Angular 6

CLI is the acronym of Command Line Interface, which can be used to create the Angular JS application. Using CLI, you can also create a unit and end-to-end tests for the Angular application.

The Angular 2 comprises of the following key components:

� Module – This is used to break the application into the logical pieces of the program code and each piece of code or module is designed to perform a single and unique task.

� Component – This is used to bring the modules together.

� Templates – This is used to define the Views of an Angular JS application.

� Metadata – This is used to add more data to an Angular JS application.

� Service – This component is used to develop the components, which can be used to share in the entire application.

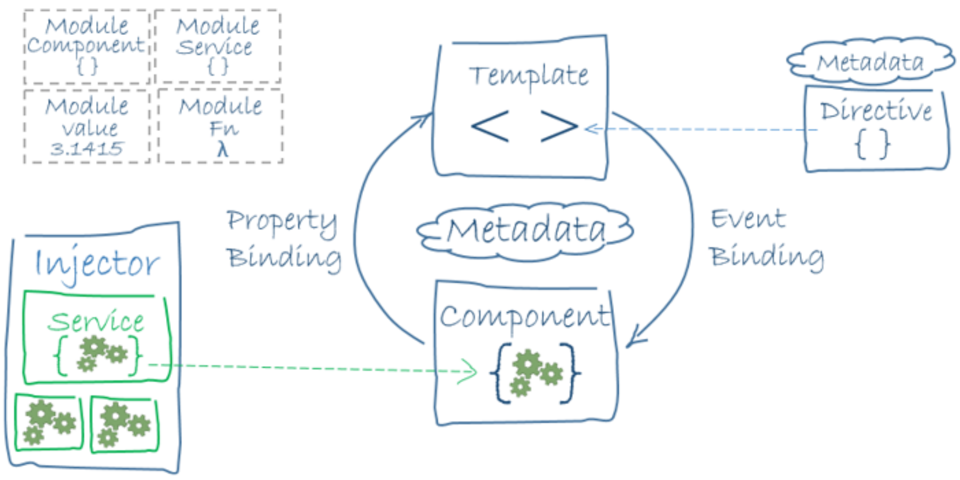
Step1:-install

npm install -g @angular/cli

-Create a workspace and initial application

ng new my-app

cd my-app ng serve –open



Step2:-

Src/app/app.module.ts

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

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

```
@NgModule({  imports:      [ BrowserModule ],
 providers:    [ Logger ],
 declarations: [ AppComponent ],
exports:      [ AppComponent ],
bootstrap:    [ AppComponent ]})
export class AppModule { }
```

An NgModule is defined by a class decorated with @NgModule(). The @NgModule() decorator is a function that takes a single metadata object, whose properties describe the module. The most important properties are as follows.

- declarations: The components, *directives*, and *pipes* that belong to this NgModule.
- exports: The subset of declarations that should be visible and usable in the *component templates*of other NgModules.
- imports: Other modules whose exported classes are needed by component templates declared in *this* NgModule.
- providers: Creators of services that this NgModule contributes to the global collection of services; they become accessible in all parts of the app. (You can also specify providers at the component level, which is often preferred.)
- bootstrap: The main application view, called the *root component*, which hosts all other app views. Only the *root NgModule* should set the bootstrap property.

Step3:-

- `selector`: A CSS selector that tells Angular to create and insert an instance of this component wherever it finds the corresponding tag in template HTML. For example, if an app's HTML contains `<app-hero-list></app-hero-list>`, then Angular inserts an instance of the `HeroListComponent`view between those tags.
- `templateUrl`: The module-relative address of this component's HTML template. Alternatively, you can provide the HTML template inline, as the value of the `template` `property. This template defines the component's` *host view*.
- `providers`: An array of providers for services that the component requires.

App.component.ts.....

title: 'nilofar';

Create component:-

ng g c test

---app.cmodule.ts should import that component

```
import { TestComponent } from './test/test.component';
@NgModule({
declarations: [
AppComponent,
TestComponent
],
```

---app component is our root component so our generated component should add into this root component

```
<app-test></app-test>
```

-----template & style can be inline

```
template : '<div>nilofar</div>',
styles: [`
div {
color: red;
}`]
```

Bootstap intall

npm install --save bootstrap@4

Step4:-

Interpolation:

```
public name = 'javed';
{{ name }}
```

Can use javascript property:

{{ name.length }}

{{ name.toUpperCase() }}

string interpolation-----

str: String = 'nilofar';

<img src="{{imageSrc}}" />   ---by interpolation

imageSrc = ('assets/1.png');

Attributes are html

Properties are dom

A<u>ngular supports data binding, a mechanism for coordinating parts of a template with parts of a component. Add binding markup to the template HTML to tell Angular how to connect both sides.</u>

<u>Property Binding:</u>

export class TestComponent implements OnInit {

public testid = "Nilo";

public isdisabled = true;

<input [disabled] = "isdisabled" id={{testid}} type="text" value="nilofar">

<u>Class Binding:-</u>

<h2 class="text-success">Codevolution</h2>

<h2 [class]="successClass">Codevolution</h2>

<h2 class="text-special" [class]="successClass">Codevolution</h2>

<h2 [class.text-danger]="hasError">Codevolution</h2>

<h2 [ngClass]="messageClasses">Message</h2>

public successClass = "text-success";

public hasError = false;

public isSpecial = true;

public messageClasses = {

"text-success": !this.hasError,

"text-danger": this.hasError,

"text-special": this.isSpecial

}

Style binding:-

1]<p>

test works!

</p>

<h1 [style.color]="'red'">hello</h1>

2]

public hasError = false;

<h1 [style.color]=" hasError ? 'red':'blue'">hello</h1>

3]public titleStyles = {

color: 'blue',

fontStyle: 'italic'

}

<h2 [ngStyle]="titleStyles">Style Binding 4</h2>

Event-Binding:-

<button (click)="onClick()">green</button>

onClick() {

console.log('hello');

By interpolation:

<button (click)="onClick()">green</button>

{{greeting}}

public greeting = "";

constructor() { }

ngOnInit() {

```
}
onClick() {
this.greeting = "hello";
console.log('hello');
}
}
```

safe navigation operator (?.) is a fluent and convenient way to guard against null and undefined values in property paths

Template Reference Variables:-

Template variables are used to access the values of DOM element properties. It is declared with the help of "#" and "ref-"as prefix. For example: – #myVar and ref-myVar. Template variable names cannot be made duplicate as in this way, it might give unpredictable values. The scope of a reference variable is the entire template. It can be used anywhere inside a template. In Angular 2, a component needs to have a view and to define a view, a template variable is used. It allows us to express data and property binding, event binding and template concerns.

Can be used as refrence for html or dom property

```
<input #myInput type="text">
<button (click)="logMessage(myInput.value)">Submit</button>

export class TestComponent implements OnInit {
logMessage(value){
console.log();
}
}
```

<u>**Two way Data Binding:**</u>

Data binding in AngularJS is the synchronization between the model and the view.

The data model is a collection of data available for the application.

You often want to both display a data property and update that property when the user makes changes.

<input [(ngModel)]="name">

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

imports: [

BrowserModule,

FormsModule

],

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

{{name}}

Structural Directive:

NgIf:

public name = false;

<h2 *ngIf="name; else elseblock">

codevolution

</h2>

<ng-template #elseblock>

<h2>

no codevolution

</h2>

</ng-template>

2]public name = true;

<h2 *ngIf="name; then thenblock; else elseblock">

</h2>

<ng-template #thenblock>

<h2>here is code</h2>

</ng-template>

<ng-template #elseblock>

```
<h2>
no codevolution
</h2>
</ng-template>
```

NgSwitch:-

```
<div [ngSwitch]="color">
<div *ngSwitchCase="'red'">you pick color red</div>
<div *ngSwitchCase="'white'">you pick white</div>
<div *ngSwitchDefault>pick any color</div>
</div>
public color = 'red';
```

NgFor:

```
<p>
test works!
</p>
<div *ngFor="let color of colors ; index as i">
{{i}} {{color}}
</div><br>

<div *ngFor="let color of colors ; first as f">
{{f}} {{color}}
</div>
<div *ngFor="let color of colors ; last as l">
```

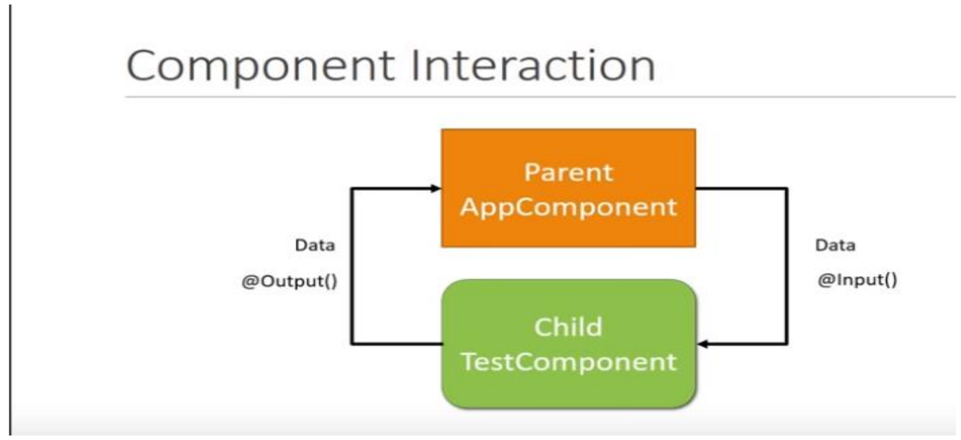
```
{{l}} {{color}}
</div><br>
<div *ngFor="let color of colors ; odd as o">
{{o}} {{color}}
</div><br>
<div *ngFor="let color of colors ; even as e">
{{e}} {{color}}
</div>
public colors = ['red', 'white', 'black'];
```

<u>**Component Interaction:**</u>

Parent to child component interactions

Child take input from parent

Parent gives output to child



There are two ways to exchange data between components: Using the @Input() and @Output() decorators or a Service.

Steps:-

1]import input ...public name="jamir"   //app.ts parent

2] *`<app-form [parentdata]="name"></app-form> //ap.comp.html parent`*

3] *`@Input() public parentdata; import input form.comp.ts child`*

4]`{{parentdata}}  form.html  child`

Steps for output: child to parent

1]Import output,eventEmitter from angular/core

@Output() public childEvent = new EventEmitter<string>(); //form.ts

fireEvent() {

this.childEvent.emit('hey');

}

<button (click)=fireEvent()>Send Event</button> //form.html

2]<app-test (childEvent)='message=$event' [parentData]='name'></app-test>`, app.html

{{message}}

public message = ""; app.ts

App.component(parent)

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


@Component({
selector: 'app-root',
template: `hi {{message}}
<app-test (childEvent)='message=$event' [parentData]='name'></app-test>`,
styleUrls: ['./app.component.css']
})
export class AppComponent {
title = 'inputoutputs';
public name = "Vishwas";
public message = "";
}
```

Test.component(child)

```
import { Component, OnInit, Input, Output, EventEmitter} from '@angular/core';
@Component({
selector: 'app-test',
template: `<h2>
{{name}}
</h2>
<button (click)=fireEvent()>Send Event</button>`,
styleUrls: ['./test.component.css']
})
export class TestComponent implements OnInit {
@Input('parentData') public name;
@Output() public childEvent = new EventEmitter<string>();


constructor() { }


ngOnInit() {
}
fireEvent() {
this.childEvent.emit('hey');
}


}
```

Pipes:-

```
<p>
test works!
</p>
{{ name| uppercase }}<br>
{{ name| lowercase }}<br>
{{ name| titlecase }}<br>
{{ name| slice:3:5 }}<br>
{{ message | json }}<br>
{{2.5670 | number:'1.2-3'}}<br>
{{2.567 | percent}}<br>
{{2.567 | currency}}<br>
{{date}}<br>
{{date | date:'short'}}<br>
{{date | date:'shortDate'}}<br>
{{date | date:'shortTime'}}<br>
{{2.567 | currency:'EUR':'code'}}<br>
```

```
export class TestComponent implements OnInit {
public name = 'nilofar';
public message = {
'firstname': 'Nilofar',
'lastname': 'shaikh'
}
public new date= Date();
```

<u>Number:-</u>

1Minimum number of integer digit

2Minimum Decimal place

3maximum to decimal

Services:-

Share data in multiple component

ng g s employee

Register injector:-register your service in module always

```
import { EmployeeService } from './employee.service';
```

```
providers: [EmployeeService],
```

1]define employee service class

2]

3]

Service.ts

Steps

1]define the class in service

2]injector..

3]define injector in component..n define empty array in your child component

4]import service in module level & selector

```
import { Injectable } from '@angular/core';
@Injectable({
providedIn: 'root'
})
export class EmployeeService {
news = [
{id: 1, title: 'news'},

{id: 2, title: 'news'},


{id: 3, title: 'news'},


];
constructor() {}
getNews() {
return this.news;

}
}
```

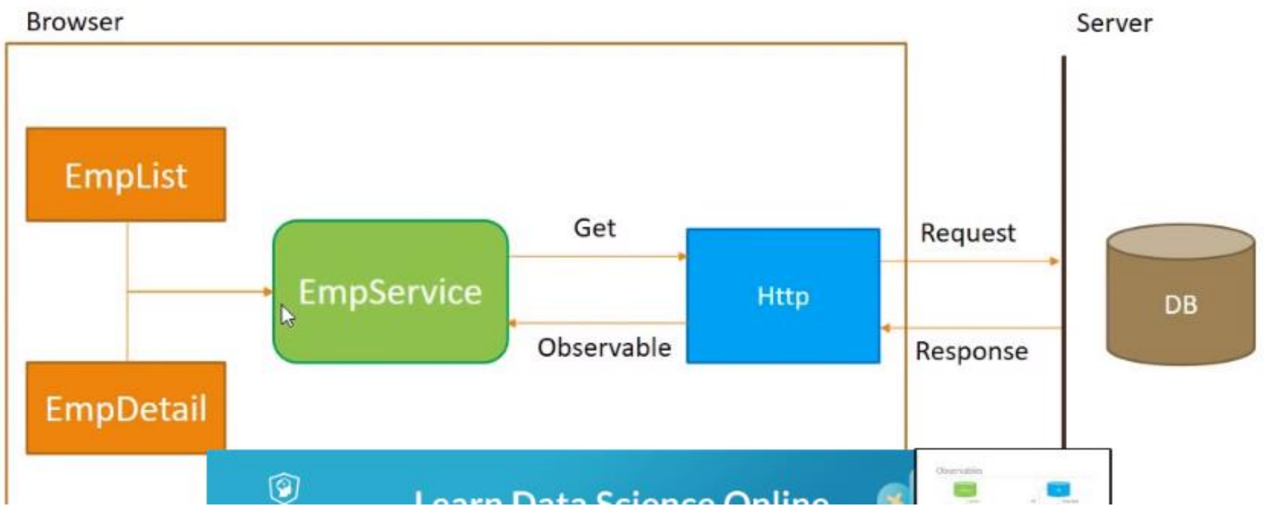
Employee-list Component.ts

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

import { EmployeeService } from '../employee.service';



@Component({

selector: 'app-employee-list',

templateUrl: './employee-list.component.html',

styleUrls: ['./employee-list.component.css']

})

export class EmployeeListComponent implements OnInit {

news: {};

constructor(newslist: EmployeeService) {

this.news = newslist.getNews()

}

ngOnInit() {

}

}
```

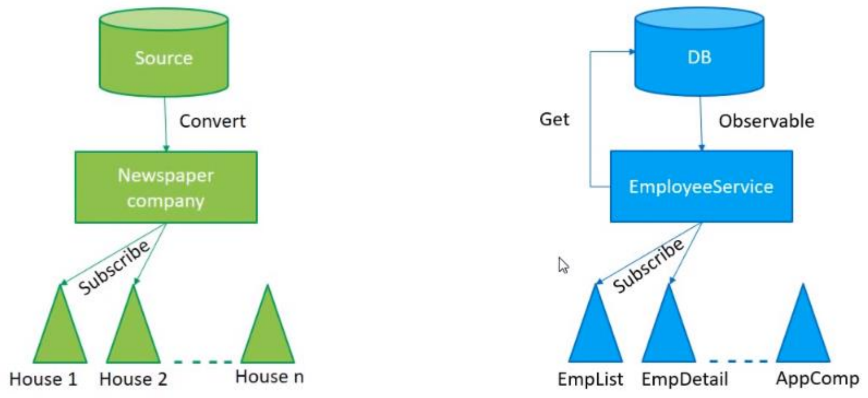
.html

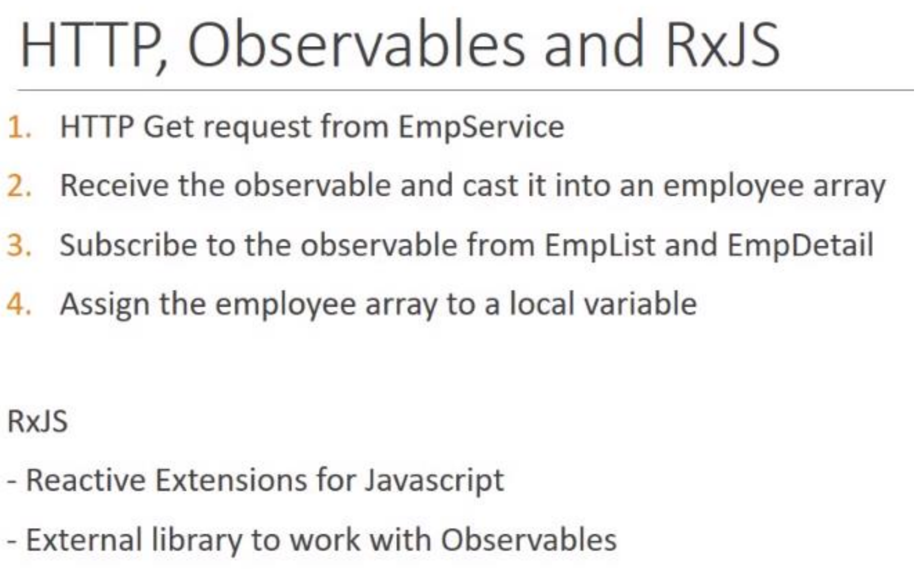
```
{{news | json}}
```

Fetch http data:



Observables





HTTP, Observables and RxJS

1. HTTP Get request from EmpService
2. Receive the observable and cast it into an employee array
3. Subscribe to the observable from EmpList and EmpDetail
4. Assign the employee array to a local variable

RxJS

- Reactive Extensions for Javascript
- External library to work with Observables

Fetch Data Using HTTP

HTTPCLIENTMODULE INSTEAD OF HTTPMODULE

{{{component....declaration

Modules...import

Services....providers}}

Service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient} from '@angular/common/http';
import { IEmployee } from './employee';
@Injectable()
export class EmployeeService {
  private _url: string = "/assets/data/employees.json";
  constructor(private http:HttpClient) { }
  getEmployees(): Observable<IEmployee[]>{
    return this.http.get<IEmployee[]>(this._url)
                  }
```

Employee.ts

```
export interface IEmployee {
   id: number,
   name: string,
   age: number
}
```

Employee.json

```
[
 {"id": 1, "name": "Andrew", "age": 30},
 {"id": 2, "name": "Brandon", "age": 25},
{"id": 3, "name": "Christina", "age": 26},
 {"id": 4, "name": "Elena", "age": 28},
 {"id": 5, "name": "Felicia", "age": 25}
]
```

EmployeeDetailComponent.ts

```
import { EmployeeService } from './../employee.service';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'employee-detail',
  template: `
    <h2>Employee Detail</h2>
    <h3>{{errorMsg}}</h3>
    <ul *ngFor="let employee of employees">
      <li>{{employee.id}}. {{employee.name}} - {{employee.age}}</li>
    </ul>
  `,
  styles: []
})
export class EmployeeDetailComponent implements OnInit {

  public employees = [];
  public errorMsg;

  constructor(private _employeeService:EmployeeService) { }

  ngOnInit() {
    this._employeeService.getEmployees()
```

```
      .subscribe(data => this.employees = data,
              error => this.errorMsg = error);

 }

}
```

Routing & Navigation

The Angular Router enables navigation from one view to the next as users perform application tasks.

This guide covers the router's primary features, illustrating them through the evolution of a small application that you can run live in the browser / download example.

Overview

The browser is a familiar model of application navigation:

- Enter a URL in the address bar and the browser navigates to a corresponding page.
- Click links on the page and the browser navigates to a new page.
- Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've se

Steps:

1]add base tag

2]app routing module import

```
imports: [
 AppRoutingModule
 ],
 import {AppRoutingModule,routingComponents } from './app-routing.module';
```

3]generate 2 component

4]

```
const routes: Routes = [
{ path: '', redirectTo: '/departments', pathMatch: 'full' },

{ path: 'departments', component: DepartmentListComponent },

]
```

4]with <router-outlet> directives

Mangodb collection & document (w3 school)

Src ...app.js (database create file)

C—data—db—folder create

C program file.mangodb.server..3.2..bin.....command window give path of storage

Give port 27017 connection message

Show dbs ...command window

Use companyDetail

G:\init\npm install express

(go to project folder & install express)

npm install mangodb@2.2.33

node app.js....show my server is running