TypeScript :

It is a superset of JavaScript. The TypeScript is JavaScript and also has some additional features

like static typing and class-based object-oriented programming,

automatic assignment of constructor parameters and assigned null values and so on.

Traceue Compiler : Traceur is a compiler which takes ECMAScript and compiles into regular Javascript that runs in your browser.

Shadow DOM is a part of the HTML spec which allows developers to encapsulate their HTML markup,

CSS styles and JavaScript.

AOT compilation stands for Ahead Of Time compilation,

in which the angular compiler, that compiles the angular components and templates to native JavaScript and HTML, during the build time.

\* What are the new features of Angular 4 over Angular 2?

➔ The new View Engine (for AOT) in Angular 4 which reduces the generated code from components up to 60%.

➔ Angular 4 we can use the "if-else" conditional , like in Angular 2 only have "if".

➔ Angular 4 requires a minimum of Typescript 2.1 or higher.

➔ Animations is removed from @angular/core so as to reduce the size of our code bundle.

But we can add animations by importing {BrowserAnimationsModule} from @angular/platform-browser/animations our NgModule.

➔ Renderer is replaced by Renderer 2 in '@angular/core'.

➔ In Angular 4 there no need of adding pattern for validation we can just define type="email" in our

HTML Input which is an Html Dom property and angular will take care of it.

What is lazy loading?

➔ Lazy loading modules speeds up our applications startup time.

➔ Lazy loading creates multiple bundles and loads them on demand at runtime.

If we had loaded all our components and templates into one big bundle,

it would lead to a large performance problem.

Angular 2 Module consist of Bootstrap Array, Export Array, Import Array.

\* Bootstrap array is used to inform Angular JS which components need to be loaded

\* Export array is used to export components, directives, and pipes which can then be

used in other Angular JS modules.

\* Import array can be used to import the functionality from other Angular JS modules.

Agular 2 Introduction:

\* Angular 2 is most popular framework for developing mobile and desktop apps.

\* Better performance.

\* Powerfull Templates.

\* Provide simple API,Lazy loading and easy to debug apps.

\* It comes with features like component, Directives, Forms, Pipes, HTTP Services, Dependency Injection etc

\* Change from Angular1 to Angular2

\* Support for ES6 : Its superSet of JavaScript , Angular 2 is completely written in Typescript and meets the ECMAScript 6 specification.

\* Components are new controller :

In Angular 1 we had Controllers. In Angular 2 Controllers are replaced with Components.

The controllers and view in Angular 1 is defined as follows

//View

<body ng-controller=’appController’>

<h1>vm.message<h1>

</body>

//Controller

angular.module(‘app’).controller(‘appController’,appcontroller){

message=’Hello Angular2’;

}

In angular 2 we are using Components. The simple component is shown below written using Typescript.

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

@Component({

selector: 'app',

template: '<h1>{{message}} </h1>'

})

export class AppComponent

{

message: string=’Hello Angular2’;

}

In angular 2 a component represents a UI element. You can have many such components in a single web page.

\* Directive

The angular1 had a lot of directives. The some of the most used directives are ng-repeat & ng-if

<ul>

<li ng-repeat =customer in vm.customers>

{{customer.name}}

</li>

</ul>

<div ng-if=”vm.isVIP”>

<h3> VIP Customer </h3>

</div>

The Angular 2 also has directives, but with a different syntax.It has a \* before the directive name indicating it as a structural directive

<ul>

<li \*ngFor =#customer of customers>

{{customer.name}}

</li>

</ul>

<div \*ngIf=”vm.isVIP”>

<h3> VIP Customer </h3>

</div>

\* Data Binding

The powerful angular data bindings stays the same, with the minor syntax changes.

Interpolation

//Angular 1

<h3> {{vm.customer.Name}}</h3>

//Angular 2

<h3> {{customer.Name}}</h3>

One way/ Property Binding

//Angular 1

<h3 ng-bind="customer.name"></h3>

//Angular 2

<h3 [innerText]="customer.name"></h3>

Event Binding

//Angular 1

<button ng-click=”vm.save()”>Save<button>

//Angular 2

<button (click)=”save()”>Save<button>

Ag1 use : ngClick directive to bind to the event

Ag2 use : U can directly bind with the DOM Event.

Two way Binding

//Angular 1

<input ng-model=”vm.customer.name”>

//Angular 2

<input [(ng-model)]=”customer.name”>

\* $scope are out

Ag1 use scope to glue view and controller.

use dirty checking to see if any changes occurred.

Ag2 use zone.js to detect changes.

\* Filters are renamed to pipes

// Angualr 1

<td>{{vn.customer.name | uppercase}}</td>

// Angular2

Angular 2 uses the same syntax but names them as pipes

<td>{{customer.name | uppercase}}</td>

\* Services

AG1 : The Angular 1 had Services, Factories , Providers, Constants and values, which used to create reusable code.

These are then injected into Controllers so that it can use it.

AG2 : The angular 2 all the above is merged into a Service.Class.

Example :

Angular 1 Controller:-

var app = angular.module("userApp", []);

app.controller("productController", function($scope) {

$scope.users = [{ name: "Anil Singh", Age:30, department :"IT" },{ name: "Aradhya Singh", Age:3, department :"MGMT" }];

});

For example as,

//Angular 1,

<div ng-repeat="user in users">

Name: {{user.name}}, Age : {{user.Age}}, Dept: {{user.Department}}

</div>

Angular 2 Components using TypeScript:-

Here the @Component annotation is used to add the metadata to the class.

import { Component } from 'angular2/core';

@Component({

selector: 'usersdata',

template: `<h3>{{users.name}}</h3>`

})

export class UsersComponent {

users = [{ name: "Anil Singh", Age:30, department :"IT" },{ name: "Aradhya Singh", Age:3, department :"MGMT" }];

}

//Angular2,

<div \*ngFor="let user of users">

Name: {{user.name}}, Age : {{user.Age}}, Dept: {{user.Department}}

</div>

Angular 2 Architecture diagram:

Module : Used to break appliaction in peices of logical code. To perform single task.

Components : Used to combine modules.

Templetes : Used to define views for angular appliaction.

Services :Set of code that are shared by different components of an appliaction.

Directives :

Data Binding

Dependency Injection

Outputs

Steps to create Angular project

\* npm install -g @angular/cli@latest / npm install -g @angular/cli

\* ng -version

\* ng new "projectName"

// ng new : new requires you to specify the folder name

and will create a folder and copies the files.

// ng init : will copy the files to the current folder.

\* ng serve

Files Description :

\* angular-cli.json : Configration file for angular cli.

\* karma.conf.js : Configration file for test runnner.

\* package.json : Is NPM configration file, that list the third party package.

\* tsconfi.json : is the typescript compiler configration file.

\* ng generate <type>

type = components , directive , route, pipe, service

\* ng serve :

ng serve does not write the output to the disk. Webpack development server manages all the information in memory and serves it to the browser.

\* ng build is similar to ng server, but outputs are written to the disk

the folder dist

Angular Bootstraping

\* Loading Angular core lib

\* Loading main entry point

\* Loading root module

\* loading Root compenent

\* loading template

Loading Angular core lib : by loading index.html file.

loading main entry point : angular-cli.json

loading root module : main.ts

loading root component : app.module.ts

\* Angular Component : Component is a class. which is decorated with @component.

The component conatins the data and user interaction logic that defines, view looks and behave.

\*Components used to split application in smaller parts.

Adding new Component

ng generate component [component-name]

Example

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

@Component({

selector: 'bank-app',

template: '<h1>Hello & Welcome to ABC Bank Ltd. </h1>'

})

export class BankComponent

{

}

The compenent have 4 part

\* Import statement

\* class

\* Template

\* Metadata

\* Import statement : import the dependencies required by third component.

\* Template : Html template that define our view.

\* MetaData : Provide additional information about component to the angular.

Angular uses this information to process the class.

like @component

\* Class : Class contains the property and method.

\* Important component metaData properties

Selector : "book-app"

Providers : for services

Directives : to extends functionality.

styles/styleUrls : css for component internaal/external

template/templateUrl : Html template that define our view.

internal/external.

Ex :

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

@Component({

selector: 'app-root',

template: `

<h1> {{title}} works </h1>

<p> a long inline template </p>

`,

styles: ['h1 { font-weight: bold; }']

})

export class AppComponent {

title = 'app';

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Data Binding :

Process of binding data from component to view and vice-versa

Angualr support four types of data binding.

\* Interpolation {{}}

\* Property Binding [] = ''

\* Event Binding click = 'getCount()'

\* Two way binding [()]

Interpolation (evaluates expressions)-

· <span>{{4+6}}</span> <!-- Result = 10 -->

· <span>{{user.name}}</span> <!-- Result = Rajesh Sharma -->

Property Binding –

· <span [property]="user"></span>

Event Binding –

· <span [click]="addUser(user)"></span>

Two Way Binding –

· <span [(property)]="user"></span>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Directive :

A directive is a class, which is decorator with @Directive.

That contains meta-data and logic to manipulate the DOM behaviour.

The directive helps us to manipulate the view.

They help you to extend HTML functionality.

Types of Directive

\* Component <app-root><app-root> (with component selector)

\* Structural like \*ngFor, \*ngIf,

\* Attribute : Attribute or style directive can change the apperance

or behaviour of an element.

Like : ngModel , ngClass

\*ngFor = "let item of items"

Angular 2 components vs directives

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Difference @Components Vs @Directive

1. @Component meta-data annotation is used to register the components.

@Directive meta-data annotation is used to register the directives.

2. The components are used to create UI widgets.

The directives are used to add behavior to existing DOM elements.

3. The components are used to split to application into smaller parts.

The directives are use to design a reusable components.

4. Only one component is used per DOM element.

More than one directive are used per DOM element.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Angular Pipe

Pipe : Same like filter in angular 1 takes data as input and transforms that data

to get desired output.

Supported Pipe in Ag2 :

DatePipe

UpperCasePipe

LowerCasePipe

CurrencyPipe

Percentpipe

Jsonpipe

Syntax

Expression | pipeOperator[:pipeArguments]

date\_expression | date[:format]

Example:

<p>short : {{toDate | date:'short'}} </p>

<p>fullDate : {{toDate | date:'fullDate'}} </p>

<p>mediumDate : {{toDate | date:'mediumDate'}} </p>

<p>Uppercase :{{msg | uppercase}} </p>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Passing data to a child/nested component

\* Use of @input Decorator (child to parent component)

In Angular, the Parent Component can communicate with the child component by setting its Property.

To do that the Child component must expose its properties to the parent component.

The Child Component does this by using the @Input decorator

In the Child Component

\* Import the @Input module from @angular/Core Library

\* Mark those property, which you need data from parent as

input property using @Input decorator

In the Parent Component

\*Bind the Child component property in the Parent Component when instantiating the Child

Ex :

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

@Component({

selector: 'app-child-compenent',

templateUrl: './child-compenent.component.html',

styleUrls: ['./child-compenent.component.css']

})

export class ChildCompenent{

private \_count :number =0;

@Input()

set count(count: number) {

this.\_count = count;

console.log(count);

}

get count(): number { return this.\_count; }

}

app.component.html

<h1>Welcome to {{title}}!</h1>

<button (click)="increment()">Increment</button>

<button (click)="decrement()">decrement</button>

<app-child-compenent [count]=Counter></app-child-compenent>

app.component.ts

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Component Interaction';

Counter = 5;

}

\* Detecting the Input changes

We looked at how to pass the data from parent to the child using @Input annotation and property binding.

Passing the data to child component is not sufficient, the child Component needs to know when the input changes so that it can act upon it.

\* Two ways of detecting when input changes in the child component in Angular

Using OnChanges LifeCycle Hook

Using Input Setter

\* Passing data to a parent component from child (Child to Parent Component)

Pass data to parent component

There are three ways in which parent component can interact with the child component

\* Parent Listens to Child Event

\* Parent uses Local Variable to access the child

\* Parent uses a @ViewChild to get reference to the child component

\* Parent Listens to Child Event

Ex.

In Child Component

import {Component,Input,Output,EventEmitter} from '@angular/core';

@Component({

selector: 'child-component',

template: `<h2>Child Component</h2>

<button (click)="increment()">Increment</button>

<button (click)="decrement()">decrement</button>

current count is {{ count }} `

})

export class ChildComponent {

@Input() count: number;

@Output() countChanged: EventEmitter<number> =new EventEmitter();

increment() {

this.count++;

this.countChanged.emit(this.count);

}

decrement() {

this.count--;

this.countChanged.emit(this.count);

}

}

In Parent Component

<child-component [count]=ClickCounter (countChanged)="countChangedHandler($event)"></child-component>

Counter = 5;

countChangedHandler(count: number) {

this.ClickCounter = count;

console.log(count);

}

\* Parent uses local variable to access the Child in Template (Template local variable)

In Child component

We have removed the input, output & eventemiitter.

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

@Component({

selector: 'child-component',

template: `<h2>Child Component</h2>

current count is {{ count }} `

})

export class ChildComponent {

count = 0;

increment() {

this.count++;

}

decrement() {

this.count--;

}

}

in parent Component

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

@Component({

selector: 'app-root',

template: `

<h1>{{title}}!</h1>

<p> current count is {{child.count}} </p>

<button (click)="child.increment()">Increment</button>

<button (click)="child.decrement()">decrement</button>

<child-component #child></child-component>` ,

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Parent interacts with child via local variable';

}

Note : We have created a local variable, #child, on the tag <child-component>.

The “child” is called template reference variable, which now represents the child component

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Angular Forms

are used to collect data from users.

2 way

\* Template driven

\* Model driven forms

\* Building blocks of angular forms

\* FormControl : Represent a single input fields

like : let firstname= new FormControl();

\* FormGroup : Is a group of FormControl instance.

Often forms have more than one field.

It is helpful to have a simple way to manage the Form controls together.

city : <input type="text" name="city" >

Street : <input type="text" name="street" >

PinCode : <input type="text" name="pincode" >

let address= new FormGroup({

street : new FormControl(""),

city : new FormControl(""),

pinCode : new FormControl("")

})

\* FormArray : FormArray are array of FormControl

useful in dynamic form

let address= new FormArray({

street : new FormControl(""),

city : new FormControl(""),

pinCode : new FormControl("")

})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SERVICE: Provide Service to the component or to the other Services.

Service is a piece of reusable code with a Focused Purpose.

A code that you will use it in many components across your application

\* creating Service

ng generate service ['service-name']

Example:

public getProducts(){

let products : Product[];

products = [

new Product(1,'Memory Card',500),

new Product(2,'Pen Drive',750),

new Product(4,'Hard Disk',98)

]

return products;

}

just invoke it from Component

export class ProductDetailsComponent{

products:Product[];

productService;

constructor() {

this.productService = new ProductService();

}

getProducts(){

this.products = this.productService.getProducts();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Dependency injection

Is a method by which, a new instance of the class is injected into the component or to other service.

\* Anglar DI consists of four part

Consumer : The component that need the dependency.

Dependency : The service that is being injected.

Provider : Maintains the list of dependencies.

Injector : Responsible to inject the instance.

Ex:

@Injectible()

export class MyLogger {

AddTolog(msg: any) //method name that accept any parameter

{

console.log(msg);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Angular HttpClient

import { HttpClient } from '@angular/common/http';

import { Observable} from 'rxjs/Rx';

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Angular 2 life-Cycle

Constructor

ngOnChange : Called when an input binding values changes.

ngOnInit : Called after thr first ngOnChange.

ngDoCheck : After every run of change detection.

ngAfterContentInit : After component content is initialized.

ngAfterContentChecked : after evert check of component content.

ngAfterViewInit : after component view is initalized.

ngAfterViewChecked : after evey check of components view.

ngonDestroy : just befor the componets is dectroy.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Angular HttpClient

import { HttpClient } from '@angular/common/http';

import { Observable} from 'rxjs/Rx';

\* Ex using Service

Angular 2 service is a class that encapsulates some methods (GET/POST/PUT)

and provides it result.

Fetures:

1. Services are singleton Object.

2. Services are capable to return data in the form of promises or observable.

3. Sevices class is decorated with Injectable decorator.

Differnces between observable vs Promises

Promise :

1. called only once.

2. Can return only a single value at a time.

3. Promise are not cancelable.

Observable :

1. Can handle multiple values over time.

2. Can return multiple values.

3. observable are cancelable.

Steps to create Angular service

1. Import the injectable member

\* import {Injectable} from '@angular/core'

2. Import the HttpModule,Http and Response member

\* import {HttpModule, Http, Response} from '@angular/http'

3. Add the @Injectable at class level

4.Export the class

export class userService{

constructor(private \_http: Http){}

}

Step to calling Angular Service in the Components

1.Creating Service

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

import { CommonModule } from '@angular/common';

import { HttpModule, Http, Response } from '@angular/http';

@Injectable()

export class UserService {

constructor(private \_http: Http) { }

getUsers(apiUrl) {

return this.\_http.get(apiUrl)

.map((data: Response) => data.json());

}

}

2. Creating Component

@Component({

selector: 'user',

templateUrl: './user.component.html',

styleUrls: ['./user.component.css'],

providers: [UserService]

})

export class UserComponent {

users = []; //USERS DECLARATIONS.

//FETCHING JSON DATA FROM REST APIS

userRestApiUrl: string = 'https://api.github.com/users/hadley/orgs';

constructor(private userService: UserService) { }

//GET USERS SERVICE ON PAGE LOAD.

ngOnInit() {

this.userService.getUsers(this.userRestApiUrl).subscribe(data => this.users = data);

}

}

3. import and declare in app.module.ts

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

import { RouterModule, Routes } from '@angular/router';

import { UniversalModule } from 'angular2-universal';

import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { HttpModule } from '@angular/http';

import { AppComponent } from './components/app/app.component';

import { UserComponent } from './components/user/user.component';

import { HeaderComponent } from './components/shared/header/header.component';

import { MenuComponent } from './components/menu/menu.component';

import { LoginComponent } from './components/login/login.component';

import { RegistrationComponent } from './components/registration/registration.component';

@NgModule({

bootstrap: [ AppComponent ],

declarations: [ UserComponent,..... ],

imports: [

UniversalModule, // MUST BE FIRST IMPORT. THIS AUTOMATICALLY IMPORTS BROWSERMODULE, HTTPMODULE, AND JSONPMODULE TOO.

RouterModule.forRoot([ //RouterModule.forRoot method in the module imports to configure the router.

{ path: '', redirectTo: 'user', pathMatch: 'full' },

{ path: 'user/:id', component: UserComponent }, //HERE ID IS A ROUTE PARAMETER.

{ path: 'login', component: LoginComponent },

{ path: 'registration', component: RegistrationComponent },

{ path: '\*\*', redirectTo: 'user' }

]),

FormsModule,

ReactiveFormsModule

]

})

export class AppModule {

}

\* Routing Concepts

used to map appliactions URLs to application components

There are 3 main components to configure routing

1. Routes : Used to discribe appliactions Routes.

RouterModule.forRoot is used to configure the route.

RouterModule.forRoot({

{path : '' , redirectTo: 'home', pathMatch: 'full'},

{path : 'home/:id' , compenent:LoginComponent}

})

2. Route imports : used to import appliaction Routes.

import { RouterModule, Routes } from '@angular/router';

3. Router Outlet: It's a placeholder component.

used to render the component to specific location.

<route-outlet> </route-outlet>

4. RouterLink : used to link application's routes

<router-link></router-link>

------------------------------------------