**Useful Links :**

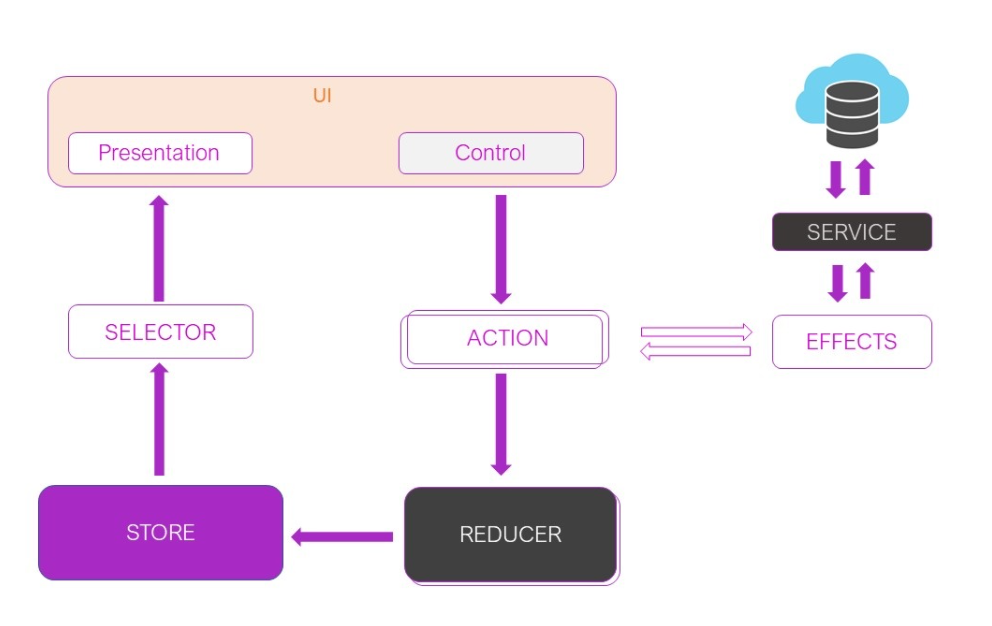
* [Angular State Management with NgRx - DZone Web Dev](https://dzone.com/articles/angular-app-state-management-with-ngrx)
* [How to Start Flying with Angular and NgRx - NgRx inDepth](https://indepth.dev/posts/1042/how-to-start-flying-with-angular-and-ngrx)
* [RxJS - Observable](https://rxjs.dev/guide/observable#pull-versus-push)
* [NgRx - @ngrx/component-store](https://ngrx.io/guide/component-store)
* [NgRx Example - Tutorial - StackBlitz](https://stackblitz.com/angular/nvplyaejvlay?file=src%2Fapp%2Fmy-counter%2Fmy-counter.component.ts)
* [JavaScript Callback Functions – What are Callbacks in JS and How to Use Them](https://www.freecodecamp.org/news/javascript-callback-functions-what-are-callbacks-in-js-and-how-to-use-them/)
* [JS Functions Are Objects](https://academind.com/tutorials/javascript-functions-are-objects/)
* [Array.prototype.reduce() - JavaScript | MDN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce)
* [NgRx - Walkthrough](https://ngrx.io/guide/store/walkthrough)
* [Overview of NgRx Benefits and Advantages | Toptal](https://www.toptal.com/angular/why-use-ngrx)
* [Angular NgRx Store and Redux - When to use a Store and Why?](https://blog.angular-university.io/angular-2-redux-ngrx-rxjs/)
* <https://blog.logrocket.com/angular-state-management-made-simple-with-ngrx/#managingstateinfrontendapplications>

**Application State**

* it is the entire memory of the application.
* it is the data received via API calls, user inputs, presentation UI State, app preferences, etc. Simply put, it is the data that can differentiate two instances of the same application.
* A simple concrete example of an application state would be a list of customers maintained in an application.

## NgRx App State Management

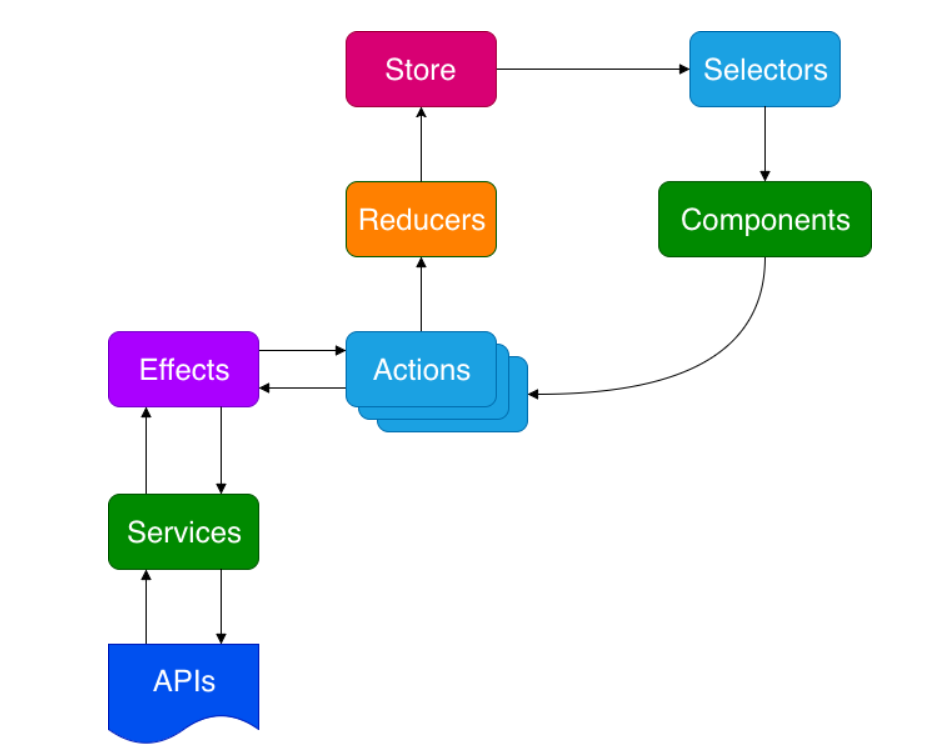
* **Store**: The store is what holds the app's state.
* **Action**: A unique event dispatched from components and services that describe how the state should be changed. For example, ‘Add Customer’ can be an action that will change the state (i.e. add a new customer to the list).
* **Reducer**: All the state changes happen inside the reducer; it responds to the action and, based on that action, it will create a new immutable state and return it to the store.
* **Selector**: Selector is a function used for obtaining a part of the state from the store.
* **Effect**: A mechanism that listens for dispatched actions in an observable stream, processes the server response, and returns new actions either immediately or asynchronously to the reducer to change the state. Please note that we are not using 'effect' in this example app.

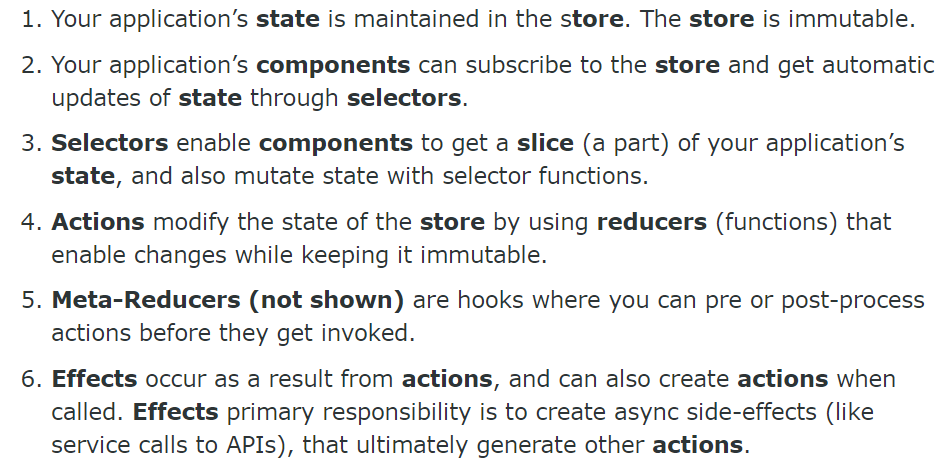


* In the world of Angular, there are a lot of patterns to follow to manage your application. These typically involve using decorators like Input and Output, or things like RxJs Observables to watch for data changes. However, there is a reactive state technology that solves this challenge called **NgRx**.
* NgRx uses streams to interact with a data **store**.
* Instead of injecting services everywhere and managing communication between them, NgRx manages your application from one singular source.
* Using NgRx, you work with your application in terms of its overall **state**, instead of individual components.

There are five parts that constitute NgRx:

1. **Store**
2. **Reducers (and Meta-Reducers)**
3. **Actions**
4. **Selectors**
5. **Effects**





Bhautik Makwana

Jr. Software Engineer

**Install NgRx and tools**

* ng add @ngrx/schematics@latest
* **ng config cli.defaultCollection @ngrx/schematics**
* npm install @ngrx/store --save
* npm install @ngrx/effects --save
* npm install @ngrx/entity --save
* npm install @ngrx/store-devtools –save
* ng generate @ngrx/schematics:store State --root --module app.module.ts

