# NGRX-Redux

TABLE OF CONTENTS

Introduction 3

What is state management? 3

What is Redux? 4

What is NGRX? 4

Working of NGRX 5

Implementing NgRx store 6

Output 9

Redux Chrome extension 11

Steps to use redux in project 11

Monitoring redux Devtools 11

Conclusion 13

Aashi Barodiya

Logo

Description automatically generated

# Introduction

State management is the process of managing the states of user controls. It helps developers build large-scale applications with heavy data communications while sustaining high application performance. In the Angular world, there are two leading state management libraries we see teams adopting NGRX and NGXS. Both are modelled on redux, and will require defining, dispatching, and handling actions.

## What is State Management?

State management is a key component when building applications. There are various approaches by which we can manage the state in an Angular application, each with its pros and cons. Store is RxJS powered global state management for Angular applications, inspired by Redux. Store is a controlled state container designed to help write performant, consistent applications on top of Angular.

When it comes to Angular, NgRx are the most-used libraries for state management, and they have some unique features that make developers’ work easy.

## What is Redux?

Redux is defined as an open-source JavaScript library that works to manage and centralize the state of an application. When combined with Angular and React, it is done with the intention of building user interfaces, like Facebook's Flux architecture. Redux is a reactive state management library developed by Facebook and used in the React library. This library is based on the flux pattern. The main difference between Flux and Redux is how they handle actions, in the case of Flux, we usually have multiple stores and a dispatcher, whereas Redux has a single store, which means a dispatcher is not needed.

Now, let's see a bit how Redux works with Angular. To achieve this, the Ngrx library must be used, it provides the necessary components to successfully implement this pattern and state management.

## What is NGRX?

[NgRx](https://ngrx.io/) is a framework for building reactive applications in Angular. It is a group of Angular libraries for reactive extensions and state management. It is inspired by the Redux pattern - It makes Angular development easier by simplifying the application’s state in objects and enforcing unidirectional data flow. At a high level, NgRx stores a single state and uses actions to express state changes.

The NgRX Store imports the state management concepts from Redux and adds to them RxJS to provide an observable means of communicate on throughout the Store APIs. It excels in managing complex states, making it ideal for applications with a lot of user interactions and multiple data sources - e.g., create a simple representation of what the state should look like, update its value, monitor the state when the value changes, and retrieve the values of the state.

Working of NGRX:

The following diagram shows the state management lifecycle in NgRx. NgRx is made up of 5 main components - Store, Actions, Reducers, Selectors, and Effects.

Database 


Database

STORE

Selector

Services

Reducer

Components

Effects

Actions

NgRx uses the Redux concept of unidirectional data flow, where all application data goes through the same lifecycle. This unidirectional data flow makes the application's state more predictable and thus easier to understand. This flow only applies to the state management layer and is not to be confused with the unidirectional data flow of the presentation layer.

**Store**

You can think of this as a client-side database. The store in NgRx acts as the application's single source of truth. It reflects the current state of the app. The store is a Redux-inspired state management system that enables you to use observables to manage state in an Angular application. The primary advantage to using the NgRx store is the ability to store all state in a single tree that is accessible from any part of the application. NgRx store is mainly for managing global state across an entire application. In cases where you need to manage temporary or local component state, consider using NgRx Component store.

**Actions**

Actions express unique events that happen in our application. These events range from application lifecycle events, user interactions, to network requests. Actions are how the application communicates with NgRx.

**Reducers**

State changes are handled by pure functions called reducers that take the current state and the latest action to compute a new state. Reducers are responsible for handling transitions between states. Reducers react to the actions dispatched and executes a pure function to update the Store. Pure functions are functions that are predictable and have no side effects. Given the same set of inputs, a pure function will always return the same set of outputs.

**Selectors**

Selectors are pure functions used to select, derive, and compose pieces of state pure functions for getting slices of the state from the Store. Selectors are how our application can listen to state changes.

**Effects**

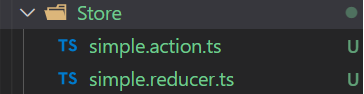
Effects handle the side effects of each Action. These side effects range from communicating with an external API via HTTP when a certain Action is dispatched to dispatching another Action to update another part of the State.

# Implementing NgRx Store

* To install the NgRx Store into your angular project by running the following command in the terminal.

**npm install @ngrx/core @ngrx/store –save**

* Create a new folder store with simple.action.ts and simple.reducer.ts



* Let’s start by building the simplest ngrx pattern possible. This app state will consist of one single variable - a hello world message in English. Then we will create three actions that will translate this message into Hindi or French or Spanish.

**app.module.ts**

For running your application, add these following imports in your application main module:

Graphical user interface, website

Description automatically generated

The forRoot method is invoked in the AppModule in the application to initialize the Store and provide the initial reducers/actions/state configuration.

Text

Description automatically generated with medium confidence

**simple.reducer.ts**

Now let’s create the reducer in a new file called src/app/modules/simple.reducer.ts.

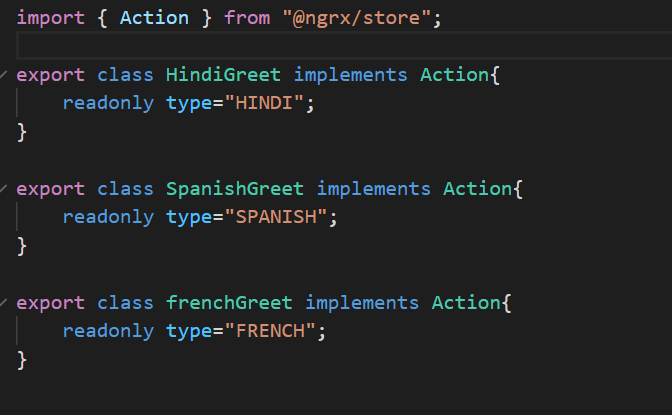
It is conventional to give action names all caps in redux, such as const SOME\_ACTION = 'SOME\_ACTION'

The reducer is just a function that runs a switch statement over possible actions that uses the old state to create a new state. Here we have three actions HINDI, SPANISH and FRENCH. If the reducer receives one of these actions, it simply converts the state string to a new value.

Text

Description automatically generated

**Simple.actions.ts:**



**app.component.ts**

Now we need a way to present and change the state in the UI. Here are a few key points about this code.

When using the ngrx Store class, it is necessary to give it a TypeScript interface that corresponds the object we passed to the NgModule. In this example, our AppState interface handles this task with it’s one message property.

A variable for message$ is set as an Observable on the component by calling **this.store.select('message')**.

We trigger state changes by sending actions to the reducer with **this.store.dispatch('ACTION').**

Text

Description automatically generated

Text

Description automatically generated

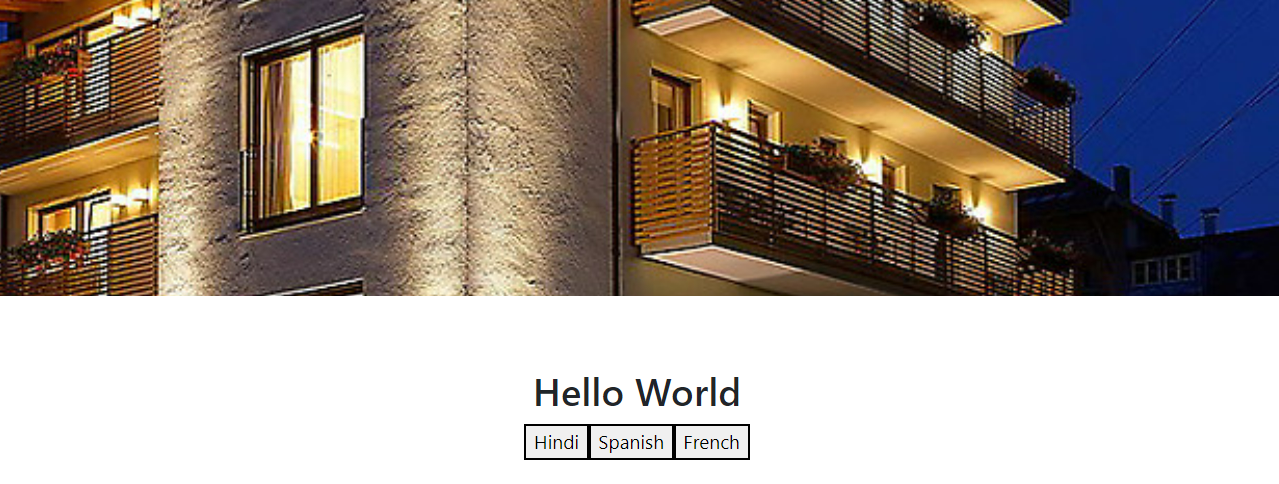
Now we can subscribe to the Observable in the HTML and trigger changes with button click events:

Text

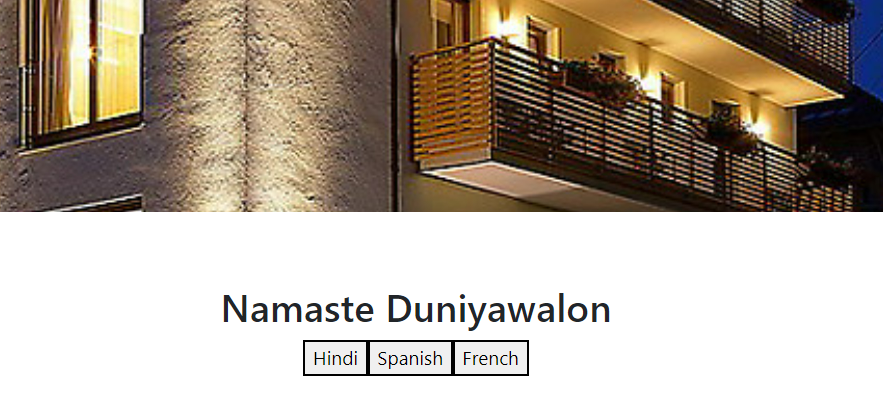
Description automatically generated

# Output

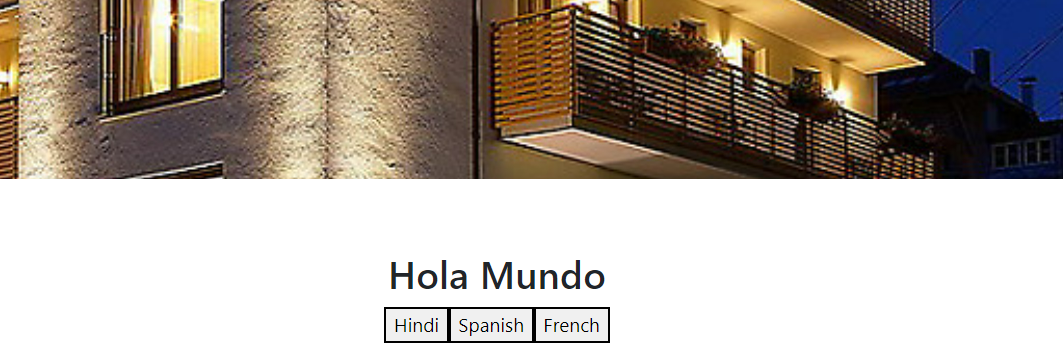
Actual output created in our project:



While clicking on the Hindi button:



On clicking on Spanish button:



On clicking on French button:

Graphical user interface

Description automatically generated

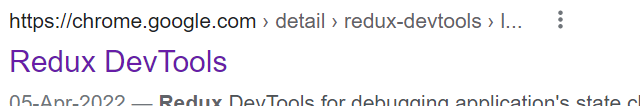
These is what we created a simple state management code.

Redux DevTools

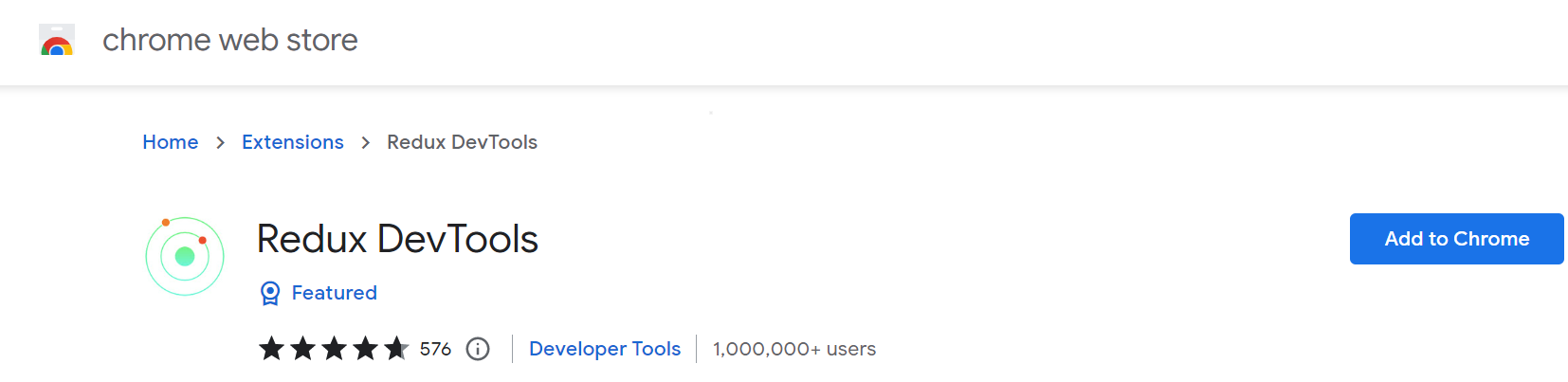
If you want to add chrome extension, we can use Redux DevTools for debugging application's state changes. The extension provides power-ups for your Redux development workflow. Apart from Redux, it can be used with any other architectures which handle the state.

**Steps to add extension redux devtools:**

Search for redux chrome extension on google, click on the first link



It will redirect on this page, as shown below

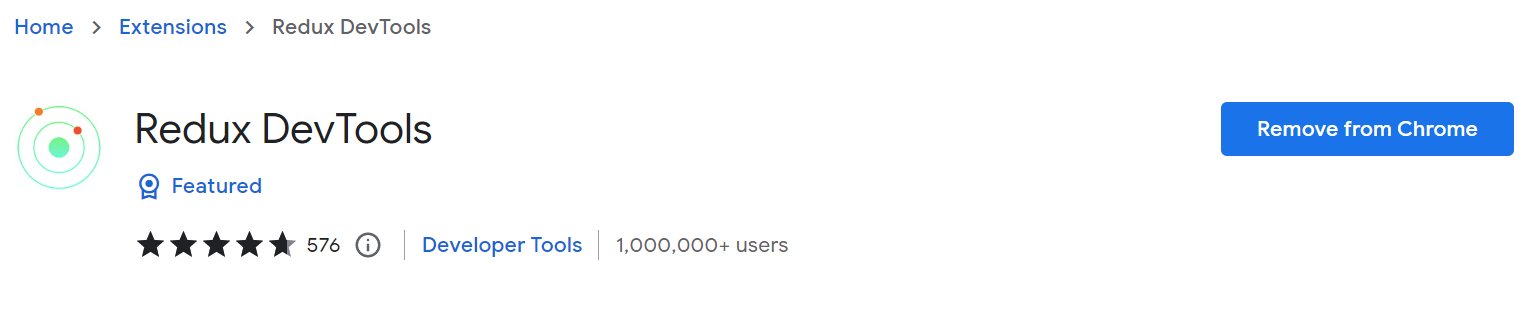


On clicking Add to chrome, it will ask with pop-up message for Add “Redux DevTools”. Just click on Add extension it will be added.

Graphical user interface, text, application, chat or text message

Description automatically generated

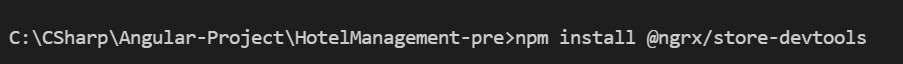
After adding, this screen will be displayed



# Steps to use redux in project

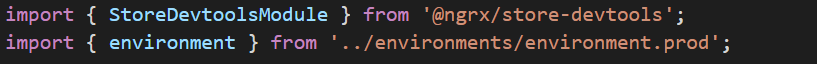
## **Npm install the ngrx StoreDevtools**

* npm install the store dev tools that expose our reducers to the chrome extension.



## **Register the StoreDevtoolsModule**

* Register the StoreDevtoolsModule in our AppModule.
* Name our app in the config to know which app we are dealing with in the browser extension.
* Also disable all the expensive compute and memory heavy functions of the devtools when we are in production to only allow logging.



Graphical user interface, text

Description automatically generated

* Run the app and explore the dev tools in the browser

# monitoring redux devtools:

We can see the output by clicking on extension button in chrome

Graphical user interface, application

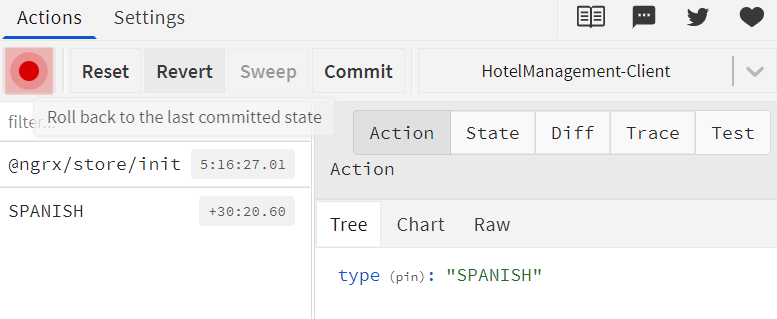
Description automatically generated

The initial output screen after clicking on Redux DevTools,

Graphical user interface, text, application

Description automatically generated

After doing it Spanish, the change can be seen here



Change in state:

Graphical user interface, text, application

Description automatically generated

Change in difference

Graphical user interface, application

Description automatically generated

# Conclusion

With the intention of providing some insight into the NgRX Store capabilities, this documentation sheds light on the application state concepts, on how to use the Store in Angular Root and demonstrates how an existing Angular application can be migrated to use the NgRX Store to manage its state.

It could well be that Store architectures initially became popular in the React world provide a client container for temporary UI state, provide a cache for avoiding excessive HTTP requests, provide a solution for concurrent data modification by multiple factors.

The same thing is happening now in the Angular world, and the outcome could be the same. Working with large-scale applications requires good architecture and organized structure. State management tools such as NgRx can help you maintain a readable codebase as your app scales.

Whether you’re an Angular developer or not, understanding the basic concepts behind state management will make it easier to use tools such as Redux in your projects.