**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 app.config.ts**

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

export const appConfig: ApplicationConfig = {

  providers: [provideHttpClient()]

};

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

**Step 2: Create a Product Interface**

ng generate interface product

**✅ product.ts**

export interface Product {

    id:number;

    name:string;

    price:number;

}

**Step 3: Create a Service for API Requests**

ng generate service product

**✅ product.service.ts**

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

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

import { Observable } from 'rxjs';

import { Product } from './product';

@Injectable({

  providedIn: 'root'

})

export class ProductService {

  private apiUrl = 'http://localhost:8080/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 4: Create a Product List Component**

**✅ product-list.component.ts**

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

import { Product } from '../product';

import { ProductService } from '../product.service';

import { CommonModule } from '@angular/common';

import { RouterModule } from '@angular/router';

@Component({

  selector: 'app-product-list',

  imports: [CommonModule, RouterModule],

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

  styles: ``

})

export class ProductListComponent {

     products: Product[] = [];

  constructor(private productService: ProductService) {}

  ngOnInit(): void {

    this.loadProducts();

  }

  loadProducts(): void {

    this.productService.getProducts().subscribe((data) => this.products = data);

  }

  deleteProduct(id: number): void {

    this.productService.deleteProduct(id).subscribe(() => this.loadProducts());

  }

}

**✅ product-list.component.html**

<h2>Products</h2>

<button routerLink="/products/new">Add Product</button>

<ul>

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

    {{ product.name }} - {{ product.price | currency }}

    <button [routerLink]="['/products', product.id, 'edit']">Edit</button>

    <button (click)="deleteProduct(product.id)">Delete</button>

  </li>

</ul>

**Step 5: Create a Product Form Component**

Both Add/Edit Form included

**✅ product-form.component.ts**

import { CommonModule } from '@angular/common';

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

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

import { RouterModule, Router, ActivatedRoute } from '@angular/router';

import { Product } from '../product';

import { ProductService } from '../product.service';

@Component({

  selector: 'app-product-form',

  imports: [CommonModule, ReactiveFormsModule, RouterModule],

  templateUrl: './product-form.component.html',

  styles: ``

})

export class ProductFormComponent {

  form: FormGroup;

  isEditMode = false;

  productId: number | null = null;

  constructor(

    private fb: FormBuilder,

    private productService: ProductService,

    private router: Router,

    private route: ActivatedRoute

  ) {

    this.form = this.fb.group({

      name: ['', Validators.required],

      price: [0, [Validators.required, Validators.min(0)]],

    });

  }

  ngOnInit(): void {

    this.productId = Number(this.route.snapshot.paramMap.get('id'));

    this.isEditMode = !!this.productId;

    if (this.isEditMode) {

      this.productService.getProduct(this.productId).subscribe(product => {

        this.form.patchValue(product);

      });

    }

  }

  onSubmit(): void {

    const product: Product = {

      ...this.form.value,

      id: this.productId ?? 0,

    };

    const request = this.isEditMode

      ? this.productService.updateProduct(product)

      : this.productService.addProduct(product);

    request.subscribe(() => this.router.navigate(['/products']));

  }

}

**✅ product-form.component.html**

<h2>{{ isEditMode ? 'Edit' : 'Add' }} Product</h2>

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

  <label>

    Name:

    <input formControlName="name" />

  </label>

  <label>

    Price:

    <input type="number" formControlName="price" />

  </label>

  <button type="submit">Save</button>

  <button routerLink="/products">Cancel</button>

</form>

**Step 6: Update routes**

**✅ app.route.ts**

import { Routes } from '@angular/router';

import { ProductFormComponent } from './p1-crud/product-form/product-form.component';

import { ProductListComponent } from './p1-crud/product-list/product-list.component';

import { NotfoundComponent } from './p1-crud/notfound/notfound.component';

export const routes: Routes = [

    { path: 'products', component: ProductListComponent },

    { path: 'products/new', component: ProductFormComponent },

    { path: 'products/:id/edit', component: ProductFormComponent },

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

    { path: '\*\*', component: NotfoundComponent }

];

**❌ Step 7: Run the Application**

**❌ Step 4: Error Handling**

**Handle errors using catchError from RxJS**

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]))]

});