Intro to Redux

Redux is a opinionated tool for managing your applications state

(The hivemind has decided that it’s the bee’s knees)

Key questions:

1) How do I get Redux into my front-end application?
2) What is the Redux store?
3) What is an Action?
4) What is a Reducer?
5) What does createStore() do?
6) What does dispatch() do?
7) What does subscribe() do?

Let’s start with practical stuff first

The theoretical stuff will come after

Load in Redux from a CDN with a script tag

- This will expose one global variable called “Redux”
- You can also use import in Redux with browserify/weback (not shown here)

```
<html>
  <head>
  </head>
  <body>
      ...
<script
      src=”https://cdnjs.cloudflare.com/ajax/lib
      redux/4.0.0/redux.min.js”></script>

  </body>
</html>
```

Redux has 4 crucial functions

- Redux.createStore()
- Redux.getState()
- Redux.dispatch()
- Redux.subscribe()

Redux.createStore()

- createStore kicks off your redux application
- It takes one parameter, your REDUCER function
- *What is a REDUCER? Stay tuned to find out*

```
<script>

        let reducer = (state, action) => {
            ...
        }

        let store = Redux.createStore(reducer);

        store.subscribe(() => {
            document.body.innerHTML = Redux.getState();

        })

        document.addEventListener('click', ()=>{
            store.dispatch({
                type: "DOCUMENT_CLICKED"
            })
        });

</script>
```

Redux.getState()

- Returns the current STATE object
- Can be called anywhere in your code

```
<script>

        let reducer = (state, action) => {
            ...
        }

        let store = Redux.createStore(reducer);

        store.subscribe(() => {
            document.body.innerHTML = Redux.getState();
        })

        document.addEventListener('click', ()=>{
            store.dispatch({
                type: "DOCUMENT_CLICKED"
            })
        });

</script>
```

Redux.dispatch()

- Takes an ACTION as a parameter
- It tells redux to calculate the new STATE using your REDUCER function
- *What's an ACTION? Stay tuned to find out*

```
<script>

        let reducer = (state, action) => {
            ...
        }

        let store = Redux.createStore(reducer);

        store.subscribe(() => {
            document.body.innerHTML = Redux.getState();

        })

        document.addEventListener('click', ()=>{
            store.dispatch({
                type: "DOCUMENT_CLICKED"
            })
        });

</script>
```

Redux.subscribe()

- Subscribe lets you specify callbacks to call whenever the STATE changes
- These callbacks are where you write your DOM manipulation code! (i.e. your render functions)

```
<script>

        let reducer = (state, action) => {
            ...
        }

        let store = Redux.createStore(reducer);

        store.subscribe(() => {
            document.body.innerHTML = Redux.getState();
        })

        document.addEventListener('click', ()=>{
            store.dispatch({
                type: "DOCUMENT_CLICKED"
            })
        });

</script>
```

So what does Redux do?

STATE

Redux manages a single object called STATE.

Your job is do define what the STATE looks like.

The STATE can be simple, (e.g. just a single number), or very complex (e.g. a large object with nested booleans, arrays, strings, etc.)

```
{
  applicationName: “Tic-Tac-Toe”,
  currentPlayer: “Player 1”,
  board: [
    [“X”, “”, “O”],
    [“”, “X”, “O”],
    [“”, “X”, “O”]
  ]
}
```

ACTIONS

Redux lets you dispatch ACTIONS

ACTIONS trigger changes in the application STATE

ACTIONS are objects with a “type” property (required), and the minimum information needed to process the state change

```
{
  type: “MARK_SQUARE”,
  row: 0,
  column: 1
}
```

REDUCER

The REDUCER is a function that defines how STATE should change for a given ACTION

The REDUCER takes the currentState and an ACTION as parameter

The REDUCER must return the resulting STATE after the given ACTION

```
const reducer = (state, action) {
  var stateCopy = Object.assign({}, state)
  if (action.type == “MARK_SQUARE”) {
    let {row, column} = action;
    stateCopy.board[row][column] = “X”
    return stateCopy;
  } else {
    return state;
  }
}
```

So, in a nutshell...

- With Redux, you define a single reducer
- The reducer should define what happens to the state for any given action
- Redux will create a single State object that holds ALL of your application's data
- When events happen (the user does something, an AJAX call comes back, a timer goes off, etc.), you can dispatch actions
- Redux will capture those actions and call your reducer, which tells redux what the new state should be
- When the state changes, Redux will call all of your callbacks you subscribed

What does this acheive?

Total separation of your data/logic and your UI/DOM manipulation!