beanie;
this.http.post('http://angular2api1.azurewebsites.net/api/internships/create',
  body,
  { responseType: 'text' } //This api sends back text
).subscribe(data => {
  //Redirect the user to the beanie list page after the server call is complete
  return this.router.navigate(['beanies']);
});
```

Figure 6 - demonstrates posting a beanie to the web API via HttpClient (http: HttpClient in constructor)

Services

Services focus on presenting data, and delegate data access to a server[11].

In the beanie project, services are used to fetch data from the server, store that data in local memory, manipulate that data, and then the components handling the data, sent new data to the online API.

[11] https://angular.io/tutorial/toh-pt4

```
ngOnInit() {
  //Show the searching spinner
  this.showSpinner = true;
  this.http.get('http://angular2api1.azurewebsites.net/api/internships/getall')
    .subscribe((data: any[]) => {
      //This code runs only once the server responds
      //Return the data back to the component using observables
      this.temp = data;
      this.temp = data.filter(x => x.customerId == "100001"); //Filter only my data from the server

      //Hide the searching spinner
      this.showSpinner = false;

      return data;
    });
}
```

Figure 7 - demonstrates a service being used to fetch data from an online API

End-to-end testing

End-to-end testing is used to test if a website is working correctly from initialization to termination by running a set of automated tests that can be seen both in Visual Codes command line terminal and in a browser[12].

An unforeseen last minute bug is that Angular Material does not simply integrate with end-to-end testing, the tests can no longer find the DOM elements. I am going to have to look deeper into end-to-end testing further, after the handin.

Conclusion

Angular is a powerful, optimized, lightweight framework that can be used to develop websites that respond to user interaction quickly thanks to data binding, while giving developers a modular and component based workflow, that gives way to more reusable code and easy to implement data bound components.

An unforeseen issue was that Angular Material would cause problems for the end-to-end testing. One would of expected the templating framework to integrate more smoothly with Angular like Bootstrap.

In addition to fixing the testing issue, I am going to add observables to my website so that the website has an async workflow when requesting data from the online API.

[12] https://angular.io/guide/testing