Redux

Overview

In this lab you'll add Redux support to the "library" web application.

The purpose of Redux is to store and maintain data that's needed by multiple components across the web application. In this way, if the data changes, all components in the web application can access that data.

Note: If you have some data that applies just to a single component, you don't need Redux for that; you can just store that data as "state" in that component. For example, the "likes" count is only used in the LikePanel component, so we just store this value in the LikePanel's state; we don't need the hassle of storing this in Redux.

Source folders

- ReactDev\Student\10-Redux
- ReactDev\Solutions\10-Redux

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 'solution' web app
2. Getting started with the 'student' web app
3. Preparing to use React Redux
4. Implementing an action function
5. Implementing reducer functions
6. Implementing Redux initialization code
7. Accessing the Redux store in components
8. Dispatching actions to Redux triggered by user actions
9. Running the application

Exercise 1: Familiarization with the 'solution' web app

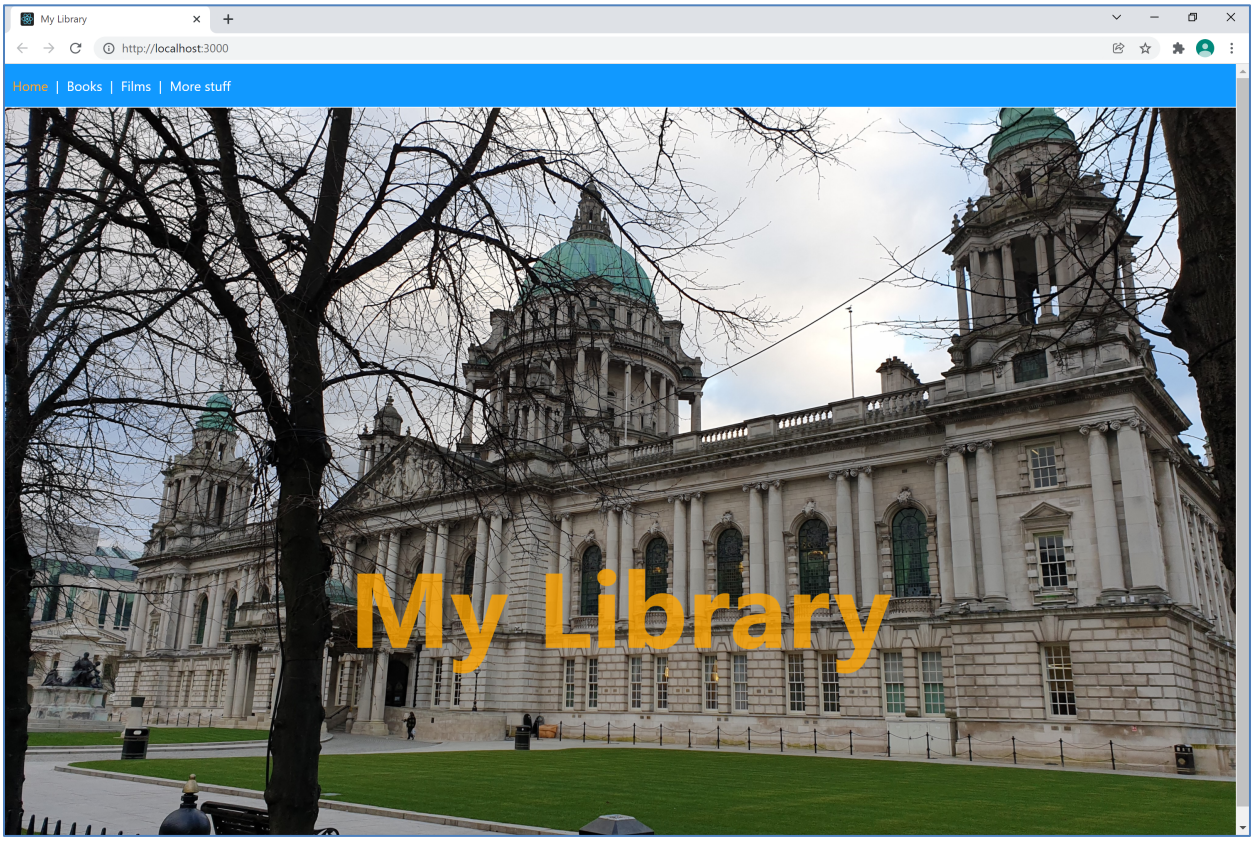
Open a Terminal window and go to the following folder:

```
ReactDev\Solutions\10-Redux\library-app
```

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

```
npm install
npm start
```

The web application appears as follows initially in the browser:



Click the *More stuff* link. Notice that it offers the user a chance to choose the format in which books/films are displayed - the user can display this data as tables, ordered lists, or unordered lists. Select a format and see how it changes how books/films are displayed.

Exercise 2: Getting started with the 'student' web app

Go to the ***student*** folder:

```
ReactDev\Student\10-Redux\library-app
```

The code is similar to before, but notice there's a reduxcode folder now. You'll put your core Redux-related code in here. There are 3 files involved:

- actions.ts
  You will define action function(s) here, to represent actions triggered by the user.
- reducers.ts
  You will define reducer function(s) here, to tell React how to update state based on the actions triggered by the user.
- types.ts
  Defines interfaces/enums that will come in handy. This file is already complete.

Exercise 3: Preparing to use React Redux

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

```
{
  "dependencies": {
    "react-redux": "^7.2.4",
    "@types/react-redux": "^7.1.18",
    …
  },
  …
}
```

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\10-Redux\library-app
```

Then run the following command:

```
npm install
```

Exercise 4: Implementing an action function

Open actions.ts in an editor.

Define an action function named changeFormat() as follows:

- The function should take a single parameter named newFormat (for example), representing the new format requested by the user (as a string).
- The function should return an action object (i.e., an object consistent with the Action interface) with 2 properties:
    - type: ActionType.FORMAT_ITEMS
    - payload: newFormat

Exercise 5: Implementing reducer functions

Open reducers.ts in an editor.

Define a reducer function named books() that will maintain the state for *books* data in Redux.

- The function should take 2 parameters:
    - The current state (i.e., array of books) in the Redux store
    - An Action object, specifying the action triggered by the user
- The function should just return the current state as-is (i.e., the application doesn't yet have any functionality to modify books).

Define a similar reducer function named films() to maintain the state for *films* data in Redux.

Define a another reducer function named format() to maintain the state of the "current format" flag in Redux (remember this is a string "TABLE", "ORDERED_ITEMS", or "UNORDERED_ITEMS"). The purpose of this reducer function is to change this value, triggered by an action by the user. Define the function as follows:

- The function should take 2 parameters:
    - The current state in the Redux store
    - An Action object, specifying the action triggered by the user
- Inside the function, write a *switch* statement to test the action.type. If it's ActionType.FORMAT_ITEMS, then return the action.payload value back to Redux. This is the new format chosen by the user.

Exercise 6: Implementing Redux initialization code

Open App.tsx in a text editor and take a look at the code. Note that we've split the *App* component into two components now (you'll see why we've done that in the next lab):

- App
- AppWrapped

Add code in the App component to implement Redux initialization, as described by the comments in the code.

Exercise 7: Accessing the Redux store in components

Open Books.tsx in a text editor and take a look at the Books component. Note that it doesn't receive any properties from its parent component anymore - the idea is that it gets the data it needs from the Redux store instead. To achieve this, call the Redux useSelector() function and then use the books and format data to do the rendering.

Open Films.tsx in a text editor and add code in a similar way, to get the data it needs from the Redux store.

Exercise 8: Dispatching actions to Redux triggered by user actions

Open MoreStuff.tsx in a text editor and take a look at the MoreStuff component. Notice we've enhanced this component now - near the bottom of the code, it displays a <select> element so the user can choose the format for displaying data (as a table, an ordered list, or an unordered list).

You need to add code where indicated by the comments, to do the following things:

- Call useSelector() to get data from the Redux store (you'll need books, films, and format).
- Call useDispatch() to enable you to dispatch actions to Redux.
- Inside the onFormatSelectionChanged() function (which is called when the user chooses a format from the <select> element), call changeFormat(newFormat) to create an action object. Then pass this action object into dispatch(), to dispatch the action object to Redux.

Exercise 9: Running the application

When you've made all these changes, you're ready to run the 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.