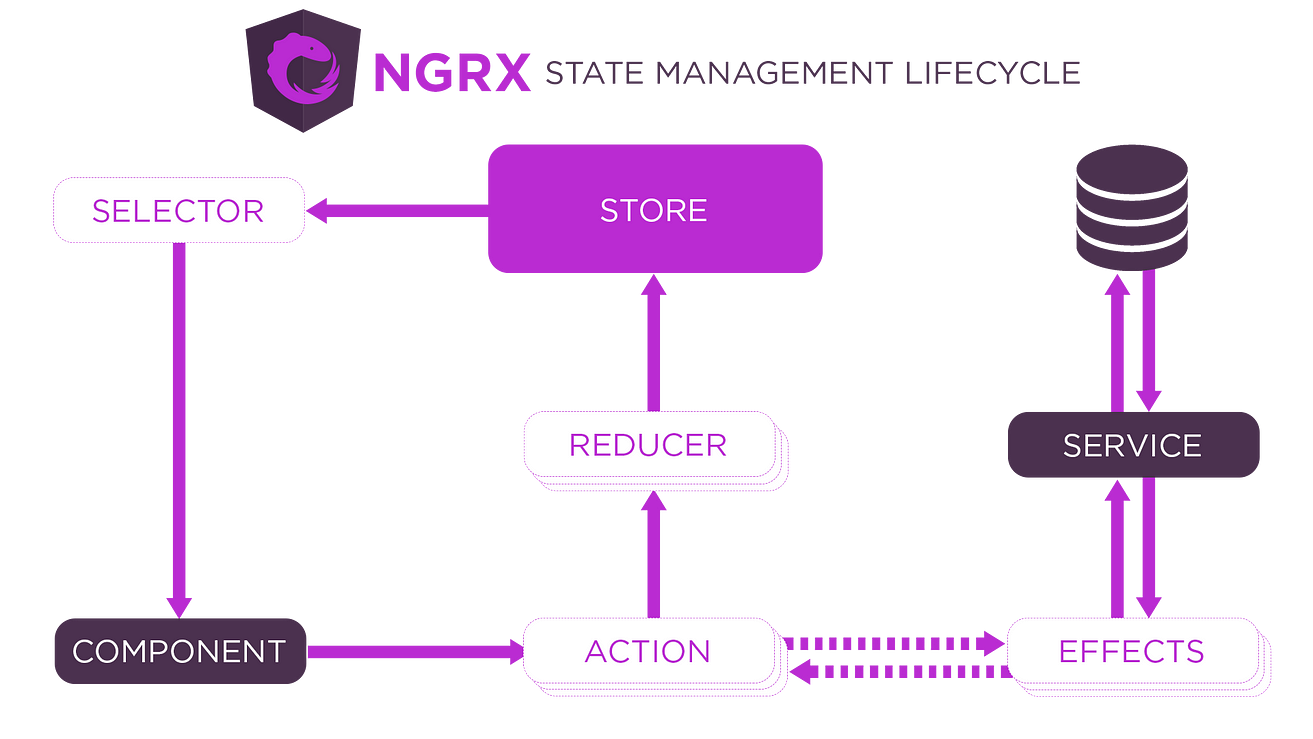
**State management with NgRx in Angular**

Modern front-end applications commonly use component concepts, which represent independent, reusable, and autonomous parts of a user interface. This concept is central in several popular front-end libraries, such as Angular, React, and Vue.js. Despite bringing several benefits and simplicity, it can also cause issues when multiple components require the same data. To address this problem, there is a pattern called Redux. Although the Redux library initially gained popularity in the React ecosystem, it can also be used in other frameworks like Angular and Vue.js. In this article, we will focus on state management with the NgRx library. Although Redux can be used in Angular, it is common to see applications using NgRx, a popular state management library specifically designed for Angular.

**What`s NgRx?**

As mentioned in the introduction, NgRx is a library for state management inspired by Redux. It implements the Redux principles for Angular, providing a predictable and structured way of managing the application state. NgRx uses concepts such as actions, reducers, effects, and selectors to manipulate the state in a controlled manner.



**Store**

“Store” is the part of the state manager that is implemented with a structure of immutable data. In other words, the data contained therein cannot be changed directly. Therefore, all state changes must be done through actions. These actions define what will be changed in the state through a mechanism called the ‘reducer,’ which we will talk about in the course of the article. This function processes communication with the state.

**Reducer**

The responsibility of the “reducer” is to process all necessary actions that will change the store's state. It receives as input the current state and the action and returns the new state after the change.

**Action**

The “actions” are simply objects that represent a change in the state. They are sent to the store with the information that will change the store’s state.

**Selectors**

The “selectors” are functions used to access and get specific information of state stored. Therefore they allow the components to request only the data necessary, instead of getting directly the complete state of the store, this makes the code more modular, reusable, and easy to maintain. Furthermore, they can make calc and transformation in the state before delivering to the component that requested it.

**Effects**

The “effects” refer to functionalities that deal with asynchronous tasks or side effects, such as network requests, database access, calls to external APIs, or any operation that is not purely synchronous. However, we will not use them in this article. I prefer to write an article specifically dedicated to this topic soon.

**NgRx in Practice**

The application that I will use is in my repository on Git Hub, check this link [igormarti/angular-ngrx-shopping (github.com)](https://github.com/igormarti/angular-ngrx-shopping" \t "_blank) if you want to clone it. However, you can also apply the concepts in an application from scratch, in my application, I will use state manager to favorite the products listed. Our application will have some components that will request the favorite products from our store. You will see more details in the following steps. Just a note: the code implementation of this project was done using Angular version 16 and NgRx version 7.8.0.

**Installing NgRx in Angular**

To install the NgRx in you project, run the following command in the root folder:

npm install @ngrx/store --save

**Structuring NgRx in our Project**

After installing the NgRx in our project, now is the hour to structure it, in the following step we will create our actions, reducers, selectors, and the initial state.