**Lesson 8 Review**

SEND FEEDBACK

**Review**

**Redux**

**Store**

Redux applications have a single store. We have to pass the Root Reducer to our createStore()function in order for the store to know what pieces of state it should have. The point of creating a store is to allow components to be able to access it without having to pass the data down through multiple components.

The Provider component (which comes from the react-redux package) makes it possible for all components to access the store via the connect() function.

**Actions and Action Creators**

Remember how normal Action Creators return actions - simple Javascript objects that then go to all of our reducers? Making an API request is an asynchronous action, so we cannot just send a plain Javascript object to our reducers. Redux middleware can gain access to an action when it's on its way to the reducers.

If the Redux Thunk middleware is enabled (which is done via the applyMiddleware() function), then any time your action creator returns a function instead of a Javascript object, it will go to the react-thunk middleware.

Dan Abramov [**describes**](https://stackoverflow.com/questions/35411423/how-to-dispatch-a-redux-action-with-a-timeout/35415559#35415559) what happens next:

“The middleware will call that function with dispatch method itself as the first argument...The action will only reach the reducers once the API request is completed. It will also “swallow” such actions so don't worry about your reducers receiving weird function arguments. Your reducers will only receive plain object actions—either emitted directly, or emitted by the functions as we just described.”

Here's what a thunk action creator looks like:

**function** **handleInitialData** () {

**return** **function** (dispatch) {}

}

Which is equivalent to this in ES6:

**function** **handleInitialData** () {

**return** (dispatch) => {}

}

**Reducers**

A [**Reducer**](https://redux.js.org/basics/reducers) describes *how* an application's state changes. You’ll often see the [**Object Spread Operator**](https://redux.js.org/recipes/using-object-spread-operator) (...) used inside of a reducer because a reducer **must return a *new* object** instead of mutating the old state. If you need a refresher on the spread operator, check out [**this ES6 lesson**](https://classroom.udacity.com/nanodegrees/nd019/parts/290ec447-6555-41bf-ac39-457220a09aae/modules/9c5b7af0-0943-4d6e-b672-520440885aba/lessons/42383e89-ac6a-491a-b7d0-198851287bbe/concepts/398d36e6-3393-4c50-b870-44a4dffb0ac4).

If you want to know *why* Redux requires immutability, check out the [**Immutable Data Section of the docs:**](https://redux.js.org/faq/immutable-data#why-is-immutability-required).

Reducers have the following signature:

(previousState, action) => newState