Parte 1: Conceptos previos sobre React, JSX e instalación del entorno

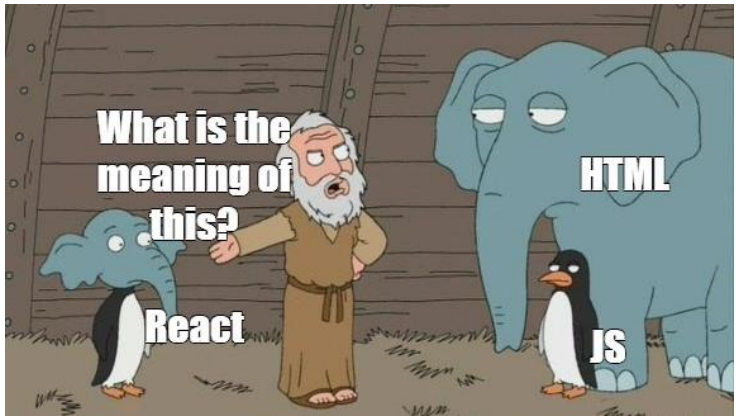
UD4: Fundamentos de React

Después de este documento…

- Sabrás de que va eso de "**React**"
- Conocerás en qué consiste el lenguaje **JSX** y algunas de sus singularidades
- Habrás escrito tu primera mini-aplicación React
- Habrás instalado el entorno que usaremos en la siguiente unidad
- Habrás ejecutado tu primera aplicación React ☺

¿Qué es React?

- **Biblioteca** *opensource* de JavaScript para crear **interfaces de usuario**
- Creada por ***Facebook***, mantenida por la comunidad de software libre
- Usada en:
  - Pinterest
  - Instagram
  - Netflix
  - Airbnb
  - …



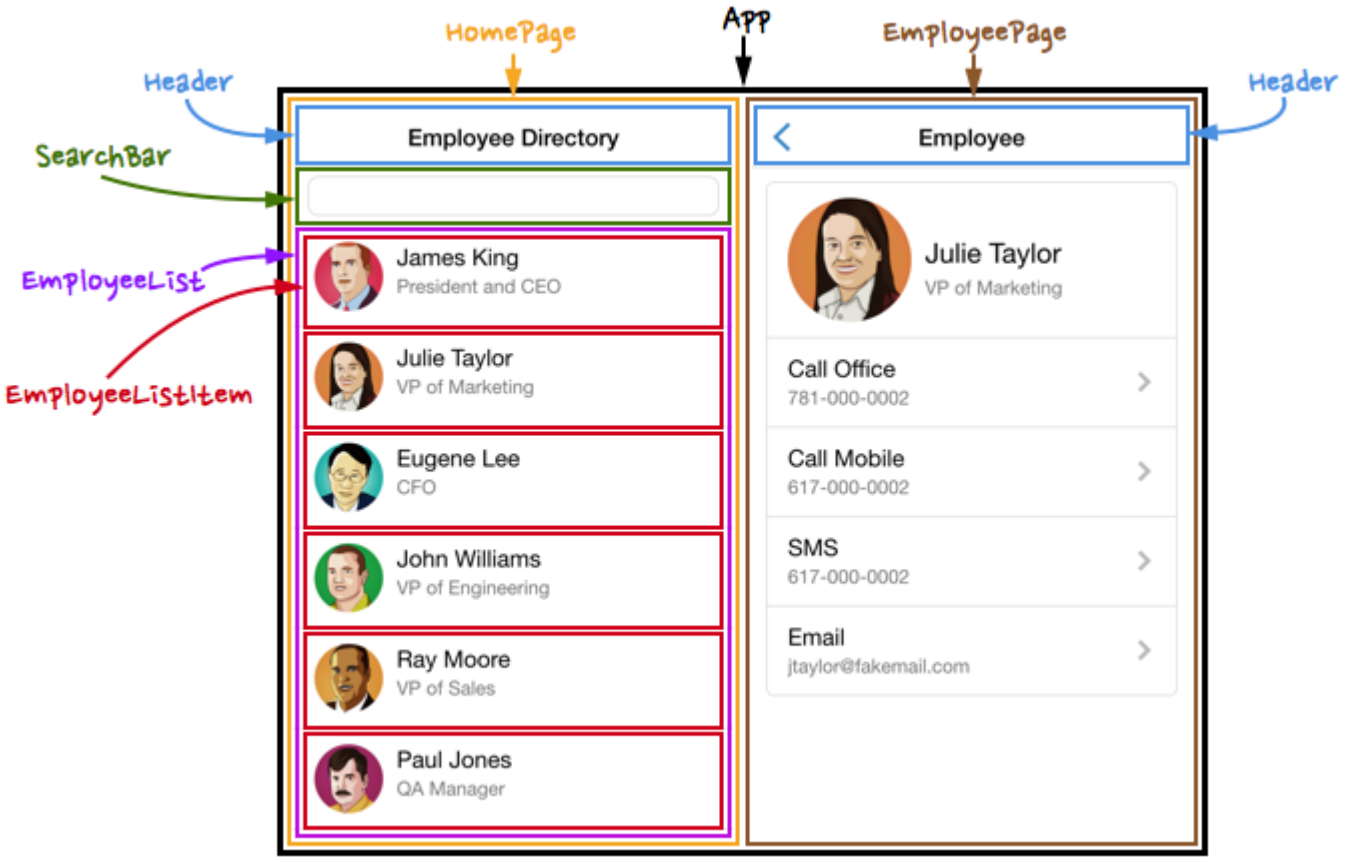
¿Por qué React?

- Tiene un porcentaje de satisfacción del **40,1%** entre los programadores, seguido por Angular (22,9%) y Vue (18,9%)
- Hay más de 5.000 ofertas de trabajo que piden React.js en España, 83.000 en la Unión Europea, y imás de 146000 en todo el mundo!
- Se puede trabajar con React del lado del **servidor** y se pueden crear aplicaciones móviles con **React Native**

Componentes en React

- Las aplicaciones en React básicamente se construyen mediante **componentes** que llaman a otros **componentes**
- Los **componentes** son porciones independientes de una interfaz
  - Facilita el mantenimiento y reusabilidad
  - En código se traducen como **funciones**
  - También se pueden hacer **componentes basados en clases** pero ni es habitual ni se recomienda (más largo, más difícil de mantener)

Sería algo así...



Node.js

- Es un entorno de ejecución de JavaScript
- Para comprobar la instalación: Abrimos una interfaz de línea de comandos (la terminal ***cmd*** o ***Windows PowerShell*** si estamos en Windows) y ejecutamos:

```
node --version
```

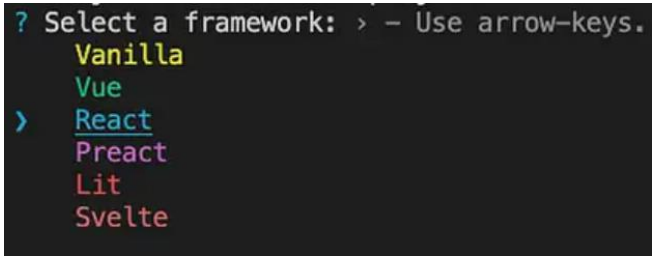
  - Si no lo tienes instalado: https://nodejs.org/es/download/
  - Si quieres actualizar
    - `npm i -g npm@latest`

npm

- Node es un sistema modular, viene prácticamente "vacío"
- Para utilizar una funcionalidad de alguna librería publicada, se instalan módulos adicionales
  - Se hace de forma muy sencilla con la herramienta **npm** (*Node Package Manager*)
    - Instalar y desinstalar paquetes
    - Gestionar versiones
    - Gestionar dependencias

Herramientas para crear un proyecto React

- **Create-React-app** (https://create-react-app.dev/)
  - Se ha usado mucho (era la "oficial"), pero ya está "abandonado"
- **Vite** (https://vitejs.dev/): Sirve para múltiples frameworks



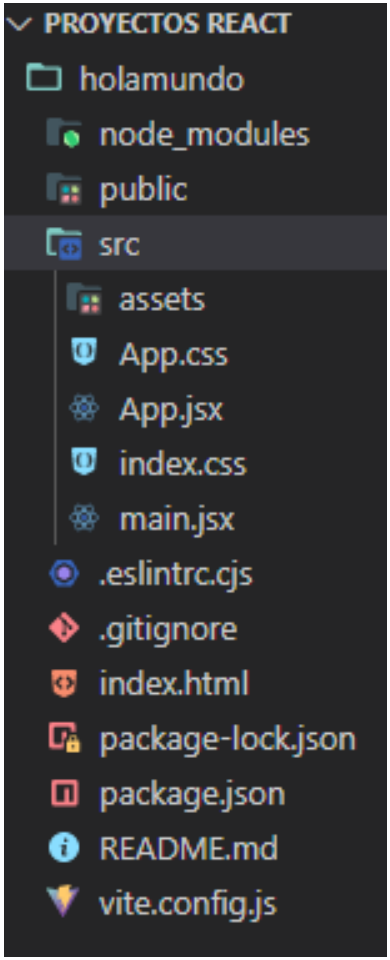
  - Es más novedosa
  - Exige extensión jsx para componentes
- **Otras**:
  - Next.js, Remix, Astro, Run, Hydrogen…

Vite

- Desde la consola del sistema o terminal de VS Code desde el directorio donde quiero trabajar, ejecutaremos: `npm create vite`
  - Nos pregunta el **nombre** del proyecto: "holamundo"
  - A continuación nos movemos con las flechas de teclado para seleccionar "**React**"
  - Escogemos "**JavaScript**" como lenguaje de desarrollo
- Nos movemos a su directorio: `cd holamundo`
- Instalamos las dependencias: `npm install`
- Y hacemos correr la aplicación: `npm run dev`
  - Vite nos indicará el host y puerto donde se está ejecutando – Con ctrl+clic lo podemos abrir

Estructura de un proyecto React

- **src**: contiene el código
  - **main.jsx + index.css:** Componente base, importación de bibliotecas
  - **App.jsx +app.css**: Componente principal
  - Resto de archivos css, jsx, js que use la aplicación, ordenados en carpetas
- **node_modules, package-lock.json, package.json :** dependencias
- **.gitignore**: crea uno por defecto (dependencias, build, etc.)
- **index.html:** estructura muy sencilla
  - `<div id="root"></div>` vacío, aquí se insertará el código que generará JavaScript a través de React



Nuestro primer componente

- Hay un componente especial que contiene a todos los demás: "***Root element***"
  - Único, situado en main.jsx
  - Se crea con **`createRoot`**
- En el resto de componentes hay dos partes:
  - Lo que está **antes** del return:
    - Código JavaScript "normal"
  - Lo que está dentro del **return**:
    - Lo que se va a visualizar: Código JSX, código JS entre llaves (ternarios, expresiones…)
      - » No puede haber funciones
      - » No puede haber condicionales
      - » …

¿Qué es JSX?

- **JSX = JavaScript XML** es una extensión de JS desarrollada por Facebook que permite escribir HTML dentro del código sin necesidad de meterlo en un *string*

```
const element = <h1>Hola mundo</h1>;
```

  - Equivalente a:

Así crea React su VirtualDOM

```
React.createElement("h1", null, "Hola mundo")
```

- Su uso no es obligatorio (al final, JSX se transforma en código JS):
  - Sí recomendado en la documentación oficial de React
- Para incrustar JS se usan {}

Ejemplo de código: Componente "Saludar"

- Cada componente devuelve código JSX

"return" obligatorio

```
function Saludar(){
 return <h1>Hola mundo, soy componente</h1>
}
```

1. Las llaves permiten ejecutar código

- Desde el componente raíz comienza la aplicación

```
ReactDOM.createRoot(document.getElementById("root")).render(
<div>
  { Saludar() }
  <Saludar></Saludar>
  <Saludar/>
</div>)
```

Hola mundo, soy componente

Hola mundo, soy componente

Hola mundo, soy componente

2. En lugar de las llaves se suele llamar al componente "convirtiéndolo" en etiqueta

3. Aún más allá, pueden usarse self-closing tags para abreviar

EJERCICIO PROPUESTO I (1/4): *Hola Mundo* con Vite

- Borramos *App.css* y *App.jsx* ya que no lo vamos a utilizar en este ejemplo
- Editamos el archivo **main.jsx**:

```
import ReactDOM from 'react-dom/client'

ReactDOM.createRoot(document.getElementById('root')).render(
<h1>Hola mundo </h1>
)
```

```
<!DOCTYPE html>
<html lang="en">
▶ <head>…</head>
▼ <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
  ▼ <div id="root">
      <h1>Hola mundo </h1>
    </div>
```

Hola Mundo

Organizar componentes

- Cada componente se corresponde con una función ("**componentes funcionales**")
  - Se usan mucho las funciones flecha
  - Pueden recibir props (ya lo veremos)
  - En el return va el código JSX, **y sólo puede devolver una cosa**
- Suele ir en un archivo con su nombre en **mayúsculas**
  - Importar react al principio ya no es necesario
  - La extensión del archivo puede ser **JS o JSX**
  - JSX permite que VSCode lo reconozca como componente. En Vite es obligatorio

```
const Intro = () => {
    return (
    <p
className="texto-intro">
      ¡Hola!
     </p>
    );
 }
export default Intro;
```

Importar el componente

- Para poder ser usado el componente debe exportarse:
  - Delante de **`function`** la palabra **“`export`”**
- Donde se quiera usar se añadirá la importación:
  - Dentro de las llaves se puede poner más de un componente separados por comas
  - Ctrl+Espacio facilita el autocompletado

```
import { Saludar } from './Saludar.jsx'
```

Importar la hoja de estilos

- OJO: La hoja de estilos se carga desde el componente, no en el html
  - Hoja.css

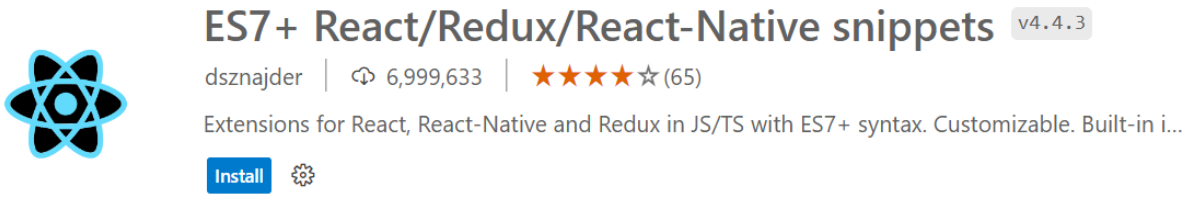
```
.card{
    background-color: "202020";
    color: 'aliceblue'
}
```

  - Componente.js

```
import './task.css'
export function TaskCard(){
    return <div className='card'>
        <h1>Mi primera tarea</h1>
        <p>Tarea realizada</p>
    </div>
}
```

Snippets

- **ES7+ React**: Extensión de **VSCode** que permite generar el código base de un componente de forma automática



  - Se genera el código escribiendo:
    - **rfce** : React function component export
    - **rafce**: Función flecha y constante
    - …
- Simple React Snippets (Burke Holland)
- Code Snippets (Eduardo Liberato)
- Etc.

EJERCICIO PROPUESTO I (2/4): *Hola Mundo* con Vite

- Crear un nuevo archivo **Saludar.jsx** para albergar un nuevo componente, que haga la misma funcionalidad que teníamos (saludar al mundo) al que llamaremos tres veces desde el main
  - Prueba a generar la estructura usando snippets
  - El componente Saludar devolverá un h1 con el texto "Hola mundo"
- Crea una hoja de estilos para el componente, **Saludar.css**
  - Haz que todo los textos h1 sean de color rojo

Particularidades de JSX

- Como en JS la palabra *class* está reservada se usa *className*
- En los estilos css en línea:
  - En lugar de envolver el estilo entre "", usamos **{{}}**:
    - La llave exterior indica que vamos a colocar una variable JS
    - La llave interior indica que es un objeto JS
  - El nombre de la propiedad pasa de usar guión entre palabras a **camelCase**
  - Valor de la propiedad entre comillas
  - Separación entre propiedades con , en lugar de ;

HTML → `class="unameinput"`

JSX → `className="unameinput"`

HTML → `style="background-color:cyan;color:blue"`

JSX → `style={{backgroundColor:'cyan',color:'blue'}}`

Insertando código JavaScript

- Podemos usar variables o funciones de JavaScript en lugar de texto usando {} donde queramos que se analice el código

```
const nombreBoton="Enviar";
<button id ={btnEnviar}>Enviar</button >
<button id="btnEnviar">{nombreBoton}</button>
<button>{getNombreBoton()}</button>
```

- También usaremos {} para incluir los comentarios // o /* */ cuando queremos por ejemplo, anular parte de lo que se está mostrando

Ejemplo

- Aplicando estilos en línea:

```
export function TaskCard(){
    return <div style={{background: '#000000', color:"#fff", padding:"20px"}}>
        <h1>Mi primera tarea</h1>
        <p>Tarea realizada</p>
    </div>
}
```

- Combinando con JS: Guardar el estilo en variable

```
export function TaskCard(){
    const cardStyles = {background: '#000000', color:"#fff", padding:"20px" }
    return <div style={cardStyles}>
        <h1>Mi primera tarea</h1>
        <p>Tarea realizada</p>
    </div>
}
```

EJERCICIO PROPUESTO I (3/4): *Hola Mundo* con Vite

- Haz que h1 con el "Hola mundo" pertenezca a la clase **"texto_rojo"** y actualiza la hoja de estilos para que solo los elementos perteneciente a la clase sean de ese color
- Define en una constante JS llamada **"resaltar"** la clase "**negrita**" y referénciala en la propiedad className, junto a la anterior
- Cambia el color de fondo usando un estilo en línea para que sea verde
- Establece en una constante el valor para el tamaño de la fuente e incorpora este valor en el estilo en línea accediendo a la variable con un *string literal*

Renderizado condicional

- Se usa intensamente el **operador ternario**

```
if (married){
    return <h1> Estoy contento </h1>
}
else
    return <h1> No estoy contento </h1>
}
```

Equivalente

```
return <h1>{married?'estoy contento 😀':"no estoy contento "}</h1>
```

Sí, se pueden usar emoticonos Unicode!

Estilos condicionales

- También se usa el operador ternario para decidir los estilos a aplicar en función de una condición

```
.bg-red{
    background: red;
}
.bg-green{
    background: green;
}
```

```
function TaskCard({ready}){
    return <div className='card'>
        <h1>Mi primera tarea</h1>
        <span
        className={ready? 'bg-green':'bg-red'}>
        {ready? "Tarea realizada":"Tarea pendiente"}
        </span>
    </div>
}
```

EJERCICIO PROPUESTO I (4/4): *Hola Mundo* con Vite

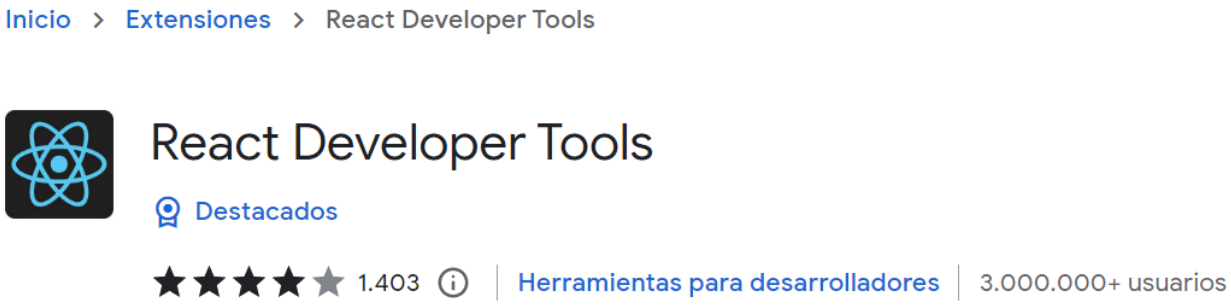
- Modifica el componente para que el mensaje a mostrar en lugar de un texto sea otro componente llamado **Mensaje** que devolverá un párrafo con un texto que cambie en función de una condición

```
const llega=false;
```

  - "llega" de momento es una constante
  - Si llega es verdadero muestra hola, sino muestra adiós
    - Usa el operador ternario para ello
- Y sí, puedes probar a añadir un emoticono ☺

React Developer Tools

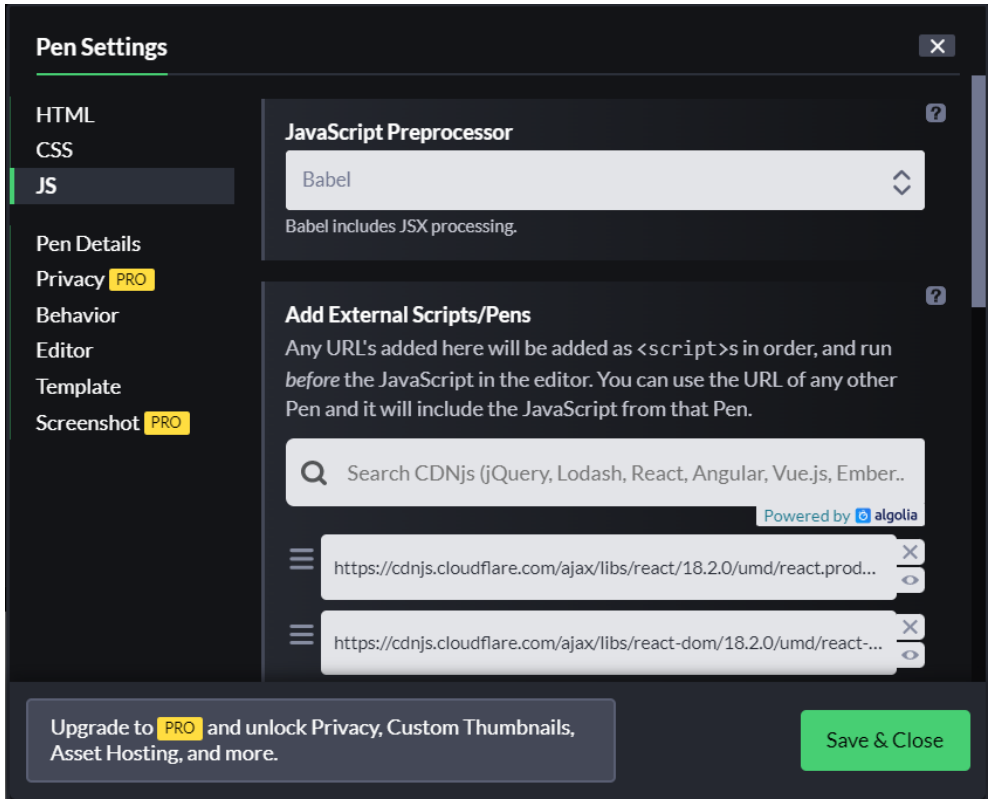
- Extensión de **Chrome** que ayuda a depurar



- Una vez instalada se encuentra una nueva pestaña en **DevTools**:

Entorno para pruebas: Codepen.io

- Editamos los **settings** de Codepen:
  - **Babel** como preprocesador de JS
  - Añadimos **React** y **React-dom** (en ese orden) a los imports



EJERCICIO PROPUESTO II: Hola Mundo con codepen.io

Hola mundo

- Si acabas prueba las extensiones y configuraciones de las siguientes diapositivas

Otras extensiones para VSCode

- **Material icons**: cambia los iconos de Visual Studio
- **Prettier-code formatter:** configurarlo la primera vez que se da formato a un código
- **Prisma**: corregir errores de sintaxis
- **Full stack react**: autocompletado de código
- **DotENV** – ayuda a resaltar variables de entorno
- **CSS Peek** – indica a qué hace referencia cada estilo
- **Autoclose tag, autorename tag**…

Otras configuraciones en VS

- F1: Abre **settings.json**
  - Añadir `"editor.wordWrap":"on"` (para que las líneas no sean "eternas")
  - Añadir `"emmet.includeLanguages":{"javascript": "javascriptreact" }` (permite escribir html en el archivo js de forma ágil
    - Ejemplo: ul>li*5{nombre$}

¿Cómo vamos según la escala Kim K?