My Notes

martes, 1 de noviembre de 2022 13:42

→ Actions are the only way to change states, through a reducer, which is a function that returns the new state. An action is launched by dispatch()

↳ any data needs to be dispatched as an action
↓
actions are info that send data to the store

→ Actions are sent to the store.

→ Reducers are functions that specify how state changes after an action

↳ accept current state + action

→ create store(reducer)

→ the <Provider/> component stores the state tree. It must enclose the app component.

```
<Comp.js>

func Comp () => {
     return(
          <p>props.message</p>
          <button onClick=props.ButtonChange()></button>
     )
}
```

→ func to use the state from props

```
const mapStatetoProps = (state) => {
     message: state.message
}
const mapDispatchToProps = (dispatch) => {
     ButtonChange(): () => dispatch({type: "MESSAGE_CHANGE"})
}
export default connect(mapStatetoProps, mapDispatchToProps)(Comp)
</Comp.js>
```

→ global state

→ now in Comp we have props.message from the common state

function to use dispatch from props

action!

action type

```
<./store/Reducer.js>
const initialState = {
     Message: "hi"
}
const reducer = (state=initialState, action) => {
     const newState = {...state}
     if (action.type === 'MESSAGE_CHANGE") {
          NewState.message = "how are you"
          Return newState
     }
};
export default reducer
</./store/Reducer.js>
```

```
<Index.js>
Store = createStore(reducer)
<Provider>
     <App>
          <Comp/>
     </App>
</Provider>
</Index.js>
```

→ dispatch ({ type: "tipo" }) + body → calls the reducer → reducer: switch (tipo)

$\underbrace{}_{./actions/}$

cambia

use Selector extracts from store { Comps usan state vars ← { state var : body }

```
<action.js>
export const myAction(dispatch){
     .fetch()
     disptach({
          type: "TYPE1",
          payload: fetchedjson
          })
}
</action.js>
```

→ connect()() connects your components to the store

→ dispatch → sends { type: "", data: from fetches } to the reducer that returns the new state

```
<Comp.js>
import myAction
class Comp {
     componentWillMount(){
          this.props.myAction()
     }
     render(
          <div>
               {this.props.posts}
          </div>
     )
}
```

→ want the action called at load

→ calls the fetch()

```
export default connect(mapStatetoProps)(Comp)
```

get the state (changed by reducer) from the store with mapStateToProps()

```
mapStatetoProps = state => ({
     posts: state.posts.items
});
</Comp.js>
```

```
<rootReducer.js>
import postReducer from …
export default combineReducers({
     posts: postReducer
})
</rootReducer.js>
```

case :
return (...state, items)

Returned new state

→ good Practice : add PropTypes

→ actions → fetches (API)
→ dispatch ({type, payload}) → calls reducer (returns new state to store)

actions ↑ import ↑ components

→ Further actions add more to the app state

→ lifecycle methods : componentWillMount()
componentWillReceiveProps() → whenever a new prop changes