



Redux

What is
Redux?

As the documentation states,
Redux is a predictable state
container for JavaScript apps.

To put it in simpler terms, Redux maintains the state of an entire app in a single immutable state tree (object).

Or even simpler,

Redux is a state management library.

Redux was created by Dan Abramov in 2015, while working for Facebook in London as part of the React Core team, and it was inspired by Facebook's Flux architecture.

It got popular very quickly because of its “simplicity”, small size (about 2.6kb), and great documentation.



- Dan Abramov



github.com/gaearon



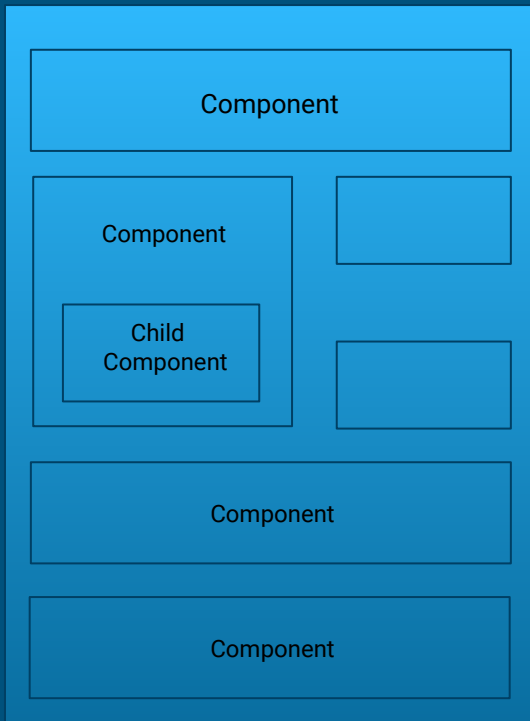
[@dan_abramov](https://twitter.com/dan_abramov)

www.egghead.io/q/resources-by-dan-abramov

Abstract Redux

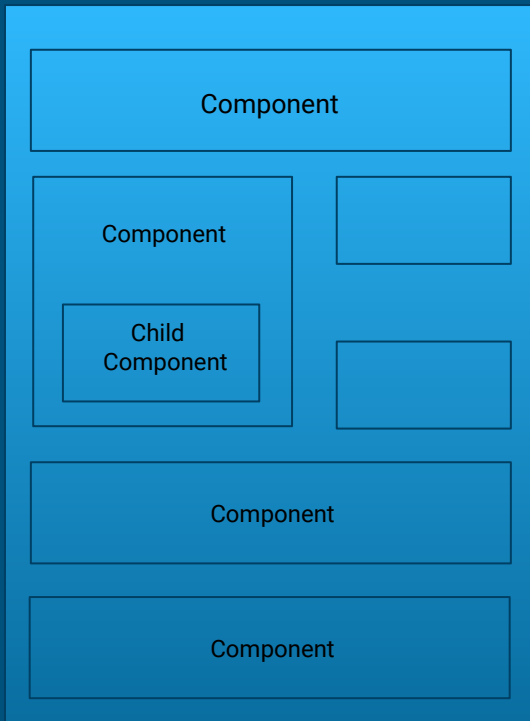
Abstract Redux

APP



Abstract Redux

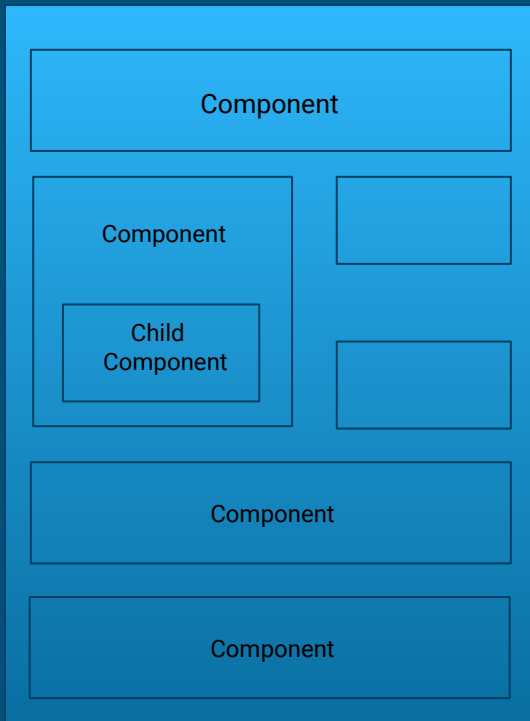
APP



- When using react each components is responsible for their state.

Abstract Redux

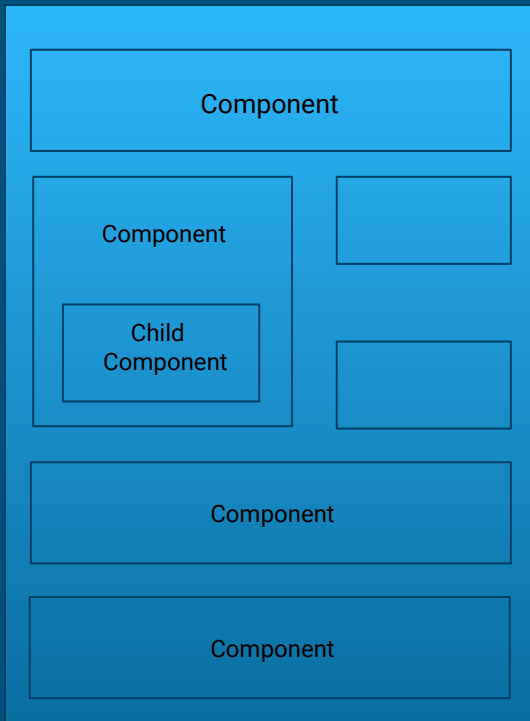
APP



- When using react each components is responsible for their state.
- You can pass state by passing props to the child components

Abstract Redux

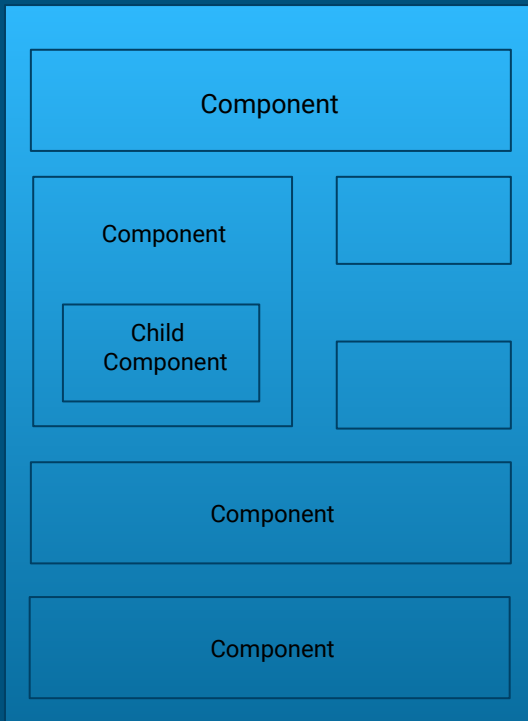
APP



- When using react each components is responsible for their state.
- You can pass state by passing props to the child components
- React encourages unidirectional data flow. That mean that data flows down from parent to child component.

Abstract Redux

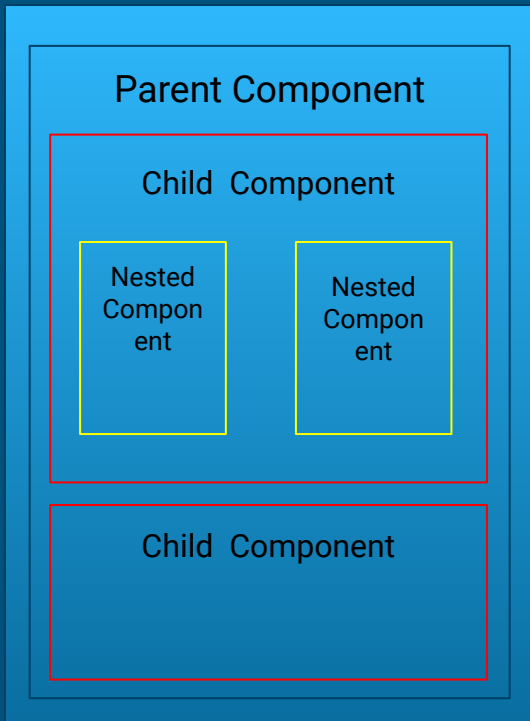
APP



- When using react each components is responsible for their state.
- You can pass state by passing props to the child components
- React encourages unidirectional data flow. That mean that data flows down from parent to child component.
- A child can only update a parents state by calling a callback function that its parent gave it.

Abstract Redux

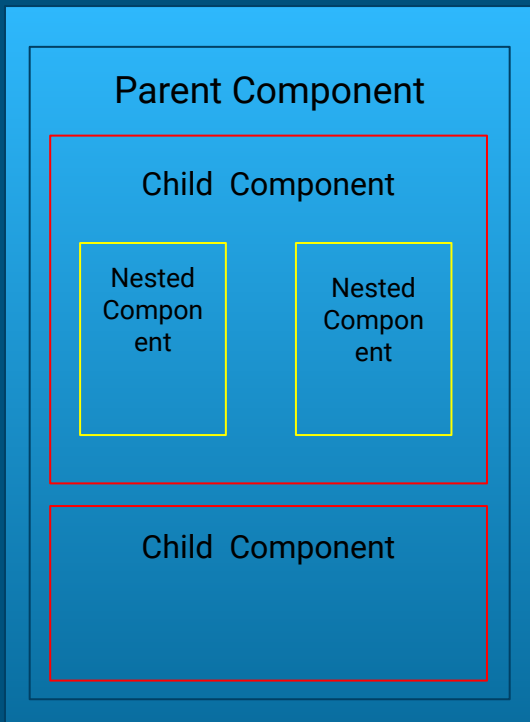
APP



- You can pass props from the parent component to the Child component

Abstract Redux

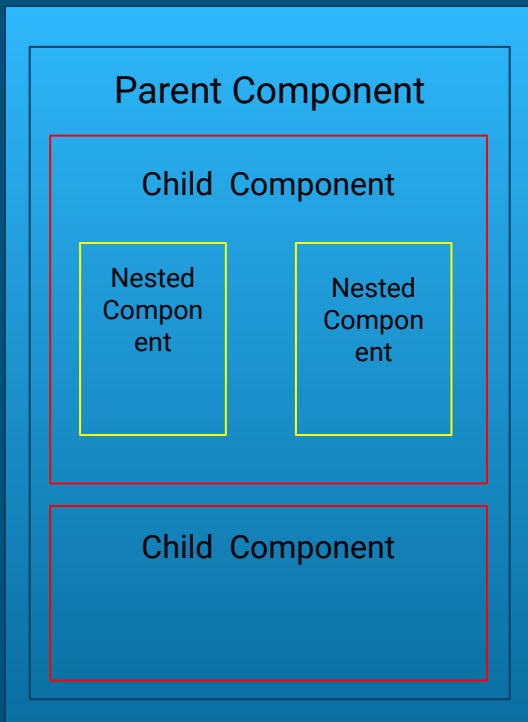
APP



- You can pass props from the parent component to the Child component
- And you can pass the same prop to the nested component.

Abstract Redux

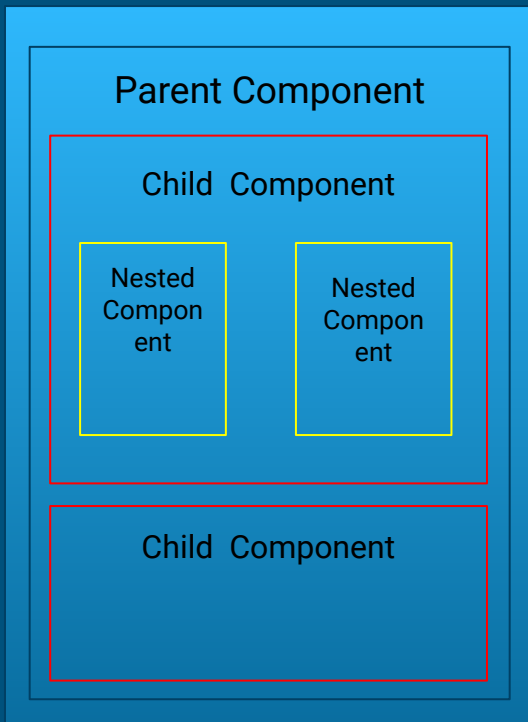
APP



- You can pass props from the parent component to the Child component
- And you can pass the same prop to the nested component.
- The problem here is, if the middle component stops working for some reason then the prop that has been passed is broken.

Abstract Redux

APP



- You can pass props from the parent component to the Child component
- And you can pass the same prop to the nested component.
- The problem here is, if the middle component stops working for some reason then the prop that has been passed is broken.

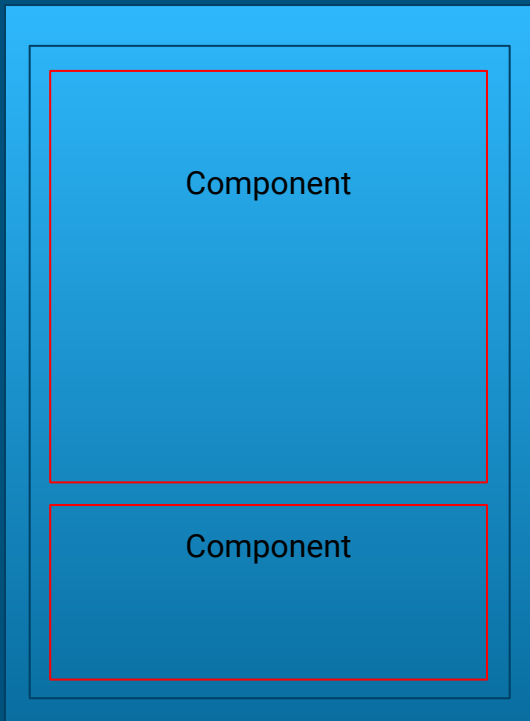


THIS IS CALLED PROP DRILLING



Abstract Redux

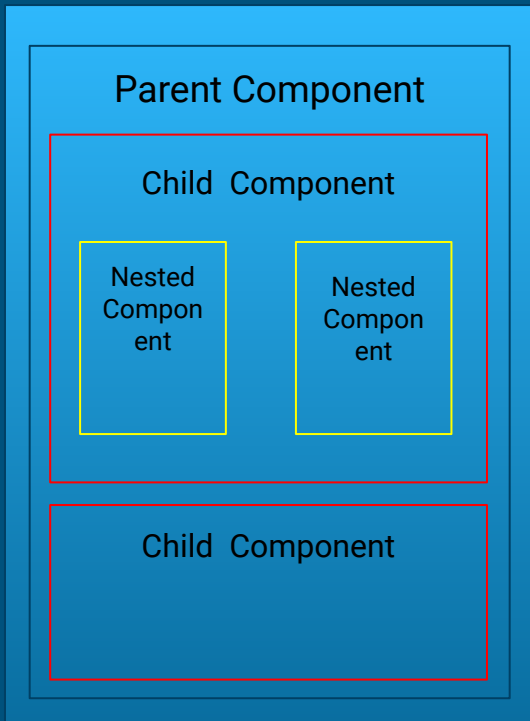
APP



- Another problem you are going to face is, what if you want to pass a prop from one adjacent component to another?

Abstract Redux

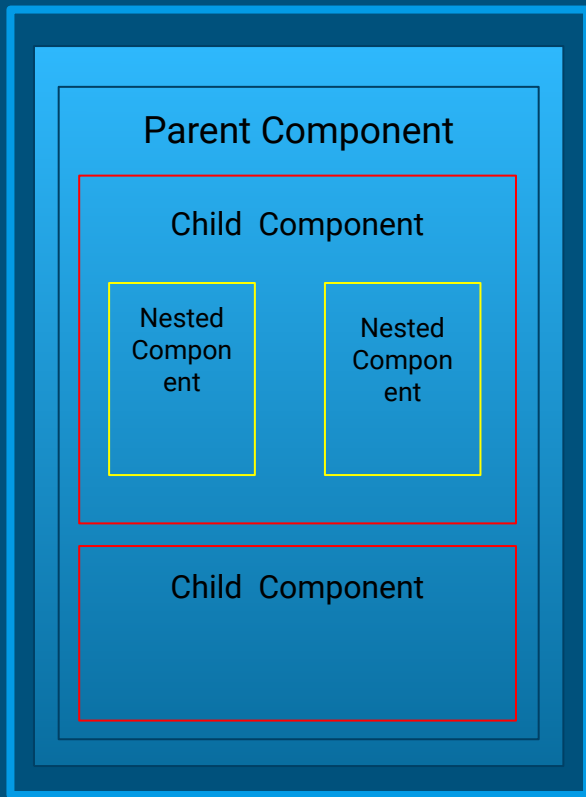
APP



- Redux wraps the whole entire app, in an HOC (Higher order component) and provides a global state.

Abstract Redux

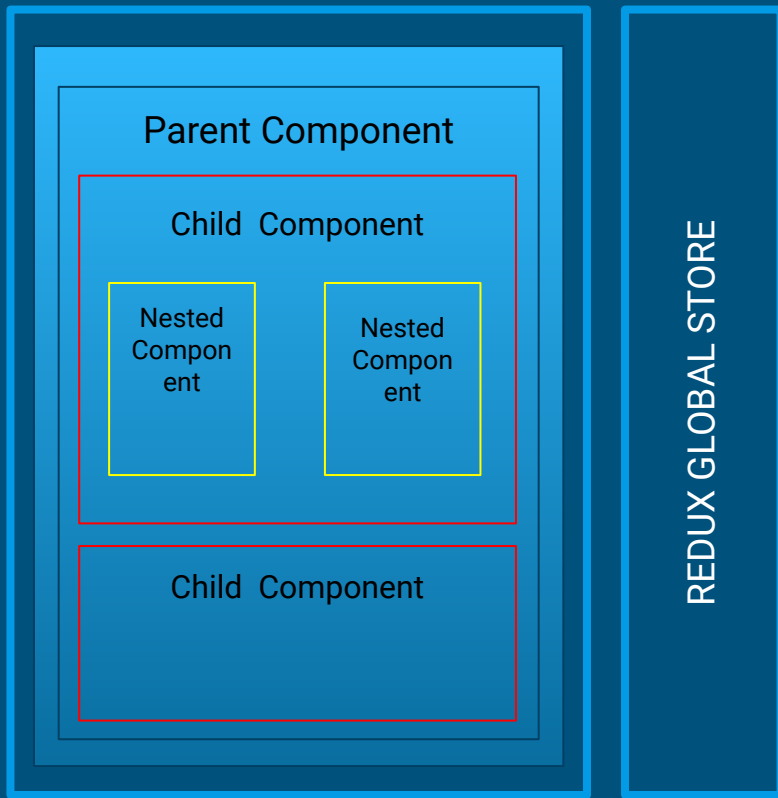
APP



- Redux wraps the whole entire app, in an HOC (Higher order component) and provides a global state.

Abstract Redux

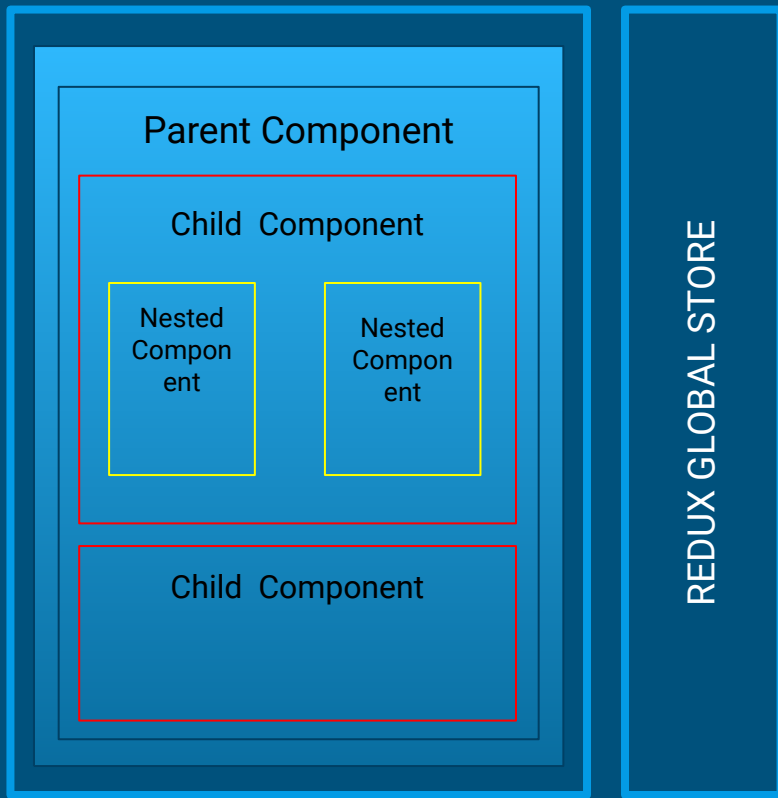
APP



- Redux wraps the whole entire app, in an HOC (Higher order component) and provides a global state.
- Creating what is called a global store

Abstract Redux

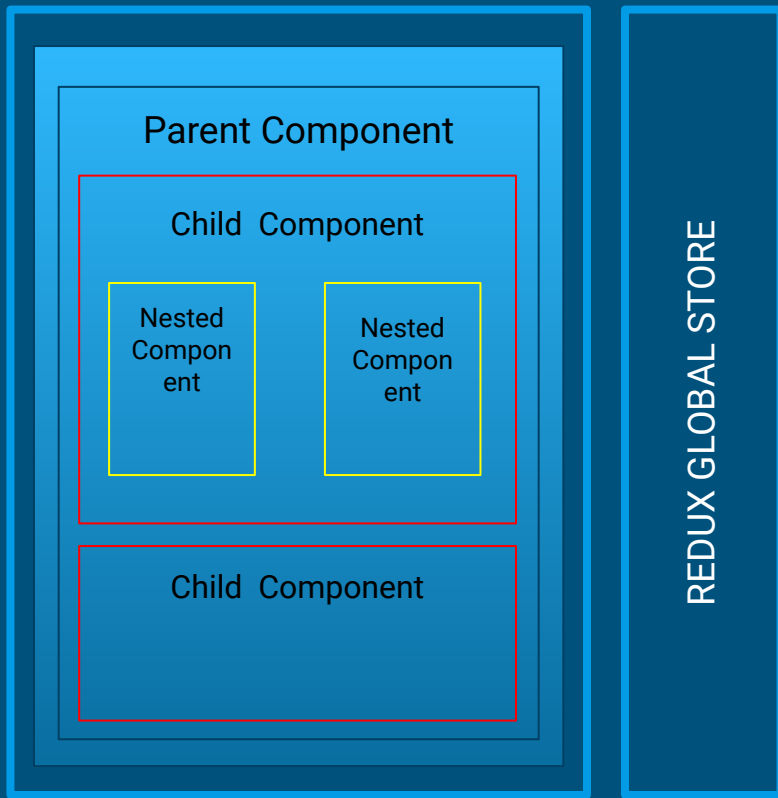
APP



- Redux wraps the whole entire app, in an HOC (Higher order component) and provides a global state.
- Creating what is called a global store
- With this architecture and functionality, we can now access any stored state in any given component.

Abstract Redux

APP



index.js

```
1 import store from "../Redux/store";
2 import { Provider } from "react-redux";
3
4 ReactDOM.render(
5   <Provider store={store}>
6     <App />
7   </Provider>,
8   document.getElementById("root")
9 );
```

Concept Redux

Concept Redux

- Redux has three building parts:

Concept Redux

- Redux has three building parts:
 - ACTIONS

Concept Redux

- Redux has three building parts:
 - ACTIONS
 - REDUCERS

Concept Redux

- Redux has three building parts:
 - ACTIONS
 - REDUCERS
 - STORE

Concept Redux

- ACTIONS

Concept Redux


- ACTIONS

Actions are pure functions that contain a certain piece of code to achieve something in your app, either storing a value that the user entered from a field, or fetching data from an api to keep in our store.

Concept Redux

- ACTIONS

Actions are pure functions that contain a certain piece of code to achieve something in your app, either storing a value that the user entered from a field, or fetching data from an api to keep in our store.




```
1 export const addTodo = (todo) => (dispatch) => {  
2   dispatch({  
3     type: "ADD_TODO",  
4     payload: todo,  
5   });  
6 };
```

Concept Redux

- ACTIONS

Actions are pure functions that contain a certain piece of code to achieve something in your app, either storing a value that the user entered from a field, or fetching data from an api to keep in our store.

An action always has to have a type when dispatched, and it can also have a payload but it is not required.



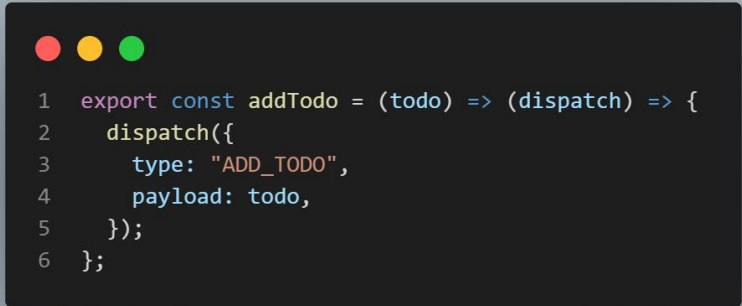
```
1 export const addTodo = (todo) => (dispatch) => {  
2   dispatch({  
3     type: "ADD_TODO",  
4     payload: todo,  
5   });  
6 };
```

Concept Redux

- ACTIONS

Actions are pure functions that contain a certain piece of code to achieve something in your app, either storing a value that the user entered from a field, or fetching data from an api to keep in our store.

An action always has to have a type when dispatched, and it can also have a payload but it is not required.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a JavaScript code snippet for a Redux action.

```
1 export const addTodo = (todo) => (dispatch) => {  
2   dispatch({  
3     type: "ADD_TODO",  
4     payload: todo,  
5   });  
6 };
```

You can think of an action as an event that describes something that happened in the application.

Concept Redux

- REDUCERS

Concept Redux

- REDUCERS

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.

Concept Redux

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.

- REDUCERS

```
1  const initialState = {
2    todos: [],
3  };
4
5  export const todosReducer = (state = initialState, action) => {
6    switch (action.type) {
7      case "ADD_TODO":
8        return {
9          ...state,
10         todos: action.payload
11       }
12     default:
13       return state;
14   }
```

Concept Redux

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.

When declaring a new reducer, you can predefine how your state object will look like and have starting values, or you can just leave it empty.

- REDUCERS

```
1  const initialState = {
2    todos: [],
3  };
4
5  export const todosReducer = (state = initialState, action) => {
6    switch (action.type) {
7      case "ADD_TODO":
8        return {
9          ...state,
10         todos: action.payload
11       }
12     default:
13       return state;
14   }
```

Concept Redux

- REDUCERS

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.

When declaring a new reducer, you can predefine how your state object will look like and have starting values, or you can just leave it empty.

```
1  const initialState = {
2    todos: [],
3  };
4
5  export const todosReducer = (state = initialState, action) => {
6    switch (action.type) {
7      case "ADD_TODO":
8        return {
9          ...state,
10         todos: action.payload
11       }
12     default:
13       return state;
14   }
```

You can think of a reducer as an event listener which handles events based on the received action (event) type.

Concept Redux

- STORE

Concept Redux

- STORE

The current Redux application STATE lives in an object called **store**.

Concept Redux

- STORE

The current Redux application STATE lives in an object called **store**.

It is the object that holds the entire applications state and provides a few helper methods to access the state. The entire state is represented by a single store.

Concept Redux

- STORE

The current Redux application STATE lives in an object called **store**.

It is the object that holds the entire applications state and provides a few helper methods to access the state. The entire state is represented by a single store.

The store is created by passing in a reducer or reducers.

Concept Redux

- STORE

The current Redux application STATE lives in an object called **store**.

It is the object that holds the entire applications state and provides a few helper methods to access the state. The entire state is represented by a single store.

The store is created by passing in a reducer or reducers.

```
1  const reducers = combineReducers({
2    todos: todosReducer,
3  });
4
5  const middleware = [thunk];
6
7  const store = createStore(
8    reducers,
9    composeWithDevTools(applyMiddleware(...middleware))
10 );
```

Concept Redux

- DISPATCH

Concept Redux

- DISPATCH

Redux has a method called *dispatch*.
This is the only way to interact with the store. By dispatching an action we interact with the reducer that then decides what to do with the state.

Concept Redux

- DISPATCH

Redux has a method called *dispatch*. This is the only way to interact with the store. By dispatching an action we interact with the reducer that then decides what to do with the state.



```
1 dispatch(addTodo(newTodo));
```

Concept Redux

- DISPATCH

Redux has a method called *dispatch*. This is the only way to interact with the store. By dispatching an action we interact with the reducer that then decides what to do with the state.



```
1 dispatch(addTodo(newTodo));
```

You can think of dispatching actions as “triggering an event” in the application.

Concept Redux

- IMMUTABILITY

Concept Redux

- IMMUTABILITY

Mutable means “changeable”. If something is “immutable”, it can never be changed.

Concept Redux

- IMMUTABILITY

Mutable means “changeable”. If something is “immutable”, it can never be changed.

JavaScript objects and arrays are all mutable by default. If we create an object, we can change the contents of its fields. Same with arrays.

Concept Redux

- IMMUTABILITY

Mutable means “changeable”. If something is “immutable”, it can never be changed.

JavaScript objects and arrays are all mutable by default. If we create an object, we can change the contents of its fields. Same with arrays.

In order to update values immutably, our code must make copies of existing objects/arrays, and then modify the copies.

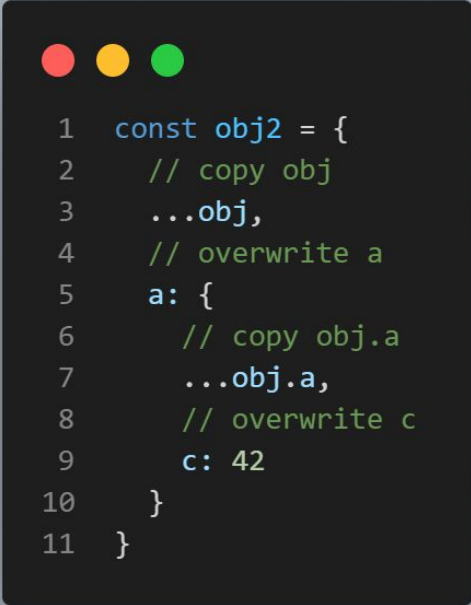
Concept Redux

- IMMUTABILITY

Mutable means “changeable”. If something is “immutable”, it can never be changed.

JavaScript objects and arrays are all mutable by default. If we create an object, we can change the contents of its fields. Same with arrays.

In order to update values immutably, our code must make copies of existing objects/arrays, and then modify the copies.



```
1  const obj2 = {  
2    // copy obj  
3    ...obj,  
4    // overwrite a  
5    a: {  
6      // copy obj.a  
7      ...obj.a,  
8      // overwrite c  
9      c: 42  
10   }  
11 }
```

Concept Redux

- DEVELOPERS TOOLS

Concept Redux

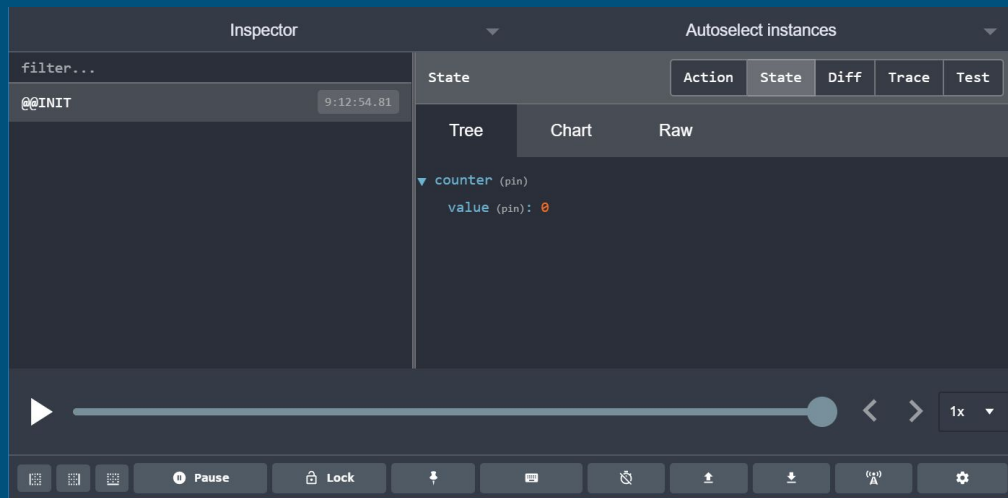
- DEVELOPERS TOOLS

To make working with Redux easier, there's the Developers Tools that allows us to visualize state changes over time (using “time travel”), real-time changes, actions, and the current state.

Concept Redux

To make working with Redux, there's the Developer Tools that allows us to visualize state changes over time (using "time travel"), real-time changes, actions, and the current state.

- DEVELOPER TOOLS



That is Redux in a nutshell

If you are still confused, don't worry it is a bit confusing, but practice makes perfection.

Now let's build a Redux app...



Pedro Gonçalves