# REDUX

Because state management can be hard

# What is state

- Eg:

```
const state = {
  posts: [],
  signUpModal: {
    open: false
  }
}
```

```
<div className={this.state.signUpModal.open ? 'hidden' : ''}>
  Sign Up Modal
</div>
```

- state references the condition of something at a particular point in time, such as whether a modal is open or not.
- In a React component the state holds data which can be rendered to the user.
- The state in React could also change in response to actions and events: in fact you can update the local component's state with this.setState().
- So, in general a typical JavaScript application is full of state. For example, state is:
  - what the user sees (data)
  - the data we fetch from an API
  - the URL
  - the items selected inside a page
  - eventual errors to show to the user

# State can be complex

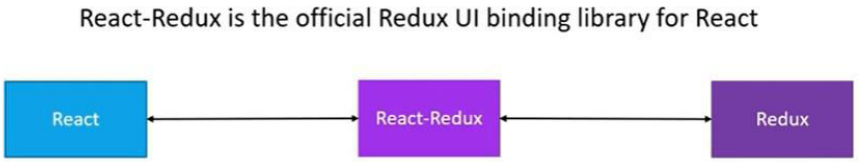
- Even an innocent single page app could grow out of control without clear boundaries between every layer of the application. This holds particularly true in React.
  - You can get by with keeping the state within a parent React component (or in context) as long as the application remains small.
  - Then things will become tricky especially when you add more behaviours to the app. At some point you may want to reach for a consistent way to keep track of state changes.

# Create a Redux application

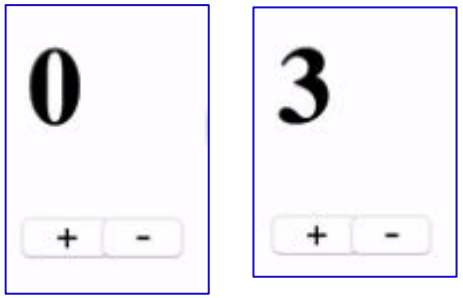
npx create-react-app redux-app

cd redux-app

npm install redux react-redux




```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

//STORE -> GLOBALISED STATE

//ACTION -> INCREMENT

//REDUCER

//DISPATCH

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```



# Create a Redux application

- Step-1 : Create the store:

```
import {createStore} from 'redux’;
const myStore = createStore(reducer-name)
```

- Step-2 : Create action

```
const increment = () => {
  return {
    type :'INCREMENT'  //name of the action
  }
}
const decrement = () => {
  return {
    type :'DECREMENT'  //name of the action
  }
}
```

- Step-3 : Create reducer

```
function reducer(state=initial-state, action){}
```

```
const counter = (state=0, action) => {
  switch(action.type){
    case "INCREMENT":
      return state + 1;
      case "DECREMENT":
      return state - 1;
  }
}
let store = createStore(counter)
```

# Create a Redux application

- Step-4 : display store on console
- `store.subscribe(() => console.log(store.getState()))`

- Step-5 : dispatch the action
- `store.dispatch(increment()); //dispatches the increment action`

```
//DISPATCH
store.dispatch(increment()); //dispatches the increment action
store.dispatch(decrement()); //dispatches the decrement action
store.dispatch(decrement()); //dispatches the decrement action again
```

- Step-6 : Execute the app. Start server
- npm start

# Need for Redux

- Redux offers a solution to storing all your application state in one place called "**Store**"
- Components then "**dispatch**" state changes to store, not directly to other components
- Components that need to be aware of state changes can "**subscribe**" to the store
- The center of every Redux application is the store. A "store" is a container that holds your application's global state.
  - A store is a JavaScript object with a few special functions and abilities that make it different than a plain global object:
  - You must never directly modify or change the state that is kept inside the Redux store
  - Instead, the only way to cause an update to the state is to create a plain action object that describes "something that happened in the application", and then dispatch the action to the store to tell it what happened.
  - When an action is dispatched, the store runs the root reducer function, and lets it calculate the new state based on the old state and the action
  - Finally, the store notifies subscribers that the state has been updated so the UI can be updated with the new data.

# Actions, reducers and dispatchers

- An **action** is a plain JavaScript object that has a type field. You can think of an action as an event that describes something that happened in the application.
  - The type field should be a string that gives this action a descriptive name. Eg "todoAdded“ or “depositFunds” or “incrementCounter”

- A **reducer** is a function that receives the current state and an action object, decides how to update the state if necessary, and returns the new state: (state, action) => newState.
  - You can think of a reducer as an event listener which handles events based on the received action (event) type.

- The Redux store has a method called **dispatch**. The only way to update the state is to call store.dispatch() and pass in an action object.
  - The store will run its reducer function and save the new state value inside, and we can call getState() to retrieve the updated value:

# My original index.js

```
//original React imports
import {createStore} from 'redux';

//STORE -> GLOBALISED STATE

//ACTION -> INCREMENT
const increment = () => {
  return {
    type :'INCREMENT'  //name of the action
  }
}
const decrement = () => {
  return {
    type :'DECREMENT'  //name of the action
  }
}

//REDUCER
const counter = (state=0, action) => {
  switch(action.type){
    case "INCREMENT":
      return state + 1;
      case "DECREMENT":
      return state - 1;
  }
}
```

```
let store = createStore(counter);

//Display it in console
store.subscribe(() =>
     console.log(store.getState()));

//DISPATCH
store.dispatch(increment());
store.dispatch(decrement());

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

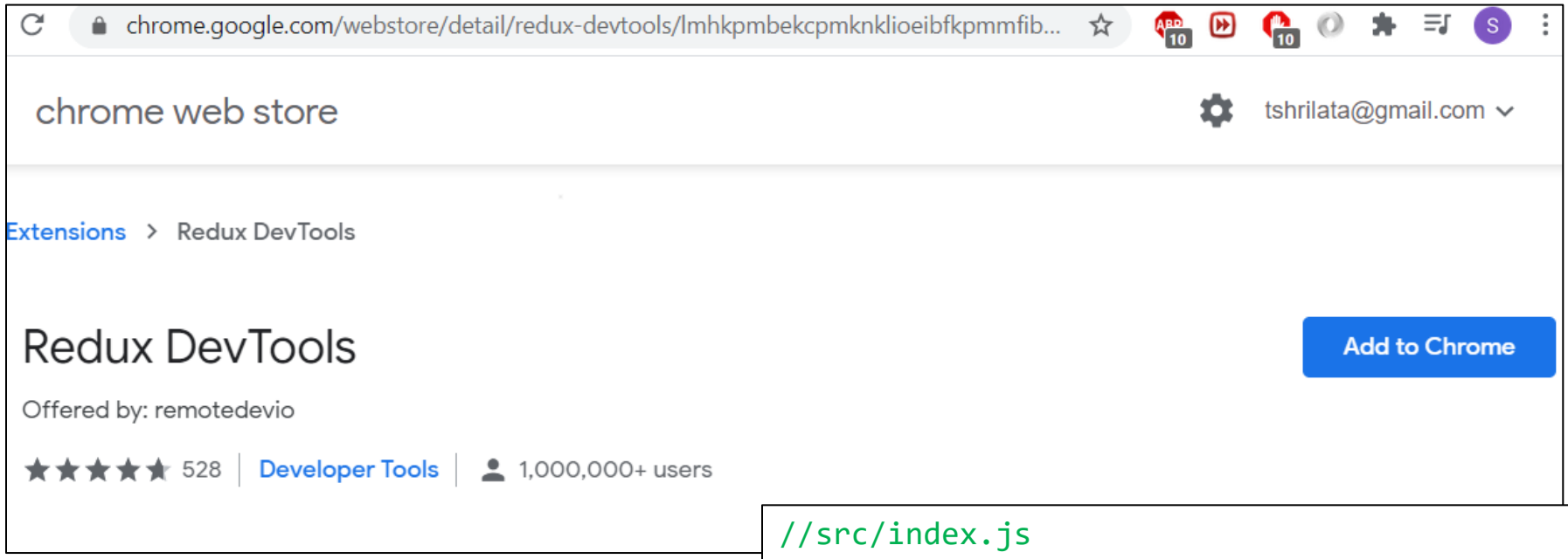
# Create a Redux app

```
//reducers/counter.js
const counterReducer = (state=0, action) => {
    switch(action.type){
      case "INCREMENT":
        return state + 1;
      case "DECREMENT":
        return state - 1;
      default: return null;
    }
  }
  export default counterReducer;
```

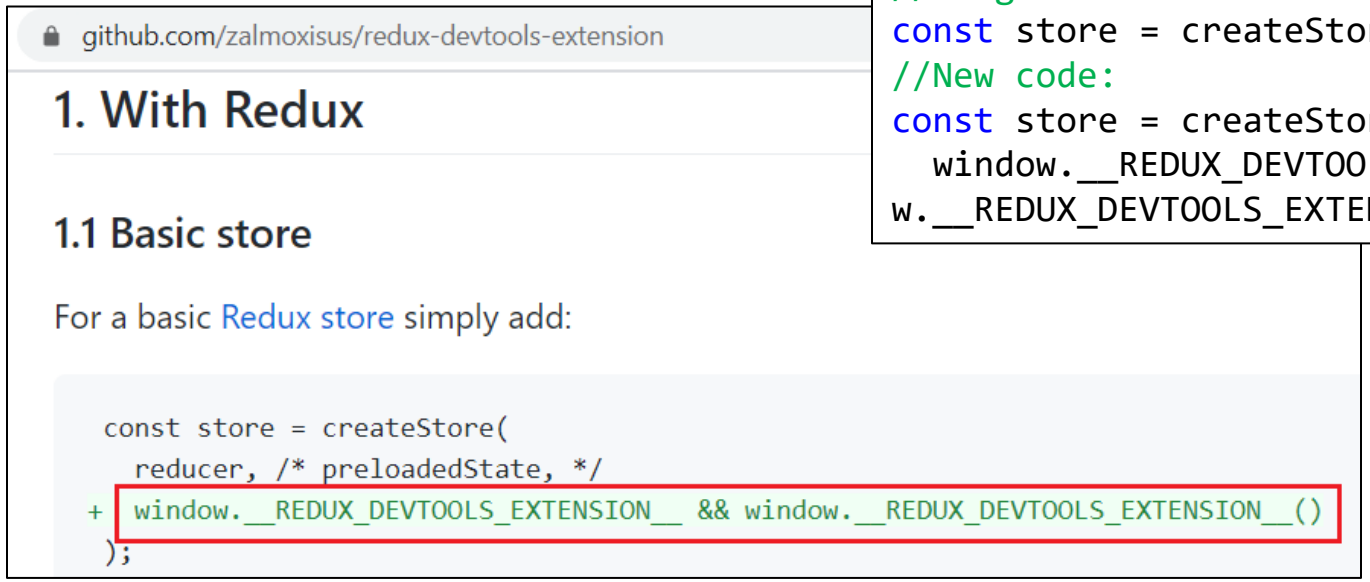
```
//src/index.js
import {createStore} from 'redux';
import allReducers from "./reducers";

const store = createStore(allReducers);
```

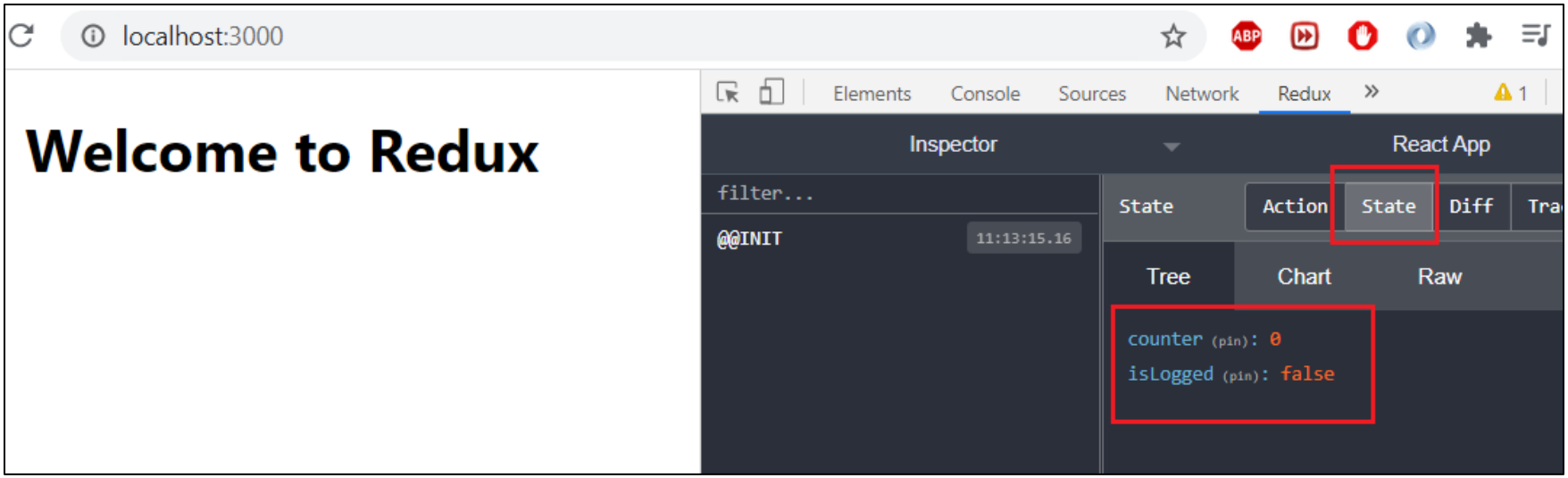
```
//reducers/isLogged.js
const loggedReducer = (state=false, action) => {
    switch(action.type){
      case "SIGNIN":
        return !state;
      default:
         return state;
    }
  }
  export default loggedReducer;
```

```
//reducers/index.js
import counterReducer from "./counter";
import loggedReducer from "./isLogged";
import {combineReducers} from 'redux';

const allReducers = combineReducers({
    counter : counterReducer,
    isLogged:loggedReducer
})
export default allReducers;
```




```
//src/index.js
//Original code:
const store = createStore(allReducers);
//New code:
const store = createStore(allReducers,
  window.__REDUX_DEVTOOLS_EXTENSION__ && windo
w.__REDUX_DEVTOOLS_EXTENSION__());
```






```
//src/index.js
import {createStore} from 'redux';
import allReducers from "./reducers";
import {Provider} from 'react-redux';

const myStore = createStore(allReducers,
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__());

ReactDOM.render(
  <React.StrictMode>
    <Provider store={myStore}>
        <App />
    </Provider>
  </React.StrictMode>,
```

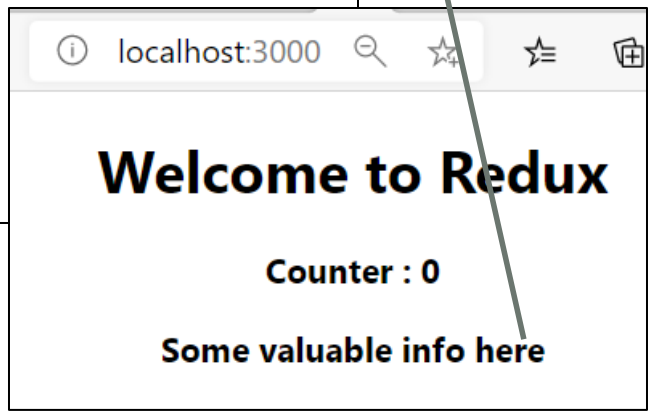
```
const counterReducer =
(state=0, action) => {…}
```

# App.js : displaying state

```
import './App.css';
import {useSelector} from 'react-redux';

 function App() {
   const counter = useSelector(state => state.counter);
   const isLogged = useSelector(state => state.isLogged);


  return (
    <div className="App">
      <h1>Welcome to Redux</h1>
      <h3>Counter : {counter}</h3>
       {isLogged ? <h3> Some valuable info here</h3> : ''}
    </div>
  );
}

export default App;
```

Allows you to extract data from the Redux store state, using a selector function.

I set the isLogged state to true



# Modifying state

```
//actions/index.js
export const increment = () => {
      return {
        type :'INCREMENT'
      }
    }
 export const decrement = () => {
      return {
        type :'DECREMENT'
      }
    }
```

```
import './App.css';
import {useSelector, useDispatch} from 'react-redux';
import {increment} from './actions';
import {decrement} from './actions';

 function App() {
   const counter = useSelector(state => state.counter);
   const isLogged = useSelector(state => state.isLogged);
   const dispatch = useDispatch();

  return (
    <div className="App">
      <h1>Welcome to Redux</h1>
      <h3>Counter : {counter}</h3>
      <button onClick={() => dispatch(increment())}>+</button>
      <button onClick={() => dispatch(decrement())}>-</button>
       {isLogged ? <h3> Some valuable info here</h3> : ''}
    </div>
  );
}
export default App;
```