*Available online www.jsaer.com*

***Journal of Scientific and Engineering Research*, 2022, 9(2):143-146**

**Research Article**

**ISSN: 2394-2630**
**CODEN(USA): JSERBR**

# State Management in Angular Applications: Enhancing Persistence through Page Refreshes with NGRX

**Bhargav Bachina**

**Abstract** Modern web applications are predominantly built using Single Page Application (SPA) frameworks like Angular, React, and Vue.js. These frameworks enable a seamless user experience by dynamically updating content without the need to reload the entire webpage. However, a significant challenge arises in maintaining the state of the application upon page refresh or browser restarts, as traditional SPAs typically lose all client-side data, necessitating a complete reload from the server. This paper addresses this issue by exploring methods to preserve application state across sessions, thereby reducing server load and enhancing user experience. We particularly focus on Angular applications, presenting a comprehensive solution that combines the NGRX store with local storage techniques. This approach ensures state persistence even when the page is reloaded. We detail the prerequisites for implementing this solution, provide a step-by-step guide through an example project, and demonstrate the practical benefits through a real-world application. Finally, we evaluate the performance implications and user experience improvements. This paper aims to provide developers with a robust strategy for state management in SPA frameworks, ensuring a more efficient and user-friendly web application.

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

Most contemporary web applications leverage Single Page Application (SPA) frameworks, including Angular, React, and Vue.js, among others. These frameworks are designed to load a single HTML page, typically named index.html, into the browser initially. After this initial load, the SPA framework takes over, dynamically managing the navigation between different views or "pages" within the application, thereby creating the illusion of a traditional multi-page website. This approach enhances user experience by providing swift transitions and reducing load times. However, this design introduces a significant challenge: maintaining the application's state. In the context of SPAs, "state" refers to the user's interactions and data currently displayed on the web page. Typically, if the user refreshes the page or restarts the browser, the SPA reloads the initial index.html file, resulting in the loss of all current state information. This can lead to a frustrating user experience as all progress or data displayed is wiped clean.

To address this issue, developers can adopt several strategies to preserve the state of the application between sessions. One common method involves making API calls to the backend server each time the application is reloaded, to fetch and restore the previous state. While effective, this approach can become impractical with an increasing number of API requests, as it can lead to server overload and increased response times. An alternative and more efficient method involves storing the state data locally within the user's browser using local storage. By doing so, the application can quickly retrieve and reinstate the state without the need for additional server requests. This technique not only improves performance but also enhances the user experience by providing immediate access to the previous state.

In this discussion, we will explore the integration of the NGRX store, a state management solution, with local storage within Angular applications. This combination offers a powerful method for retaining application state across page reloads and browser restarts, ensuring a seamless user experience and reducing the burden on the server. We will delve into the mechanics of how these technologies work together to preserve state information and provide practical examples to illustrate their implementation in real-world Angular applications.

- Prerequisites
- Example Project
- Problem
- Solution
- Implementation
- Demo
- Summary
- Conclusion

## 1. 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 separate posts for the actual implementation of the project without the NGRX store and one with 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 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)
- **How To Implement NGRX Store In Angular App:** https://medium.com/bb-tutorials-and-thoughts/how-to-implement-ngrx-store-in-angular-6ba275cd47fc

## 2. Example Project

Let's have a look at the example project. We have a simple app in which we can log in, signup, and add tasks, delete tasks, and edit tasks, etc. The entire application state is maintained with NGRX. We have actions, effects, reducers, etc. We will see all in this example.

https://miro.medium.com/v2/resize:fit:1400/0*V1kZElHOuIe-7N7i.gif

Here is the example project in GitHub where you can clone and run it on your local machine.

```
// clone the projectgit clone https://github.com/bbachi/angular-ngrx-example.git
// Run the APIcd apinpm installnpm run dev
// Run the Angular Appcd uinpm installnpm start
```

## 3. Problem

The problem we are facing is that when you load the Single Page Application in the browser you load the index.html with all the required libraries only once. From then on, the Angular framework kicks in and loads the appropriate modules and pages as route changes. When you refresh the page, you lose all the state of the application since you are reloading the index.html with all the required dependencies again.

https://miro.medium.com/v2/resize:fit:1400/1*GxqQ8xb0tGkRB-yMwfpG1w.gif

For example, let's look at the above scenario with the example project provided in the article. We signed up and logged in. You can see the logged in user profile information on the right side of the header and created tasks as well. But, as soon as you refresh the page the entire state of the app is lost, and you can't see the logged in user information in the header and tasks.

## 4. Solution

As I said earlier, we have so many options to maintain the state and we are seeing how we can preserve the state with the help of NGRX store and local storage in this example.

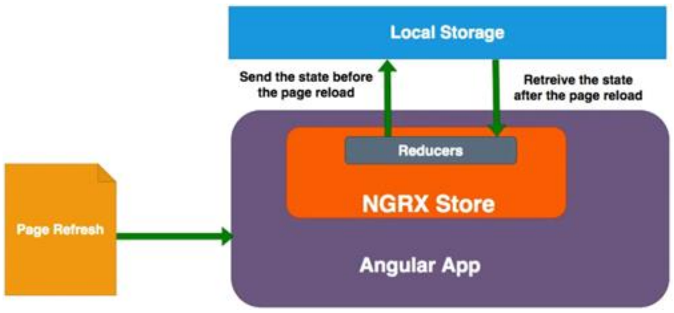
Local Storage
Send the state before the page reload
Retreive the state after the page reload
Reducers
NGRX Store
Page Refresh
Angular App

*Figure 1: Retaining the state with NGRX Store and Local Storage*

If you look at the above diagram, when you refresh the page or reload the page you are saving the application state in the local storage and retrieve it after the reload and populate the reducers. If the reducers are populated with the current state all the components subscribed to the store receive the data. This is what we call rehydration of NGRX store. You can retain the state in this way. Let’s see the implementation step by step below.

## 5. Implementation

The implementation of the above solution is straightforward. Make sure you follow the NGRX example article given above before this or click this link: (https://medium.com/bb-tutorials-and-thoughts/how-to-implement-ngrx-store-in-angular-6ba275cd47fc)

First, we need to define a separate file which handles all the storage related activities. It has the following operations such as getting current state, get item, get item by key, save state, save item, etc. You can define functions based on your requirements.

https://gist.github.com/bbachi/6313731c880b7b024459447d25d19306#file-storage-ts

Install the below library called ***ngrx-store-localstorage*** as a dependency with the following command.

npm install ngrx-store-localstorage –save

Once you installed this one you have to register the reducers with it so that the library automatically syncs that particular reducer with the localstorage. You don’t have to do anything explicitly. Here is the index.ts file where you register all the reducers with the local storage with the help of ***ngrx-store-localstorage***. Notice lines **8, 23, 24, 25**. Since we must preserve both user information and tasks information you have to register both reducers.

https://gist.github.com/bbachi/48500ac46a6b83c05cc20fd72efe3939#file-index-ts

This part takes care of automatically populating the state in the reducers to the localstorage with the key name as reducer names. Now, you have the data in the local storage how you can retrieve this and populate the reducers when the page refresh happens.

Let’s see how we can achieve that with help of ***stoarge.ts*** file that we defined above. We have an initial state in each reducer as below you need to populate the initial state from storage by reading the appropriate keys as shown in the following reducer files. Notice the initial state of each reducer we are reading from the local storage/session storage.

https://gist.github.com/bbachi/f9b53352ef2b78b171f7762763015d7f#file-todo-reducer-ts

https://gist.github.com/bbachi/f9b53352ef2b78b171f7762763015d7f#file-user-reducer-ts

## 6. Demo

We have implemented the solution, let’s see the demo we can see that we can preserve the state of the application once we login and create tasks.

https://miro.medium.com/v2/resize:fit:1400/1*SHK_vPqOqRGtvU4ueTez0g.gif

We can see the library ***ngrx-store-localstorage*** in action below. As soon as you add anything and send that it stores it automatically puts that state in the local storage. As we add tasks the library automatically updates the local storage.

https://miro.medium.com/v2/resize:fit:1400/1*iCZIERzOAPQJ934MIaUJ7Q.gif

## 7. Summary

- The problem with these SPAs is that the single page is loaded in the browser once and then the framework will take care of all the routing among pages and gives the impression to the user that it is a multi-page application.
- When you refresh your page in the browser that single page called index.html is reloaded and you will lose the entire state of the application.
- You can clone the entire project from here (https://github.com/bbachi/angular-ngrx-example).
- The solution for the above problem is that when you refresh the page or reload the page you are saving the application state in the local storage and retrieve it after the reload and populate the reducers.
- If the reducers are populated with the current state all the components subscribed to the store receive the data. This is what we call regydration of NGRX store.
- You can define a separate file which handles all the storage related activities. It has the following operations such as getting current state, get item, get item by key, save state, save item, etc.
- Install the library called ngrx-store-localstorage as a dependency with this command npm install ngrx-store-localstorage –save
- Once you installed this one you must register the reducers with it so that the library automatically syncs that particular reducer with the localstorage. You don’t have to do anything explicitly.
- We have an initial state in each reducer as below you need to populate the initial state from storage by reading the appropriate key.

## 8. Conclusion

In conclusion, this paper has thoroughly explored the challenges and solutions associated with maintaining the state of Single Page Applications (SPAs), particularly focusing on the integration of NGRX store with local storage in Angular frameworks. We have demonstrated that while SPAs offer enhanced user experiences through their dynamic content loading and transition capabilities, they also introduce significant challenges in state management, especially during browser refreshes or restarts. The traditional approach of making repeated API calls to restore state can strain server resources and degrade performance, especially with increasing user demands. Conversely, leveraging local storage as a state preservation strategy provides a more efficient and user-friendly alternative, minimizing server load and improving the responsiveness of web applications.

By implementing the NGRX store combined with local storage, developers can create more resilient and efficient Angular applications. This method ensures that the state of the application is preserved across sessions, thereby providing a seamless user experience while reducing server requests and load times. Practical examples and real-world applications discussed in this paper illustrate the effectiveness of this approach. Future research could focus on optimizing these techniques for larger-scale applications, exploring the security implications of local storage, and investigating additional methods for state management in the evolving landscape of web development. The integration of NGRX store and local storage presents a significant step forward in SPA development, offering a scalable, efficient solution to one of the most pervasive challenges in web application design.

## References

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