

# State Management



**In React**

**01 State Management**  
Definición e Importancia

**02 Redux**  
Definición y ejemplo

**03 Zustand**  
Definición y ejemplo

**04 Pros y Cons**  
Redux y Zustand

**05 Otros Conceptos**  
Medianos-Avanzados

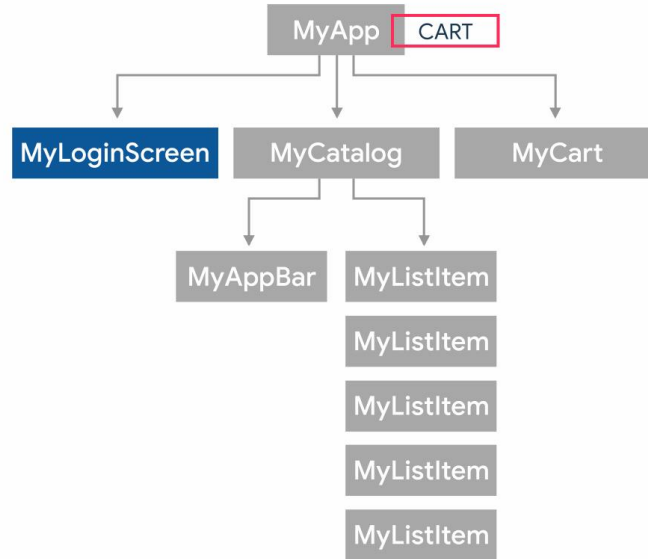
## TABLE OF CONTENTS

# Welcome

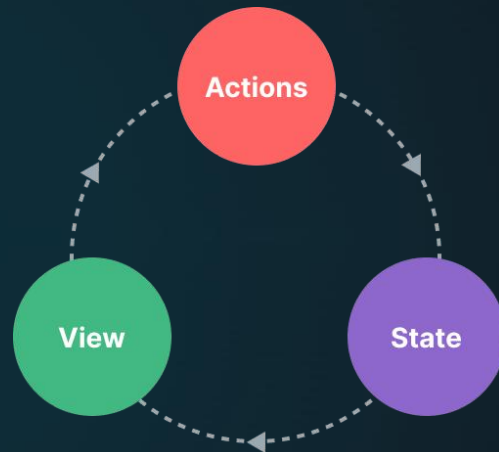
Login

Password

Enter

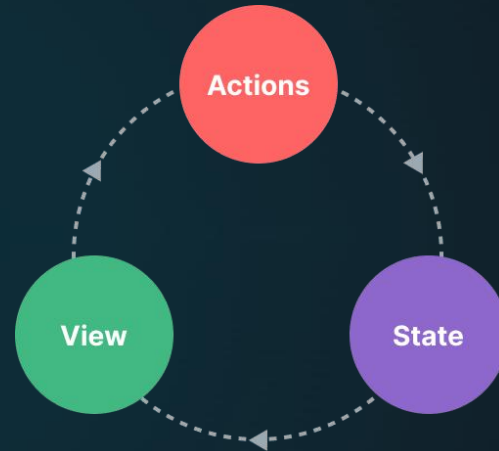


# 01 Qué es State Management



"one-way data flow"

# Qué es State Management

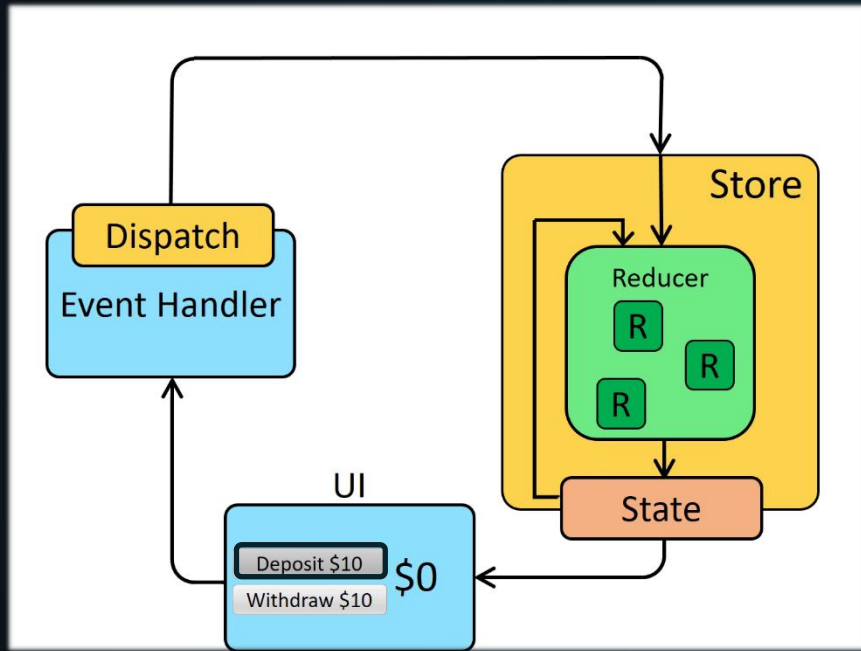


"one-way data flow"

## 02 REDUX



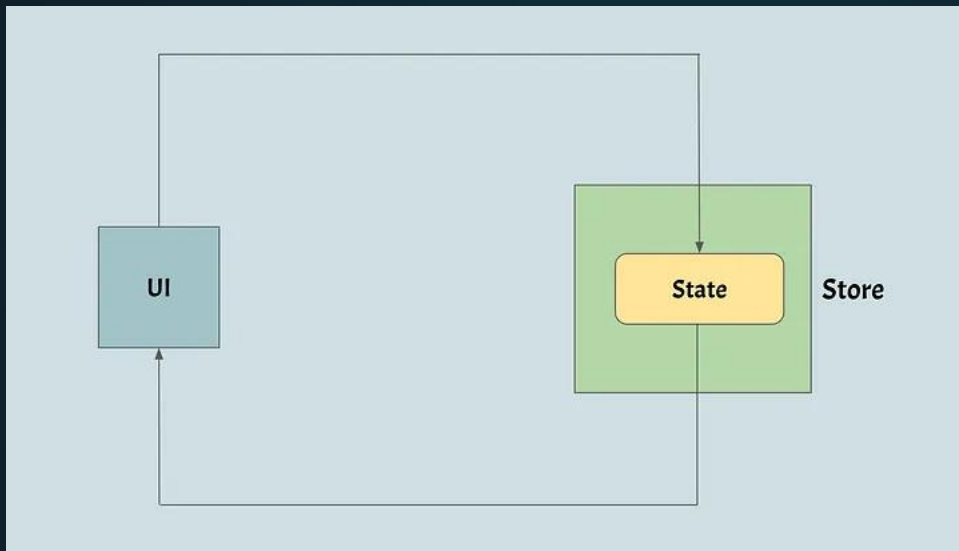
Es una biblioteca para manejar el **estado** global en aplicaciones. Proporciona un contenedor del estado de la aplicación y permite la gestión de estados complejos de manera más estructurada.



## ¿Cómo funciona?

Store  
Action  
Dispatch  
Reducers

# zustand .



¿Cómo funciona?





# To-Do List

en Redux

# . redux

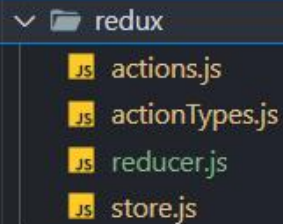


```
{  
  text: string = "Investigar logging",  
  completed: bool = false  
}
```

# . redux

01

## Estructura de carpeta




A screenshot of a file explorer window showing the contents of a folder named 'redux'. The folder is expanded, revealing four files: 'actions.js', 'actionTypes.js', 'reducer.js', and 'store.js'. Each file is preceded by a small yellow icon with the letters 'JS'.

- JS actions.js
- JS actionTypes.js
- JS reducer.js
- JS store.js

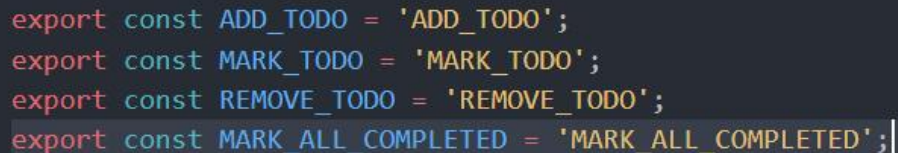
02

## Action Types



A small thumbnail image of the 'actionTypes.js' file, showing the file name and a yellow 'JS' icon.

JS actionTypes.js M



A screenshot of the 'actionTypes.js' file content, showing four exported constants for action types. The file is highlighted with a pink border.

```
// actionTypes.js
export const ADD_TODO = 'ADD_TODO';
export const MARK_TODO = 'MARK_TODO';
export const REMOVE_TODO = 'REMOVE_TODO';
export const MARK_ALL_COMPLETED = 'MARK_ALL_COMPLETED';
```

# . redux

03

## Actions

```
import {  
  ADD_TODO,  
  MARK_TODO,  
  REMOVE_TODO,  
  MARK_ALL_COMPLETED,  
} from './actionTypes';
```

Import

```
export const addTodo = (text) => ({  
  type: ADD_TODO,  
  payload: { text },  
});
```

```
export const markTodo = (id) => ({  
  type: MARK_TODO,  
  payload: { id },  
});
```

```
export const removeTodo = (id) => ({  
  type: REMOVE_TODO,  
  payload: { id },  
});
```

```
export const markAllCompleted = () => ({  
  type: MARK_ALL_COMPLETED,  
});
```

# . redux

04

## Reducers

```
2 import {  
3   ADD_TODO,  
4   MARK_TODO,  
5   REMOVE_TODO,  
6   MARK_ALL_COMPLETED,  
7 } from './actionTypes';  
8  
9 const initialState = { todos: [] };
```

JS reducer.js M

```
11 const todoReducer = (state = initialState, action) => {  
12   switch (action.type) {  
13     case ADD_TODO:  
14       return {  
15         todos: [...state.todos, { text: action.payload.text, completed: false }],  
16       };  
17     case REMOVE_TODO:  
18       return {  
19         todos: state.todos.filter((todo, index) => index !== action.payload.id),  
20       };  
21  
22     case MARK_TODO:  
23       return {  
24         todos: state.todos.map((todo, index) =>  
25           index === action.payload.id ? { ...todo, completed: !todo.completed } : todo  
26         ),  
27       };  
28  
29     case MARK_ALL_COMPLETED:  
30       return {  
31         todos: state.todos.map((todo) => ({ ...todo, completed: true })),  
32         filter: state.filter,  
33         searchTerm: state.searchTerm,  
34       };  
35  
36     default:  
37       return state;  
38   }  
}
```

# . redux

## Add Todo

```
export const addTodo = (text) => ({  
  type: ADD_TODO,  
  payload: { text },  
});
```

Action

```
case ADD_TODO:  
  return {  
    todos: [...state.todos, { text: action.payload.text, completed: false }],  
  };
```

**spread operator:** ...state.todos

**text:** texto del payload

**completed:** inicialmente en falso

# . redux

04

Store

```
import { createStore } from 'redux';
import todoReducer from './reducer';

const store = createStore(todoReducer);

export default store;
```

```
function App() {
  return (
    <Provider store={store}>
      <Todo/>
    </Provider>
  )
}
```

En App.jsx  
Encapsulamos nuestra aplicacion con el  
componente <Provider>

# . redux

05

Utilizarlo

TodoList ->

```
1 import { useSelector } from "react-redux";
2 import TodoItem from "../TodoItem";
3
4 const TodoList = () => {
5   const Todos = useSelector((state) => state.todos);
6
7   return (
8     <ul>
9       {Todos.map((todo, index) => (
10         <TodoItem key={index} todo={todo} index={index} />
11       ))}
12     </ul>
13   );
14 };
15
16 export default TodoList;
```



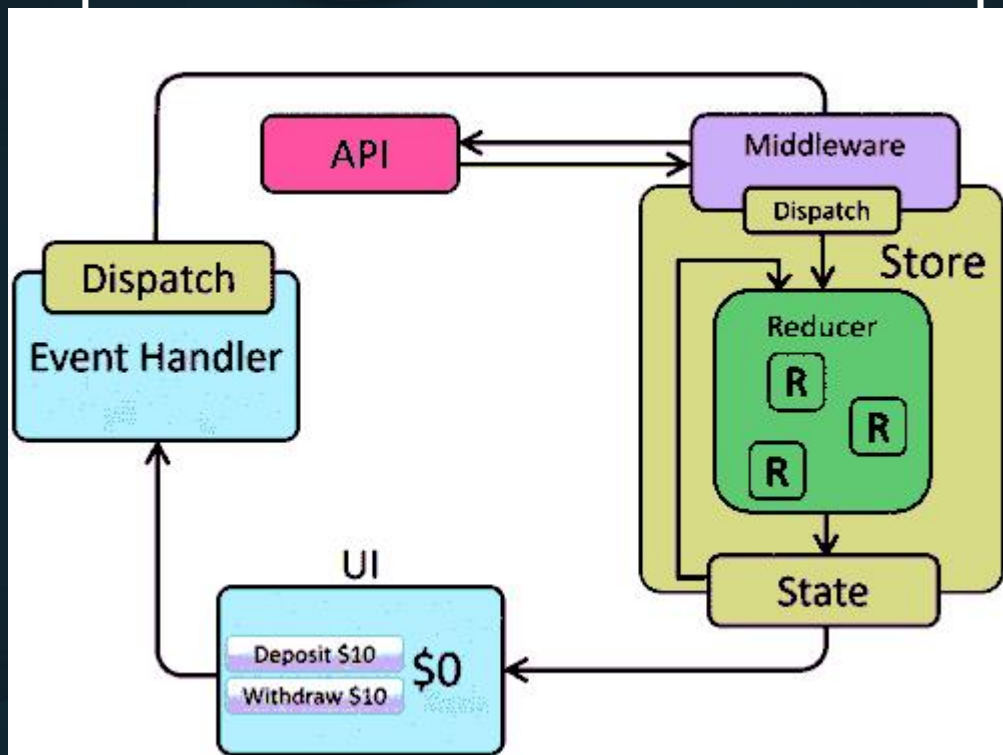
# . redux

05

Utilizarlo

TodoItem ->

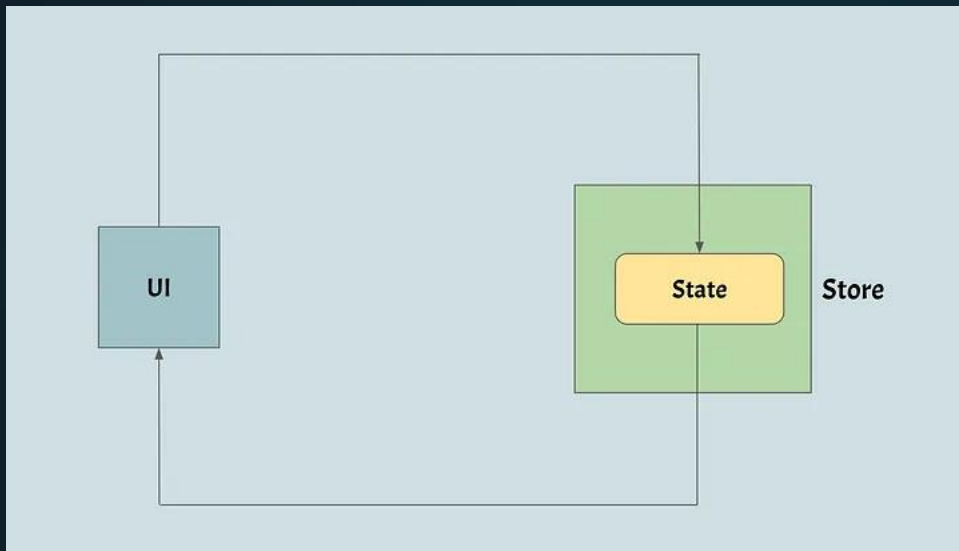
```
1 import { useDispatch } from 'react-redux';
2 import { markTodo, removeTodo } from '../redux/actions';
3 import { FaTrash, FaCheck, FaTimes } from 'react-icons/fa';
4
5 const TodoItem = ({ todo, index }) => {
6   const dispatch = useDispatch();
7
8   return (
9     <li className="flex flex-col sm:flex-row sm:items-center justify-between border-b-2 py-2 gap-4">
10       <div className="flex items-center">...
11     </div>
12     <div className="space-x-3 ml-8">
13       <button
14         className="mr-2 text-sm bg-red-500 text-white sm:px-2 px-1 py-1 rounded"
15         onClick={() => dispatch(removeTodo(index))}
16       />
17       <FaTrash />
18     </button>
19     {!todo.completed && (
20       <button
21         className="text-sm bg-green-500 text-white sm:px-2 px-1 py-1 rounded"
22         onClick={() => dispatch(markTodo(index))}
23       />
24       <FaCheck />
25     )}
26   )}
27 }
```





Es una biblioteca **minimalista** para manejar el **estado** global. Proporciona una API **simple** y funcional para definir y manipular el estado, ofreciendo un enfoque **más ligero y flexible** que Redux.

# zustand .



¿Cómo funciona?



# To-Do List

en Zustand

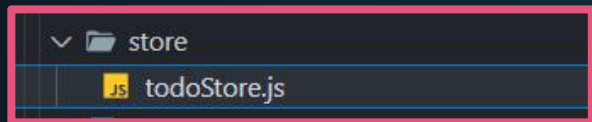
# zustand .



```
{  
  id: Date= Date.now(),  
  text: string = "Investigar logging",  
  completed: bool = false  
}
```

# . zustand .

## 01 Estructura de Carpeta



## 02 Store

```
5  const useTodoStore = create(devtools(set => ({ //Based on fucntions
6    todos: [], //Array of todos
7    // Set function updates the state. Argument (state) is the current state and returns the new state
8    addTo: (text) => set((state) => ({ todos: [...state.todos, {id: Date.now(), text, completed: false}] })),
9    removeTo: (id) => set((state) => ({ todos: state.todos.filter(t => t.id !== id) })),
10   markTo: (id) => set((state) => ({ todos: state.todos.map(t => t.id === id ? {...t, completed: !t.completed} : t) })),
11  })));
```

# . zustand .

## Paso a paso

```
const useTodoStore = create(set => ({
```

```
  todos: [],
```

```
  addTodo: (text) => set((state) => ({ todos: [...state.todos, {id: Date.now(), text, completed: false}] })),
```

```
  removeTodo: (id) => set((state) => ({ todos: state.todos.filter(t => t.id !== id) })),
```

```
  markTodo: (id) => set((state) => ({ todos: state.todos.map(t => t.id === id ? {...t, completed: !t.completed} : t) })),
```



04

---

# Pros y Cons

Redux y Zustand

# redux .

## PROS

- Centralización del estado
- Previsibilidad
- Gran ecosistema

## CONS

- Boilerplate
- Curva de aprendizaje
- Posible complejidad

{ }

## PROS

- Simplicidad
- Menos boilerplate
- Hooks nativos de React
- Rendimiento

## CONS

- Menor escalabilidad
- Menor ecosistema
- Menos control sobre el flujo de datos



**05**

---

# **Conceptos “avanzados”**

## Redux Toolkit

**Suscribirse a cambios de estado**

**Cuando utilizar: useContext**

## Middlewares

Permiten agregar funcionalidad a los Stores de Redux, manejo de errores, logging, persist..

**Cuando utilizar: useEffect,  
useState**

## Obtener datos (API)

### Acciones asíncronas

por ejemplo hacer una petición AJAX a una API

**“Definir que guardará nuestro estado”**

Does anyone have any  
questions?

**THANKS!**