

# Introducción a **React**

Ponente: **Juan Canepa**

## Consideraciones antes de empezar a usar React / Redux

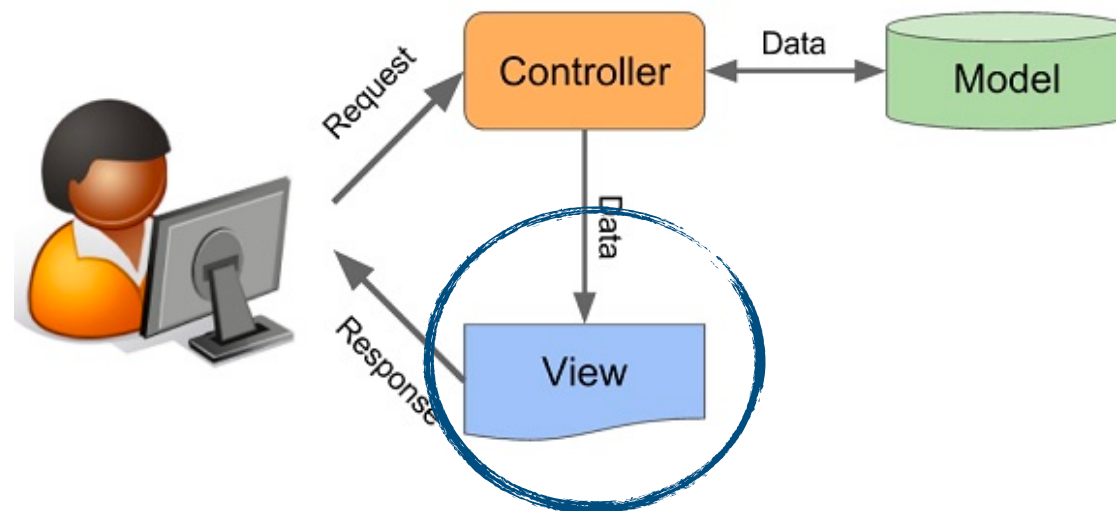
- Familiarizarse con **ES6 / ES7**, **clases**, trabajar con **promesas** (incluyendo **async / await**), los conceptos nuevos de javascript: **destructuring**, **fat arrow function**, **spread operator**, manejar **inmutabilidad**
- Comprender el concepto de **transpilar** y entender como funciona **babel**.
- No esta demás entender los conceptos de **programación funcional** como **funciones de puras**, funciones de **primer clase** y funciones de **orden superior**
- Olvidarse de **var** en favor **de let** y **const**
- Entender las apis funcionales como **lodash**, **ramdajs** o la nativa de **ES6 / ES7** (**map**, **reduce**, **filter**, **find**, **some**, **set**)
- Familiarizarse con el patron **FLUX**, para un mejor entendimiento de lo que es redux

## ¿ Que es React ?

Es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre, han participado en el proyecto más de mil desarrolladores diferentes.



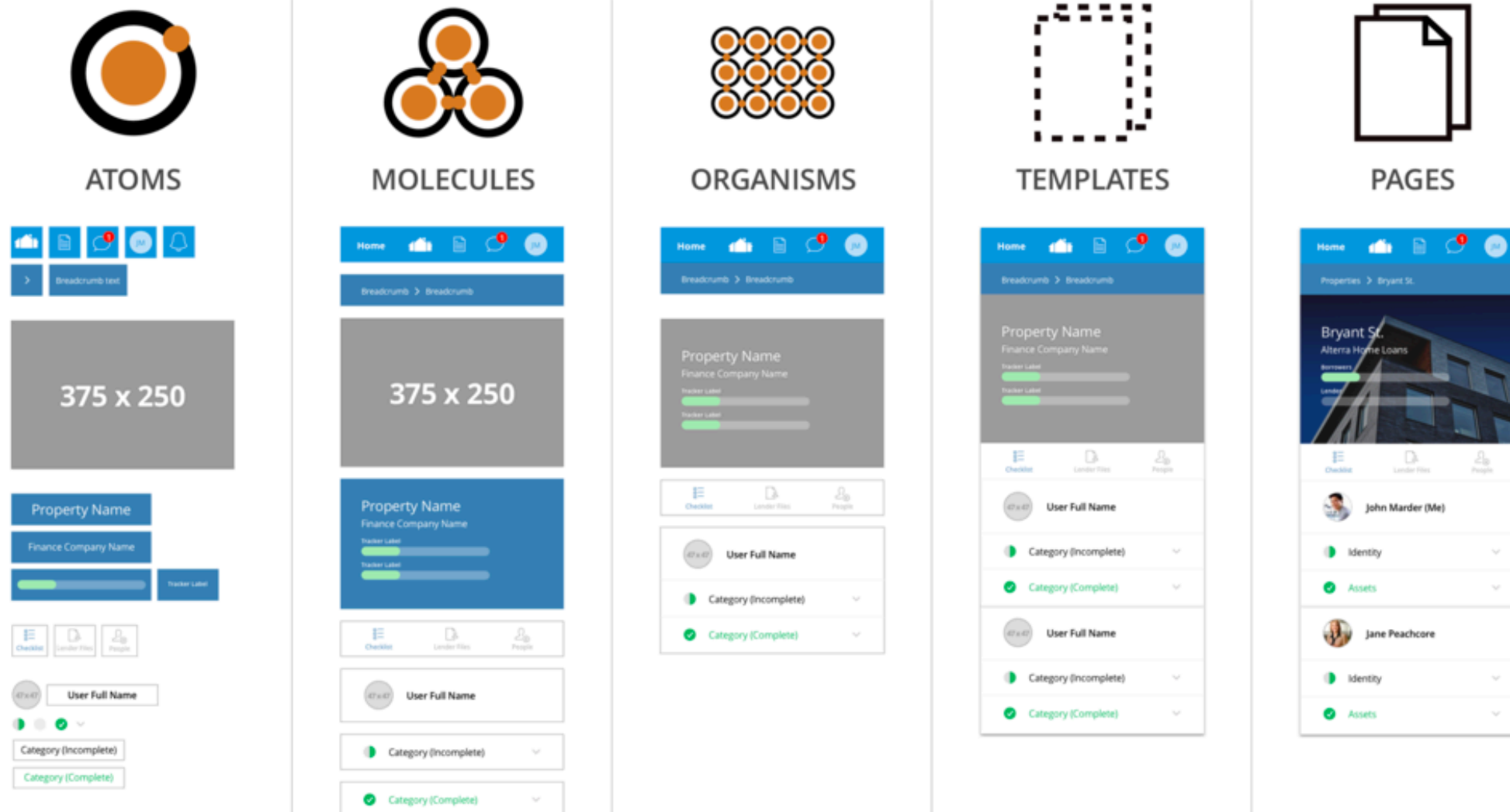
React intenta ayudar a los desarrolladores a construir aplicaciones que usan datos que cambian todo el tiempo. Su objetivo es ser sencillo, declarativo y fácil de combinar. React sólo maneja la interfaz de usuario en una aplicación; React es la Vista en un contexto en el que se use el patrón MVC (Modelo-Vista-Controlador) o MVVM (Modelo-vista-modelo de vista)



## ¿Como funciona?

Crea componentes encapsulados que manejen su propio estado, y los convierte en interfaces de usuario complejas.

Ya que la lógica de los componentes está escrita en JavaScript y no en plantillas, puedes pasar datos de forma sencilla a través de tu aplicación y mantener el estado fuera del DOM.

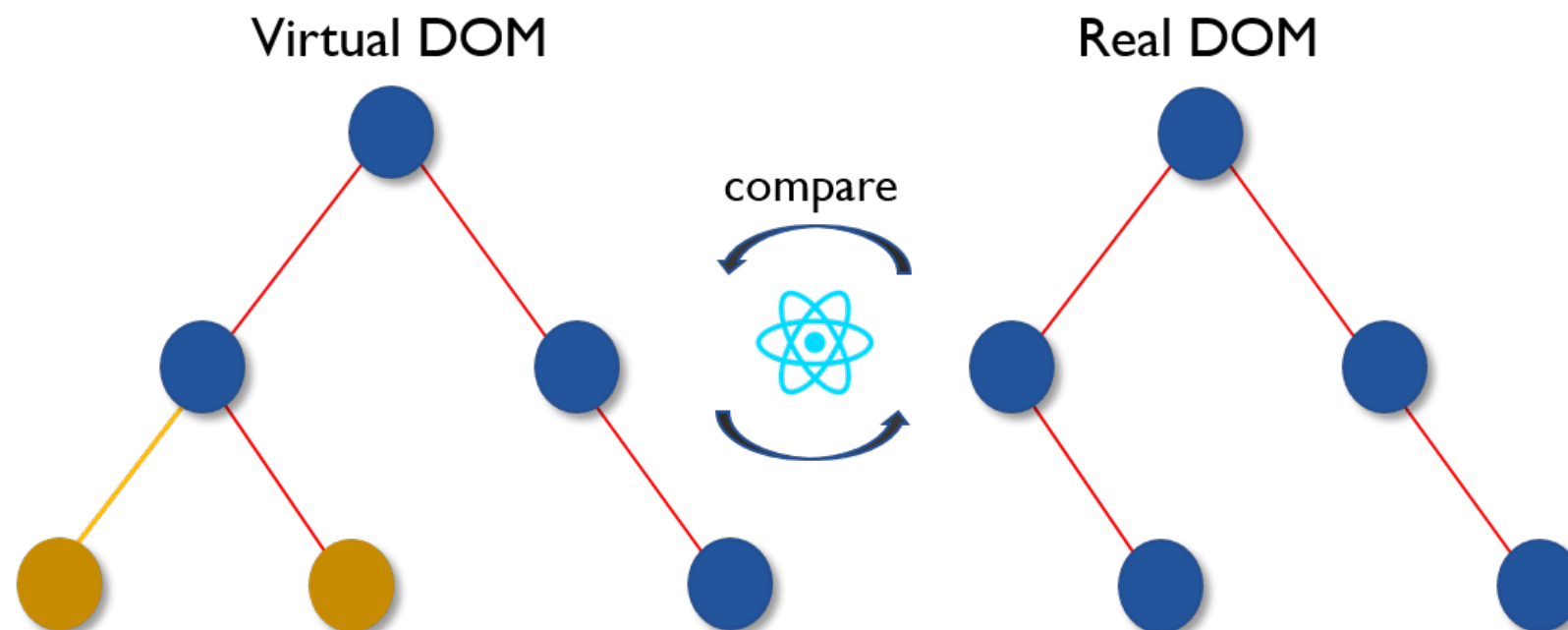


## Basado en Componentes

Crea componentes encapsulados que manejen su propio estado, y conviértelos en interfaces de usuario complejas. Ya que la lógica de los componentes está escrita en JavaScript y no en plantillas, puedes pasar datos de forma sencilla a través de tu aplicación y mantener el estado fuera del DOM.

## React mantiene un virtual DOM propio

En lugar de confiar solamente en el DOM del navegador. Esto deja a la biblioteca determinar qué partes del DOM han cambiado comparando contenidos entre la versión nueva y la almacenada en el virtual DOM, y utilizando el resultado para determinar cómo actualizar eficientemente el DOM del navegador



## create-react-app

La aplicación **create-react-app** es un entorno cómodo para aprender React , y es la mejor manera de comenzar a construir una nueva aplicación de una sola página en React.

Configura su entorno de desarrollo para que pueda usar las últimas funciones de JavaScript, proporciona una experiencia agradable para el desarrollador y optimiza su aplicación para la producción.

Necesitará tener NodeJS  $\geq 8.10$  y npm  $\geq 5.6$  en su máquina. Para crear un proyecto, ejecute:

```
npx create-react-app my-app  
cd my-app  
npm start
```

# Presentando JSX

Considera la declaración de esta variable:

```
const element = <h1>Hello, world!</h1>;
```

Esta curiosa sintaxis de etiquetas no es ni un **string** ni **HTML**.

Se llama JSX, y es una extensión de la sintaxis de JavaScript. Recomendamos usarlo con **React** para describir cómo debería ser la interfaz de usuario. JSX puede recordarte a un lenguaje de plantillas, pero viene con todo el poder de JavaScript.

**JSX** produce “**elementos**” de **React**

## ¿Por qué JSX?

React acepta el hecho de que la lógica de renderizado está intrínsecamente unida a la lógica de la interfaz de usuario: cómo se manejan los eventos, cómo cambia el estado con el tiempo y cómo se preparan los datos para su visualización.

En lugar de separar artificialmente tecnologías poniendo el maquetado y la lógica en archivos separados, React separa intereses con unidades ligeramente acopladas llamadas “**componentes**” que contienen ambas.

## Componente Stateless (Sin Estados) o Componente Presentacional



props are received from a parent component and are **read only**

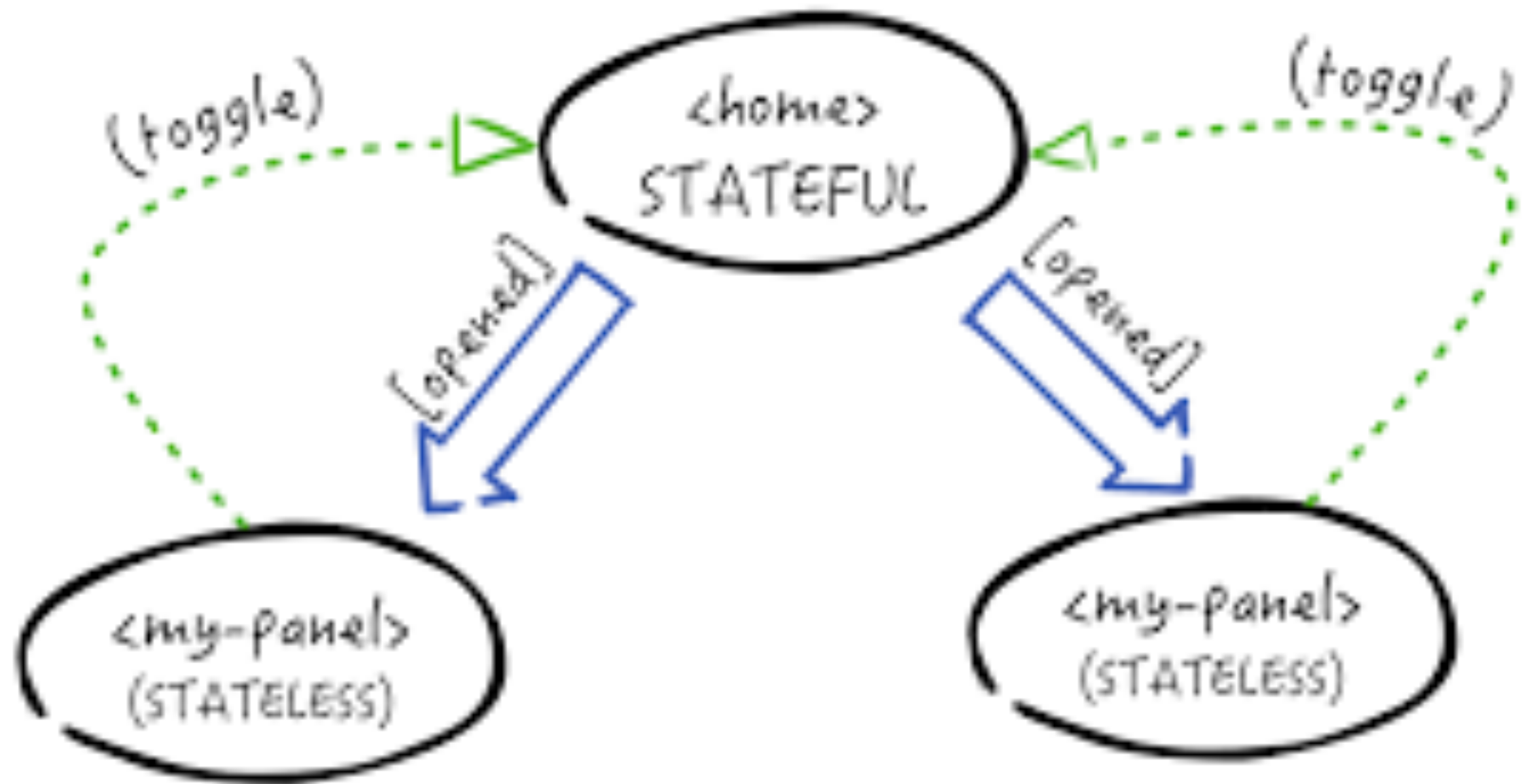


## Componente Stateless (Sin Estados) o Componente Presentacional

```
export const AgendaList = ({ list }) => (  
  <View style={styles.containerMain}>  
    <View style={styles.containerHeader}>  
      <Text style={styles.textRow}>ID</Text>  
      <Text style={styles.textRow}>Nombre</Text>  
      <Text style={styles.textRow}>Direccion</Text>  
    </View>  
    <View style={styles.containerBody}>  
      {list.map(c => (  
        <View key={c.id} style={styles.containerRow}>  
          <Text style={styles.textRow}>{c.id}</Text>  
          <Text style={styles.textRow}>{c.nombre}</Text>  
          <Text style={styles.textRow}>{c.direccion}</Text>  
        </View>  
      ) )}  
    </View>  
  </View>  
);
```

```
export default memo(AgendaList);
```

## Componente Statefull (conectado) o Contenedor



## Componente Statefull (conectado) o Contenedor

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { connect } from 'react-redux';
import { View } from 'react-native';

import { setUser, authenticateUser } from '../actions/loginActions';
import LoginForm from '../components/LoginForm';
import ButtonSubmit from '../components/ButtonSubmit';
import { Wallpaper, Logo } from '../../CommonUI';
```



**Boilerplate**

# Componente Stateful (conectado) o Contenedor

```
class LoginScreen extends Component {

  static navigationOptions = { header: false }

  _authenticateUser = () => {
    const { username, password, authenticateUser } = this.props;
    authenticateUser({ username, password });
  }

  render() {
    const { username, password, setUser, homeLoaded, loginError } = this.props;
    const { containerStyle, sectionLogo, sectionForm, sectionButton } = styles;
    return (
      <Wallpaper>
        <View style={containerStyle}>
          <View style={sectionLogo}>
            <Logo />
          </View>
          <View style={sectionForm}>
            <LoginForm
              username={username}
              password={password}
              setField={setUser}
            />
          </View>
          { /* <SignupSection /> */ }
          <View style={sectionButton}>
            <ButtonSubmit onLoginPress={() => this._authenticateUser()} loginError={loginError} homeLoaded={homeLoaded} />
          </View>
        </View>
      </Wallpaper>
    );
  }
}
```

# Componente Stateful (conectado) o Contenedor

```
LoginScreen.propTypes = {  
  username: PropTypes.string,  
  password: PropTypes.string,  
  setUser: PropTypes.func,  
  onLogin: PropTypes.func,  
  homeLoaded: PropTypes.bool,  
  authenticateUser: PropTypes.func,  
  loginError: PropTypes.any  
};
```

```
> const styles = { ...  
};
```

```
const mapStateToProps = (state) => {  
  return {  
    username: state.login.username,  
    password: state.login.password,  
    loginError: state.loginError.error,  
    isLoggedIn: state.auth.isLoggedIn,  
    homeLoaded: state.homeLoading.homeLoaded  
  };  
};
```

```
const mapDispatchToProps = dispatch => ({  
  setUser: (payload) => dispatch(setUser(payload)),  
  authenticateUser: (payload) => dispatch(authenticateUser(payload)),  
});
```

```
export default connect(mapStateToProps, mapDispatchToProps)(LoginScreen);
```

**REDUX**

## Props vs States

En un componente React, las **props** son valores pasados por su componente padre y son de solo lectura (o cambiados desde el padre).

El estado, por otro lado, sigue siendo variable, pero el componente lo inicializa y administra directamente. El estado puede ser inicializado por **props**.

Por ejemplo, un componente primario puede incluir un componente secundario:

**<ComponenteHijo>**

El padre puede pasar un accesorio usando esta sintaxis:

**<ChildComponent color=green />**

Las **props** se pueden usar para establecer el estado interno en función de un valor de en el constructor:

```
class ChildComponent extends React.Component {  
  constructor(props) {  
    super(props)  
    this.state.colorName = props.color  
  }  
}
```

```
class QueryBeneficiary extends Component {

  constructor(props) {
    super(props);
    this.state = {
      // estadoSelected: 0,
      mobilestate: true,
      dateStart: null,
      dateEnd: null,
      filterDirty: false,
      dateStartFiltered: null,
      dateEndFiltered: null,
      idConvenioFiltered: null,
      empresaSelectedFiltered: null,
      reset: false,
      showTable: false
    };
    this.dropDownRefPaymentType = React.createRef();
    this.dropDownRefAgreement = React.createRef();
    this.dropDownRefEnterprise = React.createRef();
    this.handlerOneEnterprise = this.handlerOneEnterprise.bind(this);
    this.handlerOnChangesSelected = this.handlerOnChangesSelected.bind(this);
    this.handlerOnChange = this.handlerOnChange.bind(this);
    this.handleOnChangePickerStart = this.handleOnChangePickerStart.bind(this);
    this.handleOnChangePickerEnd = this.handleOnChangePickerEnd.bind(this);
    this.handleChangeEstado = this.handleChangeEstado.bind(this);
    this.visibility = this.visibility.bind(this);
    this.handlerOnChangeFilter = this.handlerOnChangeFilter.bind(this);
    this.onClickDownload = this.onClickDownload.bind(this);
    this.setDateDefault = this.setDateDefault.bind(this);
    this.isDisabled = this.isDisabled.bind(this);
    this.handlerOnClick = this.handlerOnClick.bind(this);
    this.showFilterTable = this.showFilterTable.bind(this);
    this.onClickSelectPaysheet = this.onClickSelectPaysheet.bind(this);
    this.setStateBack = this.setStateBack.bind(this);
  }
}
```



## Estilos y CSS

### Como importar un archivo externo de css?

```
import './QueryBeneficiary.css';
```

### Como agrego una clase css al componente?

Pasa una string como la prop className:

```
render() {  
  return <span className="menu navigation-menu">Menu</span>  
}
```

Es común que las clases CSS dependan de las props o del estado del componente:

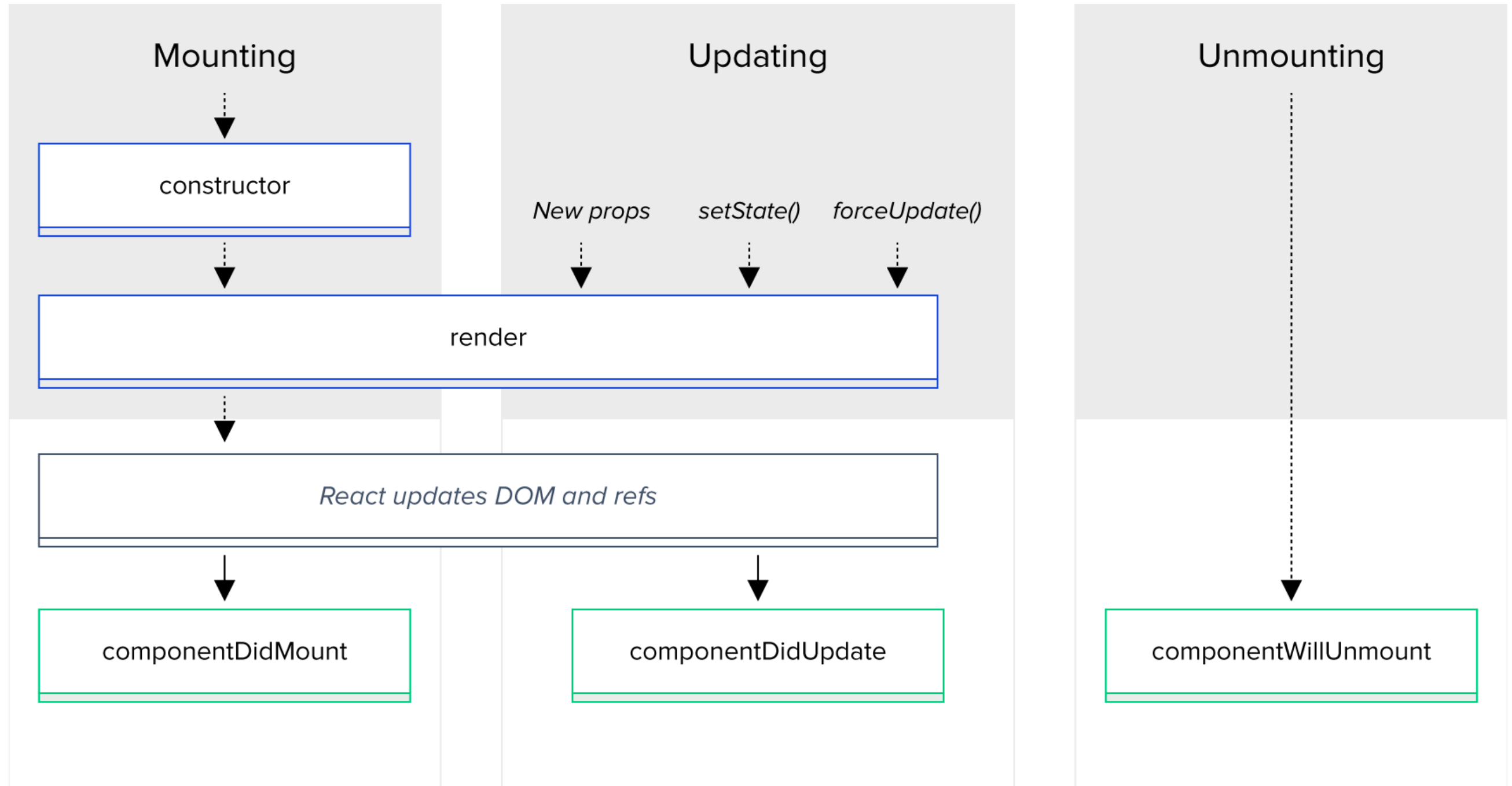
```
render() {  
  let className = 'menu';  
  if (this.props.isActive) {  
    className += ' menu-active';  
  }  
  return <span className={className}>Menu</span>  
}
```

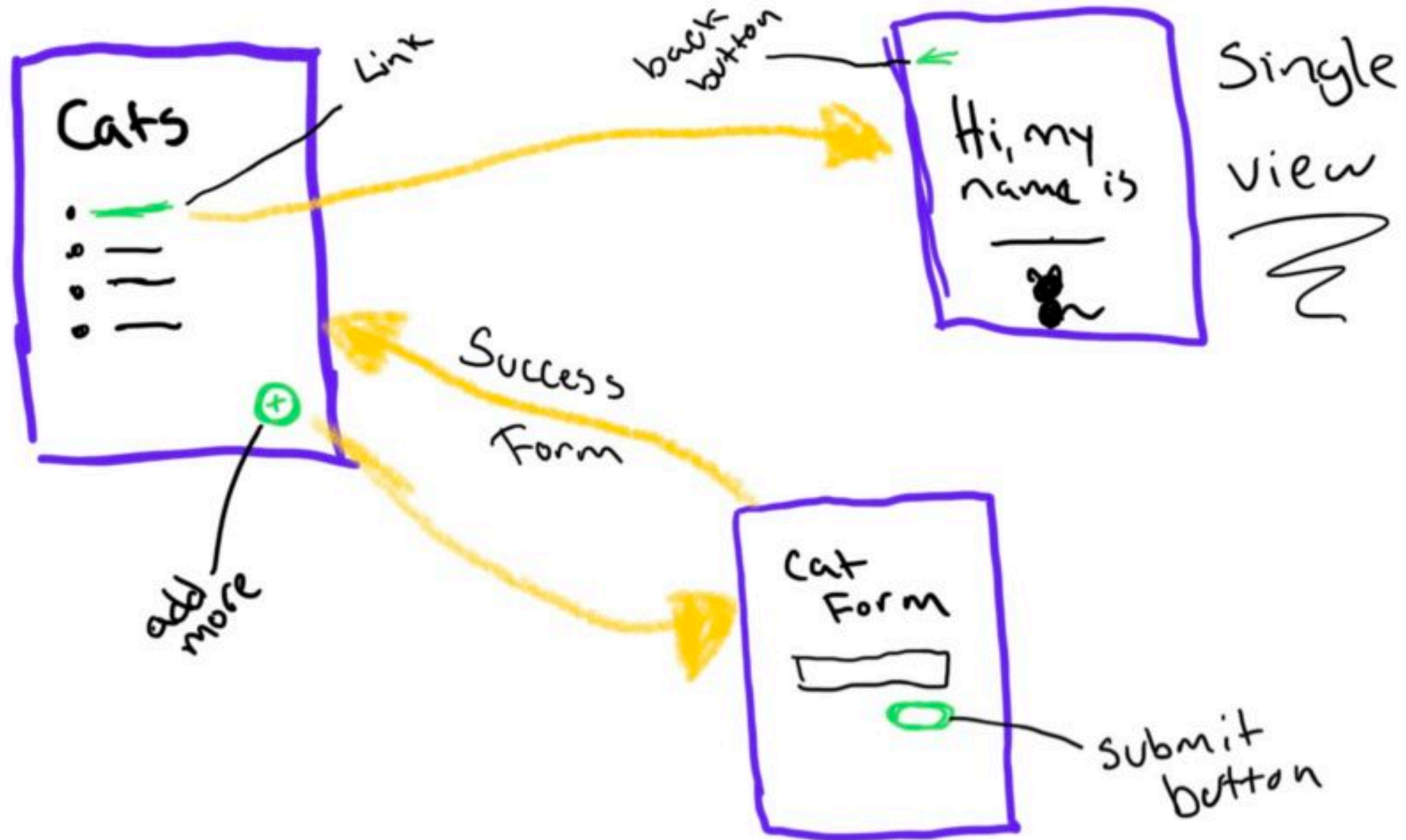
### Se puede usar CSS en Linea?

Sí! Pero no es lo recomendable y se pasa como si fuera un objeto literal de javascript  
Ejemplo : **style={ color: 'red', border: "1px solid black" }**



# Ciclo de vida de los componentes de React





## ¿ Que es React Router ?

Nos permite crear rutas entre nuestros componentes, y así poder navegar entre ellos como si fueran simples links en paginas web

### Instalación

```
npm install react-router-dom  
# o usando yarn  
yarn add react-router-dom
```

```
import React from "react"  
import { render } from "react-dom"  
import { HashRouter, Switch, Route } from "react-router-dom"  
import Home from "./Home"  
import BookList from "./BookList"  
import BookDetail from "./BookDetail"  
  
const App = () => (  
  <HashRouter> {/* envolvemos nuestra aplicación en el Router */}  
    <Switch> {/* también la envolvemos en el componente Switch */}  
      <Route path="/" component={Home} exact /> {/* y creamos nuestras rutas */}  
      <Route path="/books" component={BookList} exact />  
      <Route path="/books/:bookId" component={BookDetail} exact />  
    </Switch>  
  </HashRouter>  
)  
  
render(<App />, document.getElementById("root"))
```



# Introducción a **Redux**

Ponente: **Juan Canepa**

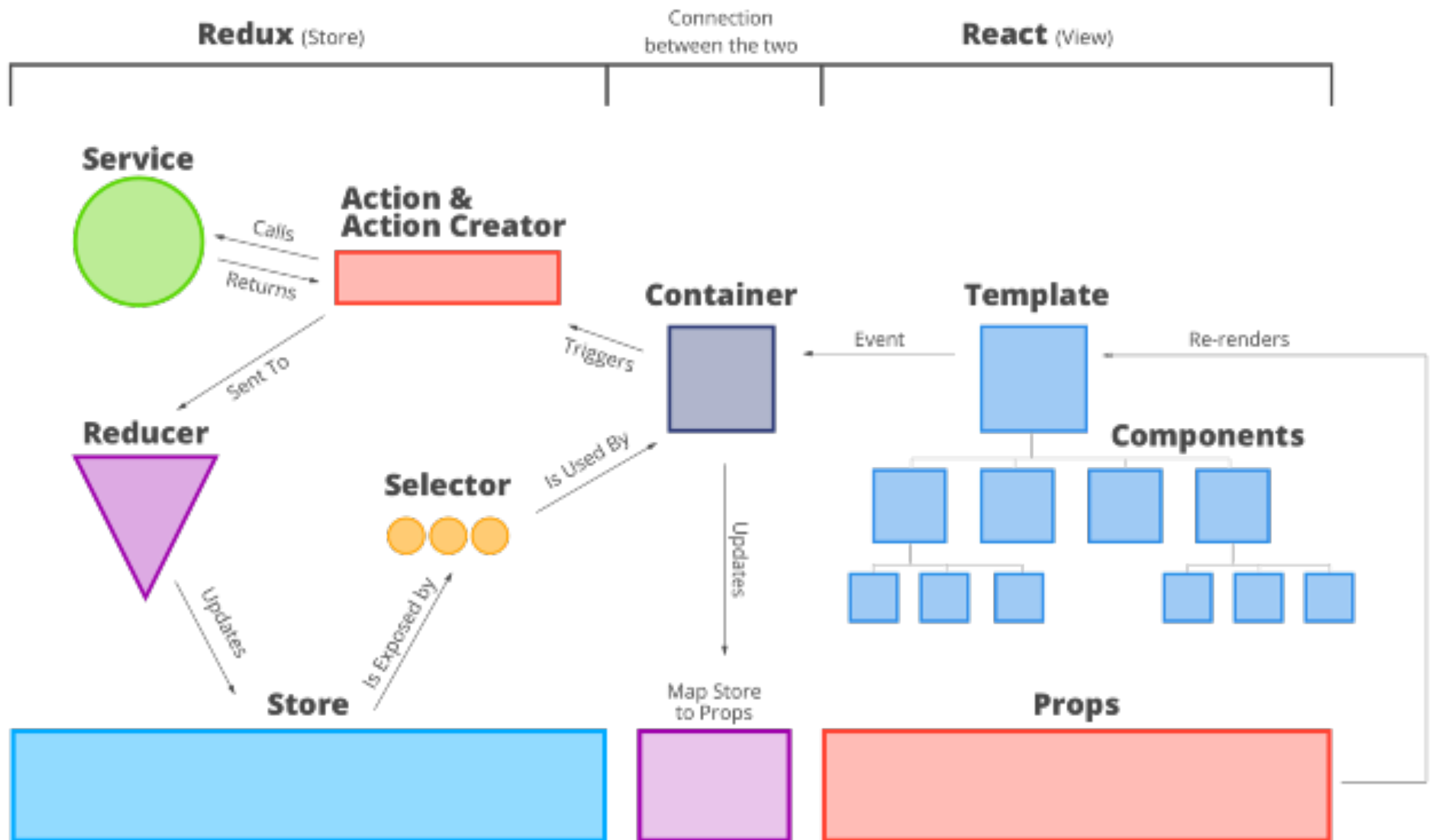
## Que es Redux

**Redux =**



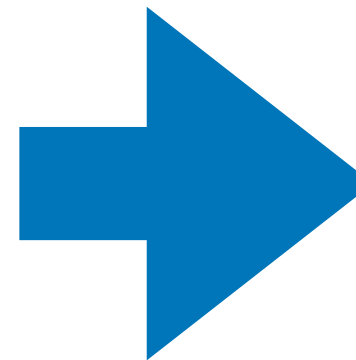
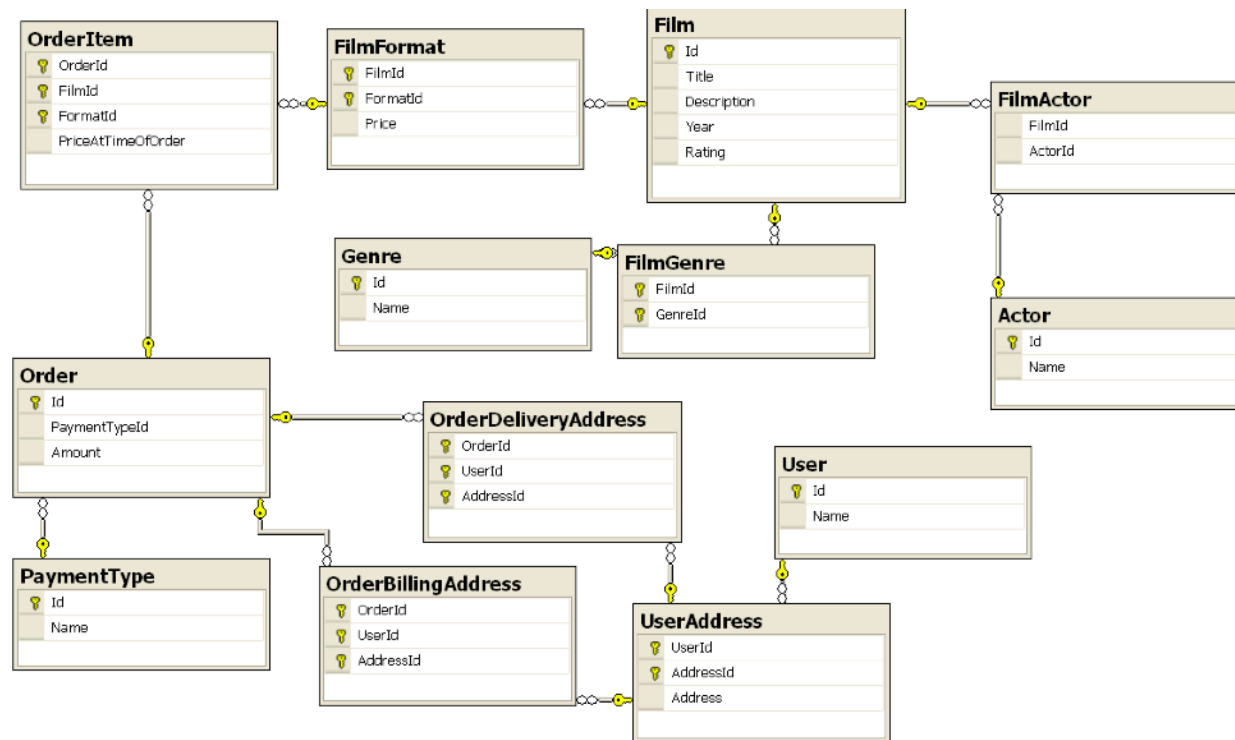
**Fuente de la verdad**

# Arquitectura de un proyecto con Redux

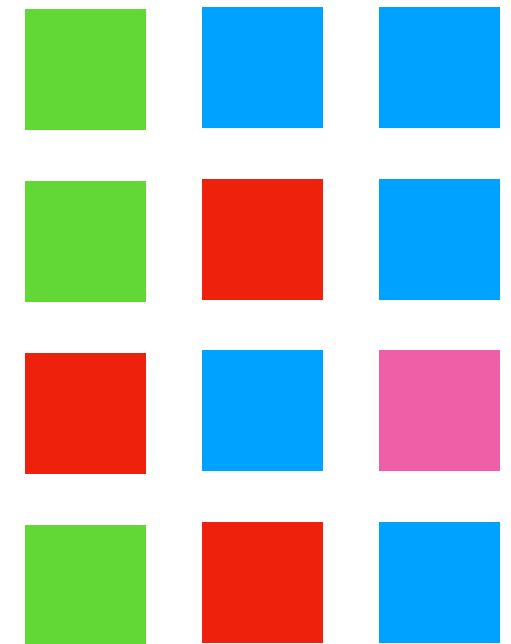


# Como funciona Redux

## Base de datos



## Reducers



**Cache de datos de la app = Funciones reductoras**

```
// types se crea un archivo ejemplo: constants/loginConstants.js
```

```
export const SET_USER = 'SET_USER';
```

```
export const BLANK = 'BLANK';
```

```
// action creators => se crea un archivo ejemplo actions/loginActions.js
```

```
import * as types from '../constants/loginConstants';
```

```
const setUser = payload => {
```

```
  return {
```

```
    type: types.SET_USER,
```

```
    payload
```

```
  };
```

```
};
```

```
// action creators => se crea un archivo ejemplo reducers/loginReducers.js
```

```
const loginReducers = (state = INITIAL_STATE_USER, action) => {
```

```
  switch (action.type) {
```

```
    case types.SET_USER:
```

```
      return { ...state, ...action.payload };
```

```
    case types.BLANK:
```

```
      return INITIAL_STATE_USER;
```

```
    default:
```

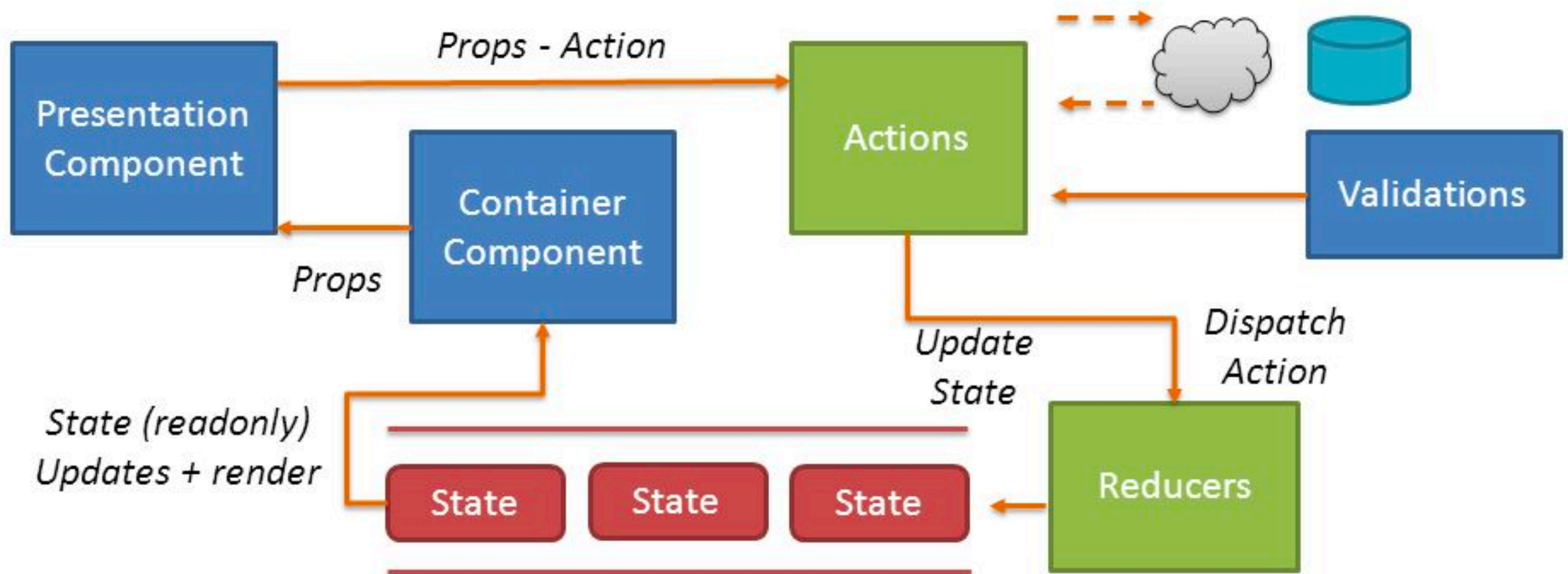
```
      return state;
```

```
  }
```

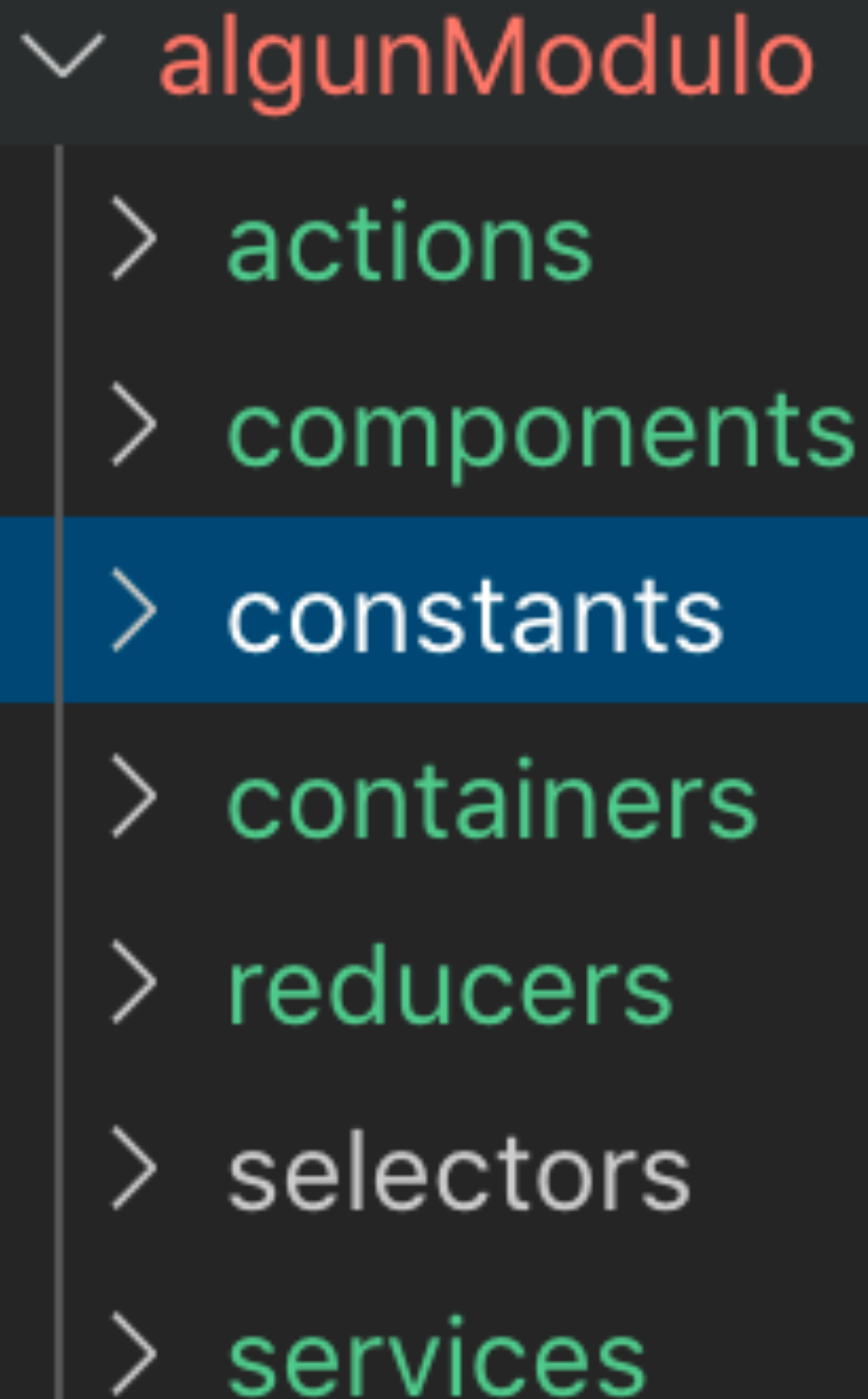
```
};
```



# Esquema Redux



## Estructura de un modulo con Redux



```

└─ algunModulo
   ├── actions
   ├── components
   ├── constants
   ├── containers
   ├── reducers
   ├── selectors
   └── services

```

- Actions: carpeta donde están los action creators
- Components: solo componentes presentaciones
- Constants: por lo general es donde están los types
- container: los componentes con conexión a redux
- Reducers: donde están las funciones reductoras
- Selectors: donde están los selectores creados con reselect
- Services: para separar la logica que contenga promesas o async / await

```

import * as types from './const';

const summaryReducers = (state = [], action) => {
  switch (action.type) {
    case types.FETCH_SUMMARY:
      return action.payload;
    case types.CLEAR_SUMMARY:
      return {};
    case types.BLANK:
      return [];
    default: return state;
  }
};

const queryPaysheetFilter = (state = {}, action) => {
  switch (action.type) {
    case 'SET_QUERY_FILTERS':
      return action.payload;
    case 'CLEAR_QUERY_FILTERS':
      return {};
    default: return state;
  }
};

const summaryHeadReducers = (state = [], action) => {
  switch (action.type) {
    case types.VALIDATION_PAYROLL:
      return action.payload;
    case types.HEADER_DETAIL_PAYROLL:
      return action.payload;
    case types.CLEAR_SUMMARY_HEAD:
      return {};
    case types.BLANK:
      return [];
    default: return state;
  }
};

const previewPaysheetReducers = (state = [], action) => {
  switch (action.type) {
    case types.PREVIEW_PAYSHEET:
      return action.payload.list;
    case types.BLANK:
      return [];
    default: return state;
  }
};

```



## Cómo es un buen diseño?

- Separar cada estado en una función reductora con su opción predecible, recordemos un poco la documentación de redux:

**Redux es un contenedor predecible del estado de aplicaciones JavaScript.**

### Ventajas de este enfoque:

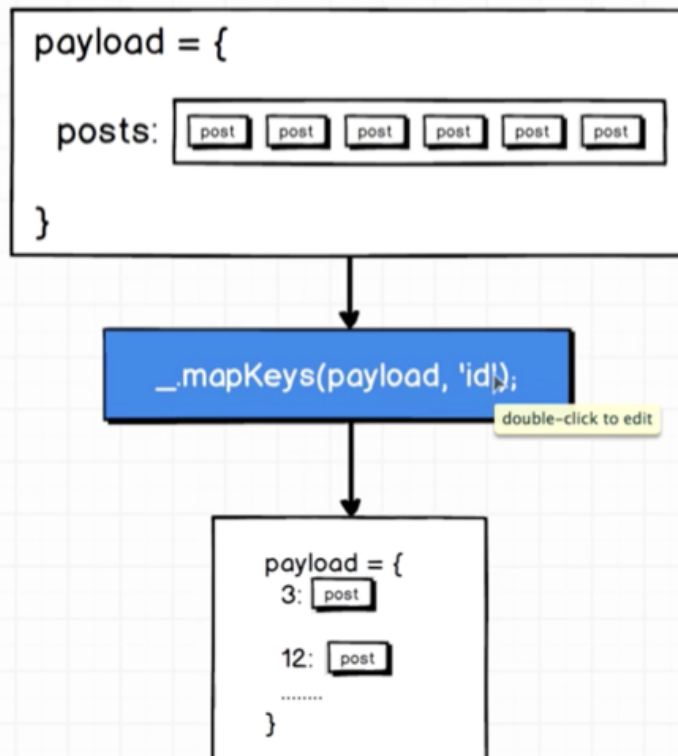
- Aplicaciones mas testeables, pruebas unitarias menos complicadas y creamos mas independencia entre cada estado.
- Mejora el rendimiento notablemente, es mas fácil determinar los cambios de estados.
- Se pueden utilizar técnicas mas avanzadas como es el uso de calculo de datos derivados (conmumente llamados selectores).
- Creamos reducers mas reutilizables.
- Nos olvidamos de destructuring complicados o mejor dicho rebuscados.
- Estructuras de datos mucho mas legibles.

# Cómo almacenar los datos en Redux?



```
const initialState = [  
  {  
    id: 1,  
    name: 'John Smith',  
    address: 'santiago',  
    group: {  
      id: 1,  
      type: 'A',  
      description: 'Customer with remuneration > 3.000.000'  
    }  
  },  
  {  
    id: 2,  
    name: 'Louis Nath',  
    address: 'santiago',  
    group: {  
      id: 1,  
      type: 'B',  
      description: 'Customer with remuneration < 1.000.000'  
    }  
  }  
];
```

# Cómo almacenar los datos en Redux?



```
const initialState = {
  1: {
    id: 1,
    name: 'John Smith',
    address: 'santiago',
    group: {
      id: 1,
      type: 'A',
      description: 'Customer with remuneration > 3.000.000'
    }
  },
  2: {
    id: 2,
    name: 'Louis Nath',
    address: 'santiago',
    group: {
      id: 1,
      type: 'B',
      description: 'Customer with remuneration < 1.000.000'
    }
  }
};
```



# Cómo crear el Storage de Redux?

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

import navReducers from '../navigators/navReducers';
import loginReducers from '../modules/Login/reducers/loginReducers';
import homeReducers from '../modules/Home/reducers/homeReducers';
import footer from '../modules/CommonUI/FooterSura/reducers/footerReducers';
import newsList from '../modules/News/reducers/newsReducers';

export default combineReducers({
  ...navReducers,
  ...loginReducers,
  ...homeReducers,
  ...footer,
  ...newsList
});
```

# Cómo configurar el Storage ?

```
import React, { Component } from 'react';
import { Root } from 'native-base';
import { applyMiddleware, createStore } from 'redux';
import { Provider } from 'react-redux';
import createSagaMiddleware from 'redux-saga';

import reducers from './reducers';
import AppWithNavigationState from './navigators/AppNavigator';
const sagaMiddleware = createSagaMiddleware();
const middlewares = [sagaMiddleware];

import rootSaga from './sagas';
const store = createStore(reducers, {}, applyMiddleware(...middlewares));
sagaMiddleware.run(rootSaga);

export default class App extends Component {
  render() {
    return (
      <Root>
        <Provider store={store}>
          <AppWithNavigationState />
        </Provider>
      </Root>
    );
  }
}
```

# Cómo usar Redux ?



- Validación de formularios,
- Mostrar/ocultar un modal,
- Datos para una UI en particular que no es reutilizable,
- Para componentes UI Kit.



- Cache de datos (ejemplo cuando se hace un fetch de clientes desde un servicio Rest)
- Datos centralizados donde se necesite un estado centralizado, ejemplo los valores de un carrito de compra que aparecerán en diferentes vistas.



# Cómo usar States ?



- Cache de datos (ejemplo cuando se hace un fetch de clientes desde un servicio Rest)
- Datos centralizados donde se necesite un estado centralizado, ejemplo los valores de un carrito de compra que aparecerán en diferentes vistas.



- Validación de formularios,
- Mostrar/ocultar un modal,
- Datos para una UI en particular que no es reutilizable,
- Para componentes UI Kit.