Redux

OBJECTIVES

- Understand why one would use Redux
- Explain Redux basics such as actions, reducers, and store
- Differentiate between plain redux and using the react-redux library
- Name the steps to refactor vanilla react state to redux.

3 principles:

1. State is tracked in a single js object
2. State is READONLY. Actions manipulate state, actions have a state
3. A function (the Reducer) calculates the next state tree based on the previous state tree and the action being dispatched.

WHY REDUX?

This complexity is difficult to handle as we're mixing two concepts that are very hard for the human mind to reason about: mutation and asynchronicity

Also skim this on why react / redux (from casidoo's newsletter) it is about separating state from dom and the problems with lifting state to top. Passing everything from top is a little crufty.

Things to notice with Redux

`const store = createStore(reducer)`

`Vanilla react = store.subscribe() & store.dispatch()`

What is the "shape" of our state?

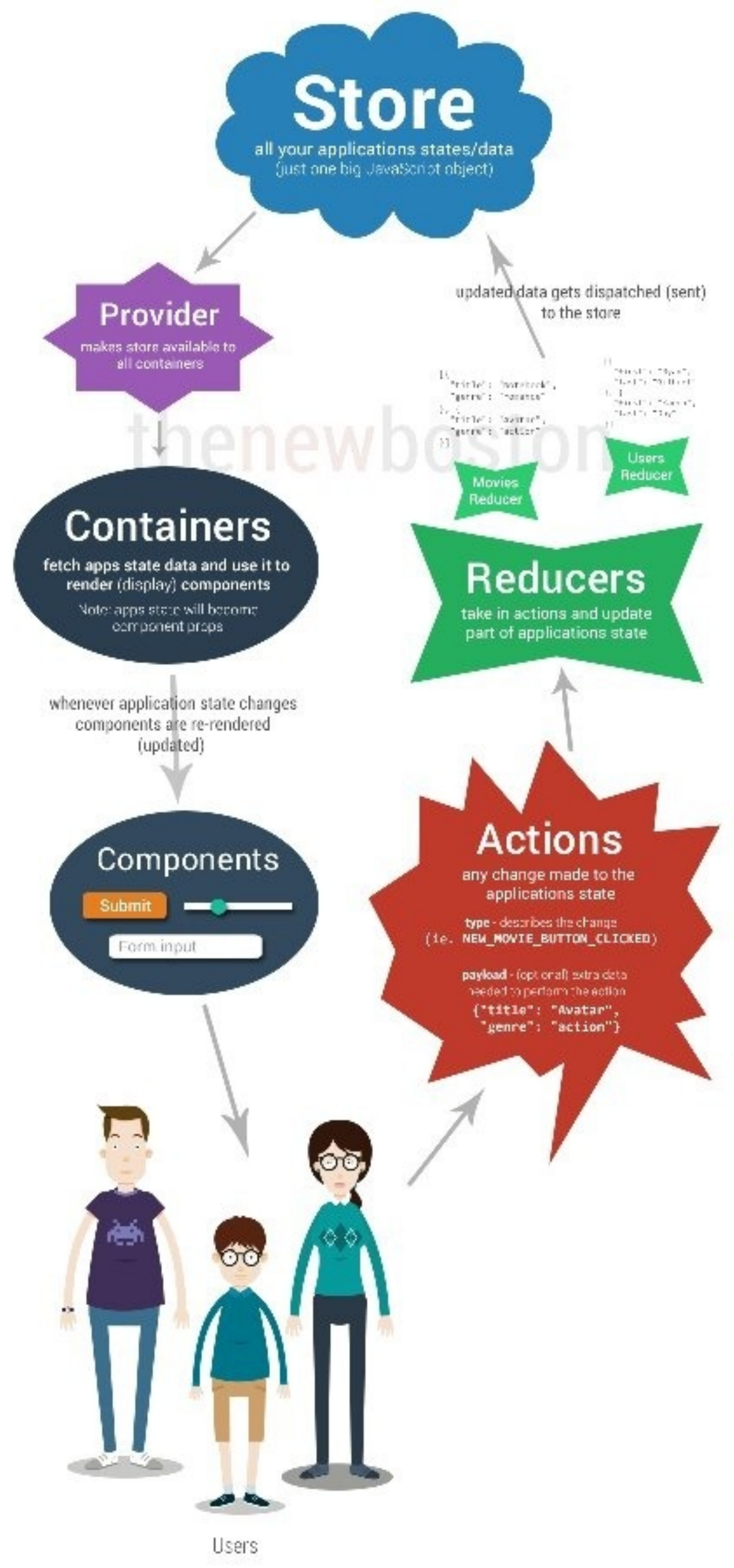
From reducers/todos.js

```
const todos = (state = [], action)
```

It's called README for a reason

Go over `counter example` together

Redux Explained


Store
all your applications states/data
(just one big JavaScript object)
Provider
makes store available to all containers
thenewboston
Containers
fetch apps state data and use it to render (display) components
Note: apps state will become component props
whenever application state changes components are re-rendered (updated)
Components
Submit
Form input
Users
Actions
any change made to the applications state
type - describes the change
(i.e. NEW_MOVIE_BUTTON_CLICKED)
payload - (optional) extra data needed to perform the action
{"title": "Avatar",
"genre": "action"}
Reducers
take in actions and update part of applications state
Movies Reducer
Users Reducer
updated data gets dispatched (sent) to the store

buckyroberts/react-redux-boilerplate. (2017). GitHub. Retrieved 18 July 2017, from https://github.com/buckyroberts/react-redux-boilerplate

QUICK SKIT

User - Actually using the app on the screen adding todos as we act it out.

Container - Managing the dumb component, dispatching actions, subscribing to changes in state.

Component - Between me and the class. Must be really dumb. Asking dumb questions, while container component is being really condescending and giving orders.

QUICK SKIT (CONTINUED)

Action - Dispatched to the store in the middle of the room. “Ok he wants to add a todo this time, something about taking out the trash, says it is urgent” or whatever.

Store - Receives actions dispatched to update state, uses the reducer to make new copy of state with update, and notifies subscribers when the state has been updated.

Reducer - Takes old state and action, producing a new state. May want to talk about doing this immutably,

Usage with React & Todos Example

In Redux docs, focus on "Usage With React" section! Trust me.

Usage With React
Todos Example

Using react-redux

Provider (react-redux) takes store

Connect (react-redux) is alternative to avoid writing store.subscribe() and store.dispatch() by hand and unnecessary renders

More on connect (react-redux)

To use connect, you need to define a special function called mapStateToProps() instead of using store.subscribe()

mapDispatchToProps() receives the store.dispatch() method and returns callback props that you want to inject into the presentational component

"DUMB" COMPONENTS VS CONTAINERS

Components tend to be “dumb” and not use connect. Containers tend to use connect and dispatch, but not always black and white.

Objectives

» Understand why one would use Redux

» Explain Redux basics such as actions, reducers, and store

» Differentiate between plain redux and using the react-redux library