Angular

Angular

1. Angular is basically is an open-source, JavaScript-based client-side framework that helps us to develop a web-based application.

2. A Framework for developing SPA (Single Page Application) based application.

3. An MVC based structure framework.

4. Provides us a facility to perform unit test so that our code can be tested before deployment.

Angular Advantages

1. Angular is a comprehensive solution for rapid front-end development. It does not need any other plugins or frameworks.

2. Faster Performance - Angular modifies the page DOM directly instead of adding inner HTML code.

3. Extended features such as dependency injection, routing, animations, view encapsulation, and more are available.

4. It is supported by IntelliJ IDEA and Visual Studio .NET IDEs.

TypeScript

1. TypeScript is a free and open-source programming language developed and maintained by Microsoft.
2. It is a strict syntactical superset of JavaScript.
3. Adds optional static typing to the language.
4. Provides much more privilege related to declarative programming like interfaces, classes, modules, static typings.
5. Typescript is basically acted as a transpiler. Transpiler means it basically converts one language to another language.

Basic Syntax:

```
var num : number = 5;
var list : Array<number> = [1,2,3];
function square(num : number) : number {
return num * num ;
}
```

Folder Structure

1. e2e – For an end to end testing purposes. Contains the configuration files related to performing the unit test.

2. node_modules - This folder contains the downloaded packages as per the configuration.

3. src - This folder contains the actual source code. It contains 3 subfolders:

   i. app - Contains the Angular project-related files like components, HTML files, etc.
   ii. assets - Contains any static files like images, stylesheets, custom javascript library files (if any required), etc.
   Iii. environments - Contains the environment-related files which are required during deployment or build.

4. tsconfig.json - Specifies the root files and the compiler options required to compile the project.

5. package.json - JSON file that contains all information related to the required packages for the project.

6. angular.json - Angular Application Environment based JSON file which contains all the information related to the project build and deployment.

Folder Structure – Contd.,

7. main.ts - Acts as the main entry point of our Angular application. This file is responsible for the bootstrapper operation of our Angular modules. It contains some important statements related to the modules and some initial setup configurations like

i. enableProdMode – Disable Angular's development mode and enable Productions mode.
   Disabling Development mode turns off assertions and other model-related checks within the framework.
ii. platformBrowserDynamic – This option is required to bootstrap the Angular app in the browser.
iii. AppModule – This option indicates which module acts as a root module in the applications.
iv. environment – This option stores the values of the different environment constants.

Angular Application Flow

1. angular.json -> properties and configuration of your Angular project.("main":"src/main.ts")
2. Main.ts -> platformBrowserDynamic().bootstrapModule(AppModule)
3. app.module.ts -> bootstrap: [AppComponent]
4. app.component.ts -> selector: 'app-root',
5. index.html -> <app-root></app-root>, templateUrl: './app.component.html'
6. app.component.html

Directives

1. Directive is a function that executes whenever the Angular compiler finds it in the DOM.
2. Angular directives are used to extend the power of the HTML by giving it new syntax.
3. Uses @Directive decorator in the class after importing the directives module.

There are 3 directives in Angular.

1. Component Directives.
2. Structural Directives.
3. Attribute Directives.

Component

1. Components are like the basic building block in an Angular application.
2. Components are defined using the @component decorator.
3. A component has a selector, template, style and other properties, using which it specifies the metadata required to process the component.

How to create a component?

1. Using CLI command, ng generate component comp_name (or) ng g c comp_name.
2. Creating 4 files(component.spec.ts, .html, .css, .ts) manually and declare it at @NgModule in app.module.ts.

Structural Directive

1. Structural directives are responsible for HTML layout.

2. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements.

3. There are three built-in structural directives.

   i. NgIf
   ii. NgFor
   iii. NgSwitch .

Structural Directives – Contd.,

Eg.,

1. ngIf:

```
<div *ngIf="a; else prompt">  //Consider a as condition
<p> a is true</p> // if a is true
<ng-template #prompt>
<p> a is false.</p>  // if a is false
<ng-template>
```

2. ngFor:

```
<tr *ngFor="let hero of heroes; let i = index;"
[ngStyle]="i%2=='0' ? {'color':'red'} : {'color': 'blue'}">
        <td>{{hero.name}}</td>
        <td>{{i}}</td>
</tr>
```

Structural Directive Contd.,

3. ngFor:

```
<container_element [ngSwitch]="switch_expression">
    <inner_element *ngSwitchCase="match_expresson_1">...</inner_element>
    <inner_element *ngSwitchCase="match_expresson_2">...</inner_element>
    <inner_element *ngSwitchCase="match_expresson_3">...</inner_element>
    <inner_element *ngSwitchDefault>...</element>
</container_element>
```

Eg.,

```
<select [(ngModel)]="selectedValue">
    <option *ngFor="let item of items;"[value]="item.name">{{item.name}}</option>
</select>
<div class='row' [ngSwitch]="selectedValue">
    <div *ngSwitchCase="'One'">One is Pressed</div>
    <div *ngSwitchCase="'Two'">Two is Selected</div>
    <div *ngSwitchDefault>Default Option</div></div>
```

Attribute directives

An Attribute directive changes the appearance or behavior of a DOM element.

1. NgStyle Directive: To modify the style of an HTML element using the expression. We can also use the ngStyle Directive to change the style of our HTML element dynamically.

Eg.,

NgStyle:

```
<tr *ngFor="let hero of heroes; let i = index; let even = even; let odd = odd"
[ngStyle]="i%2=='0' ? {'color':'red'} : {'color': 'blue'}">
    <td>{{hero.name}}</td>
    <td>{{i}}</td>
</tr>
```

Attribute Directive Contd.,

NgClass Directive: Used to add or remove CSS classes to an element.

Component.html:

```
<tr *ngFor="let item of items; let i = index; let even = even; let odd = odd
    [ngClass]={ odd: odd, even: even }">
    <td>{{hero.name}}</td>
    <td>{{i}}</td>
</tr>
```

Component.css:

```
.even{  color:red;}
.odd{  color:blue;}
```

Data Binding

1. Data binding is a communication between your typescript code of your component and your template which user sees.
2. Data binding can be either one-way data binding or two-way data binding.

One-way data binding:
In one-way data binding, the value of the Model is used in the View (HTML page) but we can't update Model from the View.
Eg., Angular Interpolation / String Interpolation, Property Binding, and Event Binding.

Two-way data binding:

In two-way data binding, Whenever you make changes in the Model, it will be reflected in the View and when you make changes in View, it will be reflected in Model.

In two way data binding, property binding and event binding are combined together.
Eg., [(ngModel)] = "[[(ngModel)] = "[property of your component]"

Types of Data Binding

Angular provides four types of data binding.

1. String Interpolation
2. Property Binding
3. Event Binding
4. Two-way Binding

String interpolation:

1. String Interpolation is a one-way data binding technique which is used to output the data from a typescript code to HTML template (view).

2. Uses the template expression in double curly braces to display the data from the component to the view.

Eg., Name: {{ user.name }}

Types of Data Binding - Contd.,

Property Binding:
1. Property Binding is also a one-way data binding technique.
2. Here we bind a property of a DOM element to a field which is a defined property in our component typescript code.

Eg., <input type="email" [value]="user.email">

Event Binding:
1. Used to handle the events raised from the DOM like button click, mouse move etc.
2. When the DOM event happens (eg. click, change, keyup), it calls the specified method in the component.

Eg., <button (click)="onClick()">Login</button>

Types of Data Binding - Contd.,

Two-way Binding:

1. In one-way data binding any change in the template (view) were not be reflected in the component TypeScript code. To resolve this problem, Angular provides two-way data binding.

2. The two-way binding has a feature to update data from component to view and vice-versa.

3. In two way data binding, property binding and event binding are combined together.

4. For two way data binding, we have to enable the ngModel directive.
   It depends upon FormsModule in angular/forms package, so we have to add FormsModule in imports[] array in the AppModule.

Eg., [(ngModel)] = "[property of your component]"

Pipes

1. Pipes are simple functions you can use in template expression to accept an input value and return a transformed value.

2. Pipes are useful because you can use them throughout your application, while only declaring each pipe once.

3. Angular provides built-in pipes for typical data transformations.

4. Commonly used built-in pipes for data formatting:

   DatePipe: Formats a date value according to locale rules.
   UpperCasePipe: Transforms text to all upper case.
   LowerCasePipe: Transforms text to all lower case.
   CurrencyPipe: Transforms a number to a currency string, formatted according to locale rules.
   DecimalPipe: Transforms a number into a string with a decimal point, formatted according to locale rules.
   PercentPipe: Transforms a number to a percentage string, formatted according to locale rules.

Pipes Contd.,

Eg.,

```
import { Component } from '@angular/core';
@Component({
selector: 'app-birthday',
template: `<p>The hero's birthday is {{ birthday | date }}</p>`
 })

export class BirthdayComponent {
birthday = new Date(1988, 3, 15); // April 15, 1988
}
```

Thank You