Unit IV – Introduction to Client-side JS Framework – Part I

Agenda

- Introduction to Angular
    - Merits of Model View Controller (MVC) at Client-side over Server-side
    - Single Page Application (SPA)
    - Progressive Web Application (PWA)
- Needs, Features & its Evolution
- Setup and Configuration
- Components and Modules
    - Work of Change Detection in Components
- Templates
- Directives

Introduction to Angular

Education, Training and Assessment

Why Angular

- It exclusively used to build single page applications
- Open source JavaScript framework
- Supports both mobile and desktop web applications
- TypeScript is used to write Angular code
  - TypeScript is Microsoft's extension for JavaScript
  - Supports object oriented features and has strong typing system
  - Supports many features and IDE's
  - TypeScript code is compiled to JavaScript code, to make browser understand the code
- Include model-view-controller capability by providing a MVC framework

MVC on server-side

- MVC or Model-View-Controller design pattern was defined in 1979
- Several frameworks sporting this pattern could help in server side logic building and other operations, for ex : Struts, SpringMVC,Rails etc.
- Web-applications were rich on the server-side with all operations such as validations, event handling, generating response based on request from client etc. happening on the side of the server

Here MVC architecture helped by separating the responsibilities of :

- Routing to an appropriate handler which can read the request,
- Rendering template using view,
- Creating model to provide response based on action
- And coordination between the three when a request comes

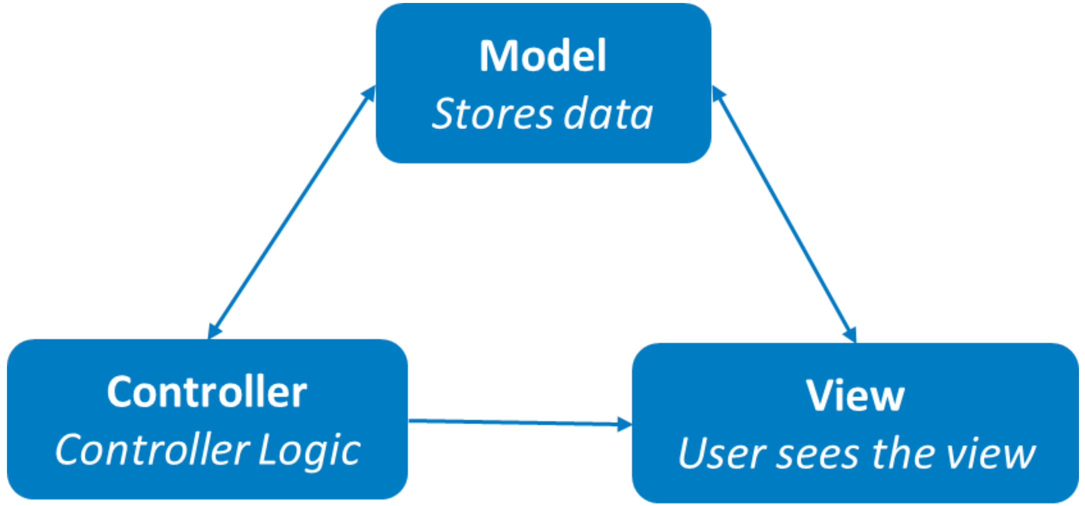
MVC paradigm on client

- Rich-client side controlled applications started becoming famous due to the emergence of a new breed of JavaScript frameworks like Prototype, jQuery etc.
- This client-side rich approach had several advantages such as –
- Elegant, simple and cheaper to code
- State changes and handlers are local and extensive use of AJAX for making calls to the server
- Validation, event handling etc. happens on client side, thus reducing server load

Another reason for this change from rich server-side to rich client-side applications is because the browser now has become a **powerful** tool which can perform much of the logic and coding at the client side with ease thanks to the JavaScript frameworks and the server side has been reduced to performing data transactions primarily in the form of JSON

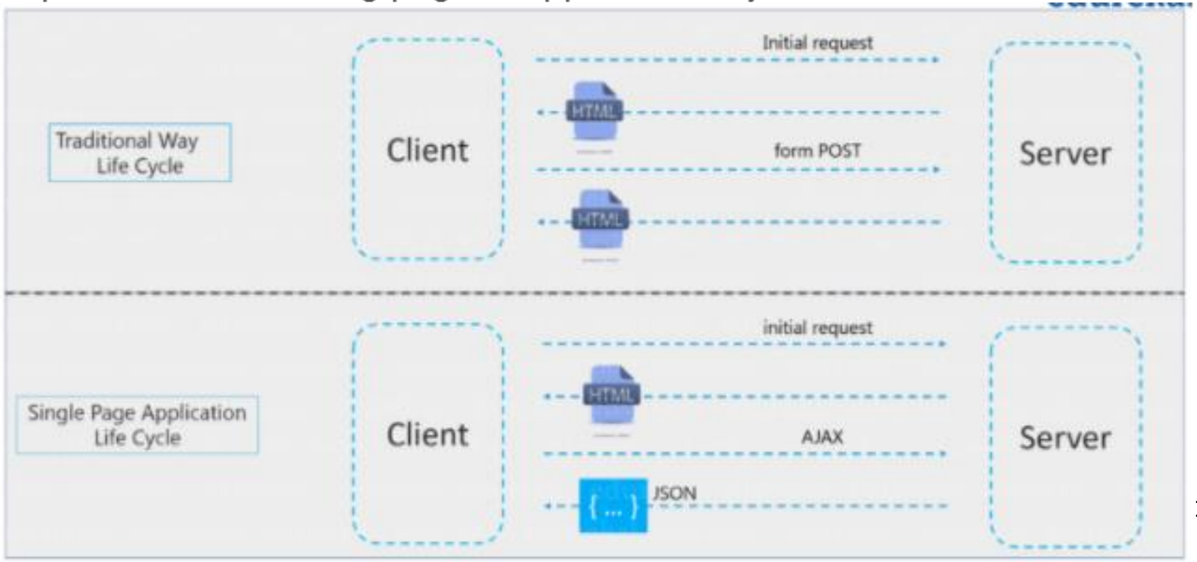
Angular MVC

- AngularJS is MVC which means we have dedicated model, view and controller defined.
- MVC structure of AngularJS enables creation of better maintainable code.



Single Page Application (SPA)

- ***Single-page application*** (or ***SPA***) are applications offer more dynamic interactions resembling native mobile and desktop apps.
- The most notable difference between a regular website and SPA is the reduced amount of page refreshes.
- SPAs have a heavier usage of AJAX- a way to communicate with back-end servers without doing a full page refresh to get data loaded into our application.
- As a result, the process of rendering pages happens mostly on the client-side.



Progressive Web Application (PWA)

- Progressive Web App, must be:
  - **Progressive** - Work for every user, regardless of browser choice, because they are built with progressive enhancement as a core tenet.
  - **Responsive** - Fit any form factor, desktop, mobile, tablet, or whatever is next.
  - **Connectivity independent** - Enhanced with service workers to work offline or on low quality networks.
  - **App-like** - Use the app-shell model to provide app-style navigation and interactions.
  - **Fresh** - Always up-to-date thanks to the service worker update process.
  - **Safe** - Served via HTTPS to prevent snooping and ensure content has not been tampered with.
  - **Discoverable** - Are identifiable as “applications” thanks to W3C manifests and service worker registration scope allowing search engines to find them.
  - **Re-engageable** - Make re-engagement easy through features like push notifications.
  - **Installable** - Allow users to “keep” apps they find most useful on their home screen without the hassle of an app store.
  - **Linkable** - Easily share via URL and not require complex installation.

Features & its Evolution

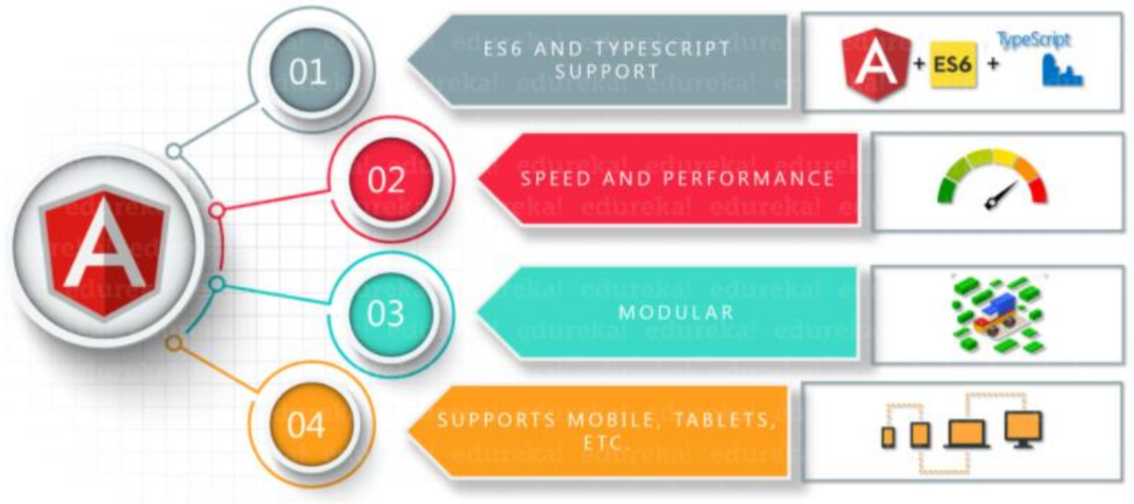
Angular Versions

- Angular 1.x - http://angularjs.org
  - JavaScript based Framework uses MVC pattern
- Angular 2.x and above – http://angular.io
  - Web component based Framework
- Angular 4.x - uses angular-cli for the ease of developers

- Current Version – Angular 5.2.4

Features

- Cross Browser Compliant
- Typescript Support
- Web Components Support
- Better support for Mobile App Development
- Better performance
- Easier to learn
- Good IDE support
- Simplicity - [ ] or ( )



Setup and Configuration

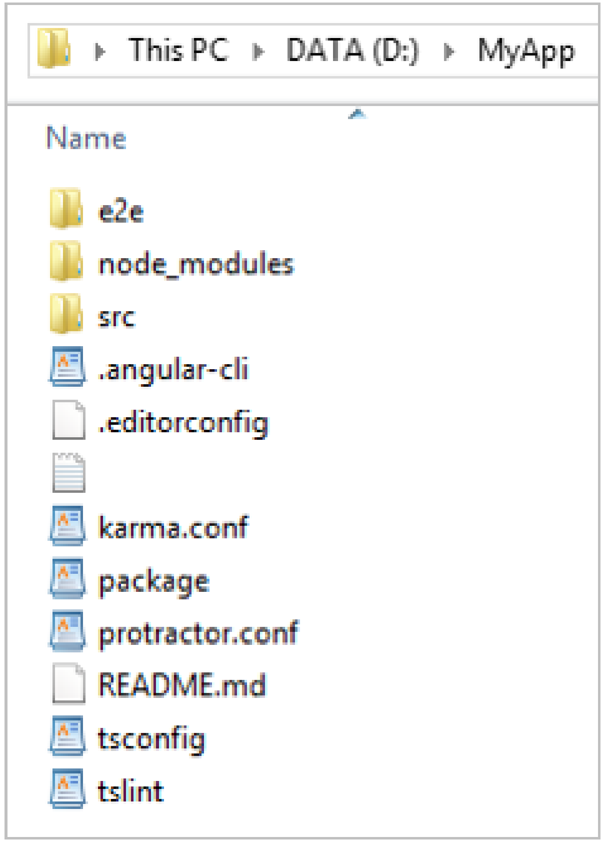
Angular Environment Setup

- Software Requirements:
  - Node.js (min version required 6.9.x) - https://nodejs.org/en/
  - Angular CLI - npm install –g @angular/cli
  - Visual Studio Code (open source)

Command	Description
npm install –g @angular/cli	To Install angular CLI globally
ng new <projectname>	Create a new angular project
ng serve –open	Builds and opens in browser
ng generate [component/service/pipe/directive/interface/module] <name>	Create new component/service/pipe/directive/interface/module
ng build	Build the application

Create Angular Application(1/3)

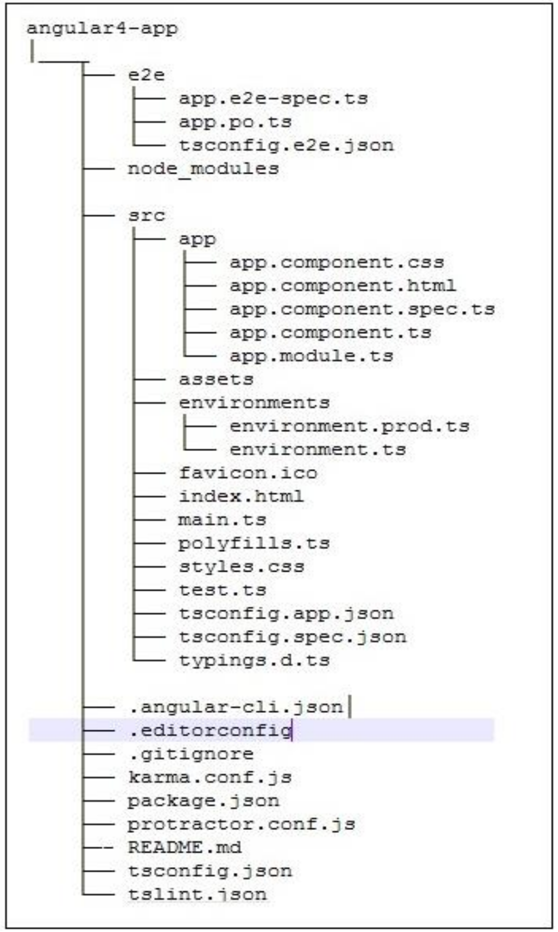
- **ng new <projectname>**
  - **e2e** – end to end test folder. Mainly e2e is used for integration testing and helps ensure the application works fine.
  - **node_modules** – The npm package installed is node_modules. You can open the folder and see the packages available.
  - **src** – This folder is where we will work on the project using Angular 4.
  - **.angular-cli.json** – It basically holds the project name, version of cli, etc.
  - **.editorconfig** – This is the config file for the editor.
  - **.gitignore** – A .gitignore file should be committed into the repository, in order to share the ignore rules with any other users that clone the repository.
  - **karma.conf.js** – This is used for unit testing via the protractor. All the information required for the project is provided in karma.conf.js file.
  - **package.json** – The package.json file tells which libraries will be installed into node_modules when you run npm install.



Create Angular Application(2/3)

- **protractor.conf.js** – This is the testing configuration required for the application.
- **tsconfig.json** – This basically contains the compiler options required during compilation.
- **tslint.json** – This is the config file with rules to be considered while compiling.

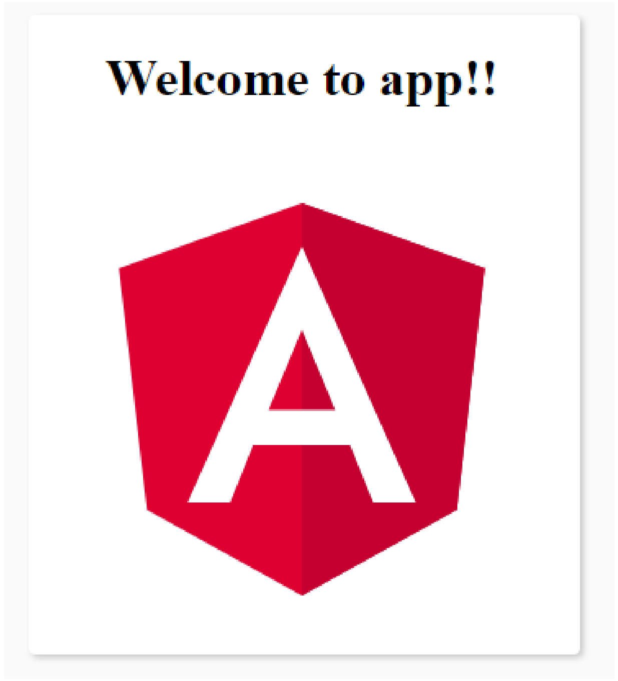
src folder is the main folder, which internally has a different file structure.



```
angular4-app
|
└───── e2e
│      ├── app.e2e-spec.ts
│      ├── app.po.ts
│      └── tsconfig.e2e.json
├───── node_modules
│
├───── src
│      ├── app
│      │   ├── app.component.css
│      │   ├── app.component.html
│      │   ├── app.component.spec.ts
│      │   ├── app.component.ts
│      │   └── app.module.ts
│      ├── assets
│      ├── environments
│      │   ├── environment.prod.ts
│      │   └── environment.ts
│      ├── favicon.ico
│      ├── index.html
│      ├── main.ts
│      ├── polyfills.ts
│      ├── styles.css
│      ├── test.ts
│      ├── tsconfig.app.json
│      ├── tsconfig.spec.json
│      └── typings.d.ts
│
├───── .angular-cli.json
├───── .editorconfig
├───── .gitignore
├───── karma.conf.js
├───── package.json
├───── protractor.conf.js
├───── README.md
├───── tsconfig.json
└───── tslint.json
```

Create Angular Application(3/3)

- Build the application using “ng serve –open” using node command prompt, it automatically builds and runs the application in web browser.



Application runs on local host with default port number 4200

http://localhost:4200/

Components and Modules

Education, Training and Assessment

Components in Angular(1/2)

- Components are basically classes that interact with the .html file of the component, which gets displayed on the browser
- **@Component** decorator is used to define component
- The file structure has the app component:

To create a new component as login:

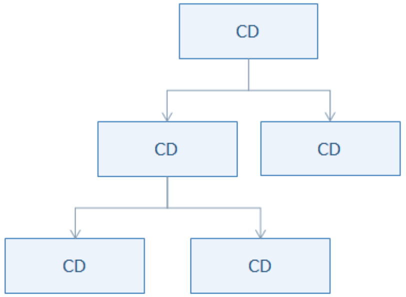
ng generate component login

app.component.css	- Style sheet of a component
app.component.html	- Html web page design view of a component
app.component.spec.ts	- Debug file used for unit testing
app.component.ts	- Class file to define modules, properties etc.,
app.module.ts	- To bind group the components, directives, pipes, and services, which are related to the application.

Refer to official site for Demos: http://angular.io

Components in Angular(2/2)

- Each component has its own Change Detector to notify the changes in components and update the application accordingly.
- Change Detection is a process in Angular which keeps views in sync with the models.
- **Zones** notifies Angular about the change detection.



CD: Change Detector

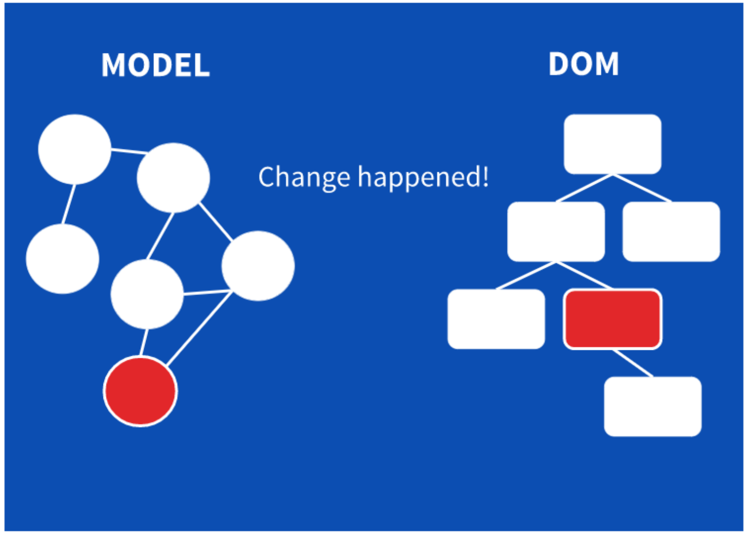
app.component.ts

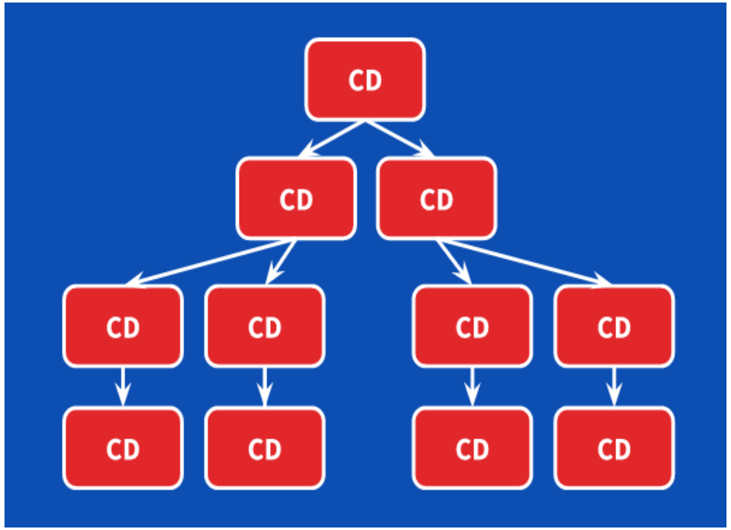
```
import { Component, OnInit } from '@angular/core'; // here
angular/core is imported .

@Component({
  // this is a declarator which starts with @ sign. The component
word marked in bold needs to be the same.
  selector: 'app-new-cmp',
  templateUrl: './new-cmp.component.html',
  // reference to the html file created in the new component.
  styleUrls: ['./new-cmp.component.css'] // reference to the
style file.
})

export class NewCmpComponent implements OnInit {
  constructor() { }
  ngOnInit() {}
}
```

Refer to official site for Demos: http://angular.io

Work of Change Detection in Components

- The basic task of change detection is to take the internal state of a program and make it somehow visible to the user interface.
- In Angular, **each component has its own change detector.**





Modules

- Modules are group the components, directives, pipes, and services, which are related to the application
- **@NgModule** decorator is used to define the modules

app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

Refer to official site for Demos: http://angular.io

Templates

Infosys | Education, Training and Assessment

Templates

- Templates separates view layer from the rest of the framework. We can change the view layer without breaking the application
- We can define a template in two ways

app.component.ts

Inline Template

```
import { Component } from '@angular/core';
@Component({
 selector: 'app-root',
 template: `
     <h1> Welcome </h1>
     <h2> Course Name: {{ courseName }}</h2>
  `,
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 courseName: string = "Angular";
}
```

External Template

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

@Component({
 selector: 'app-root',
 templateUrl:'./app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 courseName: string = "Angular";
}
```

Refer to official site for Demos: http://angular.io

Elements of Templates

- HTML
- Interpolation - **{{ }}**.
- Template expressions - {{ expression }}
- Template statements -(event) = statement

- **Example:**

```
<h1> Welcome </h1>
<h2> Course Name: {{ courseName }}</h2>
<p (click)="changeName()">Click here to change</p>
```

Directives

Infosys | Education, Training and Assessment

Directives (1/7)

- Directives are used to change the behavior of components or elements. We can use **directives in the form of HTML attributes**.
- We create directives using classes attached with @Directive decorator which adds metadata to the class.

  - **Components**

    Components are directives with a template or view.
    @Component decorator is actually @Directive with templates

  - **Structural Directives -** *directive-name = expression

    *ngIf

    *ngFor

    *ngSwitch

  - **Attribute Directives -** directives changes the appearance / behavior of a component / element

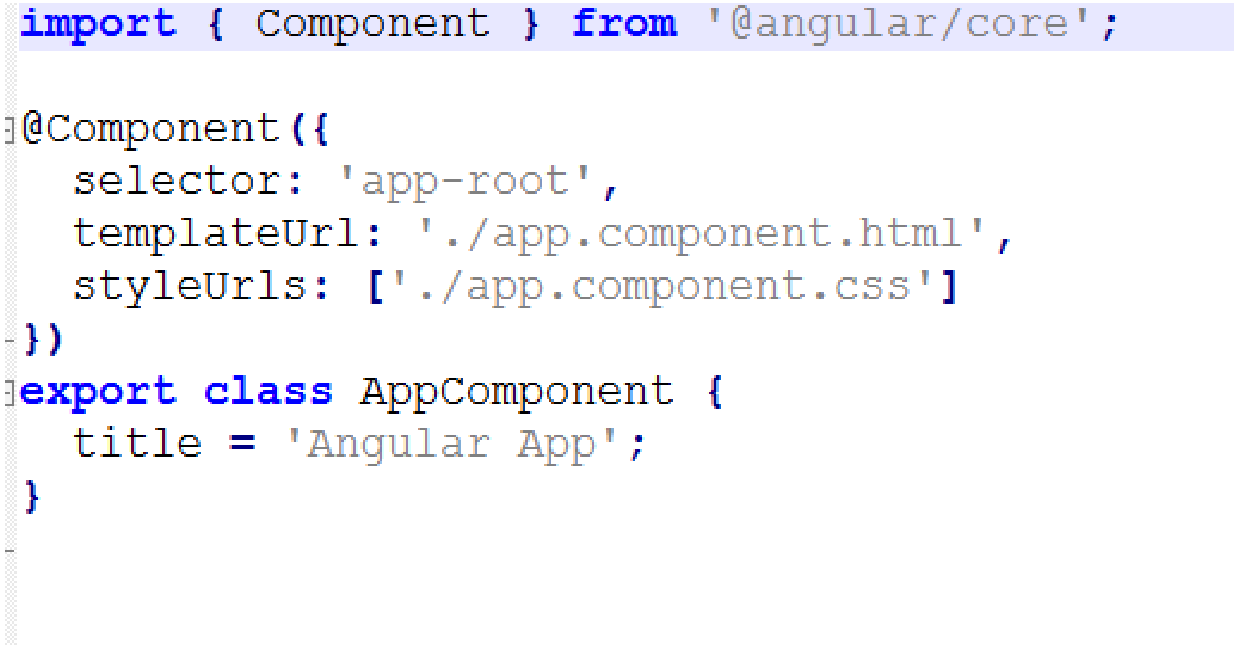
    ngStyle - [style.<cssproperty>] = “value”

    ngClass - [class.<css_class_name>] = “property/value”

Refer to official site for Demos: http://angular.io

Directives (2/7) - Component

app.component.ts



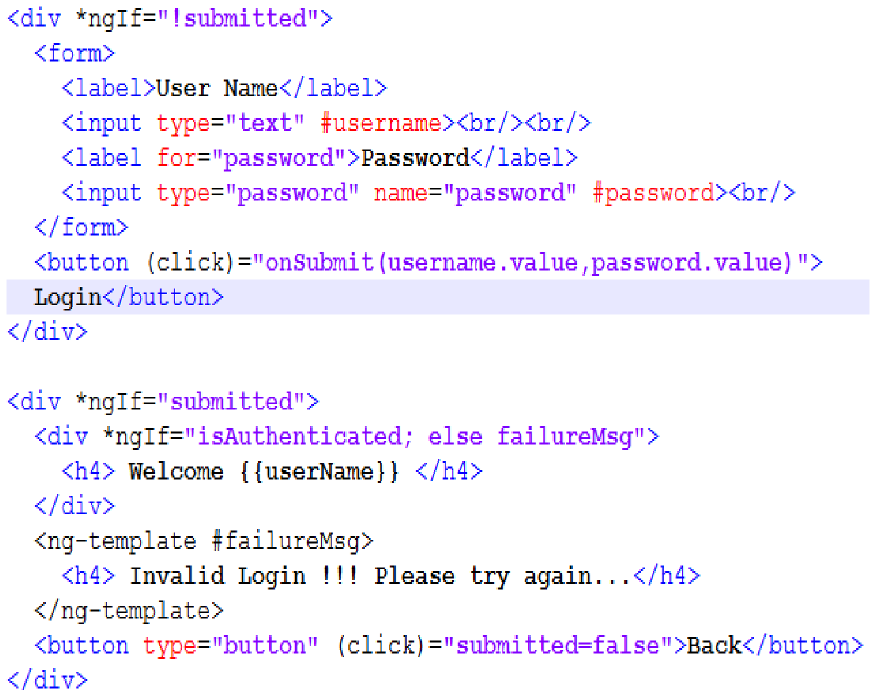
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular App';
}
```

Refer to official site for Demos: http://angular.io

Directives (3/7) – Structural

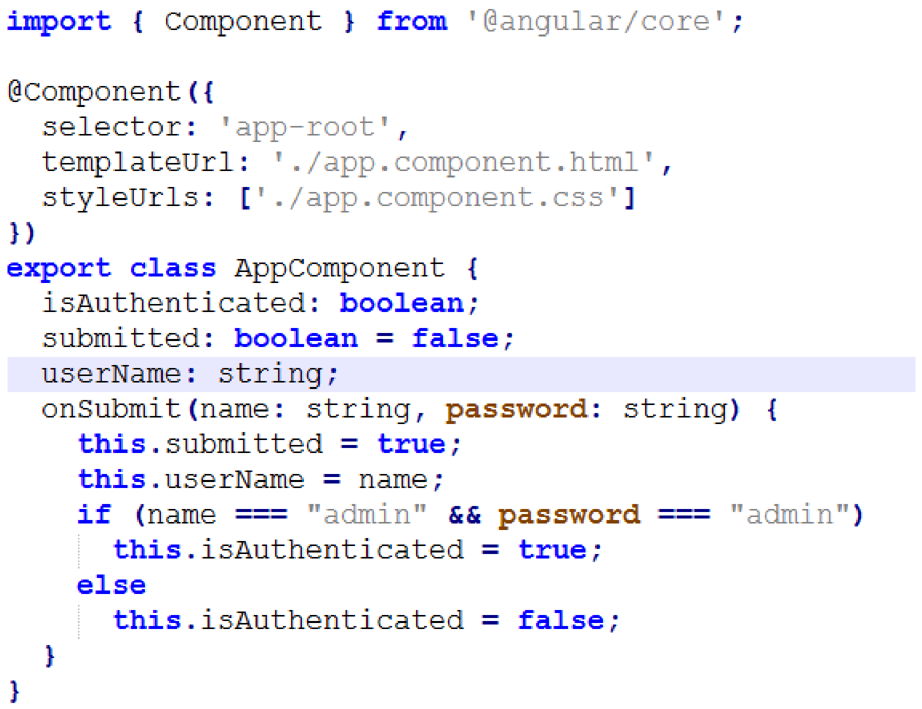
- *ngIf – add/remove elements in DOM tree based on criteria.

app.component.html



```html
<div *ngIf="!submitted">
  <form>
    <label>User Name</label>
    <input type="text" #username><br/><br/>
    <label for="password">Password</label>
    <input type="password" name="password" #password><br/>
  </form>
  <button (click)="onSubmit(username.value,password.value)">
  Login</button>
</div>

<div *ngIf="submitted">
  <div *ngIf="isAuthenticated; else failureMsg">
    <h4> Welcome {{userName}} </h4>
  </div>
  <ng-template #failureMsg>
    <h4> Invalid Login !!! Please try again...</h4>
  </ng-template>
  <button type="button" (click)="submitted=false">Back</button>
</div>
```

app.component.ts



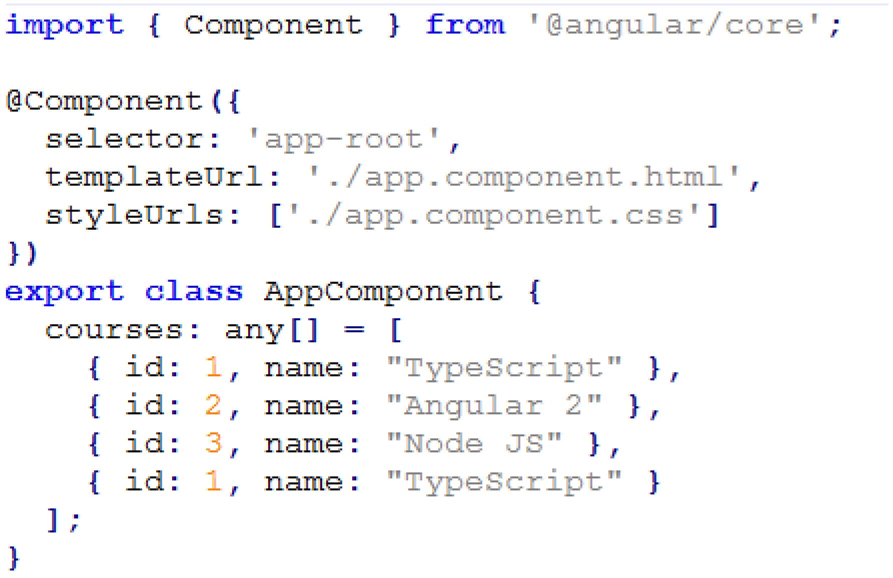
```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  isAuthenticated: boolean;
  submitted: boolean = false;
  userName: string;
  onSubmit(name: string, password: string) {
    this.submitted = true;
    this.userName = name;
    if (name === "admin" && password === "admin")
      this.isAuthenticated = true;
    else
      this.isAuthenticated = false;
  }
}
```

Refer to official site for Demos: http://angular.io

Directives (4/7) – Structural

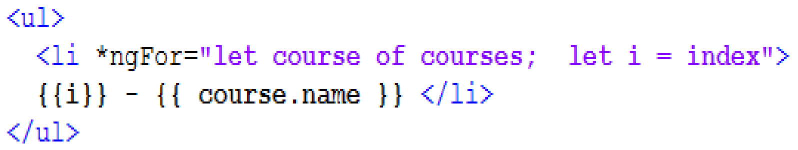
- *ngFor – use for iterating the elements.

app.component.ts



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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  courses: any[] = [
    { id: 1, name: "TypeScript" },
    { id: 2, name: "Angular 2" },
    { id: 3, name: "Node JS" },
    { id: 1, name: "TypeScript" }
  ];
}
```

app.component.html



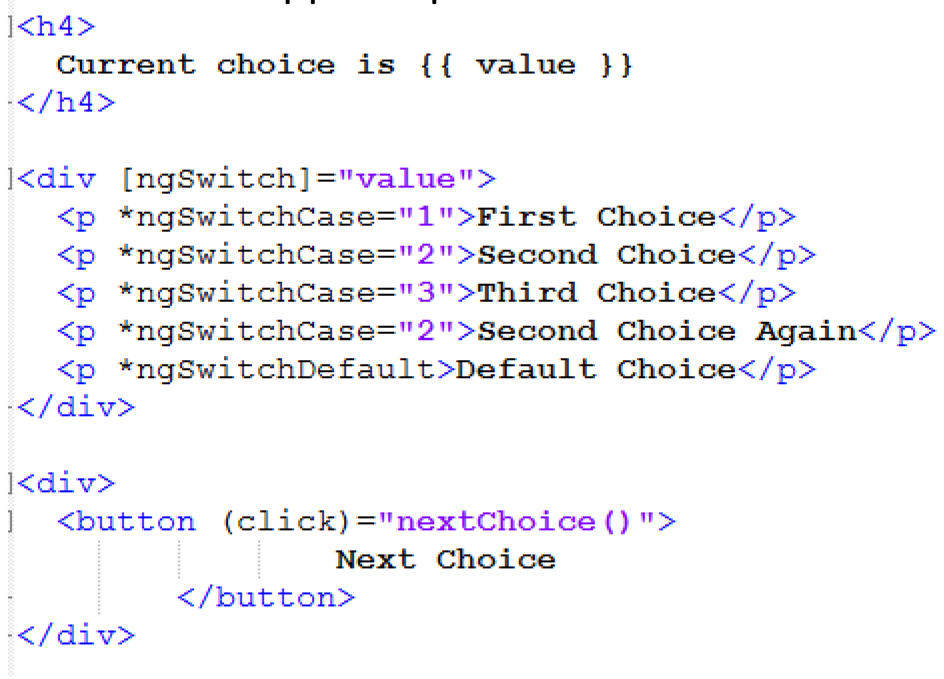
```
<ul>
  <li *ngFor="let course of courses;  let i = index">
  {{i}} - {{ course.name }} </li>
</ul>
```

Refer to official site for Demos: http://angular.io

Directives (5/7) – Structural

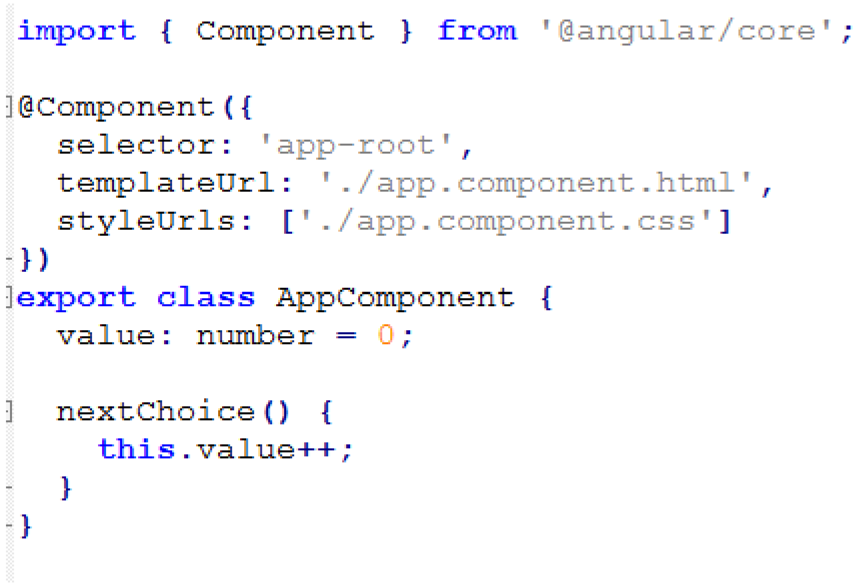
- *ngSwitch – used to check for matching cases.

app.component.html



```
<h4>
  Current choice is {{ value }}
</h4>

<div [ngSwitch]="value">
  <p *ngSwitchCase="1">First Choice</p>
  <p *ngSwitchCase="2">Second Choice</p>
  <p *ngSwitchCase="3">Third Choice</p>
  <p *ngSwitchCase="2">Second Choice Again</p>
  <p *ngSwitchDefault>Default Choice</p>
</div>

<div>
  <button (click)="nextChoice()">
                  Next Choice
            </button>
</div>
```

app.component.ts



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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  value: number = 0;

  nextChoice() {
    this.value++;
  }
}
```

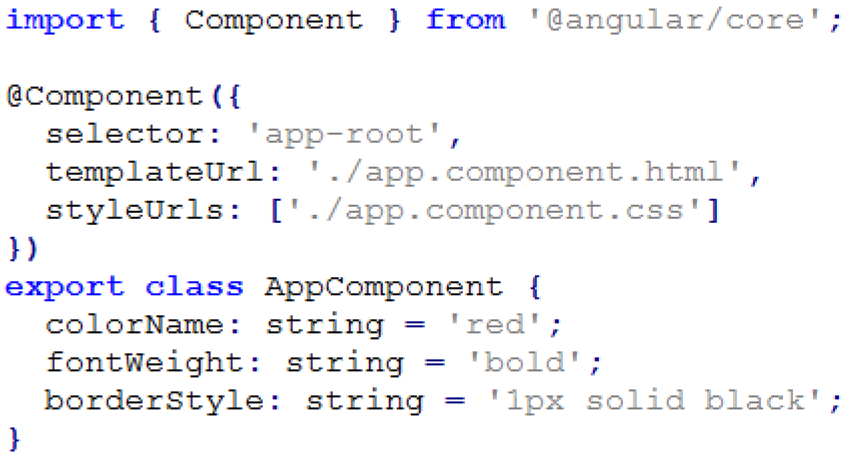
PUBLIC

Refer to official site for Demos: http://angular.io

Directives (6/7) – Attribute

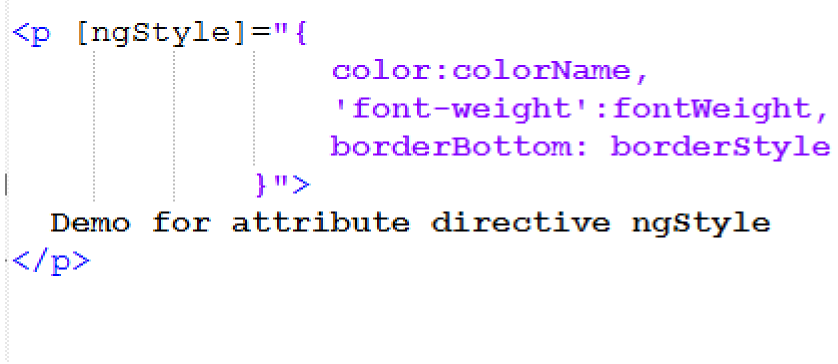
- ngStyle – apply styles to the elements.

app.component.ts



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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  colorName: string = 'red';
  fontWeight: string = 'bold';
  borderStyle: string = '1px solid black';
}
```

app.component.html



```
<p [ngStyle]="{
                color:colorName,
                'font-weight':fontWeight,
                borderBottom: borderStyle
            }">
  Demo for attribute directive ngStyle
</p>
```

Refer to official site for Demos: http://angular.io

Directives (7/7) – Attribute

- ngClass – used to enable/disable a class in an element

app.component.html

```
<div [ngClass]="{bordered: isBordered}">
  Border {{ isBordered ? "ON" : "OFF" }}
</div>
```

app.component.css

```
.bordered {
    border: 1px dashed black;
    background-color: #eee;
}
```

app.component.ts

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  isBordered: boolean = true;
}
```

Refer to official site for Demos: http://angular.io

Summary

- You are now knowledgeable on:
  - Needs, Features & its Evolution
  - Single Page Application (SPA)
  - Installation Setup and Configuration
  - Implement Components, Modules, Templates, Directives,
  - Concept of Change Detection in Components

Reference

- Links:
  - https://angular.io/tutorial
  - https://www.tutorialspoint.com/angular4/index.htm
  - https://coursetro.com/courses/12/Learn-Angular-4-from-Scratch
  - https://www.udemy.com/learn-angular-from-scratch/
  - https://programmingwithmosh.com/angular/angular-4-tutorial/
  - https://www.techiediaries.com/angular-4-tutorial/

- Videos:
  - Angular 4 Tutorial for Beginners: Learn Angular 4 from Scratch - https://www.youtube.com/watch?v=k5E2AVpwsko

Thank You

Infosys be more