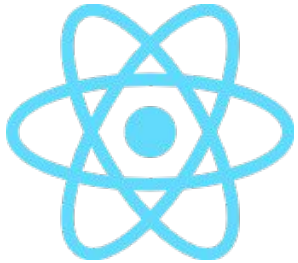
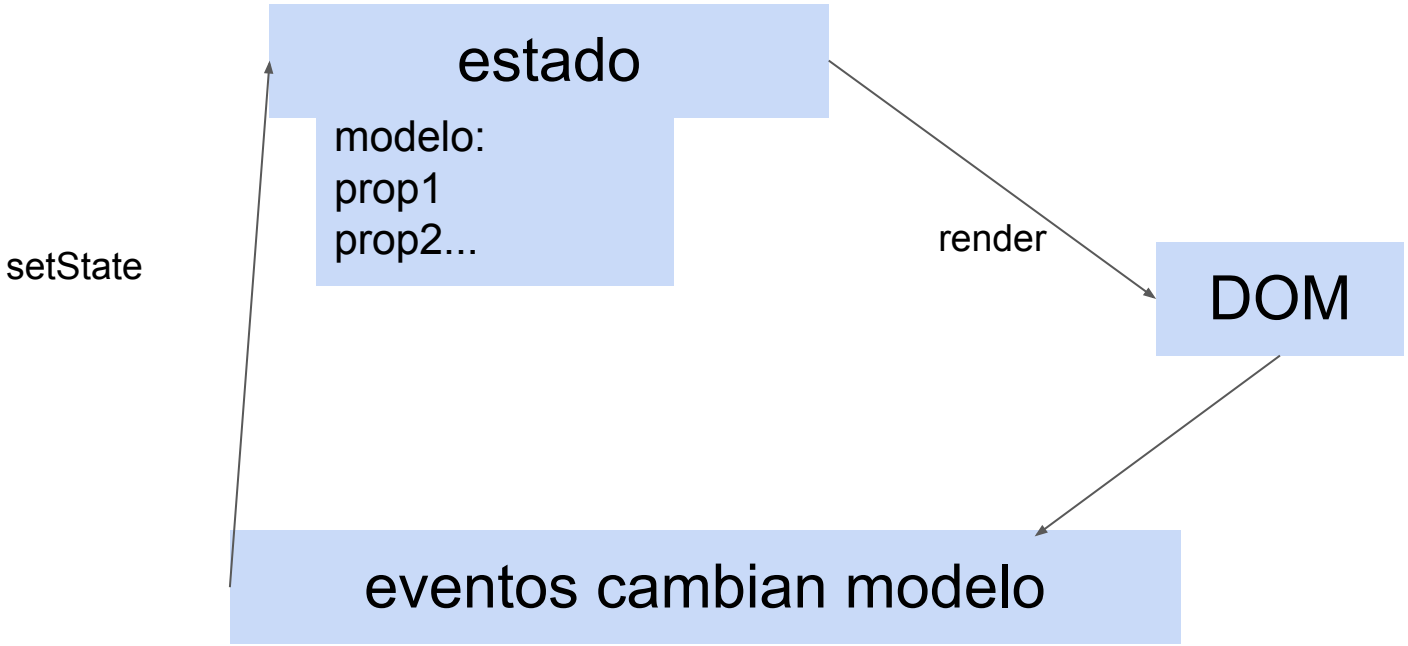
React



React. Introducción

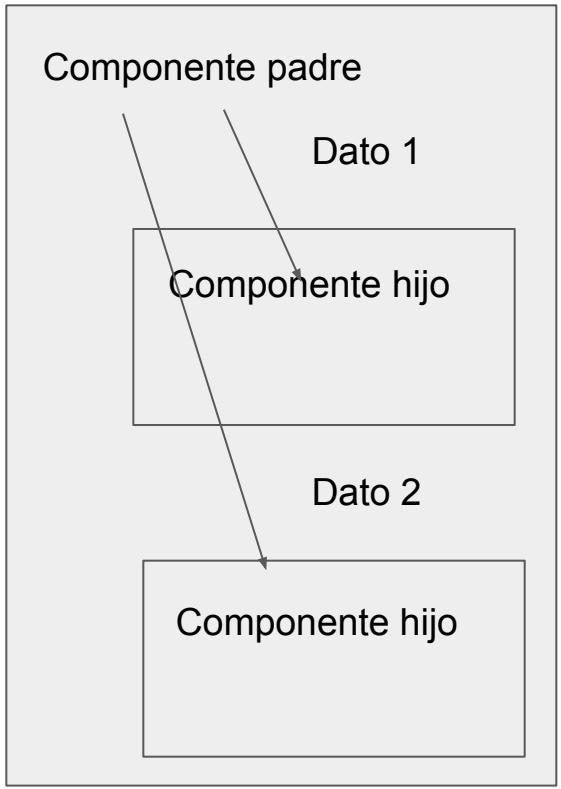
- Creado por Facebook
- Orientado a simplificar la creación de interfaces
- Compatible con cualquier backend
- Funciones que leen actualizaciones de estado
- Realiza cambios sólo en la parte afectada, lo que aumenta el rendimiento
- Virtual-DOM
- Flujo: modelo-estado-DOM

React. Introducción

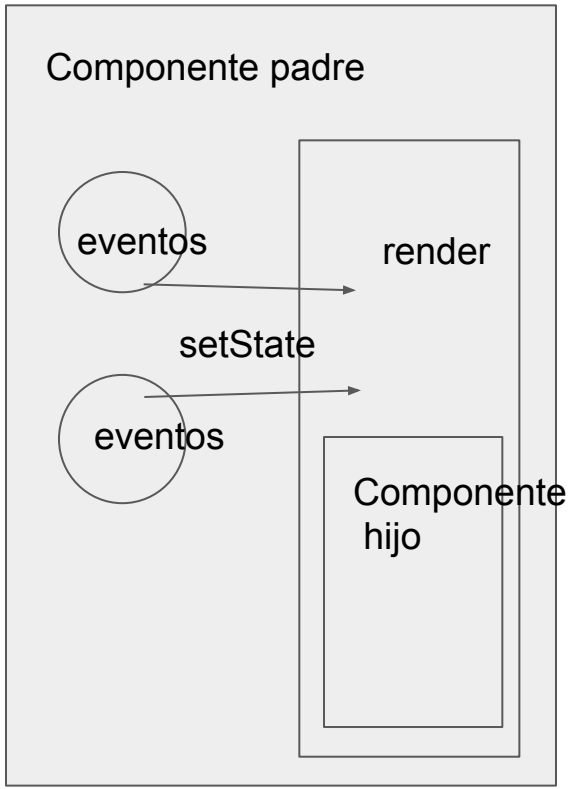


React. Estados y props

Props



State



React. React vs Angular

React

- jsx
- flujo lineal
- componentes
- core simplificado
- errores simples en la compilación
- axios o fetch
- Patrón reactive

Angular

- plantillas
- databinding bidireccional
- directivas, pipes, providers, etc
- core complejo
- errores complejos de encontrar
- rxjs
- Patrón interactive

React. Jsx

- HTML/XML + js compilado con babel
- Se puede compilar en cliente o en tiempo de desarrollo
- Virtual DOM rendering
- Evita repaint y reflow
- API limitada: setState, setProps, forceUpdate, render
- MVC + MVI (Model view Intent)
- Reactive: El modelo observa cuando hay un cambio mediante event emitters que cambian el model y activan los cambios en el render

React. MVI

Modelo

- input: eventos de interacción enviados desde el intent
- output: eventos de datos

View

- input: eventos de datos recibidos desde el model
- output: renderizado del virtual DOM

Intent

- input: eventos de usuario enviados desde el view
- output: eventos de interacción que pasan los datos al model

React. Bloques en cliente

```
<script src="https://unpkg.com/react@17/umd/react.production.min.js" crossorigin></script>

<script src="https://unpkg.com/react-dom@17/umd/react-dom.production.min.js" crossorigin></script>


<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.24/browser.js"></script>
```

React. Bloques en cliente

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}
const user = {
  firstName: 'Harper',
  lastName: 'Perez',
};
const element = <h1>Hello, {formatName(user)}!</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

React. Componentes

```
ReactDOM.render(
        <NumberList numbers={numbers}/>,
        document.getElementById('react-list')
        );
```

React. Componentes basados en funciones

```
const NumberList = (props) => {
  const numbers = props.numbers;
  const listItems = numbers.map((number) => {
    return  <li key={number}>{number}</li>
  });
  return (
    <ul>{listItems}</ul>
  );
}
```

React. Componentes basados en clases

```
class NumberList extends React.Component {
        render() {
            const listItems = numbers.map((number) => {
                return  <li key={number}>{number}</li>
            });
            return (
                <ul>{listItems}</ul>
            )
        }
}
```

React. Babel y utilidades

- npm install -g babel-cli
- npm install babel-preset-react

babel --presets react src --watch --out-dir dist

- npm install react-bootstrap --save
- npm install bootstrap@3 --save

React. Create-react-app

- npm i -g create-react-app
- npx create-react-app my-app

React. Component vs Element

- Elemento: un objeto del DOM
    - <button className=’button’>haz click</button>
- Componente: una clase o función que devuelve un elemento compuesto o no y que puede contener métodos internos
    - function Componente() {
        const clicked = () => {
            console.log(‘Se ha hecho click’);
        }
        return (

                <button className=’button’ onClick={clicked}>haz click</button>

)
    }

React. Propiedades y estados

- Propiedades
  - inmutables
  - comunican componentes
  - comunicación de arriba a abajo
- Estados
  - mutables
  - internos al componente
  - generan el renderizado de la parte que depende de ellos

```
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

React. Eventos

```
<li onClick={()=>{this.changeAmpliado()}} >
<li onClick={this.changeAmpliado} >

<li onClick={this.changeAmpliado.bind(this)} >
this.changeAmpliado = this.changeAmpliado.bind(this);
```

onTouchStart onTouchMove onTouchEnd onTouchCancel
onClick onDoubleClick onMouseDown onMouseUp onMouseOver
onMouseMove onMouseEnter onMouseLeave onMouseOut onContextMenu
onDrag onDragEnter onDragLeave onDragExit onDragStart
onDragEnd onDragOver onDrop
onKeyDown onKeyUp onKeyPress

React. Condicionales

```
let contenido_ampliado;
        if(this.state.ampliado){
            contenido_ampliado = (<div><h2>{this.props.data.edad}</h2></div>);
        }
        return (
            <li className="item" onClick={this.changeAmpliado.bind(this)} >
                {this.props.nombre} x {this.props.children}
                {contenido_ampliado}
            </li>
        );
```

React. Condicionales

```
return (
        <li className="item" onClick={this.changeAmpliado.bind(this)} >
            {this.props.nombre} x {this.props.children}
            {
                this.state.ampliado &&
                    <div><h2>{this.props.data.edad}</h2></div>
            }
        </li>
    );
```

React. Condicionales

```
return (
        <div className="list">
            <List data={this.elementos}></List>
            {
                this.state.showWarning ?
                    <Warning clickHandler={this.toggleWarning}></Warning>
                    :
                    <button onClick={this.toggleWarning}>Abrir alerta</button>
            }
        </div>
    )
```

React. Comunicación padre-hijo

- padre -> hijo mediante props
- hijo -> padre mediante handler pasado como prop

en padre:

<Warning clickHandler={this.toggleWarning}></Warning>

en hijo

<button onClick={this.props.clickHandler}>Aceptar</button>

React. Children

- Sirve para incluir contenidos desde el padre

en padre:

<Container><h1>Mi título</h1></Container>

en hijo

{this.props.children}

React. Livecicle (clases)

constructor: antes del renderizado

componentDidMount: después del primer renderizado

componentDidUpdate(prevProps, prevState, snapshot): cuando detecta un cambio de estado

componentWillunmount: antes de eliminar el componente

React. Hooks

- Sólo en funciones
- Permiten trabajar con estados en las funciones
- Usan su propio livecicle
  - useState equivalente a constructor
  - useEffect equivalente a componentWillMount o componentDidMount
  - useContext
  - userReducer
  - ...

React. Hooks y estado

```
const [fruit, setFruit] = useState('banana');
```

React. Hooks y datos

```
useEffect(() => {
    getData();
}, [prop1, prop2, ...])
```

React. Forms

- controlled
  - valores relacionados con el estado
- uncontrolled
  - this.asuntoInput = React.createRef();
  - ref={this.asuntoInput}

React. Validación de propiedades en componentes

- PropTypes
- Se pueden crear funciones propias de validación

```
import PropTypes from 'prop-types';

Componente.propTypes = {
    errorText: validacionCustom,
    touched: PropTypes.bool.isRequired,
}
```

React. Propiedades por defecto en componentes

```
Componente.defaultProps={
    errorText: "Error desconocido",
    touched: true,
}
```

React. Sass

- npm install node-sass --save
- import ‘./Container.scss’;

React. React router

- npm i --save react-router-dom

```
import {
    BrowserRouter as Router,
        Switch,
        Route,
        Link
} from "react-router-dom";
```

React. React router

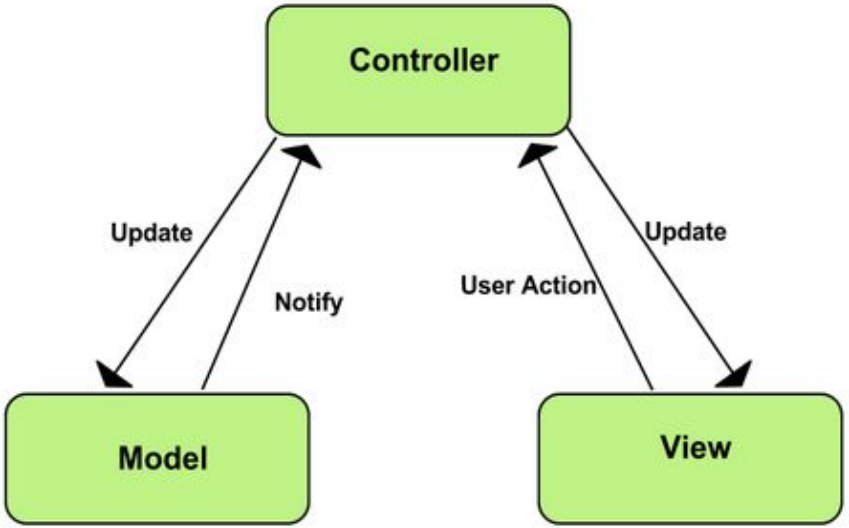
- Router
  - Link
    - to
  - Switch
    - Route
      - path
      - exact
      - component

React. React router

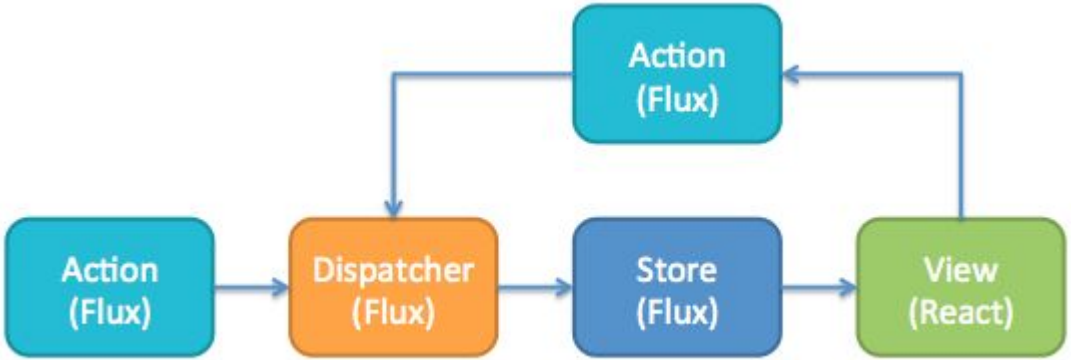
- Cada router puede tener sus propias rutas
- Se pueden añadir varios routers
- withRouter añade como propiedades location y history para tener acceso a parámetros de ruta y navegar desde js

React. Flux

- Organiza el flujo de datos en las aplicaciones



React. Flux


Action
(Flux)
Action
(Flux)
Dispatcher
(Flux)
Store
(Flux)
View
(React)

React. Flux

- Flujo de datos unidireccional
- Estado guardado en almacén
- Vista: los componentes ya creados
- Store: guarda el estado de la app
- Acción: objeto JS que indica un cambio que ha de realizarse
- Dispatcher: se ocupa de propagar las acciones al Store. al desacoplar el store de la vista, no hace falta saber qué store maneja la acción

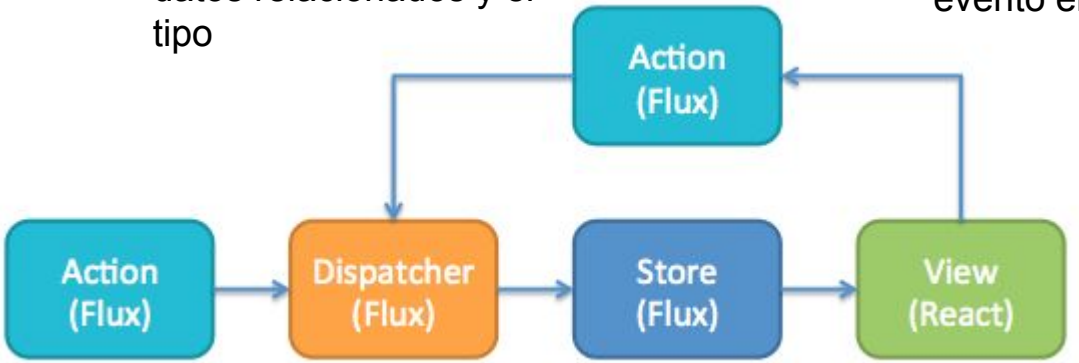
React. Flux

- Store: almacena los valores. Se suscribe a los dispatchers, hace modificaciones y lanza eventos
- Acciones: lanza desde el dispatcher la información hasta el store
- Dispatcher: se ocupa de la transmisión de la información
- Vista: en el componente se suscribe a los eventos

React. Flux

Esta acción contiene los datos relacionados y el tipo

La vista, mediante un evento envía una acción



El dispatcher propaga la acción al store y se procesa en orden de llegada

El store recibe la acción, hace los cambios pertinentes de estado según el tipo y propaga el cambio a la vista

La vista recibe la actualización y aplica los cambios

React. Flux (implementaciones)

- Redux
- Reflux
- Delorean
- Fluxor
- ...

React. Redux

- Utiliza reducers como funciones que se ocupan de modificar el estado a partir de las acciones
- Un solo store y varios reducers que guardan el estado inicial
- Los reducers “deciden” los cambios en el estado propio y lo devuelven para aplicarlo como estado interno al componente

npm install --save redux react-redux redux-thunk

React tests. Unit test con Mocha y Chai

- npm install mocha chai chai-enzyme --save-dev
- npm i --save-dev enzyme enzyme-adapter-react-16 cheerio
- npm install @babel/core @babel/register --save-dev
- npm install babel-preset-react-app-babel-7 --save-dev
- npm install @babel/plugin-transform-modules-commonjs --save-dev
- npm install ignore-styles --save-dev

React tests. Unit test con Mocha y Chai

Package.json scripts

- "test:unit": "NODE_ENV=test mocha --require @babel/register --require ignore-styles src/test/*.test.js",
- "test:watch": "npm run test:unit -- --watch"

React tests. Unit test con Mocha y Chai

Package.json scripts

- "test:unit": "NODE_ENV=test mocha --require @babel/register --require ignore-styles src/test/*.test.js",
- "test:watch": "npm run test:unit -- --watch"

```
npm install --save-optional win-node-env
```

React tests. Unit test con Mocha y Chai

babel.config.js

```
module.exports = (api) => {
    const presets = ["react-app"];
    const plugins = [
        "@babel/plugin-transform-modules-commonjs",
    ];
    api.cache(false);
    return {
        presets,
        plugins
    };
};
```

React tests. Mocha

- describe: descripción de tarea. Son anidables
- it: ejecución de un test concreto
- before, beforeEach: ejecución previa a uno o varios describes. Prepara el test
- after, afterEach: Limpia al terminar para asegurar que todo queda limpio para el siguiente test
- only, skip

React tests. Chai

- expect: define qué se espera que ocurra
- to
- be
- have
- not
- equal
- ...

React tests. Chai

- expect: define qué se espera que ocurra
- to
- be
- have
- not
- equal
- ...

React tests. React. Enzime

- shallow: renderiza sin navegador
- mount, unmount: para montar o eliminar componentes
- act: para esperar por operaciones asíncronas
- find, first: para recuperar elementos
- dive: para hacer búsquedas dentro de wrappers

React tests. Redux

- npm install --save-dev redux-mock-store