REDUX

REDUX

- **Que aprenderemos en este apartado?**

1. **¿qué es Redux?**
2. **Principios de Redux**
3. **Estado en Redux**
4. **React-redux**
5. **Store**
6. **Acciones**
7. **Acciones asíncronas**
8. **ActionTypes**
9. **Reducers**
10. **Middlewares**
11. **Redux DevTools**
12. **Tests unitarios**

QUE ES REDUX?

Sitio oficial: http://es.redux.js.org/

Redux es una librería para controlar el estado de nuestras aplicaciones web fácilmente, de una forma consistente entre cliente y servidor, testeable y con una gran experiencia de desarrollo

QUE ES REDUX?

Redux es un contenedor predecible del estado de aplicaciones JavaScript. Te ayuda a escribir aplicaciones que se comportan de manera consistente, corren en distintos ambientes (cliente, servidor y nativo), y son fáciles de probar.

Es una herramienta de uso libre que nos deja almacenar todo nuestro Estado de una aplicación en un solo lugar.

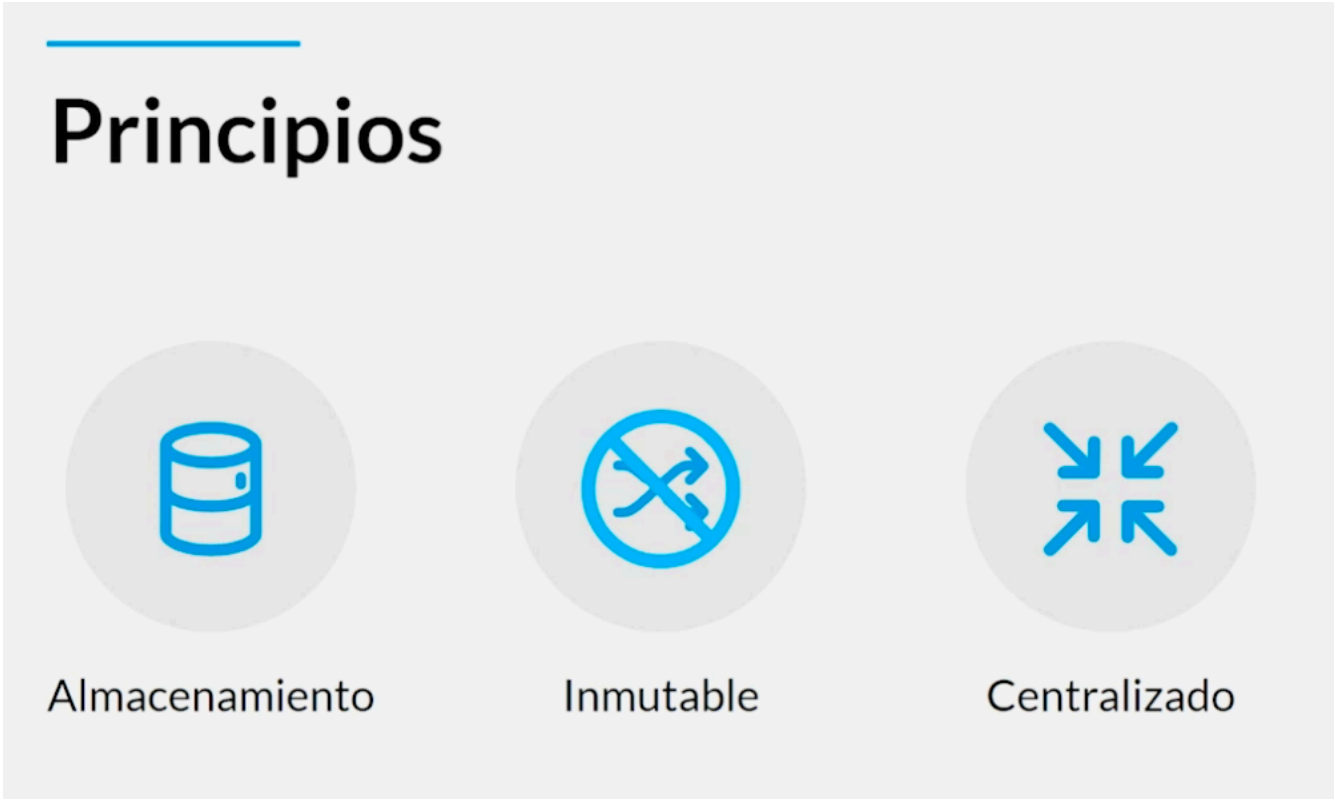
INSTALACION DE REDUX

Se necesitan 2 librerias:

- Redux
- React-redux

```
$ npm install redux react-redux
```

Principios de Redux



Principios de Redux

3 Principios

Una sola fuente de la verdad

Todo el estado de tu aplicación esta contenido en un único store

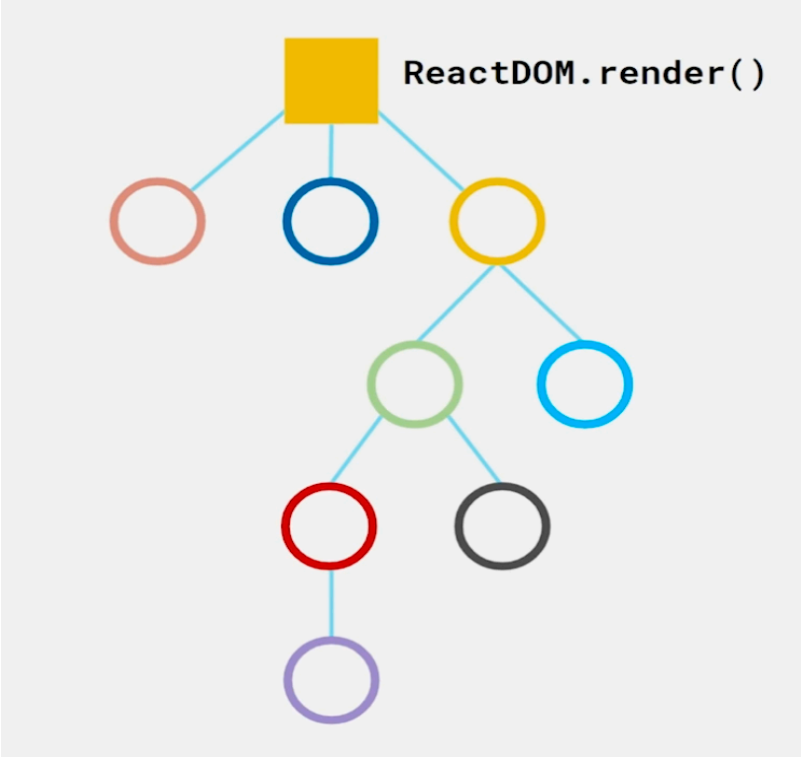
El estado es de solo lectura

La única forma de modificar el estado es emitir una acción que indique que cambió

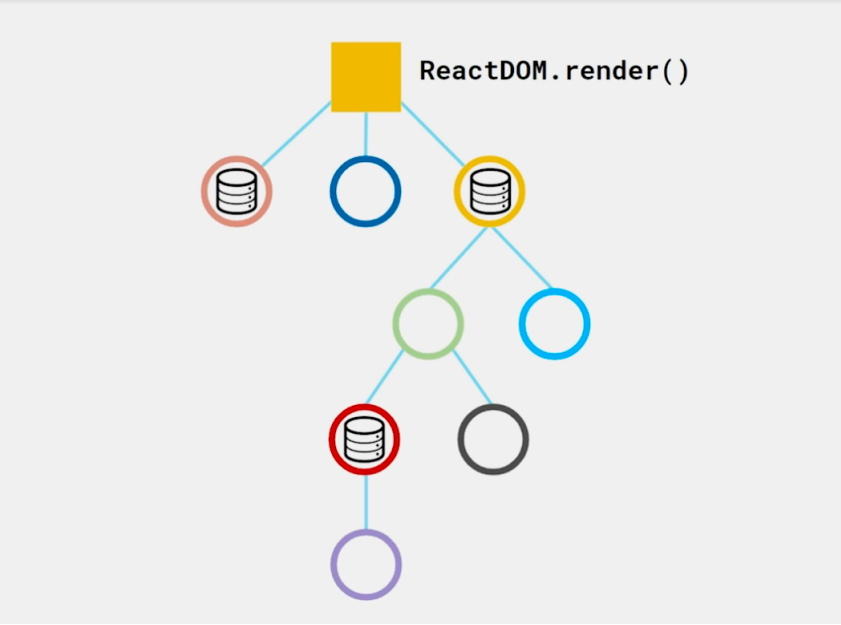
Los cambios se hacen mediante funciones puras

Para controlar como el store es modificado por las acciones se usan reducers puros

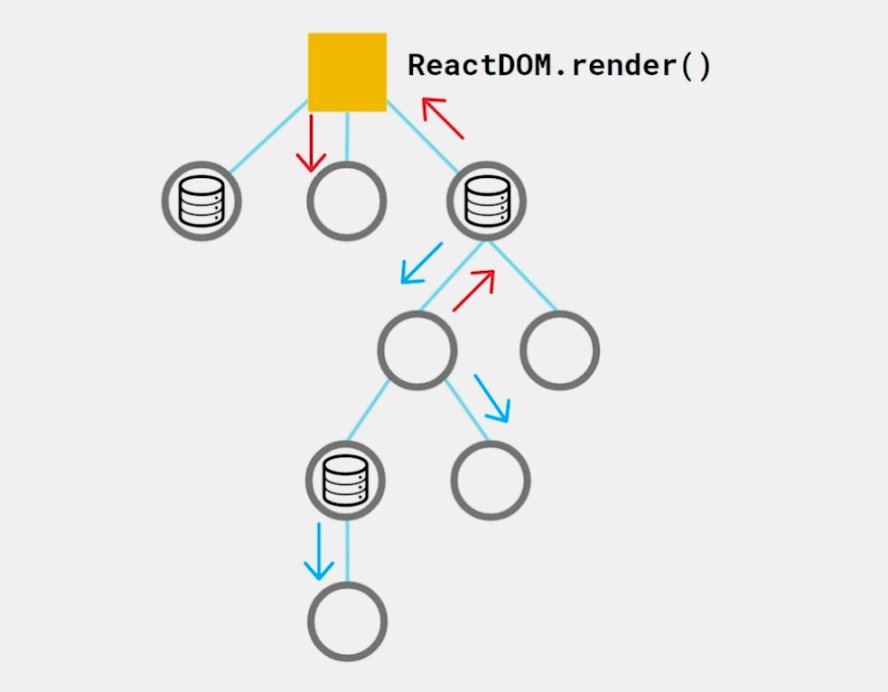
COMO FUNCIONA?



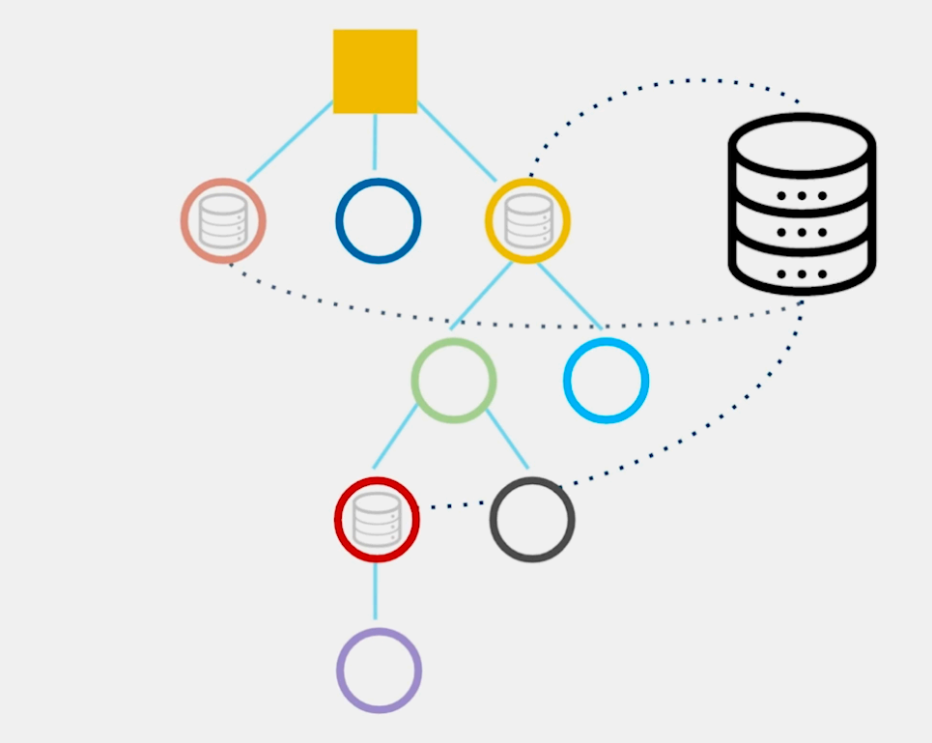
COMO FUNCIONA?



COMO FUNCIONA?



COMO FUNCIONA?

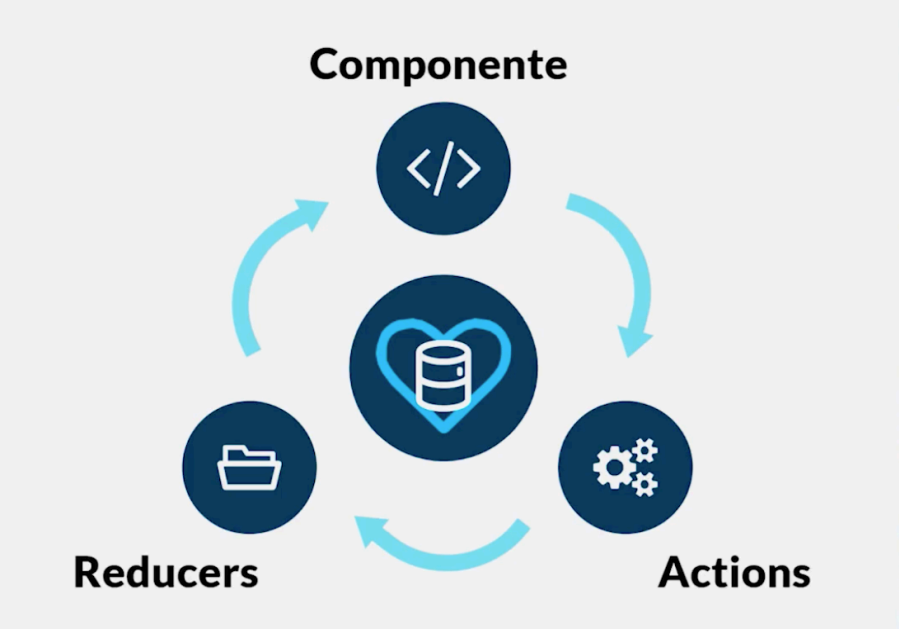


Fases de Redux

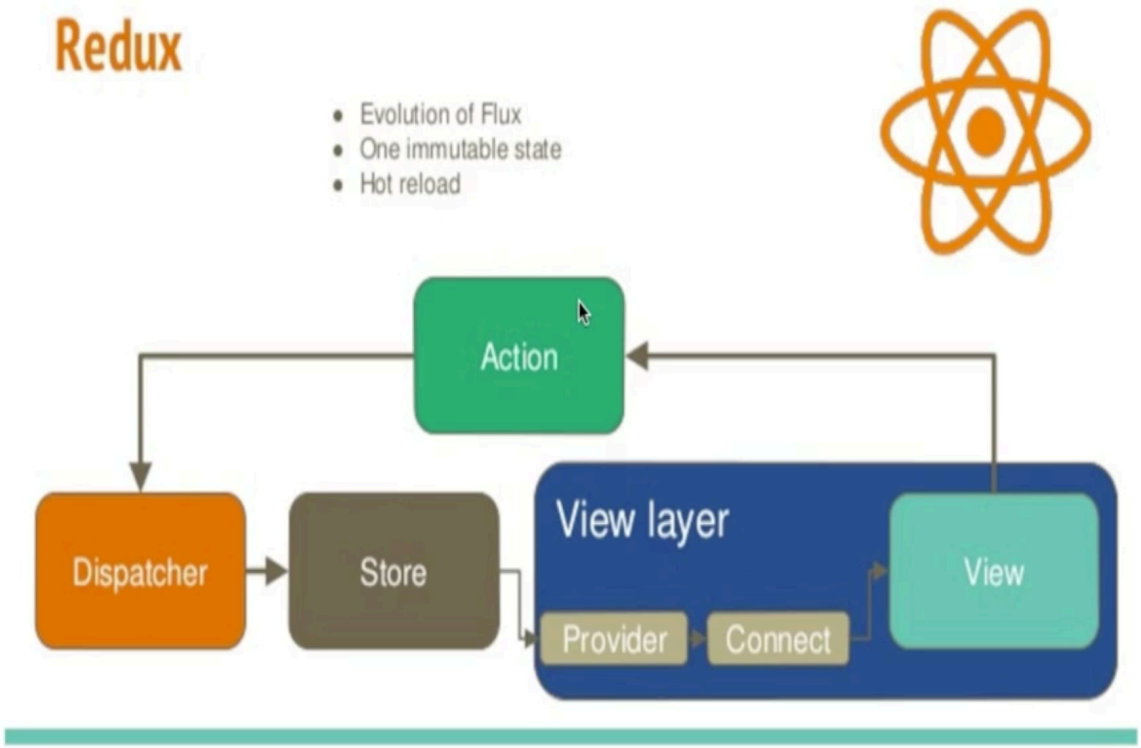
Fases de Redux

El componente se comunica con los actions para obtener información.

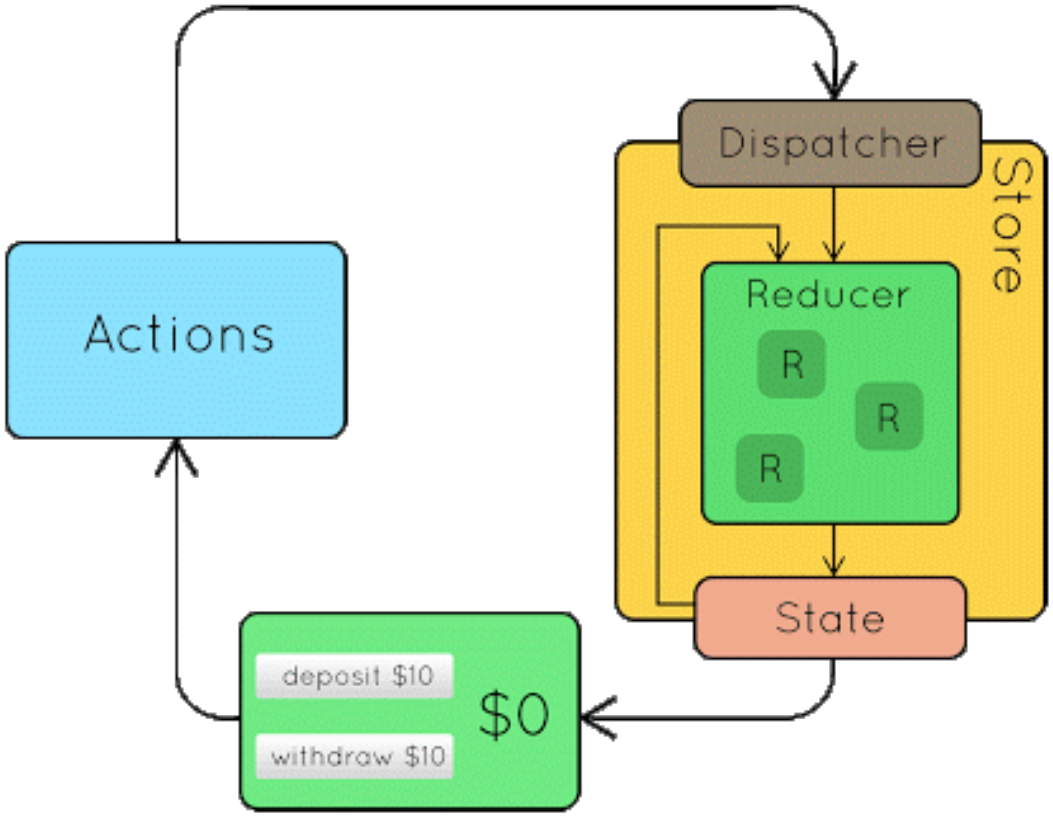
Este se lo pide al reducer que es quien tiene el estado y este ultimo se los sirve al componente que tras recibirlo se renderiza.



FUNCIONAMIENTO



FUNCIONAMIENTO



STORE

El *store* tiene las siguientes responsabilidades:

- Contiene el estado de la aplicación
- Permite el acceso al estado vía getState()
- Permite que el estado sea actualizado vía dispatch(action)
- Registra los *listeners* vía subscribe(listener)
- Maneja la anulación del registro de los *listeners* via el retorno de la función de subscribe(listener)

Reducers

Los reducers reciben el estado anterior, es decir, los datos que estaban disponibles anteriormente en la aplicación, antes de llamar a una acción y devuelven un nuevo estado, es decir, los datos de la aplicación actualizados basados en la acción ejecutada.

```
function reducer (state, action) {
   switch(action.type) {
     case 'ADD_ITEM':
       return state.concat(action.item);
     case 'REMOVE_ITEM':
     ...
     default:
       return state;
   }
}
```

Acciones

No podemos modificar el estado directamente, sólo podemos leer de él para representarlo en la vista y si queremos modificarlo, lo tenemos que hacer a través de **acciones**.

Una acción es simplemente un objeto JavaScript que incluye al menos un atributo type que indica el tipo de acción que estamos emitiendo

```
function loadProducts(products) {
  return {
    type: 'LOAD_PRODUCTS',
    products
  }
}
```

MIDDLEWARES

Los *middlewares* son funciones que se ejecutan de forma intermedia mientras se emite una acción y cambia el estado. Podemos añadir multitud de ellos,

```
import { createStore, combineReducers, applyMiddleware } from 'redux'

let todoApp = combineReducers(reducers)
let store = createStore(
  todoApp,
  // applyMiddleware() le indica a createStore() cómo manejar el middleware
  applyMiddleware(logger, crashReporter)
)
```

MIDDLEWARES

```
const logger = store => next => action => {
  console.log('dispatching', action)
  let result = next(action)
  console.log('next state', store.getState())
  return result
}

const crashReporter = store => next => action => {
  try {
    return next(action)
  } catch (err) {
    console.error('Caught an exception!', err)
    Raven.captureException(err, {
      extra: {
        action,
        state: store.getState()
      }
    })
    throw err
  }
}
```

Función createStore

Esta función, como su nombre indica, crea la *store* central donde se almacenará el estado global de la aplicación.

La función recibe como parámetro un *reducer* y opcionalmente un estado inicial y un *enhancer* que nos sirve para más adelante añadir *Middlewares*. Y devuelve la *store* creada.

```
const store = redux.createStore(reducer, [initialState], [enhancer])
```

Funcion combineReducers

```
function userReducer (state = initialState.user, action) {...}
function productsReducer (state = initialState.products, action) {...}
function currentProductReducer (state = initialState.currentProduct, action) {...}
function cartReducer (state = initialState.cart, action) {...}

const rootReducer = combineReducers({
  user: userReducer,
  products: productsReducer,
  currentProduct: currentProductReducer,
  cart: cartReducer
});
```

FUNCIÓN APPLYMIDDLEWARE

```
let store = createStore(
  todos,
  [ 'Use Redux' ],
  applyMiddleware(logger)
)
```

Redux DevTools

https://github.com/gaearon/redux-devtools