travisallen6 / **redux-drills**
forked from DevMountain/redux-drills

No description, website, or topics provided.

Edit

Add topics

28 commits | 3 branches | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

This branch is even with DevMountain:master. Pull request | Compare

joeblank Merge pull request #11 from DevMountain/drill-5 ... Latest commit 0448c40 on Aug 29, 2017

drill-1	instructions and readme formatting	9 months ago
drill-2	instructions and readme formatting	9 months ago
drill-3	instructions and readme formatting	9 months ago
drill-4	instructions and readme formatting	9 months ago
drill-5	drill 5, updated readme	7 months ago
README.md	formatting	7 months ago

README.md

Redux Drills

You are in charge of building an app that can manage the guest list of DevMountain's next big hackathon. Complete drills 1-4 to build the guest list app.

Drill-1 (Setup)

You will start by installing dependencies and creating a single reducer.

- `npm install`
- Install `redux` and `react-redux`.
- Create a `ducks` folder in `src`.
- Inside of the `ducks` folder, create a file called `guestList.js`.
    - Create a reducer. The reducer is just a function that takes in state and an action. For now, have the reducer immediately return state. Export the reducer.

        ▶ **Solution: src/ducks/guestList.js**

Create your store.

- Create a `store.js` file in the `src` folder.
    - In `store.js` import `createStore` (from redux) and the reducer.
    - Export the invocation of `createStore` with the reducer as the only argument.

        ▶ **Solution: src/store.js**

In `index.js`:

- Import `Provider` (from react-redux) and the store.
- In the render method, wrap `<App />` with `Provider`.
- Pass the store, as a prop, to `Provider`.

▶ Solution: src/index.js

Drill-2 (Display guest list)

Building on the work you did in drill-1, you will now connect a component to the store so that you can display the guest list.

- `npm install`
- Install `redux` and `react-redux`.
- In `guestList.js`:
  - Set `initialState` to

```
const initialState = {
      guests: ['Tony Stark', 'Steve Rodgers', ' Nick Fury', 'Natasha Romanova', 'Clint Barton', 'Bruce
Banner', 'Wanda Maximoff']
  }
```

In `App.js`:

- Import `connect` from 'react-redux';
- Create your `mapStateToProps` function. Pull the list of guests off of state.
- `mapStateToProps` needs to be the first argument when `connect` is invoked.
- `map` over the guest list array that is now on props. The map should return some jsx with the guest's name and a remove button.

```
<div key={i} className="list-item">
    <li>{guest}</li>
    <button type="" className="">Remove</button>
</div>
```

▶ **Solution: App.js**

Drill-3 (Add/Delete guests)

Your guest list needs to be able to add and remove guests.

- `npm install`
- Install `redux` and `react-redux`.
- Add functionality of adding guest to list. In `guestList.js`:
  - Export a function called `addGuest`. This function is an action creator. It should return an object with a `type` and `payload`.
  - The `addGuest` function should have one parameter, which will be a guest name.
  - Set up a switch statement in the reducer function. When adding a guest, we should return a new piece of state that includes the new guest we are adding.
- In `App.js`:
  - Import the `addGuest` function from guestList.js.
  - As the second argument for the connect method, pass in an object with the key and value being `addGuest`.

```
export default connect(mapStateToProps, { addGuest })(App);
```

- Add the `constructor` and set up the initial state for the component. You will need to keep track of what is typed into the input box. (Hint: you will need to use the `onChange` event handler and `this.setState()`)
- When the `add` button is clicked, you need to call the `addGuest` function (on props) and pass in a guest name (the value of the input, which is on App's component state)

- You should be able to add guests to the list now. Following a similar process as you did add a guest, add the functionality of removing a guest when the `Remove` button is clicked.

▶ **Solution: App.js**

▶ **Solution: guestList.js**

Drill-4 (Update guest names)

For drill-4, you will have limited help. You have seen the process of building a store and a reducer, connecting a component to the store, dispatching actions, and displaying data from the store. Use this knowledge to complete drill-4. Try to think through what needs to be done and how to do it.

- `npm install`
- Install `redux` and `react-redux` .

i. Create `EditGuest.js` in `src` .

ii. The `EditGuest` component should be a view component (just a function, not a class). Go ahead and set up your component.

- Import `./EditGuest.css`

▶ **EditGuest.js setup**

iii. Paste the following code inside the `return` :

```
<div className="modal-bg">
  <div className="modal">
    <input className="modal-input"/>
    <button className="modal-btn">Update</button>
    <button className="modal-btn">Cancel</button>
  </div>
</div>
```

4. Add `edit: false` to the state object in the constructor. Import the `EditGuest` component to `App.js` . Use a ternary operator to test whether `this.state.edit` is true or false. If true, display an instance of the `EditGuest` component. This code should be in the jsx under the `form` tags.

▶ **Solution**

5. We now need to add functionality to the edit button. When the edit button is clicked, the modal need to show. Create a method called `showModal` on the App class. The method should set `this.state.edit` to `true` when the `edit` button is clicked on.

▶ **App.js**

6. The modal should close if the cancel button is clicked. Create a method called `hideModal` on the App class that sets `this.state.edit` to false. Pass this method as a prop to the EditGuest component and use it to add functionality to the cancel button.

▶ **App.js**

▶ **EditGuest.js**

7. Your modal should now show when you click edit, and hide when you click cancel. We now need to populate the input box on the modal with the name that we want to edit.

- We will keep track of the name and index of the guest we are editing in App's component state.

```
this.state = {
   text: '',
   edit: false,
   guestToEdit: '',
   index: 0
 }
```

- We need to pass the guest name and index to our showModal method, and we have access to both while we map over `this.props.list`. When then `Edit` button is clicked, is should invoke `this.showModal` and pass in `guest` and `i` as arguments.

  ▶ **Solution**

- Update the showModal method so that it updates `guestToEdit` and `index` on state.

  ▶ **Solution**

8. Pass the guest name (on App state) to the EditGuest component as a prop. Display the guest's name in the modal's input (as value).
9. When you click the edit button, the modal should appear with the correct guest name displayed in the input. We now need a way to keep track of the changes that we make to the name.
   - Create a method on the App component called `editName`. This method should update `this.state.guestToEdit` with the value typed in to the EditGuest component's input.
     - HINT: Don't forget to bind!

   ▶ **editName**

- Pass the `editName` method as a prop to the EditGuest component. In EditGuest.js, use the `onChange` event with the `editName` method as the event handler.

  ▶ **Solution**

▶ **input (EditGuest.js)**

10. We now need to make the update button work. If we are going to update information in our redux store then we need to head over to our `guestList.js` file.
    - At the top of the file, create a new constant:

```
const UPDATE_GUEST = 'UPDATE_GUEST';
```

    - Export a function called updateName with two parameters, `name` and `index`.
    - The updateName function should return an object with type and payload properties. The value of `type` should be `UPDATE_GUEST`. The value of `payload` should be an object that contains the values of the name and index parameters.

      ▶ **guestList.js**

    - Update the reducer to handle an action with the type of `UPDATE_GUEST`. Use the information in `action.payload` to return a new piece of state with the updated user name.

      ▶ **guestList.js**

    - In `App.js`, import the `updateName` function. Add it to the object that is passed as the second argument in the connect method.

      ▶ **Solution**

- Create a method on the App component called `updateGuestName`. This method will invoke `updateName` (action creator) and pass in `guestToEdit` and `index` from `App's` state. This method will also invoke the `hideModal` method.
  - HINT: Don't forget to bind!

    ▶ **updateGuestName**

- Pass the `updateGuestName` method as a prop to the EditGuest component. In `EditGuest.js`, use the method as an event handler for when the `update` button gets clicked.

▶ **App.js**
▶ **EditGuest.js**

Congrats! You should now have a fully working guest list that can add, remove, and edit guest names.

Final Solution:

▶ **App.js**
▶ **EditGuest.js**
▶ **guestList.js**

Drill-5 (HTTP requests)

NOTE: This drill is completely separate from the previous drills.

Goal: You will make HTTP requests to the Star Wars API (https://swapi.co) to get information on Star Wars characters, planets, and starships.

1. This react app is already set up with redux.
   - Run `npm install` to install dependecies.
2. Run `npm start` and take a look at the browser. If you are a Star Wars fan, you can tell that I have my movies mixed up. Apparently, Harry Potter is not in Star Wars...oops!
   - Help me fix my app by making http requests to the Star Wars API so I can show Star Wars people, planets, and starships.
3. In order to make HTTP requests, we will use the `axios` library. Since we will be making these HTTP requests in our action creators, we will need an additional library called `redux-promise-middleware`.
   - Run `npm install --save axios redux-promise-middleware`.
4. We now need to set up our app to use the middleware we just installed.
   - In `store.js`, import `promiseMiddlware` from `redux-promise-middlware` and `applyMiddlware` from `redux`.
   - The second argument in the `createStore` method will be the invocation of `applyMiddleware`. Pass in `promiseMiddlware` as the only argument to `applyMiddlware`.
   - NOTE: Be sure to invoke `promiseMiddlware`. See below.

   ▶ `store.js`
5. In `star_wars.js`
   - Import 'axios'
   - Export a function called `getPeople`. We will make the HTTP request in the `getPeople` function. Using `axios`, make a `GET` request to
     - `https://swapi.co/api/people`
     - Resolve the promise with `.then` and return `response.data.results`
   - `getPeople` should return an object with `type` and `payload` properties.
   - Create a constant for your action `type`.

        - `redux-promise-middleware` will concat '_FULFILLED' to the end of your action type.
        - Remember that the case you are testing for (in the switch statement) is [ACTION TYPE] + '_FULFILLED'.
    - The value of the action payload should be the result of the HTTP request.
    - Complete the switch statment in your reducer function so that it updates state with the response from the HTTP request (sent in the action).

▶ `star_wars.js`

6. In `App.js` :

    - Import your action creator (getPeople) from `star_wars.js` .
    - The second argument in the `connect` method is going to be an object. This is where we need to put the action creator that we just imported. If this process if unfamiliar, you should revisit the previous drills for more instructions/explainations of this process.
    - The `getPeople` action creator should be invoked when the `Get correct people` button is clicked.

▶ `App.js`

7. Following the same pattern that was just used to get the correct Star Wars people, make the other two buttons (planets and starships) functional.

    - Use a GET request for the following:
        - planets: 'https://swapi.co/api/planets/'
        - starships: 'https://swapi.co/api/starships/'