



What is REDUX?





Redux is a JavaScript library for **managing the state** of applications.

It provides a way to **centralize** the state of an application in a **single store**, making it easier to debug, test, and reason about the state changes in the application.

One particularly cool feature of Redux is its support for **time travel**. With Redux DevTools, developers can **inspect** the state of their application at **any point in time**, including past and future states.

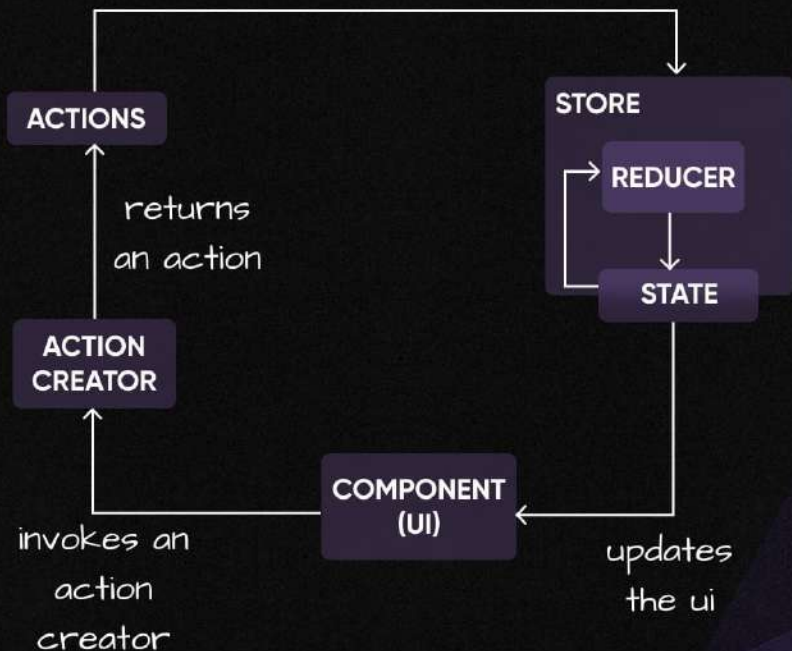
This makes it easier to debug and understand the state changes in the application.



Swipe →




action is dispatched and
gets forwarded to
reducer



Store Creation

In React, "store" refers to a centralized **container** that **holds the state** of your application. It's where you keep all of the data that your React components need to render, and it's managed using a **state management library** like Redux or MobX.

To create a Redux store, use the **createStore** function from the **redux** library, and pass in your root reducer as an argument.




```
import { createStore } from 'redux';  
import rootReducer from './reducers';  
  
const store = createStore(rootReducer);
```



Action Creation

Actions in Redux are plain **objects that describe changes** to the state. To create an action, define an object with a **type** property and any other data needed to describe the change.



```
const addTodo = (text) => {  
  return {  
    type: 'ADD_TODO',  
    text  
  };  
};
```



Dispatching Actions

To dispatch an action and update the state, call the **dispatch** method on your store and pass in the action as an argument.




```
store.dispatch(addTodo('Learn Redux'));
```



Reducer Functions

Reducers in Redux are pure functions that **take in the current state** and an action and **return the next state**.



```
const todoReducer = (state = [],
action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return [
        ...state,
        {
          text: action.text,
          completed: false
        }
      ];
    default:
      return state;
  }
};
```



Combining Reducers

If your application has multiple reducers, you can use the `combineReducers` function from the `redux` library to **combine** them into a **single root reducer**.



```
import { combineReducers } from
'redux';

const rootReducer =
combineReducers({
  todos: todoReducer,
  ...
});
```



Connecting to React Components

To connect your Redux store to React components, use the **connect** function from the **react-redux** library.

```
import { connect } from 'react-redux';

const TodoList = ({ todos }) => (
  <ul>
    {todos.map((todo, index) => (
      <li key={index}>{todo.text}</li>
    ))}
  </ul>
);

const mapStateToProps = (state) => {
  return {
    todos: state.todos
  };
};

export default
connect(mapStateToProps)(TodoList);
```



Do you find it helpful?

let me know down in the
comments !



Slobodan Gajić

Content Creator



FOLLOW FOR MORE