Angular Framework

Angular is a Javascript framework which is used to develop single page application.

Angular uses 2 technologies to create SPA

* Typescript: Superset of Javascript
* HTML

Drawbacks of Javascript

1. Javascript is not type safe
2. Javascript is not compiled to find the early errors
3. Javascript results are not reliable as the data is not type safe

Angular Tool kit

This is provided by Google who is the owner of Angular, it helps you to

* Easily create angular projects
* Compile typescript automatically
* Launch application on live server
* Commands to quickly develop angular elements like components, services, pipes and so on

Toolkit name is @angular/cli which you need to install in your machine

* You need node.js software to install angular/cli because it gives you npm[node package manager] software that installs javascript libraries / toolkits over the internet

How to install toolkit

* npm install -g @angular/cli [or]
* npm install @angular/cli

Verifying angular installation

>> ng version [or] ng --version [or] ng -v

Component:

These are reusable UI which you can independently create & use in other components, by default angular gives root component whose name is AppComponent, this will be there in the index.html by default (we don’t touch index.html file)

Creating components in angular

ng generate component componentName [or]

ng g c componentName

The above command generates 4 files

1. component.ts: Component class code
2. component.html: Component content
3. component.css: Component style
4. component.spec.ts : Testing code will be written here

Databinding in Angular

Data sharing between component class(TS) and view template(HTML) is what Databinding is, below are the types of databinding

1. Interpolation: Component to View
2. Event Binding: View to Component
3. Property Binding: Component to View
4. Two way Binding: Bidirectional, means you can share data from component to view and vice versa

Expressions of databinding:

1. Interpolation: {{ property }}
2. Event Binding: ( eventName )
3. Property Binding: [ property ]
4. Two-way binding: [( ngModel )]

Two-way binding is useful when you want to sync the data between view and the component at the same time, it is mainly useful at the time form validations

Two-way binding uses ngModel directive which is by default wouldn’t be part of the application, you must add FormsModule in the AppModule to use ngModel in the application, else you will get error

Other types of binding ( these are very rarely used )

1. Style Binding
2. Class Binding

Both are related to CSS

Adding Bootstrap.js in Angular

1. Install Bootstrap in node\_modules of your project

npm install bootstrap

1. Import the bootstrap css file downloaded in node\_modules in the global css file of angular project i.e., src/styles.css

@import url(“location of bootstrap.css file”)

Angular Directives

It deals with DOM structure, we have mainly 2 directives

1. \*ngFor: It is to iterate over the arrays
2. \*ngIf: It is to apply condition on the component property while displaying DOM elements like hiding / showing DOM elements.

Angular Pipes

It is used to transform the output, angular provides inbuilt pipes like

uppercase, lowercase, date, currency, json

{{ username | uppercase }}

{{ dob | date }} // default format will be yyyy-MM-dd

{{ dob | date: ‘dd-MM-yyyy’ } // format will be 01-12-1993

{{ salary | currency }} // default format will be $value i.e., $ 1000

{{ salary | currency : ‘INR’ }} // format will be RupeeSymbol 1000

{{ obj | json }} // converts Javascript object to JSON

Custom Pipes:

It helps you to create pipes as per your requirement

Every inbuilt pipe is decorated with @Pipe on top of some class & it implements an interface PipeTransform

Ex: date pipe has a class with @Pipe and implements PipeTransform

@Pipe({   
 name: ‘date’  
})  
export class DateFormatter implements PipeTransform {   
 transform(value, option) {   
 if(option == ‘dd-MM-yyyy’) {  
 return value.day-value.month-value.year  
 }   
 // other conditions  
 }  
}  
{{ dob | date : ‘dd-MM-yyyy’ }}

We want to format the name of the user

HTML code will be {{ user.name | nameFormat: user.gender}}

@Pipe( {   
 name : ‘nameFormat’  
} )  
export class NameFormatter implements PipeTransform {   
 transform(value, option) {   
 if(option == ‘Male’) return Mr. value;  
 else return Ms. value  
 }  
}

Note: we have a command to create custom pipe using ‘ng g p pipe’

transform(value, …args): It accepts the value that applies the pipe & …args means it can take multiple options like 0 or more options

Component Interaction

This helps you to share data between one component to another component, i.e., parent to child and child to parent data sharing

Angular gives two decorators to share the data between the components

1. @Input(): Used to share data from parent to child
2. @Output(): Used to share data from child to parent

@Input

Parent Component : <child [itemName] = “username”>

Here username is a property in parent component & itemName is a property in a child component

Child Component: Uses @Input() on top of the property that receives value  
@Input()   
itemName

@Output:

It is achieved like an event binding way

Parent Component: <child (itemCounter) = “counter = $event”>

Here itemCounter is a property of child component and counter is a property of parent component, $event is the value that is generated by the child component when an event occurs in the itemCounter

Child Component: Uses @Output() on top of the property that generates value by event

@Output()  
itemCounter : EventEmitter<number> = new EventEmitter<number>();

itemCounter.emit(25), emit() function can generate a value 25 that will be in the form $event, instead of 25 it can emit any other values

Angular Forms

1. Template Driven Form: Simple to create uses two way databinding features to pass data from view to component & vice versa
2. Model Driven Form: More widely used form, uses some inbuilt objects to pass data from view to component & vice versa without using two way databinding

Note: We need to add FormsModule for two way databinding, similarly we need to add ReactiveFormsModule for Model Driven Form, both the modules added in AppModule imports

Template Driven Form

It uses ngForm & ngModel directives of FormsModule to handle the form data

Steps:

1. Add FormsModule to the AppModule
2. Generate a component
   1. Create properties & functions that you need to access form HTML
   2. Use ngForm to create an object that stores form data
   3. Use ngModel that initializes the ngForm object

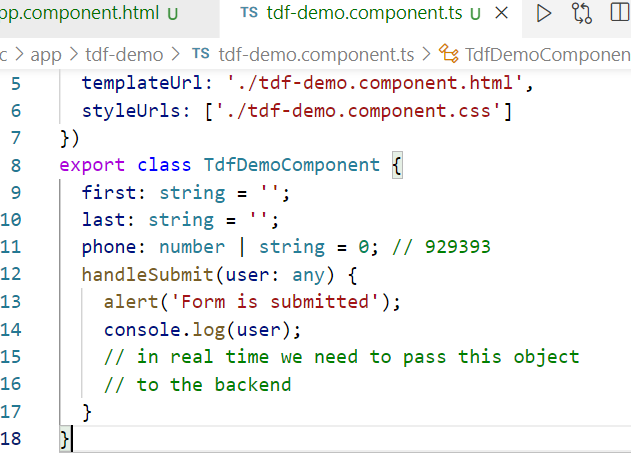
app.module.ts



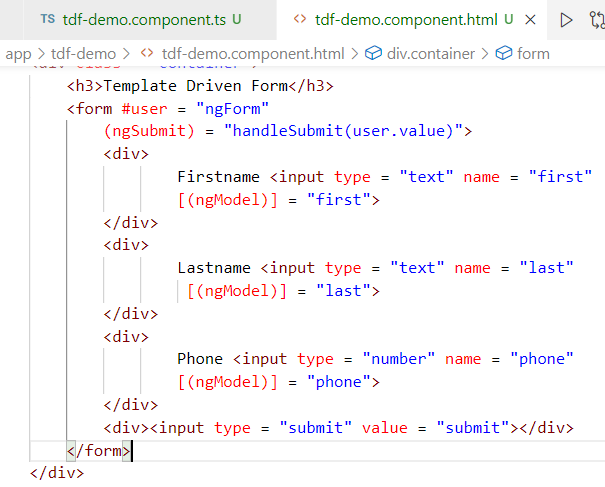
Generate component:

# ng g c tdf-demo

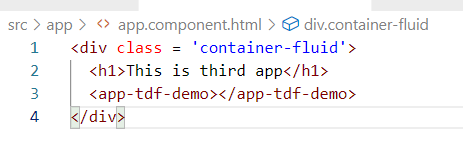
tdf-demo.component.ts



tdf-demo.component.html



app.component.html



Reactive Forms/Model Driven Forms

It uses an inbuilt object called FormBuilder to handle the form data, this FormBuilder object is automatically created by Angular when you use ReactiveFormsModule in the AppModule

Steps

1. Add ReactiveFormsModule in the AppModule.imports
2. Initialize FormBuilder in the component constructor

constructor(public builder : FormBuilder) { }

1. Create an object that maintains form data using group() function of FormBuilder

user = this.builder.group( { first: [ ], last: [ ], phone: [ ] } );

Here the properties like first, last & phone gets automatically updates from the view via [formGroup] & formControlName

Note: Without two-way data-binding i.e., [(ngModel)] FormBuilder keeps data between view & component in sync

# ng g c mdf-demo

Add the ReactiveFormsModule in the AppModule.imports

ReactiveFormsModule is in @angular/forms

Drawback of Template Driven Form over Model Driven Form

1. TDF will have less control over the form compare to MDF because form controls are created in HTML, whereas in MDF form controls are created in TS i.e., FormBuilder.group( { } )
2. TDF uses [(ngModel)], ngForm to handle Form Data, but in MDF we can handle Form Data without using [(ngModel)] and ngForm, because of same FormBuilder.group( { } ) which itself take care of two way data binding
3. Auto-populating forms or clearing forms at runtime need more code in TDF, but in MDF its very simple,

ex: this.user.controls[‘first’].setValue(‘Guest’); // sets first name in both view and component class

ex: this.user.reset({}); // clears the form in view & component class

1. Form Validation is done in the code in case of MDF and in case of TDF its done in the view template

Form Validations:

You can prevent user to submit some invalid input to the component using validations, there are many validations you can perform on the input like required, maxLength, minLength, pattern and etc

user = this.builder.group( {   
 first: [“”, Validators.compose([ Validators.required,   
 Validators.minLength(3) ])],  
 last : [“”, Validators.compose([Validators.required])],

phone: [“”]  
} )

Form Control Properties

These help in understanding form control properties like value, valid, invalid, pristine

user.controls[‘first’].value >> returns value of the control

user.controls[‘first’].valid >> returns true or false

user.controls[‘first’].pristine >> returns false when control is modified and true when control is not modified

Angular Services

These are reusable classes which can be accessed by multiple components, they can have common business logics so that they are reusable to multiple components

In Angular you have a command to create a service class

# ng g s service-name

The above command creates a class with @Injectable()  
i.e.,  
@Injectable()  
export class ClassName { }

Because of @Injectable angular creates the object automatically in any place you need this object you must use this classname as a parameter in the constructor

ex: If UserService is the class having @Injectable then TDFComponent can use constructor(public service: UserService){ }

Things we do in service class

1. In Service class we will have functions that can access backend API’s
2. Use HttpClient object that help you to send request to the backend API
3. Making 4 types of requests from HttpClient to make 4 types of operations which are called CRUD operations
   1. C (Create): Storing - POST
   2. R (Retrieve): Fetching - GET
   3. U (Update): Updating - PUT
   4. D (Delete): Deleting - DELETE

Note: If browser is showing the JSON data from an URL it means the URL must be accessed using HTTP GET

Observables:

This is an object & is the return type of all the HTTP methods that maintains the response we get from the request made by HTTP methods

i.e., get(url) : Observable  
 post(url): Observable  
 put(url): Observable  
 delete(url): Observable

Note: HttpClient object we will not create, angular only creates that we must only import one class called HttpClientModule in the AppModule.imports

Note: Service class object also we will not create, angular creates that object also because of the decorator it has like @Injectable()

Steps to access Real / Fake Data

1. Add HttpClientModule in the imports of AppModule
2. Create service # ng g s user
3. Use HttpClient in the service to send request
4. Use service object in the component to get the data from the Observable (an object that maintains response), it is done using one method called subscribe()