**Angular HttpClient**

HttpClient is Angular's mechanism for communicating with a remote server over HTTP. In this chapter, we'll cover how to use HttpClient in Angular to perform various HTTP operations.

**Setup**

First, make sure you have HttpClientModule imported in your app.config.ts file:

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

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

export const appConfig: ApplicationConfig = {

providers: [

provideHttpClient()

]

};

**Basic GET Request**

Let's start with a simple GET request to fetch data from an API.

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

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

interface Post {

userId: number;

id: number;

title: string;

body: string;

}

@Component({

selector: 'app-root',

template: `

<h1>Posts</h1>

<ul>

<li \*ngFor="let post of posts">{{ post.title }}</li>

</ul>

`,

standalone: true,

imports: [NgFor]

})

export class AppComponent implements OnInit {

posts: Post[] = [];

constructor(private http: HttpClient) {}

ngOnInit() {

this.http.get<Post[]>('https://jsonplaceholder.typicode.com/posts')

.subscribe(

(data) => {

this.posts = data;

},

(error) => {

console.error('Error fetching posts:', error);

}

);

}

}

Expected output:

Posts

- sunt aut facere repellat provident occaecati excepturi optio reprehenderit

- qui est esse

- ea molestias quasi exercitationem repellat qui ipsa sit aut

...

**POST Request**

Now, let's look at how to make a POST request to create a new resource.

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

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

interface Post {

userId: number;

id?: number;

title: string;

body: string;

}

@Component({

selector: 'app-root',

template: `

<h1>Create Post</h1>

<form (ngSubmit)="onSubmit()">

<input [(ngModel)]="newPost.title" name="title" placeholder="Title">

<textarea [(ngModel)]="newPost.body" name="body" placeholder="Body"></textarea>

<button type="submit">Create</button>

</form>

<p \*ngIf="createdPost">Created post with ID: {{ createdPost.id }}</p>

`,

standalone: true,

imports: [FormsModule]

})

export class AppComponent {

newPost: Post = { userId: 1, title: '', body: '' };

createdPost: Post | null = null;

constructor(private http: HttpClient) {}

onSubmit() {

this.http.post<Post>('https://jsonplaceholder.typicode.com/posts', this.newPost)

.subscribe(

(data) => {

this.createdPost = data;

console.log('Post created:', data);

},

(error) => {

console.error('Error creating post:', error);

}

);

}

}

Expected output (in console after form submission):

Post created: {userId: 1, id: 101, title: "New Post", body: "This is a new post."}

**PUT Request**

Let's update an existing post using a PUT request.

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

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

interface Post {

userId: number;

id: number;

title: string;

body: string;

}

@Component({

selector: 'app-root',

template: `

<h1>Update Post</h1>

<form (ngSubmit)="onSubmit()">

<input [(ngModel)]="updatedPost.title" name="title" placeholder="Title">

<textarea [(ngModel)]="updatedPost.body" name="body" placeholder="Body"></textarea>

<button type="submit">Update</button>

</form>

<p \*ngIf="response">Updated post: {{ response | json }}</p>

`,

standalone: true,

imports: [FormsModule, JsonPipe]

})

export class AppComponent {

updatedPost: Post = { userId: 1, id: 1, title: '', body: '' };

response: any;

constructor(private http: HttpClient) {}

onSubmit() {

this.http.put<Post>('https://jsonplaceholder.typicode.com/posts/1', this.updatedPost)

.subscribe(

(data) => {

this.response = data;

console.log('Post updated:', data);

},

(error) => {

console.error('Error updating post:', error);

}

);

}

}

Expected output (in console after form submission):

Post updated: {userId: 1, id: 1, title: "Updated Title", body: "This is the updated body."}

**DELETE Request**

Finally, let's delete a post using a DELETE request.

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

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

@Component({

selector: 'app-root',

template: `

<h1>Delete Post</h1>

<button (click)="deletePost()">Delete Post 1</button>

<p \*ngIf="isDeleted">Post deleted successfully!</p>

`,

standalone: true

})

export class AppComponent {

isDeleted = false;

constructor(private http: HttpClient) {}

deletePost() {

this.http.delete('https://jsonplaceholder.typicode.com/posts/1')

.subscribe(

() => {

this.isDeleted = true;

console.log('Post deleted successfully');

},

(error) => {

console.error('Error deleting post:', error);

}

);

}

}

Expected output (in console after button click):

Post deleted successfully

**Error Handling**

It's important to handle errors properly. Here's an example using catchError from RxJS:

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

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

import { catchError } from 'rxjs/operators';

import { throwError } from 'rxjs';

@Component({

selector: 'app-root',

template: `

<h1>Error Handling</h1>

<button (click)="fetchData()">Fetch Data</button>

<p \*ngIf="error">{{ error }}</p>

`,

standalone: true

})

export class AppComponent {

error: string | null = null;

constructor(private http: HttpClient) {}

fetchData() {

this.http.get('https://invalid-url.com/data')

.pipe(

catchError(this.handleError)

)

.subscribe(

(data) => console.log(data),

(error) => this.error = error

);

}

private handleError(error: HttpErrorResponse) {

if (error.error instanceof ErrorEvent) {

// Client-side or network error

console.error('An error occurred:', error.error.message);

} else {

// Backend returned an unsuccessful response code

console.error(

`Backend returned code ${error.status}, ` +

`body was: ${error.error}`);

}

// Return an observable with a user-facing error message

return throwError('Something bad happened; please try again later.');

}

}

Expected output (after clicking the button):

Something bad happened; please try again later.

**Conclusion**

This chapter covered the basics of using HttpClient in Angular for making HTTP requests. We've seen how to perform GET, POST, PUT, and DELETE operations, as well as how to handle errors. Remember to always handle errors and use strong typing with your HTTP requests for better code quality and developer experience.