Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2022, 9(4):166-171

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Advanced State Management in Angular: A Guide to Implementing NGRX Store

Bhargav Bachina

Abstract This paper explores the utilization of NGRX Store, a comprehensive state management library, within Angular-based Single Page Applications (SPAs). Traditional web applications manage user sessions server-side, maintaining state across multiple page requests. However, this paradigm shifts in SPAs, where maintaining consistent state through various user interactions and page states presents a complex challenge. NGRX Store offers a solution by enabling efficient state management, akin to Redux's approach but tailored specifically for Angular environments. The paper delves into setting up and implementing NGRX Store within an Angular application, providing a structured approach to state management. We will cover essential topics including the basics of NGRX, installation procedures, and detailed implementation steps from API integration to Angular-specific adaptations. Through an example project, this study aims to illustrate the practical application and benefits of integrating NGRX Store, enhancing SPA performance, and user experience by ensuring state persistence and efficient data handling. This research is intended to serve as a comprehensive guide for developers looking to leverage NGRX Store for state management in Angular applications, contributing to the broader understanding of managing state in modern web development environments.

Keywords JavaScript, Programming, Angular, Web Development, Software Development

The NGRX Store serves as an advanced state management library specifically designed for Angular applications. In the traditional model of web development, session management was primarily handled server-side, enabling the persistence of user sessions across multiple page requests. This method ensured that user data and application state remained intact throughout the user's interaction with the website. However, with the advent of Single Page Applications (SPAs), this approach has become less effective. SPAs dynamically update content without reloading the entire page, complicating the process of maintaining a consistent application state across user interactions. In response to these challenges, the NGRX Store emerges as a powerful tool, simplifying state management in Angular-based SPAs. By adopting a centralized store for managing the state of the application, it provides a consistent and predictable state container, inspired by the principles of Redux but tailored for the Angular ecosystem.
This article aims to delve deeply into the process of integrating and utilizing the NGRX Store within Angular applications. We will embark on a comprehensive exploration, beginning with an overview of state management necessities and progressing through the practical steps required for setting up the NGRX Store. The discussion will encompass a range of topics, including the fundamental concepts of NGRX, the prerequisites for its implementation, detailed installation instructions, and the step-by-step process of integrating the API with the Angular framework. By the end of this article, readers will gain a thorough understanding of how to effectively manage state in Angular SPAs using the NGRX Store, thereby enhancing the robustness and user-friendliness of their applications.

1. Example Project

Multiple separate builds should form a single application. These separate builds should not have dependencies between each other, so they can be developed and deployed individually.
https://miro.medium.com/v2/resize:fit:1400/1*RiV4entntuAUDyPgUEtHUA.gif
Here is the example project in GitHub where you can clone and run it on your local machine.
//clone the project git clone https://github.com/bbachi/angular-ngrx-example.git
// Run the API cd api npm installnpm run dev
// Run the Angular Appcd uinpm installnpm start

2. How it Works

NGRX is a state management tool inspired by redux for Angular Applications. When your application gets bigger and bigger communication becomes difficult to handle. NGRX provides unidirectional data flow and a single source of truth for the entire app.
Components that are aware of the store are called smart components and Components that aren't aware of the store are called dumb components. If you look at the following diagram, all the smart components send the data to the store and receive data from the store promoting unidirectional data flow for the entire app.

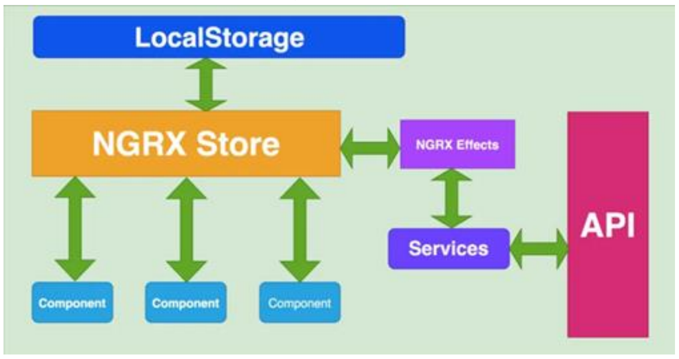


Figure 1: NGRX Store

Refreshing or reloading the page will lose the entire state of the application. That's when LocalStorage comes into the picture. The whole state of the app is serialized and saved into local storage just before reloading the page and the entire state is deserialized from LocalStorage and reinitialize the state of the app. This is called rehydrating the store.
Sometimes we must make API calls to fetch the data for the app. Whenever the store needs data from the backend API, it uses NGRX effects to make an API call and fetch the data and update the store. We will see this in detail in further sections.

3. NGRX Basics

NGRX Store is inspired by Redux which provides State management for the angular applications, and it is RXJS powered which boosts the performance and consistency of angular apps.
Let's see how it works!!

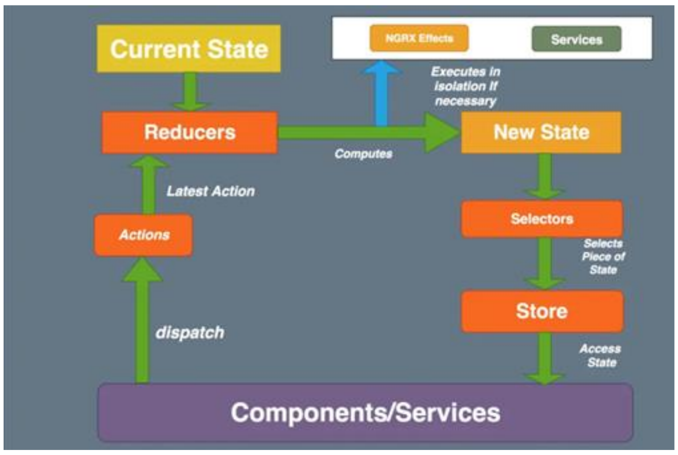


Figure 2: NGRX Store in detail

- Actions are dispatched from the components and services. These are just unique events with type and payload and can be dispatched to the store.
- Reducers are the pure functions that take the latest action and current state and return the new state.
- Selectors are the pure functions that enable us to select a slice of the state.
- The State can be accessed through Store observables in components and service.
- NGRX Effects are the functions that can be executed to get the new data for the state. For example, if your component needs new data from the API, the component dispatches an action, the reducers invoke the effects and services to get the new data, reducer returns the new state with that data from API.

4. Prerequisites
There are some prerequisites for this article. You need to have nodejs installed on your laptop and how http works. If you want to practice and run this on your laptop you need to have these on your laptop.
- NodeJS (https://nodejs.org/en)
- Angular CLI (https://angular.io/cli)
- Typescript (https://www.typescriptlang.org/)
- VSCode (https://code.visualstudio.com/)
- ngx-bootstrap (https://valor-software.com/ngx-bootstrap/#/)
- NGRX (https://ngrx.io/)

This is going to be a big post if I include the whole implementation of the project. So, I created a separate post for the actual implementation of the project without the NGRX store. If you are a beginner to the Angular you can have a look at the below post. Otherwise, you can skip to the next section. This post is about a step-by-step guide on how to develop an Angular app with NodeJS backend.
How To Develop and Build Angular App with NodeJS
(https://medium.com/bb-tutorials-and-thoughts/how-to-develop-and-build-angular-app-with-nodejs-e24c40444421)

5. Installation
When it comes to NGRX, we need to install a bunch of libraries. Let's install all the below libraries for the NGRX.
```
// install NGRX dependencies
npm install @ngrx/{effects,entity,router-store,store,store-devtools} –save
npm install ngrx-store-freeze ngrx-store-localstorage –save
```

6. API Implementation
Let's implement the API part of the whole application step by step. The API is built with a Nodejs and express framework. We have login, signup, add tasks, delete tasks, edit tasks, and get tasks operations.

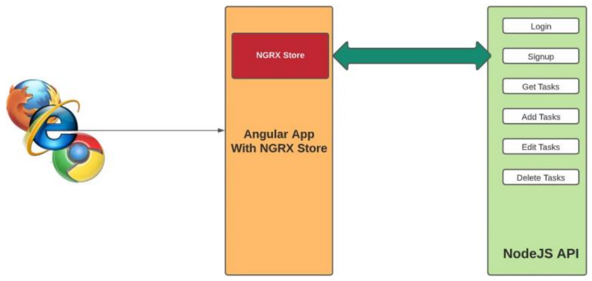


Figure 3: Angular calling NodeJS API

If you look at the above diagram, Angular with NGRX store calls the NodeJS API. Here is the file server.js which has all the operations and listens on port **3080.**
https://gist.github.com/bbachi/69e54dcdfdfc54c291efe4961c671730#file-server-js
We are using nodemon in the development environment to watch any file changes and restart the server. All you need to run is this command npm run devto start the nodejs server in the development mode.
https://gist.github.com/bbachi/078a14bcde596d64ae4bf814a1a630b1#file-package-json
Here is the demo where you start the server with the nodemon.
https://miro.medium.com/v2/resize:fit:1400/1*zULXgi9TQIEByfix3pW7EA.gif

7. NGRX Implementation

Let's implement the NGRX step by step. Here is the diagram of the NGRX structure for the application. We have actions, reducers, effects, etc.

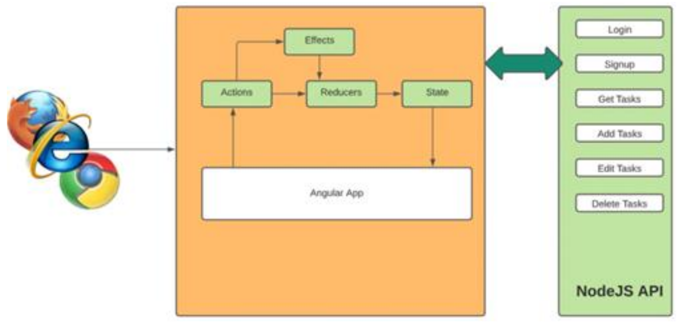
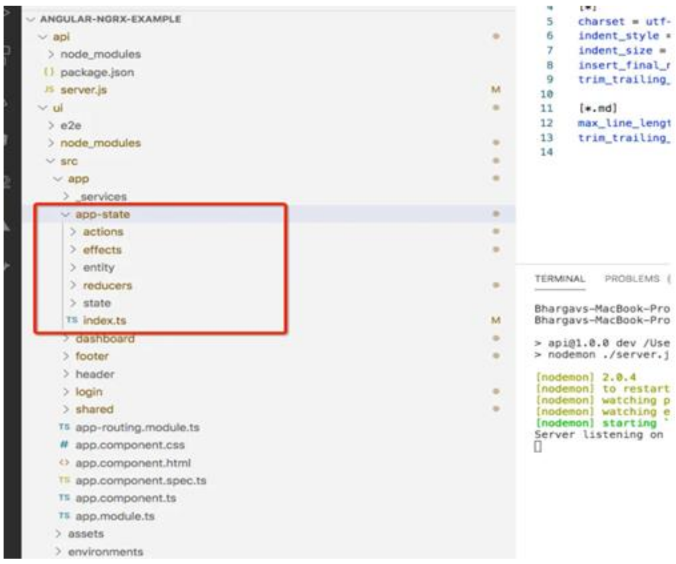


Figure 4: Angular With NGRX calling NodeJS API

If we look at the above diagram, The component from the Angular App invokes the NGRX Store by sending Actions. If the Actions have side effects such as calling the API, etc it will call the API through the effects. Once we receive the response from the API, we change the state of the application through the reducers. The reducers here are pure functions means these take the current state and output the new state without mutating the current state.

First, we need to define the state in the Angular folder structure. I usually maintain a separate folder for the state of the application.



```
ANGULAR-NGRX-EXAMPLE
  api
    node_modules
    package.json
    server.js                         M
  ui
    e2e
    node_modules
    src
      app
        _services
        app-state
          actions
          effects
          entity
          reducers
          state
          index.ts                    M
        dashboard
        footer
        header
        login
        shared
        app-routing.module.ts
        app.component.css
        app.component.html
        app.component.spec.ts
        app.component.ts
        app.module.ts
      assets
      environments
```

```
[*]
charset = utf-
indent_style =
indent_size =
insert_final_r
trim_trailing_

[*.md]
max_line_lengt
trim_trailing_
```

```
Bhargavs-MacBook-Pro
Bhargavs-MacBook-Pro

> api@1.0.0 dev /Use
> nodemon ./server.j

[nodemon] 2.0.4
[nodemon] to restart
[nodemon] watching p
[nodemon] watching e
[nodemon] starting `
Server listening on
```

Figure 5: State of the application

A.Login/Signup Flow

All the actions belong to the login and signup flow are defined below. Actions are the objects which have type and payload. The payload is optional here. Notice that createAction and props are imported from ngrx/store.
https://gist.github.com/bbachi/b295c293abc8436b6c522f1483dd0789#file-login-actions-ts
This is the Actions file for the signup flow.
https://gist.github.com/bbachi/1f0328659c876f6eaf2d279fedd8d66a#file-signup-actions-ts
Since the login and the signup actions should call the API to authenticate and register users. You need to define the effects which handle side effects.
https://gist.github.com/bbachi/561ae28fb7390baf39fa6a45389d0373#file-user-effects-ts
We have defined two functions in these effects file. One is for Action of type login, and another is for Action of type signup. You can see we are calling API calls and mapping the response to the appropriate Actions payload.
Here is the reducer for the user login and signup flow. If you notice the below file, we have defined the initial state. For each action, we are changing the state accordingly. We are also exporting some functions which give some data that is useful for the components.
https://gist.github.com/bbachi/8c95c5024fbddfd5a0d37867fa1753d2#file-user-reducer-ts

B.Todo Tasks Flow

All the actions that belong to the ToDo flow are defined below. Actions are the objects which have type and payload. The payload is optional here. Notice that createAction and props are imported from ngrx/store.
https://gist.github.com/bbachi/355bcbcfd209161bca86208dc74d17cc#file-todo-actions-ts
Since the todo actions should call the API to create, delete, edit, and get tasks from the API. You need to define the effects which handle side effects.
https://gist.github.com/bbachi/9c48287d7b49841ba3e768f5ea1424dc#file-todo-effects-ts
We have defined four functions in these effects file. One for each CRUD operation. You can see we are calling API calls and mapping the response to the appropriate Actions payload.
Here is the reducer for the todo flow. If you notice the below file, we have defined the initial state. For each action, we are changing the state accordingly. We are also exporting some functions which give some data that is useful for the components.
https://gist.github.com/bbachi/a608a7c4dce5dc1d38e3d7bd84640a9b#file-todo-reducer-ts
Here is the complete index.ts file where you define your reducers and export functions with the help of selectors.
https://gist.github.com/bbachi/11ef6889ee5a25c35e1f534ac7f37cbd#file-index-ts

8. Angular Implementation

We have seen API and NGRX implementations. Its time to see how we can integrate the NGRX store in the Angular application. The first thing we need to do is import all the NGRX related code into the App module or feature module as below.
https://gist.github.com/bbachi/c2b580829671f189b6e15eb7b896c135#file-app-module-ts
As you can see above, we need to import effects and should be registered with the EffectsModule from the NGRX Store. Let's see one example of how we can dispatch the actions and listen to the store changes.
Here is the login component in which we import NGRX store and actions we have defined above.
https://gist.github.com/bbachi/b5b689040d191fd586161e7b82704017#file-login-component-ts
As in the above component, we dispatch actions with ***this.store.dispatch*** and read from the store with the help of selectors such as ***this.store.select.***
Here are the APIs which are called by NGRX effects
https://gist.github.com/bbachi/1480590751d76f7ee3ed37ec66e4f34a#file-app-service-ts
https://gist.github.com/bbachi/bf9a095b2901f8d7d0517afa24833159#file-todo-service-ts
You can check the rest of the example in the Github link provided above.

9. Summary

- NGRX is a state management tool inspired by redux for Angular Applications.
- When your application gets bigger and bigger communication becomes difficult to handle. NGRX provides unidirectional data flow and a single source of truth for the entire app.
- Components that are aware of the store are called smart components and Components that aren't aware of the store are called dumb components.
- Sometimes we must make API calls to fetch the data for the app. Whenever the store needs data from the backend API, it uses NGRX effects to make an API call and fetch the data and update the store.
- Actions are dispatched from the components and services. These are just unique events with type and payload and can be dispatched to the store.
- Reducers are the pure functions that take the latest action and current state and return the new state.
- Selectors are the pure functions that enable us to select a slice of the state.
- The State can be accessed through Store observables in components and service.
- NGRX Effects are the functions that can be executed to get the new data for the state. For example, If your component needs new data from the API, the component dispatches an action, the reducers invoke the effects and services to get the new data, reducer returns the new state with that data from API.

10. Conclusion

In conclusion, NGRX offers a robust framework for managing state in Angular applications, particularly beneficial as applications scale and complexity increases. By establishing a unidirectional data flow and maintaining a singular source of truth, NGRX enhances communication and data consistency across the application. It distinguishes between smart and dumb components to streamline interactions and data flow. Through the integration of actions, reducers, and selectors, NGRX facilitates the efficient dispatching, processing, and retrieval of state information. Additionally, NGRX Effects play a crucial role in handling side effects, such as API calls, ensuring that the application state remains up to date with external data sources. By leveraging these mechanisms, NGRX not only simplifies state management but also contributes to more maintainable, predictable, and robust Angular applications.

References

[1]. Angular Official Doc https://angular.io/docs
[2]. TypeScript Documentation https://www.typescriptlang.org/docs/
[3]. NGRX Documentation https://ngrx.io/