**Chapter 11: RxJS & Observables in Angular**

**🔍 What is RxJS?**

**RxJS (Reactive Extensions for JavaScript)** is a library for working with asynchronous data streams using **Observables**.

Angular uses RxJS **under the hood** in:

* HttpClient
* Reactive Forms
* Router events
* Signals (in newer versions)
* User input handling, polling, websockets, etc.

**🧠 Key RxJS Concepts**

| **Term** | **What It Means** |
| --- | --- |
| Observable | A stream of data you can subscribe to |
| Observer | A handler that reacts to values from observable |
| Operators | Functions to transform or filter the data stream |
| Subject | A multicast observable (can emit + listen) |

**🧱 Basic Observable Example**

**✅ Creating an Observable**

import { Observable } from 'rxjs';

const observable = new Observable<number>(subscriber => {

subscriber.next(1);

subscriber.next(2);

subscriber.complete();

});

**✅ Subscribing**

observable.subscribe({

next: value => console.log(value),

complete: () => console.log('Done')

});

**🌊 Observables with HttpClient**

Most real-world use comes from services like this:

getProducts(): Observable<Product[]> {

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

}

Then you subscribe in your component:

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

this.products = products;

});

**⚙️ Most Common RxJS Operators**

| **Operator** | **Description** |
| --- | --- |
| map() | Transform each emitted value |
| filter() | Emit only values that match a condition |
| tap() | Perform side effects (e.g., logging) |
| switchMap() | Cancel previous observable and switch to a new one |
| catchError() | Handle errors and return fallback or rethrow |
| combineLatest() | Combine multiple observables |

**📦 Practical Example: switchMap for Live Search**

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

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

import { debounceTime, switchMap } from 'rxjs/operators';

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

@Component({

standalone: true,

selector: 'app-product-search',

template: `

<input [formControl]="searchControl" placeholder="Search products" />

<ul>

<li \*ngFor="let product of products">{{ product.name }}</li>

</ul>

`,

imports: [ReactiveFormsModule]

})

export class ProductSearchComponent {

searchControl = new FormControl('');

products: any[] = [];

constructor(private productService: ProductService) {

this.searchControl.valueChanges

.pipe(

debounceTime(300),

switchMap(query => this.productService.searchProducts(query))

)

.subscribe(results => this.products = results);

}

}

**🔄 Subjects & BehaviorSubjects**

**✅ Subject**

Can manually emit values and subscribe to them.

const subject = new Subject<number>();

subject.subscribe(val => console.log('A:', val));

subject.next(42);

**✅ BehaviorSubject**

Same as Subject but keeps the **last value** and emits it immediately to new subscribers.

const counter = new BehaviorSubject<number>(0);

counter.next(1); // All subscribers get 1

✅ Use in services to **share data across components**.

**🧪 Real-World: Sharing Data with BehaviorSubject**

**theme.service.ts**

@Injectable({ providedIn: 'root' })

export class ThemeService {

private theme$ = new BehaviorSubject<'light' | 'dark'>('light');

setTheme(theme: 'light' | 'dark') {

this.theme$.next(theme);

}

getTheme(): Observable<'light' | 'dark'> {

return this.theme$.asObservable();

}

}

**navbar.component.ts**

constructor(private themeService: ThemeService) {

this.themeService.getTheme().subscribe(mode => {

this.isDarkMode = mode === 'dark';

});

}

**🧠 Summary**

| **Feature** | **Use Case** |
| --- | --- |
| Observable | Streaming data (API, UI, events) |
| switchMap | Search, cancel old requests |
| catchError | Handle HTTP/API errors |
| Subject | Manual event stream (like emitters) |
| BehaviorSubject | Data sharing between components |

**✅ Exercises**

1. Build a search box using debounceTime() and switchMap() to query a fake API.
2. Create a service with a BehaviorSubject to store a global setting (e.g., app language).
3. Fetch data from API and transform with map() before displaying.
4. Handle API errors using catchError() and show a message.