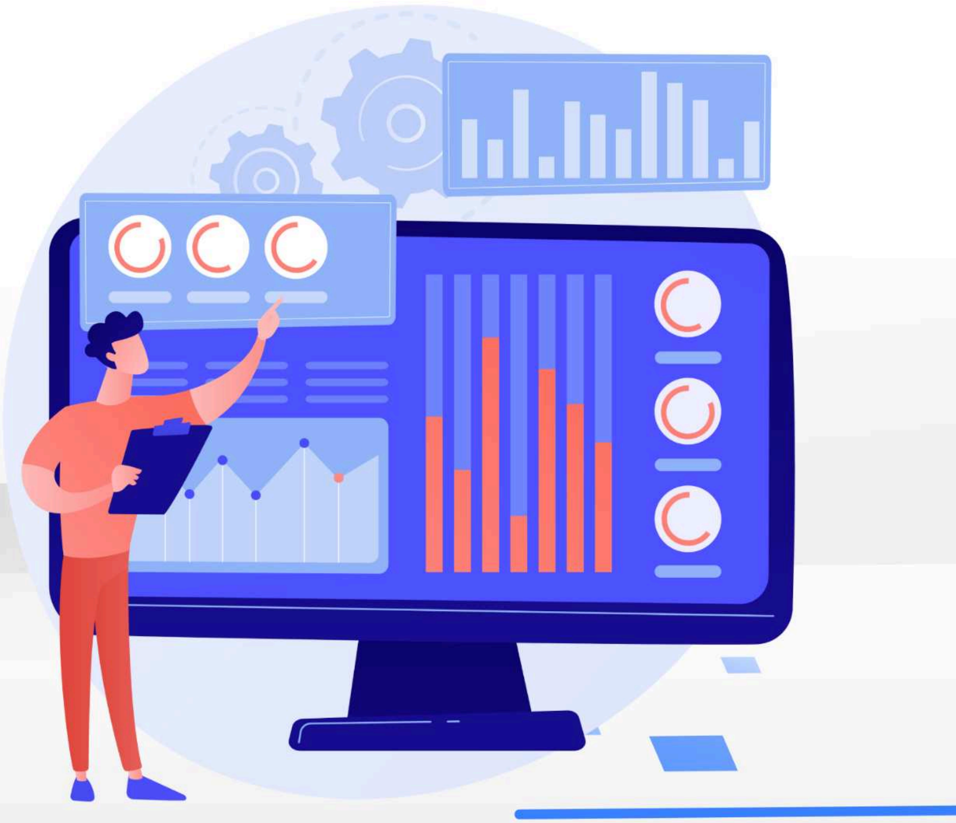
NgRx SignalStore

An effortless solution for state management



Abdelfattah Ragab

NgRx SignalStore

An effortless solution for state management

Abdelfattah Ragab

Source code

The source code is available on the book's website
https://books.abdelfattah-ragab.com

Introduction

Welcome to the book “NgRx SignalStore: An effortless solution for state management”.
Managing state in front-end applications has always been a big problem for front-end developers.
The most comprehensive solution we have always used has been the NgRx Store. However, this is not an easy solution for many developers and in most cases requires extensive initial setup.
With NgRx SignalStore, you can get all the benefits of NgRx state management without the setup complexity.
It's even easier to maintain because you only have one file per store that contains everything about the state.
Let us take a look at it in action.
Let us get started.

What is State Management?

State management in front-end applications refers to the systematic handling of the application's state, which includes all data and variables that define the current state of the application at a given point in time. This includes user input, API responses and any other dynamic data that may change when interacting with the application.

NgRx Signals

NgRx Signals is a standalone library that provides a reactive state management solution and a set of utilities for Angular Signals.

Installation

```
ng add @ngrx/signals@latest
```

SignalStore

NgRx SignalStore is a full-featured state management solution that provides a robust method for managing application state. With its native support for signals, it provides the ability to define stores in a clear and declarative way. The simplicity and flexibility of SignalStore, coupled with its thoughtful and extensible design, make it a versatile solution for effective state management in Angular.

Backend API Services

keep the API services you use to connect to the backend server as they are.
They will not change.

Books Website Example

As an example, I will create the backend and frontend for a website where users can download books.

Create the NestJS Application

```
nest new ngrx-signalstore-be
```

Create the Angular Application

```
ng new ngrx-signalstore-fe
```

The NestJS Application

Let’s finish the backend app first.

Create a new “books” module

```
nest generate module modules/books
```

Create a new “books” service

```
nest generate service modules/books
```

Create a new “books” controller

```
nest generate controller modules/books
```

The “books” service

Declare methods for getBooks, getBookById and getFileAsStream.
It will also have a private array of books that we will serve to the client.

```
import { Injectable } from '@nestjs/common';
import * as fs from 'fs';

@Injectable()
export class BooksService {
  private books: any[] = [
    {
      id: 1,
      title: 'Angular Shopping Store',
      cover:
'https://i.ibb.co/GP6Lydb/angular-shopping-stor
e.jpg',
    },
    {
      id: 2,
      title: 'Angular Portfolio App
Development',
      cover:
'https://i.ibb.co/YNx9ZPb/angular-portfolio-app
-development.jpg',
    },
    {
      id: 3,
      title: 'Responsive Layouts: Flex, Grid
and Multi-Column',
      cover:
'https://i.ibb.co/7vCtF4J/responsive-layouts.jp
g',
    },
  ];

  async getBooks() {
    return this.books;
  }

  async getBookById(id: any) {
    if (!id) return undefined;
    return this.books.find((t: any) => t.id ==
id);
  }

  async getFileAsStream(fullPath, response) {
    try {
      const stat = fs.statSync(fullPath);
      const fileBuffer =
fs.readFileSync(fullPath);
```

```
      response.setHeader('Content-Length',
stat.size);
      response.setHeader('Content-Type',
'application/octet-stream');

      const chunkSize = 64 * 1024;
      let bytesSent = 0;

      while (bytesSent < fileBuffer.length) {
        const chunk =
fileBuffer.slice(bytesSent, bytesSent +
chunkSize);
        bytesSent += chunk.length;

        const progress = (bytesSent /
stat.size) * 100;
        console.log(`Download progress:
${progress.toFixed(2)}%`);

        await new Promise((resolve) =>
setTimeout(resolve, 100)); //to slow down the
download process

        response.write(chunk);
      }

      response.end();
    } catch (error) {
      console.error('Error fetching file:',
error);
      response.status(500).send('Error fetching
file');
    }
  }
```

```
}
```

The “books” controller

```
import { Controller, Get, Param, Res } from
'@nestjs/common';
import { BooksService } from './books.service';
import { delay, of } from 'rxjs';
import * as path from 'path';

@Controller('books')
export class BooksController {
  @Get()
  async getBooks() {
    const books = await
this.booksService.getBooks();
    return await of(books).pipe(delay(2000));
  }

  @Get(':id')
  async getBookById(@Param('id') id) {
    const book = await
this.booksService.getBookById(id);
    return await of(book).pipe(delay(2000));
  }

  @Get(':id/stream')
  async getFileAsStream(@Param('id') id, @Res()
response: any) {
    const fullPath = path.resolve(__dirname,
'../../../', 'book.zip');
```