Angular 6

By : Akash Kale


Angular is a development platform for building
mobile and desktop applications

Angular 2/4

- Angular allows you to develop single page applications which reload only the part of the page when user interacts with the page by clicking on a link or a button.
- Angular uses components to display the content on the page and these components are replaced by other components at runtime when user wants to see other contents while interacting with the page.

Continue...

- Angular is a framework for building client applications using HTML and Typescript (a super set of JavaScript) which is compiled to JavaScript.
- Angular JS 1.x was limited to a framework to build only web applications, whereas Angular 2 or later has grown into a strong platform which allows you to develop fast and scalable applications across all platforms such as web, mobile web, native mobile and native desktop.

SPA Vs MPA

- SPA means Single Page Application
- MPA means Multiple Page Application
  - Faster navigation
  - Improved user experience
  - Decoupling of front-end and back-end development

Angular 2/4 setup

- Setting up and configuring angular project takes time as it involves adding lot of
  - JS libraries
  - Compiler to Typescript
  - Configuring server etc
- So Angular team has provided a high quality development toolset to ease the application set up and configuration one such toolset you can use is Angular CLI (Command Line Interface) which automate your development workflow.

Angular CLI

- Create new angular application.
- Run a development server with a LiveReload support to preview your application during development.
- Add features to your existing angular application

build deployment and production ready application

- Run your application's unit tests
- Run your application's end to end tests(E2E tests)

Pre-requisites for Angular2/4

- HTML/HTML5
- CSS/CSS3
- JavaScript
- TypeScript
- Web Technologies (Servlet/ JSP, PHP, ASP.net – Optional)

Software's required

- Node.js 6.9.0 or later
- NPM 3.0 or later (comes with Node.js).
- Editor Visual Studio Code or Brackets or Sublime.

Why node.js ?

"Node's goal is to provide an easy way to build scalable Network programs"

What is NodeJS ?

- NodeJS is an open source , cross platform runtime environment for server side and networking application.
- It is written in JavaScript and can run on Linux , Mac ,Windows , FreeBSD.
- It provided an event driven architecture and a non blocking I/O that optimize and scalability. These technology uses for real time application.
- It used Google JavaScript V8 Engine to Execute Code.
- It is used by Groupon, SAP , LinkedIn ,Microsoft,Yahoo ,Walmart ,Paypal.

What is unique about Node.js?

- JavaScript used in client-side but node.js puts the JavaScript on server-side thus making communication between client and server will happen in same language

- Servers are normally thread based but Node.JS is “Event” based. Node.JS serves each request in a Event loop that can able to handle simultaneous requests.

- Node.JS programs are executed by V8 JavaScript engine the same used by Google chrome browser.

Node.js is not.....

- Another Web framework
- For beginner
- Multi-thread

In Simple Node.js

Node.js = Runtime Environment + JavaScript library.

Why node.js ?

- Non Blocking I/O
- V8 JavaScript Engine
- Single Thread with Event Loop
- 40,025 modules
- Windows, Linux, Mac
- 1 Language for Frontend and Backend
- Active community

Why JavaScript ?!!!

- Friendly Callbacks
- Ubiquitous
- No I/o Primitives
- One language to RULE them all

JavaScript is well known for client-side scripts running inside the browser

node.js is JavaScript running on the server-side

SSJS -> Server-Side JavaScript

Use a language you know

Use the same language for client side and server side

ES6

- JavaScript is ES6 or ES 2015 as well as ES7 or ES2016, ES means ECMAScript which is a standard for scripting language ECMA means European Computer Manufacturer's Association.
- The ES6 or ES6 script can't understand by browser.
- So it required transpiler, which help to convert ES6 or ES7 to ES5(JavaScript).
  - TypeScript transpiler
  - Babel transpiler
  - Traceur transpiler

TypeScript

- *TypeScript* is a free and open source programming language developed and maintained by Microsoft.
- It is a strict superset of JavaScript, and adds optional static typing and class-based object-oriented programming to the language.
- It is designed for development of large applications and transpiler to JavaScript. As TypeScript is a superset of JavaScript, any existing JavaScript programs are also valid TypeScript programs.

Features of the TypeScript

- Data Types Supported
- ES6 function
- Classes
- Inheritance
- Interface
- Modules
- Source File Dependencies

Variable declaration

- In ES6, we can use var, let and const keyword to declare the variable.
- var keyword is used to declare the function scoped variable.
- let keyword is used to declare the block scoped variable.
- const keyword is used to declare the constant value.

Data Types

- Any
- Primitive
  - number
  - boolean
  - string
  - void
  - null
  - undefined - Same as JS
- Array
- Enum

TypeScript functions

- Normal function
- Function with parameter
- Function with parameter variable with datatype
- Function with return types.
- Function with required parameter
- Function with default parameters
- Function with optional parameters

Continue...

- Rest parameter
- Spread parameter
- Arrow function (lambda expression)

Default parameter sample

```
function empDetails1(id,name,salary){
    console.log("id is "+id+" Name is "+name+" Salary is "+salary)
}
//empDetails1();
function empDetails2(id=0,name="Unknown",salary=0.0){
    console.log("id is "+id+" Name is "+name+" Salary is "+salary)
}
empDetails2(100,"Raj",12000);
empDetails2(101,"Seeta");
empDetails2(103);
empDetails2();
```

Rest Operator

- RestParameter is use to take or receive the variable number of parameter and put into a single array.

```
let studentsNames = function(...stdName){
    for(let std in arguments){
        console.log(arguments[std])
    }
}
studentsNames("Raj");
studentsNames("Raj","Seeta");
studentsNames("Raj","Seeta","Reeta");
```

Spread parameter

- Spread parameter is used to takes an array and put individual to elements or variable.

```
let studentsNames = function(...stdName){ // Rest parameter
    for(let std in arguments){
        console.log(arguments[std])
    }
}
let stdNames=["Raj","Seeta","Veeta","Keeta"];
studentsNames(...stdNames); //spread parameter
//studentsNames("Raj");
//studentsNames("Raj","Seeta");
//studentsNames("Raj","Seeta","Reeta");
```

Rest Parameter Vs Spread parameter

- **Rest Parameter declare inside the function declaration and spread parameter declared while calling the function.**

Destructing the Array

```
let employee=[100,'Raj',12000];
let [empId,firstName,salary]=employee;
console.log("Employee id is "+empId)
console.log("First name is "+firstName)
console.log("Salary is "+salary)

let [,ename,]=employee;
console.log("Employee first nanme is ",ename);

let stdInfo=[1,"Raj",45,67,87,56];

let [sId,sName,...marks]=stdInfo;
console.log("Student Id "+sId);
console.log("Student Name "+sName);
console.log(marks);
```

Destructs the objects' property

```
let employee={
    empId:100,
    fname:"Raj",
    lname:"Deep",
    salary:12000
}

//Here the normal variable name and object property name must be match
let {empId,fname,lname,salary}=employee;
console.log("Id is "+empId);
console.log("First Name is "+fname);
console.log("Last Name is "+lname);
console.log("Salary "+salary);

//Here the object property given as a alias name
let {empId :id,fname:f,lname:l,salary:sal}=employee;
console.log("Id is "+id);
console.log("First Name is "+f);
console.log("Last Name is "+l);
console.log("Salary "+sal);
```

ES6 String

- let msg1=“Simple Msg”; //double quote
- let msg2=‘Simple Msg’; // single quote
- let msg3=`Simple Msg`;//backticks quote
- let name =“Raj Deep”;
- let msg4 = `The name is

  ${name} `; //string template

Different way for loop

- let abc=[“a”,”b”,”c”,”d”]
- First way

  for(let i=0;i<abc.length;i++){

  console.log(abc[i])

  }
- Second way

  for(let index in abc) {

  console.log(abc[index])

  }

Continue...

- Third way

```
for(let val of abc) {
 console.log(val)
}
```

- Fourth way

```
abc.forEach(function(value){
console.lot(value)
})
```

using arrow function

- Fifth way

  abc.forEach((value)=>console.log(value))

Classes

- Contains set of properties and behaviour
- Instance methods/members
- Static methods/members
- Can implement interfaces
- Inheritance
- Single constructor
- Default/Optional parameter
- ES6 class syntax

Inheritances

- Using classes
- Using interface

Source File Dependencies

- Can be done using reference keyword
- Must be the first statement of file
- Paths are relative to the current file
- Can also be done using tsconfig file

Example Demo

```
///<reference path="./a.ts">
///<reference path="./b.ts">
abc();
xyz();
```

Modules

- Modules can be defined using module keyword
- A module can contains sub-modules, class, enum or interfaces. But can't directly contains functions.
- Modules can be nested(sub-modules).
- Classes and Interfaces can be exposed using export keyword.

Types of module

- Internal module
- External module

Sample Example internal module

- **a.ts**

```
module A{
    export function abc(){
    console.log("Abc function from a.ts")
    }
}
```

- **b.ts**

```
module B {
    export function abc() {
    console.log("Abc function from b.ts")
    }
}
```

- **main.ts**

```
///<reference path="./a.ts"/>
///<reference path="./b.ts"/>
A.abc();
B.abc();
```

Sample Example external module

- **a.ts**

```
export function abc(){
console.log("Abc function from a.ts")
}
```

- **b.ts**

```
export function abc() {
console.log("Abc function from b.ts")
}
}
```

- **main.ts**

```
import {abc} from "./a";
import {xyz} from "./b";
abc();
xyz();
```

TypeScript Decorators

- Decorator is nothing but it is special kind of declaration that can be attached to the classes, properties and methods.
- Decorator can be evaluated into a function that to be called at the runtime. The decorator being with the @ symbol.

Angular 2

The Main Building Blocks

- Module
- Component
- Metadata
- Template
- Data Binding
- Pipes
- Service
- Directive
- Dependency Injection

Installation of Angular

- ng
- npm install –g @angular/cli
- ng version
- ng new projectName
- cd projectName
- npm start

Metadata (with decorators)

- Decorators are functions that modify JavaScript classes.
- Angular has many decorators that attach metadata to classes so that it knows what those classes mean and how they should work.
- Decorators like a annotation in Java Technology

Few angular 2 decorator

- Pre-defined decorator
  - @NgModule
  - @Component
  - @Injectable
  - @Input
  - @Output
- Each decorator accepts a well-defined, specific *configuration* object that contains information about the class or property to be decorated.

Modules

- Angular applications are composed of *modules*.
- Modules contain AngularJS objects (components, directives and pipes) and also our own app's objects.
- The binding points between modules are the imports and exports.
- "*In Angular a module is a mechanism to group components, directives, pipes and services that are related, in such a way that it can be combined with other modules to create an application.*"

Continue...

- Technically, a module is a class with a decorator (@NgModule) that has the information of:
  - Which components, directives and pipes belong to the module (declarations)
  - Which components do you want to start first mention in (bootstrap).
  - Which other existing modules will be used to accomplish tasks (imports)
  - Which of those components, directives and pipes are allowed to be used by other modules (exports)
  - Provide services that are going to be used by inner objects(providers)

Module syntax

```
import { NgModule } from '@angular/core';
@NgModule({
  imports: [ … ],
  declarations: [ … ],
   providers:[…],
  exports:[….],
  bootstrap: [ … ]
})
export class AppModule { }
```

Components

- A *component* is a class that controls a specific part of the screen, called the *view*.
- A component may be a single file with everything bundled in, including css and html, or can split those (usually large) sections into separate css and html files.
- The configuration object in the @Component decorator will indicate how to find each part.
- "*Angular component are a subset of directives. Unlike directives, components always have a template and only one component can be instantiated per an element in a template*"

Component Syntax

```
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = ‘app';
}
```

Bootstrapping an application

- To boot strap our module based application, we need to inform Angular which is the Root Module.
- The main.ts is the entry point script an angular would search to bootstrap the module.

Sample of main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
        enableProdMode();
}
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));
```

Understanding the File structure:-

- **app/app.component.ts –** This is where we define our root component
- **app/app.component.html --**This is the template or html page where we write presentation logic
- **app/appcomponent.css** --This is the css file where we can write formatting style
- **app/app.module.ts –** The entry Angular Module is bootstrapped
- **index.html –** This is the page the component will be rendered in
- **app/main.ts –** This the glue that combines the component and page together

Data binding

- Data binding is the connection bridge between view and the business logic (View Model) of the application.
- Data binding in Angular is the automatic synchronization between model and the view.
- When the Model changes, the view are automatically updated and vice-versa.

We can achieve totally four ways

- Interpolation
- One way binding
- Event binding
- Two way binding

DOM

{{expression}}

Interpolation

[property] = “expression”

One Way Binding

(event) = “statement”

Event Binding

[(ngModel)] = “property”

Two Way Binding

Component

Interpolation

- This is the easiest way of data binding in Angular JS. This is same as expression in Angular 1.x. In interpolation.
- we need to supply property name is the view template, enclosed in double curly braces, eg. {{name}}.
- It is used for one-way binding (Component class to View only).

Property binding

- Property binding also known as one – way data binding.
- Property binding use [] to send value from the Component to the template(View).
- The ng-bind directives is used for one-way binding in Angular 1.x.
- Angular 2 uses HTML DOM element property for one-way binding. The square brackets are used with property name for one-way data binding in Angular 2.
  - Welcome to <span [innerText]="title">
  - Enter Full Name :<input type="text" [value]="fullName">

Interpolation Vs Property binding

- String interpolation - {{expression}} - render the bound value from component's template and it converts this expression into a string.
- Property binding - [target]="expression" - does the same thing by rendering value from component to template.
- But if you want to bind the expression that is other than string (for example - boolean), then property binding is the best option.

Two – way binding

- The ng-model directives is used for two-way binding {{ }} in Angular 1.x,
- But it is replaced with [(ngModel)] in Angular 2.0. The ngModel directive is part of the build-in Angular module called "FormsModule".
- So, we must import this module in to the template module before using the ngModel directive.
  - <input [(ngModel)]="userName">
  - The User Name is {{userName}}

Event Binding

- Angular 2.0 directly uses the valid HTML DOM elements event. For example, ng-click is now replaced by (click).
- The round brackets (parentheses) are used with DOM event name for event binding in Angular 2.
- (click)
- (dblclick)
- (mouseenter)
- (mousedown)
- (keyup)
- (keydown)
- (submit) etc

Two – way binding without ngModel

- Angular 2 has feature called "**template reference variables**". With this features.
- We are able to have directive access to an element. The template reference variable is declared by preceding an identifiers with a hash/pound character (#).
- With reference template is completely self-contained and it doesn't bind to the component.
- This solution will not work unless we bind to the element's event.
  - <input #userName type="text" (keyup)="0">
  - The User Name is {{userName.value}}

Angular JS directives

- Directive allow you to attach behaviour to element in the DOM.
- There are three types of directives
  - **Component directive** : directives with a template
  - **Structural directive** : change the DOM layout by adding and removing DOM elements.
  - **Attribute directive** : change the appearance or behaviour of an element, component or another directive.

Component directives

- *Angular component are a subset of directives. Unlike directives, components always have a template and only one component can be instantiated per an element in a template.*
- The specific type of directive that allows us to utilize web component functionality encapsulated, reusable elements available through our application.

Component directive

We can create a component like so

```
@Component({
        selector:"my-component",
        tempalte:"<p>This is user defined component</p>”

})
export class MyComponent {}
```

Structure directives

- Structural directives are responsible for HTML layout.
- They shape or reshape the DOM's structure, typically by adding, removing or manipulating elements.
- An asterisk (*) precedes the directive attribute names.

Built – in structural directive

- Three of the common, built-in structural directives
  - *ngIf
  - ngSwitch, *ngSwitchCase, *ngSwitchDefault
  - *ngFor

Attribute directives

- Attribute directives changes the appearance or behaviour of an element.
- Attribute directives are surrounded by brackets which sign to Angular that the appearance or behaviour of the DOM elements within the directive may change depending on certain conditions.
- In Angular 2 are ngStyle and ngClass are the built-in attribute directives.
- In Angular 1.x there was the ng-show and ng-hide directives, which show or hide elements depending on what the given expression evaluates to by setting the display CSS property.

Sample demo

- <p [ngClass]=" 'myClass' ">Apply style binding - Using ngClass</p>
- <p [ngClass]="{classOne:cone,classTwo:ctwo}">ngClass paragraph is here!</p>
- <p [ngStyle]="{'color':colorValue,'font-size':fsize}">ngStyle paragraph is here!</p>
- <p [ngStyle]=“ styleVariable”>ngStyle paragraph is here!</p>

Custom directive

- import required modules like Directive, ElementRef from Angular 2 core library.
- Create a TypeScript class decorate the class with @directive
- Set the value of the selector property in @directive decorator function. The directive would be used, using the selector value on the elements.
- In the constructor of the class, inject ElementRef.
-

Angular pipe

- In Angular 2, pipe (filter) is use to format the data.
- A pipe takes in data as input and transforms it to a desired output.
- So In Angular we use the | pipe character to format our data.
  - Currency
  - Date
  - Decimal
  - Json
  - lowercase and uppercase
  - percent

Custom pipe

- A custom pipe is created using a Pipe decorator on a class with a name property.
- The value of the name is used as a template expression while calling custom pipe, and a class must implement PipeTransform which is inside @angular/core.
- It is an interface which has transform method, which takes the values that has to be piped

Lifecycle Hooks

- Angular calls lifecycle hook methods on directives and components as it **creates**, **changes**, and **destroys** them.

Creates:

- `OnInit`
- `AfterContentInit`
- `AfterViewInit`

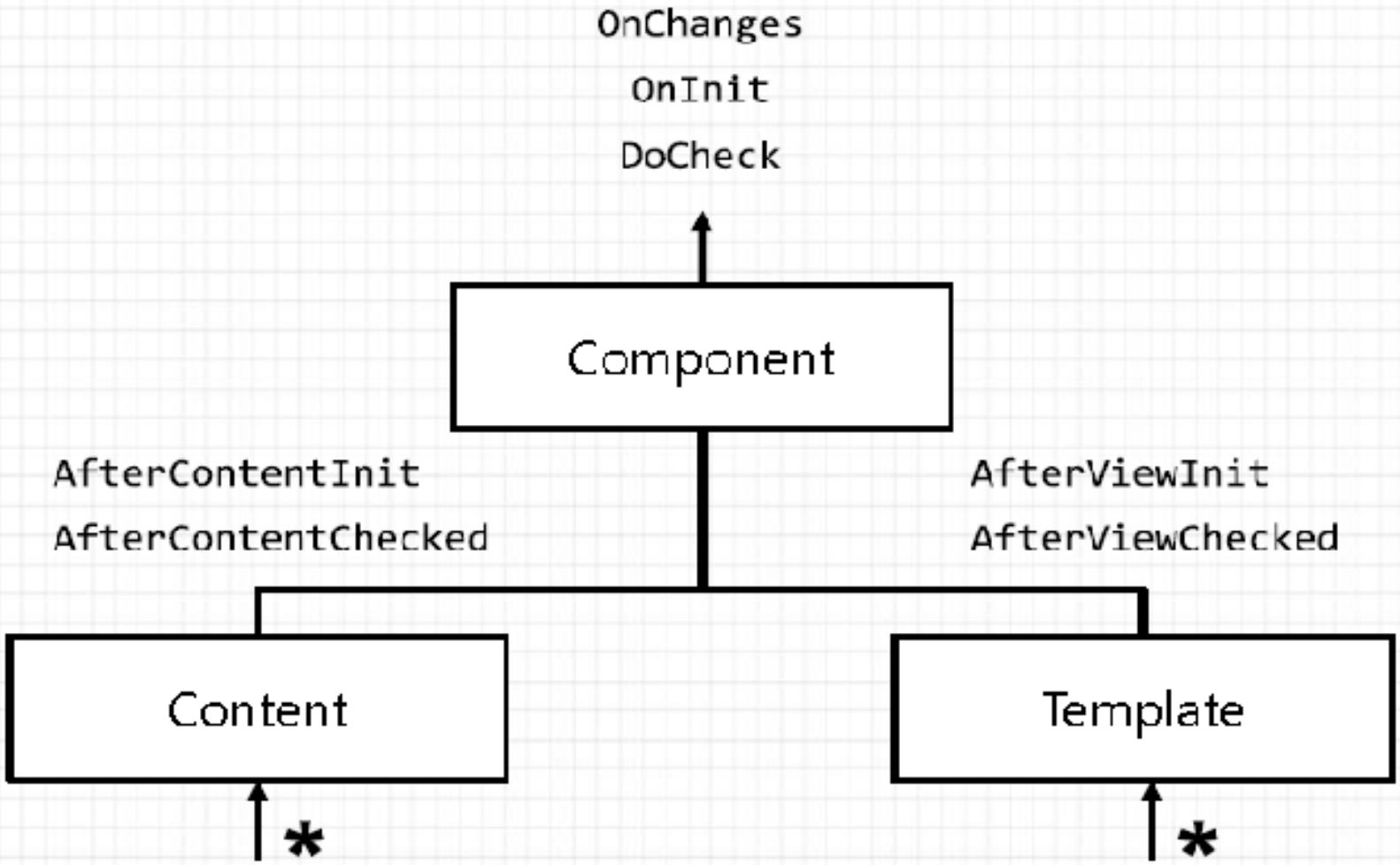
Changes:

- `DoCheck`
- `OnChanges`
- `AfterContentChecked`
- `AfterViewChecked`

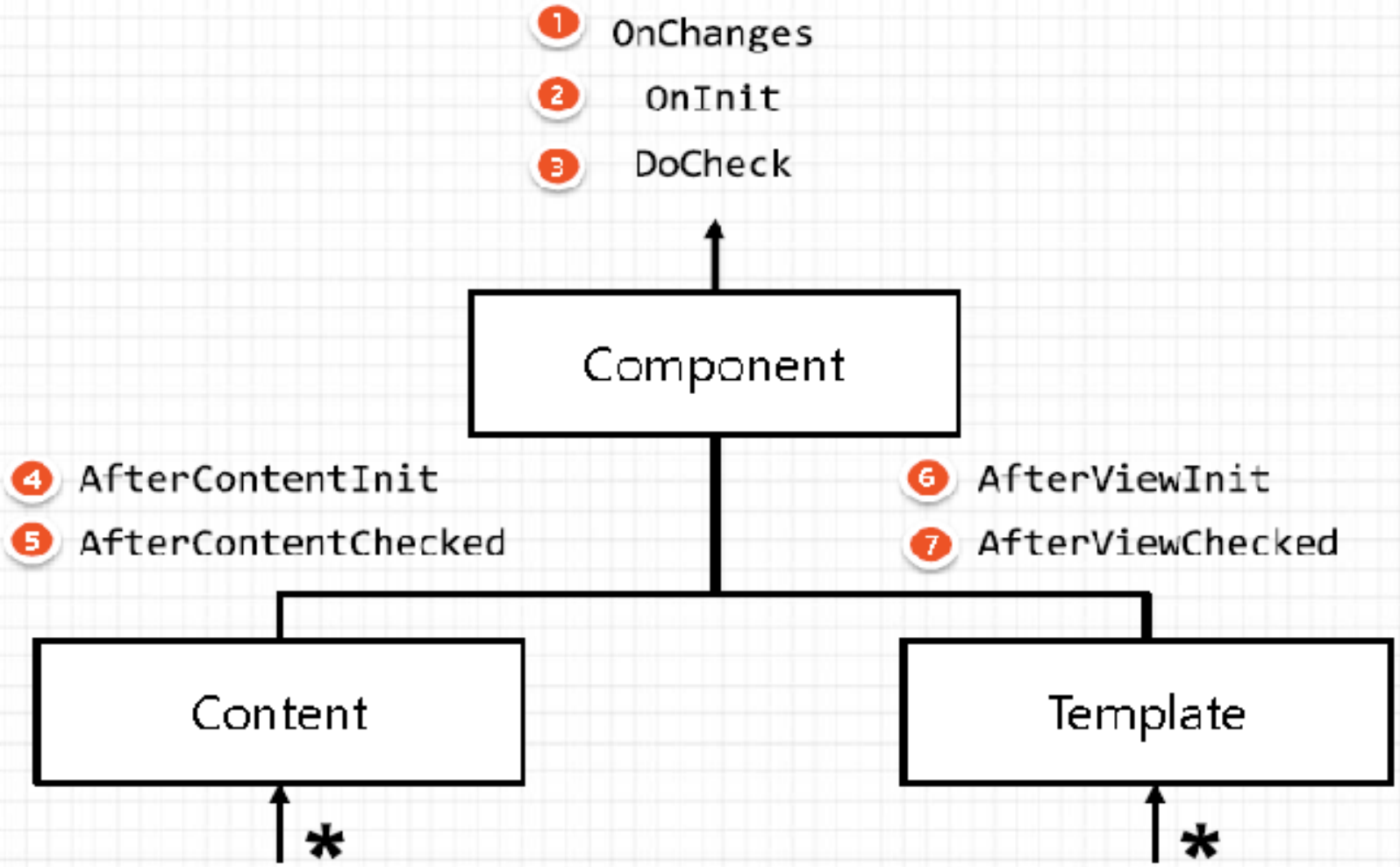
Destroys:

- `OnDestroy`

Hooks Order


OnChanges
OnInit
DoCheck
Component
AfterContentInit
AfterContentChecked
AfterViewInit
AfterViewChecked
Content
Template
*
*

Hooks Order


1 OnChanges
2 OnInit
3 DoCheck
Component
4 AfterContentInit
5 AfterContentChecked
6 AfterViewInit
7 AfterViewChecked
Content
Template
*
*

Hooks Order

Hooks	Descriptions
ngOnChanges	Called when an input or output binding value changes
ngOnInit	After the first ngOnChanges
ngDoCheck	Developer's custom change detection
ngAfterContentInit	After component content initialized
ngAfterContentChecked	After every check of component content
ngAfterViewInit	After component's view(s) are initialized
ngAfterViewChecked	After every check of a component's view(s)
ngOnDestroy	Just before the directive is destroyed

Inputs & Outputs

- Input or @Input and Output or @Output are help to communicate two components.
- Input property or @Input decorator is use to pass the value from Parent component to child component.
- Output property or @Output decorator is use to pass the value from child component to parent component with the help of event emitters.

Angular 2 forms

- Angular totally provided two types of forms
  - Template Driven Forms
  - Model Driven or Reactive approach Forms

TDF Vs MDF

- Easy to use
- Suitable for simple scenarios and fails for complex scenarios
- Two way data binding(using [(NgModel)] syntax)
- Minimal component code

- More flexible, but needs a lot of practice
- Handles any complex scenarios
- No data binding is done
- More component code and less HTML markup

Template driven form

- When we want to use the template driven form first we have to use the FormsModule in module(AppModule).
- The we have to use the **ngForm Directives.**
- It provides use information about the current state of the form state of the form including.
  - A JSON representation of the form value.
  - Validity state of the entire form.

ngFormDirective

```
<form #formRef="ngForm"
(submit)="checkValue(formRef.value)">


</form>
```

ngModel directive

- In order to register form controls on an ngForm instance we use the ngModel directive.
- In combination with a name attribute, ngModel creates a form control abstraction for use behind the scenes.
- Every form controls that is registered with ngModel will automatically shows up in form.value

ngForm with ngModel

```
<form #formRef="ngForm"
(submit)="checkValue(formRef.value)">
   <input type=“text” name=“uname” ngModel>
</form>
```

ngModelGroup

```
<div ngModelGroup="FullName">
        FirstName:<input type="text"
        name="fname"ngModel>
        LastName:<input type="text"
        name="lname"ngModel>
</div>
```

Angular pre-defined validation classes

State	Class if true	Class if false
Control has been visited	ng-touched	ng-untouched
Controls value has changed	ng-dirty	ng-pristine
Control value is valid	ng-valid	ng-invalid

componentRefName.errors.required

componentRefName.errors.minlength

componentRefName.errors.maxlength

ngForm with ngModel with Validation

```
<form #formObj="ngForm“
(ngSubmit)="verify(formObj.value)“ nonvalidate>
<input type="text" name="user" ngModel required
#userRef="ngModel” minlength="2" />
<div *ngIf="userRef.errors && (userRef.dirty ||
userRef.touched)">
            <span [hidden]="!userRef.errors.required">
             UserName is required
            </span>
            <span [hidden]="!userRef.errors.minlength">
             UserName must be 2 character
             </span>
        </div>
<input type="submit" [disabled]="!formObj.valid">
```

Model Driven Forms

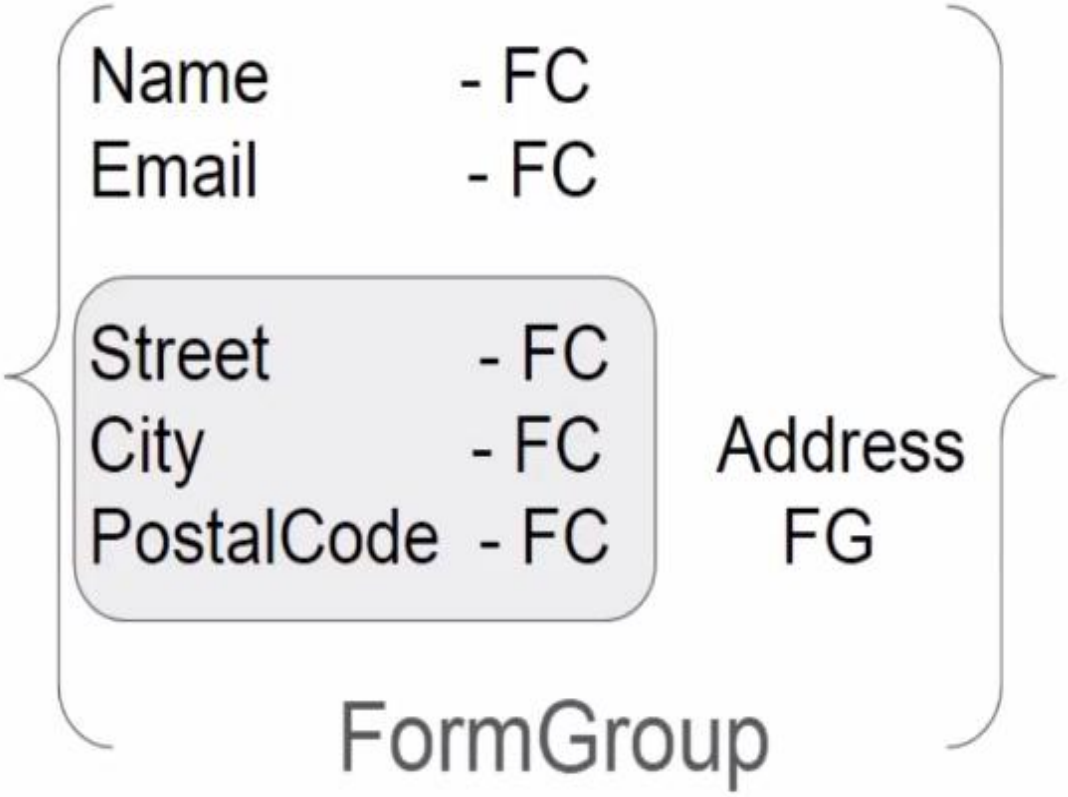
- Model Driven Forms also known as Reactive forms.
- In this approach, we create new instances of the form controls and form control group in our component.
- When we want to use the model driven form first we have to use the ReactiveFormsModule in module(AppModule).

FormGroup and FormGroupControl

- **FormControl :** The class is present in the "@angular/forms" package of Angular 2.0.Each component in form like textfield, radiobutton, checkbox is known as FormControl
- **FormGroup :** The class is present in the "@angular/forms" package of Angular 2.0. It is used to represent a set of form control inside its constructor.

Model Driven Forms

1) FormControl
2) FormGroup



In Component

```
userForm=new FormGroup({
uname:new FormControl(),
pname:new FormControl(),
        addressForm=new FormGroup({
        .....
        .....
        })
});
```

In Template

```
<form [fromGroup]="userForm">
<input type="text" formControlName="uname">
    ........
    ........
    ........
</form>
```

Model Driven form with Validation

- Import the Validators from the '@angular/forms".
- Which provide set of field which help to do the validation.

```
userForm=new FormGroup({
    uname:new
    FormControl('',[Validators.required,Validators.minLength(3)]),
    pname:new
    FormControl('',[Validators.required,Validators.maxLength(6)]),
    });
```

```
<form [formGroup]="loginForm" (submit)="verify()" nonvalidate>
<input type="text" formControlName="user" required/>
 div *ngIf="!loginForm.controls.user?.valid &&
(loginForm.controls.user?.dirty
            ||  loginForm.controls.user?.touched)">
              <div
[hidden]="!loginForm.controls.user.errors.required">
               UserName is required
              </div>
              <div
[hidden]="!loginForm.controls.user.errors.minlength">
               Min Length must be 2 character
              </div>
            </div>
<input type="submit" value="submit" [disabled]="!loginForm.valid">
```

Angular Service

- Services allow for greater separation of concerns for your application and better modularity by allowing you to extract common functionality out of component.
- Service is used when a common functionality need to be provided for various modules.
- Angular also comes with its own dependency injection framework for resolving dependencies, so you can have your services depend on other services through out your application, and dependency injection will resolve your dependencies for you.

Dependency Injection

- DI is a coding pattern in which a class receives its dependencies from external sources rather than creating them itself.

Types of Service in Angular 2

- User-defined service
- Pre-defined service

Steps to create the Service

- Import the injectable member
- Add the @injectable Decorator
- Export Service class

Register the Service

- We can register the service in two ways
  - In Component
  - In Module

In Component

- There are four simple steps to use/import Service in the Component
  - import the Service to the component
  - Add it as a provider
  - Include it through Dependency injection
  - Use the Service function

In Module

- There are four simple steps to use/import Service in the Module
  - import the Service to the module
  - Add it as a provider
  - Include it through Dependency injection in Component
  - Use the Service function

Angular 2 http service

- One of the most common scenario in any application is client interacting with the server.
- Http is the widely used protocol for this interaction.
- One can fetch data from the server, update data, create data and delete data using HTTP protocol.
- In Angular 2 http service return the Observable but Angular 1.x is http service return the Promise object.

JavaScript callback function

- A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.
- Callback function may be synchronous and asynchronous functions.

Synchronous and Asynchronous

- In *synchronous* programs, if you have two lines of code (L1 followed by L2), then L2 cannot begin running until L1 has finished executing.
- In *asynchronous* programs, you can have two lines of code (L1 followed by L2), where L1 schedules some task to be run in the future, but L2 runs before that task completes.

What is promise

- Promise is a one type of callback function, which represents the eventual result of an operation. You can use a promise to specify what to do when an operation eventually succeeds or fails.

The Promise

- The Promise is an object, with two functions.
  - then() => Promise, please get the value from that URL.
  - error() => Promise, please call this function when you have a new error for me.

Observer pattern

- The **observer pattern** is a software design pattern in which an object, called the **subject**, maintains a list of its dependents, called **observers**, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems.

Observables in Angular

- You can think of an observable as an array whose items arrive asynchronously over time.
- **Observables help you manage asynchronous data**, such as data coming from a backend service. Observables are used within Angular itself, including AngularJS event system and its http client service. To use observables, Angular uses a third-party library called Reactive Extensions (**RxJS**). Observables are a proposed feature for ES 2016, the next version of JavaScript.

The Observer

- The Observer is an object, with three functions.
  - next() => Observable, please call this function when you have a new value for me.
  - error() => Observable, please call this function when you have a new error for me.
  - complete() => Observable, please call this function when you complete your job.

Observable Vs Promise

- **Observables**
  - Observables handle multiple values over time.
  - Observable are cancellable.
-
- **Promise**
  - Promise are only called once and will return a single value.
  - Promises are not cancellable.

What is RxJS

- RxJS or *Reactive Extensions for JavaScript* is a library for transforming, composing, and querying streams of data.
- We mean all kinds of data too, from simple arrays of values, to series of events
- RxJS can be used both in the browser or in the server-side using Node.js.
  - ***Asynchronous***, in JavaScript means we can call a function and register a *callback* to be notified when results are available.
  - ***Data***, raw information in the form of JavaScript data types as: Number, String, Objects (Arrays, Sets, Maps).
  - ***Streams***, sequences of data made available over time.

Demo Example

```
var Rx=require("rx");
var source = Rx.Observable.range(1,5);
var sub = source.subscribe(
        x=>console.log(x),
        e=>console.log(e),
        ()=>console.log("completed")
)
```

http service

- **http:-** It is for callback, used for performing XMLHttpRequest operations like get(), post(), delete(), put(),
- All the Http methods return Observable, from Observable we use an operator map() to manipulate the data, the data would be in a Response which you get once you call subscribe() method
  - map(res => response.text())
  - map(res => response.json())
- The map would get the response once the subscribe() is called however the response must be assigned to a class property such that it can be displayed finally

Routing

- Routing helps to direct to different pages based on the options user select, using Routing you can render components to the users based on the options they select.
- So the browser doesn't load the whole page instead it only loads the component which we selected at runtime;

Steps to create the routing

- Create the .ts routing file (app.routing.ts)
- Import the NgModule from @angular/core, Routes and RouterModule from @angular/router' and all component which want to route at runtime.
- Now we have to create the variable of type Routes. Which contains path and component property based upon the number of component want to route.

Syntax

```
const routes:Routes=[
    {path:”,redirectTo=“\first”,pathMatch=“full”},
    {path:'first',component:FirstComponent},
    {path:'second',component:SecondComponent},
    {path:'third',component:ThirdComponent,
       children:[
     {path:‘thirdone',component:ThirdOneComponent},
     {path:‘thirdtwo,component:ThirdTwoComponent,
       ]}
 ]
```

Continue...

- Then create the class with @ngModule and use the two properties as import and export.
- With following syntax
  - imports:[RouterModule.forRoot(routes)],
  - exports:[RouterModule]

In template

```
<div>
<a routerLink="['/first']">First Component</a>
<a routerLink="['/second']">Second Component</a>
<a routerLink="['/third']">Third Component</a>
</div>

<div>
        <router-outlet></router-outlet>
</div>
```

Deploy the project in external server

- Once we created the application from Angular-CLI, we get all the configurations in the project itself for developing an app whose size will be almost 160MB or more.
- but when we deploy these applications on the server we need the build files which can process the requests from the client whose size will be less than 1MB, to build the applications and deploy in Tomcat server or any web server.
  - ng build --prod --build-optimizer