**RxJS Observables**

**RxJS**

Reactive Extensions for JavaScript (RxJS) is a third-party library used by the Angular team.

RxJS is a reactive streams library used to work with asynchronous streams of data.

Observables, in RxJS, are used to represent asynchronous streams of data. Observables are a more advanced version of Promises in JavaScript

**Why RxJS Observables?**

Angular team has recommended Observables for asynchronous calls because of the following reasons:

1. Promises emit a single value whereas observables (streams) emit many values
2. Observables can be cancellable where Promises are not cancellable. If an HTTP response is not required, observables allow us to cancel the subscription whereas promises execute either success or failure callback even if the results are not required.
3. Observables support functional operators such as map, filter, reduce, etc.,

Create and use an observable in Angular

**Example**:

**app.component.ts**

1. import { Component } from '@angular/core';
2. import { Observable } from 'rxjs';
3. @Component({
4. selector: 'app-root',
5. styleUrls: ['./app.component.css'],
6. templateUrl: './app.component.html'
7. })
8. export class AppComponent {
9. data!: Observable<number>;
10. myArray: number[] = [];
11. errors!: boolean;
12. finished!: boolean;
13. fetchData(): void {
14. this.data = new Observable(observer => {
15. setTimeout(() => { observer.next(11); }, 1000),
16. setTimeout(() => { observer.next(22); }, 2000),
17. setTimeout(() => { observer.complete(); }, 3000);
18. });
19. this.data.subscribe((value) => this.myArray.push(value),
20. error => this.errors = true,
21. () => this.finished = true);
22. }
23. }

Line 2: imports Observable class from rxjs module

Line 11: data is of type Observable which holds numeric values

Line 16: fetchData() is invoked on click of a button

Line 17: A new Observable is created and stored in the variable data

Line 18-20: next() method of Observable sends the given data through the stream. With a delay of 1,2 and 3 seconds, a stream of numeric values will be sent. Complete() method completes the Observable stream i.e., closes the stream.

Line 22: Observable has another method called subscribe which listens to the data coming through the stream. Subscribe() method has three parameters. The first parameter is a success callback which will be invoked upon receiving successful data from the stream. The second parameter is an error callback which will be invoked when Observable returns an error and the third parameter is a complete callback which will be invoked upon successful streaming of values from Observable i.e., once complete() is invoked. After which the successful response, the data is pushed to the local array called myArray, if any error occurs, a Boolean value called true is stored in the errors variable and upon complete() will assign a Boolean value true in a finished variable.

**app.component.html**

1. <b> Using Observables!</b>
2. <h6 style="margin-bottom: 0">VALUES:</h6>
3. <div \*ngFor="let value of myArray">{{ value }}</div>
4. <div style="margin-bottom: 0">ERRORS: {{ errors }}</div>
5. <div style="margin-bottom: 0">FINISHED: {{ finished }}</div>
6. <button style="margin-top: 2rem" (click)="fetchData()">Fetch Data</button>

Line 4: ngFor loop is iterated on myArray which will display the values on the page

Line 6: {{ errors }} will render the value of errors property if any

Line 8: Displays finished property value when complete() method of Observable is executed

Line 10: Button click event is bound with fetchData() method which is invoked and creates an observable with a stream of numeric values

**Output**:

