1. Angularjs and Angular  
   ng-controller=’appController’ --> components  
     
   directives are ng-repeat & ng-if -> \*ngIf and \*ngFor  
     
   one way data binding  
   //AngularJS

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

//Angular

<h3 [innerText]=”customer.name” ></h3>  
  
ng-click --> (click)=””  
  
ng-model --> [(ngModel)]   
  
<body ng-app=’app’> </html> -> platformBrowserDynamic().bootstrapModule(AppModule);

1. Execution process in angular.

index.html, main.ts (entry files), appModule, appComponent.

1. In component template url take one, but style url we can give multiple style urls [].

Different types of components selectors.

Selector:’app-root’ <app-root></app-root>  
selector:`[app-root]` <div app-root></div>

selector:`.app-root` <div class=”app-root”></div>

1. Data binding

Communication b/w ts file and html file. {{}} ‘string interpolation’. If it is boleen it convert string.

‘property binding’ []. [disabled]= “allowuser”.

‘event binding’ (). (click)=” hello()”.

Two way data binding [(ngModel)]=”hello”.

1. Directives

Structural :- ‘\*’ means manipulate Dom

\*ngIf=”show”

<div \*ngIf=”show; else noshow”>hello</div>  
 <ng-template #noshow>hello else part<ng-template>

<ng-template [ngIf]=”noshow”>hello else part<ng-template> // background conversion.

\*ngFor=”let user of users ; trackBy:identify"

<ul [ngSwitch]=”value”>  
 <li \*ngSwithCase=”1”>1</li>

<li \*ngSwithCase=”2”>2</li>

<li \*ngSwithDefault> default value</li>  
 </ul>

Attribute :-

[ngStyle]=”{‘color’:’red’}”.  
 [ngStyle]="{'opacity': is\_mail\_sent ? '0.5' : '1' }"

[ngStyle]="{'background-color’: status === 'error' ? 'red' : 'blue' }"

[ngClass]=”’showclass’”.

[ngClass]="['cssClass1', 'cssClass2']"

[ngClass]="{'cssClass1': true, 'cssClass2': true}"

1. Pass data from one component to another component.

parent to child :- @Input() username:string; -> [username]=”user1”

child to parent :- @output() sendData :EventEmitter = new EventEmitter();

this.sendData.emit(‘data’);

(sendData)=”addevent($event)”

1. View Encapsulation

a new attribute added \_ngcontent-rsm-c42 unique identifier.  
encapsulation: viewEncapsulation. Emulated.  
  
emulated :- (default)   
none :- none it will add all the tags same and remove dynamic attributes (unique identifier).

shadow Dom :- same as (emulated) . it not supported in modern browsers.

1. Local Reference in angular   
     
   <p>

<label **for**="lastName">Last Name</label>

<input (keyup)="0" type="text" #lastName id="lastName">

</p>

<b>Welcome {{lastName.value}} </b>

1. Access html element in dom using @ViewChild   
     
   we can access child component values and methods also. Using viewchild we can use component also.

<input type=”text” value=’’ #inp />

@ViewChild(inputdata) inp : Elementref;

this.inputdata.nativeElement.value; -> (elementref);  
  
The static option determines the timing of the ViewChild query resolution.

static:true will resolve ViewChild before any change detection is run.

static:false will resolves it after every change detection run.

1. Ng-content and @contentChild

When we want share dynamic html content b/w component tags. To child component

<app-user>

<div> user angular</div>  
<app-user>

Child component we need to use.

<div>

<ng-content></ng-content>

</div>

parent -> <header><h1>Angular</h1></header>

(or)

<div class="header">

(or)

<ng-container ngProjectAs="header">

<div>

<h1 style="color: red;">React</h1>

</div>

</ng-container>

child -> <ng-content select="header" ></ng-content>

(or)

<ng-content select=".header" ></ng-content>  
   
@contentchild  
 in Ng-conent placed component. We will get content projection data in contentinit life cycle.

1. ng-container   
     
    it will not appear tag in dom it eliminated . in a tag we need to use one structural directive or no need. When no need to add additional tag that time we can use. Only ng-container we use content it will appear.

<ng-container \*ngIf="true">

Container's content.

</ng-container>

1. <ng-template> ->  
    when we write any contact in this tag it will not show. If need to show this tag data we need to connect with any structural directive. To use structural directives, we need to use ‘[]’. it will not appear tag in dom it eliminated.

<h1>ng-template & TemplateRef</h1>

<h2>Using the ngTemplateOutlet</h2>

<ng-template #sayHelloTemplate>

<p> Say Hello</p>

</ng-template>

<ng-container \*ngTemplateOutlet="sayHelloTemplate">

This text is not displayed

</ng-container>

//Output

ng-template & TemplateRef

Using the ngTemplateOutlet

Say Hello

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

<ng-template [ngIf]="selected">

<div>

<p>You are selected</p>

</div>

</ng-template>

1. How to Use ngTemplateOutlet in Angular   
     
   ngTemplateOutlet is a directive. It instantiates a template dynamically using a template reference and context object as parameters. In this guide, we will learn how to use it in Angular. We will show you several ngTemplateOutlet examples to learn from.

1. Component life cycle methods.

ngOnChanges -> @Input changes. Simple changes (current value,previousvalue,firstchange). For array and object when reference changes it will called.

ngOnInit -> component initialize. Called once

ngDocheck -> every time call when any value changes. Change detection occur. (KeyValueDiffers) KeyValueDiffer, IterableDiffer

**this**.differ = **this**.differs.find(**this**.customer).create(**null**);

const customerChanges = **this**.differ.diff(**this**.customer);

ngAfterContentInit -> content projection. Ng-conent placed component

ngAfterContentChecked -> content projection changes occur in template.

ngAfterViewInit -> component view and child view initialize.

ngAfterViewChecked -> component view and child view changes occur.

ngonDestory -> component destroy.

1. Custom directives   
     
   @directive({  
    selector: ‘[text]’  
   })

ElementRef.style.backgroundColor = ‘red’;

<div text>hello</div>  
  
renderer2 -> avoid xss attack.

this.render.setStyle(elemet,style,value);

sending data to directive.

[text]=”’blue’”

@input(‘text’) defaultColor:string= ’red’;

1. HostListener

@HostListener(‘mouseenter’) onmouseover(event){  
 this.render.setStyle(elemet,style,value);

}

1. HostBinding

@HostBinding(‘style.color’) color:string;

@HostListener(‘mouseenter’) onmouseover(event){  
 this.color = ‘red’;  
}

1. Angular services

Taking service to the component. If we have common logic, we can use services.

When we use service in providers array it create instance and we need to inject service in constructor.

Hierarchy injection in app module it creates same instance for all components.

Injecting service in another service using @injectable. For components and directives we no need to use @injectable.

Cross component communication :-  
 statusUpdated = new EventEmmiter();

This. service.. statusUpdated.emit(‘abc’);

This. service.statusUpdated.subscribe((res)=>{console.log(res)});

1. Angular Routing

Const approutes:Routes = [

{path:’’,redirectUrl:’/home’,pathmatches:’full’ }  
 {path:’home’,component:HomeComponent},

{path:about,component:AboutComponent},

{path:’\*\*’,component: pagenotfoundcomponent} -> wildcard route.  
]

Routermodule.forRoot(approutes);

<router-outlet></router-outlet> -> directive

<a [routerLink]=”[‘/home’, ‘1’]” [routerLinkActiveOption]=”{exact:true}”>

<a routerLink=‘/home’ routerLinkActive=”abc”>

Exact -> which matches correctly.  
  
form ts file navigation:-

Need to import ‘Router’.

This.navigate([‘home’, ‘1’]); -> array type of data. Use for another type of segment.

This.navigateByUrl(‘/home/1’); -> we need to use string.

ActivatedRoute:- to fetch parameters and fragments.  
  
activatedroute.snapshot.params[‘id’]; -> it call when component load once.  
  
activatedroute.params.subscribe((res)=> console.log(res[‘id’]);) -> for dynamic get params.

Pass Query Parameter and Fragments :-  
  
“/users?page=1#load”

[routerLink] = “[‘/users’]” [queryParams]=” {page:1}”

[routerLink] = “[‘/users’]” [fragment]=”’load’”

This.navigate([‘home’, ‘1’], {

queryParams:{page:1},

fragment:’load’

});

activatedroute.snapshot.queryParams[‘page’];

activatedroute.snapshot.fragment;  
  
activatedroute.queryParams.subscribe((res)=> console.log(res[‘page’]);)

activatedroute.fragment.subscribe((res)=> console.log(res[‘page’]););

preserve or merge query params:-

This.navigate([‘home’, ‘1’], {

queryParamsHandling: ‘merge’

});

preserve :- it preserve query params.

merge :- what ever query params coming it merge to existing one.   
  
Nested Routing or child routing:-

Const approutes:Routes = [

{path:’’, redirectUrl:’/home’,pathmatches:’full’ }  
 {path:’home’,component:HomeComponent},

{path:about,component:AboutComponent,

children:[

{path:’:id’,component: AboutsingleComponent}

]}  
]

Routermodule.forRoot(approutes);

Routing Guards:-

canDeactivate :- calling method in that route if true it navigate otherwise stay there. Component, ActivatedRouteSnapshot , RouterStateSnapshot, RouterStateSnapShot.

canActivateChild :- to protect child routes. ActivatedRouteSnapshot , RouterStateSnapshot.

canActivate :- ActivatedRouteSnapshot , RouterStateSnapshot.

canLoad() :-

Reslove :- dynamic data getting before entering the route. ActivatedRouteSnapshot , RouterStateSnapshot.

Passing Static Data to the Route and access the static data :-

{path:’home’,component:HomeComponent, data:{page:1}}

This.activateRoute.data.subscribe((data)=> console.log(data[‘page’]));  
  
Hash URLs as Fragments :-

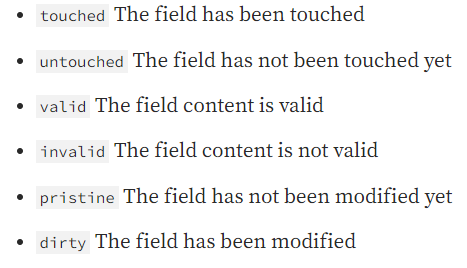
1. Form Handling

Template Driven forms:-

@viewChild(‘contant’) contantform : NgForm; -> on field when we change data we can get data.  
 <form #contant=”ngForm”>

<div ngModelGroup=”userData” #userdata=”ngModelGroup”>  
  
 </div>

</form>



setValue -> we need whole data like same as form fields.

patchValue -> here any thing is missing it ok.

Get :-

get firstname() {

return this.contactForm.get('firstname');

}

Reactive Forms Approach :-  
  
 this.userForm = new formGroup({  
 name: new formControll(‘’, Validators.requried)  
 })

<form [formGroup]=”userForm”>

<input type=”text” formControlName=”userName” />

<div formGroupName=”userForm”>  
  
 </div>

</form>

Dynamically add form controls with formArrayName:-

‘hobbies’: new FormArray([]);

Const control = new FormControl(null);

This.contantForm.get(‘hobbies’).push(control);

<div formArrayName=”hobbies” \*ngFor=”let hobb of hobbies; let I = index”>

<input type=”text” [formcontolName]=”i”

<div>

1. Pipes   
     
   Pure Pipes :-   
   impure Pipes :- pure: false. When ever data changes in components it will update

Async pipe:- it accept promise or observable. We no need to unsubscribe.

1. Http Request   
     
   HttpClientModule and HttpClient.

Observe : body (default), response, events.

Http interceptors:-

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

//do whatever you want with the HttpRequest

return next.handle(req); //invoke the next handler

}

HttpRequest, HttpHandler -> two arguments.

return next.handle(res).pipe(

tap(event => {

if(event.type === HttpEventType.Response){

console.log(event.body)

}

})

);

{provide:’HTTP\_INTERCEPTOR’,useClass:servicename,multi:true};

req.clone({

headers:req.headers.append(‘auth’,’abc’) ,

params: req.params.append(‘name’:’def’)

})

1. SwitchMap, MergeMap, ConcatMap, ExhaustMap In Angular   
     
   each value from the source observable into an inner observable, subscribes to it. It then starts emitting the values from it replacing the original value. It then waits for the inner observable to finish.

SwitchMap:- it send last came one. It skips all came previously.

MergeMap:- it send all which come first it will send.

ConcatMap:- it send all and in sequence order.

ExhaustMap:- it send first one and skip all coming next.

1. PWA In Angular   
   ng add @angular/pwa

When add service worker ng add @angular/pwa. And it add @angular/service-worker.

Import “ServiceWorkerModule” in root module. And update index.html manifest.webmanifest file. Create service worker

Configuration file ngsw-config.json (it will work in build time only not in ng serve localhost 8080).  
  
when we disconnect network and refresh page in network tab service worker enables. The service worker installed the updated

Version of application in background and next time the page is loaded or reloaded the service work switches to latest version.

The SwUpdate service gives you access to events that indicate when the service worker discovers and installs an available update for your application.

The SwUpdate service supports three separate operations:-

Get notified when an updated version is detected on the server, installed and ready to be used locally or when an installation fails

Ask the service worker to check the server for new updates

Ask the service worker to activate the latest version of the application for the current tab

In ngsw-config.json we can dataGroups -> name:api -> url (here we can add only get request only).

First we need to check wether service worker is enable are not ‘isEnabled’.

And we have property like ‘versionUpdate’ is a observable.

Unrecoverable:- the version of the application used by the service worker to serve a client might be in a broken (we will get reason also).

// poll for server update.

console.log('checked for update method call');

const interval = setInterval(async () => {

const shouldUpdate = await this.updateservice.checkForUpdate();

console.log('checked for update' + shouldUpdate);

if(shouldUpdate){

const result = await this.updateservice.activateUpdate();

console.log('Active update with result'+result);

clearInterval(interval);

}

}, 10000)

Service worker notifications:-

Invoke push notifications by pushing a message with a valid payload. See SwPush for guidance.

From “SwPush” we have requestSubscription there we need to send ‘serverPublicKey’. Here we will call api calls.

const sub = await this.pushService.requestSubscription({

      serverPublicKey: 'BLTytCGNgiogVxHoezqcGIJ1-vkVIrB4b47KZBPCzehFq9lE\_4o1YwLmJKLKBqY-cGRCPkWLMecp-\_zudy-rZvo'

    }).then((sub)=> console.log(sub))

     .catch((err)=> console.log('error occur',err));

‘messages’, ‘notificationClick’, ‘subscription’ observable’s

From messages we can update notification.

    this.pushService.messages.subscribe((message)=>{

      console.log('pushService.messages',message);

    });

    this.pushService.notificationClicks.subscribe((message)=>{

      console.log('pushService.notificationClicks',message);

    });

    this.pushService.subscription.subscribe((subscription)=>{

      console.log('pushService.subscription',subscription);

    });

Socket.io :-

import io from 'socket.io-client';

const socket = io(‘api url’);

socket.on(‘key we need to provide from server’,(res)=>{  
 // we can do our logic here.  
})

1. GruntJs

Is Task Runners. It has no.of grunt plugins

* Prefixing css rules,
* compiling sass to css,
* Minifying js / css files,
* Concatenating files
* jsHint
* live reload, page speed.

Install grunt: -

* Using grunt cli globally, and package.Json using npm init
* npm install grunt-cli -g
* npm init to create package.json
* npm install grunt –save-dev install grunt locally.
* Need to create grunt file.js.  
    
  module.export = function(grunt){

// configuration   
grunt.initConfig({  
 // pass in option to plugins , references to file etc.  
 concat:{

js:{  
 src: [‘js/\*.js’],

dest: ‘build/scripts.js’   
 },

css:{  
 src: [‘css/\*.css’],

dest: ‘build/styles.css’   
 }

},

sass:{

build:{

files:[{  
 src:’css/sass/styles.sass’,

dest: ‘build/styles.css’  
 }]

},

‘uglify’:{

Build:{

files:[{  
 src:’build/scripts.js’,

dest: ‘build/ scripts.js’  
 }]

}

}

})

// load plugins

Grunt.loadNpmTasks(‘grunt- contrib-concat’)

Grunt.loadNpmTasks(‘grunt-sass)

Grunt.loadNpmTasks(‘grunt- contrib-uglify)

// register tasks

Grunt.registerTask(‘concat-js’,[concat:js])

Grunt.registerTask(‘concat-css’,[concat:css])

Grunt.registerTask(‘run’, function (){

})

Grunt.registerTask(‘sleep’, function (){

})

Grunt.registerTask(‘all’, [‘sleep’, ‘run’])

}

* To install plugins

Npm install grunt-contrib-concat –save-dev.

1. Bower

A package manager.

npm install bower -g.

npm install jquery

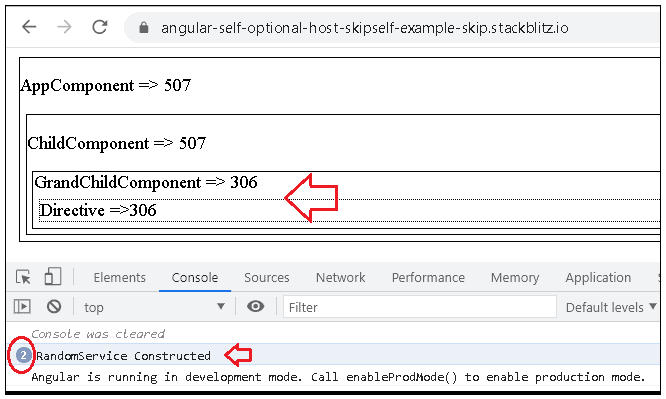
it created bower\_component folder.

Bower list –path // it shows path of files.

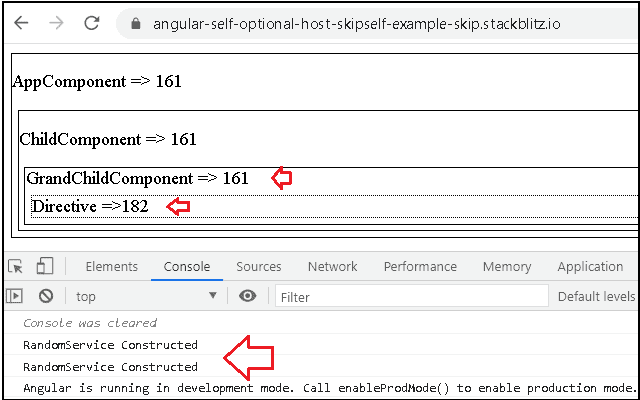
Bower init

1. @Self, @SkipSelf, @Host & @Optional

Angular Decorators that configure how the DI Framework should resolve the dependencies.

@Self:-   
 

@SkipSelf:-



@Host:-

@Optional:- if server not available it does not return null injection error.