Angular Deep Dive: Advanced State Management

Introduction to State Management in Angular

Ervis Trupja
Software Engineer, Author

@ervis_trupja | www.dotnethow.net

Course Introduction

Introduction to state management

- What is state in Angular?
- Local vs Global state
- Services for shared state

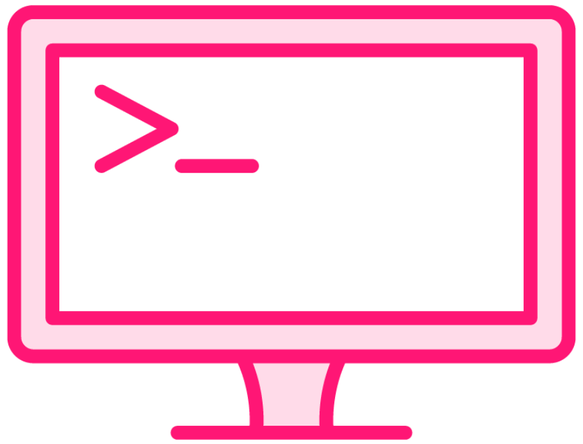
Upgrading state management to NgRx

- What is and why use NgRx?
- Setting up NgRx store, actions, reducers
- Selecting state with selectors
- Handling side effects

NgRx and Signals

- Using NgRx entity for collections
- Integrating signals with NgRx selectors

Before You Get Started



- Angular (v20)
- Angular CLI (v20)
- Visual Studio Code
- NgRx (v19.12)
- RxJS (v7.8)

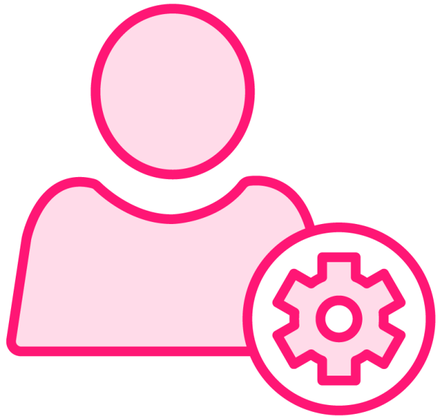
Compatibility

https://angular.dev/reference/versions

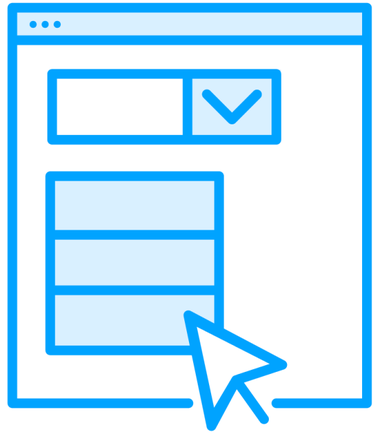
What is State in Angular?

State is the data that your Angular application holds and manages at any given time

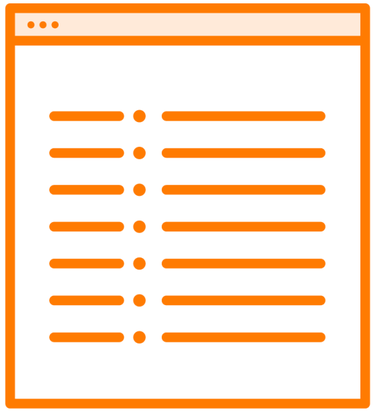
Examples of Angular State



User-specific data

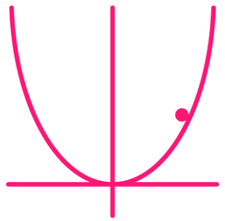


UI state

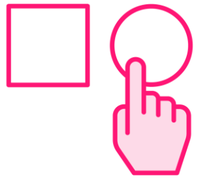


Application data

Why State Management?



Predictability: Consistent app behavior regardless of user actions.



Interactivity: Dynamic response to user input and data changes.

Maintainability: Easier to understand, debug, and scale your code.

Demo: Simple State in Angular

Tracking pending tasks with a new state piece

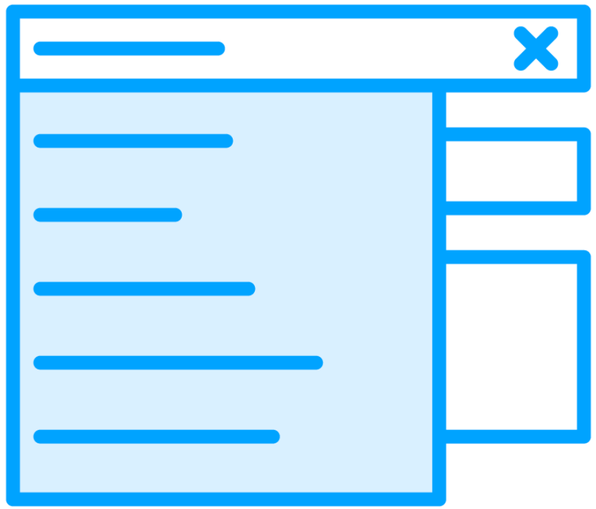
Local vs. Global State

Local State

Local State (Component State) is data that is managed and used within a single component.

It is ideal for isolated, component-specific logic.

Local State (Component State)



Data managed within a single component

Ideal for simple, isolated logic

Used with @Input/@Output or local properties

Great for toggles, counters, or form fields

Makes components reusable and testable

Global State

Global State (Application State) is data that is shared and accessed by multiple components in the application.

It helps coordinate interactions across different parts of the app.

Global State (Application State/Shared State)

Data shared across multiple components

Ideal for app-wide logic and consistency

Managed with services or state libraries

Used for auth, themes, or notifications

Avoids deeply nested prop passing

When to Choose Which?

Local when data is self-contained and doesn't affect other distant parts of the app

Global when data needs to be accessible or modified by multiple, potentially unrelated, components

Demo: Local and Global State Examples

Using Services for Shared State

Angular services are singletons, meaning only one instance exists throughout the application (when provided at the root).

Use RxJS BehaviorSubject in services to manage shared state

RxJS

(Reactive Extensions for JavaScript)

RxJS is a library

Asynchronous data streams

Event-based programming

RxJS provides tools to:

- Create
- Manipulate
- React

It uses Observables

BehaviorSubject

BehaviorSubject is a type of RxJS Subject that holds a ”current value”.

BehaviorSubject

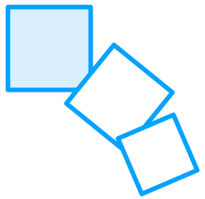
BehaviorSubject is a special type of Observable that also acts as an observer, allowing it to both emit values and be subscribed to, while always holding a "current value."

BehaviorSubject

BehaviorSubject is a type of RxJS Subject that holds a ”current value”.

How Services Manage State?

Declare a BehaviorSubject to hold a piece of shared state (e.g., tasks$: BehaviorSubject<Task[]>).



Components can subscribe to this BehaviorSubject to react to state changes.

The service can also provide methods to update the BehaviorSubject's value. (e.g., tasks$.next(...))

Demo: Using Services for Shared State

Resetting form data

Building Reactive Notifications Using RxJS in Angular

Communication Flow

1

Component A (task-form, task-list): Calls a method on the shared service to update the value.

2

Service: Emits the new value using BehaviorSubject.

3

Component B (task-notifications): Subscribes and reacts to the updated value.

Demo: Adding Notifications

Notify users when a new task is added or a task is removed

When to Move to a State Management Library

Limitations of Service-Based State Management

Scalability: Too many BehaviorSubjects can get messy.

Debugging: Hard to track changes across services.

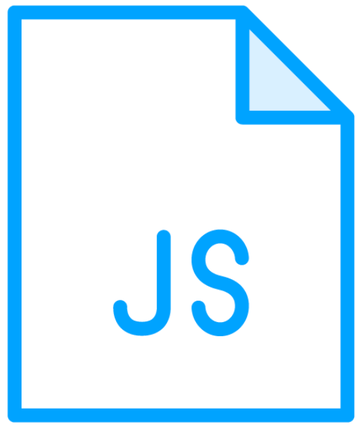
Predictability: Inconsistent updates can cause bugs.

Side Effects: Async operations make state harder to manage.

Signs Your Application Needs a Library

- Complex data flows
- Multiple sources of truth
- Difficult debugging
- Growing codebase complexity

Introducing NgRx (or similar libraries)



Adds structure for managing global state

Enforces patterns

- Actions
- Reducers
- Effects
- Selectors

Improves predictability and testability

Helps with large, complex apps

Supports powerful debugging with DevTools