Sagas

Overview

In this lab you'll enhance your "library" web application so that it gets books and films data from a REST service, rather than being hard-coded. You'll use Sagas to instigate the REST calls and to feed the responses back to Redux, so that Redux can make the data available to all components in the React app.

Source folders

- ReactDev\Student\11-Sagas
- ReactDev\Solutions\11-Sagas

Roadmap

Here's a brief summary of the exercises in this lab. More detailed instructions follow later in this lab document:

1. Familiarization with the REST service
2. Getting started with the 'student' React app
3. Preparing to use Sagas
4. Defining additional action types
5. Implementing additional action creator functions
6. Enhancing reducer functions
7. Implementing Saga functions
8. Implementing Saga initialization code
9. Running the application

Exercise 1: Familiarization with the REST service

Open a Terminal window and go to the ***student*** folder for the REST service:

```
ReactDev\Student\11-Sagas\server
```

In this folder, run the following commands to install NPM libraries and to run the REST service application:

```
npm install
npm start
```

Then open a browser and ping the following URL, to get back an array of books as a proof of concept:

```
http://localhost:8080/api/books
```

Then ping the following URL, to get back an array of films:

```
http://localhost:8080/api/films
```

Keep the REST server application running, because the React application will need to talk to it shortly…

Exercise 2: Getting started with the 'student' React application

Now go to the student folder for the React application:

```
ReactDev\Student\11-Sagas\library-app
```

The code is similar to before, but there are some immediate differences you'll notice:

- We've added a folder named sagacode, with a file named mysagas.ts. You'll implement your Saga functions here shortly.
- We've deleted the DataProvider class. Previously you used this class to get hard-coded data for books and films. You won't be doing that anymore - instead you'll get the data from the REST service, which is much more realistic.

Exercise 3: Preparing to use Sagas

The application doesn't currently support Sagas. To add support, add the following dependencies to package.json:

```
{
  "dependencies": {
    "redux-saga": "^1.1.3",
    "@types/redux-saga": "^0.10.5",
    …
  },
  …
}
```

Once you've made this change, you'll need to install these packages. Open a Terminal window and go to the following folder:

```
ReactDev\Student\11-Sagas\library-app
```

Then run the following command:

```
npm install
```

Exercise 4: Defining additional action types

Open reduxcode/types.ts in a text editor.

This file defines an enum named ActionType, which specifies all the types of actions that Redux must handle. Currently it only defines a single action type named FORMAT_ITEMS. Add 4 more action types as follows, and assign a simple string for each one:

- GET_BOOKS - Tells Redux to initiate a "get books" request
- GET_BOOKS_FINISHED - Tells Redux the "get books" request has finished
- GET_FILMS - Tells Redux to initiate a "get films" request
- GET_FILMS_FINISHED - Tells Redux the "get films" request has finished

Exercise 5: Implementing additional action creator functions

Open reduxcode/actions.ts in a text editor.

This file defines functions that create action objects. Currently it only defines a single action creator function named changeFormat(). Define 2 additional action creator functions as follows:

- getBooks()
  Returns an action object with a type property of ActionType.GET_BOOKS, representing an action object that says "get me the books, dude".
- getFilms()
  Returns an action object with a type property of ActionType.GET_FILMS, representing an action object that says "get me the films, baby".

Exercise 6: Enhancing reducer functions

Open reduxcode/reducers.ts in an editor.

This file defines reducer functions that tell Redux how to change state in its store. Enhance the books() and films() reducer functions as described by the TODO comments in the code, so that they scoop-up REST response data for books and films and convert the raw JavaScript objects into fully-fledged Book and Film objects instead.

Exercise 7: Implementing Saga functions

Open sagacode/mysagas.ts in an editor and examine the existing code. Note we've already implemented a function to call the REST service to get back data.

Complete the implementation of all the functions in this file, as described by the detailed TODO comments in the code.

Exercise 8: Implementing Saga initialization code

Open App.tsx in a text editor and take a look at the code. Add code where indicated by the TODO comments, to initialize Saga middleware in the application.

Exercise 9: Running the application

When you've made all these changes, you're ready to run the React application to see if it all works. If you get any problems, open Developer Tools in the browser and see if there are any error messages in the console window.