RxJS

Goal

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

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 RxJS 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 fetch data using RxJS.

Use NPM to install *rxjs*

```
npm install rxjs
```

Replace the use of the *fetch()* API with RxJS. Both the *MovieList* and *MovieEdit* components fetch data in their *componentDidMount()* lifecycle function. Replace these with *ajax.getJSON()*.

The *MovieEdit* component also does an update using n HTTP put. Replace the *fetch()* API here with *ajax.put()*. Make sure to send the data as *application/json* when saving the movie.

Make sure the application works and shows the list of movies.

Assignment 2

The second task is to create a Redux like store using RxJS.

Add a new file named *store.js* and use an RxJS Subject to create a store. Use the startWith() operator to specify the initial state and the scan() operator to reduce the actions dispatched. Export this store so you can use it in the *MovieList* and *MovieEdit* components.

Add and export a function *dispatch()* to dispatch new actions.

Update the *MovieList* and *MovieEdit* components to load the data from the store and dispatch the relevant actions as needed. You can find the required actions in ACTIONS/INDEX.JS.

Make sure the application works and shows the list of movies.

Optional Assignment 3

Create a Higher-Order Components named *connect* that takes a mapStateToProps function, Wrap the *MovieList* and *MovieEdit* components using this HOC and move all state management into the connect HOC.

```
class MovieList extends Component {
  componentDidMount() {
    ajax
      .getJSON("/api/movies")
      .subscribe(movies => dispatch(moviesLoaded(movies)));
  }

  render() {
    const rows = this.props.movies.map(movie => (
      <MovieRow key={movie.id} movie={movie} />
    ));

    return (
      <table className="table table-bordered table-striped">
        <thead>
          <tr>
            <th>Title</th>
            <th />
          </tr>
```

```
        </thead>
        <tbody>{rows}</tbody>
      </table>
    );
  }
}

export default connect(state => ({ movies: state.movies
}))(MovieList);
```

The MovieList and MovieEdit components should only work with props and dispatch after this.