Angular
Lab 3
Built-in Directives

1 Lab setup

Make sure you have the following items installed

- Latest version of Node and NPM (note: labs are tested with Node v16.13 and NPM v8.1 but should work with later versions with no or minimal changes)
- Latest version of TypeScript (note: labs are tested with TypeScript v4.5.4 but should work with later versions with no or minimal changes)
- Latest version of Angular (note: labs are tested with Angular v13.1 but should work with later versions with no or minimal changes)
- Visual Studio Code (or a suitable alternative IDE for Angular/TypeScript)
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

In each of the main lab folders, there are two subfolders: `changes` and `final`.

- The `changes` subfolder holds the source code and other related files for the lab. The creation of the Angular app will proceed in a step-wise fashion in order to clearly demonstrate the various features of the framework. The various project source files in this folder are numbered in a way to reflect this gradual construction. For e.g. `xxx.v1.html`, `xxx.v2.html`, etc represents successive changes that are to be made to the `xxxx.html`

- The `final` subfolder holds the complete Angular project. If there was some issue in completing the step-wise construction of the app using the files from the `changes` subfolder, you can use this to run the final working version.

The project folder in `final` is however missing the crucial `node_modules` subfolder which contains all the dependencies (JavaScript libraries) that your Angular app needs in order to be built properly. These dependencies are specified in the `package.json` file in the root project folder. They are omitted in the project commit to GitHub because the number and size of the files are prohibitively large.

In order to run the app via the Angular development server (`ng serve`), you will need to create this subfolder containing the required dependencies. There are two ways to accomplish this:

a) Copy the project root folder to a suitable location on your local drive and run `npm install` in a shell prompt in the root project folder. NPM will then create the `node_modules` subfolder and download the required dependencies as specified in `package.json` to populate it. This process can take quite a while to complete (depending on your broadband connection speed)

b) Alternatively, you can reuse the dependencies of an existing complete project as most of the projects for our workshop use the same dependencies. In this case, copy the `node_modules` subfolder from any existing project, and you should be able to immediately start the app. If you encounter any errors, this probably means that there are some missing dependencies or Angular version mismatch, in which case you will then have to resort to `npm install`.

The lab instructions here assume you are using the Chrome browser. If you are using a different browser (Edge, Firefox), do a Google search to find the equivalent functionality.

2 Generating a new Angular project

The source code for this lab can be found in the `changes` subfolder of `Directives-Demo`

In either a separate command prompt (or shell terminal) or in an embedded terminal in Visual Studio Code, create a new Angular project with:

```
ng new directivesdemo
```

Press enter to accept the default values for all the question prompts that follow regarding routing and stylesheet.

Navigate into the root project folder from the terminal with:

```
cd directivesdemo
```

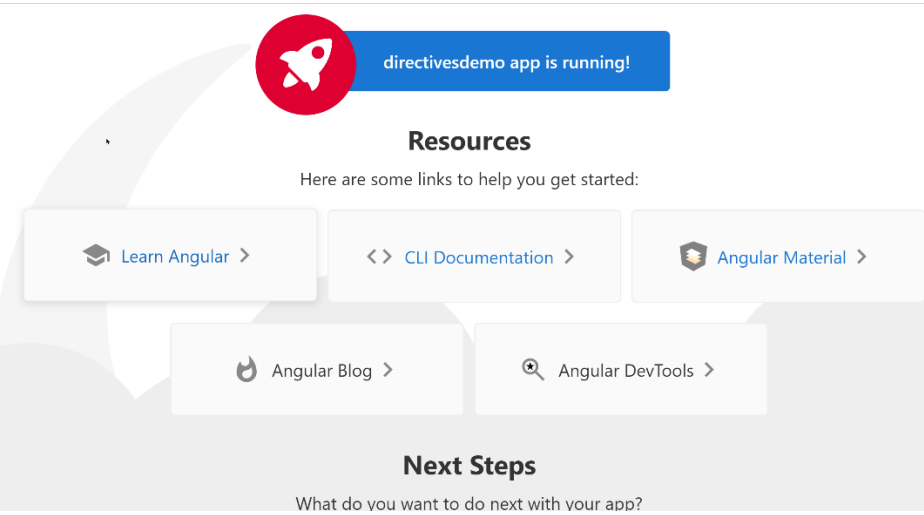
Build the app and serve it via Angular's live development server by typing:

```
ng serve
```

The final output line from this should indicate that the compilation process was successful and identify the port that the live development server is currently serving up the Angular app dynamically from (by default this will be port 4200)

Open a browser tab at:
http://localhost:4200/

You should be able to see the default landing page for all new autogenerated Angular projects.



To stop the development server, type Ctrl+C in the terminal window that it is running in. You can restart again anytime by typing `ng serve` in the project root folder.

3 NgClass

The source code for this lab can be found in the `changes` subfolder of `Directives-Demo`

NgClass is conceptually similar to standard [class] binding: it permits dynamic binding of CSS classes to a specified HTML element

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v1.ts`

`app.component-v1.html`

`app.component-v1.css`

The template expression that is used with [ngClass] can return all these items:

- A space-delimited string of class names.
- An object with class names as the keys and truthy or falsy expressions as the values.
- An array of class names

and is thus functionally identical to the multiple class binding [class] syntax that we used for property binding in an earlier lab.

Verify that the classes on the 3 paragraphs change accordingly. Notice that for the 3rd case where there are two classes with conflicting style rules now applied to the <p> element simultaneously (`normal` and `medium`), the style rule for the class that appears last in the list of class names takes effect.

You can change the truthy/falsy values of the object used in the 2nd approach and verify that the correct classes appear on the affected element.

To add or remove a single class, it is recommended to use the [class] binding covered in a previous lab.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v1-2.ts`

`app.component-v1-2.html`

Test out making dynamic changes to the classes applied to the various <p> elements by clicking on the corresponding buttons.

Here, we can see the primary difference between using ngClass and standard [class] binding to dynamically set class names on a element. For the case when we are using an object with class names as the keys and truthy or falsy expressions as the values, using ngClass allows dynamic updating of the classes when any of the object key values changes while standard [class] binding does not.
This is because the standard [class] binding requires a change in the identity of the object used in the template expression in order for there to be a dynamic change in the binding during run time. If you only change a property within the object, there will be no update in the binding.

References:

https://angular.io/guide/built-in-directives#ngClass
https://www.tektutorialshub.com/angular/angular-ngclass-directive/
https://www.netjstech.com/2020/04/angular-ngclass-directive-with-examples.html

4 NgStyle

NgStyle is conceptually similar to standard [style] binding: it permits dynamic binding of CSS properties to a specified HTML element

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v2.ts`

`app.component-v2.html`

For [ngStyle] binding, we can only use an object whose key and values are style name and style values respectively. You cannot assign a string containing these style rules as was the case in normal [style] binding that we studied in a previous lab.
Verify that changing the value of the style properties in the object causes the inline styles to be updated dynamically.

References:

https://codecraft.tv/courses/angular/built-in-directives/ngstyle-and-ngclass/#_ngstyle
https://www.netjstech.com/2020/04/angular-ngstyle-directive-with-examples.html
https://www.tektutorialshub.com/angular/angular-ngstyle-directive/

5 NgIf

The NgIf directive is a structural directive that allows us to add / remove DOM elements in a template based on whether the template expression it is bound to evaluates to a truthy or falsy value. If it evaluates to a truthy value, then the element it is attached to (and all of the descendants in its subtree) is inserted into the template DOM, otherwise it is removed.

NgIf, like all structural directives, is prefixed with a * when included in the template.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v3.ts`

`app.component-v3.html`

The effect of NgIf can also be achieved by binding a conditional template expression to the `hidden` property of an element

There is an important difference: If the template expression evaluates to a falsy value, NgIf removes the entire DOM element (along with its subtree) from the template DOM. This frees up any resources related to that element (and its subtree). The `hidden` property merely toggles the visibility of the element in the rendered DOM, but does not remove it from the template DOM.
You can verify this in the Elements view of the DevTools when toggling the First Checkbox.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v3-2.ts`

`app.component-v3-2.html`

Notice that ngIf is preceded with a *, which indicates that it is a structural directive.

We can also add a ngIf else block using the <ng-template> placeholder for a HTML snippet.
We can also extend this to a ngIf then else block using the <ng-template> placeholder

References:

https://angular.io/guide/built-in-directives#adding-or-removing-an-element-with-ngif
https://www.tektutorialshub.com/angular/angular-ngif-directive/

6 NgFor

The ngFor directive iterates over a data collection (e.g. array, list, etc) and creates snippet of HTML (containing one or more elements) for each item in the collection. It helps to simplify the construction of HTML blocks which involve repetition of standard elements (e.g. lists or tables).

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v4.ts`

`app.component-v4.html`

`app.component-v4.css`

Add this file to `src/app` from `changes`:

`Employee.ts`

This represents a sample domain model class, which models application data for the business domain that the app is being built for.

Notice that ngFor is preceded with a * (just like NgIf), which indicates that it is a structural directive.
`let <item> of <items>;`
where `item` is called the template input variable (this is different from a template reference variable). This represents the currently iterated item from the collection `<items>`. The scope of the template input variable item is within the HTML block nested inside the element that ngFor is attached to. You can access it anywhere within that, but not outside of it.

References:

https://angular.io/guide/built-in-directives#listing-items-with-ngfor
https://www.tektutorialshub.com/angular/angular-ngfor-directive/

6.1 Accessing index of item in collection

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v5.ts`

`app.component-v5.html`

`app.component-v5.css`

ngFor exposes several values, which help us to fine-tune the display. We assign these values to a local variable and use it in the nested HTML block. The list of exported values provided by ngFor directive

- index: number: The zero-based index of the current element in the collection.
- count: number: The total no of items in the collection
- first: boolean: is set to True when the item is the first item in the collection.
- last: boolean: Is set to True, when the item is the last item in the collection.
- even: boolean: is set to True when the item has an even index in the collection.
- odd: boolean: is set to True when the item has an odd index in the collection.

Here we set the local variable i to the index and use this in the HTML block to allow us to keep track of which button has been clicked on.

6.2 Styling using first, last, odd and even exported values

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v6.html`

`app.component-v6.css`

Here we use the `first, last, even` and `odd` exported values to style the divs surrounding the respective buttons

6.3 NgFor with child components

In the root folder of the project created earlier, type this command to generate a new component:

```
ng generate component Employee
```

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v7.html`

`app.component-v7.ts`

`employee.component-v7.ts`

`employee.component-v7.html`

`employee.component-v7.css`

In the root template, we use a ngFor to iterate through the Employee objects in the array, passing each object as well as its index in the collection to a newly created child component <app-employee> via two separate property bindings.

Make the following changes:

`app.module-v8.ts`

`app.component-v8.html`

`app.component-v8.ts`

`app.component-v8.css`

`employee.component-v8.ts`

`employee.component-v8.html`

`employee.component-v8.css`

Next, we provide appropriate input fields in the child template to capture new values for the Employee object passed down to the respective child component.
Notice that we use NgModel for two-way binding for both the `name` and `age` properties of the Employee object, which results in a change in the original object in the array in the parent component since data is passed by reference. We need to modify AppModule accordingly to add in the FormsModule in order to use NgModel.
However, we are not able to perform two-way binding in this way for radio buttons, so we need a event handler method in the child component that checks whether the married radio button is ticked to set the new value for the `isMarried` property.
The index of the employee object from the original array is transmitted back in the event binding from the child component so that this can be used in the dynamic binding in the [ngClass] directive to highlight the specific row that was changed.

7 Combining NgIf and NgFor

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v9.html`

`app.component-v9.ts`

Angular doesn't support more than one structural directive on the same element, so there is a need to use the <ng-container> helper element here. This element itself does not appear in the template DOM, it serves merely as a template place holder to hold a structural directive for the purposes of combination.

Here we have a ngIf followed by a ngFor, so either the entire table contents is either shown or not shown at all.

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v10.html`

`app.component-v10.ts`

Here we have a ngFor followed by a ngIf, so only specific employee objects are rendered in the table rows, depending on the evaluation of a conditional expression.

References:

https://www.netjstech.com/2021/04/how-to-use-ngfor-ngif-same-element-angular.html

8 NgSwitch

Modify the following files in `src/app` with the latest update from `changes`:

`app.component-v11.html`

`app.component-v11.ts`

ngSwitch is a structural directive that works together in conjunction with ngSwitchcase and ngSwitchDefault directives to provide the functionality of a standard switch statement in JavaScript.

Some points to bear in mind:

- The ngSwitchCase and ngSwitchDefault directives must be nested within the ngSwitch directive
- Angular will render all HTML blocks for all ngSwitchCases whose evaluated expression values match the evaluated switch expression values
- The HTML blocks of non-matching ngSwitchCases are not hidden, but removed from the DOM
- If there are no matches for any ngSwitchCases, the HTML block associated with the ngSwitchDefault directive is displayed
- You can place one or more than one ngSwitchDefault anywhere inside the container element and not necessarily at the bottom.

References:

https://www.tektutorialshub.com/angular/angular-ngswitch-directive/