
ICARO
Nuestro compromiso es con el futuro.

React

Clase 5

React: Implementaciones

Temas:

- Condiciones
- Fetch
- Mapeo/Arrays
- Custom Hooks

React

Implementaciones: Condiciones

Implementaciones: Condiciones

Renderizado condicional:

Implementaciones: Condiciones

Renderizado condicional:

- Utilización de estados.

Implementaciones: Condiciones

Renderizado condicional:

- Utilización de estados.
- Utilización de hooks.

Implementaciones: Condiciones

Renderizado condicional:

- Utilización de estados.
- Utilización de hooks.
- Utilización de operadores ternarios.

Implementaciones: Condiciones

Sobre el renderizado condicional:

Implementaciones: Condiciones

Sobre el renderizado condicional:

- *Puede ser pequeño o grande, para los casos en que se necesite armar rutas, se recomienda usar librerías.*

Implementaciones: Condiciones

Sobre el renderizado condicional:

- *Puede ser pequeño o grande, para los casos en que se necesite armar rutas, se recomienda usar librerías.*

- *A la hora de trabajar, es raro que se use otra sintaxis que no sea la de operadores ternarios.*

Implementaciones: Condiciones

Sobre el renderizado condicional:

- *Puede ser pequeño o grande, para los casos en que se necesite armar rutas, se recomienda usar librerías.*
- *A la hora de trabajar, es raro que se use otra sintaxis que no sea la de operadores ternarios.*
- *Si bien a veces es considerado como una “mala práctica”, los ternarios se pueden encadenar.*

Implementaciones: Condiciones

Ejemplo condicionales:

Implementaciones: Condiciones

Ejemplo condicionales:

```
1.  import { useState } from "react"
2.  function IntentosComponent() {
3.    const [intentos, setIntentos] = useState(5)
4.    return (
5.      <div>
6.        <p>
7.          {intentos < 1
8.            ? "Ya no quedan intentos"
9.            : intentos === 1
10.           ? `Le queda ${intentos} intento.`
11.           : `Le quedan ${intentos} intentos.`}
12.       </p>
13.       <button disabled={intentos < 1} onClick={() => setIntentos(intentos - 1)}>
14.         Intentar
15.       </button>
16.     </div>
17.   )
18. }
19. export default IntentosComponent
```

Implementaciones: Condiciones

Ejemplo de condicionales:

```
1.  import { useState } from "react"
2.  function IntentosComponent() {
3.    const [intentos, setIntentos] = useState(5) // Configuramos el hook comenzando la cuenta desde 5.
4.    return (
5.      <div>
6.        <p>
7.          {intentos < 1
8.            ? "Ya no quedan intentos"
9.            : intentos === 1
10.           ? `Le queda ${intentos} intento.`
11.           : `Le quedan ${intentos} intentos.`}
12.       </p>
13.       <button disabled={intentos < 1} onClick={() => setIntentos(intentos - 1)}>
14.         Intentar
15.       </button>
16.     </div>
17.   )
18. }
19. export default IntentosComponent
```

Implementaciones: Condiciones

Ejemplo de condicionales:

```jsx
import { useState } from "react"
function IntentosComponent() {
  const [intentos, setIntentos] = useState(5)
  return (
    <div>
      <p>
        {intentos < 1 // Mensaje cuando nos quedamos si intentos.
          ? "Ya no quedan intentos"
          : intentos === 1
          ? `Le queda ${intentos} intento.`
          : `Le quedan ${intentos} intentos.`}
      </p>
      <button disabled={intentos < 1} onClick={() => setIntentos(intentos - 1)}>
        Intentar
      </button>
    </div>
  )
}
export default IntentosComponent
```

Implementaciones: Condiciones

Ejemplo de condicionales:

```jsx
import { useState } from "react"
function IntentosComponent() {
  const [intentos, setIntentos] = useState(5)
  return (
    <div>
      <p>
        {intentos < 1
          ? "Ya no quedan intentos"
          : intentos === 1 // Mensaje en singular.
          ? `Le queda ${intentos} intento.`
          : `Le quedan ${intentos} intentos.`}
      </p>
      <button disabled={intentos < 1} onClick={() => setIntentos(intentos - 1)}>
        Intentar
      </button>
    </div>
  )
}
export default IntentosComponent
```

Implementaciones: Condiciones

Ejemplo de condicionales:

```jsx
import { useState } from "react"
function IntentosComponent() {
  const [intentos, setIntentos] = useState(5)
  return (
    <div>
      <p>
        {intentos < 1
          ? "Ya no quedan intentos"
          : intentos === 1
          ? `Le queda ${intentos} intento.` // mensaje en plural.
          : `Le quedan ${intentos} intentos.`}
      </p>
      <button disabled={intentos < 1} onClick={() => setIntentos(intentos - 1)}>
        Intentar
      </button>
    </div>
  )
}
export default IntentosComponent
```

React

Implementaciones: Fetch

Implementaciones: Fetch

Algunas ideas sobre los llamados HTTP…

Implementaciones: Fetch

Algunas ideas sobre los llamados HTTP…

- Comunicación con el backend / Firma. (JSON)

Implementaciones: Fetch

Algunas ideas sobre los llamados HTTP…

- Comunicación con el backend / Firma. (JSON)
- Son asincrónicos.

Implementaciones: Fetch

Algunas ideas sobre los llamados HTTP…

- Comunicación con el backend / Firma. (JSON)
- Son asincrónicos.
- Recepción de información. (GET)

Implementaciones: Fetch

Algunas ideas sobre los llamados HTTP…

- Comunicación con el backend / Firma. (JSON)
- Son asincrónicos.
- Recepción de información. (GET)
- Envío de información. (POST)

Implementaciones: Fetch

Firma:

Implementaciones: Fetch

Firma:

- *Por “firma” nos referimos al contrato que tendrá la comunicación de los datos.*

Implementaciones: Fetch

Firma:

- *Por “firma” nos referimos al contrato que tendrá la comunicación de los datos.*

- *Por lo general la “firma” se pacta o acuerda entre los desarrolladores en base a los requerimientos del proyecto.*

Implementaciones: Fetch

Firma:

- *Por “firma” nos referimos al contrato que tendrá la comunicación de los datos.*
- *Por lo general la “firma” se pacta o acuerda entre los desarrolladores en base a los requerimientos del proyecto.*
- *El manejo de JSONs es fundamental.*

Implementaciones: Fetch

Ejemplo JSON:

Implementaciones: Fetch

Ejemplo JSON:

```
[
    { "id": 1, "alumno": "Gaby", "edad": 24 },
    { "id": 2, "alumno": "Andy", "edad": 22 },
    { "id": 3, "alumno": "Cris", "edad": 18 },
    { "id": 4, "alumno": "Dany", "edad": 31 },
    { "id": 5, "alumno": "Alex", "edad": 41 },
    { "id": 6, "alumno": "Emma", "edad": 30 }
]
```

Implementaciones: Fetch

Ejemplo JSON:

```
[ // Este JSON lo podemos colocar en un servidor de Express o en json-server para seguir los ejemplos
    { "id": 1, "alumno": "Gaby", "edad": 24 },
    { "id": 2, "alumno": "Andy", "edad": 22 },
    { "id": 3, "alumno": "Cris", "edad": 18 },
    { "id": 4, "alumno": "Dany", "edad": 31 },
    { "id": 5, "alumno": "Alex", "edad": 41 },
    { "id": 6, "alumno": "Emma", "edad": 30 }
]
```

Implementaciones: Fetch

Ejemplo objeto JS:

Implementaciones: Fetch

Ejemplo objeto JS:

```
1.  const ejemploFirma = [
2.      { id: 1, alumno: "Gaby", edad: 24 },
3.      { id: 2, alumno: "Andy", edad: 22 },
4.      { id: 3, alumno: "Cris", edad: 18 },
5.      { id: 4, alumno: "Dany", edad: 31 },
6.      { id: 5, alumno: "Alex", edad: 41 },
7.      { id: 6, alumno: "Emma", edad: 30 },
8.  ]
```

Implementaciones: Fetch

Ejemplo objeto JS:

```
1.  const ejemploFirma = [ // En este caso recibimos un array
2.      { id: 1, alumno: "Gaby", edad: 24 }, // Dentro del array tenemos objetos
3.      { id: 2, alumno: "Andy", edad: 22 },
4.      { id: 3, alumno: "Cris", edad: 18 },
5.      { id: 4, alumno: "Dany", edad: 31 },
6.      { id: 5, alumno: "Alex", edad: 41 },
7.      { id: 6, alumno: "Emma", edad: 30 },
8.  ]
```

Implementaciones: Fetch

Asincronía:

Implementaciones: Fetch

Asincronía:

- *En React tenemos un hook para manejar información que llega de forma asincrónica: useEffect.*

Implementaciones: Fetch

Asincronía:

- *En React tenemos un hook para manejar información que llega de forma asincrónica: useEffect.*

- *Para enviar información (request) muchas veces es útil delegar la terea a los eventos.*

Implementaciones: Fetch

Asincronía:

- *En React tenemos un hook para manejar información que llega de forma asincrónica: useEffect.*
- *Para enviar información (request) muchas veces es útil delegar la terea a los eventos.*
- *Existen librerías como ReactQuery que manejan muy bien los casos comunes de los llamados HTTP.*

Implementaciones: Fetch

Llamados GET:

Implementaciones: Fetch

Llamados GET:

- *Al desarrollar en front-end los llamados del tipo “GET” suelen tener repercusiones en la UI.*

Implementaciones: Fetch

Llamados GET:

- *Al desarrollar en front-end los llamados del tipo “GET” suelen tener repercusiones en la UI.*

- *El disparo del llamado puede hacerse cuando se monta el componente (“useEffect”), o a través de un evento.*

Implementaciones: Fetch

Llamados GET:

- *Al desarrollar en front-end los llamados del tipo “GET” suelen tener repercusiones en la UI.*
- *El disparo del llamado puede hacerse cuando se monta el componente (“useEffect”), o a través de un evento.*
- *En React manejaremos la respuesta del llamado con el hook “useEffect”.*

Implementaciones: Fetch

Ejemplo de GET:

Implementaciones: Fetch

Ejemplo de GET:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null)
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos")
6.        .then(response => response.json())
7.        .then(data => setAlumnos(data))
8.    }, [])
9.
10.   return (
11.     <p>
12.       {!alumnos
13.         ? "loading..."
14.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
15.     </p>
16.   )
17. }
```

Implementaciones: Fetch

Ejemplo de GET:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null) // Iniciamos el valor inicial como vacío.
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos")
6.        .then(response => response.json())
7.        .then(data => setAlumnos(data))
8.    }, [])
9.
10.   return (
11.     <p>
12.       {!alumnos
13.         ? "loading..."
14.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
15.     </p>
16.   )
17. }
```

Implementaciones: Fetch

Ejemplo de GET:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null)
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos") // Acá inicia el llamado HTTP, en este caso es un GET.
6.        .then(response => response.json())
7.        .then(data => setAlumnos(data))
8.    }, [])
9.
10.   return (
11.     <p>
12.       {!alumnos
13.         ? "loading..."
14.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
15.     </p>
16.   )
17. }
```

Implementaciones: Fetch

Ejemplo de GET:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null)
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos")
6.        .then(response => response.json()) // Convertimos la respuesta al formato JSON.
7.        .then(data => setAlumnos(data))
8.    }, [])
9.
10.   return (
11.     <p>
12.       {!alumnos
13.         ? "loading..."
14.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
15.     </p>
16.   )
17. }
```

Implementaciones: Fetch

Ejemplo de GET:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null)
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos")
6.        .then(response => response.json())
7.        .then(data => setAlumnos(data)) // Guardamos toda la información en la variable `alumnos`.
8.    }, [])
9.
10.   return (
11.     <p>
12.       {!alumnos
13.         ? "loading..."
14.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
15.     </p>
16.   )
17. }
```

Implementaciones: Fetch

Ejemplo de GET:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null)
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos")
6.        .then(response => response.json())
7.        .then(data => setAlumnos(data))
8.    }, [])
9.
10.   return (
11.     <p>
12.       {!alumnos // En caso de no tener alumnos, asumiremos que el llamado HTTP está cargándose.
13.         ? "loading..."
14.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
15.     </p>
16.   )
17. }
```

Implementaciones: Fetch

Ejemplo de GET:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null)
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos")
6.        .then(response => response.json())
7.        .then(data => setAlumnos(data))
8.    }, [])
9.
10.   return (
11.     <p>
12.       {!alumnos
13.         ? "loading ... "// Mostramos el mensaje de carga.
14.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
15.     </p>
16.   )
17. }
```

Implementaciones: Fetch

Ejemplo de GET:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null)
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos")
6.        .then(response => response.json())
7.        .then(data => setAlumnos(data))
8.    }, [])
9.
10.   return (
11.     <p>
12.       {!alumnos
13.         ? "loading..."
14.         : `El primer alumno de la lista es ${alumnos[0].alumno}`} // Al llegar el dato mostramos el primer.
15.     </p>
16.   )
17. }
```

Implementaciones: Fetch

Llamados POST:

Implementaciones: Fetch

Llamados POST:

- *Al desarrollar en front-end solemos capturar información a menudo*

Implementaciones: Fetch

Llamados POST:

- *Al desarrollar en front-end solemos capturar información a menudo*

- *El disparo del llamado que envía los datos deberá ser preferentemente a través de un evento.*

Implementaciones: Fetch

Llamados POST:

- *Al desarrollar en front-end solemos capturar información a menudo*
- *El disparo del llamado que envía los datos deberá ser preferentemente a través de un evento.*
- *En caso de necesitar manejar la respuesta del llamado, lo haremos con el hook “useEffect”.*

Implementaciones: Fetch

Ejemplo de POST:

Implementaciones: Fetch

Ejemplo de POST:

```jsx
function FormularioAlumno() {
  const [nombre, setNombre] = useState("")
  const [edad, setEdad] = useState(0)
  const handleSubmit = () => {
    fetch("http://localhost:3000/alumnos", {
      method: "POST",
      headers: { "Accept": "application/json", "Content-Type": "application/json" },
      body: JSON.stringify({ alumno: nombre, edad }),
    })
  }
  return (
    <div>
      <input type="text" onChange={e => setNombre(e.target.value)} />
      <br />
      <input type="number" onChange={e => setEdad(e.target.valueAsNumber)} />
      <br />
      <button onClick={handleSubmit}>Enviar</button>
    </div>
  )
}
```

Implementaciones: Fetch

Ejemplo de POST:

```
1.  function FormularioAlumno() {
2.    const [nombre, setNombre] = useState("") // Estado donde guardaremos el nombre.
3.    const [edad, setEdad] = useState(0)
4.    const handleSubmit = () => {
5.      fetch("http://localhost:3000/alumnos", {
6.        method: "POST",
7.        headers: { "Accept": "application/json", "Content-Type": "application/json" },
8.        body: JSON.stringify({ alumno: nombre, edad }),
9.      })
10.   }
11.   return (
12.     <div>
13.       <input type="text" onChange={e => setNombre(e.target.value)} />
14.       <br />
15.       <input type="number" onChange={e => setEdad(e.target.valueAsNumber)} />
16.       <br />
17.       <button onClick={handleSubmit}>Enviar</button>
18.     </div>
19.   )
20. }
```

Implementaciones: Fetch

Ejemplo de POST:

```
1.  function FormularioAlumno() {
2.    const [nombre, setNombre] = useState("")
3.    const [edad, setEdad] = useState(0) // Estado donde guardaremos la edad.
4.    const handleSubmit = () => {
5.      fetch("http://localhost:3000/alumnos", {
6.        method: "POST",
7.        headers: { "Accept": "application/json", "Content-Type": "application/json" },
8.        body: JSON.stringify({ alumno: nombre, edad }),
9.      })
10.   }
11.   return (
12.     <div>
13.       <input type="text" onChange={e => setNombre(e.target.value)} />
14.       <br />
15.       <input type="number" onChange={e => setEdad(e.target.valueAsNumber)} />
16.       <br />
17.       <button onClick={handleSubmit}>Enviar</button>
18.     </div>
19.   )
20. }
```

Implementaciones: Fetch

Ejemplo de POST:

```jsx
function FormularioAlumno() {
  const [nombre, setNombre] = useState("")
  const [edad, setEdad] = useState(0)
  const handleSubmit = () => { // Manejaremos el envío de los datos a través de eventos.
    fetch("http://localhost:3000/alumnos", {
      method: "POST",
      headers: { "Accept": "application/json", "Content-Type": "application/json" },
      body: JSON.stringify({ alumno: nombre, edad }),
    })
  }
  return (
    <div>
      <input type="text" onChange={e => setNombre(e.target.value)} />
      <br />
      <input type="number" onChange={e => setEdad(e.target.valueAsNumber)} />
      <br />
      <button onClick={handleSubmit}>Enviar</button>
    </div>
  )
}
```

Implementaciones: Fetch

Ejemplo de POST:

```jsx
function FormularioAlumno() {
  const [nombre, setNombre] = useState("")
  const [edad, setEdad] = useState(0)
  const handleSubmit = () => {
    fetch("http://localhost:3000/alumnos", { // Configuración básica del fetch.
      method: "POST",
      headers: { "Accept": "application/json", "Content-Type": "application/json" },
      body: JSON.stringify({ alumno: nombre, edad }),
    })
  }
  return (
    <div>
      <input type="text" onChange={e => setNombre(e.target.value)} />
      <br />
      <input type="number" onChange={e => setEdad(e.target.valueAsNumber)} />
      <br />
      <button onClick={handleSubmit}>Enviar</button>
    </div>
  )
}
```

Implementaciones: Fetch

Ejemplo de POST:

```jsx
function FormularioAlumno() {
  const [nombre, setNombre] = useState("")
  const [edad, setEdad] = useState(0)
  const handleSubmit = () => {
    fetch("http://localhost:3000/alumnos", {
      method: "POST",
      headers: { "Accept": "application/json", "Content-Type": "application/json" },
      body: JSON.stringify({ alumno: nombre, edad }), // Acá envíamos el body del POST parseado como JSON.
    })
  }
  return (
    <div>
      <input type="text" onChange={e => setNombre(e.target.value)} />
      <br />
      <input type="number" onChange={e => setEdad(e.target.valueAsNumber)} />
      <br />
      <button onClick={handleSubmit}>Enviar</button>
    </div>
  )
}
```

Implementaciones: Fetch

Ejemplo de POST:

```jsx
function FormularioAlumno() {
  const [nombre, setNombre] = useState("")
  const [edad, setEdad] = useState(0)
  const handleSubmit = () => {
    fetch("http://localhost:3000/alumnos", {
      method: "POST",
      headers: { "Accept": "application/json", "Content-Type": "application/json" },
      body: JSON.stringify({ alumno: nombre, edad }),
    })
  }
  return (
    <div> // Caputramos el `string` que contiene el nombre del alumno.
      <input type="text" onChange={e => setNombre(e.target.value)} />
      <br />
      <input type="number" onChange={e => setEdad(e.target.valueAsNumber)} />
      <br />
      <button onClick={handleSubmit}>Enviar</button>
    </div>
  )
}
```

Implementaciones: Fetch

Ejemplo de POST:

```
1.  function FormularioAlumno() {
2.    const [nombre, setNombre] = useState("")
3.    const [edad, setEdad] = useState(0)
4.    const handleSubmit = () => {
5.      fetch("http://localhost:3000/alumnos", {
6.        method: "POST",
7.        headers: { "Accept": "application/json", "Content-Type": "application/json" },
8.        body: JSON.stringify({ alumno: nombre, edad }),
9.      })
10.   }
11.   return (
12.     <div>
13.       <input type="text" onChange={e => setNombre(e.target.value)} />
14.       <br /> // Capturamos el `number` que contiene la edad del alumno.
15.       <input type="number" onChange={e => setEdad(e.target.valueAsNumber)} />
16.       <br />
17.       <button onClick={handleSubmit}>Enviar</button>
18.     </div>
19.   )
20. }
```

Implementaciones: Fetch

Ejemplo de POST:

```
1.  function FormularioAlumno() {
2.    const [nombre, setNombre] = useState("")
3.    const [edad, setEdad] = useState(0)
4.    const handleSubmit = () => {
5.      fetch("http://localhost:3000/alumnos", {
6.        method: "POST",
7.        headers: { "Accept": "application/json", "Content-Type": "application/json" },
8.        body: JSON.stringify({ alumno: nombre, edad }),
9.      })
10.   }
11.   return (
12.     <div>
13.       <input type="text" onChange={e => setNombre(e.target.value)} />
14.       <br />
15.       <input type="number" onChange={e => setEdad(e.target.valueAsNumber)} />
16.       <br /> // Invocamos el callback que envía el POST cuando el evento `onClick` es disparado.
17.       <button onClick={handleSubmit}>Enviar</button>
18.     </div>
19.   )
20. }
```

React

Implementaciones: Mapeo/Arrays

Implementaciones: Mapeo/Arrays

Algunas ideas sobre los Arrays…

Implementaciones: Mapeo/Arrays

Algunas ideas sobre los Arrays…

- Forman parte de JavaScript.

Implementaciones: Mapeo/Arrays

Algunas ideas sobre los Arrays…

- Forman parte de JavaScript.
- Son listas ordenadas.

Implementaciones: Mapeo/Arrays

Algunas ideas sobre los Arrays…

- Forman parte de JavaScript.
- Son listas ordenadas.
- Tienen varios métodos asociados de forma nativa.

Implementaciones: Mapeo/Arrays

Algunas ideas sobre los Arrays…

- Forman parte de JavaScript.
- Son listas ordenadas.
- Tienen varios métodos asociados de forma nativa.
- Solo en React necesitaremos usar la prop “key”.

Implementaciones: Mapeo/Arrays

Algunas ideas sobre los Arrays…

- Forman parte de JavaScript.
- Son listas ordenadas.
- Tienen varios métodos asociados de forma nativa.
- Solo en React necesitaremos usar la prop “key”.
- Las mismas consideraciones que tenemos en JS deben ser tomadas en cuenta en React.

Implementaciones: Mapeo/Arrays

Arrays y JavaScript:

Implementaciones: Mapeo/Arrays

Arrays y JavaScript:

- *JavaScript incorporó muchas funcionalidades al lenguaje que serán de gran utilidad en React.*

Implementaciones: Mapeo/Arrays

Arrays y JavaScript:

- *JavaScript incorporó muchas funcionalidades al lenguaje que serán de gran utilidad en React.*

- *El conocimiento sobre algoritmos en JS es bien ponderado a la hora de desarrollar en React.*

Implementaciones: Mapeo/Arrays

Arrays y JavaScript:

- *JavaScript incorporó muchas funcionalidades al lenguaje que serán de gran utilidad en React.*
- *El conocimiento sobre algoritmos en JS es bien ponderado a la hora de desarrollar en React.*
- *Los problemas que existen en JS siguen existiendo en React, por ejemplo los problemas de mutabilidad.*

Implementaciones: Mapeo/Arrays

Arrays y React:

Implementaciones: Mapeo/Arrays

Arrays y React:

- *React espera que se devuelva un dato concreto para colocar en el JSX, por ejemplo: string, componente.*

Implementaciones: Mapeo/Arrays

Arrays y React:

- *React espera que se devuelva un dato concreto para colocar en el JSX, por ejemplo: string, componente.*

- *React no tiene métodos “especiales” de Arrays.*

Implementaciones: Mapeo/Arrays

Arrays y React:

- *React espera que se devuelva un dato concreto para colocar en el JSX, por ejemplo: string, componente.*
- *React no tiene métodos “especiales” de Arrays.*
- *En el caso del mapeo, se requerirá una nueva “prop” o atributo, la prop “key”.*

Implementaciones: Mapeo/Arrays

Casos que veremos:

- `.map()`
- `.filter()`
- `.sort()`

Implementaciones: Mapeo/Arrays

Para todos los ejemplos de Mapeo/Arrays usaremos este ejemplo previo:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null)
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos")
6.        .then(response => response.json())
7.        .then(data => setAlumnos(data))
8.    }, [])
9.
10.   return (
11.     <p>
12.       {!alumnos
13.         ? "loading..."
14.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
15.     </p>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Para todos los ejemplos de Mapeo/Arrays usaremos este ejemplo previo:

```
1.  function AlumnosComponent() {
2.    const [alumnos, setAlumnos] = useState(null)
3.
4.    useEffect(() => {
5.      fetch("http://localhost:3000/alumnos")
6.        .then(response => response.json())
7.        .then(data => setAlumnos(data))
8.    }, [])
9.
10.   return (
11.     // En los ejemplos nos limitaremos sólo a mostrar lo que retornamos acá.
12.   )
13. }
14.
15.
16.
17.
```

Implementaciones: Mapeo/Arrays

.map():

Implementaciones: Mapeo/Arrays

`.map()`:

- *El concepto “mapear” proviene directamente de la programación funcional.*

Implementaciones: Mapeo/Arrays

`.map()`:

- *El concepto “mapear” proviene directamente de la programación funcional.*

- *Consiste en utilizar una función “mapeadora” que será ejecutada en cada elemento de una lista (array).*

Implementaciones: Mapeo/Arrays

Ejemplo .map():

Implementaciones: Mapeo/Arrays

Ejemplo .map():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .map():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <> // Utilizaremos un React.Fragment como contenedor de nuestro JSX.
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .map():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..." // La lógica del renderizado condicional seguirá siendo la misma.
7.          : alumnos
8.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .map():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos // En lugar de usar un elemento en particular, usaremos la lista completa.
8.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .map():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.              // En este espacio podremos encadenar otros métodos de Arrays.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .map():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.
9.              .map((alumno, index) => ( // Declaramos los argumentos de alumno e índice, ambas provienen de JS.
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .map():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}> // Conformamos una "key prop“ única.
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .map():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.                 // Armamos el componente usando el argumento de `alumno`.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

`.filter()`:

Implementaciones: Mapeo/Arrays

`.filter()`:

- *El método de JS nos permite filtrar una lista utilizando un callback.*

Implementaciones: Mapeo/Arrays

.filter():

- *El método de JS nos permite filtrar una lista utilizando un callback.*

- *Los métodos “map” y “filter” pueden encadenarse.*

Implementaciones: Mapeo/Arrays

Ejemplo .filter():

```
1.
```

Implementaciones: Mapeo/Arrays

Ejemplo .filter():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .filter():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.              // Filtraremos por alumnos con edad superior a 29 años.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .filter():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.              .filter(alumno ⇒ alumno.edad > 29)
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

`.sort()`:

Implementaciones: Mapeo/Arrays

`.sort()`:

- *Es una funcionalidad de JS que nos permitirá ordenar una lista.*

Implementaciones: Mapeo/Arrays

`.sort()`:

- *Es una funcionalidad de JS que nos permitirá ordenar una lista.*

- *Se recomienda leer la documentación oficial de MDN para entender como se utiliza en JS.*

Implementaciones: Mapeo/Arrays

`.sort()`:

- *Es una funcionalidad de JS que nos permitirá ordenar una lista.*
- *Se recomienda leer la documentación oficial de MDN para entender como se utiliza en JS.*
- *En JS este método muta el array original. Se recomienda crear una copia para no mutar al original.*

Implementaciones: Mapeo/Arrays

Ejemplo .sort():

Implementaciones: Mapeo/Arrays

Ejemplo .sort():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .sort():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.              // En este espacio colocaremos el método sort para ordenar la lista.
9.              .map((alumno, index) ⇒ (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .sort():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.              .sort((a, b) => a.edad - b.edad) // Ordenaremos los alumnos por edad de menor a mayor.
9.              .map((alumno, index) => (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .sort():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.              .sort((a, b) => b.edad - a.edad) // Ordenaremos los alumnos por edad de mayor a menor.
9.              .map((alumno, index) => (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

`.sort()`:

Implementaciones: Mapeo/Arrays

`.sort()`:

- *Prestar atención que en el caso anterior mutamos directamente al array de alumnos.*

Implementaciones: Mapeo/Arrays

`.sort()`:

- *Prestar atención que en el caso anterior mutamos directamente al array de alumnos.*

- *Para evitar la mutación, lo mejor es crear una copia nueva.*

Implementaciones: Mapeo/Arrays

`.sort()`:

- *Prestar atención que en el caso anterior mutamos directamente al array de alumnos.*
- *Para evitar la mutación, lo mejor es crear una copia nueva.*
- *En este caso parece conveniente usar el método “.slice()”.*

Implementaciones: Mapeo/Arrays

Ejemplo .sort():

```
1.
```

Implementaciones: Mapeo/Arrays

Ejemplo .sort():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos
8.              .sort((a, b) => b.edad - a.edad)
9.              .map((alumno, index) => (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .sort():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos.slice()
8.              .sort((a, b) => b.edad - a.edad)
9.              .map((alumno, index) => (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

Implementaciones: Mapeo/Arrays

Ejemplo .sort():

```
1.  function AlumnosComponent() {
2.   …
3.    return (
4.      <>
5.        {!alumnos
6.          ? "loading..."
7.          : alumnos.slice() // Este método de JS nos creará una copia del array.
8.              .sort((a, b) => b.edad - a.edad)
9.              .map((alumno, index) => (
10.               <p key={`ejemplo-key-2-${index}-${alumno.alumno}`}>
11.
12.                 El alumno {alumno.alumno} tiene la edad de {alumno.edad} años. Su número de alumno es {alumno.id}.
13.               </p>
14.             ))}
15.     </>
16.   )
17. }
```

React

Implementaciones: Custom Hooks

Implementaciones: Custom Hooks

Algunas ideas sobre los Custom Hooks…

Implementaciones: Custom Hooks

Algunas ideas sobre los Custom Hooks…

- Reutilización de componentes/funcionalidades.

Implementaciones: Custom Hooks

Algunas ideas sobre los Custom Hooks…

- Reutilización de componentes/funcionalidades.
- Creación de componentes “inteligentes”.

Implementaciones: Custom Hooks

Algunas ideas sobre los Custom Hooks…

- Reutilización de componentes/funcionalidades.
- Creación de componentes “inteligentes”.
- Requieren buen entendimiento de los hooks comunes.

Implementaciones: Custom Hooks

Algunas ideas sobre los Custom Hooks…

- Reutilización de componentes/funcionalidades.
- Creación de componentes “inteligentes”.
- Requieren buen entendimiento de los hooks comunes.
- Las reglas de los hooks se mantienen.

Implementaciones: Custom Hooks

Un poco más en detalle…

Implementaciones: Custom Hooks

Un poco más en detalle…

- *Los “custom hooks” nos permiten aislar la lógica de un componente.*

Implementaciones: Custom Hooks

Un poco más en detalle…

- *Los “custom hooks” nos permiten aislar la lógica de un componente.*

- *Muchas veces son opcionales, pero ayudan mucho al mantenimiento del código.*

Implementaciones: Custom Hooks

Un poco más en detalle…

- *Los “custom hooks” nos permiten aislar la lógica de un componente.*
- *Muchas veces son opcionales, pero ayudan mucho al mantenimiento del código.*
- *La “firma” del hook es definida por nosotros como desarrolladores.*

Implementaciones: Custom Hooks

Vale la pena aclarar, que los “custom hooks” podrían considerarse como “patrones de diseño avanzados” en React.

Implementaciones: Custom Hooks

Vale la pena aclarar, que los “custom hooks” podrían considerarse como “patrones de diseño avanzados” en React.

Nosotros sólo veremos algunos casos a modo de ejemplo.

Implementaciones: Custom Hooks

Ejemplos:

- useCheckBox
- useGetFetch

Implementaciones: Custom Hooks

Ejemplos:

- useCheckBox (irá dentro de AsitenciasComponent)
- useGetFetch (irá dentro de AlumnosComponent)

Implementaciones: Mapeo/Arrays

useCheckBox.js:

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react"
2.  function useCheckBox(initialValue = false) {
3.    const [isChecked, setIsChecked] = useState(initialValue)
4.
5.    const customCheckbox = (
6.      props
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     />
13.   )
14.
15.   return [isChecked, customCheckbox]
16. }
17. export default useCheckBox
```

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react" // Este Custom Hook funcionará sobre un hook del tipo useState().
2.  function useCheckBox(initialValue = false) {
3.    const [isChecked, setIsChecked] = useState(initialValue)
4.
5.    const customCheckbox = (
6.      props
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     />
13.   )
14.
15.   return [isChecked, customCheckbox]
16. }
17. export default useCheckBox
```

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react"
2.  function useCheckBox(initialValue = false) { // Acá declaramos el valor inicial que pasaremos al useState().
3.    const [isChecked, setIsChecked] = useState(initialValue)
4.
5.    const customCheckbox = (
6.      props
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     />
13.   )
14.
15.   return [isChecked, customCheckbox]
16. }
17. export default useCheckBox
```

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react"
2.  function useCheckBox(initialValue = false) {
3.    const [isChecked, setIsChecked] = useState(initialValue) // Configuramos el useState().
4.
5.    const customCheckbox = (
6.      props
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     />
13.   )
14.
15.   return [isChecked, customCheckbox]
16. }
17. export default useCheckBox
```

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react"
2.  function useCheckBox(initialValue = false) {
3.    const [isChecked, setIsChecked] = useState(initialValue)
4.    // Hacemos un componente interno para el hook.
5.    const customCheckbox = (
6.      props
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     />
13.   )
14.
15.   return [isChecked, customCheckbox]
16. }
17. export default useCheckBox
```

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react"
2.  function useCheckBox(initialValue = false) {
3.    const [isChecked, setIsChecked] = useState(initialValue)
4.
5.    const customCheckbox = (
6.      props // Puede aceptar props.
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     />
13.   )
14.
15.   return [isChecked, customCheckbox]
16. }
17. export default useCheckBox
```

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react"
2.  function useCheckBox(initialValue = false) {
3.    const [isChecked, setIsChecked] = useState(initialValue)
4.
5.    const customCheckbox = (
6.      props
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     /> // Creamos el input genérico.
13.   )
14.
15.   return [isChecked, customCheckbox]
16. }
17. export default useCheckBox
```

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react"
2.  function useCheckBox(initialValue = false) {
3.    const [isChecked, setIsChecked] = useState(initialValue)
4.
5.    const customCheckbox = (
6.      props
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     />
13.   )
14.   // El primer elemento es un booleano, el segundo es el componente.
15.   return [isChecked, customCheckbox]
16. }
17. export default useCheckBox
```

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react"
2.  function useCheckBox(initialValue = false) {
3.    const [isChecked, setIsChecked] = useState(initialValue)
4.
5.    const customCheckbox = (
6.      props
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     />
13.   )
14.
15.   return [isChecked, customCheckbox] // Devolvemos un array, por lo tanto el orden importa.
16. }
17. export default useCheckBox
```

Implementaciones: Mapeo/Arrays

useCheckBox.js:

```
1.  import { useState } from "react"
2.  function useCheckBox(initialValue = false) {
3.    const [isChecked, setIsChecked] = useState(initialValue)
4.
5.    const customCheckbox = (
6.      props
7.    ) => (
8.      <input
9.        type="checkbox"
10.       checked={isChecked}
11.       onChange={e => setIsChecked(e.target.checked)}
12.     />
13.   )
14.
15.   return [isChecked, customCheckbox]
16. }
17. export default useCheckBox // Exportamos el custom hook.
```

Implementaciones: Mapeo/Arrays

AsistenciasComponent.js:

Implementaciones: Mapeo/Arrays

AsistenciasComponent.js:

```
1.  import useCheckBox from "./useCheckBox.js"
2.  function AsistenciasComponent() {
3.    const [isPresente, PresenteCheckBox] = useCheckBox()
4.    const [isCorregido, CorregidoCheckBox] = useCheckBox()
5.    const handleClick = () ⇒ {
6.      const body = { isPresente, isCorregido }
7.      console.log(body)
8.    }
9.    return (
10.     <div>
11.       <PresenteCheckBox /> ¿El alumno asistió a clases?
12.       <br />
13.       <CorregidoCheckBox /> ¿El TP del alumno fue corregido?
14.       <br />
15.       <button onClick={handleClick}>Envíar</button>
16.     </div>
17.   )
18. }
19. export default AsistenciasComponent
```

Implementaciones: Mapeo/Arrays

AsistenciasComponent.js:

```
1.  import useCheckBox from "./useCheckBox.js" // Importamos el custom Hook.
2.  function AsistenciasComponent() {
3.    const [isPresente, PresenteCheckBox] = useCheckBox()
4.    const [isCorregido, CorregidoCheckBox] = useCheckBox()
5.    const handleClick = () ⇒ {
6.      const body = { isPresente, isCorregido }
7.      console.log(body)
8.    }
9.    return (
10.     <div>
11.       <PresenteCheckBox /> ¿El alumno asistió a clases?
12.       <br />
13.       <CorregidoCheckBox /> ¿El TP del alumno fue corregido?
14.       <br />
15.       <button onClick={handleClick}>Envíar</button>
16.     </div>
17.   )
18. }
19. export default AsistenciasComponent
```

Implementaciones: Mapeo/Arrays

AsistenciasComponent.js:

```
1.  import useCheckBox from "./useCheckBox.js"
2.  function AsistenciasComponent() {
3.    const [isPresente, PresenteCheckBox] = useCheckBox() // Configuramos el custom hook.
4.    const [isCorregido, CorregidoCheckBox] = useCheckBox()
5.    const handleClick = () ⇒ {
6.      const body = { isPresente, isCorregido }
7.      console.log(body)
8.    }
9.    return (
10.     <div>
11.       <PresenteCheckBox /> ¿El alumno asistió a clases?
12.       <br />
13.       <CorregidoCheckBox /> ¿El TP del alumno fue corregido?
14.       <br />
15.       <button onClick={handleClick}>Envíar</button>
16.     </div>
17.   )
18. }
19. export default AsistenciasComponent
```

Implementaciones: Mapeo/Arrays

AsistenciasComponent.js:

```
1.  import useCheckBox from "./useCheckBox.js"
2.  function AsistenciasComponent() {
3.    const [isPresente, PresenteCheckBox] = useCheckBox()
4.    const [isCorregido, CorregidoCheckBox] = useCheckBox() // El orden en el caso de los arrays es de suma importancia.
5.    const handleClick = () ⇒ {
6.      const body = { isPresente, isCorregido }
7.      console.log(body)
8.    }
9.    return (
10.     <div>
11.       <PresenteCheckBox /> ¿El alumno asistió a clases?
12.       <br />
13.       <CorregidoCheckBox /> ¿El TP del alumno fue corregido?
14.       <br />
15.       <button onClick={handleClick}>Envíar</button>
16.     </div>
17.   )
18. }
19. export default AsistenciasComponent
```

Implementaciones: Mapeo/Arrays

AsistenciasComponent.js:

```
1.  import useCheckBox from "./useCheckBox.js"
2.  function AsistenciasComponent() {
3.    const [isPresente, PresenteCheckBox] = useCheckBox()
4.    const [isCorregido, CorregidoCheckBox] = useCheckBox()
5.    const handleClick = () ⇒ { // Manejamos el envío de los datos en esta función.
6.      const body = { isPresente, isCorregido }
7.      console.log(body)
8.    }
9.    return (
10.     <div>
11.       <PresenteCheckBox /> ¿El alumno asistió a clases?
12.       <br />
13.       <CorregidoCheckBox /> ¿El TP del alumno fue corregido?
14.       <br />
15.       <button onClick={handleClick}>Envíar</button>
16.     </div>
17.   )
18. }
19. export default AsistenciasComponent
```

Implementaciones: Mapeo/Arrays

AsistenciasComponent.js:

```
1.  import useCheckBox from "./useCheckBox.js"
2.  function AsistenciasComponent() {
3.    const [isPresente, PresenteCheckBox] = useCheckBox()
4.    const [isCorregido, CorregidoCheckBox] = useCheckBox()
5.    const handleClick = () ⇒ {
6.      const body = { isPresente, isCorregido } // Armamos el body usando los datos capturados.
7.      console.log(body)
8.    }
9.    return (
10.     <div>
11.       <PresenteCheckBox /> ¿El alumno asistió a clases?
12.       <br />
13.       <CorregidoCheckBox /> ¿El TP del alumno fue corregido?
14.       <br />
15.       <button onClick={handleClick}>Envíar</button>
16.     </div>
17.   )
18. }
19. export default AsistenciasComponent
```

Implementaciones: Mapeo/Arrays

AsistenciasComponent.js:

```
1.  import useCheckBox from "./useCheckBox.js"
2.  function AsistenciasComponent() {
3.    const [isPresente, PresenteCheckBox] = useCheckBox()
4.    const [isCorregido, CorregidoCheckBox] = useCheckBox()
5.    const handleClick = () ⇒ {
6.      const body = { isPresente, isCorregido }
7.      console.log(body) // En este caso no tenemos a donde enviar los datos, simplemente los loguearemos.
8.    }
9.    return (
10.     <div>
11.       <PresenteCheckBox /> ¿El alumno asistió a clases?
12.       <br />
13.       <CorregidoCheckBox /> ¿El TP del alumno fue corregido?
14.       <br />
15.       <button onClick={handleClick}>Envíar</button>
16.     </div>
17.   )
18. }
19. export default AsistenciasComponent
```

Implementaciones: Mapeo/Arrays

AsistenciasComponent.js:

```
1.  import useCheckBox from "./useCheckBox.js"
2.  function AsistenciasComponent() {
3.    const [isPresente, PresenteCheckBox] = useCheckBox()
4.    const [isCorregido, CorregidoCheckBox] = useCheckBox()
5.    const handleClick = () ⇒ {
6.      const body = { isPresente, isCorregido }
7.      console.log(body)
8.    }
9.    return ( // En el segundo elemento del custom hook estamos devolviendo un componente que usamos en el JSX.
10.     <div>
11.       <PresenteCheckBox /> ¿El alumno asistió a clases?
12.       <br />
13.       <CorregidoCheckBox /> ¿El TP del alumno fue corregido?
14.       <br />
15.       <button onClick={handleClick}>Envíar</button>
16.     </div>
17.   )
18. }
19. export default AsistenciasComponent
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react" // Este Custom Hook utilizará useState() y useEffect().
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") { // Lugar para capturar la url.
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null) // Estado con los datos del llamado HTTP.
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true) // Booleano para controlar si el llamado HTTP todavía está cargándose.
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false) // Booleano en caso de error.
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) { // En caso de tener un string vacío, no haremos el llamado HTTP.
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData)) // Guardamos la información del response.
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true)) // Damos aviso que hubo un error.
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false)) // Damos aviso que el llamado HTTP ya terminó.
13.     }
14.   }, [url])
15.   return { data, isLoading, isError }
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

useGetFetch.js:

```
1.  import { useState, useEffect } from "react"
2.  function useGetFetch(url = "") {
3.    const [data, setData] = useState(null)
4.    const [isLoading, setIsLoading] = useState(true)
5.    const [isError, setIsError] = useState(false)
6.    useEffect(() => {
7.      if (url) {
8.        fetch(url)
9.          .then(response => response.json())
10.         .then(responseData => setData(responseData))
11.         .catch(() => setIsError(true))
12.         .finally(() => setIsLoading(false))
13.     }
14.   }, [url])
15.   return { data, isLoading, isError } // Devolvemos todos los estados dentro de un objeto.
16. }
17. export default useGetFetch
```

Implementaciones: Mapeo/Arrays

AlumnosComponent.js:

Implementaciones: Mapeo/Arrays

AlumnosComponent.js:

```
1.  import useGetFetch from "./useGetFetch.js"
2.  function AlumnosComponent() {
3.
4.    const { data: alumnos, isLoading, isError } = useGetFetch("http://localhost:3000/alumnos")
5.    return (
6.      <p>
7.        {isLoading
8.          ? "loading ... "
9.          : isError
10.         ? "Hubo un error."
11.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
12.     </p>
13.   )
14. }
15. export default AlumnosComponent
```

Implementaciones: Mapeo/Arrays

AlumnosComponent.js:

```
1.  import useGetFetch from "./useGetFetch.js"
2.  function AlumnosComponent() {
3.    // Utilizamos desestructuración de objetos, y envíamos la url por los parámetros del useGetFetch().
4.    const { data: alumnos, isLoading, isError } = useGetFetch("http://localhost:3000/alumnos")
5.    return (
6.      <p>
7.        {isLoading
8.          ? "loading..."
9.          : isError
10.         ? "Hubo un error."
11.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
12.     </p>
13.   )
14. }
15. export default AlumnosComponent
```

Implementaciones: Mapeo/Arrays

AlumnosComponent.js:

```
1.  import useGetFetch from "./useGetFetch.js"
2.  function AlumnosComponent() {
3.
4.    const { data: alumnos, isLoading, isError } = useGetFetch("http://localhost:3000/alumnos")
5.    return (
6.      <p>
7.        {isLoading // Mostramos el mensaje de carga.
8.          ? "loading..."
9.          : isError
10.         ? "Hubo un error."
11.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
12.     </p>
13.   )
14. }
15. export default AlumnosComponent
```

Implementaciones: Mapeo/Arrays

AlumnosComponent.js:

```
1.  import useGetFetch from "./useGetFetch.js"
2.  function AlumnosComponent() {
3.
4.    const { data: alumnos, isLoading, isError } = useGetFetch("http://localhost:3000/alumnos")
5.    return (
6.      <p>
7.        {isLoading
8.          ? "loading ... "
9.          : isError // Mostroamos el mensaje en caso de error.
10.         ? "Hubo un error."
11.         : `El primer alumno de la lista es ${alumnos[0].alumno}`}
12.     </p>
13.   )
14. }
15. export default AlumnosComponent
```

Implementaciones

Implementaciones

Si es que hay tiempo para ir al código…

Implementaciones

Si es que hay tiempo para ir al código…

Traten de "jugar" con los ejemplos dados.

Implementaciones

Si es que hay tiempo para ir al código…

Traten de "jugar" con los ejemplos dados.

Son ejemplos bastante "potentes" y estándar dentro de la industria.

¡Vamos al código!

Clase 5: Implementaciones

¡Muchas gracias!

ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar