Redux store

Redux Store

- the redux store holds the entire state tree of the application
- The only way to change the state is by dispatching an action
- to create the store you use the method **createStore**
- you need to pass the root **reducer** to create store
- Redux application has one store and the entire state of the app is in that store
- after you create the store, the store will call the reducer with **state=undefined, action.type = @@redux/INIT**
- to create core redux stuff we use the **redux** package
- let’s create our simple store

Store Methods

- **getState() -** returns the state tree of the application
- **dispatch(action)** - triggers a state change, the store will ask the reducer how to change the state

Splitting Reducers

- when our app state tree becomes more complex, a single reducer to handle the entire state can be a bit messy
- We would like to split the state tree to sections and let different reducers be in charge of different sections
- We can split the root reducer to multiple small reducers
- **combineReducers(object)** - takes and object with multiple reducers functions and create a single reducer you can pass to **createStore**
- let’s try and split the current state we have to **{user: {...}, todo: {...}}**

Store middlewares

- middlewares are wrappers around the **store.dispatch**
- there can be more that one middleware
- We can use middlewares to enhance our store so we can work with async actions
- To work with async actions we will use **redux-thunk** middleware
- to install **redux-thunk: > npm install redux-thunk --save**
- we can apply a middleware on the store by calling the **applyMiddleware(...middlewares)** and supplying the result to the **createStore** second argument
- after installing **redux-thunk** and action can return a function that will be called with the **store.dispatch** argument
- let’s try and install **redux-thunk** and dispatch and async action

Summary - Combining with React

- How does it all combine with React?
- to connect React with Redux we use the **react-redux** npm package
- Let’s practice what we learned and create a react component which will do the following
  - after component is mounted grab from the todo server the list of tasks
  - display a list of all the titles of tasks you grabbed
  - The store will have a single reducer with the list of tasks in the state
  - the react component will be connected to that list of tasks