Angular Training Practical Approach

Installation

You're first going to need to install the Angular CLI (Command Line Interface) tool, which helps you start new Angular 9 projects as well as assist you during development. In order to install the Angular CLI, you will need **Nodejs**. Make sure you install this with the default options and reload your command line or console after doing so.

In your command line, type:

```
> npm install -g @angular/cli
```

Once complete, you can now access the CLI by simply starting any commands with **ng**.

Hop into whichever folder you want to store your projects, and run the following command to install a new Angular 9 project:

```
> ng new ng7-pre
```

It's going to present you with a couple questions before beginning:

```
? Would you like to add Angular routing? Yes

? Which stylesheet format would you like to use? SCSS    [ http://sass-lang.com    ]
```

It will take a minute or two and once completed, you can now hop into the new project folder by typing:

```
> cd ng7
```

Open up this project in your preferred code editor (I use Visual Studio Code, and you can launch it automatically by typing *code.* in the current folder), and then run this command to run a development server using the Angular CLI:

```
> ng serve -o
```

-o is for *open*, this flag will open your default browser at *http://localhost:4200*. **Tip:** You can type **ng** to get a list of all available commands, and **ng [command] --help** to discover all their flags.

Awesome! If all went smooth, you should be presented with the standard landing page template for your new Angular 9 project:

Angular 9 Components

The most basic building block of your Angular 9 application (and this is a concept that's not new) is the component. A component consists of three primary elements:

- The HTML template
- The logic
- The styling (CSS, Sass, Stylus, etc..)

When we use the Angular CLI to start a new project, it generates a single component, which is found in **/src/app/**:

```
/app.component.html

/app.component.scss

/app.component.ts
```

While we have three files here that represent the three elements above, the **.ts** (TypeScript) is the heart of the component. Let's take a look at that file:

```typescript
import { Component } from '@angular/core';




@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.scss']

})

export class AppComponent {

  title = 'ng9-pre';

}
```

Here, because this is a component, we're importing Component from the @angular/core library, and then it defines what's called a *@Component decorator*, which provides configuration options for that particular component.

As you can see, it's referencing the location of the HTML template file and the CSS file with the *templateUrl* property and the *styleUrls* property.

Angular Training Practical Approach

The logic of the component resides in the class at the bottom. As you can see, the CLI starter template simply defines a single property called **title**.

Let's use the Angular CLI to create our own components that we'll need going forward. In the console, issue the following commands:

```
> ng generate component nav

// output

> ng g c about

// output

> ng g c contact

// output

> ng g c home

// output
```

Notice we first use the full syntax to generate a component, and then we use a shorthand syntax, which makes life a little bit easier. The commands do the same thing: generate components.

When we generate components this way, it will create a new folder in the */src/* folder based on the name we gave it, along with the respective template, CSS, .ts and .spec (for testing) files.

Angular 8 Templating

You may have noticed that one of the components we generated was called *nav*. Let's implement a header bar with navigation in our app!

The first step is to visit the **app.component.html** file and specify the following contents:

```
<app-nav></app-nav>

<section>

  <router-outlet></router-outlet>

</section>
```

So, we've removed a bunch of templating and placed in *<app-nav></app-nav>*, what does this do and where does it come from?

Angular Training Practical Approach

Well, if you visit **/src/app/nav/nav.component.ts** you will see that in the component decorator, there's a *selector* property bound to the value of *app-nav*. When you reference the selector of a given component in the form of a custom HTML element, it will nest that component inside of the component it's that's referencing it.

If you save the file you just updated, you will see in the browser we have a simple, *nav works!* And that's because the *nav.component.html* file consists of a simple paragraph stating as much.

At this point, let's specify the following HTML to create a simple navigation:

```html
<header>

  <div class="container">

    <a routerLink="/" class="logo">apptitle</a>

    <nav>

      <ul>

        <li><a routerLink="/">Home</a></li>

        <li><a routerLink="/about">About</a></li>

        <li><a routerLink="/contact">Contact us</a></li>

      </ul>

    </nav>

  </div>

</header>
```

The only thing that might look a little strange is **routerLink**. This is an Angular 7 specific attribute that allows you to direct the browser to different routed components. The standard *href* element will not work.

While we're here on the subject of templating, what if we wanted to display properties that are coming from our component? We use what's called interpolation.

Make the following adjustment to our template:

```html
<!-- From: -->

<a routerLink="/">myapp</a>
```

Angular Training Practical Approach

```
<!-- To: -->

<a routerLink="/">{{ appTitle }}</a>
```

Interpolation is executed by wrapping the name of a property that's defined in the component between {{ }}.

Let's define that property in **nav.component.ts**:

```
export class NavComponent implements OnInit {

  appTitle: string = 'myapp';

  // OR (either will work)

  appTitle = 'myapp';

  constructor() { }

  ngOnInit() {

  }

}
```

You can use the TypeScript way of defining properties or standard JavaScript. Save the file and you will see **myapp** is back in the template.

There's a lot more to templating, but we will touch on those topics as we continue. For now, let's apply style to our header.

First, let's visit the global stylesheet by opening **/src/styles.scss** and define the following rulesets:

```
@import url('https://fonts.googleapis.com/css?family=Montserrat:400,700');



body, html {

    height: 100%;

    margin: 0 auto;

}
```

```scss
body {

    font-family: 'Montserrat';

    font-size: 18px;

}



a {

    text-decoration: none;

}

.container {

    width: 80%;

    margin: 0 auto;

    padding: 1.3em;

    display: grid;

    grid-template-columns: 30% auto;

    a {

        color: white;

    }

}

section {

    width: 80%;

    margin: 0 auto;

    padding: 2em;

}
```

Visit **nav/component.scss** and paste the following contents:

Angular Training Practical Approach

```css
header {

    background: #7700FF;

    .logo {

        font-weight: bold;

    }

    nav {

        justify-self: right;

        ul {

            list-style-type: none;

            margin: 0; padding: 0;

            li {

                float: left;

                a {

                    padding: 1.5em;

                    text-transform: uppercase;

                    font-size: .8em;

                    &:hover {

                        background: #8E2BFF;

                    }

                }

            }

        }

    }

}
```

If you save and refresh, this should be the result in the browser:

Angular 9 Routing

Now that we have a navigation, let's make our little app actually navigation between our components as needed.

Open up **/src/app/app-routing.module.ts** and specify the following contents:

```
/ Other imports removed for brevity

import { HomeComponent } from './home/home.component';

import { AboutComponent } from './about/about.component';

import { ContactComponent } from './contact/contact.component';

const routes: Routes = [

  { path: '', component: HomeComponent },

  { path: 'about', component: AboutComponent },

  { path: 'contact', component: ContactComponent },

];

// Other code removed for brevity
```

As we can see here, we're defining importing our components and defining an object for each route inside of the **routes** constant. These route objects also accept other properties, which allow you to define URL parameters, but because our app is simple, we won't be doing any of that.

Save this file and try clicking on the links above. You will see that each of the respective component's HTML templating shows up in the **<router-outlet></router-outlet>** defined in *app.component.html*.

This is what the result should look like in the browser at this point:

You now know enough about Angular 9 to create a very simple website with routing! But let's learn more than that.

Angular 9 Event Binding

In the next several sections, we're going to use our **/src/app/home** component as a playground of sorts to learn features specific to Angular 9.

One of the most used forms of event binding is the click event. You often need to make your app respond when a user clicks something, so let's do that!

Visit the **/src/app/home/home.component.html** template file and specify the following:

```html
<h1>Home</h1>

<button (click)="firstClick()">Click me</button>
```

You define an event binding by wrapping the event between (), and calling a method. You define the method in the **home.component.ts** file as such:

```typescript
export class HomeComponent implements OnInit {

  constructor() { }

  ngOnInit() {

  }
```

```
  firstClick() {

    console.log('clicked');

  }

}
```

Save it, get out the browser console (CTRL+SHIFT+i) and click on the button. The output should show *clicked*. Great!

You can experiment with the other event types by replacing *(click)* with the names below:

```
(focus)="myMethod()"

(blur)="myMethod()"

(submit)="myMethod()"

(scroll)="myMethod()"


(cut)="myMethod()"

(copy)="myMethod()"

(paste)="myMethod()"


(keydown)="myMethod()"

(keypress)="myMethod()"

(keyup)="myMethod()"


(mouseenter)="myMethod()"

(mousedown)="myMethod()"

(mouseup)="myMethod()"


(click)="myMethod()"
```

```
(dblclick)="myMethod()"



(drag)="myMethod()"

(dragover)="myMethod()"

(drop)="myMethod()"
```

Angular 9 Class & Style Binding

Sometimes you may need to change the appearance of your UI from your component logic. There are two ways to do this, through class and style binding.

There are a lot of different methods you can use to control class binding, so we won't cover them all. But I will cover some of the most common use cases.

Let's say that you want to control whether or not a CSS class is applied to a given element. Update the **h1** element in *home.component.html* to the following:

```html
<h1 [class.gray]="h1Style">Home</h1>
```

Here, we're saying that the CSS class of *.gray* should only be attached to the h1 element if the property *h1Style* results to true. Let's define that in the *home.component.ts* file:

```typescript
h1Style: boolean = false;

  constructor() { }

  ngOnInit() {

  }

  firstClick() {

    this.h1Style = true;

  }
```

Let's also define the *.gray* class in this component's scss file:

```scss
.gray {

    color: gray;

}
```

Angular Training Practical Approach

Save it, and you can now click on the *click me* button to change the color of the Home title.

What if you wanted to control multiple classes on a given element? You can use *ngClass*. Modify the home component's template file to the following:

```
<h1 [ngClass]="{

  'gray': h1Style,

  'large': !h1Style

}">Home</h1>
```

Then, add the *large* ruleset to the *.scss* file:

```
.large {

    font-size: 4em;

}
```

Now give it a shot in the browser. *Home* will appear large, but shrink down to the regular size when you click the button. Great!

You can also control appearance by changing the styles directly from within the template. Modify the template as such:

```
<h1 [style.color]="h1Style ? 'gray': 'black'">Home</h1>
```

Refresh and give this a shot by clicking the button.

Like *ngClass()* there's also an *ngStyle()* that works the same way:

```
<h1 [ngStyle]="{

  'color': h1Style ? 'gray' : 'black',

  'font-size': !h1Style ? '1em' : '4em'

}">Home</h1>
```

Give this a go! Awesome!