Redux.1

▼ What is the purpose of the state management library?

The idea of it is that the global state is represented as a single stateful object, which is altered in different parts of the app with the help of reducers – special Redux functions.

▼ With Redux, we're taking the state management out of React entirely and moving it to what?

a separate store

▼ Redux state management (steps - example)

1. User types in input box
2. Call action creator to get an action
3. Dispatch action to Redux
4. Redux inserts the action into the root reducer
5. The root reducer delegates that action to the correct reducer
6. The reducer returns a new state given the old state and the action object
7. That new state becomes the store's state
8. React is then called by Redux and told to update

▼ What is context in React?

With context, you use the provider and consumer as a sort of portal to skip passing parameters through every component.

▼ Why Redux?

Redux code is extremely testable. This is probably the most compelling reason to use it. Having your state mutation be broken up in such a way to make it easy to test is fantastic.

▼ React state management is pretty simple and what?

call setState and let React re-render

▼ What do you need to install for Redux?

```
npm install redux@4.0.5 react-redux@7.2.2
```

▼ You need to create store.js and put what in it? (example)

```
import { createStore } from "redux";
import reducer from "./reducers";

const store = createStore(
  reducer,
  typeof window === "object" &&
    typeof window.__REDUX_DEVTOOLS_EXTENSION__ !== "undefined"
    ? window.__REDUX_DEVTOOLS_EXTENSION__()
    : (f) => f
);

export default store;
```

▼ The base of a store is what?

a reducer

▼ What is a store?

A store is just basically a big object with prescribed ways of changing it.

▼ What is a reducer?

A reducer takes an old state, an action, and combines those things to make a state.

▼ Create a root reducer (example)

Make a new folder in src called reducers. Create a file called index.js in reducers and put:

```
import { combineReducers } from "redux";
import location from "./location";
import theme from "./theme";

export default combineReducers({ // root reducer
  location, // reducer
  theme,  // reducer
});
```

▼ What is combineReducers?

a convenience function from Redux so you don't have to write your own root reducer


```
export default combineReducers({ // root reducer
  location, // reducer
  theme,  // reducer
});
```


▼ Create a reducer (example)

Make a file called location.js and put in it:

A reducer takes an old state, an action, and combines those things to make a state.

In this case, if the state is San Francisco, CA and some calls it with the action {type: 'CHANGE_LOCATION': payload: 'Salt Lake City, UT' } then the new state location would be Salt Lake City, UT.

```
export default function location(state = "Seattle, WA", action) {
  switch (action.type) {
    case "CHANGE_LOCATION":
      return action.payload;
    default:
      return state;
  }
}
```

▼ How does Redux initialize a store?

by calling each reducer once to get a default state

▼ Are reducers synchronous?

yes

▼ Reducers must be pure with what?

no side-effects

▼ What are action creators?

These are the functions that the UI gives to the store to effect change: actions.

▼ Action creator (example)

Create a new folder called actionCreators and put in changeTheme.js

```
export default function changeTheme(theme) {
  return { type: "CHANGE_THEME", payload: theme };
}
```

▼ What can you use redux-thunk for?

async actions

▼ What is Provider?

Just like context makes your store available anywhere in your app, so does Provider.

▼ Adding a Provider (example)

```
// App.js

// delete ThemeContext, useState import

// import
import { Provider } from "react-redux";
import store from "./store";
```

```
// delete useState call
// delete ThemeContext

// wrap app with
<Provider store={store}>[…]</Provider>;
```

```
// SearchParams.js

// replace ThemeContext import
// delete useContext import
import { useSelector } from "react-redux";

// replace context and some usestate references
const animal = useSelector((state) => state.animal);
const location = useSelector((state) => state.location);
const breed = useSelector((state) => state.breed);
const theme = useSelector((state) => state.theme);
```

▼ What is a selector function?

it will pluck the bit of state you need from Redux

```
const theme = useSelector((state) => state.theme);
```

▼ What is the useDispatch hook?

it gives you back a dispatching function so you can dispatch actions

▼ Dispatching actions (example)

```
// SearchParams.js

import { useSelector, useDispatch } from "react-redux";
import changeLocation from "./actionCreators/changeLocation";
import changeTheme from "./actionCreators/changeTheme";
import changeAnimal from "./actionCreators/changeAnimal";
import changeBreed from "./actionCreators/changeBreed";

// up with other hooks
const dispatch = useDispatch();

// change inputs

<input
  id="location"
```

```
  value={location}
  placeholder="Location"
  onChange={(e) => dispatch(changeLocation(e.target.value))}
/>

<select
  id="animal"
  value={animal}
  onChange={(e) => dispatch(changeAnimal(e.target.value))}
  onBlur={(e) => dispatch(changeAnimal(e.target.value))}
></select>

<select
  disabled={!breeds.length}
  id="breed"
  value={breed}
  onChange={(e) => dispatch(changeBreed(e.target.value))}
  onBlur={(e) => dispatch(changeBreed(e.target.value))}
></select>

<select
  value={theme}
  onChange={(e) => dispatch(changeTheme(e.target.value))}
  onBlur={(e) => dispatch(changeTheme(e.target.value))}
></select>
```

▼ What is mapDispatchToState?

it lets us write functions to dispatch actions and thunks to Redux

▼ mapStateToProps (example)

```
// replace ThemeContext import
import { connect } from "react-redux";

// remove all the ThemeContext stuff and the interior function
// replace `context.theme` with just `this.props.theme` for the backgroundColor

// bottom
const mapStateToProps = ({ theme }) => ({ theme });

const WrappedDetails = connect(mapStateToProps)(Details);

// replace DetailsWithRouter
const ReduxWrappedDetails = connect(mapStateToProps)(Details);

const DetailsWithRouter = withRouter(ReduxWrappedDetails);
```

▼ What is a thunk?

A thunk can be used to delay the dispatch of an action, or to dispatch only if a certain condition is met.