

React

Introducción

- React es una **librería**
- Para crear **interfaces de usuario**
- Componentes
- Filosofía funcional

Requisitos

- Necesitamos:
 - webpack
 - babel
 - babel-loader
 - babel-preset-es2015
 - babel-preset-react
 - react
 - react-dom

Requisitos

```
$ npm install -S react react-dom babel-preset-react
```

Requisitos

```
{  
  module: {  
    loaders: [{  
      test: /\.js$/,  
      exclude: /node_modules/,  
      loader: 'babel-loader',  
      query: {  
        presets: ['react', 'es2015'] // <---  
      }  
    }]  
  }  
}
```

Hola, Mundo

```
import React from 'react'
import ReactDOM from 'react-dom'

class HolaMundo extends React.Component {
  render() {
    return <h1>Hola, Mundo!</h1>
  }
}

window.onload = () => ReactDOM.render(
  React.createElement(HolaMundo),
  document.getElementById('app')
)
```

Hola, Mundo

```
$ npm run build
```



```
<html>
  <Meta charset="utf-8" />
  <head>
    <title>React Experiments</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="bundle.js"></script>
  </body>
</html>
```

```
import React from 'react'
import ReactDOM from 'react-dom'

class HolaMundo extends React.Component {
  render() {
    return <h1>Hola, Mundo!</h1>
  }
}
```

```
window.onload = () => ReactDOM.render(
  React.createElement(HolaMundo),
  document.getElementById('app')
)
```

```
import React from 'react'
import ReactDOM from 'react-dom'

class HolaMundo extends React.Component {
  render() {
    return <h1>Hola, Mundo!</h1>
  }
}
```

```
window.onload = () => ReactDOM.render(
  React.createElement(HolaMundo),
  document.getElementById('app')
)
```

```
_createClass(HolaMundo, [{  
  key: 'render',  
  value: function render() {  
    return _react2.default.createElement(  
      'h1',  
      null,  
      'Hola, Mundo!'  
    );  
  }  
}]);
```

JSX

- JSX es un dialecto XML
- Facilitar la construcción de componentes
- Babel lo traduce a Javascript
- Lo podemos utilizar en cualquier expresión

```
const header = (<h1>Hello, World!</h1>)
```

```
const list = [  
  <li>Uno</li>,  
  <li>Dos</li>,  
  <li>Tres</li>  
]
```

```
class HolaMundo extends React.Component {  
  render() {  
    const m = 'Mundo'  
    return <h1>Hola, {m}!</h1>  
  }  
}
```

```
class HolaMundo extends React.Component {  
  render() {  
    const m = 'Mundo'  
    return <h1>Hola, {m} </h1>  
  }  
}
```


Ejercicio

- Escribe, desde 0...
 - Un componente Reloj
 - Que muestre horas, minutos y segundos
 - Utiliza HolaMundo como referencia

```
const HolaMundo = () => {  
  return <h1>Hola, Mundo</h1>  
}
```

```
const HolaMundo = () => <h1>Hola, Mundo!</h1>
```

Organización del Código

```
const Mundo = () => <span>Mundo</span>
```

```
class HolaMundo extends React.Component {  
  render() {  
    return <h1>Hola, <Mundo/>!</h1>  
  }  
}
```

```
const Mundo = () => <span>Mundo</span>
```

```
class HolaMundo extends React.Component {  
  render() {  
    return <h1>Hola, <Mundo/> </h1>  
  }  
}
```

Ejercicio

- Divide Reloj en 5 componentes
 - **App:** el componente de nivel superior
 - **Reloj:** que se compone de Horas, Minutos y Segundos
 - **Horas/Minutos/Segundos:** cada uno muestra el valor correspondiente

Organización de Código

- Cada componente en su propio fichero
- Utilizar **import/export**
- Múltiples filosofías para organizar los ficheros...
- Nosotros vamos a crear una carpeta **src/components**

Ejercicio

- Reorganiza el Reloj
 - **src/index.js**: importa y monta el componente App
 - **src/app.js**: Importa Reloj y exporta App
 - **src/components/reloj.js**: importa Horas, Minutos y Segundos y exporta Reloj
 - **src/components/horas.js**: exporta Horas
 - ...

Props

```
class HolaCosa extends React.Component {  
  render() {  
    return (<h1> Hola, {this.props.cosa}!</h1>)  
  }  
}
```

```
class App extends React.Component {  
  render() {  
    return <HolaCosa cosa="Mundo" />  
  }  
}
```

```
class HolaCosa extends React.Component {  
  render() {  
    return (<h1> Hola, {this.props.cosa} </h1>)  
  }  
}
```

```
class App extends React.Component {  
  render() {  
    return <HolaCosa cosa="Mundo" />  
  }  
}
```

```
const HolaCosa = (props) => <h1>Hola, {props.cosa}!</h1>
```

```
class App extends React.Component {  
  render() {  
    return <HolaCosa cosa="Mundo" />  
  }  
}
```

```
const HolaCosa = (props) => <h1>Hola, {props.cosa}!</h1>
```

```
class App extends React.Component {  
  render() {  
    return <HolaCosa cosa="Mundo" />  
  }  
}
```

Ejercicio

- Modifica el Reloj
 - Para que Horas, Minutos y Segundos reciban el valor que tienen que mostrar como **prop**

Ejercicio

- Modifica el Reloj
 - Reemplaza Horas, Minutos y Segundos por un solo componente: Segmento
 - Segmento recibe el valor que tiene que mostrar como **prop**


```
class RedBox extends React.Component {  
  render() {  
    return <div style={{background: 'red'}}>  
      {this.props.children}  
    </div>  
  }  
}
```

```
class App extends React.Component {  
  render() {  
    return <RedBox> <h1>Cuidado!!</h1> </RedBox>  
  }  
}
```

```
class RedBox extends React.Component {  
  render() {  
    return <div style={{background: 'red'}}>  
      {this.props.children}  
    </div>  
  }  
}
```

```
class App extends React.Component {  
  render() {  
    return <RedBox> <h1>Cuidado!!</h1> </RedBox>  
  }  
}
```

JSX vs. HTML

Diferencias entre JSX y HTML

- **className** en vez de **class**
- **htmlFor** en vez de **for**
- **<input/>** en vez de **<input>**
- **style** recibe un objeto, no un string
- los eventos se escriben en camelCase
 - **onChange**, **onClick**, etc...

Ejercicio

- Dale estilos al Reloj
 - Crea una hoja de estilos e inclúyela en el HTML
 - Dale la clase “segmento” a Segmento
 - Escribe algunos estilos para la clase “segmento”

Ejercicio

- Partiendo de **react/003**
 - Toma como referencia **dist/clock.html**
 - Traduce el reloj a componentes
 - Escribe la lógica para que las agujas se coloquen correctamente
 - Utiliza **style** y **rotate** para rotar las agujas

Ejercicio

- Partiendo del **HTML** en **react/004**
 - Modela la tabla con **componentes**
 - **<Table>**, **<TableRow>**, y un componente para cada tipo de celda
 - Organiza el código correctamente

Ciclo de vida

Ciclo de vida

- Podemos definir funciones que se ejecutarán en diferentes momentos del ciclo de vida de nuestro componente
- Útiles para configurar el componente o para limpiar cuando se vaya a desmontar

Inicialización

- `constructor()`
- `componentWillMount()`
- `render()`
- `componentDidMount()`

Actualización

- `componentWillReceiveProps()`
- `shouldComponentUpdate()`
- `componentWillUpdate()`
- `render()`
- `componentDidUpdate()`

Destrucción

- `componentWillUnmount()`

Error

- `componentDidCatch()`

```
class MyComponent extends React.Component {  
  constructor() {  
    super()  
    console.log('constructor!')  
  }  
  componentDidMount() {  
    console.log('componente en la página')  
  }  
  componentWillUnmount() {  
    console.log('a punto de ser eliminado')  
  }  
}
```

Estado

Estado

- Una **propiedad especial** de los componentes
 - un objeto en el que podemos guardar lo que queramos
 - cada vez que modificamos el contenido, el **componente se re-rendea**


```
class MyComponent extends React.Component {  
  constructor() {  
    super()  
    this.state = { prop: 'value' }  
  }  
}
```

```
this.setState({ prop: 'value', prop2: 'value2' })
```

`this.state.prop`

Ejercicio

- Modifica el reloj para que se actualice cada segundo

Estado

- Cada vez que se **actualiza** el estado....
 - se **re-rendea el componente**
 - y **todos** sus hijos
 - pero **React** optimiza este proceso

Estado

- La **representación** de un componente debería depender **exclusivamente** de:
 - sus **props**
 - su **estado**

Ejercicio

- ¿Qué pasa si llamamos a **setState** desde **render**?

Eventos


```
class Button extends React.Component {  
  render() {  
    return <button onClick={() => alert('Hola!')}>  
      Saludar  
    </button>  
  }  
}
```

Eventos

- Buenas prácticas
 - centralizar el estado en componentes **inteligentes**
 - que **orquestan** componentes **presentacionales**
 - se comunican con sus hijos...
 - mediante props
 - y callbacks

```
class MyButton extends React.Component {  
  render() {  
    return <button onClick={this.props.action}>Click Me!</button>  
  }  
}
```

```
class App extends React.Component {  
  render() {  
    return <MyButton action={() => alert('Thank you...')} />  
  }  
}
```

Ejercicio

- Modifica el **cronómetro** añadiendo un boton **LAP**
 - guarda el tiempo actual
 - lo añade a una lista de tiempos guardados

```
class Lista extends React.Component {  
  render() {  
    const lista = [<li>Uno</li>, <li>Dos</li>];  
    return <ul> {lista} </ul>  
  }  
}
```

Eventos

- Al interpolar componentes desde una lista...
 - Hay que añadir a cada componente una propiedad **key**
 - Única para cada elemento
 - React lo necesita para distinguir cual es cual

```
class Lista extends React.Component {  
  render() {  
    const lista = [<li key={1}>Uno</li>, <li key={2}>Dos</li>];  
    return <ul> {lista} </ul>  
  }  
}
```

Ejercicio

- Partiendo de **react/006**, programa **Tooltip**
 - configurable mediante props
 - utiliza **onmouseenter** y **onmouseleave**
 - muestra y oculta la burbuja gestionando la clase **is-active**

Eventos

- El objeto **event** que reciben los handlers...
 - construido por **React**
 - pero similar al nativo
 - **preventDefault**, **stopPropagation**, etc...

Formularios

Formularios

- Los formularios en React...
 - funcionan de manera peculiar
 - conflicto entre el estado natural de los formularios y el control de React

```
class MyForm extends React.Component {  
  render() {  
    return <input type="text" value="Fijo"/>  
  }  
}
```

Formularios

- Tenemos dos opciones:
 - componentes **controlados**
 - gestionar a mano el valor de cada input
 - componentes **no controlados**
 - respetar el comportamiento nativo y recolectar los valores al final del proceso

Componentes controlados

```
class App extends React.Component {  
  constructor() {  
    super()  
    this.state = { counter: 0 }  
  }  
  inc() {  
    this.setState({ counter: this.state.counter + 1 })  
  }  
  render() {  
    return (  
      <div>  
        <input type="text" value={this.state.counter} />  
        <button onClick={this.inc.bind(this)}>Incrementar!</button>  
      </div>  
    )  
  }  
}
```

Componentes controlados

- Para crear un input editable...
 - escuchar al evento **onChange**
 - guardar **event.target.value** en el **estado**
 - usar ese valor como propiedad **value** del input


```
class App extends React.Component {  
  constructor() {  
    super()  
    this.state = { value: '' }  
    this.handleChange = this.handleChange.bind(this)  
  }  
  handleChange(e) {  
    this.setState({ value: e.target.value })  
  }  
  render() {  
    return <input type="text"  
      value={this.state.value}  
      onChange={this.handleChange} />  
  }  
}
```

Ejercicio

- Escribe un componente **NumericInput**
 - que sólo permita **escribir números**

```
class App extends React.Component {
  constructor() {
    super()
    this.state = { value: 'dos' }
    this.handleChange = this.handleChange.bind(this)
  }
  handleChange(e) {
    this.setState({ value: e.target.value })
  }
  render() {
    return <select value={this.state.value} onChange={this.handleChange}>
      <option value='uno'>1</option>
      <option value='dos'>2</option>
      <option value='tres'>3</option>
    </select>
  }
}
```

```
class App extends React.Component {  
  constructor() {  
    super()  
    this.state = { isChecked: '' }  
    this.handleChange = this.handleChange.bind(this)  
  }  
  handleChange(e) {  
    this.setState({ isChecked: e.target.checked })  
  }  
  render() {  
    return <form>  
      <label>  
        <input type="checkbox" checked={this.state.isChecked}  
          onChange={this.handleChange} />  
        Checked?  
      </label>  
    </form>  
  }  
}
```

Ejercicio

- ¿Como podríamos controlar **múltiples** checkboxes a la vez?
 - Escribe un componente que gestione 5 checkboxes

Ejercicio

- Partiendo de **react/007**
 - Escribe un aplicación **To Do**
 - Utilizando todo lo que hemos aprendido

Componentes no controlados

```
class App extends React.Component {  
  constructor() {  
    super()  
    this.handleSubmit = this.handleSubmit.bind(this)  
  }  
  handleSubmit(e) {  
    // extraemos el valor del input  
  }  
  render() {  
    return <form onSubmit={this.handleSubmit}>  
      <input type="text" />  
      <input type="submit" value="adelante!" />  
    </form>  
  }  
}
```


Componentes no controlados

- Para guardar una referencia a un elemento...
 - utilizamos la propiedad especial **ref**
 - recibe **una función**
 - la invoca **inmediatamente**
 - pasándole **una referencia al elemento** como primer parámetro

```
class App extends React.Component {  
  constructor() {  
    super()  
    this.handleSubmit = this.handleSubmit.bind(this)  
  }  
  handleSubmit(e) {  
    e.preventDefault()  
    alert(`Has escrito: ${this.input.value}`)  
  }  
  render() {  
    return <form onSubmit={this.handleSubmit}>  
      <input type="text" ref={(el) => this.input = el}/>  
      <input type="submit" value="adelante!" />  
    </form>  
  }  
}
```

webpack-dev-server

webpack-dev-server

- Vamos a mejorar nuestra manera de trabajar
 - **webpack-dev-server** compila nuestro código
 - pero además lo sirve en local
 - instalado en **react/008**
 - **npm run server**

Routing

Routing

```
$ npm install -S react-router-dom
```

Routing

- `react/008/index.js`

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



```
class Cara extends React.Component {  
  render() {  
    return <div>  
      <h1>Cara</h1>  
      <Link to="/cruz">ir a Cruz</Link>  
    </div>  
  }  
}
```

```
class Cruz extends React.Component {  
  render() {  
    return <div>  
      <h1>Cruz</h1>  
      <Link to="/cara">ir a Cara</Link>  
    </div>  
  }  
}
```

```
class Home extends React.Component {  
  render() {  
    return <div>  
      <h1>Elige:</h1>  
      <ul>  
        <li><Link to="/cara">cara</Link></li>  
        <li><Link to="/cruz">cruz</Link></li>  
      </ul>  
    </div>  
  }  
}
```

```
class App extends React.Component {  
  render() {  
    return <Router>  
      <Switch>  
        <Route exact path="/" component={Home} />  
        <Route path="/cara" component={Cara} />  
        <Route path="/cruz" component={Cruz} />  
      </Switch>  
    </Router>  
  }  
}
```

Ejercicio

- Partiendo de **react/009**
 - Traduce el HTML a componentes
 - Conecta la navegación con un Router
 - *consejo: intenta sacar componentes comunes*
 - ***Layout, Footer, etc...***

```
export default () => {  
  return <Router>  
    <Switch>  
      <Route exact path="/" component={Home} />  
      <Route path="/timer/:time" component={Timer} />  
    </Switch>  
  </Router>  
}
```

```
class Home extends React.Component {  
  render() {  
    return <div>  
      <h1>Home</h1>  
      <Link to='/timer/190'>Go to timer</Link>  
    </div>  
  }  
}
```

```
class Timer extends React.Component {  
  render() {  
    return <div>  
      <h1>Timer: {this.props.match.params.time}</h1>  
      <Link to="/">Go to Home</Link>  
    </div>  
  }  
}
```

```
class Timer extends React.Component {  
  render() {  
    return <div>  
      <h1>Timer: {this.props.match.params.time}</h1>  
      <Link to="/">Go to Home</Link>  
    </div>  
  }  
}
```



```
export default class Home extends React.Component {  
  componentDidMount() {  
    const { push } = this.props.history  
    setTimeout(() => push('/timer/10'), 1000)  
  }  
  render() {  
    return <div>  
      <h1>Home</h1>  
      <Link to='/timer/190'>Go to timer</Link>  
    </div>  
  }  
}
```

```
export default class Home extends React.Component {  
  componentDidMount() {  
    const { push } = this.props.history  
    setTimeout(() => push('/timer/10'), 1000)  
  }  
  render() {  
    return <div>  
      <h1>Home</h1>  
      <Link to='/timer/100'>Go to timer</Link>  
    </div>  
  }  
}
```

Ejercicio

- Partiendo de **react/010**, haz una app con dos páginas
 - La primera tiene un **formulario** para introducir un número de minutos
 - La segunda muestra un **contador** con los minutos introducidos
 - Se pasan el parámetro por url

```
export default class App extends React.Component {  
  render() {  
    const { users } = this.state  
    return (  
      <Router>  
        <Switch>  
          <Route exact path="/" render={({props}) => <UserList {...props} users={users} />} />  
        </Switch>  
      </Router>  
    )  
  }  
}
```

Ejercicio

- Partiendo de **react/011**, haz una app con 4 páginas
 - **Listado** de usuarios
 - Formulario para **crear** usuarios
 - Formulario para **editar** usuarios
 - Página de confirmación de **borrado** de usuarios