**Chapter 10: HTTP & API Communication**

**🌐 What is HttpClient?**

The HttpClient service in Angular lets your app communicate with RESTful APIs over HTTP. You can:

* Fetch data from a backend
* Send form submissions
* Handle errors
* Secure requests with interceptors (e.g., for auth tokens)

**🧰 Step 1: Setup HttpClient in Angular CLI 19+**

In Angular 17+, use the **standalone setup** with provideHttpClient.

**✅ Modify main.ts**

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

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

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

bootstrapApplication(AppComponent, {

providers: [provideHttpClient()]

});

No need to import HttpClientModule in a module — it's all configured here.

**✉️ Step 2: Create a Service for API Requests**

**CLI command:**

ng generate service services/product

**✅ product.service.ts**

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

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

import { Observable } from 'rxjs';

export interface Product {

id: number;

name: string;

price: number;

}

@Injectable({ providedIn: 'root' })

export class ProductService {

private apiUrl = 'https://api.example.com/products';

constructor(private http: HttpClient) {}

getProducts(): Observable<Product[]> {

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

}

getProduct(id: number): Observable<Product> {

return this.http.get<Product>(`${this.apiUrl}/${id}`);

}

addProduct(product: Product): Observable<Product> {

return this.http.post<Product>(this.apiUrl, product);

}

updateProduct(product: Product): Observable<Product> {

return this.http.put<Product>(`${this.apiUrl}/${product.id}`, product);

}

deleteProduct(id: number): Observable<void> {

return this.http.delete<void>(`${this.apiUrl}/${id}`);

}

}

**📦 Step 3: Display Products in a Component**

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

import { ProductService, Product } from './services/product.service';

@Component({

standalone: true,

selector: 'app-product-list',

template: `

<h2>Products</h2>

<ul \*ngIf="products.length > 0; else loading">

<li \*ngFor="let p of products">

{{ p.name }} - ₹{{ p.price }}

</li>

</ul>

<ng-template #loading>Loading products...</ng-template>

`,

providers: [],

imports: []

})

export class ProductListComponent implements OnInit {

products: Product[] = [];

constructor(private productService: ProductService) {}

ngOnInit(): void {

this.productService.getProducts().subscribe(data => {

this.products = data;

});

}

}

**❌ Step 4: Error Handling**

**Handle errors using catchError from RxJS (we’ll explore RxJS deeper in the next chapter).**

import { catchError } from 'rxjs/operators';

import { of, throwError } from 'rxjs';

getProducts(): Observable<Product[]> {

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

catchError(error => {

console.error('Error fetching products', error);

return of([]); // return fallback data

})

);

}

**🔐 Step 5: Using HTTP Interceptors**

**What is an Interceptor?**

An HTTP interceptor modifies **all outgoing or incoming HTTP traffic**. Great for:

* Adding auth tokens
* Logging
* Global error handling

**✅ Example: Auth Token Interceptor**

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

import {

HttpInterceptorFn,

HttpRequest,

HttpHandlerFn

} from '@angular/common/http';

export const authInterceptor: HttpInterceptorFn = (req: HttpRequest<any>, next: HttpHandlerFn) => {

const authReq = req.clone({

setHeaders: {

Authorization: `Bearer YOUR\_TOKEN\_HERE`

}

});

return next(authReq);

};

**✅ Register Interceptor in main.ts**

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

import { authInterceptor } from './services/auth.interceptor';

bootstrapApplication(AppComponent, {

providers: [provideHttpClient(withInterceptors([authInterceptor]))]

});

**🧪 Practice Scenario: Add Product Form**

You’re building an admin dashboard with a form to **add new products via API**.

**✅ Add Form (template-driven or reactive):**

addProduct() {

const product = { name: this.name, price: this.price };

this.productService.addProduct(product).subscribe({

next: (p) => console.log('Product added', p),

error: (err) => console.error('Error adding product', err)

});

}

**🧠 Summary Table**

| **HTTP Method** | **Angular Method** | **Use Case** |
| --- | --- | --- |
| GET | http.get() | Fetch data |
| POST | http.post() | Add new item |
| PUT | http.put() | Update existing item |
| DELETE | http.delete() | Remove item |

**✅ Exercises**

1. Create a CustomerService to GET/POST customers.
2. Add error handling with catchError and fallback logic.
3. Use an HTTP interceptor to add a fake auth token.
4. Build a UI that displays, adds, and deletes products via API.