

PROYECTO EJERCICIOS CON REACT

Proyecto de estudio de ejercicios para react utilizando librerías básicas de React

Se mostrara un ejercicio y su solución utilizando lo menos posible internet ni uso de aplicaciones de chatbot de inteligencia artificial . Para practicar React, Javascript, Html y CSS.

- Crear proyecto

```
Windows PowerShell
PS C:\A_CURSOS\2024\Proyecto\Frontend> npx create-react-app react-banca
```

- Subir servidor

```
Windows PowerShell
PS C:\A_CURSOS\2024\Proyecto\Frontend\react-banca> npm start

> react-banca@0.1.0 start
> react-scripts start

(node:18864) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(node:18864) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...

One of your dependencies, babel-preset-react-app, is importing the
"@babel/plugin-proposal-private-property-in-object" package without
declaring it in its dependencies. This is currently working because
"@babel/plugin-proposal-private-property-in-object" is already in your
node_modules folder for unrelated reasons, but it may break at any time.

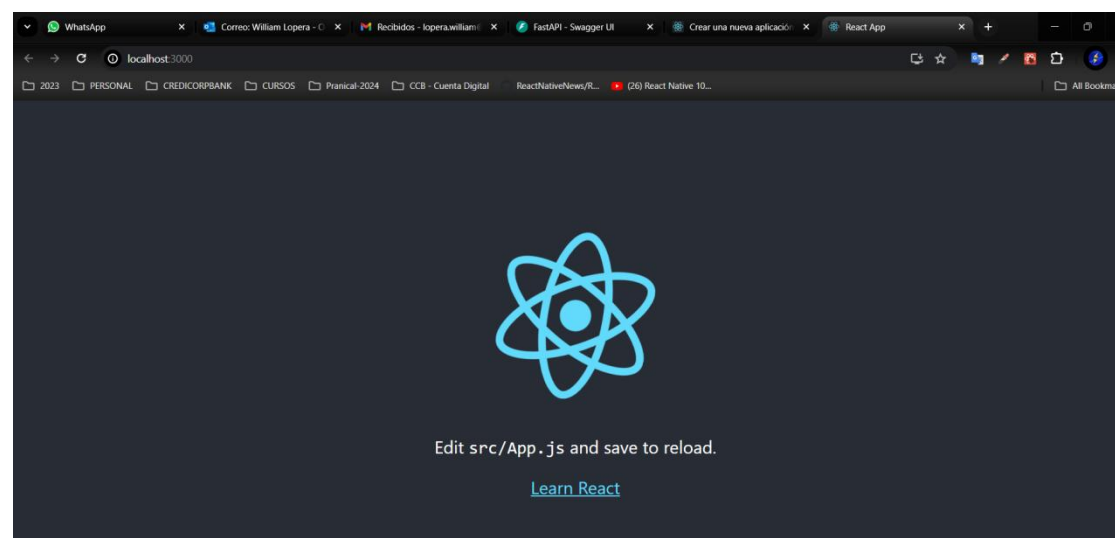
babel-preset-react-app is part of the create-react-app project, which
is not maintained anymore. It is thus unlikely that this bug will
ever be fixed. Add "@babel/plugin-proposal-private-property-in-object" to
your devDependencies to work around this error. This will make this message
go away.
Compiled successfully!

You can now view react-banca in the browser.

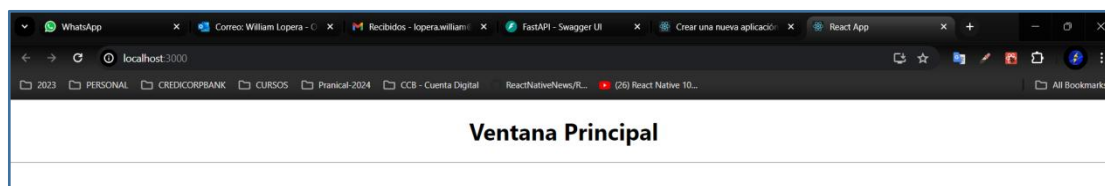
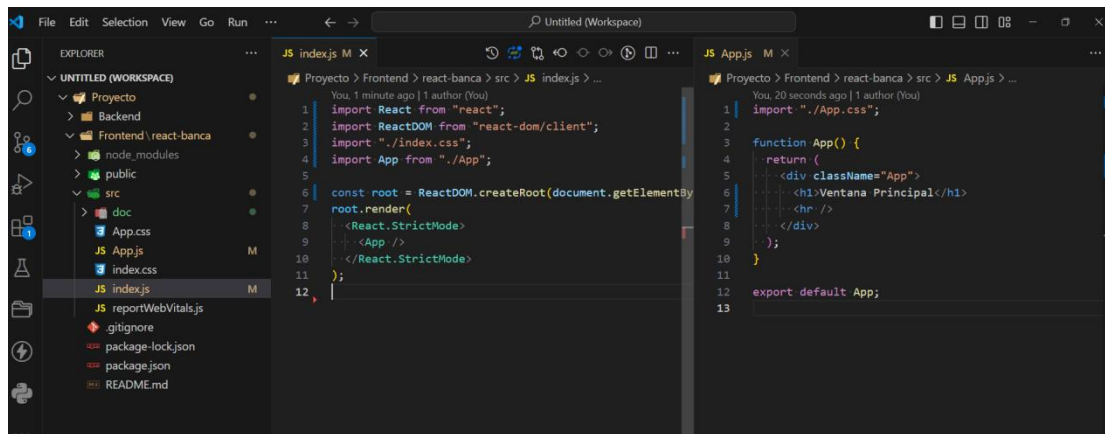
  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.10:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



- *VSCODE: Depurar el código*



Código

Se crearon cuatro componentes importantes:

- *Header: Cabecera de la Pantalla*
- *Footer: Mensajes de información al usuario y del sistema*
- *Menu: Area para agregar valores verticales de Menú (dinámico)*
- *Body: Cuerpo del programa. Este área varia a medida que se seleccionen componentes del menú*

Además se crearon componentes para el manejo de:

- *mensaje: Control de mensajes*
- *HeaderProcess: Control de titulo, descripción y problema a resolver*

Uso de mobx para Tener un almacén común de datos (por ahora mensajes al usuario)

- *Librerías:*

```
"@testing-library/jest-dom": "^5.17.0",
"@testing-library/react": "^13.4.0",
"@testing-library/user-event": "^13.5.0",
"axios": "^1.7.2",
"mobx": "^6.12.4",
"mobx-react": "^9.1.1",
"react": "^18.3.1",
"react-dom": "^18.3.1",
"react-scripts": "5.0.1",
"styled-components": "^6.1.11",
"web-vitals": "^2.1.4"
```

Componentes

Un componente de React es un código pequeño y reutilizable, que es responsable de una parte de la interfaz de usuario de la aplicación. Una aplicación de React es una agregación de componentes. React puede ayudarnos a crear componentes reutilizables. El siguiente diagrama muestra diferentes componentes. Todos los componentes tienen diferentes colores de borde. En React, ensamblamos diferentes componentes para crear una aplicación. Usamos funciones o clases de JavaScript para crear componentes. Si usamos una función, el componente será un componente funcional, pero si usamos una clase, el componente será un componente basado en clases.

Los componentes pueden ser:

Componente funcional / Componente de presentación / Componente sin estado / Componente tonto

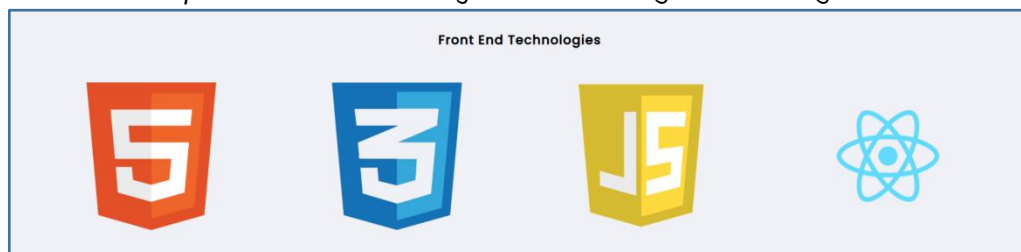
Componente de clase / Componente contenedor / Componente con estado / Componente inteligente

La clasificación de componentes anterior no funciona para la última versión de React, pero es bueno conocer la definición anterior y cómo funcionan las versiones anteriores.

Entonces, cambiemos todo el JSX por componentes. Los componentes en React son funciones o clases de JavaScript que devuelven un JSX. El nombre del componente debe comenzar con mayúscula y, si el nombre consta de dos palabras, debe ser CamelCase (un camello con dos jorobas).

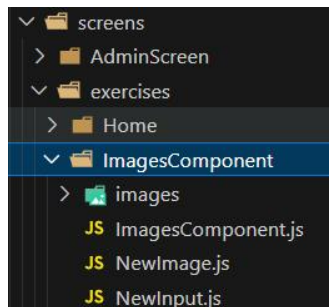
Ejercicio 1

- Cree componentes funcionales y muestre las siguientes imágenes



- *Utilice el componente funcional para crear el siguiente diseño*

- *Suscribir*
- *Regístrese con su dirección de correo electrónico para recibir noticias y actualizaciones.*
- *Tres input (entradas de texto)*
- *Boton de Enviar Subscripción*



```
import React from "react";
import { HeaderProcess } from "../../components/headerProcess/HeaderProcess";
import styled from "styled-components";
import { NewImage } from "./NewImage";
import htmlLogo from "./images/html_logo.png";
import cssLogo from "./images/css_logo.png";
import jsLogo from "./images/js_logo.png";
import reactLogo from "./images/react_logo.png";
import { NewInput } from "./NewInput";

const title = "Componentes";
const exercise =
  "Cree un componente funcional que muestre algunas imágenes y un Formulario con tres entradas de textos y un boton de envío";

export const ImagesComponent = () => {
  const printData = (e) => {
    e.preventDefault();
    const firstname = e.target.firstname.value;
    const lastname = e.target.lastname.value;
    const email = e.target.email.value;
    alert(
      JSON.stringify({
        nombre: firstname,
        apellido: lastname,
        correo: email,
      })
    );
  };
}
```

```

});

return (
  <Container>
    <HeaderProcess title={title} exercise={exercise} />
    <DivContainer>
      <h2 style={{ paddingBottom: "30px" }}>Tecnologías Actuales</h2>
      <DivImage>
        <NewImage url={htmlLogo} />
        <NewImage url={cssLogo} />
        <NewImage url={jsLogo} />
        <NewImage url={reactLogo} />
      </DivImage>
    </DivContainer>
    <Form onSubmit={printData}>
      <P1>SUSCRIBETE</P1>
      <P2>
        Regístrese con su dirección de correo electrónico para recibir
        noticias y actualizaciones.
      </P2>
      <DivInputs>
        <NewInput name="firstname" placeholder="Nombre" />
        <NewInput name="lastname" placeholder="Apellido" />
        <NewInput name="email" placeholder="correo" />
      </DivInputs>
      <Button>Suscribirse</Button>
    </Form>
  </Container>
);
});

const Container = styled.div`
display: flex;
flex-direction: column;
justify-content: center;
align-items: center;
`;

const DivContainer = styled.div`
background-color: #a3e4d7;
display: flex;
flex-direction: column;
align-items: center;
width: 98%;
border: 1px solid grey;
border-radius: 10px;
`;

const DivImage = styled.div`
display: flex;
padding-bottom: 20px;
`;

const Form = styled.form`
margin-top: 30px;
background-color: #a3e4d7;
border-radius: 10px;
width: 98%;
border: 1px solid grey;
`;

const P1 = styled.p`

```

```

font-size: 25px;
font-weight: 700;
`;

const P2 = styled.p`
font-size: 15px;
font-weight: 500;
margin-bottom: 50px;
`;

const DivInputs = styled.div`
flex-direction: rows;
`;

const Button = styled.button`
margin-top: 20px;
background-color: #f37474;
width: 18vw;
height: 4vh;
border-radius: 5px;
border: none;
margin-bottom: 30px;
text-align: center;
color: white;
`;

```

```

import React from "react";
import styled from "styled-components";

export const NewImage = ({ url }) => {
  return (
    <div>
      <Img src={url} />
    </div>
  );
};

const Img = styled.img`
width: 15vw;
height: 25vh;
margin: 20px;
`;

```

```

import React from "react";
import styled from "styled-components";

export const NewInput = ({ name, placeholder }) => {
  return <Input name={name} placeholder={placeholder} />;
};

const Input = styled.input`
background-color: white;
color: black;
width: 10vw;
height: 4vh;
border: none;
border-radius: 5px;
margin: 5px;
`;

```

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

Ejercicios Prácticos

Componente de Imágenes

Componentes

Cree un componente funcional que muestre algunas imágenes


Tecnologías Actuales



EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

Cree un componente funcional que muestre algunas imágenes y un Formulario con tres entradas de textos y un boton de envío

Tecnologías Actuales



SUSCRIBETE

Regístrese con su dirección de correo electrónico para recibir noticias y actualizaciones.

Nombre

Apellido

correo

Suscribirse

Matrices

Una matriz es la estructura de datos que se utiliza con más frecuencia para manejar muchos tipos de problemas. En React, usamos `map` para modificar una matriz a una lista de JSX agregando ciertos elementos HTML a cada elemento de una matriz.

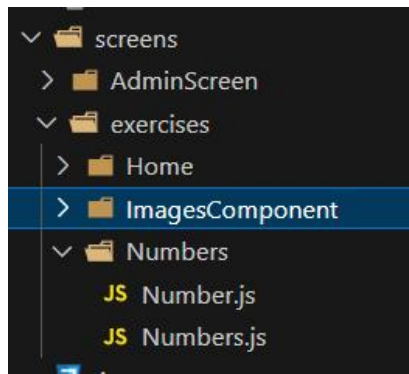
La mayoría de las veces, los datos se presentan en forma de matriz o matriz de objetos. Para representar esta matriz o matriz de objetos, la mayoría de las veces modificamos los datos mediante `map`. En la sección anterior, hemos representado la lista de tecnologías mediante un método `map`.

Las claves ayudan a React a identificar los elementos que han cambiado, añadido o eliminado. Se deben proporcionar claves a los elementos dentro de la matriz para darles una identidad estable. La clave debe ser única. La mayoría de los datos vendrán como un identificador y podemos usar el identificador como clave. Si no pasamos la clave a React durante el mapeo, se genera una advertencia en el navegador. Si los datos no tienen un identificador, tenemos que encontrar una forma de crear un identificador único para cada elemento cuando lo mapeamos.

Ejercicio 2

- En el siguiente diseño, los números pares son verdes, los impares son amarillos y los primos son rojos. Cree los siguientes colores usando el componente `React`
- Crear un área para consultar un número y obtener tipo (Par/Impar/Primo)

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31



Data.js

```
...
const getValues = (max) => {
  const numbers = [];
  for (let i = 0; i < max; i++) {
    numbers.push(i);
  }
  return numbers;
};

export const NUMBERS = getValues(32);
```

```
import React from "react";

export const Number = ({ number, bkColor, width = "6vw" }) => {
  const styles = {
    backgroundColor: bkColor,
    color: "white",
    width: width,
    height: "12vh",
    fontSize: "40px",
    border: "2px solid white",
    alignContent: "center",
  };
  return <div style={styles}>{number}</div>;
};
```

```
import React, { useEffect, useState } from "react";
import { HeaderProcess } from "../../components/headerProcess/HeaderProcess";
import styled from "styled-components";
import { Number } from "./Number";
import { NUMBERS } from "../../data/Data";

const title = "Matrices";
const exercise =
  "Cree una tabla con los números pares son verdes, los impares son amarillos y los primos son rojos.";
```

```
const pair = (number) => number === 0 || (number > 3 && number % 2 === 0);
```

```
const odd = (number) =>
  number === 1 ||
  (number > 7 && (number % 3 === 0 || number % 5 === 0 || number % 7 === 0));
```

```
const getTypeNumber = (number) => {
```

```
const isPair = pair(number);
const isOdd = odd(number);
```

```
if (!isPair && !isOdd) {
  return {
    value: number,
    color: "red",
    type: "primo",
    bkColor: "#FD5E53",
  };
}
```

```
if (isPair) {
  return {
    value: number,
    color: "green",
    type: "par",
    bkColor: "#21BF73",
  };
}
```

```
return {
  value: number,
  color: "yellow",
  type: "impar",
  bkColor: "#FDD83A",
};
};
```

```
const separateIntoGroups = (inputArr) => {
  const groups = [];
  let currentGroup = [];
```

```
  inputArr.forEach((item) => {
    currentGroup.push(item);
```

```
    if (currentGroup.length === 8) {
      groups.push([...currentGroup]);
      currentGroup = [];
    }
  });
```

```
  return groups;
};
```

```
export const Numbers = () => {
  const [data, setData] = useState(null);
  const [record, setRecord] = useState(null);
```

```
  useEffect(() => {
    const records = NUMBERS.map((value) => getTypeNumber(value));
    setData(separateIntoGroups(records));
    setRecord(getTypeNumber(0));
  }, []);
```

```
  const handleSetValue = (e) => {
    let data = e.target.value;
```

```
    if (data === "") {
      data = 0;
```

```
}
```

```
const number = parseInt(data);  
if (number < 0) {  
  return 0;  
}
```

```
setRecord(getTypeNumber(number));  
e.target.value = number;  
};
```

```
return (  
  <Container>  
    <HeaderProcess title={title} exercise={exercise} />  
    {data &&  
      data.map((record, index) => (  
        <DivRow key={index}>  
          {record.map((item) => (  
            <Number  
              key={item.value}  
              number={item.value}  
              bkColor={item.bkColor}  
            />  
          ))}  
        </DivRow>  
      )}  
    {record && (  
      <DivForm>  
        <Input value={record.value} onChange={handleSetValue} type="number" />  
        <DivRow style={{ marginTop: "20px" }}>  
          <Number  
            number={record.value}  
            bkColor={record.bkColor}  
            width={`${record.value.length / 10}vw`}  
          />  
          <DivColumn>  
            <ul style={{ alignItems: "center" }}>  
              <li>value: {record.value}</li>  
              <li>color: {record.color}</li>  
              <li>tipo: {record.type}</li>  
            </ul>  
          </DivColumn>  
        </DivRow>  
      </DivForm>  
    )}  
  </Container>  
);  
};
```

```
const Container = styled.div`  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
  align-items: center;  
`;  
;
```

```
const DivRow = styled.div`  
  display: flex;  
  flex-direction: row;  
`;  
;
```

```
const DivColumn = styled.div`
  display: flex;
  flex-direction: column;
  text-align: left;
`;
```

```
const DivForm = styled.div`
  margin-top: 20px;
`;
```

```
const Input = styled.input`
  height: 4vh;
  font-size: 20px;
`;
```

Ejercicios Prácticos

Componente de imágenes

Matrices

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

Matrices

Cree una tabla con los números pares son verdes, los impares son amarillos y los primos son rojos.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

196

196

- value: 196
- color: green
- tipo: par

Actualizar estado x accesorios o estado

Los efectos son una vía de escape del paradigma React. Te permiten “salir” de React y sincronizar tus componentes con algún sistema externo, como un widget que no sea React, una red o el DOM del navegador. Si no hay un sistema externo involucrado (por ejemplo, si quieres actualizar el estado de un componente cuando cambian algunas propiedades o el estado), no deberías necesitar un efecto. Eliminar los efectos innecesarios hará que tu código sea más fácil de seguir, más rápido de ejecutar y menos propenso a errores.

Cómo eliminar efectos innecesarios

Hay dos casos comunes en los que no se necesitan efectos:

No necesitas efectos para transformar datos para la renderización. Por ejemplo, digamos que quieres filtrar una lista antes de mostrarla. Podrías sentirte tentado a escribir un efecto que actualice una variable de estado cuando la lista cambie. Sin embargo, esto es ineficiente. Cuando actualizas el estado, React primero llamará a las funciones de tu componente para calcular lo que debería estar en la pantalla. Luego, React “confirmará” estos cambios en el DOM, actualizando la pantalla. Luego, React ejecutará tus efectos. Si tu efecto también actualiza inmediatamente el estado, ¡esto reinicia todo el proceso desde cero! Para evitar los pases de renderización innecesarios, transforma todos los datos en el nivel superior de tus componentes. Ese código se volverá a ejecutar automáticamente cada vez que cambien tus propiedades o tu estado.

No necesitas efectos para manejar eventos de usuario. Por ejemplo, digamos que quieres enviar una `/api/buysolicitud` POST y mostrar una notificación cuando el usuario compra un producto. En el controlador de eventos de clic del botón Comprar, sabes exactamente qué sucedió. Para cuando se ejecuta un efecto, no sabes qué hizo el usuario (por ejemplo, en qué botón se hizo clic). Por eso, generalmente manejarás eventos de usuario en los controladores de eventos correspondientes.

Necesitas efectos para sincronizar con sistemas externos. Por ejemplo, puedes escribir un efecto que mantenga un widget jQuery sincronizado con el estado de React. También puedes obtener datos con efectos: por ejemplo, puedes sincronizar los resultados de búsqueda con la consulta de búsqueda actual. Ten en cuenta que los marcos modernos proporcionan mecanismos de obtención de datos integrados más eficientes que escribir efectos directamente en tus componentes.

Para ayudarle a obtener la intuición correcta, veamos algunos ejemplos concretos y comunes!

Actualización del estado en función de los accesorios o el estado

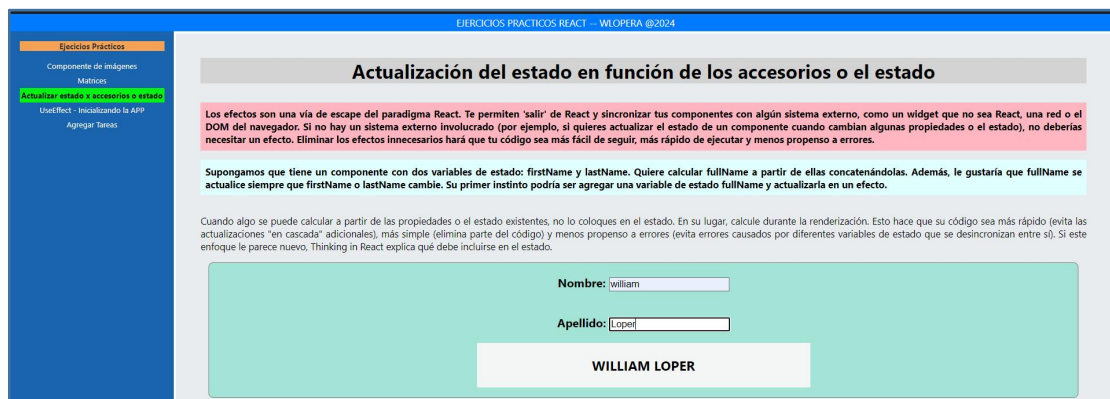
Supongamos que tiene un componente con dos variables de estado: `firstName` y `lastName`. Quiere calcular `fullName` a partir de ellas concatenándolas. Además, le

gustaría `fullName` actualizarse siempre que `firstName` o `lastName` cambie. Su primer instinto podría ser agregar una `fullName` variable de estado y actualizarla en un efecto:

Ejercicio 3

- Supongamos que tiene un componente con dos variables de estado: `firstName` y `lastName`. Quiere calcular `fullName` a partir de ellas concatenándolas. Además, le gustaría que `fullName` se actualice siempre que `firstName` o `lastName` cambie. Su primer instinto podría ser agregar una variable de estado `fullName` y actualizarla en un efecto.

Cuando algo se puede calcular a partir de las propiedades o el estado existentes, no lo coloques en el estado. En su lugar, calcule durante la renderización. Esto hace que su código sea más rápido (evita las actualizaciones "en cascada" adicionales), más simple (elimina parte del código) y menos propenso a errores (evita errores causados por diferentes variables de estado que se desincronizan entre sí). Si este enfoque le parece nuevo, *Thinking in React* explica qué debe incluirse en el estado.



ParentComponent.js

```
import React, { useState } from "react";
import { HeaderProcess } from "../../components/headerProcess/HeaderProcess";
import styled from "styled-components";
import { ChildrenComponent } from "../ChildrenComponent";

const title =
  "Actualización del estado en función de los accesorios o el estado";
const description =
  "Los efectos son una vía de escape del paradigma React. Te permiten 'salir' de React y sincronizar tus componentes con algún sistema externo, como un widget que no sea React, una red o el DOM del navegador. Si no hay un sistema externo involucrado (por ejemplo, si quieres actualizar el estado de un componente cuando cambian algunas propiedades o el estado), no deberías necesitar un efecto. Eliminar los efectos innecesarios hará que tu código sea más fácil de seguir, más rápido de ejecutar y menos propenso a errores.";
const exercise =
  "Supongamos que tiene un componente con dos variables de estado: firstName y lastName. Quiere calcular fullName a partir de ellas concatenándolas. Además, le gustaría que fullName
```

se actualice siempre que firstName o lastName cambie. Su primer instinto podría ser agregar una variable de estado fullName y actualizarla en un efecto.";

```
export const ParentComponent = () => {  
  const [firstName, setFirstName] = useState("");  
  const [lastName, setLastName] = useState("");
```

```
  return (  
    <Container>  
      <HeaderProcess  
        title={title}  
        description={description}  
        exercise={exercise}  
      />  
      <p style={{ textAlign: "justify", fontSize: "20px" }}>  
        Cuando algo se puede calcular a partir de las propiedades o el estado  
        existentes, no lo coloques en el estado. En su lugar, calcule durante la  
        renderización. Esto hace que su código sea más rápido (evita las  
        actualizaciones "en cascada" adicionales), más simple (elimina parte del  
        código) y menos propenso a errores (evita errores causados por  
        diferentes variables de estado que se desincronizan entre sí). Si este  
        enfoque le parece nuevo, Thinking in React explica qué debe incluirse en  
        el estado.  
      </p>  
      <DivContainer>  
        <DivInput>  
          <P>  
            Nombre:{" "}   
            <Input  
              name="firstname"  
              value={firstName}  
              onChange={(e) => setFirstName(e.target.value)}  
            />  
          </P>  
        </DivInput>  
        <DivInput>  
          <P>  
            Apellido:{" "}   
            <Input  
              name="lastname"  
              value={lastName}  
              onChange={(e) => setLastName(e.target.value)}  
            />  
          </P>  
        </DivInput>  
      </DivContainer>  
    </Container>  
  );  
};
```

```
    <ChildrenComponent name={firstName} surname={lastName} />  
  </DivContainer>  
</Container>  
);  
};
```

```
const Container = styled.div`  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
  align-items: center;  
  width: 95%;  
  padding-left: 40px;  
`;  
;
```

```
const DivContainer = styled.div`
  background-color: #a3e4d7;
  display: flex;
  flex-direction: column;
  align-items: center;
  width: 98%;
  border: 1px solid grey;
  border-radius: 10px;
`;
```

```
const DivInput = styled.div`
  background-color: #a3e4d7;
  flex-direction: row;
  align-items: center;
`;
```

```
const P = styled.p`
  font-size: 24px;
  font-weight: bold;
`;
```

```
const Input = styled.input`
  font-size: 20px;
`;
```

ChildrenComponent.js

```
import React from "react";
import styled from "styled-components";

export const ChildrenComponent = ({ name, surname }) => {
  // Solo se actualiza la primera vez luego ya no renderiza aunque cambie el name y el surname
  //const [fullName, setFullName] = useState(name + " " + surname);
```

```
  // const [fullName, setFullName] = useState('');
  // useEffect(() => {
  //   setFullName(name + " " + surname);
  // }, [name, surname]);
```

```
  const fullName = name + " " + surname;
```

```
  console.log("Cambio: ", fullName);
```

```
  return (
    <Container>
      <DivContainer>
        <P>{fullName.toUpperCase()}</P>
      </DivContainer>
    </Container>
  );
};
```

```
const Container = styled.div`
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
```



```
padding-bottom: 20px;  
`;  
`;
```

```
const DivContainer = styled.div`  
  background-color: #f4f6f6;  
  align-items: center;  
  width: 30vw;  
`;  
`;
```

```
const P = styled.p`  
  font-size: 28px;  
  font-weight: bold;  
`;  
`;
```

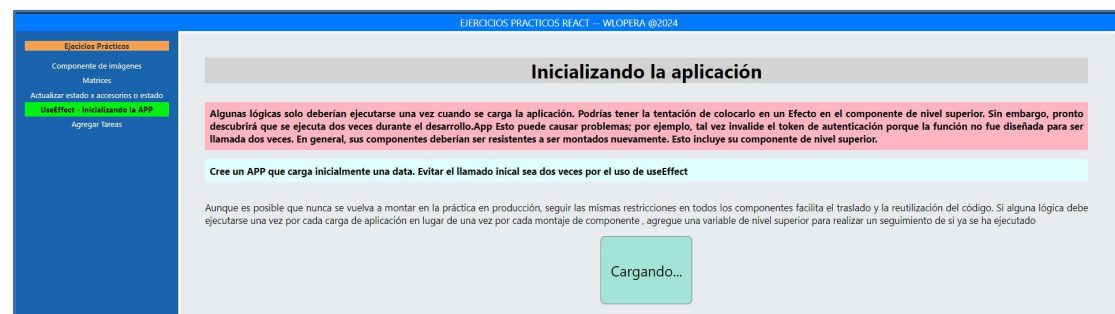
Inicializando la aplicación

Algunas lógicas solo deberían ejecutarse una vez cuando se carga la aplicación. Podrías tener la tentación de colocarlo en un Efecto en el componente de nivel superior. Sin embargo, pronto descubrirá que se ejecuta dos veces durante el desarrollo.App Esto puede causar problemas; por ejemplo, tal vez invalide el token de autenticación porque la función no fue diseñada para ser llamada dos veces. En general, sus componentes deberían ser resistentes a ser montados nuevamente. Esto incluye su componente de nivel superior.

Ejercicio 4

- Cree un APP que carga inicialmente una data. Evitar el llamado inicial se ejecute dos veces por el uso de useEffect

Aunque es posible que nunca se vuelva a montar en la práctica en producción, seguir las mismas restricciones en todos los componentes facilita el traslado y la reutilización del código. Si alguna lógica debe ejecutarse una vez por cada carga de aplicación en lugar de una vez por cada montaje de componente, agregue una variable de nivel superior para realizar un seguimiento de si ya se ha ejecutado



```
import React, { useEffect, useState } from "react";
import { HeaderProcess } from "../../components/headerProcess/HeaderProcess";
import styled from "styled-components";

const title = "Iniciando la aplicación";
const description =
  "Algunas lógicas solo deberían ejecutarse una vez cuando se carga la aplicación. Podrías tener la tentación de colocarlo en un Efecto en el componente de nivel superior. Sin embargo, pronto descubrirá que se ejecuta dos veces durante el desarrollo.App Esto puede causar problemas; por ejemplo, tal vez invalide el token de autenticación porque la función no fue diseñada para ser llamada dos veces. En general, sus componentes deberían ser resistentes a ser montados nuevamente. Esto incluye su componente de nivel superior.";
const exercise =
  "Cree un APP que carga inicialmente una data. Evitar el llamado inicial sea dos veces por el uso de useEffect";

export const UseEffectInitApp = () => {
```

```
const [data, setData] = useState([]);
```

```
useEffect(() => {  
  const fetchData = async () => {  
    setTimeout(() => {  
      setData([  
        { id: 1, name: "Java", language: "Desarrollo Backend" },  
        { id: 2, name: "React", language: "Desarrollo Frontend" },  
        { id: 3, name: "C++", language: "Desarrollo Backend" },  
        { id: 4, name: "Angular", language: "Desarrollo Frontend" },  
        { id: 5, name: "Python", language: "Desarrollo Backend" },  
        { id: 6, name: "JavaScript", language: "Desarrollo Frontend" },  
      ]);  
    }, 2000);  
  };  
});
```

```
    fetchData();  
  }, []);
```

```
    console.log("Data: ", data);  
    return (  
      <Container>  
        <HeaderProcess  
          title={title}  
          description={description}  
          exercise={exercise}  
        />  
        <p style={{ textAlign: "justify", fontSize: "20px" }}>  
          Aunque es posible que nunca se vuelva a montar en la práctica en  
          producción, seguir las mismas restricciones en todos los componentes  
          facilita el traslado y la reutilización del código. Si alguna lógica  
          debe ejecutarse una vez por cada carga de aplicación en lugar de una vez  
          por cada montaje de componente , agregue una variable de nivel superior  
          para realizar un seguimiento de si ya se ha ejecutado  
        </p>  
        <DivContainer>  
          {data.length === 0 ? (  
            <p style={{ fontSize: "30px", padding: "20px" }}>Cargando...</p>  
          ) : (  
            <ul>  
              {data &&  
                data.map((item) => (  
                  <ListItem key={item.id}>  
                    <span>{item.name}</span>  
                    <span>{item.language}</span>  
                  </ListItem>  
                ))}  
            </ul>  
          )}  
        </DivContainer>  
      </Container>  
    );  
  }  
};
```

```
const Container = styled.div`  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
  align-items: center;  
  width: 95%;
```

```
`;
padding-left: 40px;
`;
```

```
const DivContainer = styled.div`
  background-color: #a3e4d7;
  display: flex;
  flex-direction: column;
  align-items: center;
  border: 1px solid grey;
  border-radius: 10px;
`;
```

```
const ListItem = styled.li`
  display: flex;
  font-size: 30px;
  text-align: left;
  width: 35vw;
  span {
    flex: 1;
  }
`;
```

Ejercicios Prácticos

Componente de imágenes

Matrices

Actualizar estado y accesorios o estado

useEffect Inicializando la APP

Agregar Tareas

EJERCICIOS PRÁCTICOS REACT -- WLOPERA @2024

Inicializando la aplicación

Algunas lógicas solo deberían ejecutarse una vez cuando se carga la aplicación. Podrías tener la tentación de colocarlo en un Efecto en el componente de nivel superior. Sin embargo, pronto descubrirá que se ejecuta dos veces durante el desarrollo.App Esto puede causar problemas; por ejemplo, tal vez invalide el token de autenticación porque la función no fue diseñada para ser llamada dos veces. En general, sus componentes deberían ser resistentes a ser montados nuevamente. Esto incluye su componente de nivel superior.

Cree un APP que carga inicialmente una data. Evitar el llamado inicial sea dos veces por el uso de useEffect

Aunque es posible que nunca se vuelva a montar en la práctica en producción, seguir las mismas restricciones en todos los componentes facilita el traslado y la reutilización del código. Si alguna lógica debe ejecutarse una vez por cada carga de aplicación en lugar de una vez por cada montaje de componente , agregue una variable de nivel superior para realizar un seguimiento de si ya se ha ejecutado

Java

React

C++

Angular

Python

JavaScript

Desarrollo Backend

Desarrollo Frontend

Desarrollo Backend

Desarrollo Frontend

Desarrollo Backend

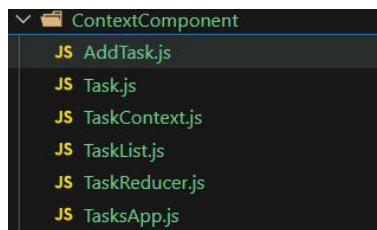
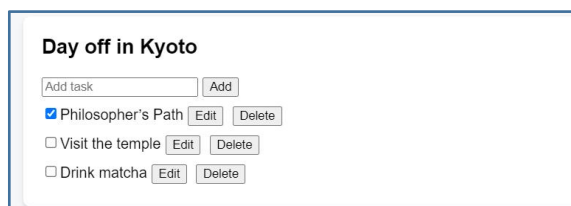
Desarrollo Frontend

Ampliación de escala con Reducer y Context

Los reducers le permiten consolidar la lógica de actualización de estado de un componente. El contexto le permite pasar información a otros componentes. Puede combinar reducers y contexto para administrar el estado de una pantalla compleja.

Ejercicio 5

Cree un componente funcional que permita agregar, editar y eliminar tareas mediante el uso de `createContext`, `useContext` y `useReducer`



TasksApp.js

```
import React from "react";
import { HeaderProcess } from "../../components/headerProcess/HeaderProcess";
import styled from "styled-components";
import { TaskProvider } from "../TaskContext";
import { TaskList } from "../TaskList";
import { AddTask } from "../AddTask";

const title = "Tareas: useContext - Reducer";
const description =
  "useContext es un React Hook que te permite leer y suscribirte al contexto de tu componente. Los reducers le permiten consolidar la lógica de actualización de estado de un componente. El contexto le permite pasar información a otros componentes. Puede combinar reducers y contexto para administrar el estado de una pantalla compleja.";
const exercise =
  "Cree un componente funcional que permita agregar, editar y eliminar tareas mediante el uso de createContext, useContext y useReducer";
```

```
export const TasksApp = () => {
  return (
    <Container>
      <HeaderProcess
        title={title}
        description={description}
        exercise={exercise}
      />
    </Container>
  )
}
```

```

    />
    <DivContainer>
      <TaskProvider>
        <AddTask />
        <TaskList />
      </TaskProvider>
    </DivContainer>
  </Container>
);
};

```

```

const Container = styled.div`
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 95%;
  padding-left: 40px;
`;

```

```

const DivContainer = styled.div`
  flex-direction: column;
  align-items: center;
  width: 100%;
`;

```

TaskReducer.js

```

export default function taskReducer(tasks, action) {
  switch (action.type) {
    case "added":
      return [
        ...tasks,
        {
          id: action.id,
          text: action.text,
          selected: false,
        },
      ];

    case "modify":
      return tasks.map((task) => {
        if (task.id === action.id) {
          return {
            id: action.id,
            text: action.text,
            selected: action.selected,
          };
        }
        return task;
      });
  }
}

```

```

    case "delete":
      return tasks.filter((task) => task.id !== action.id);
  }
}

```

```

    default:
      return tasks;
  }
}

```

TaskContext.js

```
import { createContext, useContext, useReducer } from "react";
import taskReducer from "../TaskReducer";
```

```
// Contexto que controla las tareas
export const TaskContext = createContext(null);
```

```
// Contexto que controla la acciones sobre las tareas
export const TaskDispatchContext = createContext(null);
```

```
// Funcion que permite acceso al contexto de las tareas
export const useTasks = () => {
  return useContext(TaskContext);
};
```

```
// Funcion que permite acceso a la acciones sobre las tareas
export const useTasksDispatch = () => {
  return useContext(TaskDispatchContext);
};
```

```
const initTasks = [
  {
    id: 1,
    text: "Levantarse temprano 5:00 am",
    selected: true,
  },
  {
    id: 2,
    text: "Desayunar a las 6:00 am",
    selected: false,
  },
  {
    id: 3,
    text: "Salir al trabajo 7:30 am",
    selected: false,
  },
];
```

```
export const TaskProvider = ({ children }) => {
  const [tasks, dispatch] = useReducer(taskReducer, initTasks);
```

```
  return (
    <TaskContext.Provider value={tasks}>
      <TaskDispatchContext.Provider value={dispatch}>
        {children}
      </TaskDispatchContext.Provider>
    </TaskContext.Provider>
  );
};
```

TaskList.js

```
import styled from "styled-components";
import { useTasks } from "../TaskContext";
import { Task } from "../Task";
```

```
export const TaskList = () => {
  const tasks = useTasks();
```

```
  return (
    <TaskContainer>
```

```

        {tasks.map((task) => (
          <Task key={task.id} {...task} />
        ))}
      </TaskContainer>
    );
  };
};

```

```

const TaskContainer = styled.div`
  width: 40%; /* Ajusta el ancho del contenedor */
`;

```

Task.js

```

import { useState } from "react";
import styled from "styled-components";
import { useTasksDispatch } from "../TaskContext";

export const Task = ({ id, text, selected }) => {
  const [editable, setEditable] = useState(false);

```

```

    const dispatch = useTasksDispatch();

```

```

    const handleChangeCheckedTask = (checked) => {
      dispatch({
        id,
        text,
        selected: checked,
        type: "modify",
      });
    };
  };

```

```

    const handleDeleteTask = () => {
      dispatch({
        id,
        type: "delete",
      });
    };
  };

```

```

    const textContext = () => {
      return editable ? (
        <>
          <InputText
            type={text}
            value={text}
            onChange={(e) => {
              dispatch({
                id,
                text: e.target.value,
                selected,
                type: "modify",
              });
            }}
          />
          <ButtonContainer>
            <Button onClick={() => setEditable(false)}>Salvar</Button>
          </ButtonContainer>
        </>
      ) : (
        <>
          <TaskText>{text}</TaskText>

```



```

        <ButtonContainer>
          <Button onClick={() => setEditable(true)}>Editar</Button>
        </ButtonContainer>
      </>
    );
  };
};

```

```

return (
  <Container>
    <InputChecked
      checked={selected}
      type="checkbox"
      onChange={(e) => handleChangeCheckedTask(e.target.checked)}
    />
    {textContext()}
    <ButtonContainer>
      <Button onClick={handleDeleteTask}>Eliminar</Button>
    </ButtonContainer>
  </Container>
);
};

```

```

const Container = styled.div`
  display: flex;
  align-items: center; /* Centra verticalmente los elementos */
  padding: 10px;
  border-bottom: 1px solid #ddd; /* Línea divisoria entre elementos */
`;

```

```

const InputChecked = styled.input`
  flex: 0 0 10%; /* Ancho fijo del 10% */
  text-align: left; /* Alinea el texto a la izquierda */
  height: 2vh;
  width: 2vw;
`;

```

```

const InputText = styled.input`
  flex: 0 0 80%; /* Ancho fijo del 50% */
  text-align: left; /* Alinea el texto a la izquierda */
  font-size: 20px;
`;

```

```

const TaskText = styled.div`
  flex: 0 0 80%; /* Ancho fijo del 50% */
  text-align: left; /* Alinea el texto a la izquierda */
  font-size: 20px;
`;

```

```

const ButtonContainer = styled.div`
  display: flex;
  flex: 0 0 10%; /* Ancho fijo del 20% */
  padding-left: 10px;
  justify-content: space-between; /* Distribuye el espacio entre los botones */
`;

```

```

const Button = styled.button`
  padding: 5px 10px;
  border: none;
  border-radius: 4px;
  cursor: pointer;

```

```
background-color: #007bff;
color: #fff;
font-size: 14px;
```

```
&:hover {
  background-color: #0056b3;
}
`;
```

AddTask.js

```
import styled from "styled-components";
import { useTasksDispatch } from "../TaskContext";
import { useState } from "react";
import { v4 as uuid4 } from "uuid";
```

```
export const AddTask = () => {
  const [text, setText] = useState("");
  const dispatch = useTasksDispatch();
```

```
  const handleAddedTask = () => {
    if (text.length === 0) {
      return;
    }
    dispatch({
      id: uuid4(),
      text,
      selected: false,
      type: "added",
    });
    setText("");
  };
};
```

```
  return (
    <Container>
      <InputText
        type={text}
        value={text}
        onChange={(e) => setText(e.target.value)}
      />
      <ButtonContainer>
        <Button onClick={handleAddedTask}>Agregar</Button>
      </ButtonContainer>
    </Container>
  );
};
```

```
const Container = styled.div`
  display: flex;
  align-items: center; /* Centra verticalmente los elementos */
  padding: 10px;
  border-bottom: 1px solid #ddd; /* Línea divisoria entre elementos */
`;
```

```
const InputText = styled.input`
  flex: 0 0 35.8%; /* Ancho fijo del 50% */
  text-align: left; /* Alinea el texto a la izquierda */
  font-size: 20px;
`;
```

```
const ButtonContainer = styled.div`
  display: flex;
  flex: 0 0 10%; /* Ancho fijo del 10% */
  padding-left: 20px;
  justify-content: space-between; /* Distribuye el espacio entre los botones */
`;
```

```
const Button = styled.button`
  width: 6vw;
  padding: 5px 10px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  background-color: #28a745;
  color: #fff;
  font-size: 14px;
```

```
&:hover {
  background-color: #2ecc71;
}
```

Agregar



Modificar:



Borrar

Ejercicios Prácticos

Componente de imágenes

Matrices

Actualizar estado a acciones o estado

useEffect - Incalculando la APP

Agregar Tareas

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

Tareas: useContext - Reducer

useContextes un React Hook que te permite leer y suscribirte al contexto de tu componente. Los reducers le permiten consolidar la lógica de actualización de estado de un componente. El contexto le permite pasar información a otros componentes. Puede combinar reducers y contexto para administrar el estado de una pantalla compleja.

Cree un componente funcional que permita agregar, editar y eliminar tareas mediante el uso de createContext, useContext y useReducer

Agregar

☒ Levantarse temprano 5:00 am

EditarEliminar

☐ Desayunar a las 6:00 am

EditarEliminar

☐ Salir al trabajo 7:30 am

EditarEliminar

☐ Salir del ytrabajo 5:15 pm E ir de compras

EditarEliminar