Revisit Day33

1. Angular Binding (Setting/Sending/communicating with data/method)
2. Types of Binding
   1. One Way Binding
      1. Interpolation Binding {{}} connecting html data with class property
      2. Property Binding [] – html attributes connecting the class property
      3. Event Binding () – html events connecting class method
   2. Two Way Binding [()] (Banana in a Box syntax)

Connects both view to model and model to view [MVVM]

1. Services [Way of getting External data]
2. Dependency Injection [**Design Pattern**]
3. Pipes [It’s to format the output of angular data number pipes, string pipes, currency pipes]
   1. Built-in Pipes [ uppercase, lowercase, date, usd]
   2. Custom Pipes Use @Pipe decorator, and implement PipeTransform Interface – transform();
4. @NgModule – Decorator which creates an angular module (That’s the container for the components)

@NgModule({

declarations: [AppComponent],

imports: [BrowserModule],

providers: [],

bootstrap: [AppComponent]

})

Agenda

1. httpClient (It is used to send data via http methods get, post, put & delete)
2. Routing
3. RouteGuards
4. Observables (RxJS)
5. Subjects (RxJS)
6. Sub/pub Design pattern
7. EventEmitters

Magazines

NewsPaper (Published --- Subscribers)

**HttpClient**

Most front-end applications need to communicate with a server over the HTTP protocol, in order to download or upload data and accesss other back-end services. Angular provides a simplified client HTTP API for Angular applications, the *HttpClient* service class in [@angular/common/http](https://angular.io/api/common/http).

The Angular *HttpClient* offers testability features, typed request and response objects, request and response interception, Observable APIs, and streamlined error handling.

The *HttpClient* Module is already included when creating a new Angular app. We just need to register it in our Angular application. In src/app/app.module.ts file, import the *HttpClient* module to make use of the *HttpClient* service.

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

Also, include the HttpClientModule in @NgModule's imports array.

@NgModule({

imports: [

BrowserModule,

HttpClient

]

})

Now, the Angular *HttpClient* is ready to use or inject with the Angular service or component.

The *HttpClient* service is used for communication between front-end web apps and backend services. This communication is done over the HTTP protocol. The *HttpClient* service is available as an injectable class, with methods to perform HTTP requests. The Angular *HttpClient* Methods are request(), delete(), get(), patch(), post(), put(), head(), jsonp(), and options(). All HttpClient methods return an **Observable** of something. In general, an observable can return multiple values over time. An observable from *HttpClient* always emits a single value and then completes, never to emit again.

The *HttpHeaders* service is used for the header configuration options of an HTTP request. HTTP Headers let the client and the server share additional information about the HTTP request or response. For example, we use the content-type header to indicate the media type of the resource like JSON, text, blob, etc.

**Handling Errors with HttpClient**

By 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 returns an observable object.

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.

By verifying if an error is an instance of *ErrorEvent*, we can figure out which type of error we have and handle it accordingly.

To catch errors, we "pipe" the observable result from http.get() (or any *HttpClient* methods) through an RxJS catchError() operator. Also, we add the retry(1) function to the pipe to retry all requests once before failing.

**Example:**

We are going to create a fake backend server using the [json-server](https://www.npmjs.com/package/json-server) NPM module in our Angular app. This module will allow us to communicate with a server to which we can send and receive data locally.

Run the npm install -g json-servercommand to set the fake json-server globally.

In the root folder of the Angular project, create a folder by the name of backend and also create a file by the name of database.json. This file will have our fake JSON data. Add some fake data to the database.json file:

{

"employees": [

{"id" : 12 , "name" : "Chris", "age" : 22 },

{"id" : 13 , "name" : "Joseph", "age" : 25 },

{"id" : 14 , "name" : "Alex", "age" : 35 }

]

}

We are done setting up a fake JSON server in our Angular application. To start the fake JSON server, run the json-server --watch backend/database.json command in the terminal. Now, your fake json server is up and running on the port **3000**. You are able to view this employees array by visiting <http://localhost:3000/employees> on the browser. Now that our server is ready, we communicate with the server through HTTP Requests.

We create service file that allow us to handle all HTTP requests to our application. All *HttpClient* methods return an **observable** object, so we need to cast the observable object into an *Employee* type.

Before, we create the service file we need to create an interface to define the *employee* type. So, the observable object returned by the HttpClient Methods can be cast to the *Employee* type.

export interface Employee{

id : number;

name : string;

age : number;

}

Let's create a *employee.service.ts file* to handle all HTTP requests. We import the HttpClient and HttpHeaders services to make the HTTP request work. Here, we create CRUD operations using *HttpClient* methods (GET, POST, PUT, DELETE) and also there is some error handling logic in it.

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

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

import { Observable, throwError } from 'rxjs';

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

import { Employee } from './Employee';

@Injectable({providedIn: 'root'})

export class EmployeeService {

// Base url

baseurl = 'http://localhost:3000/employees/';

constructor(private http: HttpClient) { }

// Http Headers

httpOptions = {

headers: new HttpHeaders({

'Content-Type': 'application/json'

})

}

// POST

CreateEmployee(data): Observable<Employee> {

return this.http.post<Employee>(this.baseurl , JSON.stringify(data), this.httpOptions)

.pipe(

retry(1),

catchError(this.errorHandl)

)

}

// GET

GetEmployee(id): Observable<Employee> {

return this.http.get<Employee>(this.baseurl + id)

.pipe(

retry(1),

catchError(this.errorHandl)

)

}

// GET

GetEmployees(): Observable<Employee> {

return this.http.get<Employee>(this.baseurl)

.pipe(

retry(1),

catchError(this.errorHandl)

)

}

// PUT

UpdateEmployee(id, data): Observable<Employee> {

return this.http.put<Employee>(this.baseurl + id, JSON.stringify(data), this.httpOptions)

.pipe(

retry(1),

catchError(this.errorHandl)

)

}

// DELETE

DeleteEmployee(id){

return this.http.delete<Employee>(this.baseurl + id, this.httpOptions)

.pipe(

retry(1),

catchError(this.errorHandl)

)

}

// Error handling

errorHandl(error) {

let errorMessage = '';

if(error.error instanceof ErrorEvent) {

// Get client-side error

errorMessage = error.error.message;

} else {

// Get server-side error

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

}

console.log(errorMessage);

return throwError(errorMessage);

}

}

Let's make an **HTTP POST Request** to add one employee to the *employees* array in the local server using *HttpClient* service.

In the *app.component.ts* file,

export class AppComponent implements OnInit{

constructor(private employeeService : EmployeeService){}

new\_employee = {

"id" : "18",

"name" : "Grace",

"age" :"22"

};

ngOnInit(){

this.employeeService.CreateEmployee(this.new\_employee)

.subscribe(data =>{

console.log("Post Request for creating new employee");

console.log("id: " + data.id);

console.log("name: " + data.name);

console.log("age: " + data.age);

}

);

}

}

//Logs:

//Post Request for creating new employee

// id: 18

// name: Grace

// age: 22

Now let's make an **HTTP GET Request** to get a specific employee details from the *employees* array in the local server using *HttpClient* service.

In the *app.component.ts* file,

export class AppComponent implements OnInit{

constructor(private employeeService : EmployeeService){}

employee : Employee;

ngOnInit(){

this.employeeService.GetEmployee(12)

.subscribe(data =>{

console.log("GET Request to get a employee with id - 12");

console.log("name: " + data.name);

console.log("age: " + data.age);

}

);

}

}

//Logs:

//GET Request to get a employee with id - 12

// name: Chris

// age: 22

Let's make an **HTTP GET Request** to get all the employee details in the *employees* array.

In the *app.component.ts* file,

@Component({

selector: 'app-root',

template: `

<div>

<h2> Employee List</h2>

<ul \*ngFor = "let emp of employees">

<li>{{ emp.id }} - {{ emp.name }} - {{emp.age}}</li>

</ul>

</div>

`,

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

})

export class AppComponent implements OnInit{

constructor(private employeeService : EmployeeService){}

employees : Employee;

ngOnInit(){

this.employeeService.GetEmployees()

.subscribe(data =>{

this.employees = data;

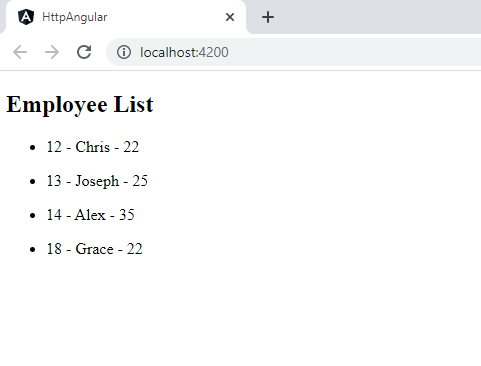
}

);

}

}

Output:



Let's make an **HTTP PUT Request** to update a specific employee details in the *employees* array.

export class AppComponent implements OnInit{

constructor(private employeeService : EmployeeService){}

employee = {

"name": "Adam",

"age": "28"

}

ngOnInit(){

this.employeeService.UpdateEmployee(18, this.employee )

.subscribe(data =>{

console.log("PUT Request to update the employee with id - 18");

console.log("updated name :" + data.name);

console.log("updated age :" + data.age);

}

);

}

}

//Logs:

//PUT Request to update the employee with id - 18

//updated name :Adam

//updated age :28

Let's make an **HTTP DELETE Request** to delete a specific employee in the *employees* array.

export class AppComponent implements OnInit{

constructor(private employeeService : EmployeeService){}

ngOnInit(){

this.employeeService.DeleteEmployee(18)

.subscribe(data =>{

console.log("DELETE Request to delete the employee with id - 18");

console.log(data);

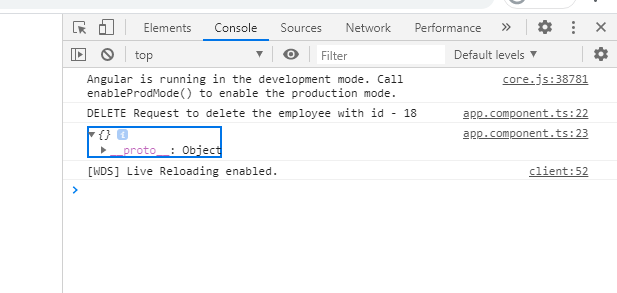
}

);

}

}

Console Logs:



We can request a specific employee's details by passing the id in the request URL. If the id in the request URL is not present in the *employee* array, it results in an server- side error. These errors can be handled by the error handler method defined in the *EmployeeService*.

//Here, we make get request for the already deleted employee record which returns HTTP error response.

ngOnInit(){

this.employeeService.GetEmployee(18)

.subscribe(data =>{

console.log("GET Request to get a employee with id - 18");

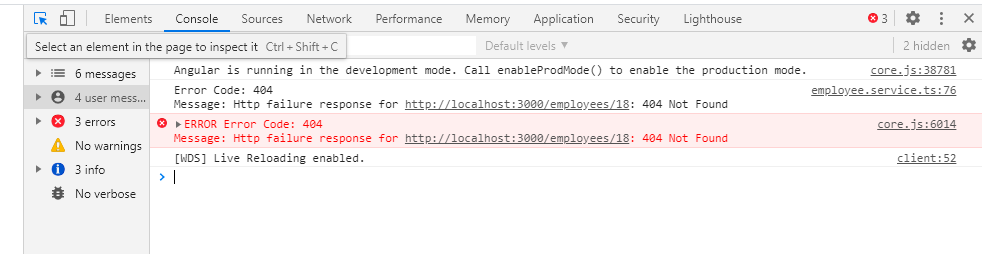
console.log(data);

}

);

}

Console Logs:



Npm I -g json-server

Data.json

{

"employees": [

{

"id": 12,

"name": "Chris123",

"age": 27

},

{

"id": 13,

"name": "Joseph",

"age": 25

},

{

"id": 14,

"name": "Alex",

"age": 35

},

{

"id": 15,

"name": "test",

"age": 35

},

{

"name": "test",

"age": 35,

"id": 16

},

{

"name": "siva",

"age": 35,

"id": 17

},

{

"name": "sample",

"age": 27,

"id": 18

}

]

}

json-server –-watch data.json

<http://localhost:3000/employees>

Synchronous vs Asynchronous

Synch = Serial Operation (do step1, wait till done, then go to step 2, run step2) blocking operation --

ASynch = parallel operation(do step1 & don’t wait go to step2 and run it also)

Non-blocking operation (Ajax) – XmlHttpRequest – (Send the request in the background – It will perform partial UI update)

Index.html Services.html

RxJS – Reactive JS (To perform Asynchronous operations)

Observables [publish & subscribe model]

Subjects

