Here’s a clear introduction to **state management in Angular**, including:

1. ✅ What state management is
2. 🧰 How to manage state using **services**
3. 🚀 How to use **NgRx** for advanced, scalable state management with an example

**🧠 1. Introduction to State Management**

**What is State?**

**State** refers to the data that defines the current condition of your application — like:

* The current user
* Form inputs
* UI flags (e.g., modal open/close)
* Data fetched from APIs (e.g., a list of products)

**Why Do We Need State Management?**

As your app grows, managing state across multiple components becomes complex. State management helps:

* Share data between components
* Make state predictable and traceable
* Improve maintainability and scalability

**🧰 2. Using Services for State Management (Basic Approach)**

Angular services are **singleton** classes — ideal for holding shared state.

**Example:**

**✅ counter.service.ts**

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

@Injectable({

providedIn: 'root',

})

export class CounterService {

private count = 0;

getCount(): number {

return this.count;

}

increment(): void {

this.count++;

}

reset(): void {

this.count = 0;

}

}

**✅ Use in Components:**

@Component({ /\* ... \*/ })

export class CounterComponent {

constructor(public counterService: CounterService) {}

increment() {

this.counterService.increment();

}

get count() {

return this.counterService.getCount();

}

}

🔍 Services work great for small to medium apps, but can get messy in complex apps with multiple state dependencies.

**🚀 3. NgRx for Advanced State Management**

**NgRx** is Angular's official **Redux-style** state management library — inspired by Redux (used in React).

It provides:

* A single **global store**
* **Actions** to describe state changes
* **Reducers** to handle state transitions
* **Effects** for handling side-effects (e.g., API calls)
* Integration with **Redux DevTools**

**🏗 NgRx Setup Example (Counter App)**

**Step 1: Install NgRx**

ng add @ngrx/store

ng add @ngrx/effects

ng add @ngrx/store-devtools

**Step 2: Create Actions**

// counter.actions.ts

import { createAction } from '@ngrx/store';

export const increment = createAction('[Counter] Increment');

export const decrement = createAction('[Counter] Decrement');

export const reset = createAction('[Counter] Reset');

**Step 3: Create Reducer**

// counter.reducer.ts

import { createReducer, on } from '@ngrx/store';

import \* as CounterActions from './counter.actions';

export const initialState = 0;

export const counterReducer = createReducer(

initialState,

on(CounterActions.increment, (state) => state + 1),

on(CounterActions.decrement, (state) => state - 1),

on(CounterActions.reset, () => 0)

);

**Step 4: Register the Reducer**

// app.module.ts

import { StoreModule } from '@ngrx/store';

import { counterReducer } from './counter.reducer';

@NgModule({

imports: [

StoreModule.forRoot({ count: counterReducer })

],

})

export class AppModule {}

**Step 5: Use Store in a Component**

// counter.component.ts

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

import { Store } from '@ngrx/store';

import { increment, decrement, reset } from './counter.actions';

@Component({

selector: 'app-counter',

template: `

<h1>{{ count$ | async }}</h1>

<button (click)="increment()">+</button>

<button (click)="decrement()">−</button>

<button (click)="reset()">Reset</button>

`,

})

export class CounterComponent {

count$ = this.store.select('count');

constructor(private store: Store<{ count: number }>) {}

increment() {

this.store.dispatch(increment());

}

decrement() {

this.store.dispatch(decrement());

}

reset() {

this.store.dispatch(reset());

}

}

**🧠 Summary: When to Use What**

| **Method** | **Use When** |
| --- | --- |
| Services | Small/medium apps with local state or minimal sharing |
| NgRx (Redux pattern) | Large apps with complex state, many interactions, or need for time-travel debugging |

Let me know if you want a sample GitHub repo, a diagram of NgRx flow (action → reducer → store), or how to integrate **NgRx Effects** for API calls.