**What is Angular?**

Angular 4 is a JavaScript framework for develop dynamic web applications. It is not replacement of angular js , it was rewritten from scratch using Typescript.

Angular 4 is component based. Controllers and $scope are no longer used.

Angular 4 uses Typescript for development, where typescript not understands by browser so tsc convert into JavaScript while compilation and JavaScript render in browser for dynamic app.

It has extra features of mobile driven application development, support multiple browsers

**Extra:**

The digest cycle from Angular 1.x has been replaced by another internal mechanism known as “**Change Detection**”.

**What is component?**

It is class along with component decorator. It is building blocks of angular application i.e every angular app is parts of components and modules. It should contain template.

Component decorator contains metadata to the class which includes selector, template, templateUrl, styaleUrl. So that component knows what those classes mean and how they should work.

Angular app should have root component and root module. Root component should mention in bootstrap section of module metadata.

**What are component decorators in Angular 4?**

The main objectives of decorators are to add some metadata to the class. so that component/directive knows what those classes mean and how they should work.

**What is Modules?**

Module is a class along with ngmodule decorator. Ngmodule decorator should have metadata like declarations, import, providers, and bootstrap. Angular app should have root module which is bootstrapped in main.ts file. It just like a startup module for app.

Every application can have only one root module whereas, it can have one or more feature modules.

**What is directive? What are all the types of Directives?**

Directives extend html attributes it has some logic (html and JavaScript) which dynamically includes in DOM element. it enhance the functionality of component or directives,

There are three types of directives in Angular. They are **attribute directives**, **structural directives**, and **components**.

* **Structural directives** change the DOM layout by adding and removing DOM elements. For ex: \*ngIf and \*ngFor, ngswitch
* **Attribute directives** change the appearance or behavior of an element. . For ex: \*ngStyle and \*ngClass
* **Components** are basically directives with a template.

### name the building blocks of Angular:

The Angular application is made using the following:

Modules

Component

Template

Directives

Data Binding

Services

Dependency Injection

Routing

pipe

**What is Data Binding? How many ways it can be done?**

In order to connect application data with the DOM (Data Object Model), data binding is used. It happens between the templates (HTML) and component (Typescript). There are 3 ways to achieve data binding:

Data-binding is a feature in angular, which provides a way to communicate between the component(Model) and its view(HTML template).

1. Event Binding – used for user input events like click change ..
2. Property Binding – used to bind application data into the HTML
3. Two-way Binding – Changes made in the application state gets automatically reflected in the view and vice-versa. The ngModel directive is used for achieving this type of data binding.

**What is Pipe?**

Pipe is used for format the data it is applied in collections or angular data in html, by using pipe we need not to communicate with server for sort or filter data. We can do it with pipes.

Here it is pipe is a class along with pipe decorator should have metadata like **name**.

It is similar to filters in Angular Js.

Some of Predefined pipes are uppercase, lowercase, date, currency, json, number.

We can also create custom pipes in angular using concept called pipes.

A pure pipe is only called when Angular detects a change in the value or the parameters passed to a pipe.

@Pipe({  
name: 'filterPipe',   
pure: true   
})  
export class FilterPipe {}

An impure pipe is called for every change detection cycle no matter whether the value or parameter(s) changes.which leads to bad performance. thats why it is not recommneded to use pipes for filtering data.

@Pipe({  
name: 'filterPipe',   
pure: false  
})  
export class FilterPipe

### What is an AsyncPipe in Angular?

When an observable or promise returns something, we use a temporary property to hold the content. Later, we bind the same content to the template. With the usage of AsyncPipe, the promise or observable can be directly used in a template and a temporary property is not required.

[AsyncPipe with promises](https://codecraft.tv/courses/angular/pipes/async-pipe/#_asyncpipe_with_promises):

<p class="card-text">{{ promiseData }}</p>

<p class="card-text" ngNonBindable>{{ promiseData }}</p>

class AsyncPipeComponent {

promiseData: string;

constructor() {

this.getPromise().then(v => this.promiseData = v);

}

getPromise() {

return new Promise((resolve, reject) => {

setTimeout(() => resolve("Promise complete!"), 3000);

});

}

}

<p class="card-text" ngNonBindable>{{ promise }}</p>

<p class="card-text">{{ promise | async }}</p>

class AsyncPipeComponent {

promise: Promise<string>;

constructor() {

this.promise = this.getPromise();

}

getPromise() {

return new Promise((resolve, reject) => {

setTimeout(() => resolve("Promise complete!"), 3000);

});

}

}

[AsyncPipe with Observables](https://codecraft.tv/courses/angular/pipes/async-pipe/#_asyncpipe_with_promises):

<p class="card-text" ngNonBindable>{{ observable | async }}

<p class="card-text">{{ observable | async }}</p>

class AsyncPipeComponent {

observable: Observable<number>;

constructor() {

this.observable = this.getObservable();

}

getObservable() {

return Observable

.interval(1000)

.take(10)

.map((v) => v\*v)

}

}

**Forms in angular?**

Template driven forms:

* With a template driven form, most of the work is done in the template;
* Easy to use
* Suitable for simple scenarios and fails for complex scenarios, Custom validations are required
* Similar to AngularJS
* Two way data binding(using [(NgModel)] syntax)
* Minimal component code
* Unit testing is another challenge

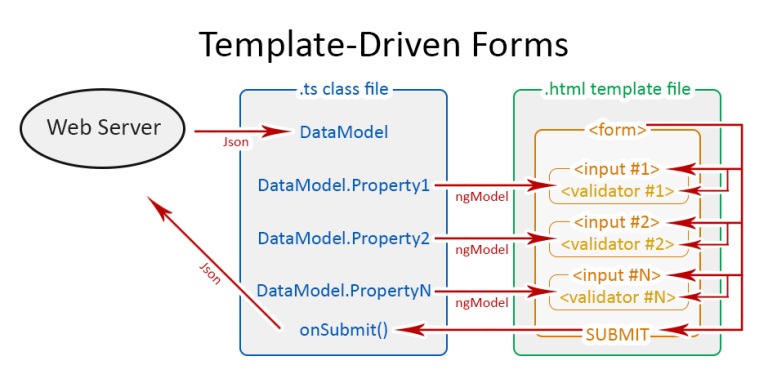
<form #myForm="ngForm" (ngSubmit) = "submitForm()">  
  
  <label>Name:</label>  
  <input type="text" [(ngModel)]="user.name" name="name" #nameField="ngModel" required />  
  
  <label>Age:</label>  
  <input type="text" [(ngModel)]="user.age"  name="age" #ageField="ngModel"  />  
  
  <button type="submit" [disabled]="!myForm.valid">Submit</button>  
</form>

<p>Name: {{user.name}}</p>

<p>Age: {{user.age}}</p>  
  
<p style="color:red" [hidden] = "nameField.valid || !nameField.touched">Your name is required!</p>

Or





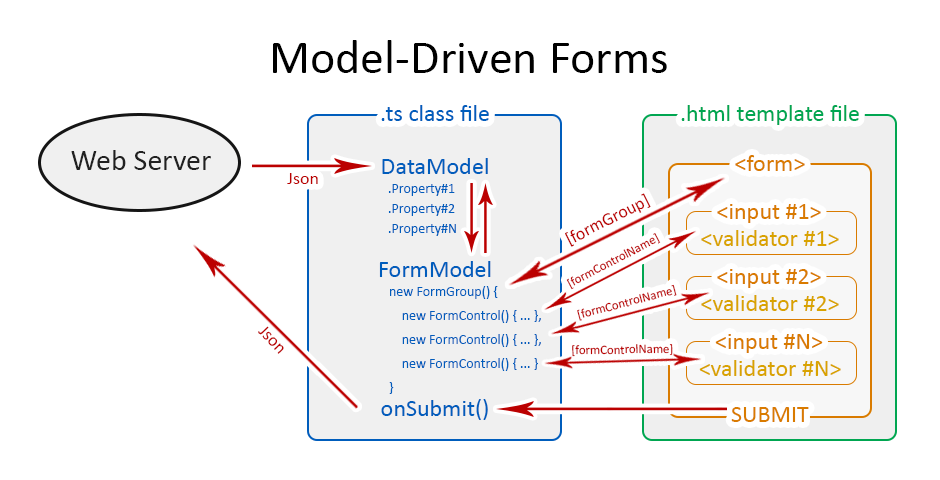
Reactive (model driven) forms:

* with the model driven form, most of the work is done in the component class
* More flexible, but needs a lot of practice
* Handles any complex scenarios
* No data binding is done (immutable data model preferred by most developers)
* More component code and less HTML markup
* Easier unit testing

|  |  |
| --- | --- |
|  | <section class="sample-app-content"> |
|  | <h1>Model-based Form Example:</h1> |
|  | <form [formGroup]="form" (ngSubmit)="onSubmit()"> |
|  | <p> |
|  | <label>First Name:</label> |
|  | <input type="text" formControlName="firstName"> |
|  | </p> |
|  | <p> |
|  | <label>Password:</label> |
|  | <input type="password" formControlName="password"> |
|  | </p> |
|  | <p> |
|  | <button type="submit" [disabled]="!form.valid">Submit</button> |
|  | </p> |
|  | </form> |
|  | </section> |
|  |  |

import{FormGroup,FormControl,Validators,FormBuilder} from from '@angular/forms';

|  |
| --- |
|  |
|  |  |
|  |  |
|  | @Component({ |
|  | selector: "model-driven-form", |
|  | templateUrl: 'model-driven-form.html' |
|  | }) |
|  | export class ModelDrivenForm { |
|  | form: FormGroup; |
|  |  |
|  | firstName = new FormControl("", Validators.required); |
|  |  |
|  | constructor(fb: FormBuilder) { |
|  | this.form = fb.group({ |
|  | "firstName": this.firstName, |
|  | "password":["", Validators.required] |
|  | }); |
|  | } |
|  | onSubmitModelBased() { |
|  | console.log("model-based form submitted"); |
|  | console.log(this.form); |
|  | } |
|  | } |



<https://www.stackchief.com/tutorials/Angular%20Form%20Examples:%20Template%20Driven%20vs%20Reactive%20Forms>

<https://www.ryadel.com/en/angular-forms-template-driven-model-driven-reactive-pros-cons-tutorial-guide/>

**Services in angular?**

These are re-usable Pease of code , which can be used by multiple components, directives

We can create service with class and along with injectable decorator , in this decorator should have metadata like **providerIn.**

We can inject services in provider metadata section of modules. And we can access in components by using constructor parameter.

Services can communicate with server through http services to get data from server.

And update or insert data back to server.

The main objective of a service is to share data, functions with different components of an Angular application.

import { [Injectable](https://angular.io/api/core/Injectable) } from '@angular/core';

@[Injectable](https://angular.io/api/core/Injectable)({

providedIn: 'root',

})

export class HeroService {

constructor() { }

}

@[Component](https://angular.io/api/core/Component)({

selector: 'app-hero-list',

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

providers: [ HeroService ]

})

Ways to consume services:

If we want to share *one* instance of a service across the *entirety* of our application we configure it on our NgModule.

If we want to have *one* instance of a service *per* component, and shared with all the component’s children, we configure it on the providers property on our component decorator.

If we want to have *one* instance of a service *per* component, and shared with *only* the component’s view children and *not* the component’s content children, we configure it on the viewProviders property of our component decorator.

https://codecraft.tv/courses/angular/dependency-injection-and-providers/ngmodule-providers-vs-component-providers-vs-component-viewproviders/

https://itnext.io/understanding-provider-scope-in-angular-4c2589de5bc

**Routes in Angular?**

**What is the process called by which Typescript code is converted into JavaScript code?**  
**Answer**: It is called Transpiring. Even though Typescript is used for writing code in Angular applications, it gets internally transpired into equivalent JavaScript.

**What is the sequence of Angular Lifecycle Hooks?**

OnChange()  -  OnInit()  -  DoCheck()  -  AfterContentInit()  -  AfterContentChecked()  -  AfterViewInit()  -  AfterViewChecked()  -  OnDestroy().

### Differentiate between Observables and Promises.

Observable is a stream in which passing of zero or more events is possible and the callback is called for each event. Whereas, promise handles a single event.

Observables are lazy, which means nothing happens until a subscription is made. Whereas Promises are eager; which means as soon as a promise is created, the execution takes place.

|  |  |
| --- | --- |
| Promise | Observable |
| Emits a single value | Emits multiple values over a period of time |
| Not Lazy | Lazy. An observable is not called until we subscribe to the observable |
| Cannot be cancelled | Can be cancelled by using the unsubscribe() method |
|  | Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc. |

**What is Promise and $q Service in angular Js:**

‘Promises’ is nothing but result of some operation/action is completed, these promises either success or failuer, accourding to theses we have to perform action.

‘Deferred’ is used to control how and when those promise logics will execute.

We can think about promises as “WHAT” we want to fire after an operation is completed while deferred controls “WHEN” and “HOW” those promises will execute.

“$q” is the angular service which provides promises and deferred functionality.

A new instance of defer is created by calling $q.defer(). It has mainly three methods.

var defer = $q.defer();

defer.resolve(value); resolve(value) – This method is use to resolve the derive promise

defer.reject(value);This method is use to reject the derived promise

defer.notify(value) – This method is used to notify the current state of the derived promise

### What does a Subscribe method do in Angular 4?

It is a method which is subscribed to an observable. Whenever the subscribe method is called, an independent execution of the observable happens.

Observables are result of some operation/action is completed, these Observables either success or failure, according to theses we have to perform action.

Subscribe is used to control how and when those Observables logics will execute.

In Observable passing of zero or more events is possible and the callback is called for each event. Whereas promise handles a single event.

**What are Template reference variables?**

A template reference variable (#var) is a reference to a DOM element within a template. We use hash symbol (#) to declare a reference variable in a template.

we can access any property of the inputDOM, using this reference variable. For example, we can get the **value** of the input element as name.value and the value of the **placeholder** property by name.placeholder

**What is new in Angular 6?**

* Angular Elements – It allows converting Angular components into web components and embeds the same in some non-Angular application
* rxJS 6 – Angular 6 makes use of RxJS 6 internally
* Instead of using <template>, we now need to use <ng-tempalate>.

**What is View Encapsulation and how many ways are there do to do it in Angular?**  
**Answer**: To put simply, View Encapsulation determines whether the styles defined in a particular component will affect the entire application or not. Angular supports 3 types of ViewEncapsulation:

* Emulated – Styles used in other HTML spread to the component
* Native – Styles used in other HTML doesn’t spread to the component
* None – Styles defined in a component are visible to all components of the application

encapsulation: ViewEncapsulation.None,

@Component({

selector: 'app-root',

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

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

encapsulation: ViewEncapsulation.None

})

### Explain Authentication and Authorization.

Authentication: The user login credentials are passed to an authenticate API (on the server). On the server side validation of the credentials happens and a JSON Web Token (JWT) is returned. JWT is a JSON object that has some information or attributes about the current user.  Once the JWT is given to the client, the client or the user will be identified with that JWT.

Authorization: After logging in successfully, the authenticated or genuine user does not have access to everything. The user is not authorized to access someone else’s data; he/she is authorized to access some data.

### What is AOT Compilation?

In JIT compilation, the application compiles inside the browser during runtime.  
Whereas in the AOT compilation, the application compiles during the build time.

Angular offers two ways to compile your application:

* Just-in-Time (JIT), which compiles your app in the browser at runtime. This was the default until Angular 8.
* Ahead-of-Time (AOT), which compiles your app and libraries at build time. This is the default since Angular 9.
* with JIT it runs every time for every user at runtime
* With AOT, the compiler runs once at build time

Features:

Faster rendering

Detect template errors earlier

Better security

Fewer asynchronous requests

*Smaller Angular framework download size*

### 27. What is Redux?

It is a library which helps us maintain the state of the application. Redux is not required in applications that are simple with the simple data flow, it is used in Single Page Applications that have complex data flow.

### Which of the Angular life cycle component execution happens when a data-bound input value updates?

ngOnChanges is the life cycle hook that gets executed whenever a change happens to the data that was bound to an input.