
ICARO
Nuestro compromiso es con el futuro.

React

Clase 8

React: Producción 2

Temas:

- **Estados REST**
- **Estilado**
- **Componentes npm**

React: Producción 2

Esta clase es una continuación de la clase anterior.

Es importante que la sección de “Conceptos” explicada en la clase “Producción 1” sea tomada en cuenta.

* Recordar: Performance, Accesibilidad, Escalabilidad, etc ...

React

Producción 2: Estados REST

Producción 2: Estados REST

¿Qué veremos sobre Estados REST?

- Problemas con useEffect
- Axios
- React Query
- Cacheo
- DevTools
- Mutaciones
- Custom Hooks

Producción 2: Estados REST

Problemas con useEffect:

- *Este hook está pensado para sincronizar estados, está más relacionado en el ciclo de vida del componente.*
- *El manejo de useEffect puede traernos errores que son muy difíciles de corregir ya que no es muy intuitivo.*
- *El entendimiento en profundidad del funcionamiento del hook requiere conocimientos avanzados de la programación en general.*

Producción 2: Estados REST

Axios:

- *Es una de las librerías más utilizadas para realizar peticiones HTTP tanto en el front como en el back.*
- *Es mucho más simple de utilizar que “fetch()” y requiere menos configuración.*
- *Fetch tiene funcionalidades que axios no posee aún, sin embargo axios nos servirá en un 99% de los casos.*

Producción 2: Estados REST

Axios:

npm:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  npm install axios
```

yarn:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  yarn add axios
```

documentación: https://axios-http.com/docs

Producción 2: Estados REST

Axios (se puede probar en el archivo App.jsx):

```
1.  import axios from "axios“
2.
3.  const getProductsFetch = () ⇒
4.    fetch("https://fakestoreapi.com/products")
5.      .then(response ⇒ response.json())
6.      .then(response ⇒ console.log("Ejemplo GET fetch:", response))
7.
8.  const getProductsAxios = () ⇒
9.    axios.get("https://fakestoreapi.com/products")
10.     .then(response ⇒ response.data)
11.     .then(response ⇒ console.log("Ejemplo GET axios:", response))
12.
13. getProductsFetch()
14. getProductsAxios()
15.
16. function App() { ... }
```

Producción 2: Estados REST

Axios (se puede probar en el archivo App.jsx):

```
1.  import axios from "axios" // Importamos axios.
2.
3.  const getProductsFetch = () => // Creamos función de ejemplo GET con fetch.
4.    fetch("https://fakestoreapi.com/products") // Configuración básica del GET.
5.      .then(response => response.json()) // Transformamos el response a JSON.
6.      .then(response => console.log("Ejemplo GET fetch:", response)) // Logueamos el resopnse.
7.
8.  const getProductsAxios = () => // Creamos función de ejemplo GET con axios.
9.    axios.get("https://fakestoreapi.com/products") // Configuración básica del GET.
10.     .then(response => response.data) // Extraemos la data del response.
11.     .then(response => console.log("Ejemplo GET axios:", response)) // Logueamos el resopnse.
12.
13. getProductsFetch() // Invocamos la función con fetch a modo de ejemplo.
14. getProductsAxios() // Invocamos la función con axios a modo de ejemplo.
15.
16. function App() { ... }
```

Producción 2: Estados REST

ejemplosPost.js:

```
1.  import axios from "axios“
2.
3.  const postProductsFetch = body ⇒
4.    fetch("https://fakestoreapi.com/products", {
5.      method: "POST",
6.      body: JSON.stringify(body),
7.    })
8.      .then(response ⇒ response.json())
9.      .then(response ⇒ console.log(response))
10.
11. const postProductsAxios = body ⇒
12.   axios.post("https://fakestoreapi.com/products", body)
13.     .then(response ⇒ response.data)
14.     .then(response ⇒ console.log(response))
15.
16. export { postProductsAxios, postProductsFetch }
```

Producción 2: Estados REST

ejemplosPost.js:

```
1.  import axios from "axios“
2.
3.  const postProductsFetch = body ⇒
4.    fetch("https://fakestoreapi.com/products", { // La configuración de un POST es un poco más larga en fetch.
5.      method: "POST",
6.      body: JSON.stringify(body),
7.    })
8.      .then(response ⇒ response.json())
9.      .then(response ⇒ console.log(response))
10.
11. const postProductsAxios = body ⇒
12.   axios.post("https://fakestoreapi.com/products", body) // En axios no necesitamos convertir a JSON.
13.     .then(response ⇒ response.data)
14.     .then(response ⇒ console.log(response))
15.
16. export { postProductsAxios, postProductsFetch }
```

Producción 2: Estados REST

Axios (ejemplo en App.jsx):

```
1.  import { postProductsAxios, postProductsFetch } from "./ejemplosPost.js“
2.
3.  const ejemploBody = {
4.    title: "test product",
5.    price: 13.5,
6.    description: "lorem ipsum set",
7.    image: "https://i.pravatar.cc",
8.    category: "electronic",
9.  }
10.
11. postProductsFetch(ejemploBody)
12. postProductsAxios(ejemploBody)
13.
14. function App() { ... }
```

Producción 2: Estados REST

Axios (ejemplo en App.jsx):

```
1.  import { postProductsAxios, postProductsFetch } from "./ejemplosPost.js“
2.
3.  const ejemploBody = { // Armamos el body que requiera el servicio.
4.    title: "test product",
5.    price: 13.5,
6.    description: "lorem ipsum set",
7.    image: "https://i.pravatar.cc",
8.    category: "electronic",
9.  }
10.
11. postProductsFetch(ejemploBody) // Invocamos el ejemplo de fetch.
12. postProductsAxios(ejemploBody) // Invocamos el ejemplo de axios.
13.
14. function App() { ... }
```

Producción 2: Estados REST

React Query:

- *Popular librería que mejora el desarrollo y la performance, está inspirada en eventos y observables.*
- *Nos va permitir almacenar (cachear) los llamados HTTP, y consumirlos de manera global en toda la APP.*
- *Puede utilizar tanto Axios como fetch, y nos permitirá evitar usar useEffect en muchos casos.*

Producción 2: Estados REST

React Query:

npm:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  npm install @tanstack/react-query
```

yarn:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  yarn add @tanstack/react-query
```

documentación: https://tanstack.com/query/latest

Producción 2: Estados REST

React Query:

```
1.  import React from "react"
2.  import { BrowserRouter, Routes, Route } from "react-router-dom“
3.  import { QueryClient, QueryClientProvider } from "@tanstack/react-query"
4.  const queryClient = new QueryClient()
5.  function App() {
6.    return (
7.      <QueryClientProvider client={queryClient}>
8.        <BrowserRouter>
9.          <Routes>
10.           <Route path="/" element={<h1>Bienvenidos!</h1>} />
11.           <Route path="/error" element={<h1>Error!</h1>} />
12.         </Routes>
13.       </BrowserRouter>
14.     </QueryClientProvider>
15.   )
16. }
17. export default App
```

Producción 2: Estados REST

React Query:

```
1.  import React from "react"
2.  import { BrowserRouter, Routes, Route } from "react-router-dom“ // Nos basamos en el ejemplo anterior de React Router.
3.  import { QueryClient, QueryClientProvider } from "@tanstack/react-query" // Importamos los componentes de React Query.
4.  const queryClient = new QueryClient() // Este es el cliente en si mismo, se encargará de controlar los llamados HTTP.
5.  function App() {
6.    return (
7.      <QueryClientProvider client={queryClient}> // Con este componente vamos a proveer todas las funcionalidades a la APP.
8.        <BrowserRouter>
9.          <Routes>
10.           <Route path="/" element={<h1>Bienvenidos!</h1>} />
11.           <Route path="/error" element={<h1>Error!</h1>} />
12.         </Routes>
13.       </BrowserRouter>
14.     </QueryClientProvider>
15.   )
16. }
17. export default App
```

Producción 2: Estados REST

Cacheo:

- *La capacidad de conservar los llamados HTTP casi siempre es de gran utilidad.*
- *React Query utiliza “queryKeys” para validar o invalidar llamados desactualizados.*
- *La “queryFn” va a ser cacheada tal y como llega a los métodos de React Query.*

Producción 2: Estados REST

React Query(cacheo):

```
1.  import { useQuery } from "@tanstack/react-query"
2.  import axios from "axios"
3.
4.  const getTitles = () ⇒
5.    axios.get("https://fakestoreapi.com/products")
6.      .then(({ data }) ⇒ data.map(({ title }) ⇒ title))
7.     // .then(({ data }) ⇒ data)
8.
9.  function Ejemplo() {
10.   const { data: titles = [], isLoading } = useQuery({
11.     queryKey: ["products"],
12.     queryFn: getTitles,
13.   })
14.   return <>{isLoading ? "cargando ..." : JSON.stringify(titles)}</>
15. }
16.
17. export default Ejemplo
```

Producción 2: Estados REST

React Query(cacheo):

```
1.  import { useQuery } from "@tanstack/react-query" // Importamos el hook useQuery.
2.  import axios from "axios" // Usaremos axios.
3.
4.  const getTitles = () => // Creamos una función para trabajar los datos.
5.    axios.get("https://fakestoreapi.com/products") // Configuración básica del fetch.
6.      .then(({ data }) => data.map(({ title }) => title)) // Extraemos los títulos del response.
7.     // .then(({ data }) => data) // Probar esta también esta alternativa para ver que ocurre en este caso.
8.
9.  function Ejemplo() { // Creamos un componente para probar el caso.
10.   const { data: titles = [], isLoading } = useQuery({ // Utilizaremos los títulos y el estado isLoading.
11.     queryKey: ["products"], // Configuramos las queryKeys, servirá para realizar tareas de caché.
12.     queryFn: getTitles, // Le decimos a React Query que vamos a usar la función que creamos arriba.
13.   })
14.   return <>{isLoading ? "cargando..." : JSON.stringify(titles)}</> // Renderizamos los datos a modo de ejemplo.
15. }
16.
17. export default Ejemplo
```

Producción 2: Estados REST

DevTools:

- *La librería nos permite instalar herramientas que facilitan el desarrollo.*
- *Requiere configuración extra, pero no afecta la performance de nuestra APP en producción.*
- *Estas herramientas nos facilitarán el seguimiento de los datos, conocer el estado de las “queryKeys” y realizar re-fetchs desde las herramientas.*

Producción 2: Estados REST

React Query:

npm:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  npm install @tanstack/react-query-devtools
```

yarn:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  yarn add @tanstack/react-query-devtools
```

documentación: https://tanstack.com/query/latest

Producción 2: Estados REST

React Query(DevTools):

```
1.  import { BrowserRouter, Routes, Route } from "react-router-dom“
2.  import { ReactQueryDevtools } from "@tanstack/react-query-devtools"
3.  import { QueryClient, QueryClientProvider } from "@tanstack/react-query"
4.  const queryClient = new QueryClient()
5.  function App() {
6.    return (
7.      <QueryClientProvider client={queryClient}>
8.        <BrowserRouter>
9.          <Routes>
10.           ...
11.         </Routes>
12.       </BrowserRouter>
13.       <ReactQueryDevtools initialIsOpen={false} />
14.     </QueryClientProvider>
15.   )
16. }
17. export default App
```

Producción 2: Estados REST

React Query(DevTools):

```
1.  import { BrowserRouter, Routes, Route } from "react-router-dom“
2.  import { ReactQueryDevtools } from "@tanstack/react-query-devtools" // Importamos el componente de DevTools.
3.  import { QueryClient, QueryClientProvider } from "@tanstack/react-query"
4.  const queryClient = new QueryClient()
5.  function App() {
6.    return (
7.      <QueryClientProvider client={queryClient}>
8.        <BrowserRouter>
9.          <Routes>
10.           ...
11.         </Routes>
12.       </BrowserRouter>
13.       <ReactQueryDevtools initialIsOpen={false} /> // Colocamos el componente de DevTools.
14.     </QueryClientProvider>
15.   )
16. }
17. export default App
```

Producción 2: Estados REST

Mutaciones:

- *Están asociadas a los métodos POST/PUT y también a los eventos como “onClick” u “onChange”.*
- *Funcionan de manera similar a “useQuery”, pero vienen con un método disparador.*
- *La invalidación de “queryKeys” puede logarse con la ayuda del método queryClient.invalidateQueries().*

Producción 2: Estados REST

Custom Hooks:

- *Los hooks de “useQuery()” y “useMutation()” son compatibles con el patrón de custom hooks.*
- *La implementación de custom hooks nos ayudará a separar la lógica del llamado HTTP (escalabilidad).*
- *El cacheo también será más sencillo implementando estos hooks.*

Producción 2: Estados REST

React Query(Custom Hook):

```
1.  import { useQuery } from "@tanstack/react-query"
2.  import axios from "axios“
3.  const getProducts = () ⇒
4.    axios.get("https://fakestoreapi.com/products")
5.      .then(({ data }) ⇒ data) // Simplemente extraemos el objeto data de axios.
6.
7.  const useProductosQuery = () ⇒ {
8.    const { data: products = [], isLoading } = useQuery({ // Extraemos los productos y el estado isLoading.
9.      queryKey: ["products"],
10.     queryFn: getProducts,
11.   })
12.   const categories = [ ... new Set(products.map(product ⇒ product.category))] // Creamos categorías únicas.
13.   const maxPrice = Math.max( ... products.map(product ⇒ product.price)) // Buscamos el precio más alto.
14.   const minPrice = Math.min( ... products.map(product ⇒ product.price)) // Buscamos el precio más bajo.
15.   return { categories, isLoading, products, maxPrice, minPrice } // Devolvemos valores procesados listos para consumir.
16. }
17. export default useProductosQuery
```

React

Producción 2: Estilado

Producción 2: Estilado

¿Qué veremos sobre Estilado?

- Ejemplo className
- Problema de styles.css
- Estilos de línea
- Styled Components

Producción 2: Estilado

Ejemplo className:

- *Son clases comunes para la manipulación por CSS, requieren poca configuración.*
- *className es una sintaxis estándar de vanilla JS, la palabra “class” es una palabra reservada en JS.*
- *Usaremos este ejemplo en todos los casos, simplemente para comparar.*

Producción 2: Estilado

styles.css:

```css
.btn {
  cursor: pointer;
  background-color: #7286d3;
  color: #e5e0ff;
  border-radius: 0.5rem;
  border: 0;
  margin: 0.5rem 0;
  padding: 0.5rem 1rem;
  font-weight: bold;
}

.btn:hover {
  background-color: #e5e0ff;
  color: #7286d3;
}
```

Producción 2: Estilado

Ejemplo className:

```
1. import "./styles.css“
2.
3. function Ejemplo() {
4.   return <button className="btn">Agregar</button>
5. }
6.
7. export default Ejemplo
```

Producción 2: Estilado

Problemas de styles.css:

- *Los nombres de las clases deben estar bien pensados ya que pueden sobreponerse fácilmente.*
- *Muchas veces sólo necesitamos estilos con usos muy específicos.*
- *Los patrones de arquitectura de estilos clásicos como BEM, SMACSS u OOCSS son útiles para mejorar la escalabilidad, pero de difícil implementación.*

Producción 2: Estilado

Estilos de línea:

- *Es la forma en la que React nos permite dar estilo a un componente en particular.*
- *Es de fácil implementación y muchas veces funciona como uno espera sin mayores problemas.*
- *Existen algunos patrones de diseño para escalar esta técnica, sin embargo en proyectos grandes conviene usar librerías.*

Producción 2: Estilado

Ejemplo estilos de línea:

```
import { useState } from "react"
function Ejemplo() {
  const [isHover, setIsHover] = useState(false)
  return (
    <button
      style={{
        backgroundColor: !isHover ? "#7286d3" : "#e5e0ff",
        color: isHover ? "#7286d3" : "#e5e0ff",
        cursor: "pointer", borderRadius: "0.5rem", border: "0", margin: "0.5rem 0", padding: "0.5rem 1rem",
        fontWeight: "bold",
      }}
      onMouseEnter={() => setIsHover(true)}
      onMouseLeave={() => setIsHover(false)}
    >Agregar</button>
  )
}
export default Ejemplo
```

Producción 2: Estilado

Ejemplo estilos de línea:

```
import { useState } from "react" // Para recrear el hover necesitaremos usar estados
function Ejemplo() {
  const [isHover, setIsHover] = useState(false) // La ventaja es que podemos crear funcionalidades que CSS no tiene.
  return (
    <button
      style={{ // Prestar atención a las doble llaves.
        backgroundColor: !isHover ? "#7286d3" : "#e5e0ff", // Manejamos los colores de forma condicional.
        color: isHover ? "#7286d3" : "#e5e0ff", // Abajo están agrupados simplemente por motivos de espacio.
        cursor: "pointer", borderRadius: "0.5rem", border: "0", margin: "0.5rem 0", padding: "0.5rem 1rem",
        fontWeight: "bold", // Atentos al reemplazo de guiones por camel case en “font-weight” o “background-color”.
      }}
      onMouseEnter={() => setIsHover(true)} // ¿Se animan a agregarle lógica de cambiar color en base a la posición?
      onMouseLeave={() => setIsHover(false)} // ¿Se animan a agregarle lógica de tiempo o que quede siempre seleccionado?
    >Agregar</button>
  )
}
export default Ejemplo
```

Producción 2: Estilado

Styled Components:

- *Es la librería por excelencia para organizar y crear estilos en React.*
- *Es flexible a varios patrones de diseño de estilos y tiene varios niveles de configuración.*
- *Tiene soporte de sintaxis de compiladores de CSS como SASS.*

Producción 2: Estilado

Styled Components:

npm:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  npm install styled-components
```

yarn:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  yarn add styled-components
```

documentación: https://styled-components.com/docs

Producción 2: Estilado

Ejemplo.styled.js:

```
1.  import styled from "styled-components“
2.
3.  export const Button = styled.button`
4.    cursor: pointer;
5.    background-color: #7286d3;
6.    color: #e5e0ff;
7.    border-radius: 0.5rem;
8.    border: 0;
9.    margin: 0.5rem 0;
10.   padding: 0.5rem 1rem;
11.   font-weight: bold;
12.
13.   &:hover {
14.     background-color: #e5e0ff;
15.     color: #7286d3;
16.   }
17. `
```

Producción 2: Estilado

Ejemplo.styled.js:

```
1.  import styled from "styled-components“ // Importamos el objeto styled de la librería.
2.
3.  export const Button = styled.button` // El objeto styled tiene muchas configuraciones, esta es simplemente una.
4.    cursor: pointer; // Dentro de los template-strings pondremos CSS.
5.    background-color: #7286d3;
6.    color: #e5e0ff;
7.    border-radius: 0.5rem;
8.    border: 0;
9.    margin: 0.5rem 0;
10.   padding: 0.5rem 1rem; // Si queremos ver estos colores en el VSCode, debemos instalar la extensión de styled-components.
11.   font-weight: bold;
12.
13.   &:hover { // Este tipo de sintaxis es similar a la de SASS.
14.     background-color: #e5e0ff;
15.     color: #7286d3;
16.   }
17. `
```

Producción 2: Estilado

Ejemplo.jsx:

```jsx
import { Button } from "./Ejemplo.styled"

function Ejemplo() {
  return <Button>Agregar</Button>
}

export default Ejemplo
```

Producción 2: Estilado

Ejemplo.jsx:

```jsx
import { Button } from "./Ejemplo.styled" // La convención para un componente `Xxx.jsx` es `Xxx.styled.js`.

function Ejemplo() { // Creamos otro componente de ejemplo
  return <Button>Agregar</Button> // Simplemente usamos el nuevo componente de styled-components como un componente más.
}

export default Ejemplo
```

React

Producción 2: Componentes npm

Producción 2: Componentes npm

¿Qué veremos sobre Componentes npm?

- Librerías de componentes
- Ventajas
- Desventajas
- Ant Design
- Ejemplo

Producción 2: Componentes npm

Librerías de componentes:

- *Son librerías con componentes que se importan como si los hubiéramos hecho nosotros en nuestro proyecto.*
- *Traen documentación, estilos y configuración propia. En general están fuertemente testeadas.*
- *Las del tipo “headless” no traen estilos en absoluto, sólo la funcionalidad.*

Producción 2: Componentes npm

Ventajas:

- *La velocidad de desarrollo se incrementa mucho con este tipo de librerías.*
- *La mayoría están fuertemente testeadas en aspectos como accesibilidad, performance, UX/UI, etc.*
- *Muchas empresas optan por crear sus propias librerías de estilos por motivos de “branding”, ya que unifican el concepto de “marca” a través de estilos.*

Producción 2: Componentes npm

Desventajas:

- *Traen problemas similares a los de frameworks como Bootstrap o Bulma.*
- *Se requiere un aprendizaje muy específico de cada librería que no es intercambiable entre librerías.*
- *Los estilos pueden ser difíciles de personalizar muchas veces y la documentación para componentes muy específicos puede llegar ser escasa.*

Producción 2: Componentes npm

Ant Design:

- *Es una librería de componentes popular, robusta, escalable, sencilla de utilizar y muy testeada.*
- *La documentación es extensa, detallada, con ejemplos visuales y ejemplos de código.*
- *Tiene soporte para TypeScript y hooks modernos de React.*

Producción 2: Componentes npm

Ant Design:

npm:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  npm install antd
```

yarn:

```
alumno@DESKTOP-ABCDFGH9 IJKLM0 ~/Documents/mi-proyecto
$  yarn add antd
```

documentación: https://ant.design/components/overview/

Producción 2: Componentes npm

ExampleCard.jsx:

```jsx
import React from "react"
import { Link } from "react-router-dom"
import { Button, Card, Image } from "antd"
const ExampleCard = () => {
  return (
    <Card
      title="Lorem Ipsum"
      hoverable
      style={{ width: "15rem" }}
      cover={<Image src={"https://os.alipayobjects.com/rmsportal/QBnOOoLaAfKPirc.png"} />}
      extra={<Link to="/error">
                <Button type="primary">btn</Button>
              </Link>}
    ><Card.Meta title="Lorem Ipsum Faciebat" description="Lorem ipsum dolor sit amet consectetur adipisicing." /></Card>
  )
}
export default ExampleCard
```

Producción 2: Componentes npm

ExampleCard.jsx:

```jsx
import React from "react"
import { Link } from "react-router-dom"
import { Button, Card, Image } from "antd" // Importamos los componentes de la librería.
const ExampleCard = () => {
  return (
    <Card
      title="Lorem Ipsum" // Seteamos el título de la Card.
      hoverable // Booleano en valor true que permite que se pueda hacer hover en la Card.
      style={{ width: "15rem" }} // Estilos de línea.
      cover={<Image src={"https://os.alipayobjects.com/rmsportal/QBnOOoLaAfKPirc.png"} />} // Componente Image de antd.
      extra={<Link to="/error"> // Link de navegación aplicado a todo el botón.
                <Button type="primary">btn</Button> // Botón de antd.
              </Link>}
    ><Card.Meta title="Lorem Ipsum Faciebat" description="Lorem ipsum dolor sit amet consectetur adipisicing." /></Card>
  ) // Metadata que se puede agregar a la card.
}
export default ExampleCard
```

Producción 2

Traten de Recrear cualquier página que hayan creado hasta ahora.

Estas herramientas trabajan sobre los fundamentos de React.

Es importante leer la documentación oficial de cada herramienta.

¡Vamos al código!

Clase 8: Producción 2

¡Muchas gracias!

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