**What is Angular 2?**

* It is a**JavaScriptframework**for creating web and mobile applications.
* It supportsTypeScript, a super script of Javascript that helps in safe coding.
* It has several enhancements on top of Angular 1.x, which makes it simple to use and get the desired output. But knowledge in Angular 1.x is not necessary to learn Angular 2, since the whole framework is re-write.

**Why Angular 2?**

* **Easy**: Unlike previous versions, Angular 2 focuses only on building JavaScript classes; hence it is easier to learn.
* **Freedom**: It provides more choices of languages for consumption i.e. ES5, ES6, Typescript, and Dart. **TypeScript** is recommended here.
* **Flexible**: It is a Cross-platform framework.
* **Faster**: Because of server side rendering.
* **Simpler**: Component based approach with no controllers and $scope.
* **Performance**: Uses Hierarchical Dependency Injection system, which is a performance booster.

**Angular 2 Environment Setup**

To setup an Angular 2 environment, node.js is mandatory. Once node.js and npm are available, you can run the following to complete the setup in cmd.

* Install Angular CLI(command line interface) Command: **npm i -g @angular/cli** (-g installs angular globally for all users)
* Install TypeScript Command: **npm install -g typescript**

**Module**

* A Module is a way of organizing related Components, Services, Directives, and Pipes with a specific functionality.
* It is a block of code meant to do certain tasks.
* There can be several Modules within an app, but it should consist of at least one root module. Root Module, by convention, must be named: AppModule.
* It can be exported and imported in other modules.
* @NgModule is used to declare a Class as Module.
* Module as library - Angular Module can act as a library of modules. @angular/core is a most common library being used in building angular application. It has most of the modules that are required for your app.

**Module - Example**

This is a sample code for module.

import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

@NgModule({

imports: [BrowserModule],

declarations: [AppComponent],

bootstrap: [AppComponent]

})

export class AppModule { }

// AppModule Class is now defined as Module : @NgModule

* Trick to identify Root Module is that it **imports "BrowserModule"**. Other modules of that same application imports "CommonModule".
* imports:[...] - define array of modules required to build application here.
* declarations:[...] - define components, directives and pipes for this module here.
* bootstrap:[...] - define root component of this module here.

**What is Component?**

* A component is the basic block of angular 2 app.
* It handles the view part of the app and has the core logic of the page.
* It can render the view by itself based on the dependency Injection.
* There can be only one component per DOM element (i.e. only one selector element).
* Element's behavior can be manipulated by the properties associated with corresponding component.
* It can be identified using the decorator @Component.
* Components can be created, updated or destroyed based on the user actions on the page.

##### Components - Example

This example will render Hello, World!in the browser. Angular will search for <my-app> element in HTML file and replace it with the template content.

File: app.component.ts

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

@Component({

selector: 'my-app',

template: '<p>Hello, World!</p>',

})

export class AppComponent{}

File: index.html

............

<body>

<my-app> Loading... </my-app>

</body>

............

**What is Template?**

Template is a simple HTML code that creates view for a component, which you can dynamically manipulate.

There are two ways to define templates:

* template
* templateUrl

##### How to use Template?

**When template is used, it defines code in the same file**

import {Component} from 'angular2/core';

@Component({

selector: 'greet',

template: `

//this defines template for 'greet' component

<h1>Hello There!!!</h1>

<h2>Have a nice day.</h2>

` // back tick symbol (~ key on your keyboard)

//back tick is explained in detail in TypeScript

})

export class GreetApp {}

**When templateUrl is used, the code is defined in different files and URL of the corresponding files are referred.**

import {Component} from 'angular2/core';

@Component({

selector: 'greet',

templateUrl: 'app.component.html'

//this defines "URL to the external file"

that defines template for 'greet' component

})

export class GreetApp {}

Data Binding

It is the process of automatic synchronization of view and business logic. It helps in connecting the Template (view - what user sees) with Component (back end data/source).

There are four ways you can bind a data to the DOM depending on the direction the data flows.

* Data flows into the view by **Interpolation** and **Property Binding**.
* Data flows outside the view into the class by **Event Binding**.
* Data flows both ways by **Two-Way Data Binding**.

##### Property Binding vs Interpolation

Both are same, as they make the data flow from "Component" to "Template". The only difference is the way they are defined or used.

#### Interpolation Demo

import { component } from `@angular2/core`;

@Component ({

selector: 'myApp',

template: `

<h1> {{title}} </h1> //Interpolation Binding

`

})

export class AppComponent {

title: "Hello Fresco !"

}

#### Property Binding Demo

import { component } from `@angular2/core`;

@Component ({

selector: 'myApp',

template: `

<h1 [innerHtml] = "title"></h1> //Property Binding

`

})

export class AppComponent {

title: "Hello Fresco !"

}

##### Two-Way Data Binding

**Property Binding** and **Event Binding**areOne Way Data Binding. In other words, data will flow either from Component to View**or** the other way, but not both.

**Two-way data binding** is a combination of both Property Binding and Event Binding allowing us to update data in both directions. It is defined using [( )] popularly known as banana brackets.

##### What are Directives?

Directive is a class with @Directive decorator. They make DOM elements dynamic, by changing their behavior.

Directive is of three types: Structural, Attribute and Component.

##### Structural Directive

They manipulate the DOM elements. i.e. Delete or update the elements. This **changes the structure of webpage**.

Some of the built-in structural directives, Angular 2 offers are **NgIf**, **NgFor** and **NgSwitch**.

**Example**: <div \*ngIf="employee">{{employee.name}}</div>

2

##### Attribute Directive

These directives add styling by manipulating the **Attributes** of DOM element. They **do not change the structure of webpage**.

NgStyle and NgClass are some of the built-in attribute directives, Angular 2 provides.

##### What is Service?

Services are functions that are designed to complete a specific task. If you want to bring external data from server into your app or perform some repetitive task, then services come handy.

**How to use Services?**

**How to use Services?**

* Create a file <serviceName>.services.ts.
* Import injectable from @angular/core.
* Create a class with the required function and use the decorator @injectable() to specify that it is a service.
* Then import the service in root component, which is discussed in Dependency Injection.

**Example:** This simple service will perform add operation whenever it is used.

import {Injectable} from 'angular2/core';

@Injectable()

export class MyService {

addFunction(a,b){ return a+b; }

}

**Dependency Injection**

Dependency Injection is a way to "Inject" the parameters or services (dependencies), on which the new Instance is "Dependent" for its creation. How the dependencies are created is not a point of concern for the instance consuming the dependency.

* Angular finds which services are required to the component, by looking at the "type" of component's constructor parameters.
* Angular then checks for the service in Injector, which makes a container of all services that have already been created.
* When Object 'X' needs Object 'Y' to run, then Y is dependency of X or we can say in simple terms X is dependent on Y.
* Here in Angular, dependecies are defined as **Services**, which will be injected into those Objects which asks for it.
* ***Note***: Services / Dependencies are always defined inside "constructor" of the class.
* Let us consider the following example:
* export class SmartPhone {
* constructor (bat: Battery, disp: Display){}
* }
* Here SmartPhone is dependent on Services, that supply power(Battery) and takes input and shows output(Display), for its creation. So SmartPhone dependencies are: Battery and Display.

**How to Inject Dependencies?**

The whole technique of Injecting Dependencies can be summed up as:

* Creating a Service
* Registering Service with Providers.
* Defining Service inside constructor of the class that needs the Service. (Injecting)

You might wonder, what is a provider?

Well, It is a meta-data that **generates instances of services that the injector injects into components and other services.**.

##### What is a Service?

**Service** is a class that has a specific purpose. Services are mainly used to provide data to components that ask for it.

**Components** are used only to provide view to user. The Service class is Decorated with @Injectable().

import { Injectable } from `@angular/core`

// @Injectable() defines class "EmpService" as a "Service Class"

@Injectable()

export class EmpService {

// do something

}

**Difference between @Inject and @Injectable**

**@Inject()**

* It is a parameter decorator
* It explicitly tells what are the dependencies of the class inside constructor parameters

export class SmartPhone {

constructor (@Inject(Battery) bat){}

}

The same functionality can be easily achieved using TypeScript.

export class SmartPhone {

constructor ( bat: Battery ){}

}

**@Injectable()**

* It is a class decorator
* This tells Angular that this class can be used with Dependency Injector. So whenever a component is dependent on this class, Injector can create the instance of this class.

@Injectable()

export class SmartPhone{}

##### Observables In Angular

* In case of HTTP request, Observable is a single value (and not sequence of values) called **HTTP Response Object**.
* To make use of Observables, you use RxJS( Reactive Extensions for JavaScript). RxJS is just an external Library to work with Observable.
* Few areas in Angular 2 where reactive programming (RxJS) and Observables are used: EventEmitter, HTTP, Forms.

**What are Routers?**

Router in Angular defines route to navigate from one view to another, as user performs task on the application.

\*\* Routers can be used to develop 'Single Page Application'\*\*

* Traditionally websites used to load the whole webpage, on cliking any of the links. This increases page load time, bandwidth consumption affecting user-experience
* Now you can just load the required view, instead of sending the request to server to send a whole new webpage
* This saves bandwidth, page load time and improves user-experience
* The good thing is, you can trace back the history of navigations, and bookmark paticular view. This is achieved using path, which is visible in the URL

**Routing and Navigation**

Let us now define routerLink and router-outlet in template.

* routerLink defines the "path" of the "anchor tag". (i.e), when a link is clicked, **which "path"** should be displayed in URL and **which "component"**is to be loaded.
* router-outlet defines the location on webpage ***where*** the component is to be loaded.

<nav>

<a routerLink= "/my-cources" > My Cources</a>

<a routerLink= "/explore"> Explore </a>

</nav>

<router-outlet></router-outlet>

**Parameterised Routing**

In Parameterised Routing, you have a variable in path (url), that accepts "parameter".

For example, if you click any category, say, Data Science in Explore, it will show the details of that category and the path can be ---.com/explore/data-science. So you can write routes to look like:

const routes: Routes = [

{ path: 'explore/data-science', component: DataScienceComponent },

{ path: 'explore/devops', component: DevopsComponent }

and so on.....

];

**But this could get messy**, if we have a lot of categories in Explore. A better approach can be:

const routes: Routes = [

{ path: 'explore/:category', component: ExploreDetailsComponent }

];

Short and Sweet, is it not?

* :category is a variable, as it starts with : (colon)
* :category is palceholder for PARAMETER that will have category name.
* You can have any number of variables in path, as long as they begin with : and have different name.

**Non-Parameterised Routing**

**Non-parameterised** Routing always has ***higher priority*** over "Parameterised" Routing. Let us consider the example:

const routes: Routes = [

{ path: 'explore/:category', component: ExploreDetailsComponent },

{ path: 'explore/settings', component: ExploreSettingsComponent }

];

* settings does not start with :, hence is non-parameterised.
* Even if ExploreSettingsComponent "path" matches with explore/:category, precedence is given to explore/settings (non-parameterised). So, **ExploreSettingsComponent** is loaded instead of **ExploreDetailsComponent**.

**Reading Parameterised Routing**

The ExploreDetailsComponent should first read the parameter, then load the details based on the :category given in the parameter.

* This is achieved using ActivatedRoute service.
* To use it, you first need to ***import it***. Then ***inject it into the constructor*** of ExploreDetailsComponent.
* It provides a params Observable which you can subscribe to receive the route parameters.

import {ActivatedRoute} from "@angular/router";

.

.

.

constructor(private route: ActivatedRoute) {

this.route.params.subscribe( value => console.log(value) );

}