

# PROYECTO EJERCICIOS PRACTICOS CON REACT

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

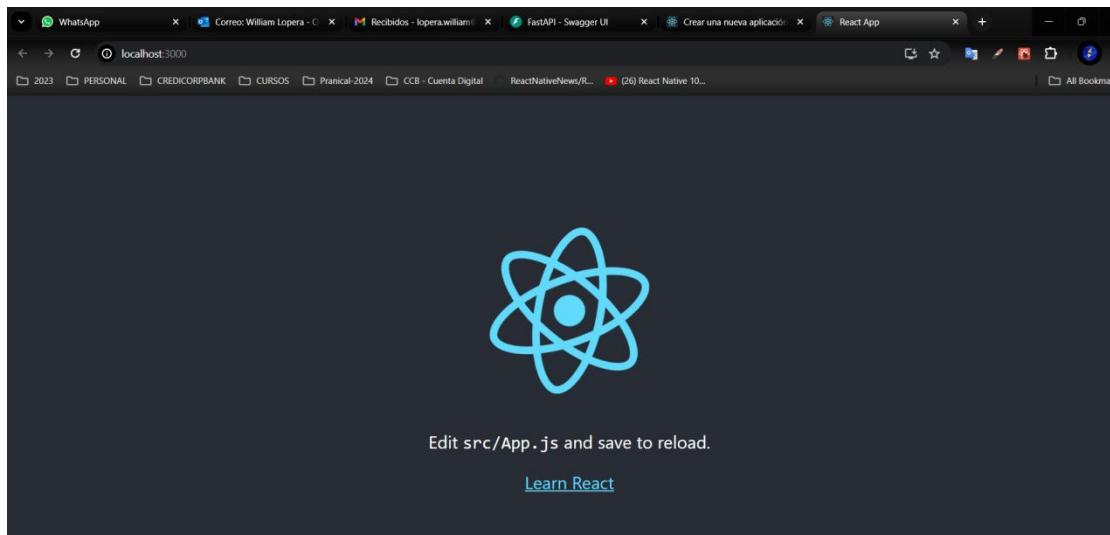
Se mostrara un ejercicio y su solución utilizando lo menos posible internet ni uso de aplicaciones de chatbot de inteligencia artificial . Para practicar React, Javascript, Html y CSS.

- 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.
(Use 'node --trace-deprecation ...' to show where the warning was created)
(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
```



- VS CODE: Depurar el código

The image shows the Visual Studio Code interface with two code editors open. The left editor displays the file 'index.js' containing:

```

1 import React from "react";
2 import ReactDOM from "react-dom/client";
3 import "./index.css";
4 import App from "./App";
5
6 const root = ReactDOM.createRoot(document.getElementById("root"));
7 root.render(
8   <React.StrictMode>
9     <App />
10   </React.StrictMode>
11 );
12
13

```

The right editor displays the file 'App.js' containing:

```

1 import "./App.css";
2
3 function App() {
4   return (
5     <div className="App">
6       <h1>Ventana Principal</h1>
7       <hr />
8     </div>
9   );
10
11
12 export default App;
13

```

Below the code editors is a screenshot of a web browser window showing the application running at 'localhost:3000'. The page has a single heading element: <h1>Ventana Principal</h1>.

## Código

Se crearon cuatro componentes importantes:

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

Además se crearon componentes para el manejo de:

- mensaje: Control de mensajes
- HeaderProcess: Control de título, 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"

```

## Ejercicio 1

Disponer dos controles de formulario HTML input='number' y un botón. Al presionar el botón mostrar en un alert su suma."

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

const title = "Captura de eventos";
const description =
  "Los nombres de eventos en React comienzan con 'on' y luego el primer caracter de cada palabra en mayúsculas: [onClick, onDoubleClick, onMouseEnter, onMouseMove, onKeyPress, onSubmit, etc]";
const exercise =
  "Ejercicio: Disponer dos controles de formulario HTML input='number' y un botón. Al presionar el botón mostrar en un alert su suma.";

export const EventCapture = () => {
  const sum = (e) => {
    e.preventDefault();
    const x = parseFloat(e.target.x.value);
    const y = parseFloat(e.target.y.value);
    return alert(`${x} + ${y} = ${x + y}`);
  };
  return (
    <Container>
      <HeaderProcess
        title={title}
        description={description}
        exercise={exercise}
      />

      <Form onSubmit={sum}>
        <Label>
          Ingrese primer valor:
          <input type="number" name="x" />
        </Label>
        <Label>
          Ingrese segundo valor:
          <input type="number" name="y" />
        </Label>
        <Button>Sumar</Button>
      </Form>
    </Container>
  );
};

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

const Form = styled.form`
  display: flex;
  background-color: lightgreen;
  flex-direction: column;
`;
```

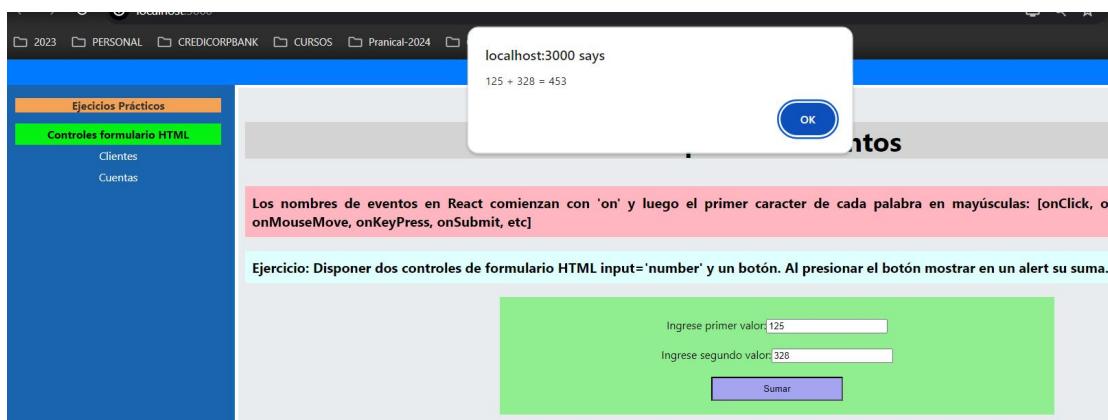
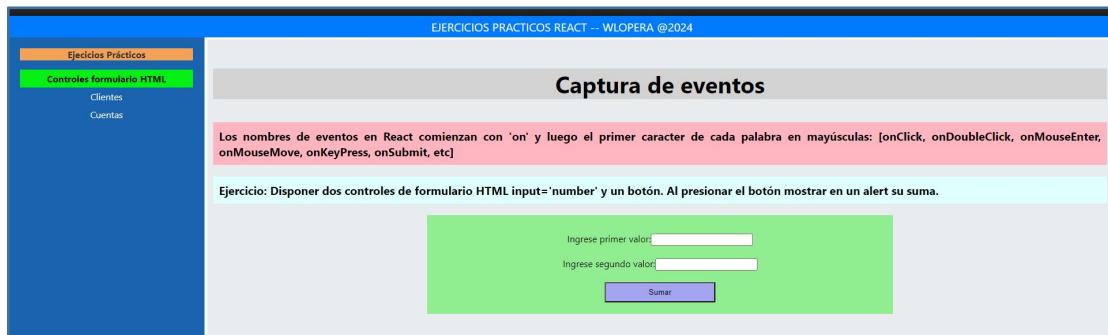
```

    align-items: center;
    width: 40vw;
    padding: 20px;
  `;

const Label = styled.label`
  padding: 10px;
`;

const Button = styled.button`
  margin-top: 10px;
  background-color: rgb(164, 164, 241);
  width: 10vw;
  height: 4vh;
`;

```



## Ejercicio 2

Definir en la interfaz visual un botón que cada vez que se presione se actualice en pantalla un número aleatorio entre 0 y 9.

```
import React, { useState } from "react";
import styled from "styled-components";

import { HeaderProcess } from "../../components/headerProcess/HeaderProcess";

const title = "Variables de estado de una componente mediante Hook";
const description =
  "Un Hook de estado es una función especial que nos permite conectarnos a las funciones de la librería de React. Una componente en React si necesita almacenar valores que luego en forma dinámica se actualizarán en pantalla, lo podemos resolver mediante Hook de estado. Por ejemplo un contador de productos seleccionados, un contador de segundos que se muestra en pantalla, la hora etc. Debemos importar la función 'useState' si queremos administrar Hook de estados:
[import React, { useState } from 'react'];
const exercise =
  \"Definir en la interfaz visual un botón que cada vez que se presione se actualice en pantalla un número aleatorio entre 0 y 9.\"

export const RandomNumber = () => {
  const [number, setNumber] = useState();

  const handleRandomNumber = () => {
    const value = Math.floor(Math.random() * 10);
    setNumber(value);
  };

  return (
    <Container>
      <HeaderProcess
        title={title}
        description={description}
        exercise={exercise}
      />
      <Div>
        <Button onClick={handleRandomNumber}>Número</Button>
        <Label>Número aleatorio generado: {number}</Label>
      </Div>
    </Container>
  );
};

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

const Div = styled.div`
  display: flex;
  flex-direction: column;
  align-items: center;
  padding: 20px;
  background-color: #90ee90;
`;
```

```

const Button = styled.button`  

  background-color: #007bff;  

  color: white;  

  border: 1px solid black;  

  border-radius: 20px;  

  width: 10vw;  

  height: 3vh;  

`;  
  

const Label = styled.label`  

  margin-top: 20px;  

  font-size: 30px;  

  color: black;  

  width: 40vw;  

  background-color: #ffc107;  

  text-align: left;  

`;

```

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Variables de estado de una componente mediante Hook**

Un Hook de estado es una función especial que nos permite conectarnos a las funciones de la librería de React. Una componente en React si necesita almacenar valores que luego en forma dinámica se actualizan en pantalla, lo podemos resolver mediante Hook de estado. Por ejemplo un contador de productos seleccionados, un contador de segundos que se muestra en pantalla, la hora etc. Debemos importar la función 'useState' si queremos administrar Hook de estados: [Import React, { useState } from 'react';]

Definir en la interfaz visual un botón que cada vez que se presione se actualice en pantalla un número aleatorio entre 0 y 9.

Número

Número aleatorio generado:

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Variables de estado de una componente mediante Hook**

Un Hook de estado es una función especial que nos permite conectarnos a las funciones de la librería de React. Una componente en React si necesita almacenar valores que luego en forma dinámica se actualizan en pantalla, lo podemos resolver mediante Hook de estado. Por ejemplo un contador de productos seleccionados, un contador de segundos que se muestra en pantalla, la hora etc. Debemos importar la función 'useState' si queremos administrar Hook de estados: [Import React, { useState } from 'react';]

Definir en la interfaz visual un botón que cada vez que se presione se actualice en pantalla un número aleatorio entre 0 y 9.

Número

Número aleatorio generado: 6

### Ejercicio 3

Almacenar en el estado de la componente el siguiente vector con artículos:

```
[  
  {  
    codigo: 1,  
    descripcion: 'papas',  
    precio: 12.52  
  },  
  {  
    codigo: 2,  
    descripcion: 'naranjas',  
    precio: 21  
  },  
  {  
    codigo: 3,  
    descripcion: 'peras',  
    precio: 18.20  
  }]
```

- Mostrar en una tabla HTML dichos datos.
- Borrar el artículo cuando se presione el botón borrar de la tabla.

Data

```
export const ARTICLES = [  
  {  
    codigo: 1,  
    descripcion: "papas",  
    precio: 12.52,  
  },  
  {  
    codigo: 2,  
    descripcion: "naranjas",  
    precio: 21,  
  },  
  {  
    codigo: 3,  
    descripcion: "peras",  
    precio: 18.2,  
  },  
];
```

Componente

```
import React, { useState } from "react";  
import styled from "styled-components";  
  
import { HeaderProcess } from "../../components/headerProcess/HeaderProcess";  
import { ARTICLES } from "../../data/Data";  
  
const title = "Variables de estado de una componente mediante Hook";  
const description =
```

"Un Hook de estado es una función especial que nos permite conectarnos a las funciones de la librería de React. Una componente en React si necesita almacenar valores que luego en forma dinámica se actualizarán en pantalla, lo podemos resolver mediante Hook de estado. Por ejemplo un contador de productos seleccionados, un contador de segundos que se muestra en pantalla, la hora etc. Debemos importar la función 'useState' si queremos administrar Hook de estados:

```
[import React, { useState } from 'react'];
const exercise =
```

"Ejercicio: Almacenar en el estado de la componente un vector con artículos. Mostrar en una tabla HTML dichos datos. Borrar el artículo cuando se presione el botón borrar de la tabla. [Se utiliza una data dummy de artículos]";

```
const TableArticles = () => {
  const [articles, setArticles] = useState(ARTICLES);

  const handleDeleteArticle = (codigo) => {
    const filter = articles.filter((article) => article.codigo !== codigo);
    setArticles(filter);
  };

  return (
    <Container>
      <HeaderProcess
        title={title}
        description={description}
        exercise={exercise}
      />
      <table>
        <thead>
          <tr>
            <TH>Código</TH>
            <TH>Descripción</TH>
            <TH>Precio</TH>
            <TH>Acción</TH>
          </tr>
        </thead>
        <tbody>
          {articles.map((article) => (
            <TR key={article.codigo}>
              <TD>{article.codigo}</TD>
              <TD>{article.descripcion}</TD>
              <TD>{article.precio}</TD>
              <TD onClick={() => handleDeleteArticle(article.codigo)}>x</TD>
            </TR>
          ))}
        </tbody>
      </table>
    </Container>
  );
};

export default TableArticles;

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

const TR = styled.tr`
  border: 1px solid black;
`;
```

```

  &:hover {
    background-color: #bdbdbd;
  }

const TH = styled.th`  

  border: 1px solid black;  

  font-size: 20px;  

  width: 10vw;  

  background-color: #28a745;  

  color: white;
`;

const TD = styled.td`  

  border: 1px solid black;  

  font-size: 16px;  

  font-weight: bold;
`;

```

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Variables de estado de una componente mediante Hook**

Un Hook de estado es una función especial que nos permite conectarnos a las funciones de la librería de React. Una componente en React si necesita almacenar valores que luego en forma dinámica se actualizan en pantalla, lo podemos resolver mediante Hook de estado. Por ejemplo un contador de productos seleccionados, un contador de segundos que se muestra en pantalla, la hora etc. Debemos importar la función 'useState' si queremos administrar Hook de estados: [import React, { useState } from 'react';]

Ejercicio: Almacenar en el estado de la componente un vector con artículos. Mostrar en una tabla HTML dichos datos. Borrar el artículo cuando se presione el botón borrar de la tabla. [Se utiliza una data dummy de artículos]

Código	Descripción	Precio	Acción
1	papas	12.52	x
2	naranjas	21	x
3	peras	18.2	x

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Variables de estado de una componente mediante Hook**

Un Hook de estado es una función especial que nos permite conectarnos a las funciones de la librería de React. Una componente en React si necesita almacenar valores que luego en forma dinámica se actualizan en pantalla, lo podemos resolver mediante Hook de estado. Por ejemplo un contador de productos seleccionados, un contador de segundos que se muestra en pantalla, la hora etc. Debemos importar la función 'useState' si queremos administrar Hook de estados: [import React, { useState } from 'react';]

Ejercicio: Almacenar en el estado de la componente un vector con artículos. Mostrar en una tabla HTML dichos datos. Borrar el artículo cuando se presione el botón borrar de la tabla. [Se utiliza una data dummy de artículos]

Código	Descripción	Precio	Acción
1	papas	12.52	x
2	naranjas	21	x
3	peras	18.2	x

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Variables de estado de una componente mediante Hook**

Un Hook de estado es una función especial que nos permite conectarnos a las funciones de la librería de React. Una componente en React si necesita almacenar valores que luego en forma dinámica se actualizan en pantalla, lo podemos resolver mediante Hook de estado. Por ejemplo un contador de productos seleccionados, un contador de segundos que se muestra en pantalla, la hora etc. Debemos importar la función 'useState' si queremos administrar Hook de estados: [import React, { useState } from 'react';]

Ejercicio: Almacenar en el estado de la componente un vector con artículos. Mostrar en una tabla HTML dichos datos. Borrar el artículo cuando se presione el botón borrar de la tabla. [Se utiliza una data dummy de artículos]

Código	Descripción	Precio	Acción
1	papas	12.52	x
3	peras	18.2	x

## Componentes

Las componentes son una característica fundamental de React. Nos permiten dividir la aplicación en trozos de nuestra interfaz visual con objetivos bien definidos: 'menú de opciones', 'formulario de búsqueda', 'ventanas de mensajes', 'tablas de datos' etc.

La división de nuestra aplicación en componentes nos ayudan a reutilizar dicho código en proyectos futuros.

Hasta ahora cada proyecto que hemos desarrollado ha implementado una sola componente funcional (hacemos referencia a que se trata de una componente funcional, ya que versiones antiguas de React proponían por defecto una componente de clase)

## Ejercicio 4

Crear un nuevo proyecto con la herramienta `create-react-app` llamado `proyecto006`. Confeccionar una aplicación que muestre tres dados. Plantear una componente funcional llamada 'Dado' que tenga una propiedad llamada 'valor' a la cual le llega el dato a mostrar. Componente principal 'ThreeDices' y dentro de la misma definiremos 3 componentes de tipo 'Dado'

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

const title = "Componentes";
const description =
  "Las componentes son una característica fundamental de React. Nos permiten dividir la aplicación en trozos de nuestra interfaz visual con objetivos bien definidos: 'menú de opciones', 'formulario de búsqueda', 'ventanas de mensajes', 'tablas de datos' etc. La división de nuestra aplicación en componentes nos ayudan a reutilizar dicho código en proyectos futuros. Hasta ahora cada proyecto que hemos desarrollado ha implementado una sola componente funcional llamada 'App' (hacemos referencia a que se trata de una componente funcional, ya que versiones antiguas de React proponían por defecto una componente de clase)";

const exercise =
  "Ejercicio: Confeccionar una aplicación que muestre tres dados. Plantear una componente funcional llamada 'Dado' que tenga una propiedad llamada 'valor' a la cual le llega el dato a mostrar. Componente principal 'ThreeDices' y dentro de la misma definiremos 3 componentes de tipo 'Dado'";

export const ThreeDices = () => {
  const [dices, setDices] = useState([]);

  const handleGetValues = () => {
    let values = [];
    for (let i = 0; i < 3; i++) {
      values.push(Math.floor(Math.random() * 6) + 1);
    }
    setDices(values);
  }

  return (
    <div>
      <h1>Tres Dados</h1>
      <p>Los resultados son:</p>
      <ul>
        <li>{dices[0]}</li>
        <li>{dices[1]}</li>
        <li>{dices[2]}</li>
      </ul>
      <button onClick={handleGetValues}>Obtener resultados</button>
    </div>
  );
}
```

```

    };

    return (
      <Container>
        <HeaderProcess
          title={title}
          description={description}
          exercise={exercise}
        />
        <Button onClick={handleGetValues}>Lanzar dados</Button>
        {dices?.map((dice, index) => (
          <Dice key={index} value={dice} />
        )))
      </Container>
    );
  };

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

const Button = styled.button`
  margin-bottom: 20px;
  background-color: #007bff;
  color: white;
  border: 1px solid black;
  border-radius: 8px;
  width: 8vw;
  height: 4vh;
`;

```

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Componentes**

Las componentes son una característica fundamental de React. Nos permiten dividir la aplicación en trozos de nuestra interfaz visual con objetivos bien definidos: 'menú de opciones', 'formulario de búsqueda', 'ventanas de mensajes', 'tablas de datos' etc. La división de nuestra aplicación en componentes nos ayudan a reutilizar dicho código en proyectos futuros. Hasta ahora cada proyecto que hemos desarrollado ha implementado una sola componente funcional llamada 'App' (hacemos referencia a que se trata de una componente funcional, ya que versiones antiguas de React proponían por defecto una componente de clase)

Ejercicio: Confeccionar una aplicación que muestre tres dados. Plantear una componente funcional llamada 'Dado' que tenga una propiedad llamada 'valor' a la cual le llega el dato a mostrar. Componente principal 'ThreeDices' y dentro de la misma definiremos 3 componentes de tipo 'Dado'

Lanzar dados

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Componentes**

Las componentes son una característica fundamental de React. Nos permiten dividir la aplicación en trozos de nuestra interfaz visual con objetivos bien definidos: 'menú de opciones', 'formulario de búsqueda', 'ventanas de mensajes', 'tablas de datos' etc. La división de nuestra aplicación en componentes nos ayudan a reutilizar dicho código en proyectos futuros. Hasta ahora cada proyecto que hemos desarrollado ha implementado una sola componente funcional llamada 'App' (hacemos referencia a que se trata de una componente funcional, ya que versiones antiguas de React proponían por defecto una componente de clase)

Ejercicio: Confeccionar una aplicación que muestre tres dados. Plantear una componente funcional llamada 'Dado' que tenga una propiedad llamada 'valor' a la cual le llega el dato a mostrar. Componente principal 'ThreeDices' y dentro de la misma definiremos 3 componentes de tipo 'Dado'

Lanzar dados

1

3

4

## Componentes: propiedades

El elemento fundamental que tenemos para comunicarnos con una componente son las propiedades. Mediante las propiedades podemos enviar datos a la componente para que los muestre.

En el concepto anterior vimos como pasar un valor entero a la componente Dado mediante la sintaxis (empleando una variable de estado con el API de de Hook de React)

Luego dentro de la componente 'Dado' accedemos al valor pasado mediante el parámetro 'props'.

Podemos pasar cualquier tipo de datos a una componente, no solo tipos primitivos como enteros, reales, string.

### Ejercicio 5

Confeccionar una aplicación que permita ingresar por teclado dos enteros y nos muestre la suma. Agregar a una lista todas las operaciones ejecutadas hasta este momento.

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

const title = "Componentes: propiedades";
const description =
  "El elemento fundamental que tenemos para comunicarnos con una componente son las propiedades. Mediante las propiedades podemos enviar datos a la componente para que los muestre. Podemos pasar cualquier tipo de datos a una componente, no solo tipos primitivos como enteros, reales, string.";

const exercise =
  "Confeccionar una aplicación que permita ingresar por teclado dos enteros y nos muestre la suma. Agregar a una lista todas las operaciones ejecutadas hasta este momento.";

export const ListSum = () => {
  const [records, setRecords] = useState([]);

  const sum = (e) => {
    e.preventDefault();
    const value1 = e.target.param1?.value;
    const value2 = e.target.param2?.value;

    if (value1.length === 0 || value2.length === 0) {
      return;
    }

    const response = `${value1} + ${value2}= ${
      parseInt(value1) + parseInt(value2)
    }`;
    setRecords([...records, response]);
  };
}
```

```

)`;

setRecords((prevRecord) => [...prevRecord, response]);
e.target.param1.value = "";
e.target.param2.value = "";
};

return (
  <Container>
    <HeaderProcess
      title={title}
      description={description}
      exercise={exercise}
    />
    <Form onSubmit={sum}>
      <Input type="number" name="param1" placeholder="Ingrese primer valor" />
      <Input
        type="number"
        name="param2"
        placeholder="Ingrese segundo valor"
      />
      <Button>Sumar</Button>
    </Form>
    <ul>
      {records.map((record, index) => (
        <Li key={index}>{record}</Li>
      )));
    </ul>
  </Container>
);
};

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

const Form = styled.form`
  display: flex;
  flex-direction: column;
`;

const Input = styled.input`
  display: flex;
  margin: 10px;
  font-size: 20px;
  width: 20vw;
`;

const Button = styled.button`
  font-size: 20px;
  background-color: #007bff;
  color: white;
  width: 20vw;
  margin-left: 15px;
`;

const Li = styled.li`
  font-size: 20px;

```

```
width: 20vw;  
text-align: left;  
`;
```

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

## Componentes: propiedades

El elemento fundamental que tenemos para comunicarnos con una componente son las propiedades. Mediante las propiedades podemos enviar datos a la componente para que los muestre. Podemos pasar cualquier tipo de datos a una componente, no solo tipos primitivos como enteros, reales, string.

Confeccionar una aplicación que permita ingresar por teclado dos enteros y nos muestre la suma. Agregar a una lista todas las operaciones ejecutadas hasta este momento.

Ingrese primer valor  
Ingrese segundo valor  
Sumar

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

## Componentes: propiedades

El elemento fundamental que tenemos para comunicarnos con una componente son las propiedades. Mediante las propiedades podemos enviar datos a la componente para que los muestre. Podemos pasar cualquier tipo de datos a una componente, no solo tipos primitivos como enteros, reales, string.

Confeccionar una aplicación que permita ingresar por teclado dos enteros y nos muestre la suma. Agregar a una lista todas las operaciones ejecutadas hasta este momento.

1555  
2500  
Sumar

- 100 + 200= 300
- 450 + 350= 800

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

## Componentes: propiedades

El elemento fundamental que tenemos para comunicarnos con una componente son las propiedades. Mediante las propiedades podemos enviar datos a la componente para que los muestre. Podemos pasar cualquier tipo de datos a una componente, no solo tipos primitivos como enteros, reales, string.

Confeccionar una aplicación que permita ingresar por teclado dos enteros y nos muestre la suma. Agregar a una lista todas las operaciones ejecutadas hasta este momento.

Ingrese primer valor  
Ingrese segundo valor  
Sumar

- 100 + 200= 300
- 450 + 350= 800
- 1555 + 2500= 4055

## Hook de efecto (función useEffect)

Vimos en un concepto anterior los hook más utilizados que son los de variables de estado, ahora en orden de uso presentaremos los hook de efectos.

Recordemos que los hook solo pueden ser utilizados cuando implementamos componentes funcionales, que es la metodología más actual y propuesta por los creadores de la librería React.

El Hook de efecto permite llevar a cabo efectos secundarios en componentes funcionales, ejemplos de estos efectos son:

Actualizaciones manuales del DOM.

Suscribir y desuscribir a eventos (ej. addEventListener, removeListener)

Peticiones de datos a un servidor (ej. API fetch)

La sintaxis básica la podemos ver en el siguiente código:

```
import { useEffect } from "react";
function App() {
  useEffect(() => console.log("ejecución de useEffect"))
  return (
    <div>
      Hola Mundo
    </div>
  );
}

export default App
```

Debemos importar la función:

```
import { useEffect } from "react";
```

Y en su sintaxis más sencilla pasamos a la función useEffect una función flecha:

```
useEffect(() => console.log("ejecución de useEffect"))
```

Sintaxis de la función useEffect

- La función useEffect tiene dos parámetros: el primero una función y el segundo un array cuyos valores serán variables de las que depende nuestro algoritmo que implementa la función que le pasamos (este arreglo es opcional como vemos en el ejemplo codificado)
- La función useEffect se ejecuta en cada renderizado, inclusive en el primero.
- Podemos llamar a la función useEffect más de una vez y crear varios hook de efecto.

- Está diseñado para que si la función que pasamos por parámetro retorna otra función, React va a ejecutar dicha función cuando se desmonte el componente del DOM (normalmente en esta función liberamos recursos como desuscribirse a eventos)

## Ejercicio 6

Actualizar en tiempo real el título de la página con los caracteres ingresados en un control input. Para esto debemos acceder directamente al DOM.

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

const title = "Hook de efecto (función useEffect)";
const description =
  "La función useEffect se ejecuta en cada renderizado, inclusive en el primero. Podemos llamar a la función useEffect más de una vez y crear varios hook de efecto. Está diseñado para que si la función que pasamos por parámetro retorna otra función, React va a ejecutar dicha función cuando se desmonte el componente del DOM (normalmente en esta función liberamos recursos como desuscribirse a eventos). Actualizar en tiempo real el título de la página con los caracteres ingresados en un control input. Para esto debemos acceder directamente al DOM.";
const exercise =
  "Actualizar en tiempo real el título de la página con los caracteres ingresados en un control input. Para esto debemos acceder directamente al DOM.";

export const PageTitle = () => {
  const [text, setText] = useState();

  const handleText = (e) => {
    document.title = e.target.value;
    setText(e.target.value);
  };

  return (
    <Container>
      <HeaderProcess
        title={title}
        description={description}
        exercise={exercise}
      />
      <Input
        type="text"
        value={text}
        onChange={handleText}
        placeholder="Ingrese título"
      />
      <P>{text}</P>
    </Container>
  );
};

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

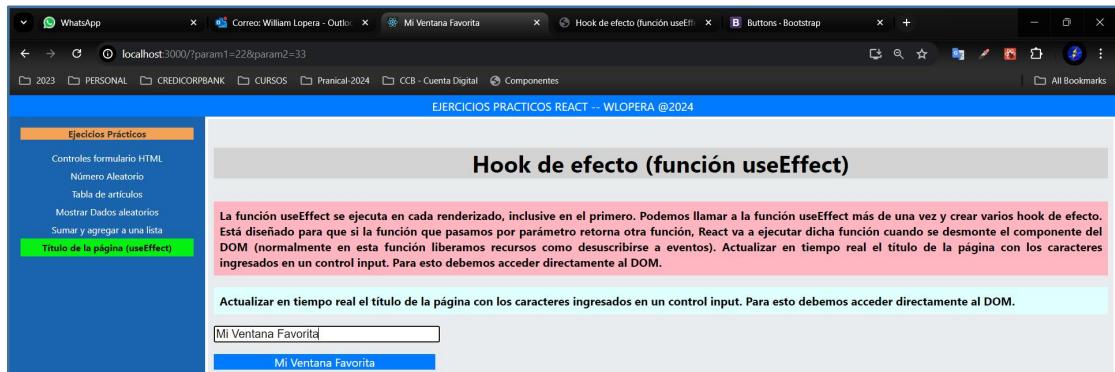
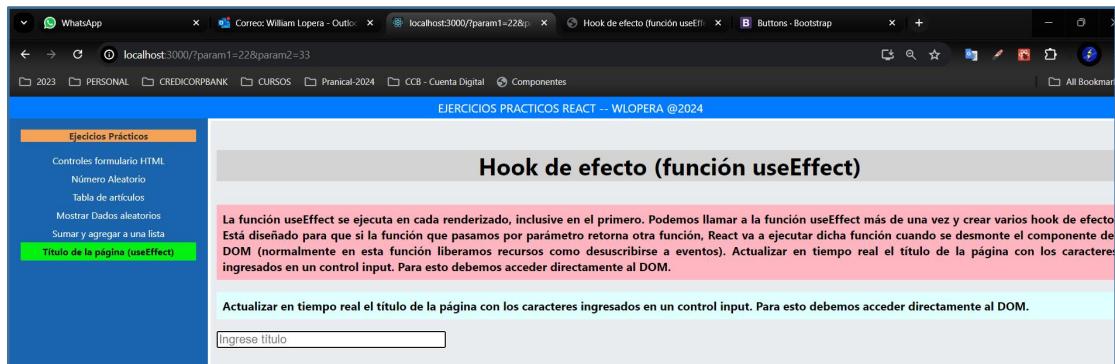
```

    align-items: left;
  `;

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

const P = styled.p`
  font-size: 20px;
  background-color: #007bff;
  color: white;
  width: 20vw;
`;

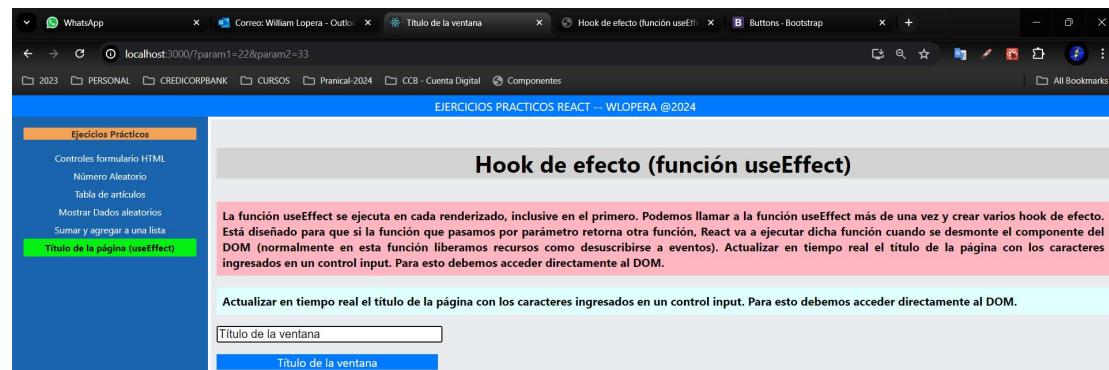
```



Uso de UseEffect. Al modificar el estado “text” se dispara el useeffect y actualiza el título de la página

```
...
export const PageTitle = () => {
  const handleText = (e) => {
    setText(e.target.value);
  };

  useEffect(() => {
    document.title = text;
  }, [text]);
...
}
```



## Ejercicio 7

Crear una componente que muestre por pantalla la coordenada donde se encuentra la flecha del mouse. En la componente principal disponer un botón para que muestre u oculte las coordenadas.

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

const title = "Hook de efecto (función useEffect)";
const description =
  "La función useEffect se ejecuta en cada renderizado, inclusive en el primero. Podemos llamar a la función useEffect más de una vez y crear varios hook de efecto. Está diseñado para que si la función que pasamos por parámetro retorna otra función, React va a ejecutar dicha función cuando se desmonte el componente del DOM (normalmente en esta función liberamos recursos como desuscribirse a eventos). Actualizar en tiempo real el título de la página con los caracteres ingresados en un control input. Para esto debemos acceder directamente al DOM.";
const exercise =
  "Crear una componente que muestre por pantalla la coordenada donde se encuentra la flecha del mouse. En la componente principal disponer un botón para que muestre u oculte las coordenadas.";

export const MouseMoveCursor = () => {
  const [position, setPosition] = useState({ x: 0, y: 0 });
  ...
}
```

```

const [visible, setVisible] = useState(true);

const handlePosition = (e) => {
  setPosition({ x: e.clientX, y: e.clientY });
};

useEffect(() => {
  window.addEventListener("mousemove", handlePosition);

  return () => {
    window.removeEventListener("mousemove", handlePosition);
    console.log("Borrado registro de eventos");
  };
}, []);

return (
  <Container>
    <HeaderProcess
      title={title}
      description={description}
      exercise={exercise}
    />
    {visible && (
      <P>
        {position.x} - {position.y}
      </P>
    )}
    <Button onClick={() => setVisible((prev) => !prev)}>
      {visible ? "Ocultar Coordenadas" : "Mostrar Coordenadas"}
    </Button>
  </Container>
);
};

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

const P = styled.p`
  font-size: 20px;
  background-color: #28a745;
  color: white;
`;

const Button = styled.button`
  font-size: 20px;
  background-color: #007bff;
  color: white;
`;

```

A la función useEffect le pasamos dependencias vacías '[]', para que se ejecute una sola vez y se utiliza el return para desmontar el uso del mousemove cada vez que se inicie el useEffect

## Peticiones con la API fetch en React

Cuando necesitamos hacer peticiones a un servidor web podemos utilizar el API fetch de Javascript. Nos permite obtener en forma asíncrona recursos de un servidor web.

### Ejercicio 8

- Confeccionar una aplicación que recupere una respuesta en JSON de la dirección

```
https://scratchya.com.ar/react/datos.php
```

La estructura del archivo JSON es:

```
[  
  {  
    "codigo": 1,  
    "descripcion": "papas",  
    "precio": 12.33  
  },  
  {  
    "codigo": 2,  
    "descripcion": "manzanas",  
    "precio": 54  
  }]
```

- Luego de recuperar los datos mostrarlos en una tabla HTML

```
import React, { useEffect, useState } from "react";  
import { HeaderProcess } from "../../components/headerProcess/HeaderProcess";  
import styled from "styled-components";  
import { GenericTable } from "./GenericTable";  
  
const title = "Hook de efecto (función useEffect)";  
const description =  
  "Cuando necesitamos hacer peticiones a un servidor web podemos utilizar el API fetch de Javascript. Nos permite obtener en forma asíncrona recursos de un servidor web.";  
  
const exercise =  
  "Confeccionar una aplicación que recupere una respuesta en JSON (codigo, descripcion y precio) de la dirección: [https://scratchya.com.ar/react/datos.php]. Luego de recuperar los datos mostrarlos en una tabla HTML";  
  
const columns = [  
  {  
    header: "Código",  
    accesor: "code",  
  },  
  {  
    header: "Descripción",  
    accesor: "description",  
  },  
  {  
    header: "Precio",  
    accesor: "price",  
  },
```

```

];
};

export const ApiFetch = () => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    const getDataApi = async () => {
      setLoading(true);
      const response = await fetch("https://scratchya.com.ar/react/datos.php");
      const values = await response.json();
      const records = values.map((value) => ({
        code: value.codigo,
        description: value.descripcion,
        price: value.precio,
      }));
      // Colocar delay para simular consulta
      setTimeout(() => {
        setData(records);
        setLoading(false);
      }, 1000);
    };
    getDataApi();
  }, [setData]);

  return (
    <Container>
      <HeaderProcess
        title={title}
        description={description}
        exercise={exercise}
      />
      {loading ? (
        <P>Recuperando datos...</P>
      ) : (
        <GenericTable columns={columns} rows={data} />
      )}
    </Container>
  );
};

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

const P = styled.p`
  font-size: 25px;
  font-weight: bold;
  text-align: left;
  width: 100%;
`;

```

```

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

export const GenericTable = ({ columns, rows }) => {
  if (columns.length === 0 || rows.length === 0) {
    return null;
  }

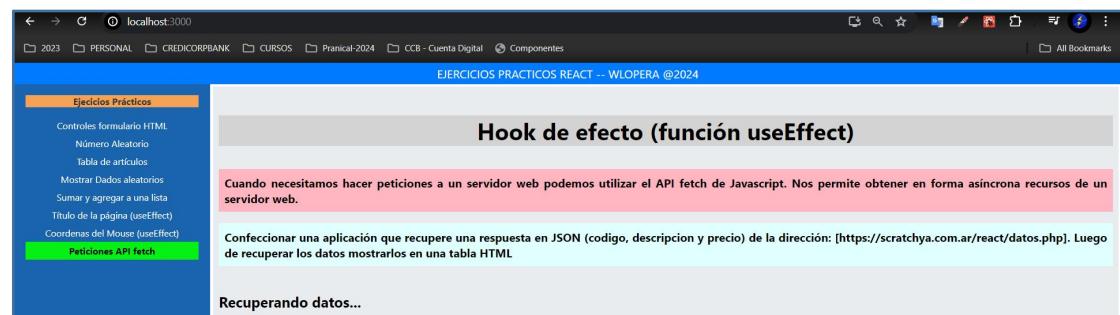
  return (
    <Table>
      <thead>
        <tr>
          {columns.map((column) => (
            <TH key={column.accesor}>{column.header}</TH>
          )))
        </tr>
      </thead>
      <tbody>
        {rows.map((row) => (
          <tr key={row.code}>
            {columns.map((column) => (
              <TD>{row[column.accesor]}</TD>
            )))
          </tr>
        )))
      </tbody>
    </Table>
  );
};

const Table = styled.table`
  border: 1px solid black;
`;

const TH = styled.th`
  border: 1px solid black;
  font-size: 20px;
  width: 10vw;
  background-color: #28a745;
  color: white;
`;

const TD = styled.td`
  border: 1px solid black;
  font-size: 16px;
  font-weight: bold;
`;

```



Código	Descripción	Precio
1	papas	34
2	manzanas	23.5
3	sandia	31

## Formularios: enlace de controles con variables de estados (Hooks de estados)

Cuando trabajamos con formulario HTML los controles almacenan el valor cargado. Con React es muy común enlazar cada control de formulario con una o más variables de estado mediante el empleo de Hook.

Este enlace nos facilita implementar validaciones de ingreso de datos inmediatamente después que el operador los carga.

### Ejercicio 9

Confeccionar un formulario HTML que solicite la carga del nombre, edad y si tiene estudios o no una persona. Utilizar controles:

```
<input type="text" />
<input type="number" />
<input type="checkbox" />
```

Almacenar en una variable de estado de la componente cada vez que ocurre un cambio en un control de formulario. Mostrar en todo momento que datos están almacenado en la variable de estado.

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

const title =
  "Formularios: enlace de controles con variables de estados (Hooks de estados)";
const description =
  "Cuando trabajamos con formulario HTML los controles almacenan el valor cargado. Con React es muy común enlazar cada control de formulario con una o más variables de estado mediante el empleo de Hook. Este enlace nos facilita implementar validaciones de ingreso de datos inmediatamente después que el operador los carga.";
const exercise =
```

```
"Confeccionar un formulario HTML que solicite la carga del nombre, edad y si tiene estudios o no una persona. [Utilizar input: text, number y check]"';

export const ControlForm = () => {
  const [dataForm, setDataForm] = useState({
    name: '',
    age: 0,
    isWorking: false,
  });

  const procesar = () => {
    alert(JSON.stringify(dataForm, null, 2));
  };

  const handleDataForm = (e) => {
    const name = e.target.name;
    const value = name === "isWorking" ? e.target.checked : e.target.value;
    setDataForm((prevData) => ({ ...prevData, [name]: value }));
  };

  return (
    <Container>
      <HeaderProcess
        title={title}
        description={description}
        exercise={exercise}
      />
      <Form onSubmit={procesar}>
        <Label>
          Ingrese nombre:
          <Input
            type="text"
            name="name"
            value={dataForm.name}
            onChange={handleDataForm}
          />
        </Label>
        <Label>
          Ingrese edad:
          <Input
            type="number"
            name="age"
            value={dataForm.age}
            onChange={handleDataForm}
          />
        </Label>
        <Label>
          Estudios:
          <Input
            type="checkbox"
            name="isWorking"
            value={dataForm.isWorking}
            onChange={handleDataForm}
          />
        </Label>
        <Button>Enviar</Button>
      </Form>
      <Line />
      <P>Datos Ingresados</P>
      <Label>Nombre: {dataForm.name}</Label>
      <Label>Edad: {dataForm.age === 0 ? "" : dataForm.age}</Label>
    
```

```
    <Label>Estudios: {dataForm.isWorking ? "Si" : "No"}</Label>
  </Container>
);
};

const Container = styled.div`  

  display: flex;  

  flex-direction: column;  

  justify-content: center;  

  align-items: center;  

`;  
  

const Form = styled.form`  

  display: flex;  

  flex-direction: column;  

  text-align: left;  

  width: 100%;  

`;  
  

const Label = styled.label`  

  font-size: 20px;  

  text-align: left;  

  width: 100%;  

  padding-bottom: 10px;  

`;  
  

const Input = styled.input``;  
  

const Button = styled.button`  

  margin-top: 8px;  

  width: 8vw;  

  background-color: #d5f5e3;  

`;  

const P = styled.p`  

  font-size: 30px;  

  font-weight: bold;  

  text-align: left;  

  width: 100%;  

  margin-bottom: 10px;  

`;  
  

const Line = styled.div`  

  margin-top: 40px;  

  width: 100%;  

  border-top: 2px solid Black;  

`;
```

EJERCICIOS PRACTICOS REACT -- WILOPERA @2024

**Ejercicios Prácticos**

- Controles formulario HTML
- Número Aleatorio
- Tabla de artículos
- Mostrar Datos aleatorios
- Sumar y agregar a una lista
- Título de la página (useEffect)
- Coordenadas del Mouse (useEffect)
- Peticiones API fetch
- Formulario**

## Formularios: enlace de controles con variables de estados (Hooks de estados)

Cuando trabajamos con formulario HTML los controles almacenan el valor cargado. Con React es muy común enlazar cada control de formulario con una o más variables de estado mediante el empleo de Hook. Este enlace nos facilita implementar validaciones de ingreso de datos inmediatamente después que el operador los carga.

Confeccionar un formulario HTML que solicite la carga del nombre, edad y si tiene estudios o no una persona. [Utilizar input: text, number y check]

Ingrese nombre:

Ingrese edad:

Estudios:

---

### Datos Ingresados

Nombre:

Edad:

Estudios:



localhost:3000 says

```
{
  "name": "William Lopera",
  "age": 50,
  "isWorking": true
}
```

OK

Formularios: enlace de controles con variables de estados (Hooks de estados)

Cuando trabajamos con formulario HTML los controles almacenan el valor cargado. Con React es muy común enlazar cada control de formulario con una o más variables de estado mediante el empleo de Hook. Este enlace nos facilita implementar validaciones de ingreso de datos inmediatamente después que el operador los carga.

Confeccionar un formulario HTML que solicite la carga del nombre, edad y si tiene estudios o no una persona. [Utilizar input: text, number y check]

Ingrese nombre:

Ingrese edad:

Estudios:

---

### Datos Ingresados

Nombre: William Lopera

Edad: 50

Estudios: Si

## Formularios: validación inmediata

Vimos en el concepto anterior lo usual en React es asociar a cada control de un formulario HTML un 'estado' en la componente. Esto nos permite inmediatamente se produzca el cambio reaccionar ya sea aceptando o rechazando dicho dato.

Cada vez que se realiza la carga de datos es un momento muy adecuado para implementar validaciones y no cambiar el 'estado' si los datos son incorrecto.

### Ejercicio 10

Confeccionar una aplicación que permita ingresar en un control 'input' de tipo 'text' solo valores binarios, es decir solo ceros y unos.

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

```

const title = "Formularios: validación inmediata ";
const description =
  "Vimos en el concepto anterior lo usual en React es asociar a cada control de un formulario HTML un 'estado' en la componente. Esto nos permite inmediatamente producir el cambio reaccionar ya sea aceptando o rechazando dicho dato. Cada vez que se realiza la carga de datos es un momento muy adecuado para implementar validaciones y no cambiar el 'estado' si los datos son incorrectos.";
const exercise =
  "Confeccionar una aplicación que permita ingresar en un control 'input' de tipo 'text' solo valores binarios, es decir solo ceros y unos.";

const binaryRegex = /^[01]+$/;

export const ValidationForm = () => {
  const [number, setNumber] = useState("");
  const [valid, setValid] = useState(true);

  const handleValue = (e) => {
    const value = e.target.value;
    if (value === "" || binaryRegex.test(value)) {
      setNumber(value);
      setValid(true);
    } else {
      setValid(false);
    }
  };

  return (
    <Container>
      <HeaderProcess
        title={title}
        description={description}
        exercise={exercise}
      />
      <Form>
        <Label>
          Ingrese un número binario:
          <input type="text" value={number} onChange={handleValue} />
        </Label>
        {!valid && <P>Solo se permiten valores binarios [0 y 1]</P>}
      </Form>
    </Container>
  );
};

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

const Form = styled.form`
  display: flex;
  flex-direction: column;
  text-align: left;
  width: 100%;
`;

const Label = styled.label`
  font-size: 20px;
`;

```

```

text-align: left;
width: 100%;
padding-bottom: 10px;
`;

const Pspan = styled.span`
  color: red;
  font-size: 18px;
  font-weight: bold;
  background-color: #fff9c4;
  width: 28vw;
  padding-left: 2px;
`;

```

EJERCICIOS PRACTICOS REACT -- WILOPERA @2024

**Formularios: validación inmediata**

Vimos en el concepto anterior lo usual en React es asociar a cada control de un formulario HTML un 'estado' en la componente. Esto nos permite inmediatamente se produzca el cambio reaccionar ya sea aceptando o rechazando dicho dato. Cada vez que se realiza la carga de datos es un momento muy adecuado para implementar validaciones y no cambiar el 'estado' si los datos son incorrecto.

Confeccionar una aplicación que permita ingresar en un control 'input' de tipo 'text' solo valores binarios, es decir solo ceros y unos.

Ingrese un número binario:

EJERCICIOS PRACTICOS REACT -- WILOPERA @2024

**Formularios: validación inmediata**

Vimos en el concepto anterior lo usual en React es asociar a cada control de un formulario HTML un 'estado' en la componente. Esto nos permite inmediatamente se produzca el cambio reaccionar ya sea aceptando o rechazando dicho dato. Cada vez que se realiza la carga de datos es un momento muy adecuado para implementar validaciones y no cambiar el 'estado' si los datos son incorrecto.

Confeccionar una aplicación que permita ingresar en un control 'input' de tipo 'text' solo valores binarios, es decir solo ceros y unos.

Ingrese un número binario:  1110001010101

EJERCICIOS PRACTICOS REACT -- WILOPERA @2024

**Formularios: validación inmediata**

Vimos en el concepto anterior lo usual en React es asociar a cada control de un formulario HTML un 'estado' en la componente. Esto nos permite inmediatamente se produzca el cambio reaccionar ya sea aceptando o rechazando dicho dato. Cada vez que se realiza la carga de datos es un momento muy adecuado para implementar validaciones y no cambiar el 'estado' si los datos son incorrecto.

Confeccionar una aplicación que permita ingresar en un control 'input' de tipo 'text' solo valores binarios, es decir solo ceros y unos.

Ingrese un número binario:  111000  
Solo se permiten valores binarios [0 y 1]

## Formularios: control textarea

Para enlazar un control HTML de tipo textarea debemos definir la propiedad value en forma similar a los controles input que hemos visto hasta ahora.

### Ejercicio 11

Mostrar un control de tipo textarea. Cada vez que se ingrese un caracter actualizar la cantidad ingresados dentro del control.

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

const title = "Formularios: control textarea";
const description =
  "Para enlazar un control HTML de tipo textarea debemos definir la propiedad value en forma similar a los controles input que hemos visto hasta ahora.";
const exercise =
  "Mostrar un control de tipo área de texto. Cada vez que se ingrese un caracter actualizar la cantidad ingresados dentro del control.";

export const TextAreaForm = () => {
  const [text, setText] = useState("");

  const handleText = (e) => {
    setText(e.target.value);
  };

  return (
    <div>
      <HeaderProcess
        title={title}
        description={description}
        exercise={exercise}
      />
      <form>
        <textarea cols="100" rows="10" value={text} onChange={handleText} />
        <p>Cantidad de caracteres ingresados: {text.length}</p>
      </form>
    </div>
  );
};
```

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Ejercicios Prácticos**

- Controles formulario HTML
- Número Aleatorio
- Tabla de artículos
- Mostrar Datos aleatorios
- Sumar y agregar a una lista
- Título de la página (useEffect)
- Coordenadas del Mouse (useEffect)
- Peticiones API fetch
- Formulario enlace de controles con variables de estados
- Formularios: validación inmediata
- Formularios: control textarea**

## Formularios: control textarea

Para enlazar un control HTML de tipo textarea debemos definir la propiedad value en forma similar a los controles input que hemos visto hasta ahora.

Mostrar un control de tipo área de texto. Cada vez que se ingrese un carácter actualizar la cantidad ingresados dentro del control.

Cantidad de caracteres ingresados: 0

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Ejercicios Prácticos**

- Controles formulario HTML
- Número Aleatorio
- Tabla de artículos
- Mostrar Datos aleatorios
- Sumar y agregar a una lista
- Título de la página (useEffect)
- Coordenadas del Mouse (useEffect)
- Peticiones API fetch
- Formulario enlace de controles con variables de estados
- Formularios: validación inmediata
- Formularios: control textarea**

## Formularios: control textarea

Para enlazar un control HTML de tipo textarea debemos definir la propiedad value en forma similar a los controles input que hemos visto hasta ahora.

Mostrar un control de tipo área de texto. Cada vez que se ingrese un carácter actualizar la cantidad ingresados dentro del control.

Prueba de área de texto  
Mostrar un control de tipo área de texto. Cada vez que se ingrese un carácter actualizar la cantidad ingresados dentro del control.]

Cantidad de caracteres ingresados: 155

## Formularios: control select

Otro control que debemos ver como lo trabaja la librería React es el 'select'.

Veamos como procesar un control HTML select de selección única. Debemos definir la propiedad value directamente al elemento 'select' y asociarlo con el 'estado'.

### Ejercicio 12

Mostrar en un control de tipo 'select' los días de la semana. Cuando el operador lo selecciona mostrar cual se seleccionó en la página.

```
import React, { useState } from "react";

import { HeaderProcess } from "../../components/headerProcess/HeaderProcess";

const title = "Formularios: control select";

const description =
  "Otro control que debemos ver como lo trabaja la librería React es el 'select'. Veamos como procesar un control HTML select de selección única. Debemos definir la propiedad value directamente al elemento 'select' y asociarlo con el 'estado'.";

const exercise =
  "Mostrar en un control de tipo 'select' los días de la semana. Cuando el operador lo selecciona mostrar cual se seleccionó en la página.";

const records = [
  { value: "monday", label: "Lunes" },
  { value: "tuesday", label: "Martes" },
  { value: "wednesday", label: "Miércoles" },
  { value: "thursday", label: "Juves" },
]
```

```

        { value: "friday", label: "Viernes" },
        { value: "saturday", label: "Sábado" },
        { value: "sunday", label: "Domingo" },
    ];
}

export const SelectForm = () => {
    const [day, setDay] = useState("monday");

    const handleDay = (e) => {
        setDay(e.target.value);
    };

    const getDay = () => {
        return records.filter((record) => record.value === day)[0].label;
    };

    return (
        <div>
            <HeaderProcess
                title={title}
                description={description}
                exercise={exercise}>
            />
            <form>
                <select
                    value={day}
                    onChange={handleDay}
                    style={{ width: "10vw", height: "4vh", fontSize: "18px" }}>
                    {records.map((record) => (
                        <option key={record.value} value={record.value}>
                            {record.label}
                        </option>
                    ))}
                </select>
                <p style={{ fontSize: "20px" }}>Día seleccionado: {getDay()}</p>
            </form>
        </div>
    );
};

```

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

## Formularios: control select

Otro control que debemos ver como lo trabaja la librería React es el 'select'. Veamos como procesar un control HTML select de selección única. Debemos definir la propiedad value directamente al elemento 'select' y asociarlo con el 'estado'.

Mostrar en un control de tipo 'select' los días de la semana. Cuando el operador lo selecciona mostrar cual se seleccionó en la página.



EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

## Formularios: control select

Otro control que debemos ver como lo trabaja la librería React es el 'select'. Veamos como procesar un control HTML select de selección única. Debemos definir la propiedad value directamente al elemento 'select' y asociarlo con el 'estado'.

Mostrar en un control de tipo 'select' los días de la semana. Cuando el operador lo selecciona mostrar cual se seleccionó en la página.

Sábado

Día seleccionado: Sábado

## Mostrar imágenes localizadas en la carpeta public

La primer forma de trabajar con imágenes es React, es localizando los mismos en la carpeta public o una subcarpeta de la misma.

Si abrimos el archivo index.html que tiene todo proyecto en React podremos ver que el archivo favicon.ico se localiza en la carpeta public:

```
<link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
```

Luego cuando compilamos la aplicación React se remplaza la variable de entorno PUBLIC\_URL y queda según el valor inicializado (si no hacemos nada cuando creamos un proyecto luego tiene un path nulo):

```
<link rel="icon" href="/favicon.ico" />
```

Con un problema.

### Ejercicio 13

Crear una aplicación que muestre una bandera de un país Latinoamericano una a la vez, permitir con dos botones cambiar a la siguiente o anterior bandera.

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

const title = "Mostrar imágenes localizadas en la carpeta public";

const description =
  "La primer forma de trabajar con imágenes es React, es localizando los mismos en la carpeta public o una subcarpeta de la misma.";

const exercise =
  "Crear una aplicación que muestre una bandera de un país una a la vez, permitir con dos botones cambiar a la siguiente o anterior bandera.";

const flagCountries = [
  "argentina",
  "brasil",
  "bolivia",
  "chile",
  "colombia",
  "ecuador",
  "paraguay",
  "peru",
  "uruguay",
  "venezuela",
];

export const ShowImage = () => {
  const [positionFlag, setPositionFlag] = useState(0);
```

```
const previousFlag = () => {
  if (positionFlag === 0) {
    return;
  }
  setPositionFlag((currentPos) => currentPos - 1);
};

const nextFlag = () => {
  if (positionFlag === flagCountries.length - 1) {
    return;
  }
  setPositionFlag((currentPos) => currentPos + 1);
};

const flag = flagCountries[positionFlag];

return (
  <Container>
    <HeaderProcess
      title={title}
      description={description}
      exercise={exercise}
    />
    <Div>
      <h2>Banderas de países Latinoamericanos</h2>
      <Img src={process.env.PUBLIC_URL + "/images/" + flag + ".png"} />
      <DivImage>
        <input type="button" value={"<"} onClick={previousFlag} />
        <input type="button" value={">"} onClick={nextFlag} />
      </DivImage>
      <P>{flag.toUpperCase()}</P>
    </Div>
  </Container>
);
};

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

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

const DivImage = styled.div`
  display: flex;
  flex-direction: row;
  margin-top: 20px;
`;

const Img = styled.img`
  width: 20vw;
  height: 25vh;
  border: 2px solid grey;
`;
```

```
`;
```

```
const P = styled.p`  
  font-size: 16px;  
  font-weight: 500;  
`;
```

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Ejercicios Prácticos**

- Controles formulario HTML
- Número Aleatorio
- Tabla de artículos
- Mostrar Datos aleatorios
- Sumar y agregar a una lista
- Título de la página (useEffect)
- Coordenadas del Mouse (useEffect)
- Peticiones API fetch
- Formulario enlace de controles con variables de estados
- Formularios: validación inmediata
- Formularios: control textarea
- Formularios: control select
- Mostrar imágenes carpeta pública**

**Mostrar imágenes localizadas en la carpeta public**

La primer forma de trabajar con imágenes es React, es localizando los mismos en la carpeta public o una subcarpeta de la misma.

Crear una aplicación que muestre una bandera de un país una a la vez, permitir con dos botones cambiar a la siguiente o anterior bandera.

**Banderas de países Latinoamericanos**



< >

ARGENTINA

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Ejercicios Prácticos**

- Controles formulario HTML
- Número Aleatorio
- Tabla de artículos
- Mostrar Datos aleatorios
- Sumar y agregar a una lista
- Título de la página (useEffect)
- Coordenadas del Mouse (useEffect)
- Peticiones API fetch
- Formulario enlace de controles con variables de estados
- Formularios: validación inmediata
- Formularios: control textarea
- Formularios: control select
- Mostrar imágenes carpeta pública**

**Mostrar imágenes localizadas en la carpeta public**

La primer forma de trabajar con imágenes es React, es localizando los mismos en la carpeta public o una subcarpeta de la misma.

Crear una aplicación que muestre una bandera de un país una a la vez, permitir con dos botones cambiar a la siguiente o anterior bandera.

**Banderas de países Latinoamericanos**



< >

VENEZUELA

## Formularios: control select dependiente de otro control select

Una situación común en un formulario web es cuando un control select depende del elemento seleccionado de otro control select.

### Ejercicio 14

Vamos a crear un control select que muestre distintos rubros (por ejemplo: Microprocesadores, Placas de video, Gabinetes etc), luego cuando el operador selecciona un rubro, se debe actualizar otro control select que muestra todos los artículos que pertenecen a dicho rubro, por ejemplo si seleccionar Microprocesadores el segundo select puede mostrar Intel Core 15, Intel Core 17 etc.

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

const title = "Formularios: control select dependiente de otro control select";
const description =
  "Cuando trabajamos con formulario HTML los controles almacenan el valor cargado. Con React es muy común enlazar cada control de formulario con una o más variables de estado mediante el empleo de Hook. Este enlace nos facilita implementar validaciones de ingreso de datos inmediatamente después que el operador los carga. ";
const exercise =
  "Confeccionar un formulario HTML que solicite la carga del nombre, edad y si tiene estudios o no una persona. [Utilizar input: text, number y check]";

export const NestedCombo = () => {
  const [item, setItem] = useState(ITEMS[0]);
  const [products, setProducts] = useState(
    PRODUCTS.filter((product) => product.codigorubro === item.codigo)
  );
  const [article, setArticle] = useState(products[0]);

  const handleItem = (e) => {
    const idItem = Number.parseInt(e.target.value);
    const filterItem = ITEMS.find((item) => item.codigo === idItem);
    setItem(filterItem);

    const filterProducts = PRODUCTS.filter(
      (item) => item.codigorubro === idItem
    );
    setProducts(filterProducts);

    setArticle({ ...filterProducts[0], rubro: filterItem.nombre });
  };

  const handleProduct = (e) => {
    const idProduct = Number.parseInt(e.target.value);
    const filterProducts = PRODUCTS.find((item) => item.codigo === idProduct);
    setArticle({ ...filterProducts, rubro: item.nombre });
  };
}
```

```

    return (
      <Container>
        <HeaderProcess
          title={title}
          description={description}
          exercise={exercise}
        />
        <Select value={item.codigo} onChange={handleItem}>
          {ITEMS.map((record) => (
            <Option key={record.codigo} value={record.codigo}>
              {record.nombre}
            </Option>
          )))
        </Select>
        <Select value={article.codigo} onChange={handleProduct}>
          {products.map((record) => (
            <Option key={record.codigo} value={record.codigo}>
              {record.nombre}
            </Option>
          )))
        </Select>
        <Ul>
          <li>
            Rubro: <Span>{article.rubro}</Span>
          </li>
          <li>
            ARTículo: <Span>{article.nombre}</Span>
          </li>
          <li>
            Precio: <Span>{article.precio}</Span>
          </li>
        </Ul>
      </Container>
    );
  );

```

```

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

```

```

const Select = styled.select`
  font-size: 15px;
  height: 4vh;
  margin-bottom: 10px;
  padding-left: 10px;
  padding-right: 10px;
`;
const Option = styled.option`
  font-size: 15px;
`;
const Ul = styled.ul`
  text-align: left;
  font-size: 20px;
`;
const Span = styled.span`
  font-size: 20px;
  font-weight: bold;
`;

```

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Ejercicios Prácticos**

- Controles formulario HTML
- Número Aleatorio
- Tabla de artículos
- Mostrar Datos aleatorios
- Sumar y agregar a una lista
- Título de la página (useEffect)
- Coordenadas del Mouse (useEffect)
- Peticiones API fetch
- Formulario enlace de controles con variables de estados
- Formularios: validación inmediata
- Formularios: control textarea
- Formularios: control select
- Mostrar imágenes carpeta pública
- Combos anidados

## Formularios: control select dependiente de otro control select

Cuando trabajamos con formulario HTML los controles almacenan el valor cargado. Con React es muy común enlazar cada control de formulario con una o más variables de estado mediante el empleo de Hook. Este enlace nos facilita implementar validaciones de ingreso de datos inmediatamente después que el operador los carga.

Confeccionar un formulario HTML que solicite la carga del nombre, edad y si tiene estudios o no una persona. [Utilizar input: text, number y check]

- Rubro:
- Artículo: **Micróprocesador Intel Core i5**
- Precio: **1000**

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Ejercicios Prácticos**

- Controles formulario HTML
- Número Aleatorio
- Tabla de artículos
- Mostrar Datos aleatorios
- Sumar y agregar a una lista
- Título de la página (useEffect)
- Coordenadas del Mouse (useEffect)
- Peticiones API fetch
- Formulario enlace de controles con variables de estados
- Formularios: validación inmediata
- Formularios: control textarea
- Formularios: control select
- Mostrar imágenes carpeta pública
- Combos anidados

## Formularios: control select dependiente de otro control select

Cuando trabajamos con formulario HTML los controles almacenan el valor cargado. Con React es muy común enlazar cada control de formulario con una o más variables de estado mediante el empleo de Hook. Este enlace nos facilita implementar validaciones de ingreso de datos inmediatamente después que el operador los carga.

Confeccionar un formulario HTML que solicite la carga del nombre, edad y si tiene estudios o no una persona. [Utilizar input: text, number y check]

- Rubro:
- Artículo: **Tarjeta de video MSI GeForce GTX 1080**
- Precio: **790**

EJERCICIOS PRACTICOS REACT -- WLOPERA @2024

**Ejercicios Prácticos**

- Controles formulario HTML
- Número Aleatorio
- Tabla de artículos
- Mostrar Datos aleatorios
- Sumar y agregar a una lista
- Título de la página (useEffect)
- Coordenadas del Mouse (useEffect)
- Peticiones API fetch
- Formulario enlace de controles con variables de estados
- Formularios: validación inmediata
- Formularios: control textarea
- Formularios: control select
- Mostrar imágenes carpeta pública
- Combos anidados

## Formularios: control select dependiente de otro control select

Cuando trabajamos con formulario HTML los controles almacenan el valor cargado. Con React es muy común enlazar cada control de formulario con una o más variables de estado mediante el empleo de Hook. Este enlace nos facilita implementar validaciones de ingreso de datos inmediatamente después que el operador los carga.

Confeccionar un formulario HTML que solicite la carga del nombre, edad y si tiene estudios o no una persona. [Utilizar input: text, number y check]

- Rubro: **Tarjetas de video**
- Artículo: **Tarjeta de video MSI GeForce GTX 1080**
- Precio: **790**

