HANDS-ON

Hands-on No : 29

Topic : Angular

Date : 06.11.2025

Practice- State Management

Task #1:

Create a component that:

1. Calls a fake API (`setTimeout`) after 2 sec
2. Shows "Loading..." while waiting
3. Shows data when ready
4. Must use `async pipe` (no manual subscribe)

Hint:
```
data$ = of('Hello Async').pipe(delay(2000));
```

Bonus: Add a button that cancels the request using `takeUntil()`.

Task #2 :

Create a simple notification service using Subject

Requirements:

- Create a `NotificationService`
- Use `Subject<string>` to push new messages
- Component subscribes and displays them in UI
- Use async pipe (no manual subscribe)

Bonus tasks:

- Add auto-dismiss after 3 seconds (RxJS timer)
- Convert to `BehaviorSubject` to show last message on reload
- Replace with `Signal` version later

Task#3:

Create a component that:

- Subscribes to `interval(1000)`
- Displays current count in UI
- Stops when component is destroyed ☑
- Use `takeUntilDestroyed(DestroyRef)` (no manual Subject)

Bonus:

- Convert to async pipe & show in template
- Add start/stop button using `Subject`
- Convert to **Signal** and remove RxJS entirely

Task#4:

Create a component that:

- Uses `fromEvent(document, 'click')`
- Counts and displays click count using template
- Must auto unsubscribe when destroyed
- Use **takeUntilDestroyed(DestroyRef)** (no Subjects)

Bonus:

- Convert click stream to Signal using `toSignal()`
- Add reset button using Subject
- Show leak version vs clean version

Task # 5:

Build a simple counter store using signals

1. Create a Signal store with: `count`, `doubleCount`, `increment()`
2. Show count in UI
3. Log count change using effect()
4. Add reset button

Task #6:

Write a stream that:

1. Emits a value every 1 second
2. Stops after 5 values
3. Logs each value to console

Hint: use `interval()` + `take(5)`

Task #7

Create a counter service using `BehaviorSubject`

- start at 0
- method: `increment()`, `reset()`
- expose `count$` observable
- log every update from component

Task # 8

Create a search input:

1. Emits keystrokes
2. Ignore empty strings
3. Wait 300ms (debounce)
4. Cancel previous API call if new keystroke
5. Log results

Hint: use `fromEvent`, `filter`, `debounceTime`, `switchMap`, `tap`

Task #9:

Create a service that fetches `users`

- Retry 2 times if API fails
- Fallback to empty array
- Use `finalize` to toggle loading state
- Log error to console

Task #10:

1. Create an input box with FormControl
2. Emit valueChanges
3. Bind the stream to template using AsyncPipe
4. Add debounce of 500ms
5. Ensure no manual subscribe is used

```
searchControl = new FormControl('');
search$ = this.searchControl.valueChanges.pipe(debounceTime(500));

<p>Search: {{ search$ | async }}</p>
```

- Typing updates template **automatically**
- No memory leaks

Task #11:

Build a **Counter + User Loader App**:

1. Counter:
   - increment/decrement buttons
   - display double counter using selector
2. User Loader:
   - Button → dispatch `loadUser(id)`
   - Display user name using selector
3. Use **Effects** to fetch API
4. Optional: enable DevTools to inspect actions