**Intro a React**

¡Bienvenidos a la primera clase! A lo largo de esta materia, nos adentraremos completamente en el uso de React, la librería creada por Facebook para manejar interfaces de usuario. Esta librería es una de las más usadas en el desarrollo de aplicaciones del lado del front end.

Veremos cómo crear una aplicación con esta tecnología, trabajando con componentes, entendiendo su ciclo de vida y cómo enlazarlos dentro de la aplicación.

**¿Qué es React?**

React es una biblioteca de JavaScript declarativa y flexible, que se emplea para crear interfaces de usuario. Es de código abierto y es gratuita, y está mantenida por Facebook y una comunidad de desarrolladores y empresas individuales.

React se puede utilizar para desarrollar aplicaciones móviles o de una sola página conocidas como SPA (por Single Page Applications).

Con React las interfaces de usuario se componen a partir de piezas de código más chicas y aisladas que llamaremos componentes. A partir de dichas piezas, podremos componer interfaces de usuario grandes y complejas que funcionarán como un solo sistema.

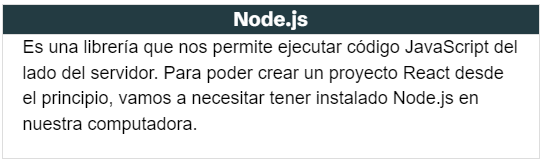
Entre muchas de sus ventajas, su implementación permite que la carga de la aplicación sea más rápida y competente. ¿Cómo logra esto? ¡Descubrámoslo en el siguiente video!

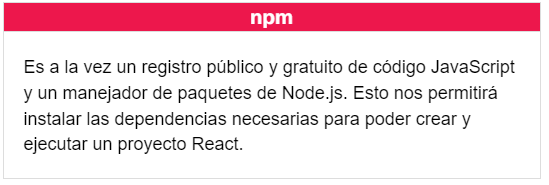
**Set de herramientas para trabajar con React**

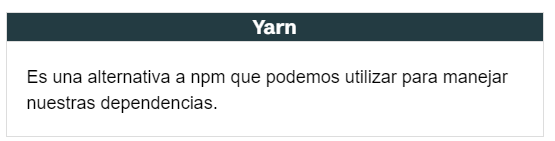
Para poder trabajar en un proyecto React, necesitaremos hacer uso de algunas herramientas que seguramente ya conocen.

Dado que existen diferentes formas de utilizar React en un proyecto, puede que en alguno de los casos no todas las herramientas que mencionaremos sean necesarias. De todas formas, tomando como referencia los estándares de la industria en el uso de React, podemos mencionar las siguientes herramientas:

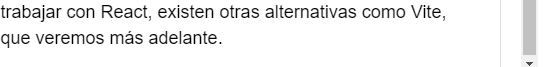


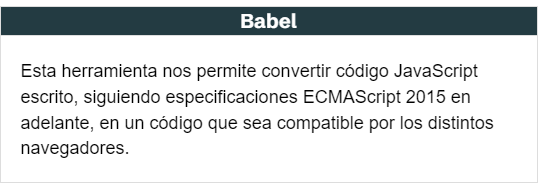












**Agregar React en nuestros proyectos**

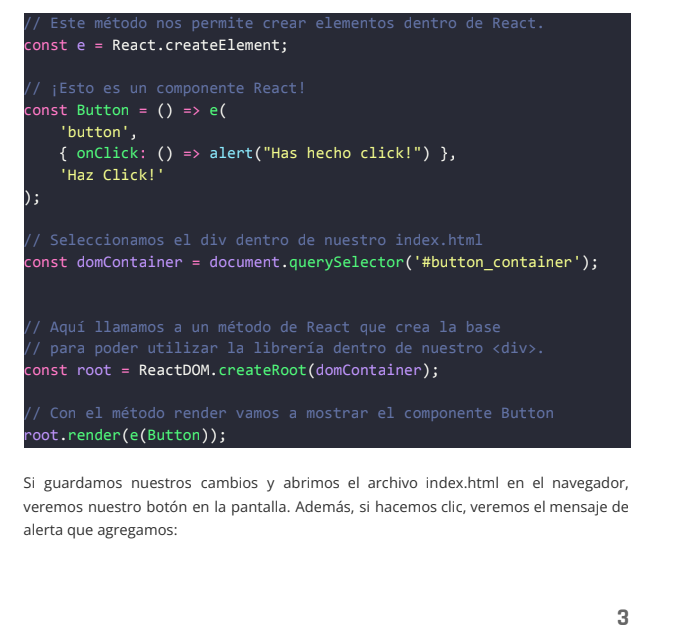
Si bien en la actualidad es normal encontrarse con proyectos construidos 100% en React, también puede darse el caso de que deseemos agregar gradualmente esta librería en un proyecto construido con HTML, CSS y JavaScript. Por suerte, React también nos da esa posibilidad.

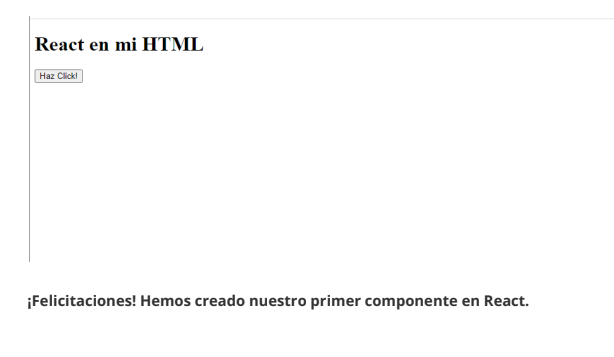












**¿Qué aprendimos hoy?**

En esta primera clase, hicimos una introducción a React.js. Vimos que la librería se basa en la reutilización de componentes, partiendo de los más básicos hasta llegar a construir interfaces de usuario realmente complejas. Además, aprendimos de qué manera React elige qué parte del DOM actualizar frente a cualquier cambio, lo que permite crear interfaces dinámicas sin sacrificar la performance de la aplicación. Finalmente, aprendimos de qué manera podemos agregar React.js a una aplicación existente, y creamos nuestro primer componente.

Clase 2: Herramientas y primer proyecto

En la clase anterior presentamos React, una librería de JavaScript especializada en escribir interfaces de usuario que nos permite fragmentar nuestras aplicaciones web en componentes reutilizables y con lógica propia. Además, implementamos un potente y eficaz sistema de renderizado mediante un DOM Virtual que permite aplicar cambios solamente a la porción aislada donde se realizó una modificación en nuestra web.

La aparición de React generó un antes y después en el mundo del desarrollo web: se comenzaron a escribir cada vez páginas mucho más poderosas y complejas. Ahora bien, estos cambios no vinieron solos. Además de todas las ventajas que ya nos da React, también nos ofrece la opción de integrar una nueva extensión que nos permitirá eficientizar aún más su escritura.

Te presentamos JSX. Por si la eficacia de React no era suficiente, hoy nos adentraremos en esta nueva extensión que nos permitirá escribir JavaScript y HTML en un mismo archivo.

Además, profundizaremos en la creación de un proyecto desde cero en segundos (si, en segundos) con Vite y Create React App, más la implementación de una devtool pensada específicamente para el desarrollador de React.

¡Vayamos a verlo!

**Integración de JSX en React**:

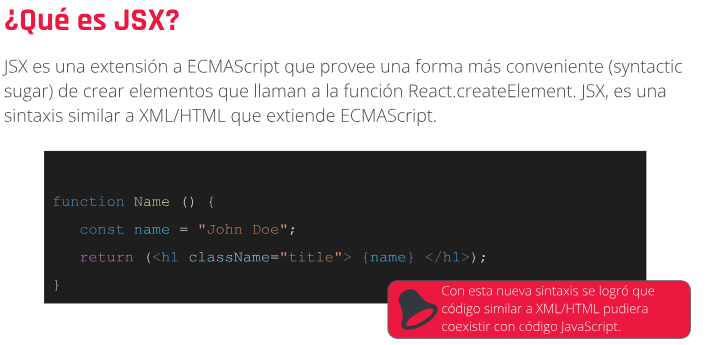
JSX o JavaScript XML es una extensión de la sintaxis de JavaScript que nos permite escribir JavaScript de otra forma diferente a la manera estándar.

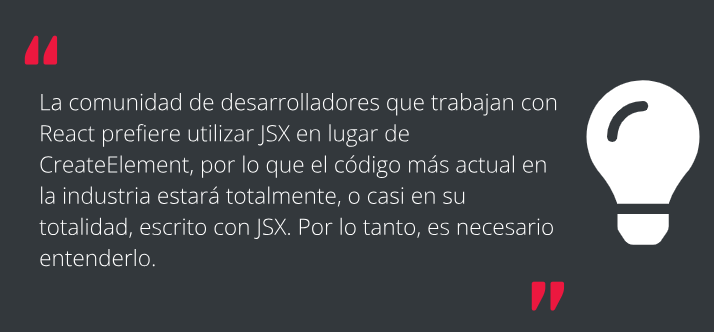
JSX no es un requisito para usar React, pero es muy conveniente usar esta sintaxis porque (y aquí viene su principal ventaja) nos permite escribir HTML directamente dentro de código JavaScript.

React acepta el hecho de que la lógica de renderizado está intrínsecamente unida a la lógica de la interfaz de usuario. Entonces, en vez de separar estos mundos en archivos separados .js y .html, con JSX podemos combinar el marcado (HTML) con la lógica (JavaScript) en un mismo lugar.

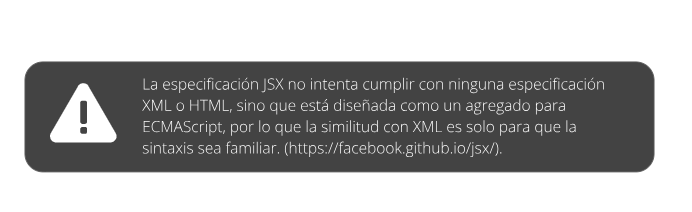
Veamos más acerca de la integración con React en el siguiente recurso.













**¡Seguimos!**

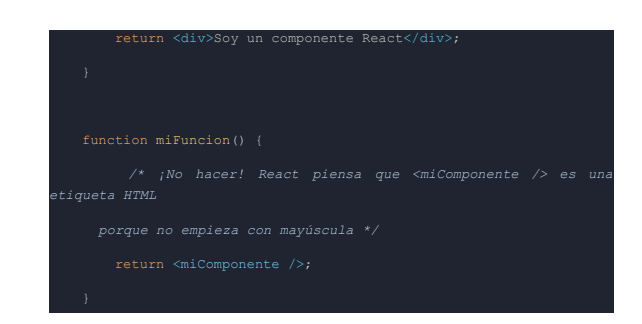
Para adentrarnos en el formato de JSX proponemos ver la declaración de la siguiente variable:

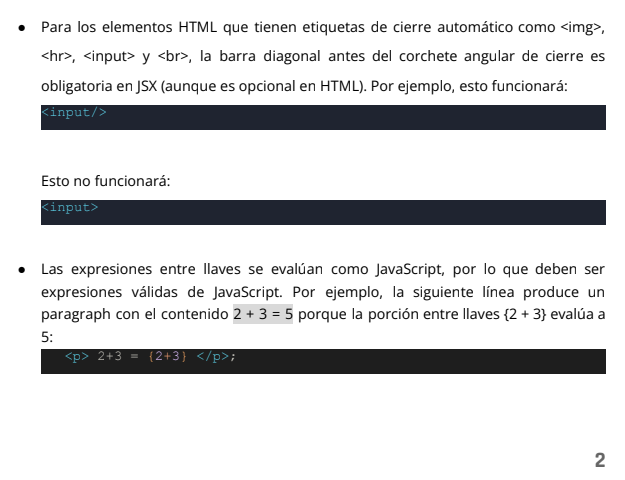
const title = <h1>Hello JSX!</h1>

Esta curiosa sintaxis de etiquetas no es ni un string ni un HTML, sino que es un elemento JSX. ¡Veamos un poco más!

Nos interesa tu opinión











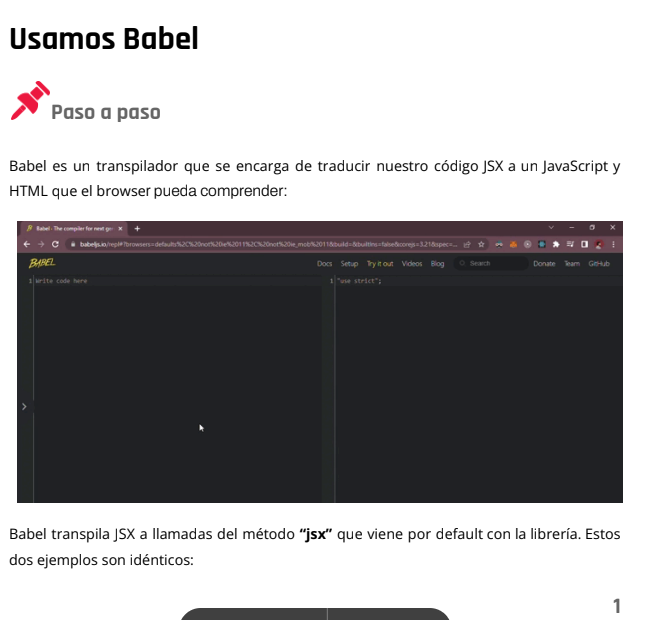


**¿Cómo funciona JSX por detrás?**

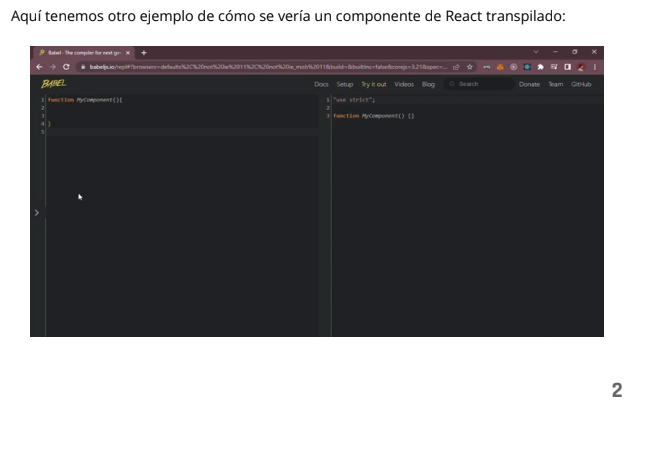
Lo cierto es que suena ilógico pensar que el browser (que solo reconoce HTML, JavaScript y CSS) pueda llegar a comprender esta peculiar sintaxis que le estamos mandando.

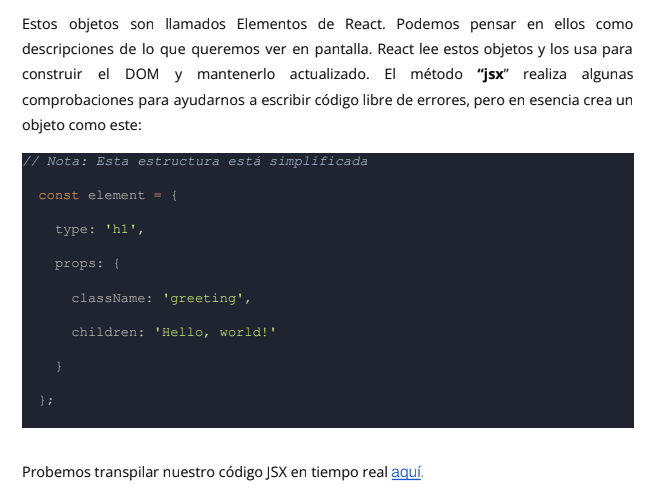
Pues bien, ahí es donde entran los transpiladores como Babel que se encargan de traducir nuestro código JSX a un JavaScript y HTML que pueda comprender el browser.

Veamos cómo funciona…









**Integración de JSX en React**:

Ya introducidos en el mundo de JSX y su facilidad para escribir interfaces con React, estamos listos para comenzar a crear nuestro proyecto. Pero, ¿cómo lo hacemos? ¿A puro mouse y teclado nos ponemos a crear a mano cada uno de los archivos necesarios para que todo funcione?

Bueno, si bien esta es una solución, ¡es una solución de 2013! Hoy en día contamos con herramientas que nos permiten crear toda una plantilla para nuestro proyecto corriendo solo dos comandos. En segundos tendremos un template que integra React listo para comenzar a desarrollar.

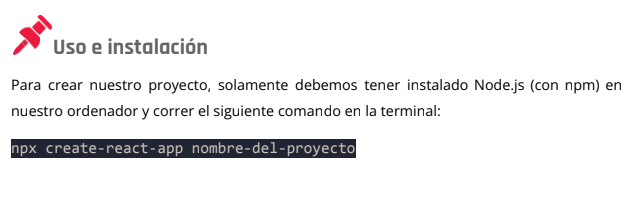
Existen dos grandes opciones para la creación de estos templates:

Por un lado, tenemos a Create React App, creado por Facebook (también creador de React) y con años de antigüedad en el mercado.

Por otro, Vite, una alternativa relativamente “nueva” que vino a innovar y cubrir algunas de las falencias que dejaba CRA (Create React App).

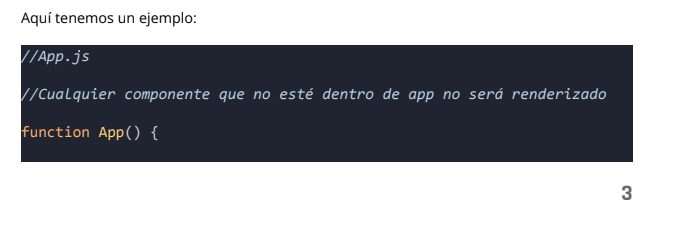
Conozcamos cada herramienta en profundidad.













**¿Con cuál nos quedamos?**

Create-react-app y Vite son muy buenas alternativas para iniciar nuestros proyectos de React de una forma rápida y eficiente. Optar por uno u por otro es opción del desarrollador.

Vite es rápido, pero no viene con aspectos configurados como Jest, por lo que si quisiéramos escribir nuestros tests deberíamos hacer el set-up manualmente nosotros mismos; en cambio, con CRA podríamos empezar a escribirlos al instante.

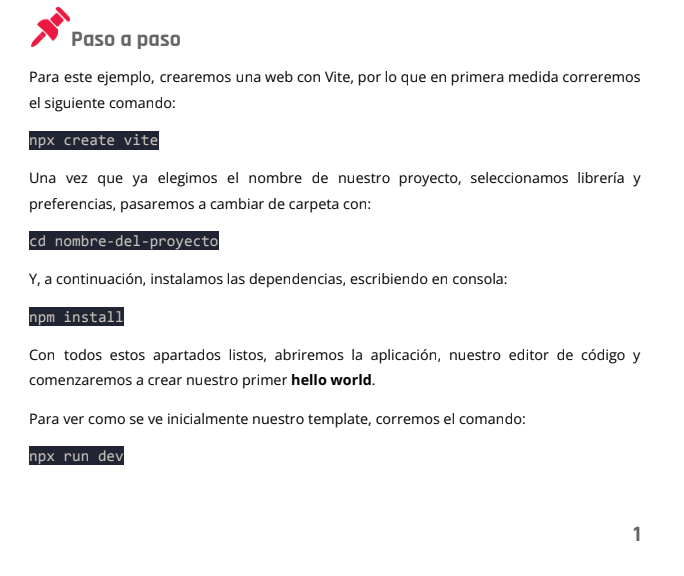
Sin embargo, es una realidad que esbuild (bundler que utiliza Vite por detrás) en la actualidad es muy superior a webpack (utilizado por create-react-app) y muchas otras opciones del mercado. Por lo tanto, a la hora de trabajar con proyectos grandes, su superioridad en cuanto a rendimiento y velocidad es notable y deja atrás a la competencia.

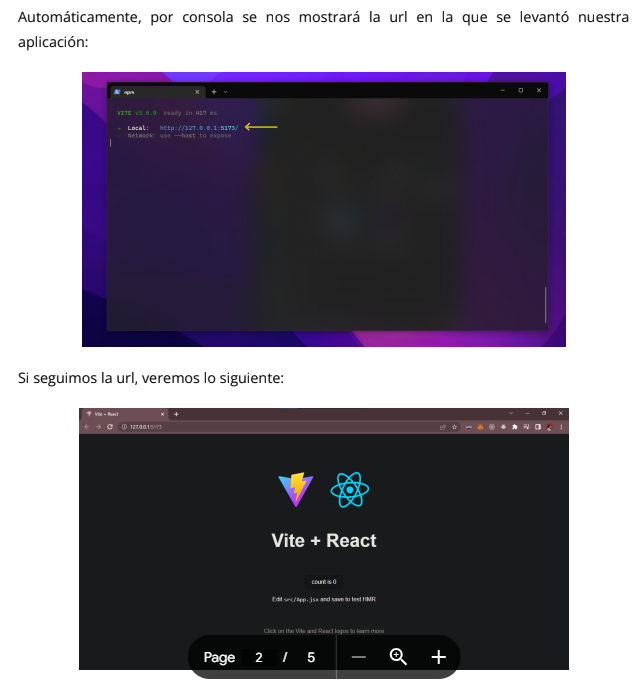
Si la rapidez es nuestra prioridad, Vite es nuestra opción predilecta.

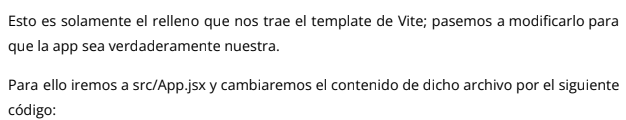
¡Probamos!

Como cierre vamos a crear nuestro primer Hello World! en la web, pero como esta vez será con React haremos un Hello World React! Ya nos introdujimos en el mundo de la sugar sintax con JSX y en cómo crear un proyecto base en pocos segundos y clics. Ahora toca juntar todo lo aprendido y verlo funcionando en nuestra propia aplicación web.

¡Vamos a verlo!

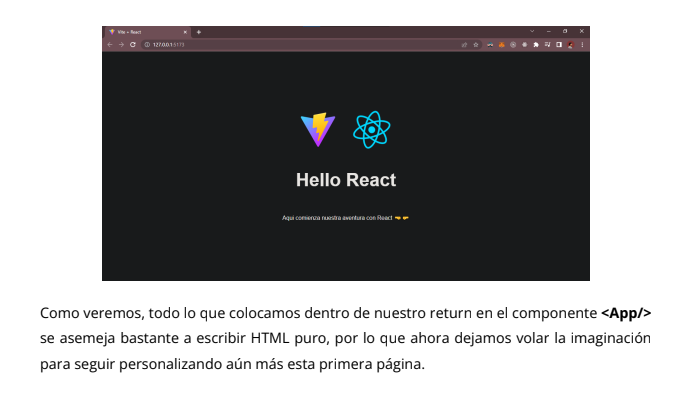


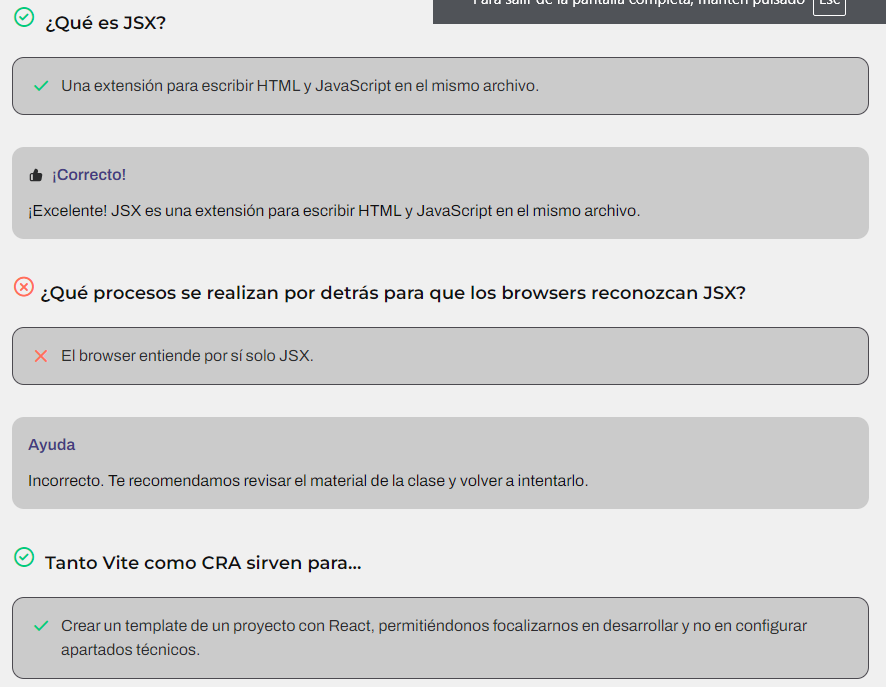


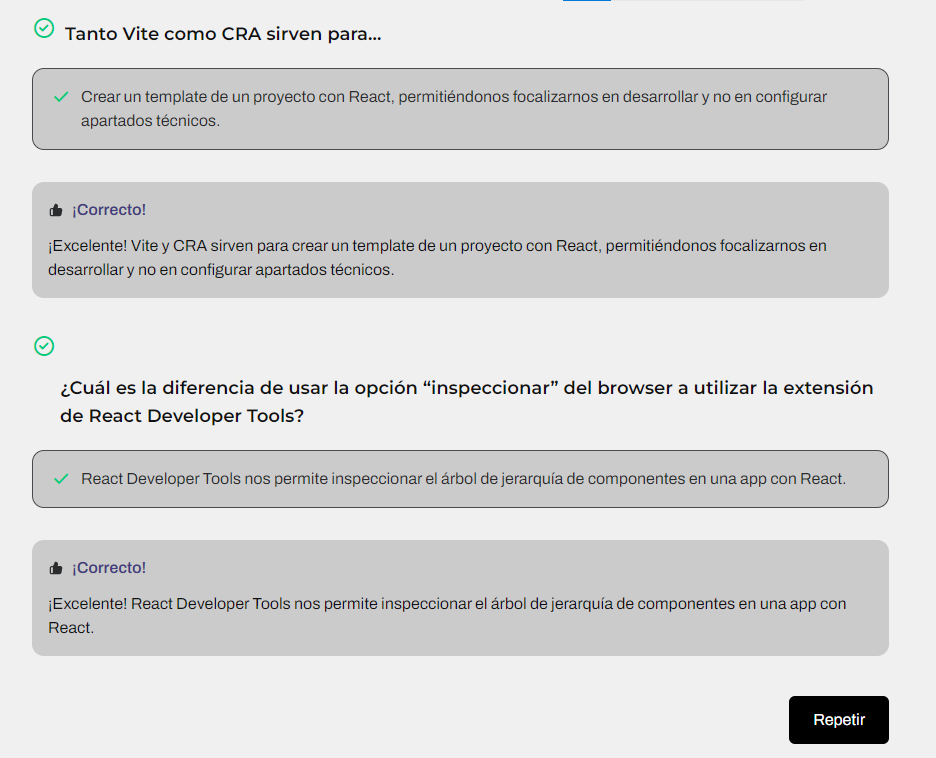












**Configuracion react clase:**

'react/prop-types':'off'

rules: { "react-refresh/only-export-components": [ "warn", { allowConstantExport: true }, ], "react/prop-types": "off", },

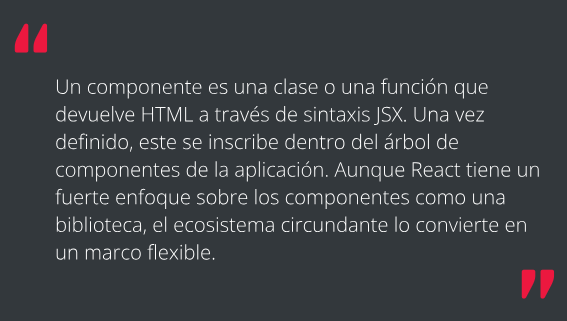
**¿Qué es un componente?**

Un componente es una clase o una función que devuelve un elemento HTML, expresado generalmente a través de sintaxis JSX (estos conceptos los vimos en la clase anterior). Su finalidad principal es la de permitirnos separar la interfaz de usuario en piezas independientes, reutilizables y pensar en cada pieza de forma aislada.

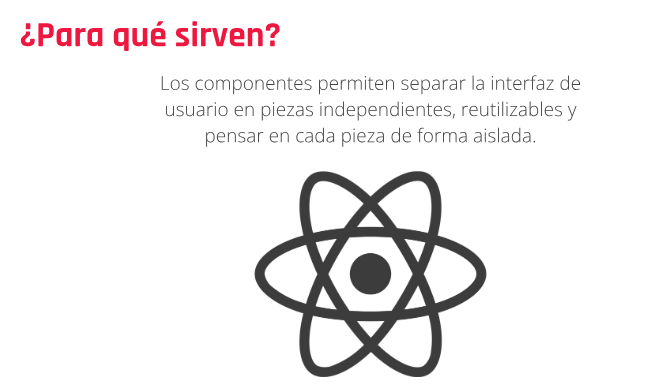
Conozcamos más acerca de los componentes y su uso.





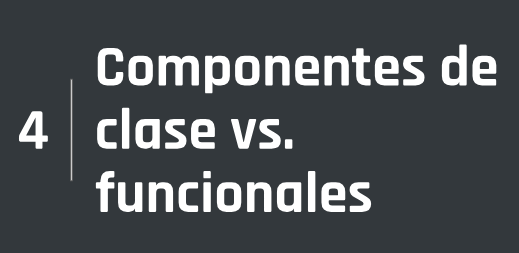




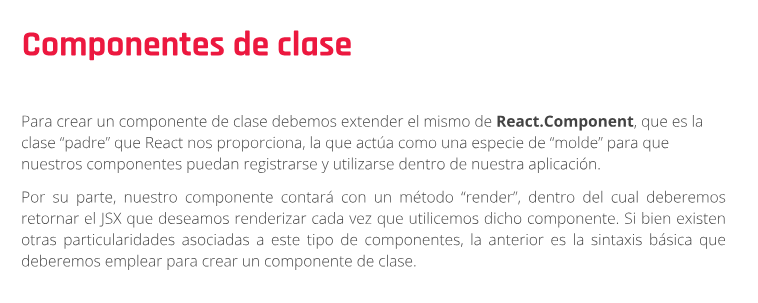














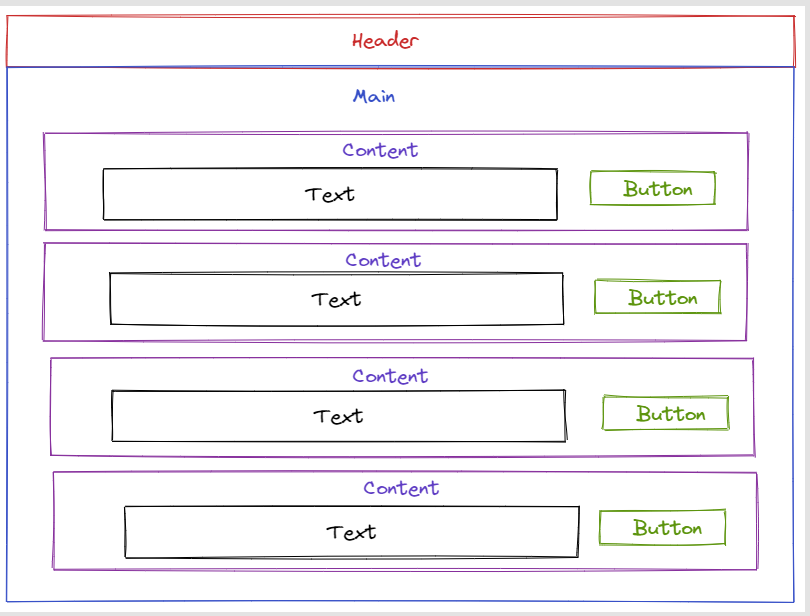
Como consecuencia de lo anterior y gracias al surgimiento de los “hooks”, actualmente el paradigma funcional es el más utilizado para la creación de nuevas aplicaciones en React. Sin embargo, y tal como React lo menciona en su documentación, al día de hoy no existe un plan inmediato para eliminar los componentes de clase, razón por la cual es posible que nos encontremos trabajando en proyectos en donde convivan ambos tipos de componentes. No obstante, dado que en esta materia nos enfocaremos en la creación de proyectos utilizando las últimas versiones de React, nuestro foco de atención estará puesto en la utilización de componentes funcionales y hooks (que veremos más adelante).

**Reutilización de componentes:**

Tal y como venimos viendo, una de las principales finalidades de React es crear interfaces de usuario mediante componentes. Ello supone “partir” nuestra aplicación en diferentes partes, representando cada una de ellas mediante componentes.

Asimismo, dado que otro de los objetivos es procurar la mayor reutilización de componentes posible, podemos particionar una y otra vez cada parte de nuestra interfaz, hasta llegar a un nivel que nos permita aprovechar al máximo esta funcionalidad.

Pensemos en el siguiente esquema:



Allí podremos comenzar nuestra partición dividiendo la aplicación en dos grandes componentes: Header y Main. Pero, como vemos, podríamos luego subdividir el contenido de Main en un componente Content que sea reutilizable. A su vez, Content podría ser nuevamente subdividido en dos componentes (Text y Button) que pueden utilizarse luego.

Vamos a lo importante. El límite a esta subdivisión está dado por las necesidades de nuestra aplicación y la capacidad que tengamos para poder analizar y abstraer componentes que puedan luego ser reutilizados en varias partes de la misma.

La creación y empleo de componentes reutilizables nos plantea la necesidad de analizar la Arquitectura de componentes.

**Componentes puros**

Cuando hablamos de reutilización de componentes, uno de los principales conceptos que debemos tener en cuenta es el de componentes puros. ¿De qué se trata?

Antes de responder esta pregunta, recuperemos el concepto de “función pura”. Una definición generalmente aceptada nos dice que una función es “pura” cuando, dado los mismos argumentos, retorna siempre el mismo resultado. Su principal característica es que no tienen la posibilidad de causar ningún efecto secundario (side effect), tales como podrían ser la modificación de alguna variable global o la alteración del estado de la aplicación.

Si llevamos este concepto a React, podemos decir que un componente será “puro” cuando su finalidad sea exclusivamente presentacional. En otras palabras, un componente puro recibirá determinadas propiedades (como veremos más adelante) y se limitará a retornar un elemento JSX basado en la información recibida, sin generar ningún tipo de efecto secundario. Dadas las mismas propiedades, el contenido del JSX que retornara el componente será idéntico.

Si hablamos de reutilización de componentes, poder crear componentes lo más puro que sea posible nos dará la posibilidad de utilizarlos a lo largo de toda la aplicación, sin necesidad de preocuparnos por los posibles efectos secundarios que los mismos puedan introducir en nuestro código.