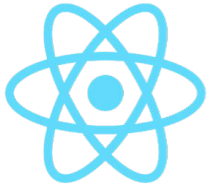
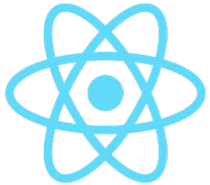
DESARROLLO FRONT-END CON REACT.JS

Diego Gil - FLISOL 2018 - Salta - Argentina

ÍNDICE

¿QUÉ ES  React ?

- Es una librería, en js, para crear interfaces de usuario.
- Una librería... No un framework
- Es la “V” en “MVC”
- Declarativo vs Imperativo
- Basado en componentes
- https://github.com/facebook/react/wiki/Sites-Using-React

¿QUÉ ES  React ?

Imperativo

```
$('form').on('submit', function(e) {
    e.preventDefault();
    $.ajax({
        url: '/customers',
        type: 'POST',
        data: $(this).serialize(),
        success: function(data) {
            $('.status')
                .append('<h3>' + data + '</h3>');
        }
    });
});
```

Declarativo

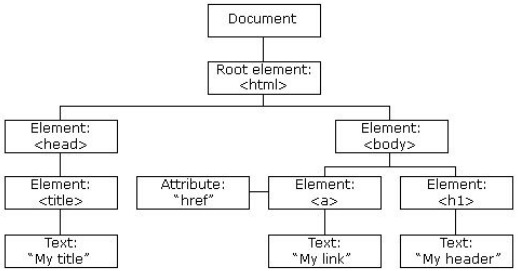
```
class NoteBox extends React.Component {
    // ... more code ...
    render() {
        return (
            <div className="NoteBox">
                <h1>Notes</h1>
                <NoteList data={this.state.data} />
                <NoteForm onPost={this.handlePost} />
            </div>
        );
    }
};
```

- React nos abstrae del DOM
- Nos hace pensar en *estados* de la UI en lugar de manipulaciones de la misma
- Permite actualizar toda la app en cada actualización*
- Simplifica la forma en que fluyen los datos (one way)

(*) Frase válida hasta ~~agotar stock~~ explicar Virtual DOM

Dom virtual y JSX

- DOM === Document Object Model === Objeto que crea el navegador para representar un documento (en nuestro caso, HTML)
- DOM Virtual es una versión simplificada del DOM, creado y gestionado por React
  - Permite a React saber qué se modificó.
  - Mantiene el DOM intacto hasta que se decida hacer cambios.



- JSX es una forma cómoda de crear elementos en el DOM Virtual.

```
const MyComponent = (props) => (
    React.createElement("div", null, "Hello World")
);

ReactDOM.render(
    React.createElement(MyComponent),
    document.getElementById("root")
);
```

```
const MyComponent = (props) => (
    <div>Hello World!</div>
);

ReactDOM.render(
    <MyComponent />,
    document.getElementById("root")
);
```

<Componentes />

- Componentes === Máquinas de estado
  - React interpreta el UI como máquinas de estado simples
  - Cada estado significa renderizar la UI de alguna forma
  - React actualiza el DOM de forma eficiente en base al estado
- Componentes === Funciones
  - Un componente recibe datos y retorna UI
  - Entonces podemos decir: UI = F(estado)

<Componentes />

Componentes de Clase

```
import React from "react";

// Modern classes use ES6 class syntax
class ClassSyntax extends React.Component {
    render() {
        return (
            <div> Hello World!</div>
        );
    }
}
```

Componentes Funcionales

```
import React from "react";

// The "functional component" syntax lets you use a function
// as a component. It's effectively just the render method
function FunctionalComponentSyntax(props) {
    return (
        <div>Hello World!</div>
    );
}

// Or, use ES6 arrow function syntax to declare the component
const ArrowFunctionComponent = (props) => (
    <div>Hello World!</div>
);
```

<Componentes />

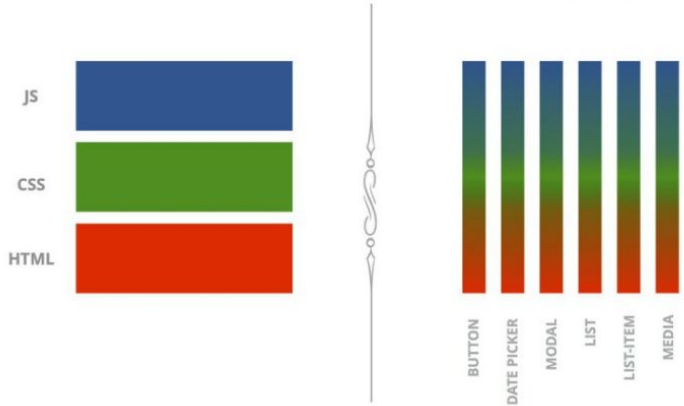
Uso básico

```
// Note use of className for HTML classes
const MyComponent = (props) => {
    const someString = "Something else to
display";

    return (
        <div className="class1 class2">
            Hello World!
            {/* <div>Commented out code</div>
*/}
            <div>{someString}</div>
            <div>{28 + 14}</div>
        </div>
    );
}


// Illegal!
const BrokenComponent = () => (
    <div />
    <div />
);
```

“Hay un HTML en mi ~~sopa~~ JS”



Datos entre Componentes: Props

- Se combinan como un solo objeto
  - Class components: this.props
  - Func components: Único Parámetro
- Son read-only: Declarativo!
- Se pueden validar (PropTypes = Datos consistentes)
  - array, func, object
  - bool, number, string
  - node, element
  - any
  - instanceOf(Componente)
  - oneOf(['Componente1', 'Componente2'])
  - oneOfType([...])
  - arrayOf(number)
  - objectOf(number)
  - shape({})
  - customProp: function(props, name, _) {}

```
class HelloWorld extends React.Component {
    render() {
        return (
            <div>Hello {this.props.name}!</div>
        );
    }
}

render(
    <HelloWorld name="Mark"/>
);
```

Gestión del estado

- Cada componente puede tener un estado interno
- Componentes funcionales no tienen estado
- Se puede acceder al mismo con *this.state*
- No se debe modificar directamente
- Cada cambio al estado ocasiona el re-render del componente

```
class Counter extends React.Component {
    state = {counter : 0}

    onClick = () => {
        this.setState({counter : this.state.counter + 1});
    }

    render() {
        const {counter} = this.state;

        return (
            <div>
                Button was clicked:
                <div>{counter} times</div>

                <button onClick={this.onClick} >
                    Click Me
                </button>
            </div>
        );
    }
}
render(<Counter />);
```

Componentes: Ciclo de vida

“Los componentes se montan, actualizan, y desmontan dependiendo de nuestros datos.”

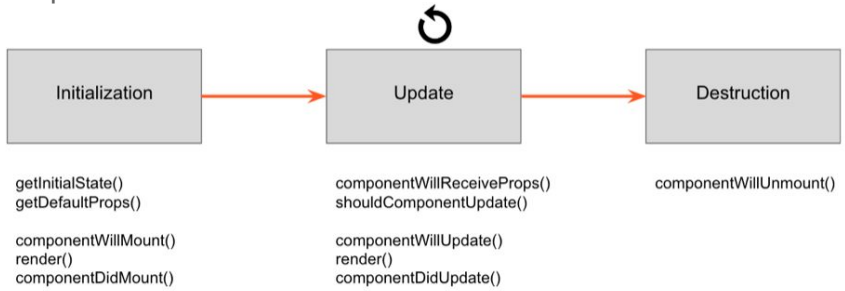
- *componentDidMount*: Se llama una vez, luego del primer renderizado. Se tiene acceso al DOM, y es un buen lugar para obtener datos de afuera (hacer llamadas AJAX)
- *shouldComponentUpdate*: Se llama antes de re-renderizar, se tiene acceso al estado y props anteriores, si decidimos que no debería actualizar basta retornar *false*.
- *componentDidUpdate*: Se llama después de re-renderizar, se tiene acceso a los nuevos elementos del DOM.
- *componentWillUnmount*: Se llama inmediatamente después que el componente se desmonta del DOM. Buen lugar para eliminar listeners, mensajes, etc.

Componentes: Ciclo de vida

- *componentWillMount*: Se llama una vez, antes de correr el render inicial.
  - Desde React 16.3 no se debe usar (getDerivedStateFromProps)

- *componentWillReceiveProps*: Se llama cuando el componente está por recibir nuevas props, da acceso a las props anteriores y nuevas. Aquí se puede definir state (si quisiéramos hacerlo en base a props)
  - Desde React 16.3 no se debe usar (getDerivedStateFromProps)

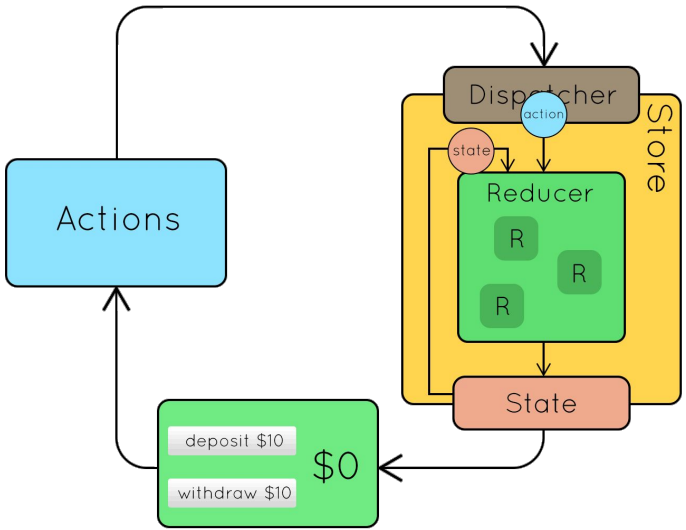
- *static getDerivedStateFromProps*: Se llama cada vez que las props se actualizan, se tiene acceso a props anteriores, si retornamos un objeto, el mismo se va a mergear al estado del componente



Estado de la aplicación con Redux

"Redux es un contenedor predecible y centralizado para el estado de la aplicación."

- Todo el estado de la aplicación está contenido en una sola *store*
- El estado es read-only. Los cambios al mismo se realizan por funciones puras (*reducer*)
  - *(state, action) => newState*
- *State* (estado): El estado de la aplicación se guarda en un objeto
- *Actions*: Para cambiar algo del state, necesitamos lanzar una acción. Por consistencia, se utilizan *action creators* para crear acciones.
- *Reducers*: Son funciones que reciben el estado actual y una acción, y en base a ello devuelven el nuevo estado.
  - Deben ser funciones puras
  - Deben mantener la información inmutable



Estado de la aplicación con Redux

State y actions

```
// App state: a plain object with many keys or "slices"
{
    todos: [{
        text: "Eat food",
        completed: true
    }, {
        text: "Exercise",
        completed: false
    }],
    visibilityFilter : "SHOW_COMPLETED"
}

// Actions: plain objects with a "type" field
{ type: "ADD_TODO", text: "Go to swimming pool" }
{ type: "TOGGLE_TODO", index: 1 }
{ type: "SET_VISIBILITY_FILTER", filter: "SHOW_ALL" }


// Action creators: functions that return an action
function addTodo(text) {
    return {
        type : "ADD_TODO",
        text
    };
}
```

Reducer

```
function todosReducer(state = [], action) {
    switch (action.type) {
        case "ADD_TODO":
            return state.concat([{
                text: action.text, completed: false
            }]);
        case "TOGGLE_TODO":
            return state.map((todo, index) => {
                if(index !== action.index) return todo;
                return { text: todo.text, completed: !todo.completed }
            })
        default: return state;
    }
}
```

Otras librerías y links útiles

- Styled Components https://github.com/styled-components/styled-components
- Thunks https://github.com/gaearon/redux-thunk
- Inmutable.js https://github.com/facebook/immutable-js/
- Reselect https://github.com/reactjs/reselect
- Flux https://facebook.github.io/flux/docs/overview.html#content

TERMINAMOS..

¿PREGUNTAS?

https://github.com/dieguit/flisol2018