Redux

Goal

The goal of this lab is to manage the state of a *React* application using Redux.

The Mission

In this lab you will be updating a movie editor using React. The application will let you browse through the list of known movies and select one to edit. Once you have made changes to a movie you have the option of saving the changes or canceling them.

The starter application in this exercise contains a working movie editor. This movie editor doesn't use Redux yet. Instead, it manages all its state inside of the UI components. In this lab you will use the Redux library to improve the application.

You can start the application by running the start.bat file in the root folder. When you do a small Node.js server will start. Next the default browser will start to display the index.html page. Please note that the first time you start this will take some time as number of NPM packages will download. This requires an internet connection where the NPM registry is not blocked by a firewall.

Note: The **FINISHED** folder contains a finished version of this lab as a reference.

Type it out by hand?
Typing it drills it into your brain much better than simply copying and pasting it. Because you're forming new neuron pathways that are going to help you in the future, help them out now!

Assignment 1

Start the application in the **BEGIN** folder and familiarize yourself with the functionality.

The first task is to create a basic Redux store

Use NPM to install *redux* and *react-redux*.

```
npm install --save redux react-redux
```

Once installed, you can create a folder named **REDUCERS** which will hold all our reducers. In this folder create two files **INDEX.JS** and **MOVIES.JS**. Write the basic implementation of a movies collection reducer in **MOVIES.JS**. As we have no actions yet all this reducer will do is return the passed in state which defaults to an empty array. Require this movies collection in the **INDEX.JS** and use the Redux *combineReducers()* function to export this. Note: the reason we are using *combineReducers()* is because we will be adding more reducers later.

Create an **ACTIONS** folder and add an **INDEX.JS** file to it. This file will remain empty for now but will be used to add the *Redux* actions to.

Open APP.JS and use the Redux *createStore()* function and the previously created *reducer* to construct a new Redux store. Next require the *Provider* component from react-redux and wrap the application being rendered inside of an instance of this Provider. Make sure to pass the previously created *store* into the Provider as the *store* property.

Update the MovieList component to use the Redux store

You now have an application that is ready to start using Redux. The next step is to change the *MovieList* component to stop managing its own state and use Redux instead.

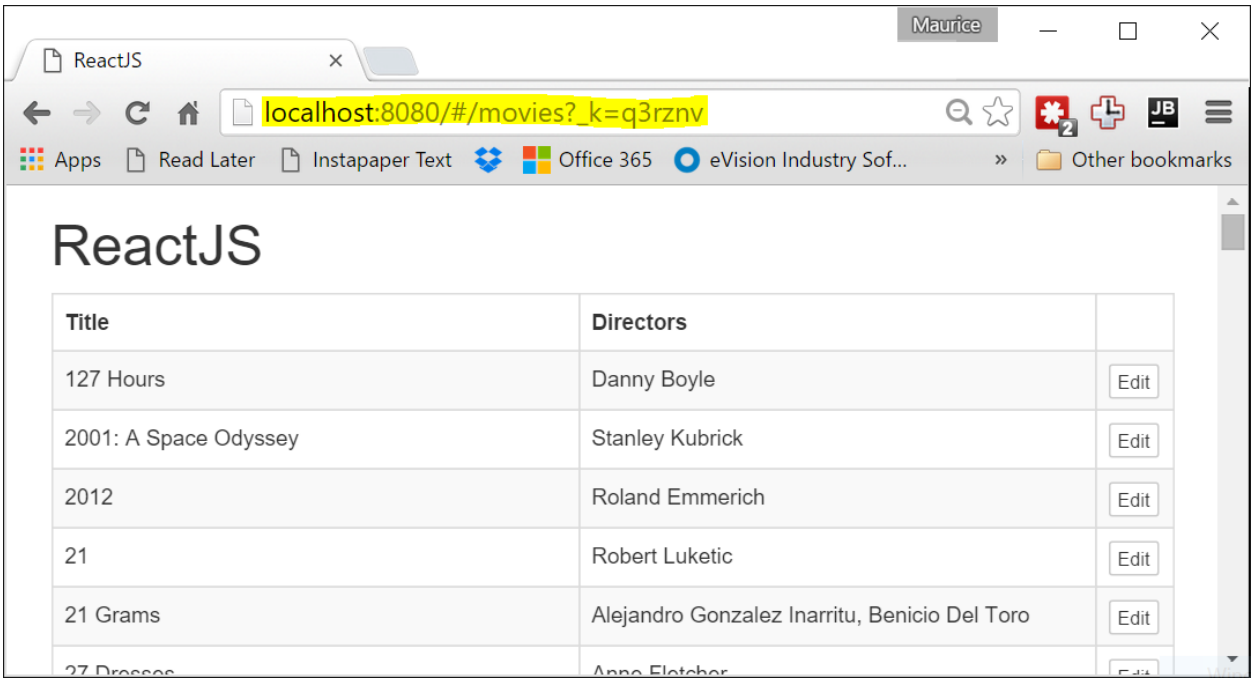
Open the ACTIONS/INDEX.JS and add a *MOVIES-LOADED* action with as payload the list of movies that are loaded from the server.

Open the REDUCERS/MOVIES.JS and handle the *MOVIES-LOADED* action by returning its *movies* as the new state.

Open the MOVIELIST.JSX and add a *mapDispatchToProps* function to dispatch the *MOVIES-LOADED* action when the movies are loaded. Add a *mapStateToProps* function to extract and return the list of movies from the Redux store state.

Next use the react-redux *connect()* function to wrap the *MovieList* component with the *mapStateToProps* and *mapDispatchToProps* functions and export this resulting component instead of the original *MovieList* component.

The last step is to update the *MovieList* component itself to no longer use its own state. Replace the *setState()* call after the ajax request to dispatch the *MOVIES-LOADED* action. Replace the *this.state.movies* with *this.props.movies* in the *render()* function. Finally, remove any remaining state references from the *MovieList* component.



Finally, make sure the application works and shows the list of movies.

Assignment 2

The second task is updating the MovieEdit component to use the Redux store.

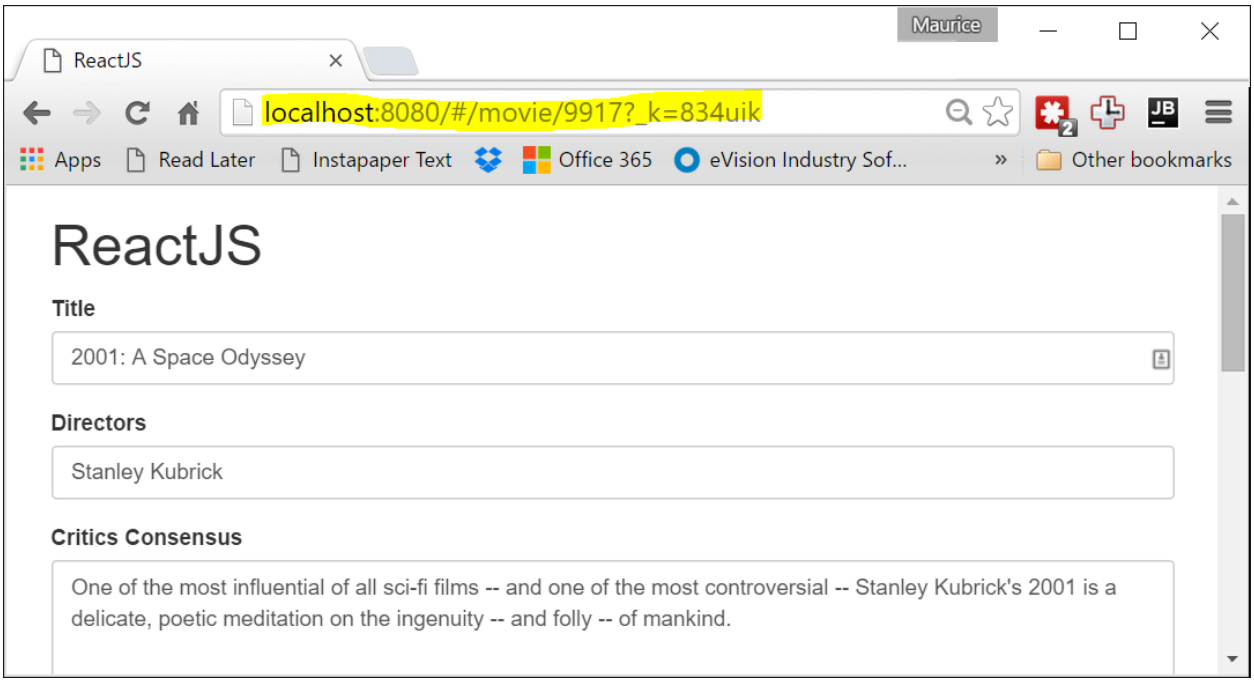
You should begin by opening **actions/index.js** and adding a *MOVIE-LOADED* action. This action will be called when a single movie is loaded and to be shown in the edit form.

Add a second reducer **currentMovie.js** with a default *state* of null. Handle the *MOVIE-LOADED* action by returning the *movie* object as the current *state*. Add the *currentMovie* reducer to the call to *combineReducers()* function in the **reducers/index.js**.

Open **movieEdit.jsx** and update the *MovieEdit* component in a similar way as you did with the *MovieList* component above. First add a *mapStateToProps()* function to return the *currentMovie*. Next add a *mapDispatchToProps()* function to dispatch the *MOVIE-LOADED* action. Next use the react-redux *connect()* function to wrap the *MovieEdit* component with the *mapStateToProps* and *mapDispatchToProps* functions and export this resulting component instead of the original *MovieEdit* component.

The last step is to update the *MovieEdit* component itself to no longer use its own state. Replace the *setState()* call after the ajax request to dispatch the *MOVIE-LOADED* action. Replace the *this.state.movie* with *this.props.movie* in the *render()* function. Finally, remove any remaining state references from the *MovieEdit* component.

It is important to note that even though you can load individual movies you cannot edit them yet. We will add that capability next,


Maurice
ReactJS
localhost:8080/#/movie/9917?_k=834uik
Apps
Read Later
Instapaper Text
Office 365
eVision Industry Sof...
Other bookmarks
ReactJS
Title
2001: A Space Odyssey
Directors
Stanley Kubrick
Critics Consensus
One of the most influential of all sci-fi films -- and one of the most controversial -- Stanley Kubrick's 2001 is a delicate, poetic meditation on the ingenuity -- and folly -- of mankind.

Assignment 3

The final task is to allow editing by updating the currentMovie object in the store.

Begin by opening the **actions.js** and adding *MOVIE-PROP-CHANGED* action.

Next open **reducers/currentMovie.js** and add a handler for the *MOVIE-PROP-CHANGED* action.

Next open **movieEdit.jsx** and add functions to dispatch the *MOVIE-PROP-CHANGED* actions from the *mapDispatchToProps()* function. Update the *MovieEdit* component to dispatch in the *onChange()* and *onChangeRatings()* functions instead of updating the local state.

After these changes, the *MovieEdit* component should be fully functional and the whole application should work as before.

Optional Assignment 4

Use the *redux-thunk* NPM package and move the three fetch API calls into actions of their own.