



Agencia de Habilidades  
para el Futuro

<Talento  
Tech/>

# ReactJS

---

**Clase 05 | Uso de useEffect y Peticiones a APIs**

# ¡Les damos la bienvenida!



Vamos a comenzar a grabar la clase.

## Índice

---

### React Hooks y Eventos

- Introducción al manejo del estado local con useState.
- Comprensión y manejo de eventos en React (clics, formularios).
- Proyecto final Talento Lab

### Uso de useEffect y Peticiones a APIs

- **Introducción al manejo de efectos secundarios con useEffect.**
- **Realización de peticiones a APIs para cargar productos.**
- **Manejo del estado de carga y errores en aplicaciones React.**

### Router - Rutas Estáticas y Dinámicas

- Introducción al manejo de efectos secundarios con useEffect.
- Realización de peticiones a APIs para cargar productos.
- Manejo del estado de carga y errores en aplicaciones React.

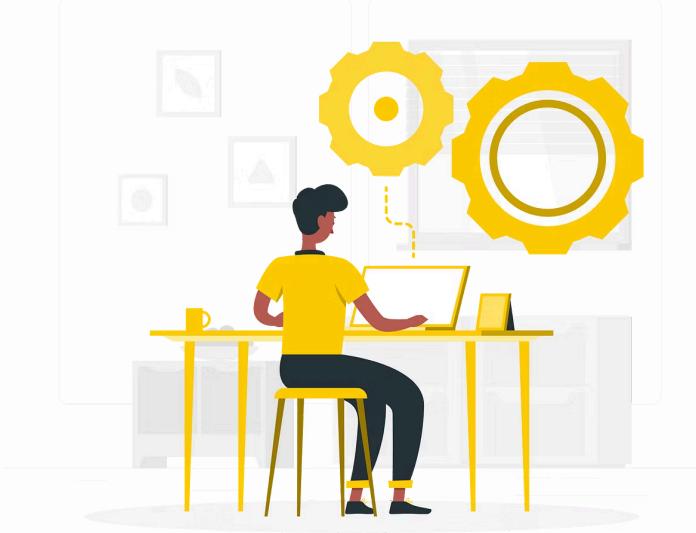
# Objetivos de la Clase

- 1 Comprender qué son los efectos secundarios y cómo gestionarlos con el hook **useEffect**.
- 2 Realizar peticiones a APIs para obtener datos dinámicamente.
- 3 Integrar MockAPI como una herramienta para simular datos en tiempo real.
- 4 Manejar estados de carga y errores para mejorar la experiencia del usuario.

# Introducción a **useEffect**



# ¿Qué son los efectos secundarios?



## Definición

Operaciones que ocurren fuera del flujo principal de renderizado de un componente.

## Ejemplos

Obtener datos de APIs, manipular el DOM, establecer suscripciones o temporizadores.

## Importancia

Permiten interactuar con el mundo exterior y manejar operaciones asíncronas.

# Hook useEffect

---

## ¿Para qué sirve?

Nos permite gestionar estos efectos secundarios en los componentes funcionales de React.

## Ventajas

Permite controlar cuándo se ejecutan operaciones y cómo afectan al ciclo de vida del componentes.

# Sintaxis básica de useEffect

---

```
useEffect(() => {  
  // Código del efecto secundario  
  return () => {  
    // Limpieza del efecto          //(opcional)  
  };  
}, [dependencias]);
```



# Componentes de useEffect

---



## 1 Efecto

Función que define el efecto secundario a ejecutar.

## 2 Limpieza

Función opcional que se ejecuta antes de desmontar o actualizar el efecto.

## 3 Dependencias

Array opcional que indica cuándo ejecutar el efecto.

# Ejemplo de useEffect

---



```
import React, { useEffect } from 'react';

function Mensaje() {
  useEffect(() => {
    console.log('El componente se ha montado.');
    return () => {
      console.log('El componente se ha
desmontado.');
    };
  }, []);
}

return <h1>Hola, React!</h1>;
}
```

# Casos comunes de uso de useEffect

---



## Obtener datos de API

Al montar el componente.



## Actualizar título

Modificar el título del documento.



## Temporizadores

Establecer y limpiar temporizadores.

# Peticiones a APIs

---

# Realización de Peticiones a APIs

Las peticiones a APIs permiten cargar datos dinámicamente desde un servidor. Esto es fundamental para crear aplicaciones que interactúen con datos en tiempo real o contenidos dinámicos.

**Pasos principales:**



## 1 Usar `fetch` o `axios`

Para realizar la petición HTTP.

## 2 Gestionar estado local

Almacenar los datos obtenidos.

## 3 Usar `useEffect`

Realizar la petición al montar el componente.



# Ejemplo: Obtener productos desde una API

```
import React, { useEffect, useState } from 'react';

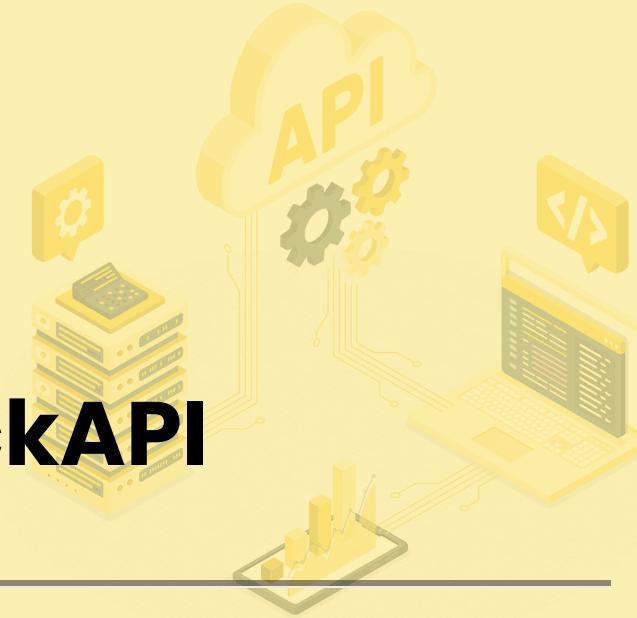
function Productos() {
  const [productos, setProductos] = useState([]);

  useEffect(() => {
    fetch('https://api.ejemplo.com/productos')
      .then((respuesta) => respuesta.json())
      .then((datos) => setProductos(datos))
      .catch((error) => console.error('Error:', error));
  }, []);

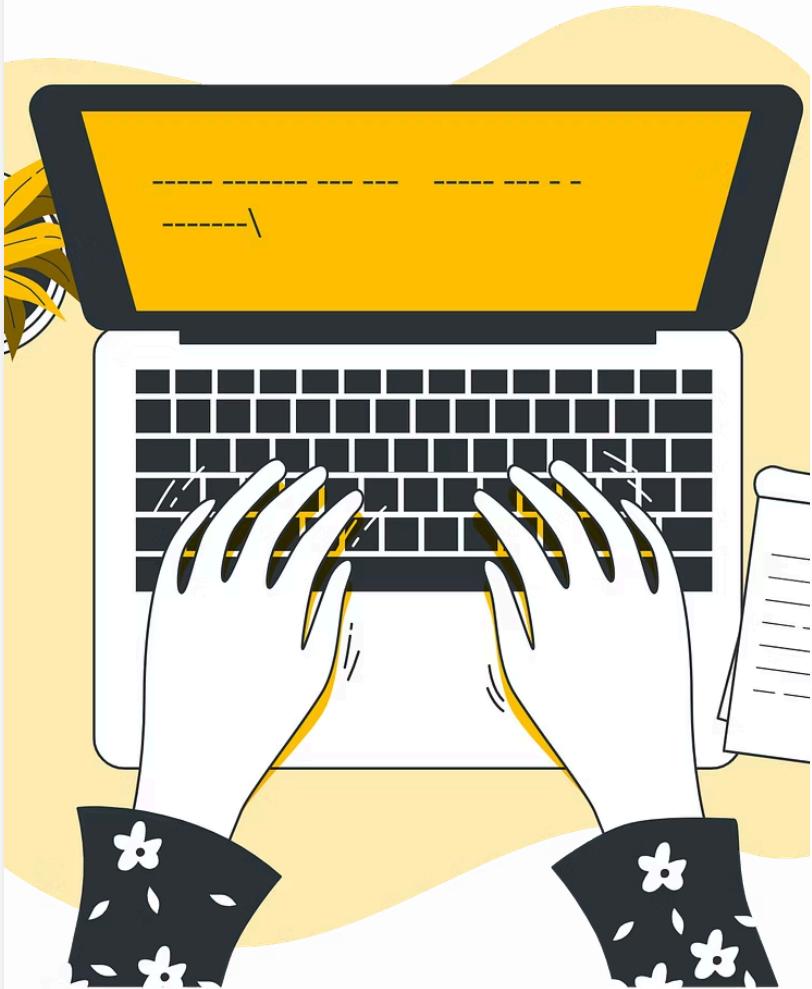
  return (
    <ul>
      {productos.map((producto) => (
        <li key={producto.id}> {producto.nombre}
      </li>
    ))
  )
}
```

```
)})  
</ul>  
 );  
}  
export default Productos;
```

# Introducción a MockAPI







# Introducción a MockAPI

## ¿Qué es?

Herramienta en línea para simular un backend RESTful.

## Uso

Ideal para proyectos de desarrollo y aprendizaje sin servidor real.

## Ventajas

Permite trabajar con datos realistas sin configurar un servidor.

# Características principales de MockAPI

---

## 1 Creación rápida de endpoints

Para realizar peticiones HTTP simuladas.

## 2 Interfaz gráfica sencilla

Facilita la gestión de datos.

## 3 Operaciones CRUD

Simula Crear, Leer, Actualizar y Eliminar datos.

# Ventajas de usar MockAPI

---



## Desarrollo rápido

Facilita pruebas y desarrollo sin configurar servidor.



## Datos realistas

Trabaja con información que simula un backend real.



## Compatibilidad

Funciona con fetch y axios para peticiones HTTP.

# Configuración de MockAPI



- 1** **Crear cuenta**  
Accede a MockAPI y regístrate.
- 2** **Crear proyecto**  
Define un nuevo proyecto en la plataforma.
- 3** **Definir recurso**  
Crea un recurso, por ejemplo "productos".
- 4** **Obtener endpoint**  
MockAPI generará una URL para tu recurso.

# Ejemplo con MockAPI

---

```
useEffect(() => {
  fetch('https://63f123.mockapi.io/productos')
    .then((respuesta) => respuesta.json())
    .then((datos) => setProductos(datos))
    .catch((error) => console.error('Error:', error));
}, []);
```

# **Manejo del Estado de Carga y Errores**

---



# Manejo del Estado de Carga y Errores

## Importancia

Mejora la experiencia del usuario al informar sobre el estado de las peticiones.

## Implementación

Usa estados adicionales para controlar la carga y los posibles errores.

# Estado de carga

## Definición

Indica si los datos están en proceso de ser obtenidos.

## Implementación

```
const [cargando,  
setCargando] =  
useState(true);
```

## Uso

Muestra un spinner o mensaje mientras se cargan los datos.





# Estado de error

## Definición

Guarda información sobre posibles fallos en la petición.

## Implementación

```
const [error, setError] = useState(null);
```

## Uso

Informa al usuario si ocurre un problema al cargar los datos.

# Ejemplo completo: Manejo de carga y errores

```
function Productos() {
  const [productos, setProductos] = useState([]);
  const [cargando, setCargando] = useState(true);
  const [error, setError] = useState(null);
  useEffect(() => {
    fetch('https://63f123.mockapi.io/productos')
      .then((respuesta) => respuesta.json())
      .then((datos) => {
        setProductos(datos);
        setCargando(false);
      })
      .catch((error) => {
        setError('Hubo un problema al cargar los productos.');
        setCargando(false);
      });
  }, []);
}

if (cargando) return <p>Cargando productos...</p>;
```

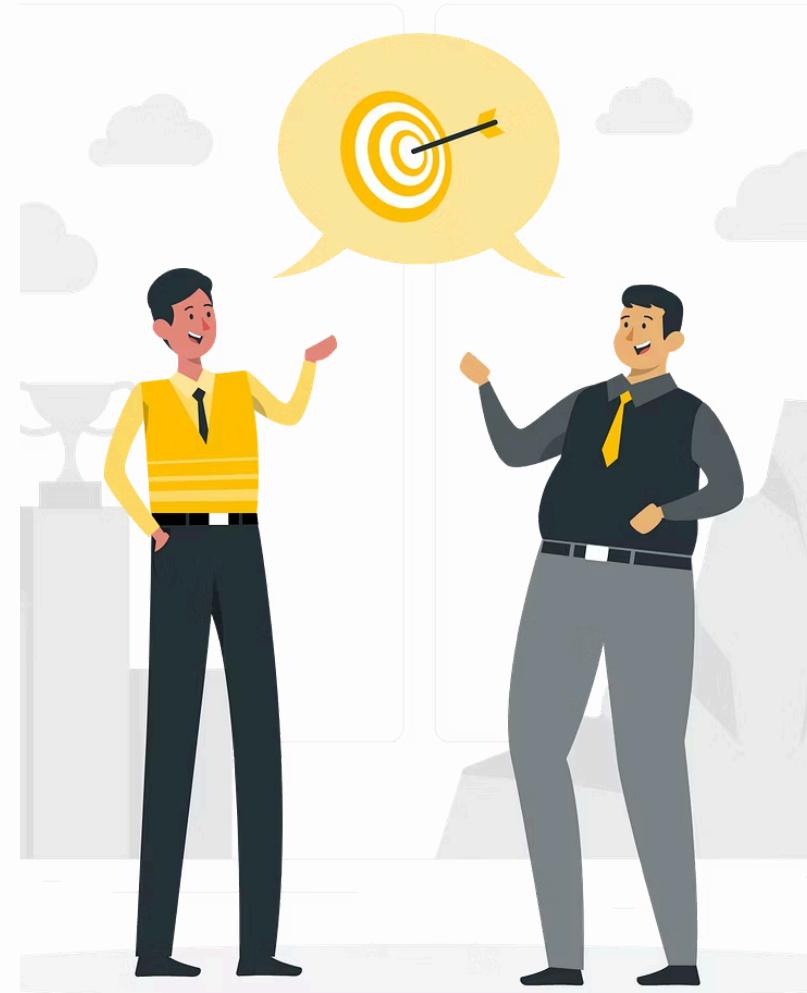
```
if (error) return <p>{error}</p>;\n\nreturn (\n  <ul>\n    {productos.map((producto) => (\n      <li key={producto.id}>{producto.nombre}</li>\n    ))}\n  </ul>\n);\n}\n\nexport default Productos;
```

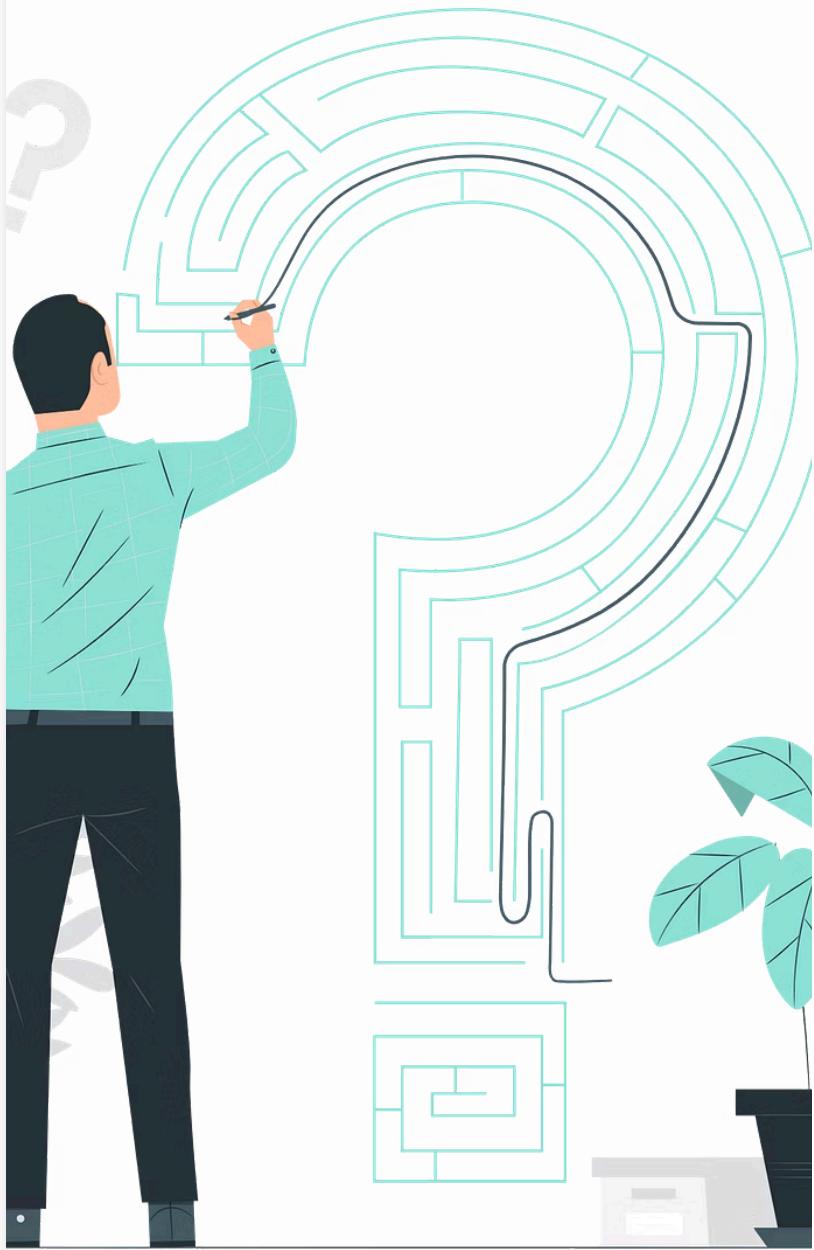
# Reflexión final

---

En esta clase, aprendimos a gestionar efectos secundarios con **useEffect**, realizar peticiones a APIs para cargar datos y manejar estados de carga y errores. También descubrimos cómo utilizar MockAPI para simular un backend, lo que nos permite trabajar con datos realistas sin necesidad de configurar un servidor.

Estas habilidades son esenciales para construir aplicaciones React que interactúen con datos externos, brindando una experiencia de usuario fluida y profesional. Al dominar estos conceptos, estarás preparado para desarrollar aplicaciones más completas y robustas.





# Preguntas para Reflexionar

## **useEffect vs useState**

¿Qué diferencia a useEffect de otros hooks como useState?

## **Dependencias**

¿Cómo decidir qué variables incluir en el array de dependencias?

## **Estados de carga**

¿Por qué es importante manejar estados de carga y errores?

## **MockAPI**

¿Qué ventajas tiene usar MockAPI frente a un servidor propio?

# Próximos Pasos

1

## React Router

Instalación y configuración.

2

## Rutas estáticas

Crear rutas para inicio y lista de productos.

3

## Navegación

Usar Link para navegar entre componentes.

# Recursos Adicionales

---

-  [Documentación oficial de useEffect](#)
-  [Guía introductoria a las promesas en JavaScript](#)
-  [MockAPI: Plataforma de simulación de datos](#)
-  [Estado y ciclo de vida en React](#)
-  [Cómo manejar errores en JavaScript](#)



# Ejercicios Prácticos

---



## TalentoLab – Proyecto Final

---

¡Felicitaciones nuevamente! 🎉 Ahora que has dominado la primera etapa de tu tarea en **TalentoLab**, el equipo ha decidido asignarte una responsabilidad más avanzada.



# Ejercicio Práctico

Obligatorio



**Silvia**

Product Owner

**Descripción de tu tarea:**

Tu próximo objetivo será conectar la aplicación a una API que provea información sobre los productos. Deberás mostrar los datos en tiempo real, manejar estados de carga y errores, y continuar utilizando un diseño atractivo para el eCommerce.



# Ejercicio Práctico

Obligatorio

## Requisitos del Proyecto:

### Integración con una API:

- Usa **MockAPI** para obtener una lista de productos desde un endpoint.
- Asegúrate de configurar correctamente la URL del recurso.

### Gestión del estado con useState:

- Almacena los productos obtenidos de la API en un estado local.
- Usa otro estado para gestionar si la aplicación está en proceso de carga.

### Manejo de efectos secundarios con useEffect:

- Realiza la petición a la API cuando el componente principal de productos se monte.
- Gestiona posibles errores y muestra un mensaje adecuado si ocurre algún problema.



# Ejercicio Práctico

Obligatorio

## Requisitos del Proyecto:

### Estado de carga y errores:

- Mientras los productos están cargando, muestra un mensaje o un spinner con "Cargando productos...".
- Si ocurre un error, muestra un mensaje como: "Error al cargar productos. Inténtalo más tarde."

### Actualizar el diseño del eCommerce:

- Crea una sección de productos dinámicos que se cargue desde la API.
- Asegúrate de mantener el diseño atractivo y responsive.

### Ampliación del carrito:

- Permite agregar al carrito productos que ahora se obtendrán desde la API.
- Si el carrito está vacío, muestra el mensaje "El carrito está vacío", como en el ejercicio anterior.



# Ejercicio Práctico

Obligatorio

Pautas generales del proyecto:

## 1 Estructura

Continuar con el proyecto de la clase anterior, agregando nueva funcionalidad.

## 2 Componentes

Mantener componentes organizados y reutilizables.

## 3 Código limpio

Evitar redundancias y organizar estados y efectos claramente.

## 4 Diseño

Usar Bootswatch o alternativa para mantener un diseño profesional.