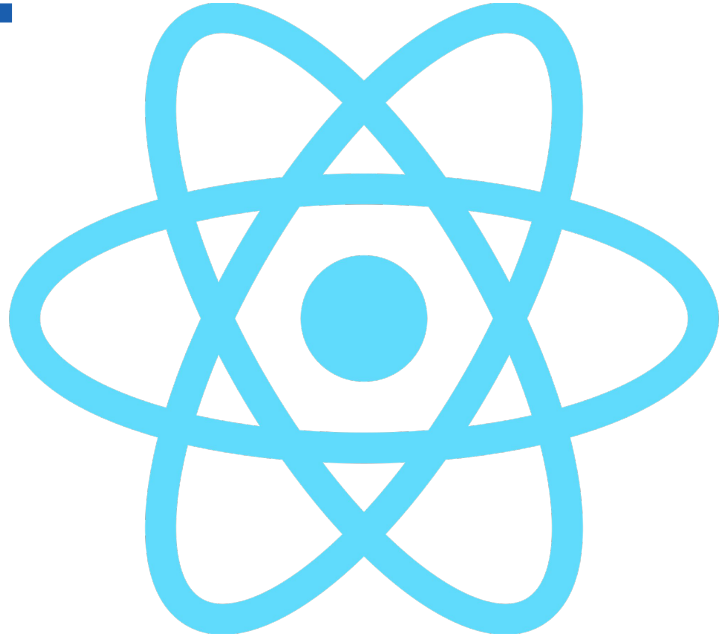
Clase 2

REACT



Temario

1. React
2. **React Components and Props**
3. Composing React Components and Stateful React Components
4. Events, conditional rendering
5. Lists and forms
6. React Router I
7. React Router II
8. Debugging and Testing

¿Qué son los elementos?

Los elementos son los bloques más pequeños de las aplicaciones de React.

Un elemento describe lo que quieres ver en la pantalla.

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

A diferencia de los elementos del DOM de los navegadores, los elementos de React son objetos planos, y su creación es de bajo coste. React DOM se encarga de actualizar el DOM para que coincida con los elementos de React.

No hay que confundir los ***elementos*** con los "***componentes***". Los elementos son los que constituyen los componentes.

Renderizando un elemento en DOM

Las aplicaciones construidas solamente con React usualmente tienen un único nodo raíz en el DOM. Dado el caso que estés integrando React en una aplicación existente, puedes tener tantos nodos raíz del DOM aislados como quieras.

Se llama nodo "raíz" porque todo lo que esté dentro de él será manejado por React DOM.

Para renderizar un elemento de React, primero pasamos el elemento del DOM a ***ReactDOM.createRoot()***, luego pasamos el elemento de React a ***root.render()***:

```
const root = ReactDOM.createRoot(
  document.getElementById('root')
);
const element = <h1>Hello, world</h1>;
root.render(element);
```

Actualizar un elemento renderizado

Los elementos de React son inmutables. Una vez creas un elemento, no puedes cambiar sus hijos o atributos.

Con nuestro conocimiento hasta este punto, la única manera de actualizar la interfaz de usuario es creando un nuevo elemento, y pasarlo a root.render().

En la práctica, la mayoría de las aplicaciones de React solo llaman a root.render() una vez. En las siguientes secciones aprenderemos cómo el código se puede encapsular en componentes con estado.

Aunque creamos un elemento en cada instante, React DOM sólo actualiza el texto del nodo cuyo contenido cambia. Abrir el inspector y revisarlo.

```
// src/index.js
import React from "react";
import ReactDOM from "react-dom/client";
import './index.css';

const root = ReactDOM.createRoot(
  document.getElementById('root')
);

function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  root.render(element);}

setInterval(tick, 1000);
```

Componentes

Los componentes son como las funciones de JavaScript, aceptan entradas arbitrarias (llamadas “props” - abreviatura de propiedades) y retornan elementos de React que describen lo que debe aparecer en la pantalla.

Los componentes permiten separar la interfaz de usuario en piezas independientes, reutilizables y pensar en cada pieza de forma aislada.

Casi todo en React consiste en componentes. Hasta ahora hemos creado un único componente App dentro de index.js.

Los componentes suelen tener su propio archivo, así que vamos a copiar el proyecto my-second-app de la clase 1 en otra carpeta de nombre **my-third-app** en la carpeta de la clase 2.

1. Creamos un archivo llamado App.js y movemos nuestra clase App de index.js a App.js
2. Exportamos nuestro componente con: export default App;
3. Importamos el componente App en index.js: import App from ‘./App’;
4. Observa la web, el resultado es el mismo.

Componentes funcionales y de clase

Existen 2 tipos de componentes:

Funcionales

- Es la forma más sencilla de definir un componente
- Se trata de una función de javascript que devuelve JSX
- Debe devolver un único elemento padre

Clase

- Debe tener un método render y debe devolver un único elemento padre

El nombre debe empezar por una letra mayúscula para diferenciarlo de los elementos HTML.

Se pueden mezclar componentes de Clase y Funcionales. Por ejemplo:

- Copiamos el proyecto my-third-app en my-fourth-app
- Creamos un componente Table.js
- Importamos el componente en App.js y lo mostramos

```
// src/Table.js
import React, { Component } from "react";
// Componente funcional con arrow function
const TableHeader = () => {
  return (
    <thead>
      <tr>
        <th>Name</th>
        <th>Job</th>
      </tr>
    </thead>
  );
};
// Componente funcional con función tradicional
function TableBody() {
  return (
    <tbody>
      <tr>
        <td>Charlie</td>
        <td>Janitor</td>
      </tr>
      <tr>
        <td>Mac</td>
        <td>Bouncer</td>
      </tr>
    </tbody>
  );
}
// Componente de clase
class Table extends Component {
  render() {
    return (
      <table>
        <TableHeader />
        <TableBody />
      </table>
    );
  }
}

export default Table;
```

Props

Siguiendo con el ejemplo anterior, ahora tenemos un componente **Table** pero con datos hardcodeados, por tanto, en una aplicación dinámica no tiene sentido. React gestiona los datos a través de ***props*** (abreviatura de propiedades) y estado.

- Copiamos nuestro proyecto my-fourth-app en my-fifth-app
- Creamos un array people con algunos registros con name y job en el método render de App.js
- Enviamos el array a Table a través de una propiedad peopleData
- En el componente Table de Table.js obtenemos la propiedad y se la enviamos a TableBody en una propiedad con el mismo nombre

```
class App extends Component {
  render() {
    const people = [
      {
        name: "John",
        job: "Developer"
      },
      {
        name: "Maya",
        job: "Architect"
      }
    ];
    return (
      <div className="container">
        <Table peopleData={people} />
      </div>
    );
  }
}
```

```
class Table extends Component {
  render() {
    const { peopleData } = this.props;
    return (
      <table>
        <TableHeader />
        <TableBody peopleData={peopleData} />
      </table>
    );
  }
}
```

Props

- Modificamos nuestro componente TableBody para recoger la propiedad peopleData y renderizar los datos.
  Usamos el método map para obtener las filas a mostrar.

- Las keys ayudan a React a identificar qué datos han cambiado, han sido añadidos o eliminados.

```
function TableBody(props) {
  const rows = props.peopleData.map((row, index) => {
    return (
      <tr key={index}>
        <td>{row.name}</td>
        <td>{row.job}</td>
      </tr>
    );
  });

  return <tbody>{rows}</tbody>;
}
```

Props

Otra de las características de las props es que aceptan elementos o componentes, es decir, se puede enviar un componente a través de una prop.

- Creamos un elemento title que sea un h1 con un texto
- Enviamos ese elemento al componente Table con una prop llamada title
- En el componente Table de Table.js obtenemos la propiedad title y la renderizamos arriba de la tabla

```
class Table extends Component {
  render() {
    const { peopleData, title } = this.props;
    return (
      <>
        {title}
        <table>
          <TableHeader />
          <TableBody peopleData={peopleData} />
        </table>
      </>
    );
  }
}
```

```
// src/App.js
import React, { Component } from "react";
import Table from './Table';
// Componente de clase
class App extends Component {
  render() {
    const people = [
      {
        name: "John",
        job: "Developer"
      },
      {
        name: "Maya",
        job: "Architect"
      }
    ];
    const title = <h1>Nice People</h1>;
    return (
      <div className="container">
        <Table peopleData={people} title={title} />
      </div>
    );
  }
}
```

Ejercicios

1. Replicar el código de clase:
    - Crear app my-timer-app con el código de la sección **Actualizar un elemento renderizado**
    - App my-third-app con la refactorización de los componentes
    - App my-fourth-app con el renderizado de la tabla con datos hardcodeados
    - App my-fifth-app con el uso de props
2. Crear un App llamada my-greetings-app que contenga
    - Un componente llamado Welcome que acepte una prop **name** y muestre un componente HTML h2 con el texto: "*Hola {name}, bienvenido al curso de los Albañiles Digitales*"
    - El componente App debe renderizar 3 componentes Welcome con diferentes nombres

ALBAÑILESDIGITALES

Developing your new career

servicio navarro de empleo

das·nano

VeriDas