Angular 6 Training Course

Exercise P-Angular6

- This document describes some of the new features added to Angular version 6 in May 2018.

Setup

- To build Angular 6 projects, install the latest version of the Angular CLI and the latest version of Node.
- Angular 6 CLI requires Node v9.8 or later. Check the Node version

```
node --v
```

- Install the latest version of the Angular CLI.

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

- *Mac users may need to prefix this command with sudo.*
- Check the Angular CLI version is 6 or later.

```
ng --version
```

Bootstrap

- The Angular 6 CLI makes it easier to add **Bootstrap** to an Angular project.
- Create a new project.

```
ng new boot
```

- Add Bootstrap support to the project

```
ng add @ng-bootstrap/schematics
```

- This adds Bootstrap-specific code to your main app.module.ts file.

```
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';

@NgModule({
  imports: [
      BrowserModule, NgbModule.forRoot()
  ]
})
```

- Bootstrap-specific code can now be used in templates such as app.component.html:

```
// Progress Bar ========================
<ngb-progressbar showValue="true" type="success" [value]="25">
</ngb-progressbar>
<ngb-progressbar [showValue]="true" type="info" [value]="50">
</ngb-progressbar>

// Accordion ============================
<ngb-accordion [closeOthers]="false" activeIds="copenhagen">
  <ngb-panel id="oslo" title="Oslo">
    <ng-template ngbPanelContent>
      Oslo, the capital of Norway....
    </ng-template>
  </ngb-panel>
   <ngb-panel id="helsinki" title="Helsinki">..</ngb-panel>
```

Service Workers (Progressive Web Apps)

- Service Workers are custom Javascript code which run before an HTTP request to a web page.
- They can be used to cope with the case where an internet connection is lost.
- Create an Angular project

```
ng new worker
```

- Add Service Worker support

```
ng add @angular/pwa
```

- This command will change app.module.ts:

```
import { ServiceWorkerModule } from '@angular/service-worker';
```

imports: [ServiceWorkerModule.register('/ngsw-worker.js', { enabled: environment.production })],

- Add an image to the project assets folder and display that image in the main template. *When you test the Service Worker later, this image should still display even if the page is offline.*

```
<img src="assets/book.jpg"/>
```

- Build the project with the production flag on

```
ng build --prod
```

- This will create a folder dist/worker inside your folder folder.
- This folder needs to be served from an HTTP web server.
- One solution is to globally install http-server

```
npm install http-server -g
```

- Change director to the dist/worker folder and run a local web server from there.

```
cd dist/worker
http-server -p 4000
```

- Open a browser at localhost:4000
- The Angular page should now run.
- In the Chrome web tools open the Network tab.
- Select the offline checkbox and reload the page.
- Normally this would cause an internet connection error.
- But the Service Worker intercepts your HTTP request and serves up a cached version of the page if it cannot connect to the server.
- For more information about Service Workers, check this eBook by Jeremy Keith : https://abookapart.com/products/going-offline

Service Providers

- Services are utility classes, instances of which can be used in components using **Dependency Injection.**
- In Angular 5, you need to define any services used in the **providers** array of your **app.module.ts** file.

```
providers:[ DataService ]
```

- Forgetting to define this provider is a common source of bugs.
- Angular 6 removes the need to define this providers array.
- Instead you can defined a providedIn property in the service.
- This change makes the service more portable/self-contained.

```
    @Injectable(
        { providedIn : 'root' }
    )
    export class DataService { .. }
```

Observables , RxJS

- The syntax for using Observables has changed. *The course examples have been updated to reflect this.*
- The syntax of import statements has changed:

```
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
```

- Some functions such as map, now need to wrapped within a pipe method:

```
pipe( map( .. ))
```

Angular Elements

- Angular 6 let you package a component as a **custom element**.

- Custom elements are a web-standard for creating new custom HTML elements.
- This allows you to add an Angular component to a web page which is not itself an Angular project.
- Create a new project and then add support for Angular elements:

```
ng new project
ng add @angular/elements
```

- The main changes to create a custom element happen in **app.module.ts**

```
    import { NgModule, Injector } from '@angular/core';
    import { createCustomElement } from '@angular/elements';
```

- This example assumes the component is named AppComponent.
- No instance of the component is created within your Angular code. An instance will be created in the HTML page where it is used.
- So the **bootstrap** array is empty and a new **entryComponents** array is defined:

```
@NgModule({
    ....
  bootstrap: [],
  entryComponents: [AppComponent]
})
```

- The AppModule adds code which registers the component as a custom element called my-city with the browser on the web page where it is used.

```
    export class AppModule {
      constructor(private injector: Injector) {}
      ngDoBootstrap() {
        const el = createCustomElement(SomeComponent, {
injector: this.injector });

        customElements.define('my-city', el);
      }
  }
```

- Edit **app.component.ts**.
- We want the component to accept the name of a city as an input. In this example, Seville is also defined as the default value.

```
export class AppComponent {
  @Input() name = 'Seville';
}
```

- Create a simple template in app.component.html to display this name.

```
<h1>{{ name }}</h1>
```

- We want the component to use style encapsulation, but it will be running within a normal web page, not an Angular app.
- We set it to Native to use the Shadow DOM.

```
import { ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'my-city',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.Native
})
```

- We can generate a production build, disabling random cache-busting file names:

```
ng build –prod --output-hashing none
```

- This will create a new folder in the dist folder, based on the name of the project. In this case it creates **dist/project**.
- Edit the index.html file to create an instance of the component:

```
  <my-city name="Granada"></my-city>

  <script type="text/javascript" src="runtime.js"></script>
  <script type="text/javascript" src="polyfills.js"></script>
  <script type="text/javascript" src="scripts.js"></script>
  <script type="text/javascript" src="main.js"></script>
```

- *Note: the multiple JS files here could be bundled/minified into a single JS file.*
- To view this page, it needs to be served from an HTTP url.
- In the dist/project folder, open an HTTP server

```
    http-server
```

- Open a browser at localhost:8080
- The Angular component should now run as a custom element on an ordinary web page.
- Check browser support for custom elements at https://caniuse.com/#search=custom%20ele