**Angular Questions:**

1. **Data bindings in Angular**

**Answer:**

Data binding deals with how to bind your data from component to HTML DOM elements (Templates).

* **One-way data binding:**  is a one-way interaction between component and its template. If you perform any changes in your component, then it will reflect the HTML elements. **For example**: String interpolation, Event binding, Property binding, Attribute binding, Class binding, style binding
* **Two-way data binding:** is a two-way interaction, data flows in both ways (from component to views and views to component). Simple example is **ngModel**.

**NgModel** is a standalone directive. **ngModel** directive binds form control to property and property to form control. <input type="text" **[(ngModel)]**="model.name" />

1. **Does Typescript support multiple inheritance ?**

**Answer: No, Typescript is OOP & it’s have feature of interface.**

1. **How to consume WebAPI in angular application?**

**Answer: using HttpClient modules**

1. **What is modules in angular 2? NgModule? What are properties of it?**

Answer: Name the three Module Arrays

Modules break your applications into logical boundaries. They are, in alphabetical order:

Bootstrap array - Tells Angular which components to load so the functionality can be accessed

Export array - Sends out components, directives, and pipes to be used in other modules.

Import array - Brings in functionality from other Angular modules

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

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

import { AppComponent } from './app.component';

@NgModule({

imports: [ BrowserModule ],

declarations: [ AppComponent ],

bootstrap: [ AppComponent ]

})

export class AppModule { }

Let's go through each line of the code in detail.

The import statement is used to import functionality from the existing modules. Thus, the first 3 statements are used to import the NgModule, BrowserModule and AppComponent modules into this module.

The NgModule decorator is used to later on define the imports, declarations, and bootstrapping options.

The BrowserModule is required by default for any web based angular application.

The bootstrap option tells Angular which Component to bootstrap in the application.

1. **What are the chief components of Angular 2?**

The main components are:

**Component**- Brings the modules together

**Metadata**- Adds more data to the Angular JS class

**Modules**- Breaks up the application into logical pieces of code, each module performing a single task

**Service**- Creates a component that can be shared across the whole application

Templates- Defines an Angular JS application’s views

1. **Explain routing in Angular2+?**

Routing aids in directing users to different pages depending on what option they chose on the main page.

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

const appRoutes: Routes = [

  { path: 'list', component: ListEmployeesComponent },

  { path: 'create', component: CreateEmployeeComponent },

  { path: '', redirectTo: '/list', pathMatch: 'full' }

];

@NgModule({

 imports: [

RouterModule.forRoot(appRoutes)

]

<router-outlet></router-outlet>

1. **What is routeroutlet why we use it?**
2. **Define the role of a decorator in Angular 2.**

Decorators identify an object type or class that has been created by the TypeScript as an Angular component. The decorators provide additional metadata like componentName, templateUrl, StyleUrl etc. that dictates how the component should be processed and used when it’s run.

1. **Main advantages of Angular 2+ ?**

Answer: Angular 2’s main advantages are:

Mobile-Friendly: Angular 2 was created with the mobile app industry in mind.

Browser Compatibility: Angular 2 supports Internet Explorer 9, 10, 11, Firefox, Chrome, Safari, Android 4.1, as well as Microsoft Edge.

Cross-Platform Flexibility: When using Angular 2, developers can design applications that can run on desktop systems, Android-based devices, iOS-based devices, etc.

High Performance: Angular2 uses a highly optimized superset of JavaScript that makes both the app and website load faster. Additionally, Angular2’s new component router results in faster loads.

Code Generation: Templates are turned into code that's highly optimized for today's JavaScript virtual machines. This gives the developer all of the benefits of hand-written code, but with a framework’s productivity.

Cost-Effective: And of course, Angular is open source, so the price is right!

1. **What are Angular 2 directives? Explain with examples.**

Directives are classes that add additional behavior to elements in your Angular applications

The different types of Angular directives are as follows:

1. [Components](https://angular.io/guide/component-overview)—directives with a template. This type of directive is the most common directive type.
2. [Attribute directives](https://angular.io/guide/built-in-directives" \l "built-in-attribute-directives)—directives that change the appearance or behavior of an element, component, or another directive. Build-in directive are ngModel, ngStyle, ngClass.

Custom attribute directive:

@Directive({

selector: '[appHighlight]'

})

<p [appHighlight]>Highlight me!</p>

1. [Structural directives](https://angular.io/guide/built-in-directives" \l "built-in-structural-directives)—directives that change the DOM layout by adding and removing DOM elements.

Angular's built-in structural directives, such as \*ngIf, \*ngFor, and \*ngSwitch.

1. **What are the different ways to disabled textboxes in reactive form in angular2+ ?**

Answer:

HTML: <input formControlName="firstName" type="text"  [attr.disabled]="true" / >

Typescript:

1. Setup the formControl with disabled state:

this.employeeForm = this.formBuilder.group({

firstName: [{ value: '', disabled: true }, Validators.required],

1. Use FormControl.enable() or FormControl.disable()

this.form.get('name').enable()

1. Create a custom directive

I will write a DisabledControlDirective. This directive is applying in combination with formControlName directive or formControl directive. When you want to disable FormControl on the template, you use [disableControl] instead of the built-in [disabled].

import { Directive, Input } from '@angular/core'

import { NgControl } from '@angular/forms'

@Directive({

selector: '([formControlName], [formControl])[disabledControl]',})export class DisabledControlDirective {

@Input() set disabledControl(state: boolean) {

const action = state ? 'disable' : 'enable'

this.ngControl.control[action]()

}

constructor(private readonly ngControl: NgControl) {}}

<form [formGroup]="form">

<input type="text" formControlName="name" **[disabledControl]**="disabledName" /></form>

1. **What’s an EventEmitter?**

Answer: It’s a class in the Angular framework that emits custom events.

1. **When ngOnInit event get called in Angular 2 Application Lifecycle?**

This is called whenever the initialization of the directive/component after Angular first displays the data-bound properties happens.

1. **What is the use of tsconfig.json & package.json file?**
2. **Explain component lifecycle hooks in Angular 2 ?**

The component lifecycle hooks overview the life cycle sequence and the interfaces. Angular manages the life cycle of a component. Angular creates it, renders it. It can also create and render its children. It also checks when its data-bound properties change. It can even destroy it before removing it from the DOM. The life cycle hook offered by angular provides the visibility into these key life moments and the ability to act when they occur. The components go through an entire set of processes or life cycle right from its initiation to the end of the application.

There are a number of lifecycle hooks which are listed below:–

ngOnChanges

ngOnInit

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

ngOnDestroy

1. **What are pipes in angular 2+ ?**

The pipes are used to transform the input data into the desired output. There are some built-in pipes provided by the Angular such as DatePipe, UpperCasePipe, LowerCasePipe, Currency Pipe, and PercentPipe. E.g. {{ birthday | date }}, {{ ‘some text here’| upperCase}}

1. **What is RouterOutlet in Angular 2?**

the RouterOutlet is a directive present in the router library to be used as a component. It marks the spot in a template for the router to display the components for that outlet.

Every outlet can have its unique name, which is determined by the optional name attribute. The name once set cannot be changed dynamically. If no value has been set, the default value is "primary".

<router-outlet></router-outlet>

<router-outlet name="left"></router-outlet>

1. **What is dependency injection in Angular?**

Dependencies are services or objects that a class needs to perform its function

Angular's DI framework provides dependencies to a class upon instantiation. You can use Angular DI to increase flexibility and modularity in your applications.

* To inject a service, you must first **create and register the injectable service**.

Example

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

**@Injectable**({ providedIn: 'root', })

export class SampleService { constructor() { } }

* **Injecting services into components**

Injecting services results in making them visible to a component.

constructor(heroService: HeroService)

1. **What is RxJS in Angular?**

In Angular, Reactive Extensions for the JavaScript (RxJS) is a reactive stream library that allows you to work with asynchronous data streams in Angular 2. RxJS can be used in the browser or on the server-side while using Node.js.

1. **How to create a component in Angular 2?**

Import the top-level component using import { Component } from '@angular/core';

Describe the component using the @ symbol. Create instances of @Directive, @Injectable, @RouterConfig etc.

Add meta-data like selector, providers, styles, template.

Export the component

@Component({

  selector: 'app-list-employees',

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

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

})

1. **Explain the steps for creating a services in Angular 2?**

Here are the steps:

Import injectable member

Add @Injectable Decorator

Export Service class

Here is the syntax:

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

@Injectable()

export class MyCustomService {

}

1. **What is the Life Cycle Event to check whether the Dom is fully loaded?**

Answer: ngAfterViewInit

1. **How many types of forms support in angular ?**

Answer:

**Reactive Forms:** Reactive forms provide a model-driven approach to handling form inputs whose values change over time.

Reactive Forms is created inside component class so it is also referred as model driven forms. Every form control will have an object in the component and this provides greater control and flexibility in the form programming. Reactive Form is based on structured data model.

FormControl − Define basic functionality of individual form control

FormGroup − Used to aggregate the values of collection form control

FormArray − Used to aggregate the values of form control into an array

ControlValueAccessor − Acts as an interface between Forms API to HTML DOM elements.

**There are three steps to using form controls:**

To add a form group to the component, take the following steps.(Grouping Form Controls)

1. Create a [FormGroup](https://angular.io/api/forms/FormGroup) instance.

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

 employeeForm: FormGroup;

ngOnInit() {

    this.employeeForm = this.formBuilder.group({

      firstName: ['', Validators.required],

      lastName: ['', Validators.required],

     });

  }

1. Associate the [FormGroup](https://angular.io/api/forms/FormGroup) model and view.

<form [formGroup]="employeeForm" (ngSubmit)="onSubmit()">

 <input id="firstName" formControlName="firstName" />

1. Save the form data.

**Template driven forms:**

Template driven forms is created using directives in the template. It is mainly used for creating a simple form application.

Configure **FormsModule** in **AppComponent** as shown below −

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

@NgModule({

imports: [

BrowserModule,

FormsModule

],

we used **ngModel** attribute in **input** text field

<form #userName="ngForm" (ngSubmit)="onClickSubmit(userName.value)">

<input type="text" name="username" placeholder="username" **ngModel**>

<input type="submit" value="submit"> </form>

<app-test></app-test>

@Component({

selector: 'app-test',

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

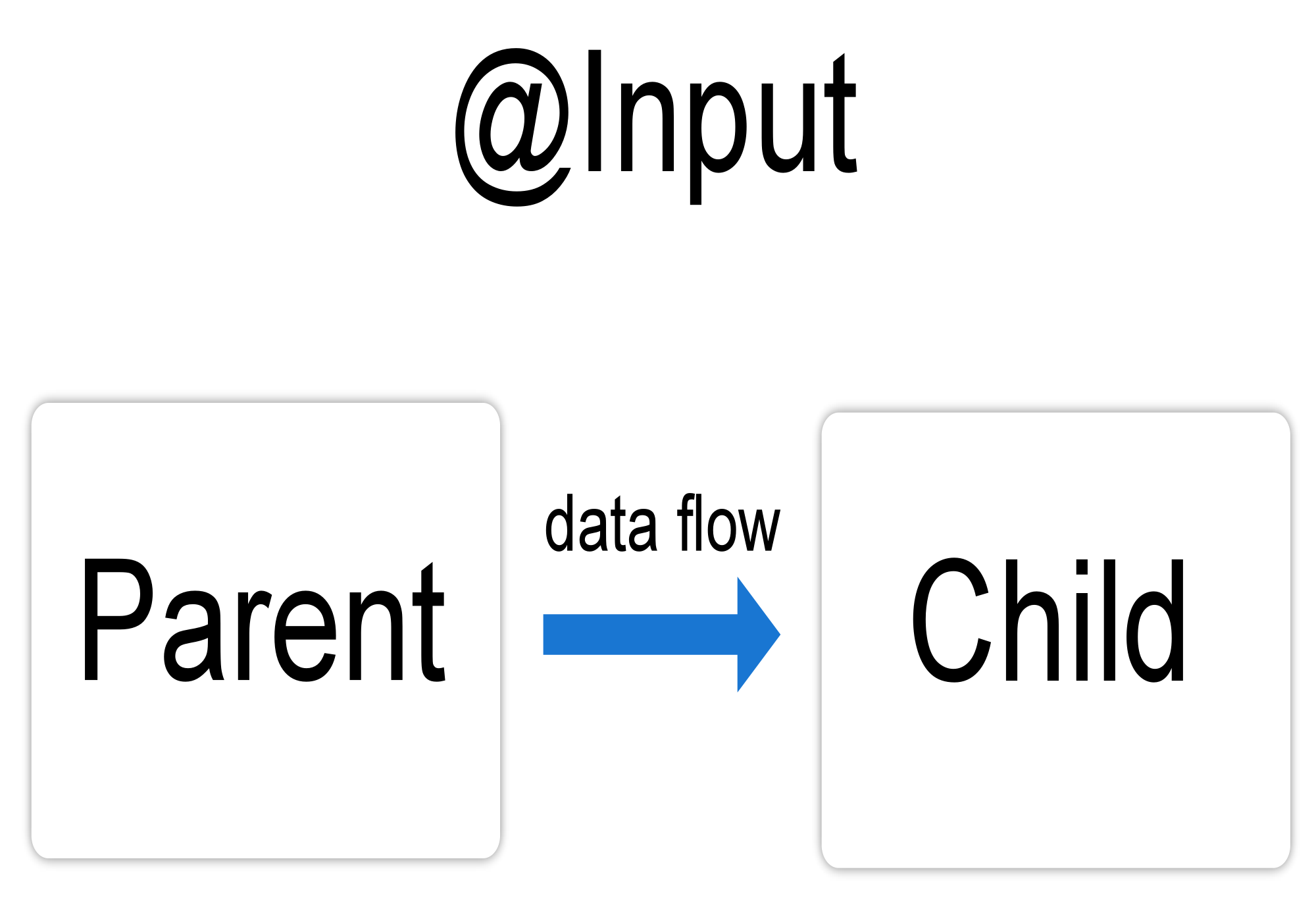
styleUrls: ['./test.component.scss']})

1. **What is @inputs In Angular 2?**

Answer:

The @Input decorator binds a property with our child component so that it can communicate and pass values from **parent to child.**

Sending data to a child component



**Configuring the child component**

To use the @Input() decorator in a child component class, first import Input and then decorate the property with @Input(), as in the following example.

import { Component, Input } from '@angular/core'; // First, import Input

export class ItemDetailComponent {

**@Input() item: string;** // decorate the property with @Input()

}

### Configuring the parent component

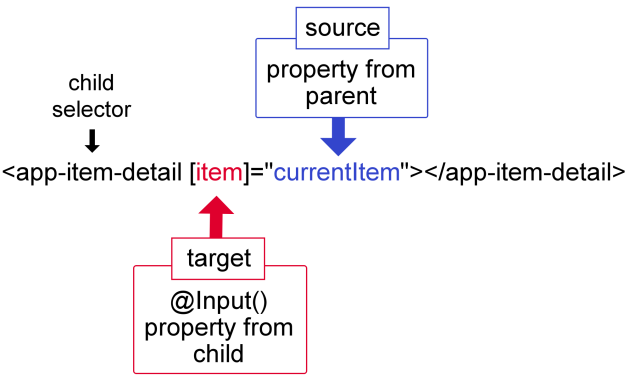
The next step is to bind the property in the parent component's template.

<app-item-detail **[item]**="currentItem"></app-item-detail>

export class AppComponent {

**currentItem** = 'Television';

}

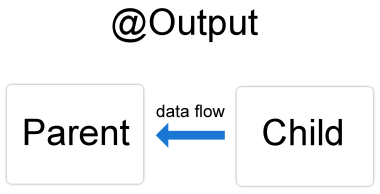


1. **What is @outputs In Angular 2?**

Answer: @Output along with EventEmitter is used to push out events in components.

The @Output decorator binds a property with our child component so that our child component can call its parent component. We can say it is used for communication between parent and child component.

The child component uses the @Output() property to raise an event to notify the parent of the change. To raise an event, an @Output() must have the type of EventEmitter, which is a class in @angular/core that you use to emit custom events.



### Configuring the child component

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

@Output() newItemEvent = new EventEmitter<string>();

export class ItemOutputComponent {

@Output() newItemEvent = new EventEmitter<string>();

addNewItem(value: string) {

this.newItemEvent.emit(value);

}

}

### Configuring the child's template

<label>Add an item: <input #newItem></label>

<button (click)="addNewItem(newItem.value)">Add to parent's list</button>

### Configuring the parent component

export class AppComponent {

items = ['item1', 'item2', 'item3', 'item4'];

addItem(newItem: string) {

this.items.push(newItem);

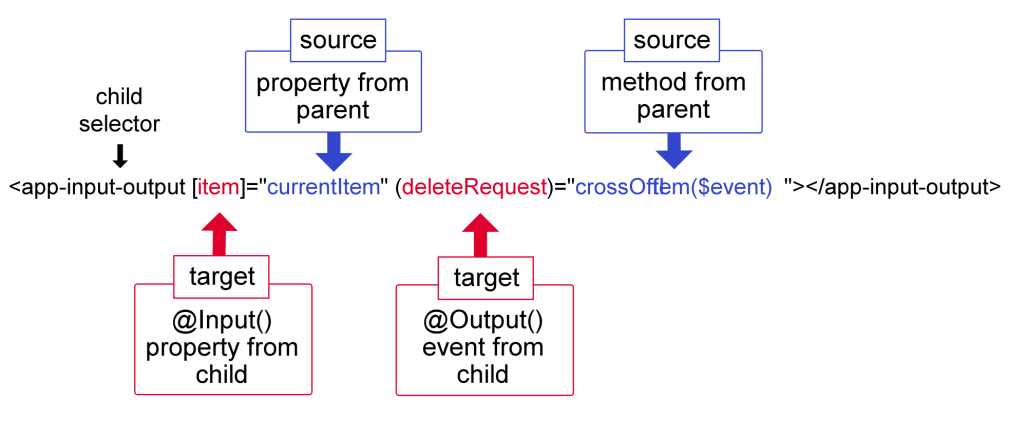
}

}

### Configuring the parent's template

<app-item-output (newItemEvent)="addItem($event)"></app-item-output>

Using @Input() and @Output() together:



1. **What is the difference between directive and component in Angular 2?**

|  | **Components** | **Directive** |
| --- | --- | --- |
| 1. | To register, use @Component meta-data annotation | To register, use @Directive meta-data annotation |
| 2. | Used to create UI widgets and break up app into smaller components | Use to design re-usable components and add behavior to existing DOM element. |
| 3. | Only one component allowed per DOM element | Many directives allowed per DOM element. |
| 4. | @View decorator is mandatory | Does not use View. |

1. **How can you handle errors in Angular 2+ application?**

The Angular 2 Applications provide with the option of error handling.The errors in Angular 2 can be handled by including the React JS catch library and later using the catch function.

The catch function, which is used after adding the catch library contains the link to the Error handler function.

And in this error, handler function, the errors are sent to the error console, and also the errors are thrown back to continue the execution.

So, whenever an error occurs it will be redirected to the error console of the web.

**Handling errors with HttpClient**

Using Angular's HttpClient along with `catchError` from RxJS, we can easily write a function to handle errors within each service. HttpClient will also conveniently parse JSON responses and return a javascript object in the observable. There are two categories of errors which need to be handled differently:

Client-side: Network problems and front-end code errors. With HttpClient, these errors return ErrorEvent instances.

Server-side: AJAX errors, user errors, back-end code errors, database errors, file system errors. With HttpClient, these errors return HTTP Error Responses.

Example:

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

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

import { Observable, throwError } from 'rxjs';

import { retry, catchError } from 'rxjs/operators';

import { User } from './user.model';

@Injectable({

providedIn: 'root'

})

export class UserService {

private apiUrl = 'https://localhost:8080/api/users';

constructor(private http: HttpClient) { }

getUsers(): Observable<User[]> {

return this.http.get<User[]>(this.apiUrl)

.pipe(

retry(1),

catchError(this.handleError)

);

}

handleError(error) {

let errorMessage = '';

if (error.error instanceof ErrorEvent) {

// client-side error

errorMessage = `Error: ${error.error.message}`;

} else {

// server-side error

errorMessage = `Error Code: ${error.status}\nMessage: ${error.message}`;

}

window.alert(errorMessage);

return throwError(errorMessage);

}

}

## **[A better solution with HttpInterceptor](https://scotch.io/bar-talk/error-handling-with-angular-6-tips-and-best-practices192" \l "toc-a-better-solution-with-httpinterceptor)**

we implement an interceptor to handle errors across our application: