Angular Js 5 - vatsank@gmail.com - Freelance Trainer - 20Aug2018

http://bit.ly/ang4\_vatsan

http://bit.ly/ang4\_02\_vatsan

https://github.com/vatsank

Initial Angularjs 1.x has Directive

Angular 2 - 14th Sept 2016

Angular 4 - 23rd March 2017

Angular 5 - 11th Nov 2017

Angular 6 - 03 May 2018

Angular js 1 Used JIT - Just Intime Compiliers

Angular 4 used AOT(Ahead of Time).

Angular 4 Debugging much more easier.

Angular Contains Only 8 Components:

Module

Component

Template

Metadata

DataBinding

Service

Directive

Dependency Injection

Angular is a client side MVC Application

Angular is a Single Page Application

Single Page Application(SPA)

One Real HTML Page, whose content can be changed in Javascript without having to download a new page.

Server Sends an application engine along with the HTML Page.

This Application Engine controls the entier application including processing, input, output,painting and loading of the HTML Pages.

Four Principles of the SPA:

Chunking:

Pages are constructed by loading chunks of HTML Fragements

Controllers

Javascript code that handles complex DOM

Templating:

Coding UI and DOM manipulates are replaced

Rourting

Selection of Views and Navigation that preserves page state, elements and data

**Introduction of javascript**

Added by using Script Tag

<script></script>

When

**<noscript> ==** When javascript is disabled.

Define a function or variable we have to create at the top called head

Functional Programming Language

Object Based Programming Language

Event Driven Programming Language

Any function written is associated to a window object in the javascript.

Javascript is a loosely typed

**Anonymous Types:**

Var greet=function(person, greet){

Var text = greet +”-”+person;

Console.log(text);

}

Also called as function definition.

Anonymous Functions -> Functions Passing Functions.

Function useAnnonFunction(RefFun){

Var ref=RefFun();

Console.log(ref)

}

useAnnonFunction(function()

{

Return “I am from Annon function”

}

);

Also called the call back functions.

**Event Programming Languauge**

Callback Functions.

addEventlistener

var clickButton = document.getElementById(‘btn2’)

clickButton.addEventListener(‘click’,function(){

console.log(‘I am added’); }

);

**Object Based Programming Language:**

Var obj={}

Obj.FName = “Krish”;

Obj.LName = “Kumar”;

Obj.concat=function(){

Return this.FName+this.LastName;

}

Property without brasses

**Local and Global Variables:**

With or without Var would define the local or global variables.

Variables without var would make variables as Global.

**Nested Functions:**

Prototype vs Dynamic Properties.

Dynamic property are set are to local in nature

Prototype would be access to all the objects created.

**Node Js**

Has three modules

-Core Modules

-Local Modules

-Third Party Modules

Day 2:

Modular Loader.

Two Popular Module Loaders are Webpack and SystemJS

Angular JS itself moved to Webpack, Its able to bundle any kind of file and come with a self server.

Install Angular CLI

Npm install –g @angular/cli

Ng –version

Creating the Project

-Created by executing the following commond:

Ng new “gettingstarted” –routing

Creates a folder getting started and copies all the required files and configuration setting.

Angular Application Folder Structure:

It contains the following three folders:

Src - source Code of the project

e2e – end to end testing

node – all node pages will be installed.

Angular.json is the one changed with multiple versions.

Karma.conf.json Uses Jasmine testing framework.

Packages.json:

Scripts, dependencies and devDependencie

"dependencies": {

"bootstrap":"^4.1.3",

"font-awesome":"^4.7.0",

Then just do npm install then

PS C:\Users\Palmeto\Documents\Krish\CodeExcersies\foodhub> npm install

Install Augury Chrome Extension for seeing the component structure.

ngModule:

Angular js everything is a

Creating a Component:

ng g c header

CREATE src/app/header/header.component.html (25 bytes)

CREATE src/app/header/header.component.spec.ts (628 bytes)

CREATE src/app/header/header.component.ts (269 bytes)

CREATE src/app/header/header.component.css (0 bytes)

UPDATE src/app/app.module.ts (475 bytes)

Angular Template:

Create a footer Component:

Ng g c footer

Add the div in the app.components . html

Ng g I restaurant

I is for interface.

Angular Service:

Ng g s catalogAPI > this is for creating a service

Ng g I catalog > This is used for INterface

Add code to the interface:

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

In @NgModule({

Add the provider

providers: [HttpClient],

in the Catalog-api.service.ts

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

--- Angular Js Exercise from Scratch – **21/08/2018**

New project south Food.

ng new southfood –routing

ng g I southcusinie > create an interface

ng g c southcatalog > create an catalog

npm start / ng serve –o to run the application

To start a json server to load the json data:

json-server .\southcatalog.json –watch

in the interface southcusine.ts write the following code:

export interface Southcusine {

id: number;

name: string;

ratePerUnit: number;

img: string;

}

In the southcatalog component create list of SouthCusine:

southList: Southcusine[];

In the constructor create a list

southList: Southcusine[];

constructor() {

const southupma = {id: 200, name: 'Upma',

ratePerUnit: 60, image: 'assets/images/kara.jpg' };

const southpongal = {id: 300, name: 'pongal',

ratePerUnit: 90, image: 'assets/images/biryani.jpg' };

const southwada = {id: 400, name: 'wada',

ratePerUnit: 160, image: 'assets/images/nan.jpg' };

this.southList = [southupma, southpongal, southwada];

}

Add bootstrap js files

Add the following package.json

"bootstrap": "^4.1.3",

"font-awesome": "^4.7.0"

In Style.css use the following references:

@import "../node\_modules/font-awesome/css/font-awesome.min.css";

@import "../node\_modules/bootstrap/dist/css/bootstrap.min.css";

Then run npm install

Then ng serve –o to test the application

Angular Service:

Json Data:

{

"southindian": [

{

"id":200,

"name":"idly",

"ratePerUnit":25.00,

"img":"assets/images/kara.jpg"

},

{

"id":300,

"name":"dosa",

"ratePerUnit":35.00,

"img":"assets/images/biryani.jpg"

}

]

}

Create a Service

ng g s southServiceAPI

Create a Interface

Ng g I southservice

Got to app>app.module.ts

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

And import it in the NgModule

Before:

imports: [

BrowserModule,

AppRoutingModule

],

After:

imports: [

BrowserModule,

AppRoutingModule, HttpClientModule

],

Then add the service in the provider:

Before:

providers: [],

After:

providers: [HttpClient],

In the south service.ts add the Httpclient Import

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

Then import the Observer in the service

import { observable } from 'rxjs';

Put the base url in the South Service Class

export class SouthServiceAPIService {

baseURL = 'http://localhost:3000/';

constructor() { }

}

Then update the constructor:

constructor(private http: HttpClient) { }

Then start the json server:

json-server .\southcatalog.json – watch

go to southcatalog component and add the following in the service:

@Component({

selector: 'app-southcatalog',

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

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

})

After

@Component({

selector: 'app-southcatalog',

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

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

providers: [SouthServiceAPIService]

})

Create Interface SouthInterface data:

export interface SouthInterface {

id: number ;

name: string ;

ratePerUnit: number;

image: string;

}

Now update the ngOnInit()

Before:

ngOnInit() {

}

Add the service in the constructer and using this service call the API

constructor(private service: SouthServiceAPIService) {

}

ngOnInit() {

this.service.findAll().subscribe(data => this.southListAPI = data);

}

Now update the following and see the south API is assigned with data.

<div class="card-deck">

<div \*ngFor="let eachItem of southListAPI" class="card">

<div class="card-img">

<img src="{{eachItem.img}}" alt="logo"></div>

<div class="card-body">{{eachItem.id}}</div>

<div class="card-body">{{eachItem.name}}</div>

<div class="card-body">{{eachItem.ratePerUnit}}</div>

</div>

</div>

{{southListAPI|json}}

!!!!! Congrats you are done!!!!!!!!!

Directives:

* NgFor, ngif, NgModel, ngclass

Attribute Directives

Ngclass and ngstyle attribute level directives.

**Day 3:**

Creating nested components:

ng g c menu

We need to add forms Module to use two a binding

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

imports: [

BrowserModule,

AppRoutingModule, HttpClientModule, FormsModule

],

Now create a order component

Ng g c order

**Parent to Child Communication:**

Ng g c parent

Ng g c child

In Child Component:

@Input() itemToAdd: string;

In Parent Component:

name = 'Crocin';

in Parent Component.html

<app-child [itemToAdd]='name'></app-child>

Here name is passed to child as itemtoAdd

In Child Component.html print the variable:

{{itemToAdd}}

Communication from child to parent

Childcomponent.ts

Import output and Event Emitter

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

Declare and EventOutput

@Output() result: EventEmitter<string> = new EventEmitter();

Then emit the output

confirm() {

this.result.emit(this.itemToAdd + 'Added');

}

Go to parent component.ts

response: string;

onChange(val) {

this.response = val;

}

Parentcomponent.html

<p>

parent works!

<input type="text" [(ngModel)]='name'>

<app-child [itemToAdd]='name'

(result)='onChange($event)'></app-child>

</p>

{{response}}

Another example

**Using viewchild to get the child details**

Create distributor component & salesmen component

In salesMen write the extended method:

getSalesMen(): string[]{

return ['reddy','karthik','nair','khan'];

}

In Distributors write the following method:

getDistributors(): string[]{

return ['Novartis','Sun Pharma','Aurobindo']

}

In Parent where we called Salesmen and Distributors use the following to call viewChild

@ViewChild(SalesmanComponent) salesman: SalesmanComponent;

@ViewChild(DistributorComponent) distributors: DistributorComponent;

Now extend the Parent class to implement AfterContentInit

export class ParentComponent implements OnInit, AfterContentInit {

Now call the showdetails to get the lists populated

showDetails() {

this.salesmanlist = this.salesman.getSalesMen();

this.distlist = this.distributors.getDistributors();

}

Now implement ngAftercontentInit as viewChild only available after the Init

ngAfterContentInit(): void {

this.showDetails();

}

In the parentcomp.html use the following to pull the Salesman and distribution list:

<app-distributor></app-distributor>

<app-salesman></app-salesman >

{{salesmanlist}}

{{distlist}}

Sibling Concept using a service:

Create a service through Behaviour:

message: BehaviorSubject<string> = new BehaviorSubject('');

currentMessage = this.message.asObservable();

constructor() { }

change(msg: string): void {

this.message.next(msg);

}

Now in parent component

ngOnInit() {

this.service.change('Send Items the earliest');

}

In the sibling Component:

Inject the service and then subscribe the message

export class SiblingComponent implements OnInit {

info: string;

constructor(private service: CommServiceService) { }

ngOnInit() {

this.service.currentMessage.subscribe(msg => this.info = msg);

}

Do the reverse set the change in the sibling and subscribe in the component and print the {{info}} in html

Custom Directive

Create a ng g d to create a directive

ng g d highlight

A highlight.directive.ts is created.

Update the constructor

constructor(private element: ElementRef, private rendere: Renderer2) { }

Create hostlistener and create a custom hover method to create a custom listener

constructor(private element: ElementRef, private rendere: Renderer2) {

}

@HostListener('mouseover') over() {

this.hover(true);

}

@HostListener('mouseout') out() {

this.hover(false);

}

hover(status: boolean)

{

if (status) {

this.rendere.setStyle(this.element.nativeElement, 'text-decoration', 'underline');

} else {

this.rendere.setStyle(this.element.nativeElement, 'text-decoration', 'none');

}

}

Showing Dynamic Components at Runtime:

Create a ng g c details Component

Add EntryComponents the component to tell the angular we will use this component real time:

App.module.ts add the component in the entry components to tell we use this dynamically

entryComponents: [DetailsComponent]

Then create a service to inject this component:

ng g s compAdder

declare a viewRef of ViewContainerRef type

private viewRef: ViewContainerRef;

constructor(private factResolver: ComponentFactoryResolver) { }

Inject it in the constructor

And the set the viewref

setViewRef(viewRef: ViewContainerRef) {

this.viewRef = viewRef;

}

Now write code to addcomponent:

addComponent(compToAdd: any) {

const factory = this.factResolver.resolveComponentFactory(compToAdd);

const compRef = factory.create(this.viewRef.parentInjector);

this.viewRef.insert(compRef.hostView);

}

Go to childcomponent or some component where we want to use the component:

constructor(private service: CompAdderService) { }

Then import in the action method:

this.service.addComponent(DetailsComponent);

Now go to child component html and create a place holder where to render the component.

<div #placeholder></div>

Create the viewChild

@ViewChild('placeholder', {read: ViewContainerRef}) inPos: ViewContainerRef;

Then

confirm() {

this.result.emit(this.itemToAdd + 'Added');

this.service.setViewRef(this.inPos);

this.service.addComponent(DetailsComponent);

}