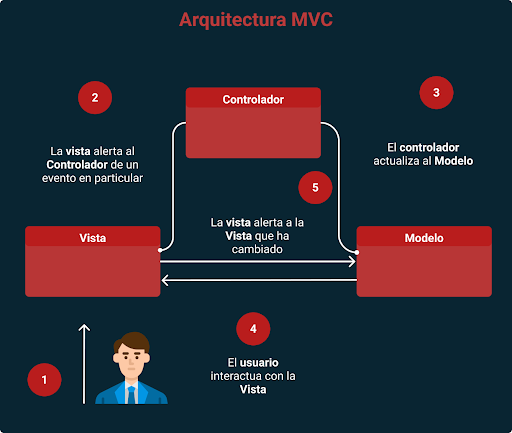
1. Fundamentos Basicos en react
   1. Prework
      1. ¿Cómo vivíamos antes de React.js?
         1. La web tradicionalmente se ha construido en HTML, JavaScript y CSS, separados en distintas carpetas. Eso hace que sea complicado escalar y extraer ciertas partes del código (especialmente JS y CSS) para migrar o reutilizar funcionalidades debido a las dependencias entre estos lenguajes.
         2. La web está construida con 3 lenguajes básicos:
            1. HTML es la estructura e información de la página web y es completamente estático.
            2. CSS es el maquillaje para HTML, con CSS puedes hacer que la información se vea bonita y se adapte a todos los tamaños de pantalla (responsive).
            3. JavaScript a diferencia de HTML y CSS es un lenguaje de programación. Podemos darle vida a la página haciendo que responda a las acciones del usuario
      2. Pero . ¿Por qué no separar HTML CSS y JavaScript y dejarlos juntos?
         1. Un día llegó el equipo de React y propuso meter todo en un solo paquete llamado componente, y de seguro estás pensando "Esto va contra las buenas prácticas" parece que intentan regresar a los años 90 cuando en el mismo HTML metemos el CSS y JavaScript. ¿Por qué lo proponen? Al guardar estos lenguajes en carpetas, estas separando por código y no por funcionalidad.
         2. Recuerda
            1. Ahora que hemos visto porque en React es importante mantener los 3 archivos juntos, vamos a comprender otro concepto y su importancia
      3. ¿Qué es MVC?
         1. Es una arquitectura de software que permite organizar las partes de una aplicación de la siguiente manera:
            1. Modelo: Los datos de la aplicación y el acceso a ellos.
            2. Vista: La interfaz con la que interactúa el usuario con los datos entregados por el modelo.
            3. Controlador: La lógica de negocio, lo que debe hacer la aplicación. El controlador es una especie de puente entre la vista y el modelo.
         2. 
      4. Y a todo esto…. ¿Qué es React?
         1. React también llamada ReactJS es una librería de JavaScript creada por Facebook y publicada como OpenSource, que se diseñó para ayudarnos a desarrollar aplicaciones SPA (Single Page Applications).
         2. El lanzamiento de React.js
            1. En 2013, Pete Hunt presentó React como solo la vista (V) en el modelo MVC en el JSConf EU en una charla titulada: "React: Repensando las buenas prácticas". En esencia React proponía:

Construir componentes, no templetes porque la lógica está inevitablemente unida al marcado del elemento.

Los componentes son bloques cohesionados de piezas de UI reutilizables.

No más two way data binding (enfoque de AngularJS) sino cambiar cada componente en cada actualización.

Virtual DOM para optimizar las actualizaciones del DOM.

JSX para escribir el marcado dentro de JavaScript con una sintaxis sencilla en que incluso los diseñadores web puedan contribuir porque es casi idéntico a HTML.

* + - * 1. A continuación, te dejamos el video del lanzamiento de React.
        2. <https://www.youtube.com/watch?v=x7cQ3mrcKaY&feature=emb_logo>

Meet React

A library for creating user interfaces

Renders your UI and responds to events

Aka the v in mvc

Plays nicely with your stack

Rethinking best practices

Prerequiste

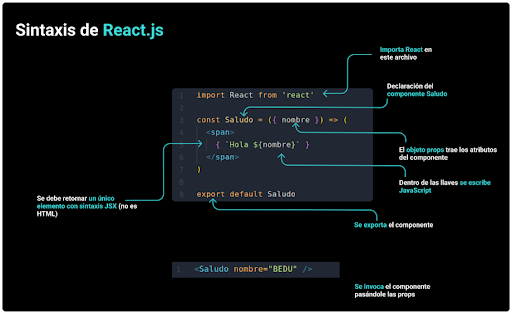
Combinew DOM generation and dispolay logic

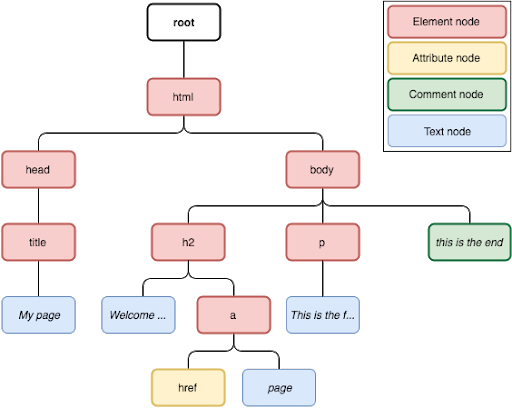
Reacts design

Rerender the whole app on every update

Reacts implementation

Virual Dom and Synthetic events

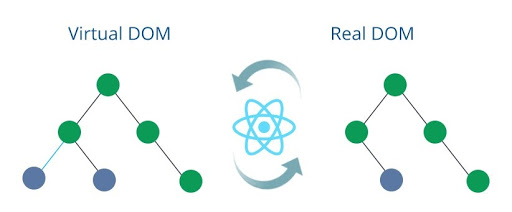
* + - 1. Recuerda
         1. React está basado en un paradigma llamado programación orientada a componentes en el que cada componente es una pieza con la que el usuario puede interactuar. Estas piezas se crean usando una sintaxis llamada JSX
    1. ¿Qué es JSX?
       1. JSX es un lenguaje, el cual, es una extensión al lenguaje JavaScript que permite agregar etiquetas XML en un fragmento de código JavaScript. Esto nos permite definir el código HTML de nuestro componente directamente en el archivo JavaScript.
       2. No es obligatorio usar JSX, se pueden implementar usando código JavaScript regular, con la desventaja que el código resultante es más difícil de entender para los desarrolladores.
       3. Debido a que los navegadores web no entienden el código escrito en JSX, se hace necesario usar un traductor para convertirlo a código JavaScript regular y este traductor es Babel , un compilador para la nueva generación de JavaScript que precisamente puede realizar dicha traducción.
       4. En resumen, es una forma de escribir HTML en JavaScript con una sintaxis muy parecida a HTML que luego es transformada en la compilación. Cuando se presentó fue amada y odiada por igual. ¿HTML en JavaScript? ¡Qué abominación!😅, tranquilo no todo está perdido.
       5. Sin embargo, la comunidad terminó aceptando que JSX les ahorraba mucho tiempo en la creación de componentes y hacía el desarrollo mucho más natural.
          1. let element = <h1>Hello World! </h1>
       6. En cambio, si no tenemos JSX, el proceso es mucho más lento y menos legible:
          1. let element = React.createElement(
          2. 'h1',
          3. null,
          4. 'Hello World!')
       7. Además, JSX permite escribir expresiones más complejas:
          1. let item = {
          2. name: "Cheese",
          3. price: 5
          4. }
          5. let element = <p>{item.name} : ${item.price} </p>.
       8. Estos componentes son reutilizables y se combinan para crear componentes mayores hasta configurar una web completa. Esta es la forma de tener HTML con toda la funcionalidad de JavaScript y el estilo gráfico de CSS centralizado y listo para ser abstraído y usado en cualquier otro proyecto. React aporta una serie de claras ventajas frente a la forma clásica de realizar una web, sus facilidades para el desarrollo unido al rendimiento, la flexibilidad y organización del código.
    2. ¿Qué aceptación tiene en el mercado?
       1. La última encuesta de StackOverflow nos revela que React es el frame Work más querido y más deseado en este último año, además de ser ya el segundo más usado solo por detrás de jQuery. Esto está fuertemente ligado a que JavaScript vuelve a ser el lenguaje más popular por séptimo año consecutivo, cosa que marca una tendencia claramente ascendente.
       2. No es casualidad que empresas como Netflix, Airbnb, Dropbox, PayPal, Tesla, WhatsApp, Uber, etc. hayan apostado por esta tecnología, la inversión que ha hecho Facebook junto al gran número de desarrolladores que han contribuido a esta librería de código abierto hacen que sea una apuesta segura de futuro.
    3. Syntaxis básica de React.js
       1. 
       2. Tranquilo, sé que pueden ser muchos conceptos, pero recuerda que, conociendo JavaScript, React se hace un poco más sencillo. 😉
    4. Virtual DOM
       1. Para empezar... ¿Qué es Real DOM?
          1. Pero, antes de nada, si hay un Virtual DOM tendrá que haber algo real, ¿no? En efecto, el DOM (cuyas siglas son “Document Object Model") es la representación de la interfaz gráfica de nuestra aplicación. Por tanto, cada vez que el estado de la aplicación cambia, lo esperable es que también lo haga dicha interfaz para adaptarse a las modificaciones introducidas. Sin embargo, actualizar el DOM es una tarea costosa en cuanto a rendimiento se refiere, por lo que cuantos más cambios de estado sea necesario reflejar en él, más lento irá nuestra web.
       2. ¿Y por qué es tan lento realizar esos cambios?
          1. El DOM posee una estructura en forma de árbol:



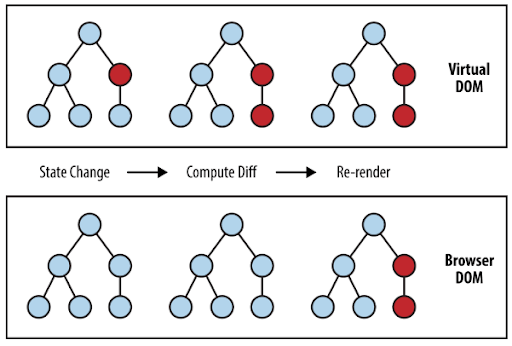
Esto provoca que cada vez que modificamos un elemento dentro de él, todos sus hijos tengan que ser pintados de nuevo (hayan o no cambiado). Y justo este proceso es el que provoca los problemas de rendimiento, ya que de por sí renderizar elementos en una interfaz gráfica es una tarea costosa. Por tanto, cuantos más elementos queden por debajo de nuestro elemento modificado en la estructura del DOM más elementos tendrán que ser vueltos a pintar en la interfaz gráfica. Y es aquí donde entra el concepto del Virtual DOM con el objetivo de conseguir optimizar esta tarea.

* + - * 1. ¿Qué es Virtual DOM?

Virtual DOM es una representación en memoria del DOM real que actúa de intermediario entre el estado de la aplicación y el DOM de la interfaz gráfica que está viendo el usuario.



Puesto que cada elemento es un nodo en el árbol del DOM, cada vez que se produce un cambio en cualquiera de estos elementos (o un nuevo elemento es añadido) se genera un nuevo Virtual DOM con el árbol resultante. Dado que este DOM es virtual, la interfaz gráfica aún no es actualizada, sino que se compara el DOM real con este DOM virtual con el objetivo de calcular la forma más óptima de realizar los cambios (es decir, de renderizar los menos cambios posibles). De este modo se consigue reducir el coste en términos de rendimiento de actualizar el DOM real.



State Change. En este primer paso, se produce un cambio en el estado del nodo de color de rojo, lo cual provoca que se genere en memoria un Virtual DOM con el árbol resultante tras ese cambio.

Compute diff. A continuación se realiza la comparación entre el árbol del Virtual DOM y el del navegador (DOM real) con el fin de detectar los cambios producidos. Cómo ves, el cambio afecta a toda la rama descendiente del nodo cuyo estado cambió.

Re-render. Finalmente, se consolida el cambio en el DOM real y la interfaz gráfica es actualizada de golpe.

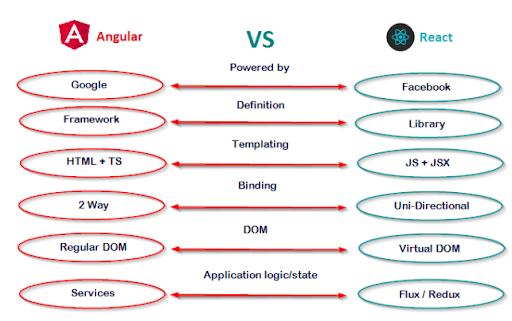
* + - * 1. ¿Por qué usar Virtual DOM y no solo el DOM?

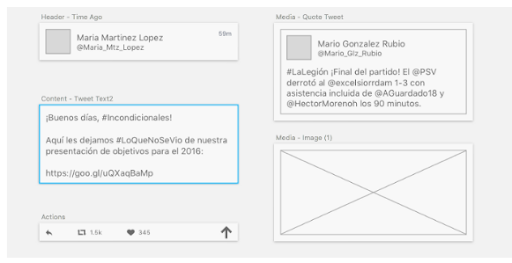
El Virtual DOM es una de las características importantes en la toma de decisión.

Es también una de las diferencias importantes con uno de los frameworks más utilizados como es Angular junto con el binding que en Angular es bidireccional y en React es unidireccional.

¿Esto último qué significa? Que el flujo de información a través de los componentes de React es unidireccional. Cada componente pasa la información a sus componentes hijos y no al revés. Esto hace fácil el seguimiento y razonamiento sobre cómo los datos se mueven a través de la aplicación. Gracias a este detalle, es más sencilla la planeación y detección de errores en aplicaciones complejas.

En la imagen siguiente se puede ver una comparativa entre el frame Work de Google y la librería de Facebook:



* + 1. ¿Qué es la arquitectura basada en componentes y qué ventajas aporta?
       1. Vamos a construir interfaces de usuario (UI) utilizando componentes reutilizables, que poseen y manejan un estado propio. Usamos estos componentes como si fueran bloques de Lego, para construir componentes más complejos y eventualmente una aplicación entera.
          1. 
       2. Recuerda
          1. Llamamos estado(state) a las características propias de un componente. Por ejemplo, cuando tenemos un componente que hace requests a un server, puede tener dos estados posibles, pendiente o finalizado. La lógica de los componentes se escribe en JavaScript (y no utilizando templetes, como es el caso de otras librerías o frameworks de front end), por lo que podemos pasar propiedades (props) de forma simple y mantener el estado fuera del DOM.
       3. ¿Qué es un componente?
          1. “Los componentes nacen, crecen, se renderizan y mueren 🙂 ”.
          2. Un componente es un bloque de código reutilizable, una pieza de UI con contenido, estilos y comportamiento definidos: contiene todo el HTML, CSS y JS necesario para funcionar.
          3. React.js separa tu código en pequeñas partes llamadas Componentes que pueden crearse/definirse como class o como función. Cada componente es como una aplicación de React más pequeña que tiene su propia lógica y tiene un propósito final que es mostrar (Render) algo.
          4. 
          5. Cada elemento de esta página (logo, icono de búsqueda, carrito, icono de menú, texto, botones, etc.) es un bloque de interfaz. La filosofía de React.js es separar el proyecto por piezas de tal manera que el código pueda ser reutilizable, por ejemplo, si yo necesito otro botón, simplemente reutilizó el componente que genera el botón.
       4. Recuerda
          1. Las props son inmutables y siempre se pasan de componentes superiores a componentes inferiores.
          2. // Como: función
          3. función NavBar(props){
          4. return (<nav className="navbar navbar-light bg-light">
          5. <a className="navbar-brand" href="#">Navbar</a>
          6. </nav>);
          7. }
          8. //Como: class
          9. import React from 'react';
          10. class Navbar extends React.Component{
          11. render(){
          12. return (<nav className="navbar navbar-light bg-light">
          13. <a className="navbar-brand" href="#">Navbar</a>
          14. </nav>);
          15. }
          16. }
       5. Recuerda, React es:
          1. 
       6. Recuerda
          1. Recuerda que la documentación oficial va a ser tu mejor amiga, por eso al final de tu Prework podrás ver recursos de los temas vistos anteriormente
    2. Recursos
       1. <https://es.reactjs.org/>
       2. <https://es.reactjs.org/docs/introducing-jsx.html>
       3. <https://es.reactjs.org/docs/rendering-elements.html>
       4. <https://es.reactjs.org/docs/components-and-props.html>
  1. Work
     1. Sesión 01 - Fundamentos de React
        1. Objetivos
           1. Instalación de Node.js
           2. Entender la diferencia entre HTML y JSX.
           3. Ver y aprender cómo React permite la reutilización de componentes.
           4. Maneras de insertar CSS a proyectos con React.
        2. Secciones
           1. Prework

Objetivos:

Entender qué es Node.js

Saber cómo vivíamos antes de React.js

Saber que es react

Conocer es MVC

* + - * 1. ¿Qué es react?

React también llamada ReactJS es una librería de JavaScript creada por Facebook y publicada como OpenSource, que se diseñó para ayudarnos a desarrollar aplicaciones SPA (Single Page Applications).

* + - 1. 🕵 Ejemplos:
         1. Ejemplo 1

OBJETIVO

Comenzar proyectos con buenas prácticas.

Diferenciar HTML y JSX.

Conocer 2 maneras de usar CSS con React.

DESARROLLO

Comenzar nuevo proyecto de React con el comando npx create-react-app ejemplo1.

Seguir las buenas prácticas para empezar un proyecto.

Crear nuestro div principal que nos servirá como wrapper(envoltura).

import React from 'react';

function App() {

return (

<div className="wrapper">

</div>

);

}

export default App;

Ir a index.css para crear el CSS de nuestro techo.

.wrapper {

min-height: 100vh;

display: flex;

justify-content: center;

align-items: center;

}

Crear otro div hijo que nos servirá como marco. Notar primer diferencia en como usamos las clases de CSS.

import React from 'react';

function App() {

return (

<div className="wrapper">

<div className="card frame">

</div>

</div>

);

}

export default App;

Creamos nuevo componente dentro de la carpeta componentes como Header.js.

import React from 'react';

function Header() {

return (

<div className="card-header">

<h1 className="card-header-title header">

Hay 0 tareas

</h1>

</div>

)

};

export default Header;

Creamos un nuevo css en css para almacenar el css de este nuevo componente header.css

.header {

display: inline;

text-align: center;

}

Importamos el componente en App.js y lo usamos. Hay que notar que podemos usar componentes en 1 sola línea.

import React from 'react';

import Header from './Header';

function App() {

return (

<div className="wrapper">

<div className="card frame">

<Header />

</div>

</div>

);

}

export default App;

De manera análoga hacemos lo propio con el componente Form

import React from 'react';

import Header from './Header';

import Form from './Form';

function App() {

return (

<div className="wrapper">

<div className="card frame">

<Header />

<Form />

</div>

</div>

);

}

export default App;

* + - * 1. Reto 01

DESARROLLO

Comenzar un nuevo proyecto con npx create-react-app reto1.

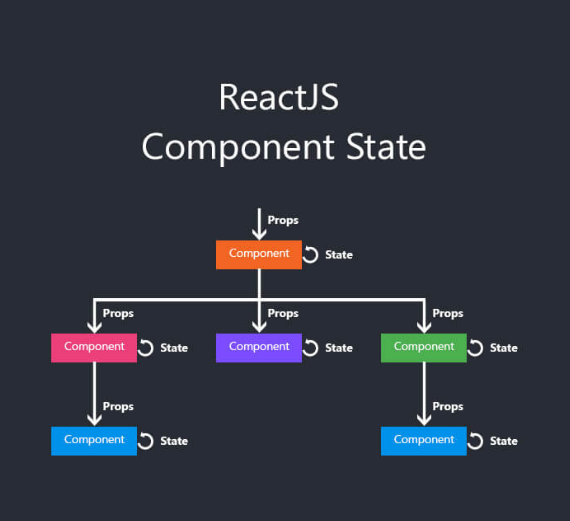
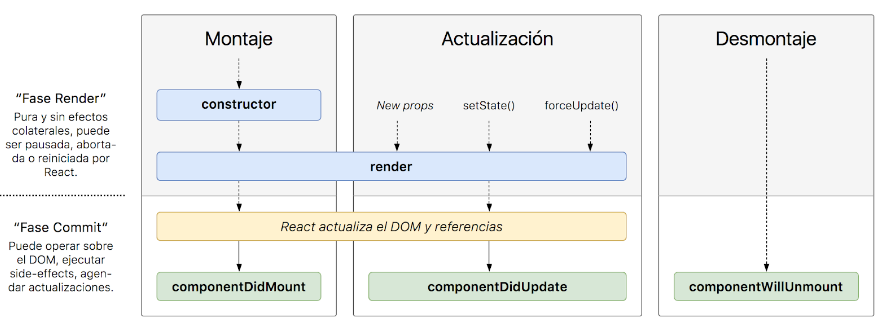
Seguir las buenas prácticas para empezar un proyecto.

Crear los componentes Todo, y TodoList.

Crear el CSS para los componentes en dado caso que sea necesario; puede ser global o por componente.

Si no pudiste resolver el reto, no te preocupes,😉 en la parte superior del repositorio encontrarás los archivos con la solución para tu consulta, pero recuerda lo importante es que lo intentes primero. 🤓

* + - * 1. .
  1. PostWork

1. Estado (state) y Propiedades (props)
   1. Prework
      1. Que son los props
         1. Las propiedades son la manera que tienen los componentes de React de comunicarse entre sí.
         2. Cualquier componente puede invocar a otro, al que podemos definir como su hijo, y pasarle propiedades que conformarán datos que este va a recibir y poder gestionar de manera interna.
         3. Este paso de propiedades es la forma que tienen de comunicarse entre ambos, y es una forma muy parecida a la que nos podemos encontrar en las funciones.
         4. A veces un componente necesita información dinámica para mostrar.
      2. Que es el estado (state)
         1. El estado se sitúa dentro de este componente, y al final influye también en la salida, en forma de vista, que acabará devolviendo dicho componente.
         2. Este estado puede recibir datos de las propiedades, gestionarlos, almacenarlos, etcétera, y acaba formando parte también de este renderizado después del procesamiento.
         3. El componente se puede apoyar en este estado para interacciones, tanto por parte del usuario con la propia vista, como por parte de la lógica interna que hayamos podido definir para dicho componente.
         4. El estado puede ser de dos tipos:
            1. Estado del componente: Cada componente tiene su propio estado que no es compartido con otros, a excepción de sus componentes hijos.
            2. Estado global de la aplicación: La información del estado de todos los componentes, por medio de la cual estos pueden comunicarse.
      3. Que es el this state
         1. Es una variable de class (disponible en toda la clase que usa esto) que necesita declarar e inicializar dentro del método del constructor de clases. React.js volverá a representar el DOM completo cada vez que actualice esa variable en particular.
         2. Sin embargo, hay una trampa. El estado no se puede cambiar, lo que significa que no se puede editar, por lo que debemos anularlo mediante la función this.setState () que recibe el nuevo objeto de estado (que reemplaza al anterior).
         3. Props, componente y state
            1. 
      4. Debería de usar Función o class
         1. ¡Ambos! Depende de cuán complejo sea su componente:
            1. Las funciones son super simples y pequeñas pero muy limitadas, trate de usarlas siempre a menos que esté obligado a usar una Clase.
            2. Las clases son para componentes más grandes que requieren más lógica. Tienen un estado local (this.state) y métodos de ciclo de vida de componentes.
         2. ¡Pero no te preocupes! ¡Puedes cambiar de un tipo de declaración a la otra sin ningún problema! Aquí hay una comparación de ambos tipos de componentes:
      5. Eventos en React
         1. Recuerda
            1. La forma como manejamos eventos en React es muy parecida a la forma en cómo los manejamos sobre el DOM.
         2. A continuación veremos cómo manejamos eventos en JS:
            1. <div onclick="handleClick()">click me</div>
            2. function handleClick() {
            3. alert('clicked');
            4. return false:
            5. }
         3. A continuación veremos cómo manejamos eventos en React:
            1. Para hacer lo mismo en React tenemos que hacer los siguientes cambios:
            2. El nombre del evento tiene que ser camel Casé y no minúscula.
            3. Al evento se le pasa la función y no una cadena de texto.
            4. En react si quieres prevenir un comportamiento por defecto o la propagación de un evento debes hacerlo explícitamente llamando los métodos preventDefault() y stopPropagation() respectivamente.
            5. Y se antepone la palabra on.
         4. Eventos sintéticos
            1. En este caso event es un evento sintético de React, en React todos los manejadores de eventos son instancias de SyntheticEvents.
            2. Los eventos sintéticos son una envoltura de los eventos nativos del navegador, por lo que estos eventos cuentan con la misma interfaz de los eventos nativos, como por ejemplo preventDefault() y stopPropagation(), con la ventaja de que todos estos eventos funcionan idénticamente en la mayoría de los navegadores.
      6. Recursos
         1. <https://es.reactjs.org/docs/conditional-rendering.html>
         2. <https://es.reactjs.org/docs/rendering-elements.html>
         3. <https://es.reactjs.org/docs/rendering-elements.html#react-only-updates-whats-necessary>
   2. Work
   3. Postwork
      1. Sesión 02: Estado (state) y Propiedades (props) > Postwork
         1. OBJETIVO
            1. Aplicar el conocimiento de props en los componentes.
            2. Aplicar el conocimiento de state en los componentes.
         2. DESARROLLO
            1. Hacer que los <Checkmark /> reciban e interactúen como Todo.
            2. Que Checkmark tenga interacción cuando un elemento está completado o no.
2. CICLO DE VIDA, SETSTATE, EVENTOS Y .MAP()
   1. Prework
      1. Ciclo de vida de los componentes
         1. Se llama ciclo de vida al tiempo que pasa desde que un objeto se crea desde el código hasta que se elimina. En un nivel un poco más técnico, podríamos decir que desde que se carga en memoria hasta que se elimina de la memoria. Durante la vida de un componente de React, se ejecutan varios métodos, en función del momento.
         2. A estos métodos se les llama métodos del ciclo de vida. Algunos métodos del ciclo de vida que ya conocemos son el constructor(), que se ejecuta, solo una vez, cuando se crea el componente, y render(), que sabemos que se ejecuta en algún momento después de crearse y cada vez que cambia el estado.
            1. 
         3. Montaje de un componente
            1. El montaje es la primera fase del ciclo de vida de un componente. Es la parte en la que se crea el componente. Sabemos que un componente de React representa un elemento del DOM y lo que contiene. En el momento en que ese elemento se pinta en el DOM, aparece visualmente en la página web, decimos que ese componente está montado. Como ya sabemos que el método render() es el encargado de pintar el componente, podemos decirlo de otra manera: un componente se monta en el momento en que se ejecuta su render() por primera vez.
            2. Sin perder de vista la primera ejecución de render(), que nos servirá de referencia, vamos a ver el resto de métodos de la fase de montaje en orden de ejecución:
            3. constructor(): Este ya lo conocemos. Se ejecuta según se crea el componente por código y se le pasan las props iniciales. Aquí:

Inicializamos el estado.

Enlazamos los event handlers a la instancia con .bind(this).

* + - * 1. render(): Otro viejo amigo. devolvemos lo que se pinta en función de props y state.
        2. componentDidMount(): literalmente, "el componente se ha montado". Este método se ejecuta justo después de que el componente se haya montado (pintado en pantalla).

podemos pedir datos remotos, con fetch(), por ejemplo podemos suscribir el componente, por ejemplo, a un set Interval() u otro código que nos dé datos de manera periódica o de tanto en tanto.

* + 1. Recursos
       1. <https://es.reactjs.org/docs/state-and-lifecycle.html>
       2. <https://es.reactjs.org/docs/handling-events.html>
       3. <https://es.reactjs.org/docs/lists-and-keys.html>
  1. Work
  2. Postwork

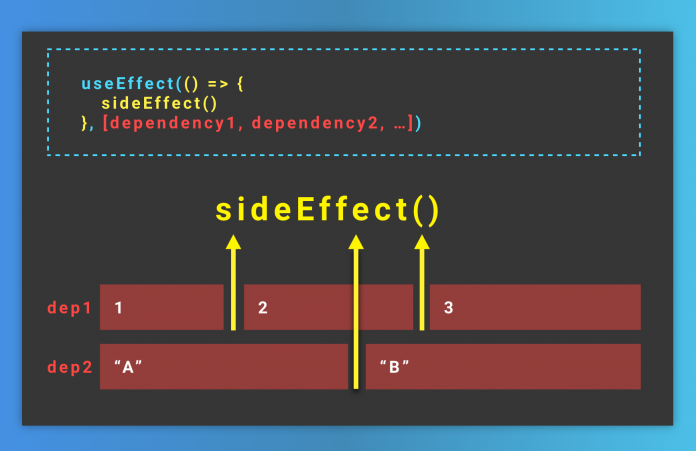
1. Hook and useState
   1. Prework
      1. Hooks y useState
         1. La palabra Hook se traduce como Gancho en español y la razón de que este nuevo api tenga este nombre es como indica la documentación un Hook en React te permite colgarte o si buscáramos una mejor traducción seria algo como integrarte con el manejo de estado y los métodos del ciclo de vida de un componente desde un componente funcional
         2. Hasta el día de hoy los componentes clase poseían habilidades que los componentes funcionales no tenían principalmente la habilidad de conservar un estado y la habilidad de definir acciones a realizar en el ciclo de vida a un componente vía los métodos componentedDidMount, compontndidUpdate entre otros mas
         3. Loos hooks proveen de una API que permite a los componentes funciónales manejar un estado y definir lógica para el ciclo de vida de un componente
      2. Por qué los Hooks
         1. Como explica Sophie Albert en su presentación durante ReacConf 2018 existen 3 problemas que el equipo de React esta buscando solucionar con los Hooks:
            1. Compartir y Ritualizar lógica entre componentes
            2. Reducir el tamaño de componentes muy complejos
            3. Eliminar la confusión que las clases generan.
         2. Estos con problemas identificados por la comunidad en pocas palabras es complicar reciclar lógica entre compo0ntenes algunas componentes puedes volverse verboso y terminar conteniendo miles de líneas de código y las clases no son tan amigables como pensábamos, son complicadas de entender para algunos programadores especialmente por algunas caracterizas que las hacen particulares en JavaScript, y además son complejas para alguna tarea de las computadoras como la optimización de cogido minificacion etc.
         3. Y aunque estos parezcan problemas independientes Dan Abramov Explica que en realidad son problemas que están relacionados entre si y que seguramente dar solución a uno puede significar dar solución a los 3 aquí entran los Hooks
         4. Además de los Hooks de manejos de estado React tiene otros Hooks entre los que destacan
            1. useEffect
            2. useContext
            3. useReducer
      3. El hook de estado: useState
         1. El hook useState es el que nos permite agregarle un estado local a un componente funcional y cambiar ese estado.
         2. Ejemplo
            1. Para que un componente funcional tenga estado propio podemos hacer uso del hook useState.

import React, { useState } from 'react';

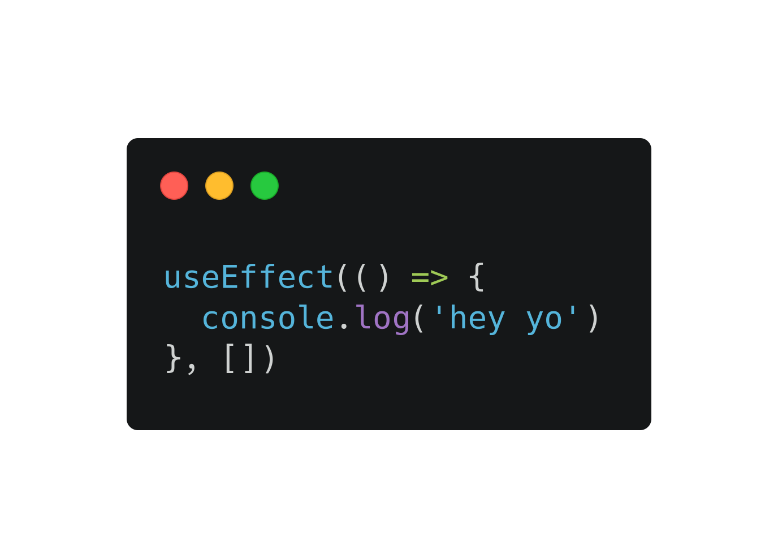
* + - * 1. useState() es una función que crea internamente una variable donde podremos almacenar el estado de nuestro componente. Acepta un valor inicial para esa variable y devuelve un array con dos elementos, el valor de la variable y la función para modificarla. Como el valor devuelto por la función es un array, podemos descomponerlo para acceder a sus elementos de manera individual.

const [count, setCount] = useState(0);

* + - * 1. Puedes añadir tantos useState() como quieras, cada uno de ellos para una variable diferente. La única condición es que se llame desde un nivel superior de código, no en un bloque. Es importante saber que cuando llamamos a la función set de un useState(), se sobrescribe el contenido de la variable.
  1. Work
  2. Postwork

1. HOOKS Y USEEFFECT
   1. Prework
      1. Los hooks
         1. Recuerda
            1. Los Hooks son una API que nos permite usar estado y otras características en un componente funcional.
         2. Los componentes funcionales no podían tener estado, se utilizaban principalmente para:
            1. Componentes simples sin estado.
            2. Estilos reutilizables.
         3. Con la API de Hooks, los componentes funcionales pasan a tener el mismo potencial que los componentes de clases. Y de manera más declarativa y escueta.
         4. Con el hook de React useEffect podemos especificar cuando queremos que un código determinado se ejecute:
            1. Sólo una vez
            2. Sólo cada vez que cambie una (o más) variables en concreto
         5. En lugar de ejecutarse a cada renderizado de nuestro componente
            1. useEffect nos permite ejecutar un efecto secundario cuando se modifica una prop o un estado de useState.
            2. useEffect sustituye a las funciones de ciclo de vida (componentDidMount, componentDidUpdate,componentWillUnmount).
            3. 
      2. Para que es util el useEffect
         1. Los hooks de React son herramientas muy potentes que nos sirven sobre todo para extraer código y poder reutilizarlo de manera muy simple y agradable. El useEffect es la manera que tenemos en React de controlar y decidir cuándo queremos que se ejecute un código concreto.
         2. Ejemplo:
            1. Vamos a introducir un console.log() en un componente que también tendrá un botón, y cada vez que lo apretemos actualizaremos un contador y esto obligará a React a renderizar el componente otra vez.



* + - 1. Recuerda
         1. Cada vez que actualizamos el contador el componente Button se vuelve a renderizar y con ello vuelve a ejecutar el console.log
      2. Pero y ¿si queremos que este console.log sólo se ejecute una vez y al principio?
         1. Exacto, con el useEffect 🤓
      3. Recuerda
         1. Quizás está bien que cada vez que se apriete el botón se ejecute el console.log, pero a medida que vayamos añadiendo complejidad el número de renderizados crecerá, y entonces igualmente necesitaremos forzar que ese console.log no se ejecute siempre sino cuando nosotros queramos.
      4. ¿Cómo controlamos que se ejecuta y cuándo se ejecuta dentro de un componente?
         1. 
         2. Y aquí puedes ver como el console.log sólo se ejecuta una única vez, y aunque el componente se renderize mil veces sólo veremos una orden en la consola.
         3. Esto es así porque hemos encapsulado el código dentro de un useEffect, pero...
    1. Como funciona UseEffect
       1. La función se especifica de la siguiente manera:
          1. 
       2. Tiene dos argumentos
          1. 

El primer argumento es una función, y el código de adentro será el que ejecutemos.

El segundo argumento define cuándo se ejecutará este código, y es un array de dependencias, esto es todo lo que se encuentra dentro de este array, cuando cambie, se volverá a ejecutar el código del useEffect.

* + - 1. Pregunta, ¿cuántas veces se ejecutará este código?
         1. 
         2. Respuesta

Las mismas veces que antes, porque la variable mivariable no cambiará nunca (por lo tanto se ejecuta al principio y nunca más)

Pero y si queremos ¿que se ejecute al apretar otro botón? y la respuesta es la misma que anterior.

Al dar clic en el botón console.log estaremos imprimiendo el console.log, pero si te fijas sólo lo hará la primera vez, las siguientes ya no.

Esto ocurre porque cuando hacemos un setMivariable ('ya no') por segunda vez en realidad no estamos cambiando el estado (cambiarlo por el mismo contenido es lo mismo que no cambiarlo), por lo que no se ejecuta el useEffect.

Por lo tanto y en conclusión, el useEffect es la manera React que tenemos para controlar cuándo se ejecuta un código determinado.

* + 1. Recursos
       1. <https://es.reactjs.org/docs/hooks-effect.html>
       2. <https://es.reactjs.org/docs/hooks-rules.html>
  1. Work
  2. Postwork.