Lecture 15:
RxJs and Observables

What are Observables?

- **Observables** represent a collection of future values or events that can be observed and manipulated. Think of them as “streams” of data that arrive over time.
- **Core Concepts:**
    - **Producer –** Emits data/events
    - **Observer –** Listens to the data and acts upon it
    - **Lazy Execution –** Observables don’t do anything until they’re subscribed to.

Observable Lifecycle

1. **Creation** – Define an Observable using **new Observable ( )** or creation methods like **of, from.**
2. **Subscription –** Observers subscribe to the Observable to start receiving data
3. **Emission –** The Observable emits values using **next ( )**
4. **Completion / Error –** Observable signals when it’s done with **complete ( )** or **error ( )**

Creating Observables

```
const observable = new Observable(subscriber => {
  subscriber.next('Data 1');
  subscriber.next('Data 2');
  subscriber.complete();
});

// Subscribing to the Observable
observable.subscribe({
  next: value => console.log('Received:', value),
  complete: () => console.log('Done!'),
  error: err => console.error('Error:', err)
});
```

Key RxJs Operators

Transformation Operators

- **map –** Modify each value in the stream

```
of(1, 2, 3).pipe(
  map(x => x * 2)
).subscribe(value => console.log(value));
// Output: 2, 4, 6
```

- **scan –** Accumulate values like reduce

```
of(1, 2, 3).pipe(
  scan((acc, value) => acc + value, 0)
).subscribe(value => console.log(value));
// Output: 1, 3, 6
```

Key RxJs Operators

Filtering Operators

- **filter –** Emit only values that meet a condition

```
of(1, 2, 3, 4, 5).pipe(
  filter(x => x % 2 === 0) // Emit only even numbers
).subscribe(value => console.log(value));
// Output: 2, 4
```

- **debounceTime –** Ignores emissions that occur too quickly

```
fromEvent(document, 'click').pipe(
  debounceTime(1000) // Emit only after 1 second of inactivity
).subscribe(event => console.log('Clicked:', event));
// Output: Logs clicks spaced by more than 1 second
```

Key RxJs Operators

Combination Operators

- **merge –** Combine multiple Observables into one

```
const obs1 = of('A', 'B');
const obs2 = of(1, 2);

merge(obs1, obs2).subscribe(value => console.log(value));
// Output: 'A', 'B', 1, 2
```

- **combineLatest –** Emit combined latest values from Observables

```
const obs1 = of(1, 2, 3);
const obs2 = of('A', 'B', 'C');

combineLatest([obs1, obs2]).subscribe(([num, char]) => console.log(num, char));
// Output: 3 'A', 3 'B', 3 'C'
```

Key RxJs Operators

Error – Handling Operators

- **catchError –** Handles errors in a stream

```
throwError('Error!').pipe(
  catchError(err => {
    console.error('Caught:', err);
    return of('Recovered value'); // Fallback value
  })
).subscribe(value => console.log(value));
// Output:
// Caught: Error!
// Recovered value
```

Key RxJs Operators

Error – Handling Operators

- **retry –** Resubscribe in case of errors

```
interval(1000).pipe(
  map(value => {
    if (value > 2) throw new Error('Value too high!');
    return value;
  }),
  retry(2) // Retry twice before throwing an error
).subscribe(
  value => console.log(value),
  err => console.error('Error:', err)
);
// Output: 0, 1, 2, 0, 1, 2, 0, 1, 2, Error: Value too high!
```

What is a Subject?

A **Subject** is a special type of Observable that allows multicasting to multiple Observers.

Types of Subjects:

- **Subject –** Basic Subject that emits values to all subscribers
- **BehaviorSubject –** Emits the most recent value to new subscribers
- **ReplaySubject –** Replays a set of number of past values to new subscribers

Practical Use Cases

- **HTTP Requests from Server**

```
this.http.get('/api/data').subscribe((data: any) => console.log(data));
```

- **Listening to Form Value Changes**

```
this.form.get('name')?.valueChanges.pipe(
  debounceTime(300),
  distinctUntilChanged()
).subscribe(value => console.log(value));
```