# TWSM Documentation

I have built a small portfolio site with the use of Angular, jQuery and Ajax. Ajax was only used because I required using 3 ‘technologies’ and therefor the usage of Ajax is a bit superfluous and unnecessary.

Angular is a development framework to efficiently create single-page applications for mobile and desktop [1]. I used it because I wanted to learn how to use Angular and learn how to create single-page websites since that seems to be the right way to go now adays. Alternatives could have been AngularJS, React or Bootstrap.

I use Angular’s built-in ‘routing’, which handles the page navigation and renders the respective content [2]. This can be seen in the *app.component.html* file with *<router-outlet></router-outlet>*. This is where the router path content will be added/rendered.

```
1  <!DOCTYPE html>
2  <body>
3    <div id="app">
4      <router-outlet></router-outlet>
5    </div>
6  </body>
```

JQuery is a javascript library which allows for easier/simpler DOM-manipulation [3] and was used together with ajax. Ajax, which means ‘Asynchronous JavaScript And XML’, allows for asynchronous, aka non-interrupting, server communication e.g. fetch a file, upload a file etc. This was used to get the ‘templates’ for html-elements, as well as information in a JSON file for those elements.

## The ‘App’ setup

When trying to make a full-page image for the home page I noticed Angular had ~8px margin around the entire page, which led me to find a solution to remove that padding. This was done wrapping the *<router-outlet>* inside a div with an id, in this case ‘app’. Then I could use CSS to fix the issue + also removed the scroll, since I did not want it the pages to be scrollable.

```
1  #app {
2    position: fixed; /* Required to remove the y-axis scroll bar*/
3    overflow: hidden;
4
5    /*Size after after fixing Angular's default padding / margins */
6    width: calc(100% + 20px);
7    height: calc(100% + 30px);
8
9    background-color: var(--black);
10   text-align: center;
11 }
12
13 body {
14   /* Needs to be marked important otherwise it won't change the margins */
15   /* Removes Angular's default margin on the left side and top*/
16   margin: -8px !important;
17   margin-top: -25px !important;
18 }
19
```

Line 2 + 3 is used to remove the scrollbars on the page. Line 6+7 is the new size for the page after fixing Angular’s default margins.

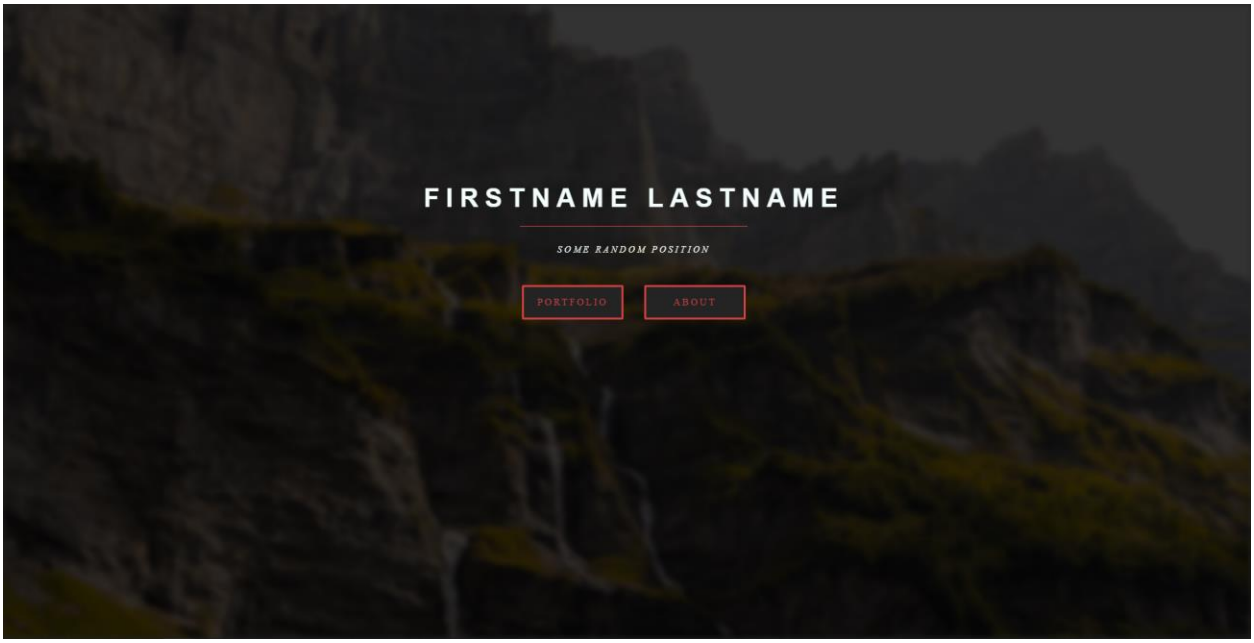
In line 9, it can be seen that I use *var(--black)* which is a reference to a global variable in Angular's equivalent of the normal CSS *:root { … }* called *:host* instead.

```
1  /*Angular's equivalent to ':root' */
2
3  :host {
4    --white: #f4fffd;
5    --black: #242424;
6    --accent: #d64343;
7    --active: rgba(145, 23, 31, 0.45);
8  }
```

*Note: The color previews are from a plugin in my Visual Studio Code.*

## Home Page

The home page consists of a full-size background, which is random using Unsplash's public API for a random image (https://source.unsplash.com/1920x1080), simply because I did not want to bother looking for an image. Along with a title, sub-title and two buttons.




### Full-page background

The full-page background image is an Angular Component. A component is a 'view', which are how screen elements are shown and interacted with. Each component also has a template, which is their html structure. The full-page background image's template is seen to the right.

```
1  <div id="FullpageImage">
2    <!--
3      To be able to remove the scrollbars, this extra div is needed,
4      since overflow does not work with images directly
5      -->
6    <div id="Wrapper">
7      <img src="{{ image_path }}" />
8    </div>
9  </div>
```

In line 6 is a ‘wrapper’ div, which is used to prevent scrollbars appearing on the image, since I learned that *‘overflow: hidden’* does not work on images. In line 7 sees an example of interpolation, which allows for one-way databinding. That way I can re-use the component in multiple places with different images if needed. One more thing is required for the interpolation to work, and that is adding a bit of code to the component, as seen in the code snippet below in line 15:

```
14  export class FullPageBGComponent implements OnInit {
15    @Input('image-path') image_path: string;
16    constructor() {}
17
```

Then I can use the component as follows:

```
15  <app-full-page-bg id="BG" image-path="https://source.unsplash.com/1920x1080">
16  </app-full-page-bg>
```

## Navbar

When moving away from the home page, a ‘navbar’ at the top appears:


The navbar uses Angular’s ‘RouterActive’ which indicates what path is currently active. It is then possible to add a class to the active path navigation elements, as seen on the top right of the image with the underlined ‘Portfolio’. This is achieved by simply adding ‘routerLinkActive=”active” to all the list items that contain a navigation link as such:

```
1   <nav>
2     <ul id="navbar">
3       <li
4         class="nav-button"
5         routerLinkActive="active"
6         [routerLinkActiveOptions]="{ exact: true }"
7       >
8         <a class="nav-link" routerLink="/">Home</a>
9       </li>
10      <li></li>
11      <li class="nav-button" routerLinkActive="active">
12        <a class="nav-link" routerLink="/portfolio">Portfolio</a>
13      </li>
14      <li class="nav-button" routerLinkActive="active">
15        <a class="nav-link" routerLink="/about">About</a>
16      </li>
17    </ul>
18  </nav>
19
```

*Note: The [RouterLinkActiveOptions]=”{exact: true}” is used to prevent partial active router paths would also show up e.g the path “” (home page) will also be marked when the active path is ‘portfolio’ (portfolio page)*

## Portfolio page

The portfolio page is quite boring, it is simply just a small gallery with random images pulled from Unsplash. The most interesting part would be that I use some jQuery to populate the 'src' attribute of all the *<img>* in the gallery to show a random image.

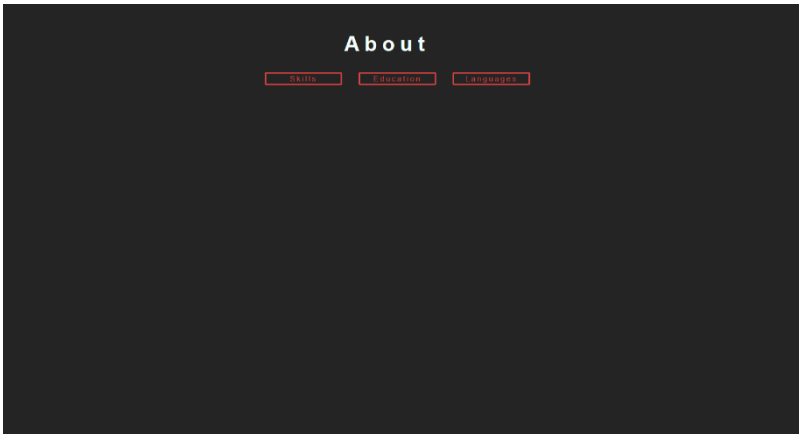
```
12  ngOnInit(): void {
13    var unsplashURL = 'https://source.unsplash.com/random?sig=';
14    var count = -1; //Start from -1 because we increment first to get 0. Not that it matters.
15    var images = $('img');
16
17    $.each(images, function (i, element) {
18      //Generate a unique sig id
19      //Only the last number in the signature is used, so it can only be between 0-9
20      //Therefor simply using a counter will suffice
21      count++;
22
23      //Concatenate to make url
24      var url = unsplashURL + count.toString();
25
26      //Set url to be the img src
27      $(this).attr('src', url);
28    });
29  }
30  }
```

In line 15 I grab a reference to all the images on the page, then in line 17 to 28 I run over each image to generate an Unsplash link for a random image, that then set the link to the 'src' attribute in line 27.

## About page

This is the interesting part with a decent bit of ajax and jQuery.

The page is setup to have 3 buttons, which appends additional content onto the page using Ajax to fetch html templates for the elements. Normally you would simply just use Angular's routing system, however since I still lacked using Ajax and more jQuery I decided to do it this way, which I would never do if it was not because of the requirements for the hand-in, since this is basically just a static website.



The about page contains a ‘content’ div, seen in line 27, which everything is appended to when the user click on any of the 3 buttons (line 8, 14, 20) and calls the async function ‘OpenSection(‘<section name>’, $event).

```
1  <app-navbar></app-navbar>
2
3  <div id="about">
4    <h1 id="title" class="title_about">About</h1>
5
6    <ul>
7      <li>
8        <button class="EmptyButton" (click)="OpenSection('Skills', $event)">
9          Skills
10       </button>
11     </li>
12
13     <li>
14       <button class="EmptyButton" (click)="OpenSection('Education', $event)">
15         Education
16       </button>
17     </li>
18
19     <li>
20       <button class="EmptyButton" (click)="OpenSection('Languages', $event)">
21         Languages
22       </button>
23     </li>
24   </ul>
25
26   <div>
27     <ul id="content"></ul>
28   </div>
```

The *OpenSection* function starts with resetting the sections (like removing existing content) then adding an ‘active’ class to the button just clicked. It then proceeds to do two parallel ajax calls, and await until both are finished, using *Promise.all (…)*. The two ajax calls fetches a html template for the elements, and a json file that contains the information for the elements.

```
24  async OpenSection(section, event) {
25    this.ResetSections();
26
27    //Add active to nav button
28    $(event.target).addClass('active');
29
30    /*
31     * Fetch both the html template and json file in parallel,
32     * but wait for both to be fetched
33     */
34    let [htmlTemplate, jsonFile] = await Promise.all([
35      //Get html template for section
36      $.ajax({
37        url: './assets/templates/' + section.toLowerCase() + '.html',
38        type: 'GET',
39        datatype: 'html',
40        success: function (data) {
41          return data;
42        },
43      }),
44
45      //Get json file with the content
46      $.ajax({
47        url: './assets/about.json',
48        type: 'GET',
49        datatype: 'json',
50        success: function (data) {
51          return data;
52        },
53      }),
54    ]);
```

The fetched files look like the following.

The html template for a ‘skill item’ when opening the ‘skills’ section.

```html
<li class="skill-item">
  <img
    id="img"
    src="./assets/images/conscious-design-ZQ24OL-QpYg-unsplash.jpg"
  />
  <h3 id="name">Test name</h3>
  <p id="mastery">Proficient</p>
</li>
```

Snippet of the json file being fetched:

```json
{
  "skills": [
    {
      "name": "C#",
      "mastery": "★★★★☆",
      "img": "https://upload.wikimedia.org/wikipedia/commons/4/4f/Csharp_Logo.png"
    },
    {
      "name": "Java",
      "mastery": "★★★★☆",
      "img": "https://upload.wikimedia.org/wikipedia/en/3/30/Java_programming_language_logo.svg"
    },
    {
      "name": "JavaScript",
      "mastery": "★★★☆☆",
      "img": "https://upload.wikimedia.org/wikipedia/commons/thumb/d/d4/Javascript-shield.svg/1200px-Javascript-shield.svg.png"
    },
```

Every ID in the html file can also be found in the json file. This is done to easily map the json values into the correct places on the html element.

The is being done with the following code, part of the *OpenSection()* function

l. 62 for each element in the json belonging to the section e.g. “skills” do the following:

l. 63 I am copying the html template to be able to change it properly.

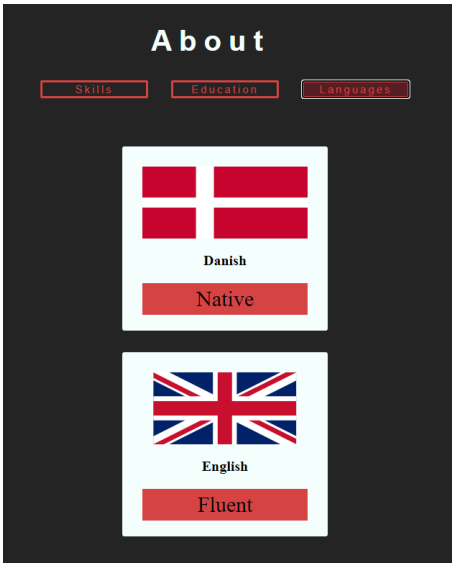
l. 64 For each key (e.g. “name”, “mastery”, “img”):

l. 66 – 74 map the values from the key to the attribute / text of the html.

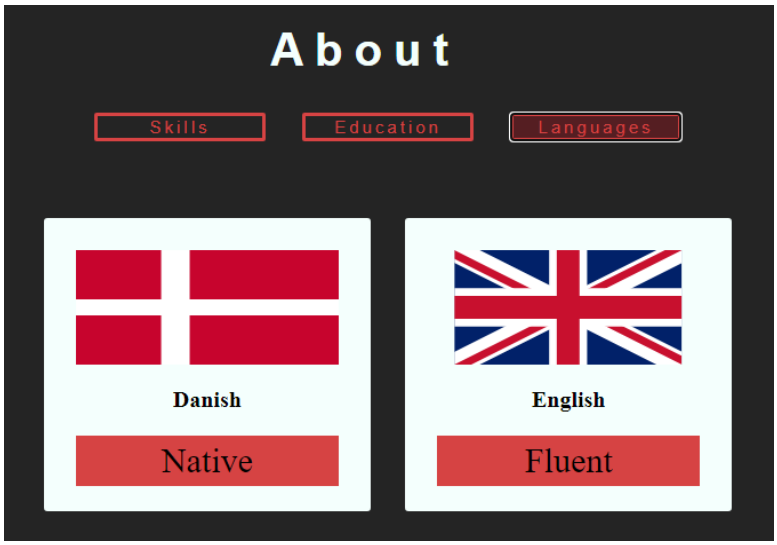
In l 75 I append the html to a div with the id ‘content’.
Lastly, I add a no-wrap if the section is languages to prevent the flexbox from wrapping with only 2 elements.

```
61    console.log(jsonFile[section.toLowerCase()]);
62    jsonFile[section.toLowerCase()].forEach((element) => {
63      var html = $(htmlTemplate);
64      Object.keys(element).forEach((key) => {
65        console.log(key + ' : ' + element[key]);
66        if (key == 'img') {
67          $(html)
68            .find('#' + key)
69            .attr('src', element[key]);
70        } else {
71          $(html)
72            .find('#' + key)
73            .text(element[key]);
74        }
75        $('#content').append(html);
76
77        /* Don't wrap the languages since there are only 2 */
78        if (section.toLowerCase() == 'languages') {
79          $('#content').addClass('no-wrap');
80        }
81      });
82    });
```

The difference between wrap and no-wrap be seen below:




*Wrap*




*No Wrap*

The very last thing I had to do to make the CSS work for the about page, is to add one line to the about component, being l. 17:

```
4   @Component({
5     selector: 'app-about',
6     templateUrl: './about.component.html',
7     styleUrls: [
8       './about.component.css',
9       '../app.component.css',
10      '../big-nav-button/big-nav-button.component.css',
11      './edu-item.css',
12      './lang-item.css',
13      './skill-item.css',
14    ],
15    //Prevents encapsulation of css to allow for dynamic injected html to use css
16    //from the styleUrls above
17    encapsulation: ViewEncapsulation.None,
18  })
```

This tells Angular not to encapsulate the styleUrls and therefor allows the non-angular html templates to still work with the css templates in Angular [5].

# References

[1] https://angular.io/docs

[2] https://angular.io/guide/router

[3] https://jquery.com/

[4] https://angular.io/guide/architecture

[5] https://angular.io/api/core/ViewEncapsulation

[6] https://unsplash.com/license

All images used can be found in the source code, and Unsplash images are random but are copyright free and they do not require to be credited [6]. The only non-random Unsplash image is the following link, used as placeholder in the json file:
https://images.unsplash.com/photo-1588250004030-a30208876432