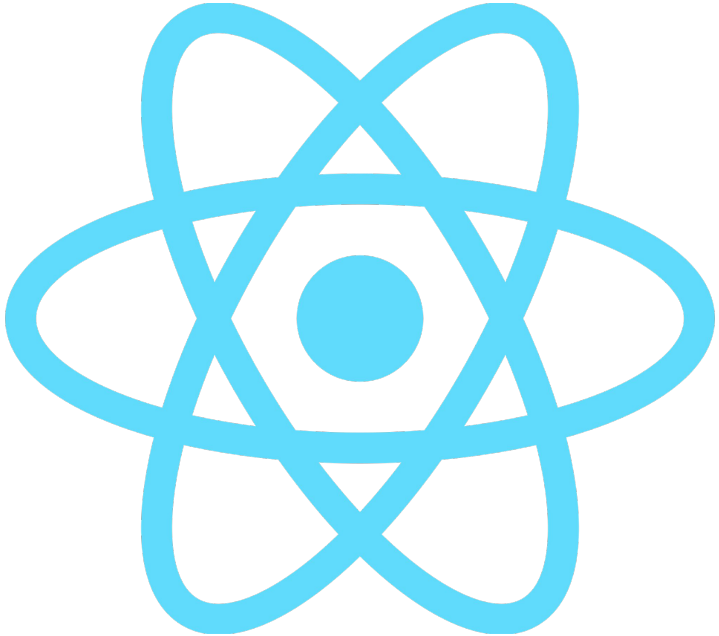
Clase 1

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 es REACT?

Es una biblioteca de JavaScript declarativa, eficiente y flexible para construir interfaces de usuario:

- **Declarativo**: Diseña vistas simples para cada estado en tu aplicación y React se encargará de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambien.
- Basado en **componentes**: Crea componentes encapsulados que manejen su propio estado, y conviértelos en interfaces de usuario completas. Permite componer interfaces de usuario a través de piezas pequeñas y aisladas llamadas “componentes”.
- NO es un framework.
- Es open source.
- Es utilizado para construir interfaces en el frontend.

Instalando React

Hay varios métodos para configurar React

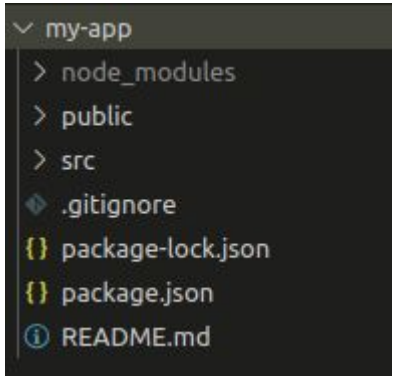
1. **Static HTML File**: El primer método consiste en crear un archivo básico index.html en el que cargaremos React, React DOM y Babel a través de un CDN (Content Delivery Network)

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />

    <title>Hello React!</title>

    <script src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
    <script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>
</head>

<body>
    <div id="root"></div>

    <script type="text/babel">
        // El código React va aquí

        class App extends React.Component {
            render() {
                return <h1>Hello world!</h1>
            }
        }

        ReactDOM.render(<App />, document.getElementById('root'))
    </script>
</body>

</html>
```

Instalando React

2. **Create React App**: Facebook ha desarrollado un entorno que viene con todo lo que se necesita para arrancar una aplicación de React pre-configurado.
   Crea un servidor de desarrollo, usa Webpack para compilar JSX y ES6, configura archivos css y usa ESLint para evitar errores en el código.

```
npx create-react-app my-app
```




```
cd my-app && npm start
```

```
Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.123:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Abre http://localhost:3000/

Create React App

Estructura

Si observamos la estructura del proyecto podemos encontrar 2 carpetas:

- **/public**: donde se almacenan los datos públicos como el icono de la página, logos, un archivo de manifiesto que contiene metadatos, y el archivo más importante **index.html** que es similar al creado con el primer método.
- **/src**: este directorio contiene todo nuestro código de React.

Además, se han creado algunos otros archivos ya conocidos como .gitignore, package.json y README.md junto a la carpeta node_modules.

Servidor en vivo

Para verificar que nuestro servidor compila automáticamente tras un cambio.

- Busca el archivo /src/App.js y reemplaza el texto “Edit <code>src/App.js</code> and save to reload.” por cualquiera a tu elección y guarda.
- Observa que el servidor en la consola así como el navegador se ha recargado automáticamente.

Create React App

Ejercicio

Replicar la misma aplicación generada con el primer método de configuración e instalación de React: Hello World!

1. Creamos otra aplicación de react con: npx create-react-app my-second-app
2. Eliminamos todos los archivos de la carpeta src/ excepto index.js e index.css
3. En el archivo index.js importamos: import React from 'react'; import ReactDom from 'react-dom'; import './index.css';
4. Copiamos la clase de nuestro primer ejemplo.
5. Cambiamos el estilo del tag h1 a través del archivo index.css y vemos el comportamiento en el navegador.

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

class App extends React.Component {
  render() {
    return (
      <h1>
        Hello world!
      </h1>
    );
  }
}

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root"));
```

JSX

Cómo habréis notado, hemos estado usando algo que parece HTML dentro de nuestro código React, pero no es HTML ni un string, es JSX (JavaScript XML)

JSX es una extensión de JavaScript con el que podemos escribir lo que parece HTML.

No es obligatorio usar JSX para escribir código React, en realidad, el código JSX usa por debajo **createElement** para crear un objeto con sus propiedades y contenido. Por ejemplo, ambas variables contienen el mismo objeto.

JSX está más cerca de JS que de HTML por tanto hay que tener en cuenta algunas características:

- ***className*** se usa en lugar de ***class*** ya que class es una palabra reservada en JS
- Las propiedades y métodos in JSX están en camelCase, por ejemplo: ***onclick => onClick***
- Los tags de cierre automático deben acabar con un slash: **<img />**

Existen herramientas para convertir tu código HTML en JSX: https://transform.tools/html-to-jsx

```
const heading = <h1 className="site-heading">Hello world!</h1>
```

```
const heading = React.createElement('h1', { className: 'site-heading' }, 'Hello world!');
```

JSX

Se pueden insertar expresiones javascript dentro de JSX. Para ello usamos las llaves.

A continuación vamos a declarar una variable name que insertaremos dentro de JSX.

```
const name = 'Fausto López';
const heading = <h1>Hello {name}</h1>;
```

Se puede poner cualquier expresión de javascript, por ejemplo: 2 + 2, user.first_name, formatName(user);

```
const suma = <span>La suma de 2 más 2 es: {2+2}</span>;

const name = "Fausto López";
const upperFunction = (name) => String(name).toUpperCase();
const heading = <h1>Hello {upperFunction(name)}</h1>;
```

Puesto que JSX también es una expresión se puede usar dentro de condicionales ***if*** y bucles ***for*** y asignarlo a variables, aceptarlo como argumento y retornarlo desde funciones.

```
class App extends React.Component {

  upperFunction = (name) => String(name).toUpperCase();
  getGreeting(user) {
    if (user) {
      return <h1>Hello, {this.upperFunction(user)}!</h1>;
    }
    return <h1>Hello, Stranger.</h1>;
  }
  render() {
    const name = "Fausto López";
    return this.getGreeting(name);
  }
}
```

JSX

Se puede especificar atributos con JSX, tanto con comillas para expresar strings literales, así como usando llaves para insertar una expresión JS.

```
const element = <a href="https://react.dev/"> React </a>
```

```
const element = <img src={user.avatarUrl}></img>;
```

Si un tag está vacío se puede cerrar inmediatamente con ' />':

```
const element = <img src={user.avatarUrl} />;
```

JSX previene ataques de inyección puesto que escapa cualquier valor insertado antes de renderizarlo. De este modo, se asegura de que nunca se pueda insertar nada que no esté explícitamente escrito en tu aplicación. Esto ayuda a prevenir vulnerabilidades XSS.

Al igual que en HTML los tags pueden contener hijos:

```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

Ejercicios

1. Configurar vuestro editor VSCode: Linting (sólo ESLint) y Formatting
2. Instalar las React Developer Tools para vuestro navegador
3. Replicar el código de clase:
    - App con método Static HTML file
    - App con create-react-app modificando App.js
    - App con create-react-app replicando el ejemplo “Hello world!” con cambio de estilos y añadiendo algún elemento visto en clase.

ALBAÑILESDIGITALES

Developing your new career

servicio navarro de empleo

das·nano

VeriDas