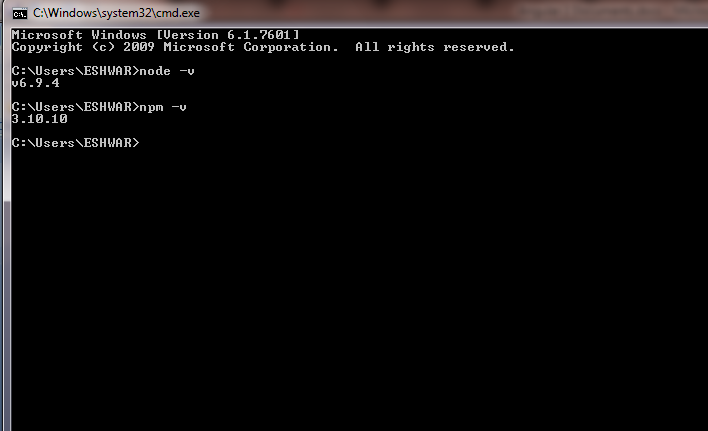
**Angular 2 Documents:-**

**Installation:-**

Node Js 6.9.4, NPM 3.10.10.

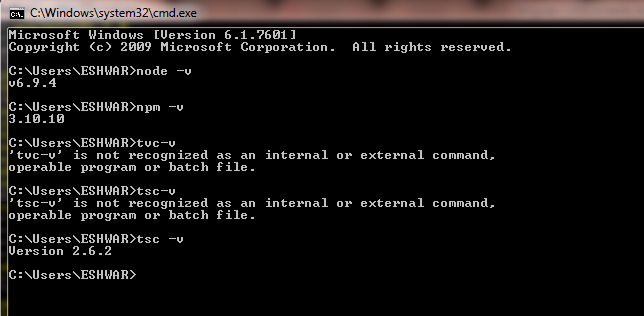


Installation of Typescript:-

Typescript version :- 2.6.2

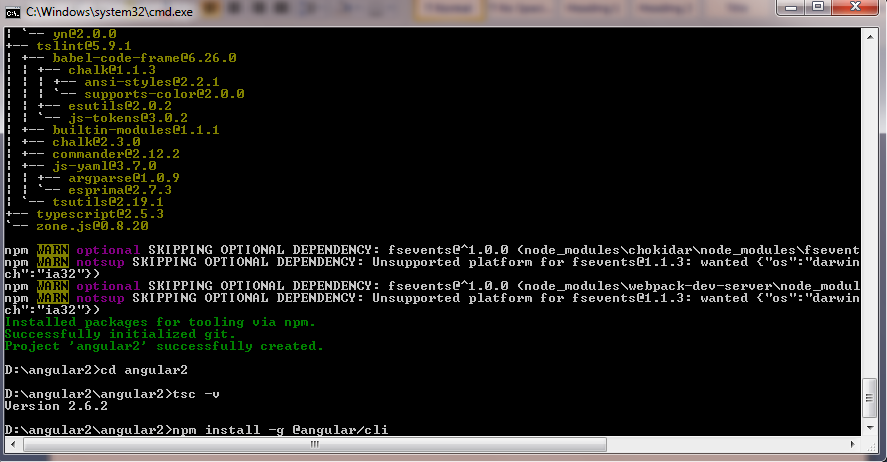
npm install -g typescript

tsc –v



Create new Directory in system for project and run command

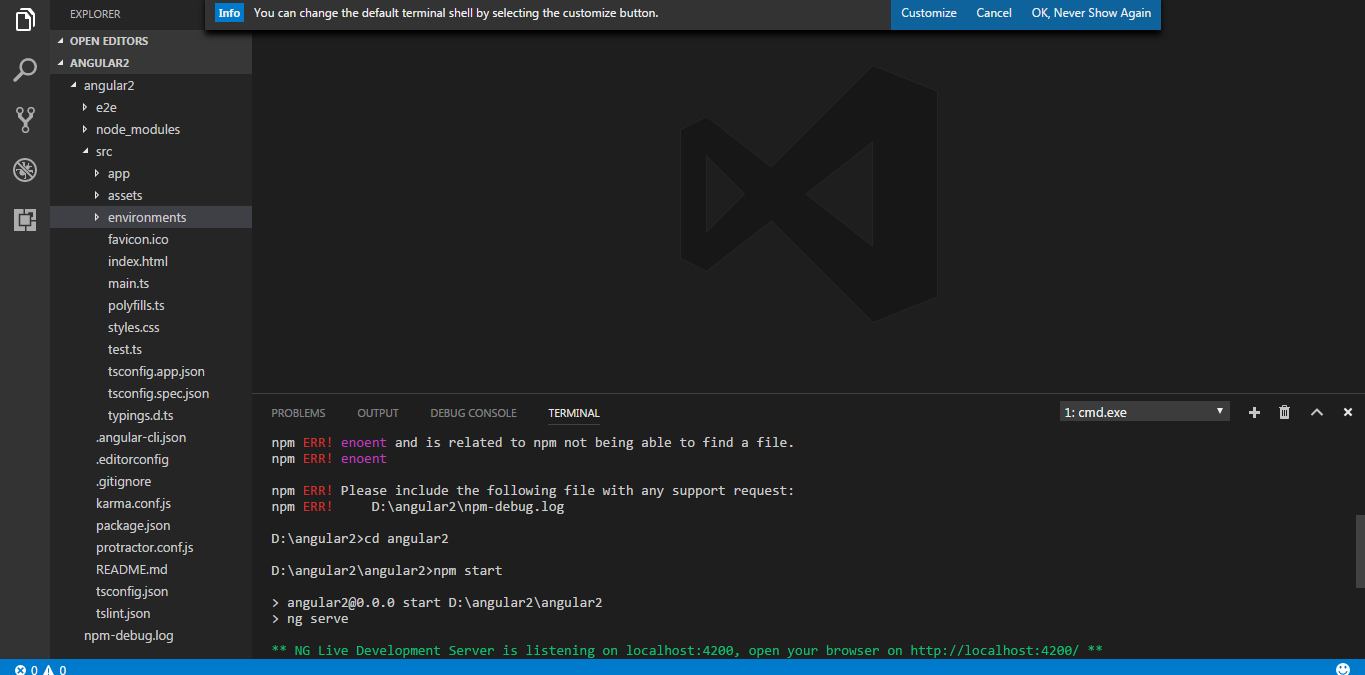
npm install -g @angular/cli



Install visual studio Code:-

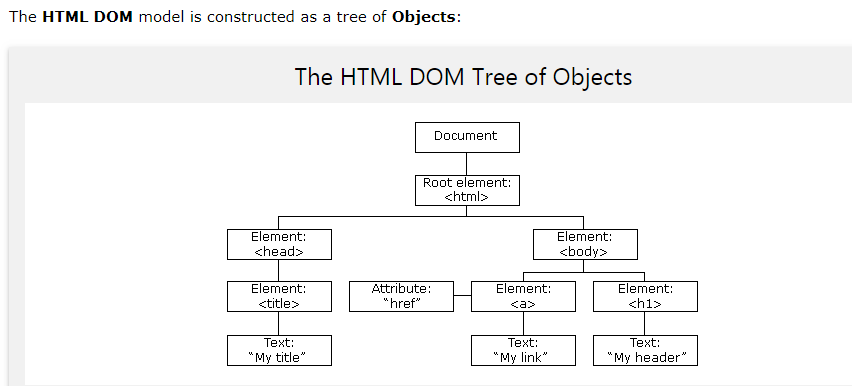
To start the local server,

Cmd:- npm start,



**Introduction**

DOM –Document object model:- Tree structure of object model.



* Angular js 1.x used –Javascript.
* Angular2 used – TypeScript.
* If you include Angular js then doesn’t need to include jQuery. JQlite concept included in angular instance of jQuery.
* Framework means all libraries included in on one library and divided structure of code.
* Angular 3.3.0 removed routing concept that’s why this version skip and introduces angular 4.
* API :- Application Program Interface

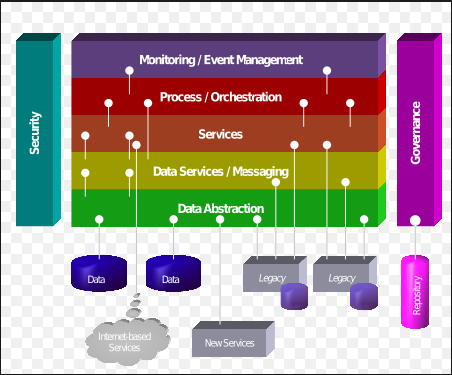
Request

Response 

Response

Front-End Back-End

* Javascript version ES5, ES2016, ES2017 is depending upon latest version Browser.
* SOA :- Service Oriented Architecture.



**Angular 2**

* $scope and controller concept removed from angular2.
* Angular2 to used Type script.

**TypeScript**

* TypeScript is developed by Microsoft.
* Type script is Superset of Javascript.
* Type script strongly data type depends upon data type.
* Type script is compiled and convert code in javascript.
* It's great Tooling.



**Variable Declaration in TypeScript:-**

* Var :- Scope of Var nearest function
* Let:- Scope of let nearest block and strongly type.

function variable(){

for(let i=0; i<5; i++){

console.log(i); Block

}

console.log("finally"+i);

}

variable();

**DataType Declaration in TypeScript:-**

**DataType:-**

* **Number**
* **String**
* **Boolean**
* **Array**
* **Enum**
* **Object**
* **Undefined**
* **Null**

let a = 20; //Number

let b = "eshwar"; //string

let c = true; //boolean

let d = {

"name": "eshwar joshi",

"address": "pune"

}; //object

let e =[1,2,3,4,5]; //arary

let text;

enum Color {red,green,blue} ; //assign to constant value

let backgroundColor = Color.blue;

let g = "";

let h = null;

**Interface:- This is additional in TypeScript, we can add interface in class,if we have used string number objects etc value there in one object so we have there Interface.**

interface Place {

x: number;

y: string;

}

function place(place:Place){

console.log(place.x);

}

let user = { x: 2, y: "User" };

let j = place(user);

interface Address{

permanent:string;

current:string;

}

enum Gender {male,female}; //enum also add outside of class

class Patient {

constructor(private nameOfPatient?:string,private address?:Address,

private mobile?:number,private age?:number ,private sex?:Gender

){

}

setx(nameOfPatient?:string){

this.nameOfPatient = nameOfPatient;

}

getx(){

return this.nameOfPatient;

}

display(){

console.log("Name" +this.nameOfPatient);

console.log("Address" +this.address.permanent);

console.log("Mobile" +this.mobile);

console.log("Age" +this.age);

console.log("Sex" +this.sex);

}

}

let obj = {"current":"Pune","permanent":"Chikhli"}

let patientDetails = new Patient( "",obj,8793248082,26,Gender.male);

patientDetails.display();

patientDetails.setx("arvind");

let newName = patientDetails.getx();

console.log(newName);

**Class:-**

* Finally, let’s extend the example one last time with classes. TypeScript supports new features in JavaScript, like support for class-based object-oriented programming.
* Here we’re going to create a EMP class with a constructor and a few public fields. Notice that classes and interfaces play well together, letting the programmer decide on the right level of abstraction.
* Also of note, the use of public on arguments to the constructor is a shorthand that allows us to automatically create properties with that name.

class Emp { //class is logical concept not physical place

name:string;

mobile:number;

emp(mobile){

console.log(mobile);

};

}

let object = new Emp; //Emp type of object object in physical place constructor

object.name = "eshwar";

let mobile = object.mobile = 8793248082;

object.emp(mobile);

**In Javascript**

var Emp = /\*\* @class \*/ (function () {

function Emp() {

}

Emp.prototype.emp = function (mobile) {

console.log(mobile);

};

;0

return Emp;

}());

var object = new Emp; //Emp type of object object in physical place

object.name = "eshwar";

var mobile = object.mobile = 8793248082;

object.emp(mobile);

* **Constructor:-** Initialization value to function, we can add Constructor in class, Constructor we can pass parameter.
* **This :-** this is pointing to currently object.
* **Access specify :-**  There is three part of specify

1. **Public :-** we can use anywhere in the code and inside class and outside class. This is default access specify
2. **Private:**- we can used inside the class.
3. **Protected:**- we can use same class, same package, child class.

* **?: :-** if we can assign variable in class and constructor with access specify then there is parameter validation. We cannot pass any another argument to class and constructor so there is optional parameter dependencies “ ?:”
* **Seter and geter method:-** set and get is method and x and y is function name.

1. **setx :-** in set we can pass value as argument to inner set call in class. We can get set value in class depend upon datatype. **Syntax:- setx x :- variable name**
2. **getx:-** in get return the value of set in Seter. So we can get private value in outside class. **Syntax :- getx x:-variable name**

* **Properties specify :-** In properties specify seter and geter but it will show outside of class as function return value not set function.

**Syntax:**- set x(){}; get x(){}

class Empl {

constructor(private x ?:number,private y?:number){

this.x = x;

this.y = y;

}

setx(x?:number){

this.x =x;

}

getx(){

return this.x;

}

}

let EmpObject = new Empl();

EmpObject.setx(5);

let t = EmpObject.getx();

console.log(t);

console.log(EmpObject);

class Point {

private \_x:number; // private acess specifir

private \_y:number; //variable class shown for naming variable

private z: object;

constructor(\_x:number,\_y:number){ //howw to make paramiter is optional add ?:

this.\_x = \_x;

this.\_y = \_y;

}

display(){

console.log("value x" +" " +this.\_x + " " + "value y" + " " +this.\_y);

}

set x(x:number){ // set the value of pravite variable private x:number;

this.\_x = x;

}

set y(y:number){

this.\_y = y; // properties acces specifer set Y

}

get x(){ // get value from privite variable private x:number;

return this.\_x;

}

get y(){

return this.\_y;

}

}

let pointObject = new Point(1,2); //optional paramiter

pointObject.x = 10; //call setx function pass value as argument

let x = pointObject.x;

pointObject.y = 20;

let y = pointObject.y;

console.log(pointObject);

console.log( "x:" +x + " " +"y:" + y ); // display value

**Module -> sub module.**

implements OnInit

**summery implement**

**softstring**

**custom pipe.**

**c→v pro**

**v→c event**

**Camal case assignment.**

**@input Decorator this is the properties make globel.**

**Child to parent component**

**Count increase and decrease**

**Camel case.**

**jeliken**

**agile development**

**dev ops concept**

**properties binding**

**event binding**

**two way data binding**

**pollution**

**\*ngFor= “let course of couseses; index as I” → syntax of ng for**

**let index =this.couse.indexOf(course),-→finding index**

**this.couses.push({id:2, name:java}) → add the new object.**

**this.couses.slice(**

**trackby**

**safe traversal operator**

{{data?.firstName}}

Template form and Reactive form.

1> Template form:- we can create form from view site validation

ngForm ngModel

HTTP services:-

The $http service is a function which takes a single argument — a [configuration object](https://docs.angularjs.org/api/ng/service/$http#usage) — that is used to generate an HTTP request and returns a [promise](https://docs.angularjs.org/api/ng/service/$q) that is resolved (request success) or rejected (request failure) with a [response](https://docs.angularjs.org/api/ng/service/$http#$http-returns) object.

Promises and observable:- Both are working same, both are asynchronous call.

In angular2 we are using observable.

|  |  |
| --- | --- |
| observable | promise |
| Subscribe() | then().catch() |
| lasy | eager |
| here operator like map to promise |  |

Error:- There are two type of error.

Exception and non exception.

Exception mean known error, its depend upon the status code like 400,401,500,etc

non exception:- internet not avilable, out of memory.

first add HttpModule in appComponent.ts file

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

import [HttpModule]

Methods:-

Post:- create new data/ save new data to backend,

pass the json with this method

posts:any[];

private url='https://jsonplaceholder.typicode.com/posts';

createPost(input:HTMLInputElement){

let post={title:input.value};

this.http.post(this.url,JSON.stringify(post)).

subscribe(res =>{

post['id'] = res.json().id;

this.posts.splice(0,0,post);

console.log(res);

});

}

Put:- modify and update the data to backend (Whole Object)

pass the json with this method

updatePosts(post){

this.http.put(this.url+'/'+post.id,JSON.stringify(post))

.subscribe(res=>{

console.log(res.json());

})

}

Patch:- modify and update the data to backend (single entites)

pass the json with this method

updatePosts(post){

this.http.patch(this.url+'/'+post.id,JSON.stringify({title:"Arun"}))

.subscribe(res=>{

console.log(res.json());

})

}

Get:- retrieve and get data from backend

getPosts(){

this.http.get(this.url)

.subscribe(res =>{

this.posts = res.json();

console.log(this.posts);

});

}

Delete:- Delete the data from backend

deletePosts(post){

this.http.delete(this.url+'/'+post.id).subscribe(res=>{

let index = this.posts.indexOf(post);

this.posts.splice(index,1);

console.log(res);

console.log(index);

})

}

https://youtu.be/dNTC\_MWuW58

Error Handling :-

there are two type of error handling.

Routing part of angular2

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

First import the routerModule in appModule.ts

RouterModule.forRoot([

{

path: '',

component:HomeComponent

},

{

path: 'followers/:username',

component:GitHubProfileComponent

},

{

path: 'followers',

component:GithubFollowersComponent

},

{

path: 'post',

component:OptimisticComponent

},

{

path: 'form',

component:ReactiveFormComponent

},

{

path: '\*\*',

component:NotFoundComponent

},

])

RouterModule we have to mention the forRoot to the checking the path of the the component

<router-outlet></router-outlet>

router oulet add in main html file app.component.ts file

<h5 class="mt-0"><a [routerLink]="['/followers',follower.id]" >{{follower.login}}</a></h5>

if we have to pass value to link the we have to write in property binding