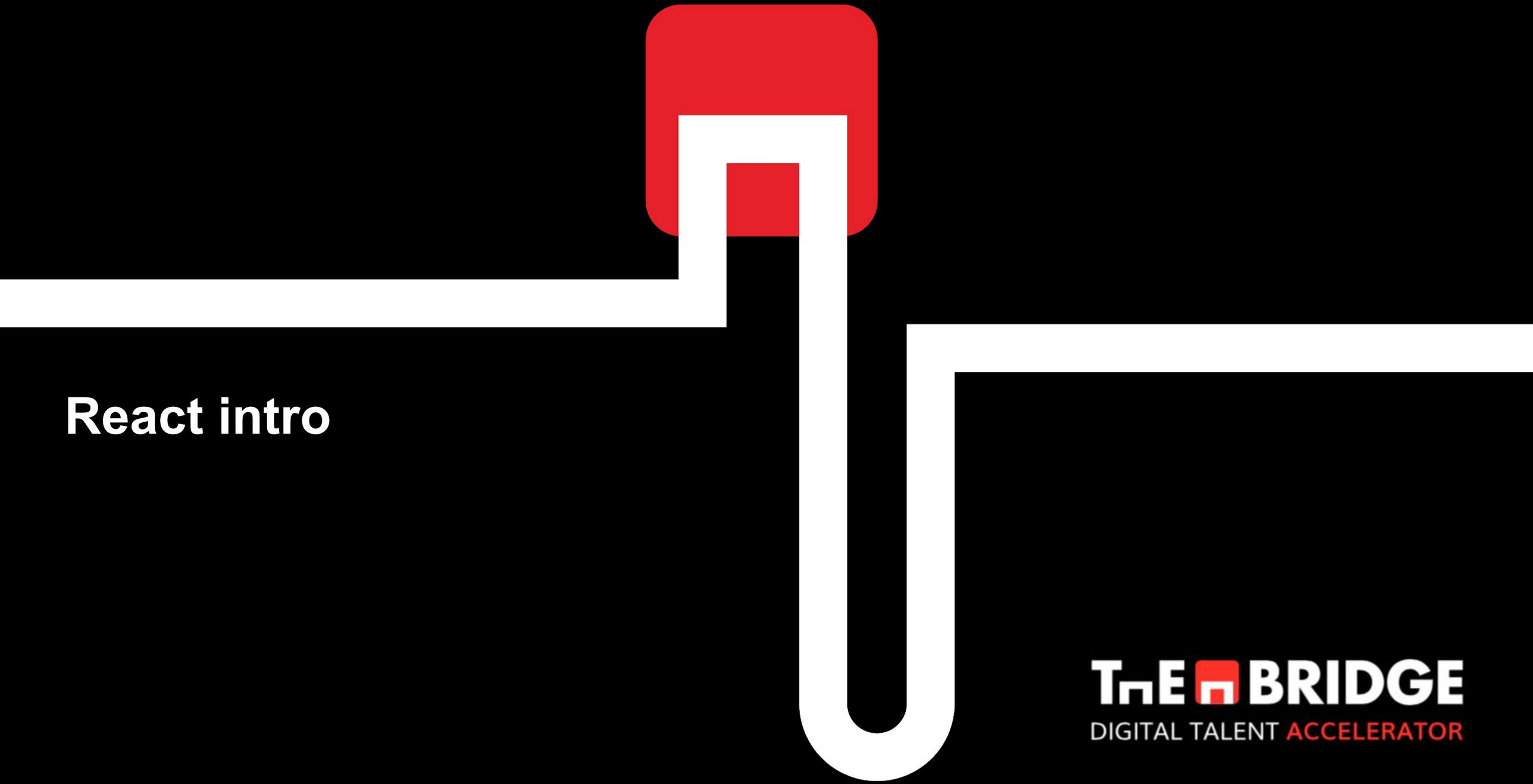
React intro



THE BRIDGE
DIGITAL TALENT ACCELERATOR

Índice

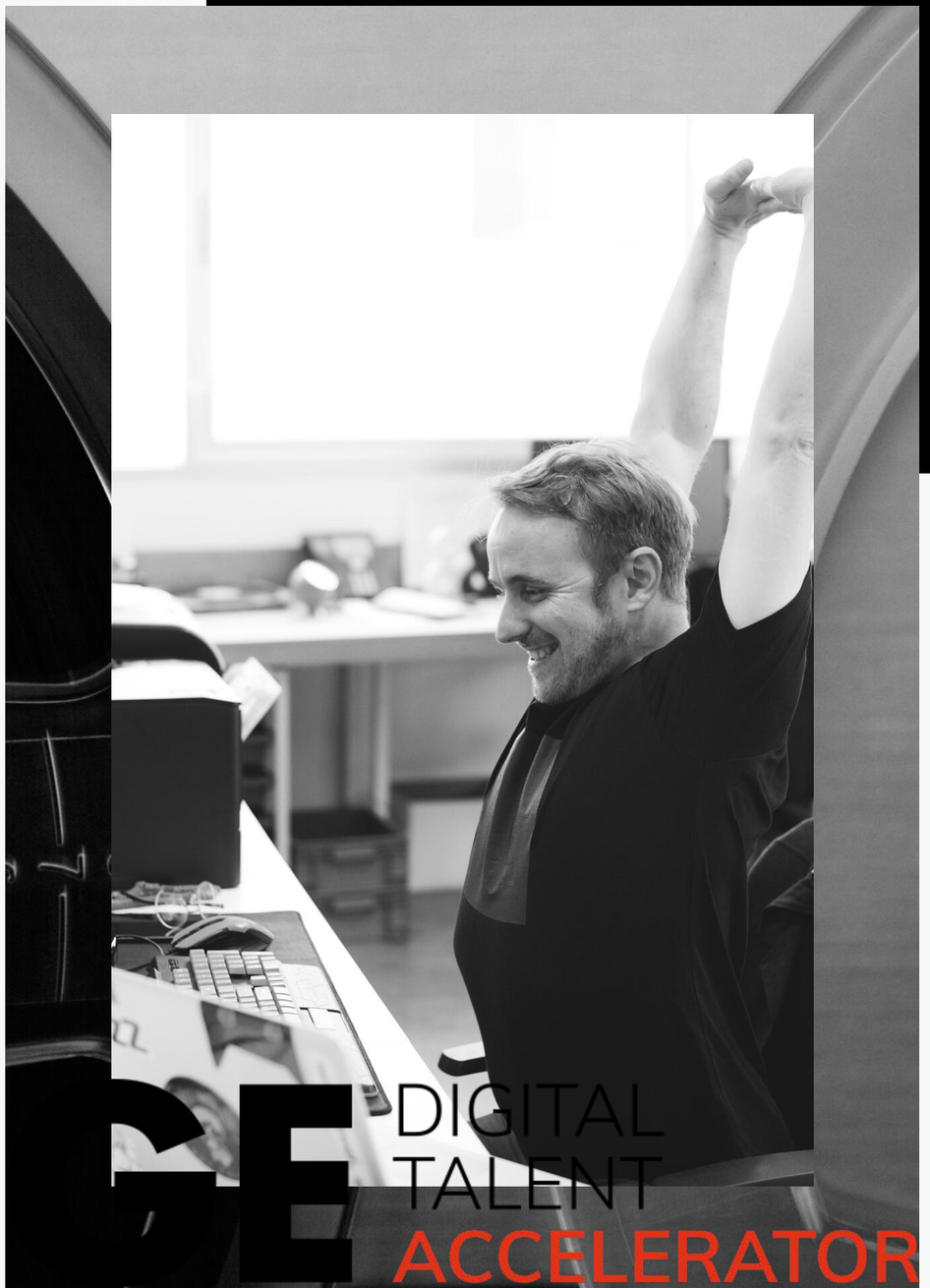
¿Que es React?

Empaquetadores de módulos JS

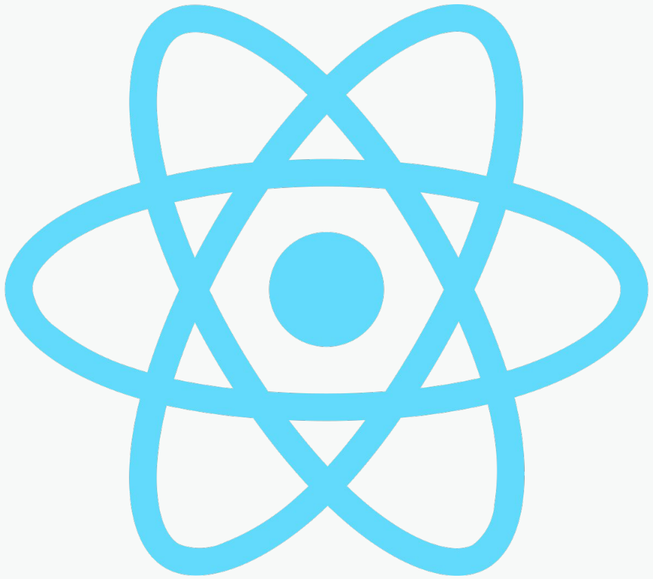
JSX

Componentes y Props

Listas



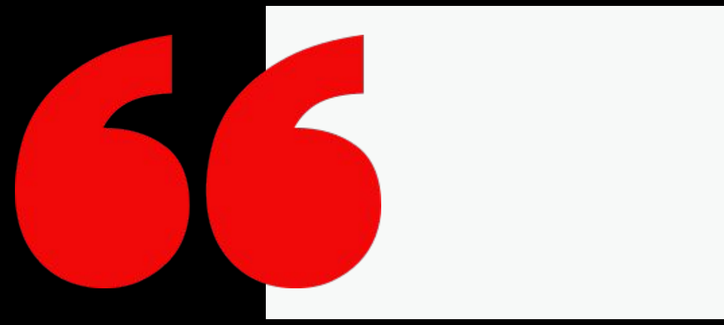
¿Qué es React?



React es una **librería de JavaScript** para construir interfaces de usuario. Es una librería front-end de **código abierto**, basada en **componentes**, responsable sólo de la capa de visualización de la aplicación.
Una de sus **principales características** es que no se limita a las web, Por ejemplo con **React Native** podemos crear **aplicaciones móviles** y con **React 360 aplicaciones de realidad virtual**.

React es una **SPA** (single page application), una web donde la mayoría de la **interaction con el usuario ocurre en una solo página**. Nuestra aplicación tendrá **un solo fichero HTML**, llamado **index.html**, donde REACT cargará dinámicamente otras piezas de HTML.

¿Quién usa React?



Empaquetadores de módulos JS

Module bundler

Empaquetadores de módulos

Cada vez vamos a requerir más código JavaScript. Más código implica un mayor orden, a fin de tener **todo en su sitio** y tener una aplicación **no muy costosa de mantener**. Además, antes de subir a producción requerimos de otros procesos:

- **Minificar** nuestros assets (archivos Javascript y CSS).
- **Empaquetar** muchos módulos en una cantidad mínima de archivos Javascript.
- Hacer que nuestro código Javascript sea **compatible con todos los navegadores** (uso del último estándar de Javascript, que no siempre se encuentra implementado por los navegadores).
- Si usamos **preprocesadores**, también necesitamos desde un inicio el **proceso de conversión**.

Un empaquetador de módulos (module bundler) permite optimizar todo este proceso.

Vite

Vite es una herramienta frontend (**empaquetador de módulos**) que permite trabajar con el código. Realiza por nosotros las correspondientes **transformaciones necesarias** para poder ejecutarse en los navegadores de manera optimizada.

En particular, **Vite React** es una configuración predeterminada de Vite que está optimizada para proyectos de React.

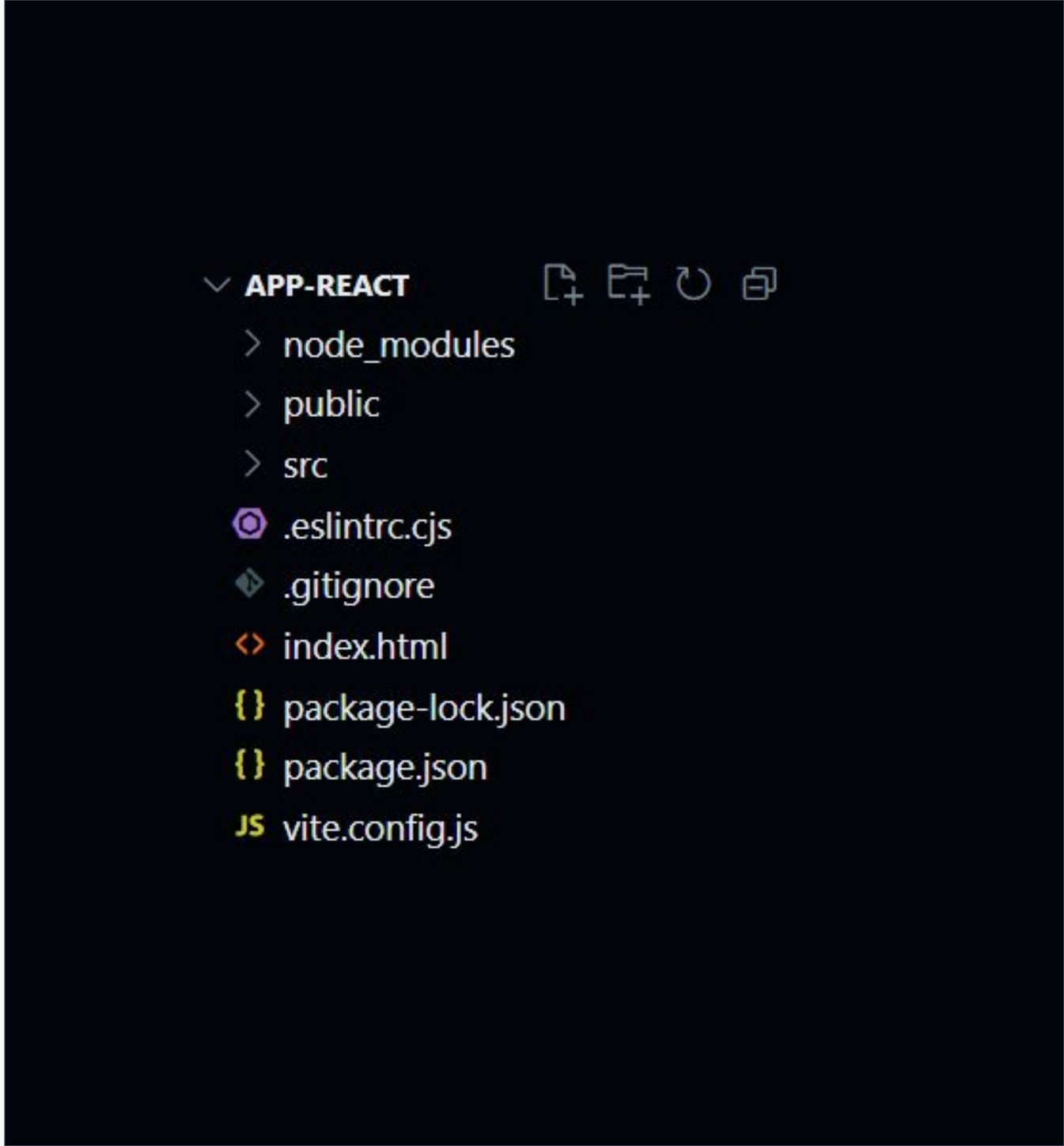
Instalación

Creando nuestra app de React con vite

```
$ npm create vite@latest
```

Estructura de carpetas

Una vez que creamos un proyecto en React, obtenemos una **estructura de carpetas** general de la aplicación como se muestra en la imagen:

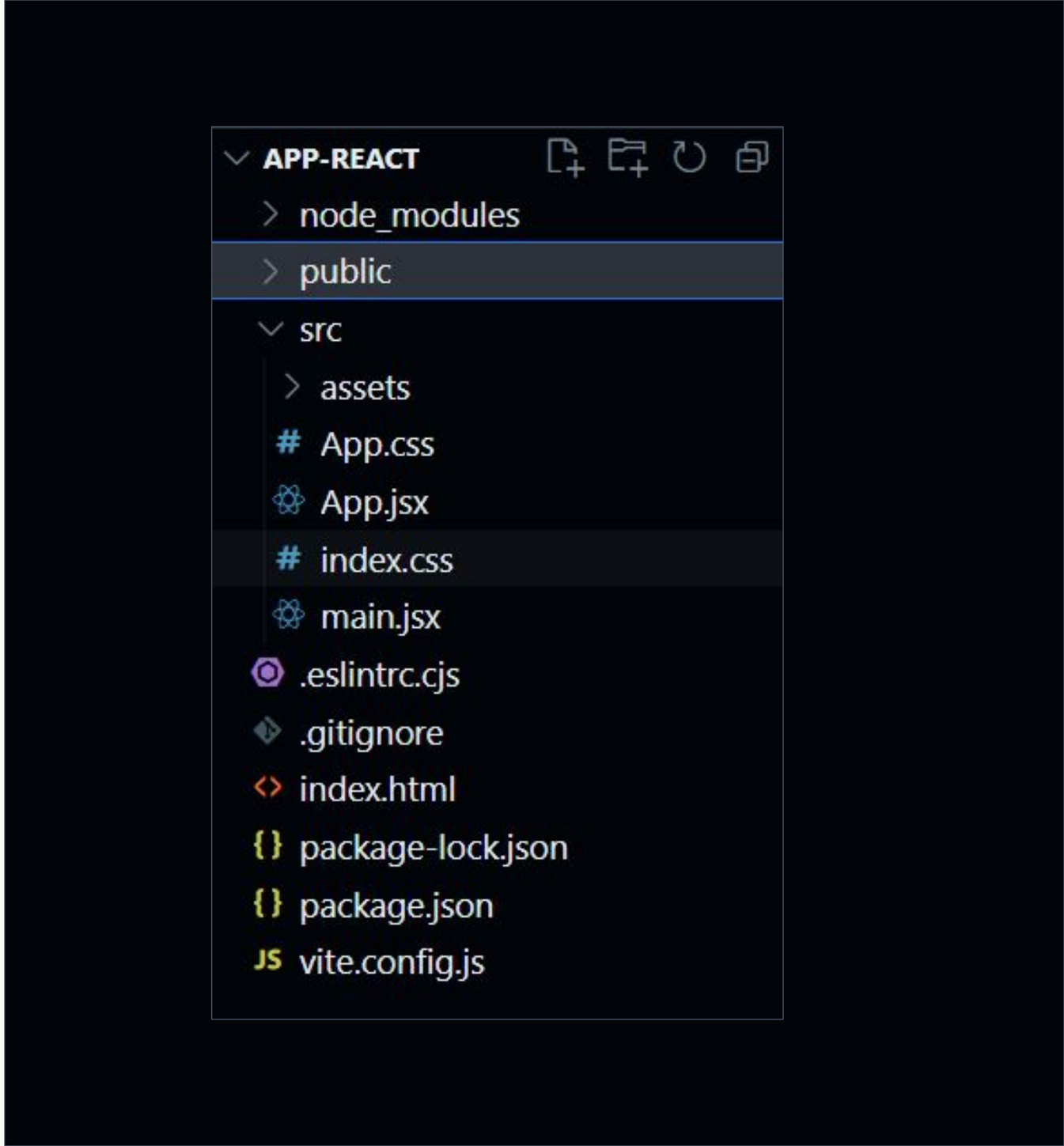


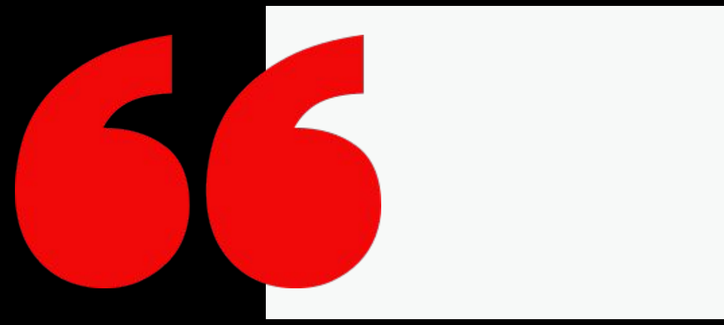
src

src es donde está la mayor parte del código de la aplicación.

App.jsx es el **archivo principal** de la aplicación que define el **componente principal**.

main.jsx es el **archivo de entrada** de la aplicación que se encarga de **renderizar** el componente principal **(App.jsx)** en el DOM.





JSX

JavaScript Syntax Extension

JSX

JSX es una **extensión de sintaxis** para JavaScript que permite escribir **marcado similar a HTML** dentro de una archivo JavaScript. Hasta ahora, el contenido en HTML, el diseño en CSS, y la lógica en JavaScript, iban archivos separados.

Al usar JSX, podemos escribir **estructuras HTML** en el mismo archivo que contiene código **JavaScript**. Esto hace que el código sea más fácil de entender y depurar, ya que evita el uso de estructuras DOM de JavaScript complejas.

Ejemplo JSX

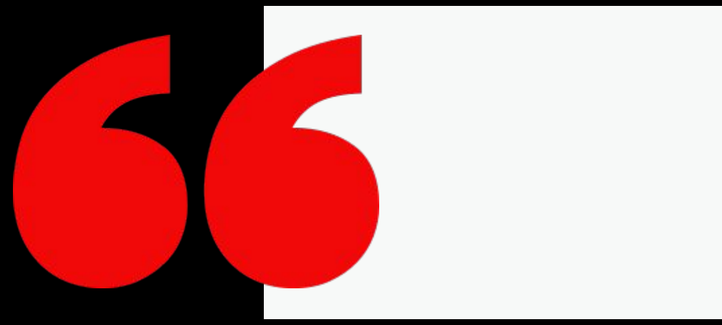
En el siguiente ejemplo, declaramos una variable llamada **name** y luego la usamos dentro de la variable **whoIAm**, envolviéndola entre llaves:

```
const name = "Patricia"
const whoIAm = <h1>Hola, me llamo { name }</h1>
```

Ejemplo JSX en App.js

En el siguiente ejemplo, declaramos una variable llamada name y luego la usamos en nuestro componente principal App:

```
const name= 'Patricia'

function App() {
  return (
    <div className="App">
      <p>Hola, me llamo {name}</p>
    </div>
  );
}

export default App;
```



Implementando estilos

App.css

Creamos una clase en el archivo App.css

```
.text-color{
  color: red;
}
```

Implementamos la clase

Nos importamos el archivo.

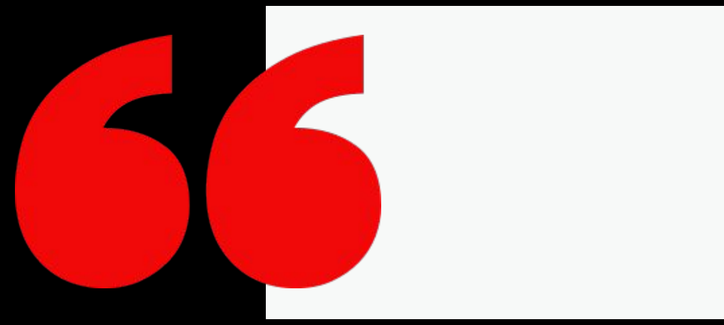
Implementamos la clase que hemos creado donde queramos.

Importante: no ponemos “**class**” utilizamos la palabra “**className**”.

```
import './App.css';
```

Nos importamos el archivo css para conectarlo a nuestro App.jsx

```
const name= "Patricia"

function App() {
  return (
    <div className="App">
      <p>Hola, me llamo {name}</p>
    </div>
  );
}

export default App;
```



Componentes & Props

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

Instagram

Echemos un vistazo a un ejemplo de página web de Instagram , para entender mejor cómo funciona **React** y sus **componentes**.

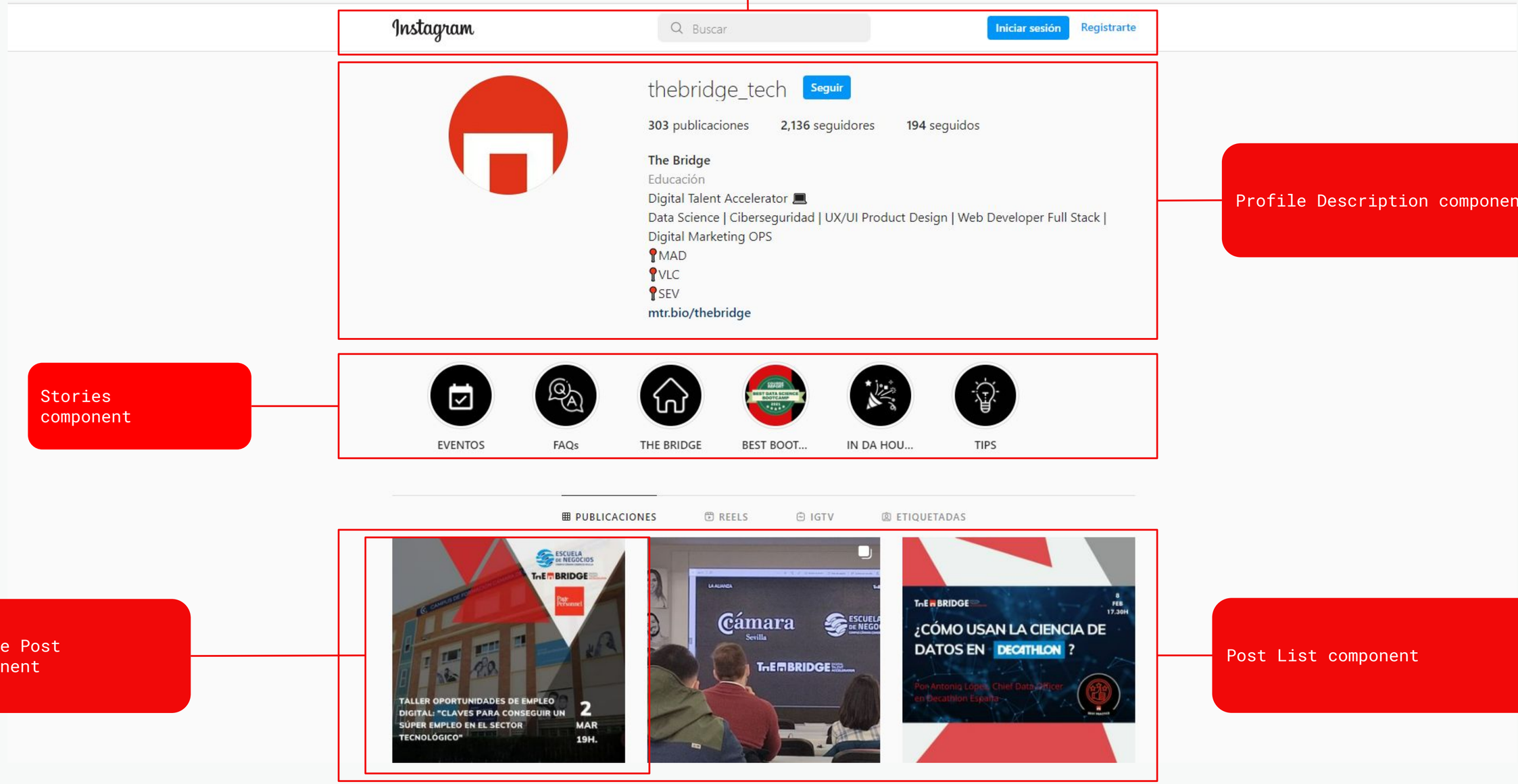
Header component

Profile Description component

Stories component

Single Post component

Post List component



Instagram

Como muestra el ejemplo anterior, React divide la interfaz de usuario en varios **componentes**, lo que hace **debugear el código más fácilmente**. De esta forma, **cada componente tiene su propia función**.

Ejemplo

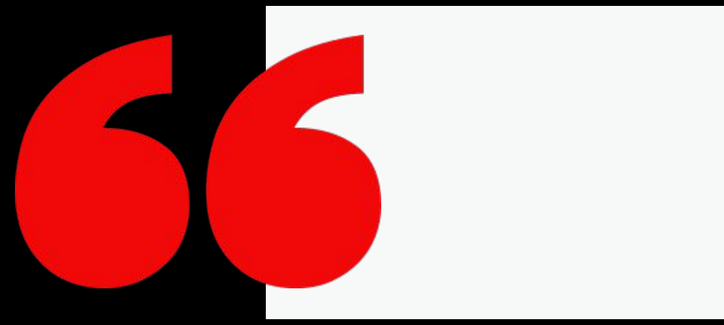
Así es como los **componentes** se distribuyen en el código.

```
function App() {
  return (
      <div className="App">
          <Header />
          <ProfileDescription />
          <Stories />
          <PostList />
      </div>
  );
}
```

Extensión

Extensión VSC para React

https://marketplace.visualstudio.com/items?itemName=dsznajder.es7-react-js-snippets



Creando nuestro primer componente

Componente funcional

La forma más sencilla de definir un **componente** es escribir una **función de JavaScript**:

```
const Welcome = () => {
  return <h3>Hello The Bridge</h3>
}
```

Componente de Clase

También puede usar una clase ES6 para definir un componente:

```
import React from 'react';

class Welcome extends React.Component {
  render() {
      return <h1>Hello The Bridge</h1>;
  }
}
```

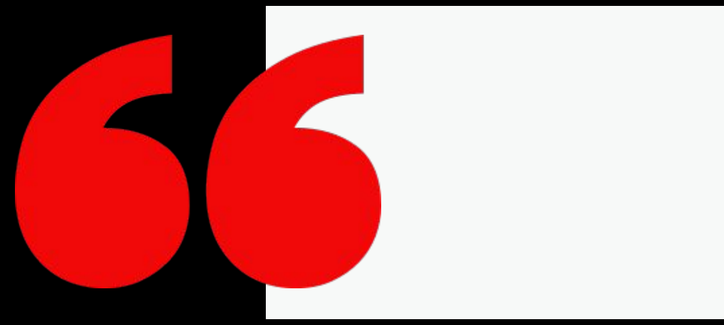
Reutilizando componentes

Ahora en nuestro archivo App.jsx reutilizamos el componente Welcome tantas veces como queramos:

```
import './App.css';

const Welcome = () => {
  return <h3>Hello The Bridge</h3>
}

function App() {
  return (
    <div className="App">
      <Welcome/>
      <Welcome/>
    </div>
  );
}

export default App;
```

Naming de los componentes

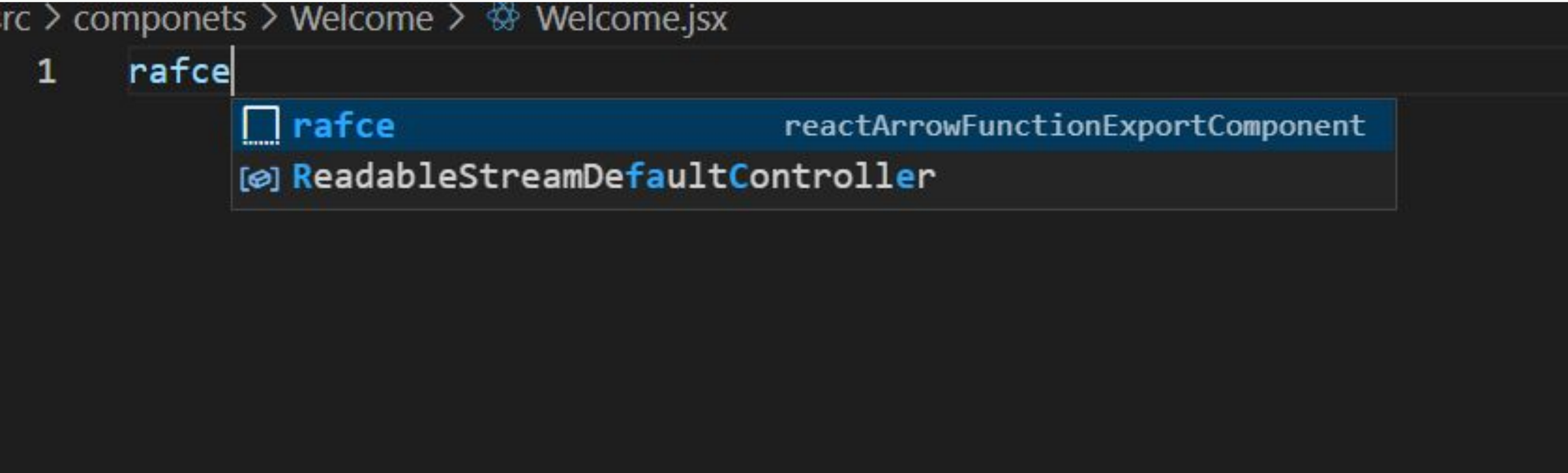
***Importante**: los nombres de los componentes **siempre** empiezan con una **letra mayúscula**. React trata los componentes que empiezan con letras minúsculas como etiquetas DOM.



Creando componentes en otros archivos

Snippet para crear componentes

Con el siguiente **snippet** nos crea automáticamente la **estructura** de un componente de **componente funcional** (rafce) o **componente de clase** (rcc):



Creando un componente

Creamos la **carpeta components**. Dentro de esta carpeta creamos una por cada componente. En este caso una carpeta **Welcome** y dentro un **archivo** llamado **Welcome.jsx** con el siguiente código:

```jsx
import React from "react";

const Welcome = () => {
  return <h3>Hello The Bridge</h3>;
};

export default Welcome;
```

Importando componente

Ahora en nuestro archivo **App.js** nos **importamos el componente** y lo utilizamos **dentro del return**:

```
import './App.css';
import Welcome from "./components/Welcome/Welcome"

function App() {
  return (
    <div className="App">
      <Welcome />
    </div>
  );
}

export default App;
```

Nos importamos el componente que acabamos de crear

Implementamos el componente Welcome

Props

React es una **librería basada en componentes** que divide la interfaz de usuario en pequeñas piezas reutilizables. En algunos casos, estos componentes necesitan comunicarse (**enviarse datos entre sí**) y la forma de pasar datos entre componentes es mediante **props**.

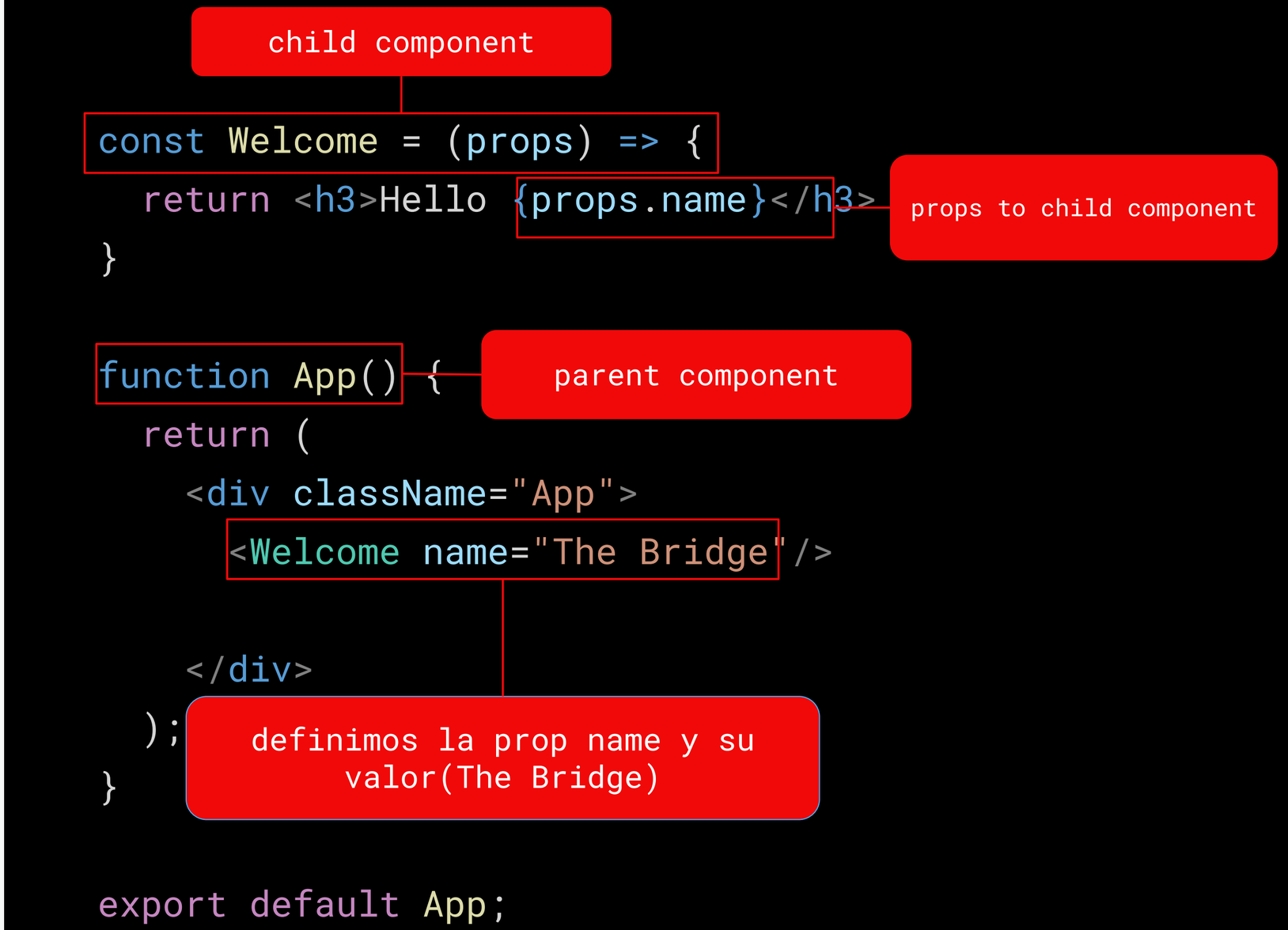
"props" es una **palabra clave especial,** que significa **propiedades** y se usa para **pasar datos de un componente a otro**.

Usando props

Props paso a paso:

- En primer lugar, pasamos las **props** a los **componentes hijos**.
- Luego definimos un **atributo** y su **valor** (datos) al componente principal(el padre)
- Finalmente, renderizamos los datos de **props**

Usando props en componentes funcionales



Listas

Podemos hacer colecciones de elementos e incluirlos en JSX usando llaves {}. Para ello, recorremos el array numbers usando la función **map()** de Javascript. Devolvemos un elemento **<li>** por cada ítem. Finalmente asignamos el array de elementos resultante a **listItems**:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) => <li>{number}</li>);

function App() {
  return (
    <div className="App">
      <ul>{listItems}</ul>
    </div>
  );
}
```