Tema 1

Introducción a React JS

Introducción a React JS

- React es una biblioteca Javascript de código abierto para la creación de interfaces de usuario web con el objetivo de alcanzar el desarrollo de **aplicaciones en una sola página**.

- Uno de los objetivos principales de React es el de facilitar la construcción de aplicaciones cuyos datos se encuentren en constante actualización.

- Los componentes dentro de la librería son de uso sencillo y fáciles de combinar con otra serie de bibliotecas que abarquen otros ámbitos diferentes (Angular, Ruby on Rails…).

- React está siendo usado actualmente en las páginas principales de Feedly, Airbnb, Imgur…

Introducción a React JS

- Se trata de una librería de software libre que proviene de Facebook. Por su sencillez y gran rendimiento una gran comunidad de desarrolladores ya lo están utilizando actualmente.

- Nació como una necesidad propia dentro del desarrollo de la conocida red social. Necesitaban mejorar la velocidad entre la obtención de los datos y la renderización de los mismos.

- Está totalmente probada en Facebook y en Instagram, empresa propiedad de los primeros.

Características de React JS

- Una de las características más destacadas de React JS es su **velocidad en el renderizado de vistas.** Hace que los cambios dentro de nuestras interfaces se apliquen de manera muy rápida y sin penalizar la carga general de la aplicación.

- Esto se logra gracias a la generación por parte de React de un **DOM Virtual** para cada componente generado.

- Es capaz de detectar qué elementos se necesita modificar sin necesidad de recargar todo el contenido de nuestra página.

- En el ámbito de las aplicaciones web, es el proceso de renderizado el que más tiempo requiere. React evita este costoso proceso y por eso hace que sea tan rápido.

Características de React JS

- **React nos obliga a pensar en componentes** a la hora de definir nuestros interfaces.

- La organización a través de componentes es una tendencia cada vez más extendida en el desarrollo de aplicaciones *frontend.* Este tipo de estructuras nos permiten organizar mejor cada uno de los bloques de nuestras aplicaciones y facilita la reutilización de los mismos.

- Cada uno de los componentes que desarrollemos encapsularán su comportamiento, la vista a renderizar y su estado. Un componente bien desarrollado tendrá las mínimas dependencias con aquellos componentes que formen parte de su entorno.

- El objetivo es desarrollar componentes que resuelvan pequeños problemas. Conectando diferentes componentes podremos llegar a resolver problemas mayores.

Características de React JS

- Con React seguimos un **desarrollo declarativo**. Nuestras aplicación posee un estado y los diferentes componentes que forman parte de ella reaccionan ante los cambios en ese estado.

- Los componentes poseen una funcionalidad y cuando alguna de sus propiedades se modifica, ellos producen un cambio.

- Podemos usar los componentes tantas veces como queramos dentro de nuestras aplicaciones, definiendo su estado y las propiedades que los identifican.

- Cada utilización de un componente en concreto será diferente ya que los valores que lo definen serán diferentes.

Características de React JS

- Una de las características de los componentes que vamos a desarrollar con ReactJS es que se pueden anidar, formando así estructuras más complejas.

- Los componentes de orden superior **propagan datos a los componentes de orden inferior**.

- Los componentes de orden inferior trabajan con estos datos y, en caso de ser necesario, pueden propagar diferentes eventos para que sean reconocidos por los componentes de orden superior.

- Este patrón de actuación nos permite tener una comunicación unidireccional entre los diferentes componentes de nuestra aplicación.

Características de React JS

- React admite **isomorfismo.** Esto implica que con un mismo código, se puede renderizar HTML tanto en el servidor como en el cliente.

- Las aplicaciones desarrolladas en Javascript suelen recibir los datos del servidor y con estos datos renderizan el HTML que vamos a ver en el cliente.

- Esta sería la manera más óptima para mostrar estos datos, porque nos permite mantener aislados los desarrollos en la parte de cliente y en la parte de servidor.

- Por otro lado, es un aspecto muy negativo de cara al posicionamiento de nuestra aplicación de cara a los buscadores de contenido como Google. En cuanto a posicionamiento se refiere, el cuerpo de la página no tendría contenido.

Características de React JS

- En el caso de React, al permitir **isomorfismo**, podemos renderizar nuestro contenido tanto en el cliente como en el servidor.

- Con esta medida, con la misma base de código se le puede entregar el HTML renderizado a los buscadores para que mejore nuestro posicionamiento.

- Todo esto se consigue gracias a NodeJS.

Características de React JS

- ReactJS aporta una serie de posibilidades muy interesantes en comparación con otra serie de librerías Javascript parecidas como JQuery.

- Al tener las vistas asociadas a los datos, no necesitamos escribir código extra cuando los datos cambian para visualizar dichos cambios, React lo hace automáticamente.

- Permite a su vez una arquitectura de desarrollo más avanzada, como la encapsulación del código en componentes.

- Con el uso de ReactJS tendríamos todas las funcionalidad que nos ofrece jQuery cubiertas. Las dos librerías pueden convivir sin problemas, pero no debería ser necesario.

Ecosistema de React

- ReactJS por si mismo es una librería y no alcanza todas las funcionalidades que puede abarcar un framework completo.

- Sin embargo, existen una serie de herramientas, aplicaciones y librerías extra que nos permiten equiparar el uso de React con cualquiera de los framework existentes.

- Aparte, el desarrollo de componentes dispone de una comunidad muy activa que nos permite encontrar desarrollos muy variados. Por tanto, es muy interesante, antes de desarrollar nuestros propios componentes, investigar si algún desarrollador ya ha creado algo parecido.

- Una de las herramientas más interesantes es **React Native**, la cual nos permite llevar una aplicación escrita con Javascript y React como aplicación nativa para dispositivos iOS o Android.

Instalación

Instalación

- Para comenzar nuestro trabajo con ReactJS vamos a instalar la herramienta **Créate React App,** la cual nos permite generar la estructura completa necesaria para la generación de aplicaciones con ReactJS.

- Para su instalación, necesitamos trabajar con **npm**, el gestor de paquetes de NodeJS.

- Podemos descargarlo e instalarlo junto a NodeJS desde el siguiente enlace:

https://nodejs.org/en/download/

Instalación

- Instalamos la herramienta **Create React App** con el siguiente comando:

npm install -g create-react-app

- Una vez instalado de manera global, podemos generar nuestras aplicaciones con el comando:

create-react-app test_app

Instalación

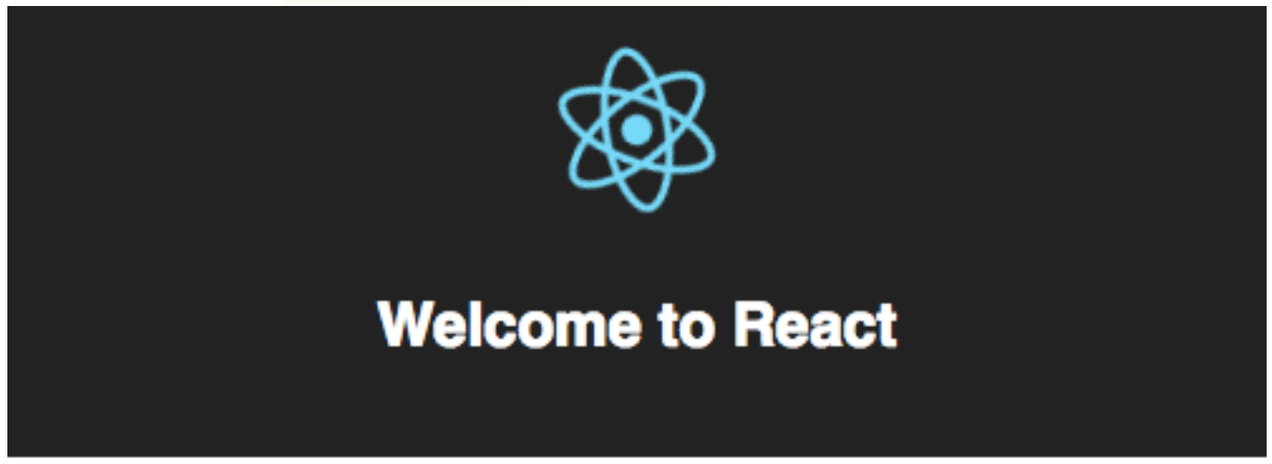
- Con el comando anterior cargamos los archivos de un proyecto vacío y todas las dependencias necesarias para poder arrancar el proyecto ReactJS.

- Para arrancar la aplicación tenemos que acceder al directorio generado y lanzar el el siguiente comando.

```
cd test_app
npm start
```

Instalación

- Si todo ha ido correctamente, se abrirá el navegador automáticamente con la aplicación mostrando una imagen de bienvenida.



- Aparte, en la consola obtendremos las instrucciones para poder acceder a dicha aplicación.

Estructura de App

Estructura

- La estructura de nuestra aplicación es bastante sencilla. Estos son los directorios existentes:

- **node_modules:** directorio con las dependencias ppm del proyecto.

- **public:** se trata de la raíz de nuestro servidor, donde podremos encontrar el archivo principal (index.html) y el icono de la aplicación (favicon.ico).

- **src:** se trata del directorio donde encontramos nuestro proyecto, los componentes propios de React.

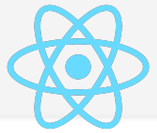
- **README.md:** es el fichero de información sobre el proyecto generado por el gestor de aplicaciones de React.

- **package.json:** enumera las dependencias de npm, tanto para desarrollo como para producción.

- **.gitignore:** el archivo para decirle a git qué elemento no queremos controlar en el repositorio de versiones.

Estructura

- Dentro del directorio **src** tenemos el código propio de nuestra aplicación.
- Encontramos el fichero raíz de nuestra aplicación (index.js), acompañado de nuestro primer componente (App.js).
- Aparte, para cada uno de los componentes, encontramos los ficheros que les complementan, como por ejemplo, los ficheros que representan las imágenes a mostrar o los estilos que vamos a aplicar en cada caso.
- Más adelante entraremos a analizar cada uno de los ficheros en profundidad.



Introducción a JSX

JSX

- Si analizamos el contenido del fichero App.js generado en los pasos anteriores, podemos ver como la mayoría del código está formado por una serie de etiquetas “HTML”.

- En realidad se trata de **JSX**, una extensión sintáctica de Javascript.

- Se trata del lenguaje empleado por React para construir cómo vamos a visualizar la interfaz de usuario.

- Tiene cierto parecido con algunos lenguajes populares para la gestión de plantillas pero tiene la ventaja de contener toda la potencia de Javascript.

- El resultado del uso de JSX en nuestras plantillas es la creación de *elementos React* y por lo tanto, el aprovechamiento de todas las características que nos ofrece el trabajo con el DOM virtual de esta plataforma.

JSX - Expresiones Javascript

- Podemos incluir cualquier tipo de expresión Javascript en JSX siempre y cuando las delimitemos con llaves { }.

- Por ejemplo, podemos ejecutar una función definida dentro de nuestro componente, la cual recoge sus datos de una constante, definida dentro del mismo componente.

```
function nombre_completo(usuario){
        return usuario.nombre + ' ' + usuario.apellidos;
}

const usuario = {
        nombre: 'Pepe',
        apellidos: 'García'
}

const elemento = (
        <h1>
                Hola, {nombre_completo(usuario)}
        </h1>
);

ReactDOM.render(
  elemento,
  document.getElementById('root')
);
```

JSX - Expresiones Javascript

- En el ejemplo anterior se ha separado el código JSX en varias líneas, pero no es obligatorio, simplemente mejora la legibilidad del código.

- Es recomendable en estos casos, incluir el código JSX entre paréntesis para que quede englobado como un único elemento dentro del componente donde lo estamos utilizando.

```
const elemento = (<h1>Hola, {nombre_completo(usuario)}!</h1>);
```

JSX - Expresiones Javascript

- Después de pasar por la compilación, nuestro código JSX se convierte en Javascript.

- Por lo tanto, podemos utilizar cualquier tipo de expresión JSX mezclada con estructuras condicionales, bucles iterativos, como argumentos de funciones o como respuesta de estas.

```
function saludo(usuario){
    if (usuario){
        return <h1>Hola {nombre_completo(usuario)}!</h1>;
    }else{
        return <h1>Hola persona desconocida</h1>
    }
}
```

JSX - Atributos

- Para definir atributos literales para las diferentes directivas utilizadas dentro de nuestro código JSX, podemos delimitarlos a través de comillas dobles.

```
const element = <div tabIndex="0"></div>;
```

- De la misma manera que en los ejemplos anteriores, podemos usar expresiones Javascript para definir dichos atributos.

- **No debemos usar comillas para delimitar dichos atributos** si se generan a través de este tipo de expresiones.

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

JSX - Anidando

- Si la etiqueta definida dentro de nuestro código JSX no tiene ningún tipo de contenido, debe cerrarse inmediatamente para evitar errores.

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

- Todos los elementos definidos dentro de una plantilla JSX pueden contener otra serie de elementos anidados.

```
const elemento = (
  <div>
    <h1>¡Hola Mundo!</h1>
    <h2>Bienvenido a mi aplicación.</h2>
  </div>
);
```

JSX

- Sintácticamente JSX se encuentra más cercano a Javascript que a HTML, por lo que utiliza la nomenclatura *camelCase* para la definición de los diferentes atributos en las etiquetas que estemos implementando.

- Por ejemplo, en HTML disponemos del atributo **class** para definir la clase css que aplicamos a cada elemento. Dicho atributo en JSX sería **className.**

```
const elemento = (
  <div>
    <h1 className="cabecera">¡Hola Mundo!</h1>
    <h2>Bienvenido a mi aplicación.</h2>
  </div>
);
```

JSX

- El compilador, transforma el código definido en JSX a una llamada al método **React.createElement()**.

- Por lo tanto, los dos ejemplos siguientes, son equivalentes:

```
const cabecera = (
  <h1 className="cabecera">
    ¡Hola, mundo!
  </h1>
);

const cabecera = React.createElement(
    'h1',
    { className: 'cabecera' },
    '¡Hola, mundo!'
);
```