***CRUD application with ngrx store and angular***

**Introduction:**

Today we are going to build a crud bookstore application with the help of ngrx store and angular concepts. Here, in this app you can store the information of your books and we will the four rules of crud. We will also api intergration to access,display,modify data in our webpage.So, wihtout any late let’s get started.

**Why we should we use ngrx store?**

The NgRx Store is a powerful state management solution for Angular applications. It is built on top of RxJS, which is a popular reactive programming library. NgRx Store provides a predictable and centralized state management approach, which offers several benefits:

1. **Unidirectional data flow:** NgRx enforces a unidirectional data flow in your application, which makes it easier to understand how data changes and propagates throughout the system. This improves the overall maintainability and testability of your codebase.

2. **Centralized state:** The NgRx Store provides a central repository for storing and managing the state of your application. This eliminates the need for scattered local component state and facilitates better data sharing and consistency across components.

3. **Immutable state:** The state in NgRx Store is immutable, meaning it cannot be directly modified. Instead, you create new state objects whenever a change occurs. This immutability simplifies change detection and enables better performance optimizations, as you can rely on reference equality checks.

4. **Time-travel debugging:** NgRx Store integrates with the Redux DevTools extension, which allows you to record and replay state changes over time. This powerful debugging tool helps you track down issues, understand how your application's state evolves, and reproduce specific scenarios.

5. **Middleware support:** NgRx Store supports middleware, which enables you to intercept and modify actions and state updates. This middleware functionality can be useful for implementing features like logging, caching, asynchronous operations, or API interactions.

6. **Scalability:** As your application grows in complexity, managing state can become challenging. NgRx Store provides a scalable solution by offering patterns and abstractions for organizing and structuring your state management code. It encourages modularization, separation of concerns, and maintainable code practices.

7. **Integration with Angular ecosystem:** NgRx Store is specifically designed for Angular applications and integrates seamlessly with other Angular features, such as Angular Router, Forms, and Effects. It aligns with the reactive programming paradigm promoted by Angular and leverages the power of RxJS observables for managing state changes.

Overall, using NgRx Store in your Angular applications provides a robust and well-established approach to state management, offering benefits like improved maintainability, testability, scalability, and debugging capabilities. However, it's important to note that NgRx Store might introduce some additional complexity, especially for small or simple applications, so it's crucial to evaluate whether the benefits outweigh the added overhead for your specific project.

**Prerequisites:**

1. We need to ngrx store and ngrx effects in our application

2. Open your project folder with vs code editor and open terminal and install ngrx store and effects with below commands.

3. “npm install @ngrx/store@ --angular version”

4. “npm install @ngrx/effects@ --angular version”

5. As we used bootstrap as our primary css famework in our project becuase it is so easy to use and it is easy to make responsive websites with bootstrap.

6. To install bootstrap “npm install bootstrap”

7. Make sure add styles and script files in angular.json file to make bootstrap work.

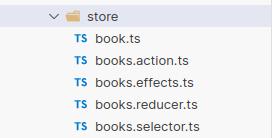


8. Now, lets start building the project.

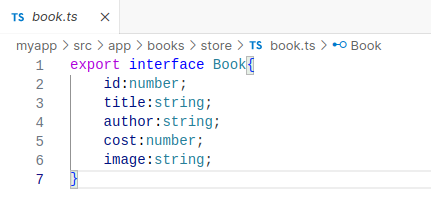
**Procedure to code:**

1. First, create a angular project with your desired name in your project folder.

2. Now, after our project created we will create a folder with name of store and we wiil create four files with below name as shown in the image



3. After creating the store folder now lets start with our program. Let’s first define the structure of our book in book.ts file



4. Now, we will create the all the actions which we will use in our program to create. The screenshot is below add the code and refer to the explanation.

**I) invokeBooksApi:**

This action is used to invoke the fetch API for retrieving books.

**II) bookApiFecth:**

This action is dispatched when the books fetch API is successful.

Payload: { allBooks: Book[] } (an array of Book objects representing all the fetched books)

**III) invokeSaveBookApi:**

This action is used to invoke the save book API for creating a new book.

Payload: { payload: Book } (the Book object to be saved)

**IV) saveBookApiSuccess:**

This action is dispatched when the save book API is successful.

Payload: { response: Book } (the Book object returned by the API after saving)

**V) invokeupdateBookApi:**

This action is used to invoke the update book API for modifying an existing book.

Payload: { payload: Book } (the updated Book object)

**VI) updateBookApiSuccess:**

Description: This action is dispatched when the update book API is successful.

Payload: { response: Book } (the Book object returned by the API after updating)

**VII) invokedeleteBookApi:**

Description: This action is used to invoke the delete book API for removing a book.

Payload: { id: number } (the ID of the book to be deleted)

**VIII) deleteBookApiSuccess:**

Description: This action is dispatched when the delete book API is successful.

Payload: { id: number } (the ID of the deleted book

5. Now after creating the action file successfully, create the effects in order work with all the api instructions . but we will design this effects file in such a way that it will implement a API optimization method to reduce unwanted api calls from the server**(books.effects.ts)**

6. The **loadAllBooks$** effect listens for the invokeBooksApi action. It checks if the books are already available in the store using the selectBooks selector. If the books are not present, it calls the get() method from the bookService to fetch the books and dispatches the bookApiFecth action with the retrieved data.

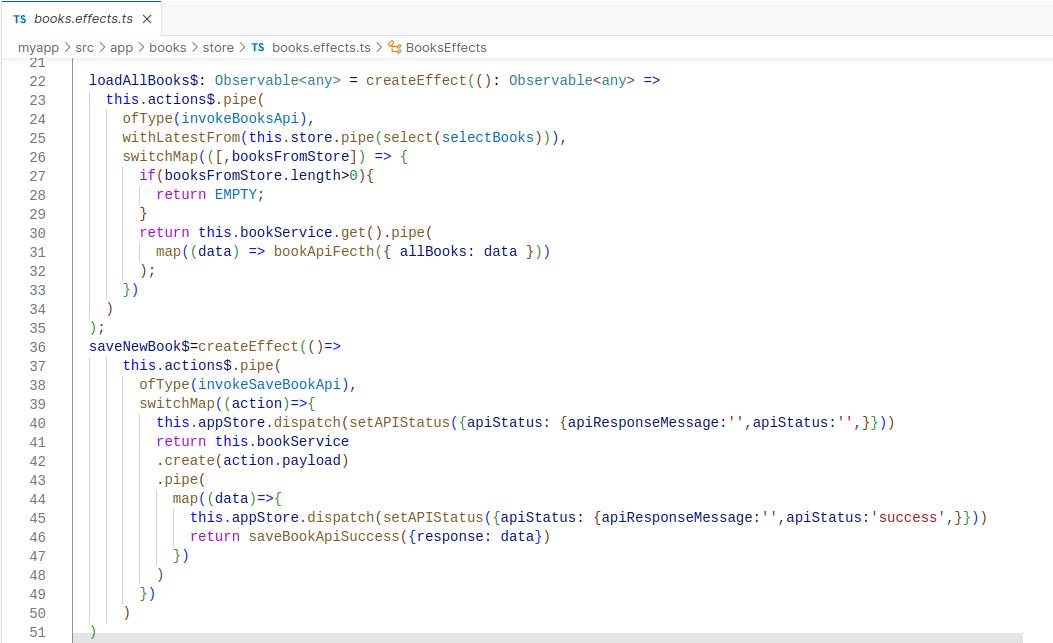
7. Below is the screen shot of **books.action.ts**



8.The **saveNewBook$** effect listens for the invokeSaveBookApi action. It dispatches the setAPIStatus action to set the API status before calling the create() method of the bookService to save the new book. After a successful response, it dispatches the setAPIStatus action again with a success status and the saveBookApiSuccess action with the saved book data.

9. The **updateBook$** effect listens for the invokeupdateBookApi action. Similar to the previous effect, it dispatches the setAPIStatus action, updates the book using the update() method of the bookService, and then dispatches the setAPIStatus action with a success status and the updateBookApiSuccess action with the updated book data.

10.I have attached the screenshot of **books.effects.ts** file for reference.

11.Since, Here the sreenshot size is becoming to big so i have just took the methods of loadallbooks and savenewbook. Don’t worry i will provide you reference video link for this project and my github link for the entire source code.

12.The **deleteBook$** effect listens for the invokedeleteBookApi action. It dispatches the setAPIStatus action, calls the delete() method of the bookService to delete the specified book, and then dispatches the setAPIStatus action with a success status and the deleteBookApiSuccess action with the ID of the deleted book.

These effects use the actions$ observable to listen to specific actions and perform the necessary operations. They also interact with the appStore and store instances of the NgRx Store for dispatching actions and selecting state.

***Now let’s start creating a file book.reducer.ts:***

The reducer manages the state of books in the application by handling specific actions and returning updated state accordingly

Below explanation and reference screenshot for reducer file.

The initial state is defined as an empty array intialState to represent an initial collection of books.

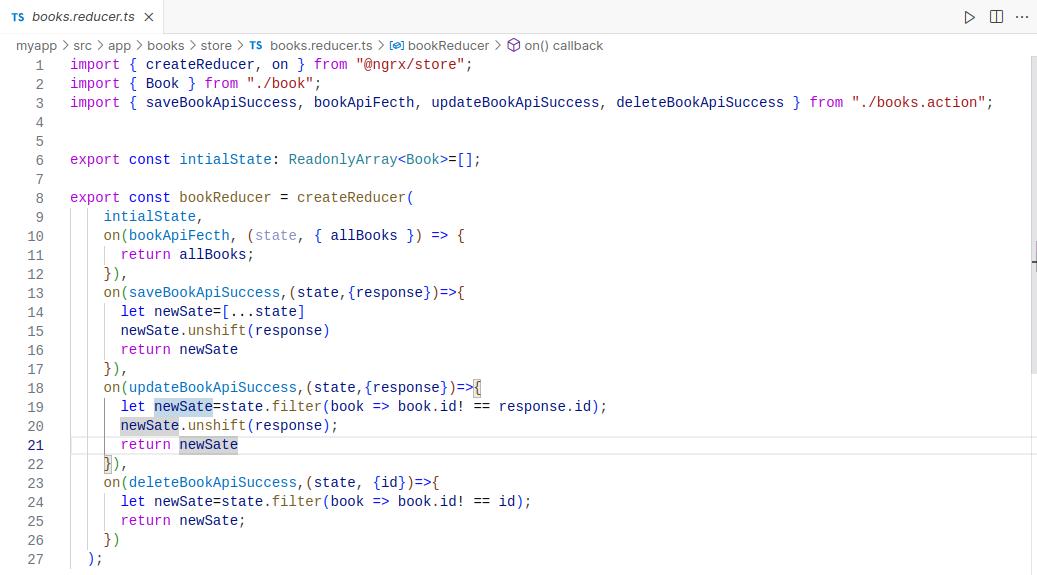
The **bookApiFecth** action updates the state by replacing the existing books with the allBooks payload received from the action.

The **saveBookApiSuccess** action adds the response (the newly saved book) to the beginning of the state array. It creates a new state array by spreading the existing state and adding the new book at the beginning.

The **updateBookApiSuccess** action updates the state by removing the book with the same ID as the response from the state array. It creates a new state array by filtering out the book with the matching ID, and then adds the response (the updated book) at the beginning of the new state array.

The **deleteBookApiSuccess** action removes the book with the specified id from the state array. It creates a new state array by filtering out the book with the matching ID.

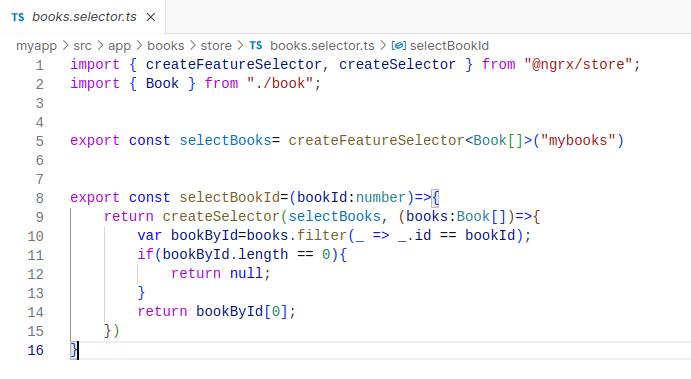
The **createReducer** function combines the specified action handlers to create a reducer function that takes the current state and an action, and returns the updated state based on the handled actions.



***Creating file books.selector.ts:***

The **selectBooks** selector is created using **createFeatureSelector.** It selects the portion of the state identified by the key **"mybooks"** and returns it as an array of **Book** objects.

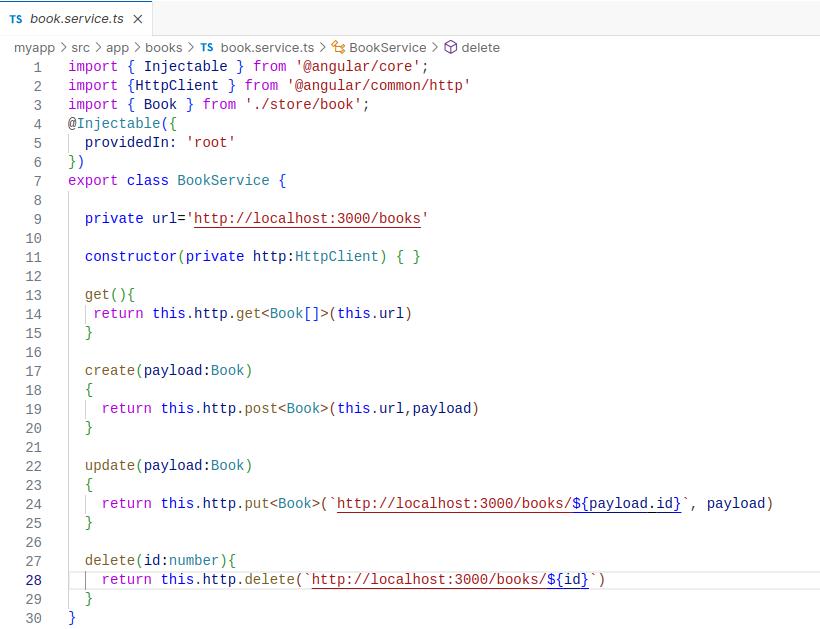
The **selectBookId** selector takes a **bookId** parameter and returns a memoized selector function created using **createSelector**. It uses the selectBooks selector to get the array of books from the state. It then filters the books based on the bookId and returns the book with the matching ID. If no book is found, it returns **null.**

13. Below is the screenshot attached for the selector file

14.Now, create a three different components with naming as home,add,edit.



15.Now, create a service file to make http request calls and different api calls which we will use in our program.



Now the creation part of our logical program is completed and from now we have implement it in our UI to make visible in webpage

1.**home.component.html:**

In this file we are going to create our navbar design and cards design, Modal popup design with help of bootstrap components

Go, to bootstrap and find out these three components and add it in our home component file

So, likewise we will implement all the logic to make these three components functionable will be written in **home.component.ts** file.

2.**add.component.html:**

In this file we are going to build reactive form to get the details of the book that will be add by the user and the screenshot is provided below .

And now we need to implement the logic of this form in app.component.ts file to make this form interractive

So, in this file we have created a method called **“save()”** which will invoke when user clicks on create button and this function will dispatch a method **invokeSaveBookApi** from the store and adds the data in the form to api by post method in the api and it navigates back to home page after clicking on create button.

Overall this component is used to add a new book details including image to our bookstore.

Now we will add edit button functionality to our book details since user can edit the details of the book.

**Creation of edit.component.html file:**

**edit.component.ts:**



After completing this all things save this files and now we are going to implement the most important thing in our that is api file creation using json server

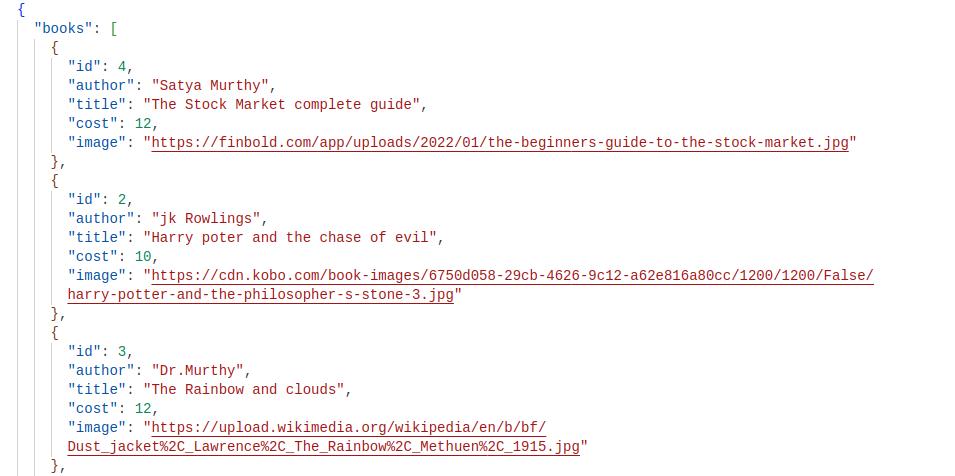
**Installation:**

By typing the below command install the json-server in your local system to create our json file in the system.

“npm install -g json-server”

After installing it in the local machine then open “pacakage.json” file and add below code in your script section. Here, we are just creating a npm command to run json-server.

4. Now , create some data in the db.json file in the below manner to display books and you can details and images according to your wish.



Finally, We have configured everything what we needed for our program to run.

**Final Steps:**

Run your angular project by using “ng serve” in the terminal.

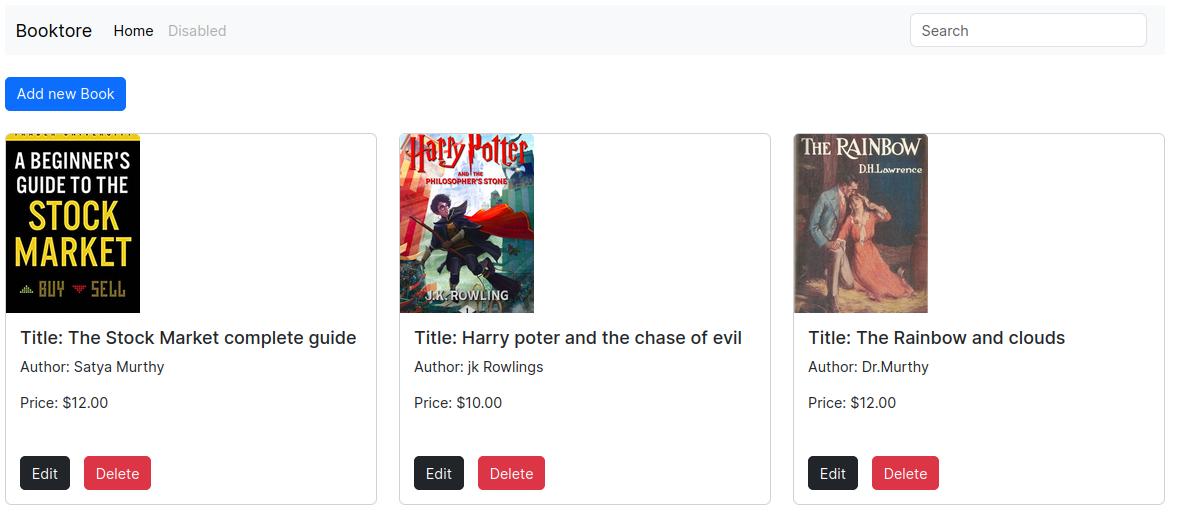
And run your json server by typing “npm-run” in the terminal.

If you perform everything well the project will run without any errors and if any errors occurs for you then here complete making youtube video for this project and github link for the source code of this project.

**YOUTUBE:** [**https://www.youtube.com/watch?v=G0jPyuXjxKk**](https://www.youtube.com/watch?v=G0jPyuXjxKk)

**GitHub:** [**https://github.com/BANUTEJAGOUD/Internship\_Daily\_Code**](https://github.com/BANUTEJAGOUD/Internship_Daily_Code)

**Output:**

This is the final output of our project. Hope, you all learned some thing from this project.

***Thank you...***