Módulo 5
Desarrollo de aplicaciones Front-End con React

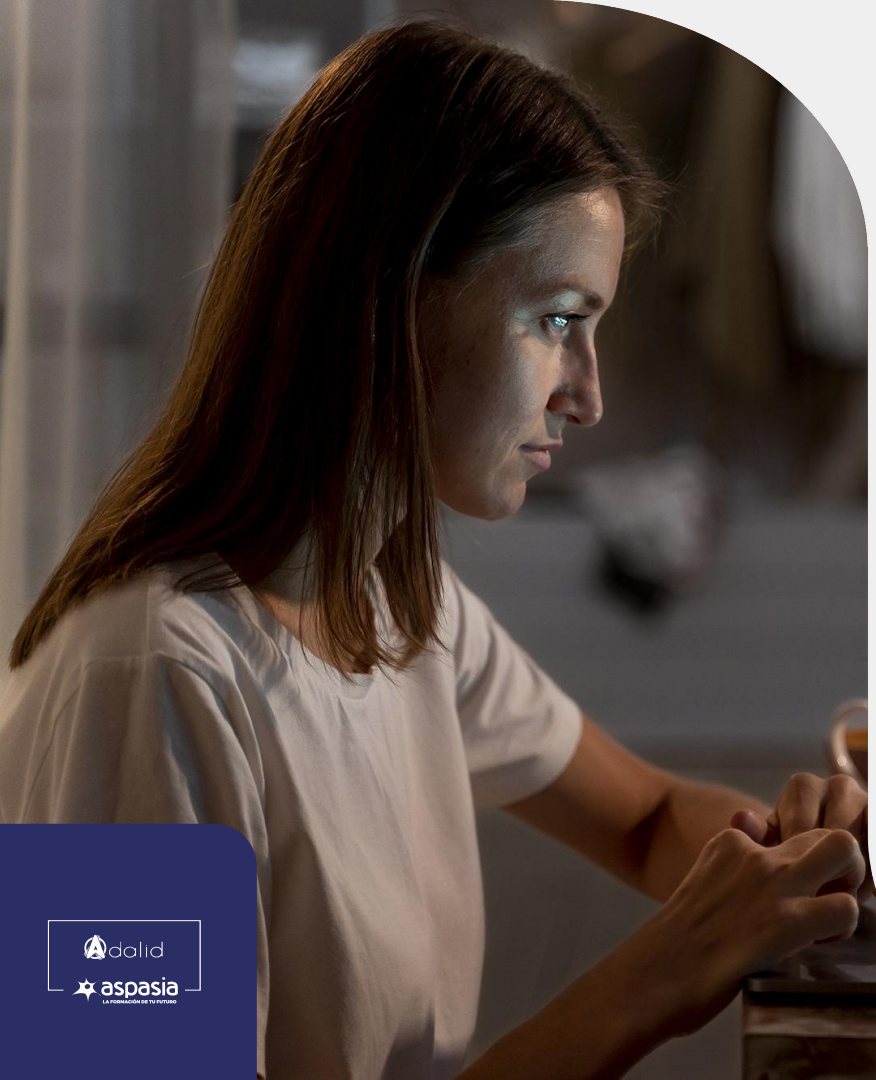
Hooks y Manejo de Errores

Módulo 5

AE 4.1

OBJETIVOS

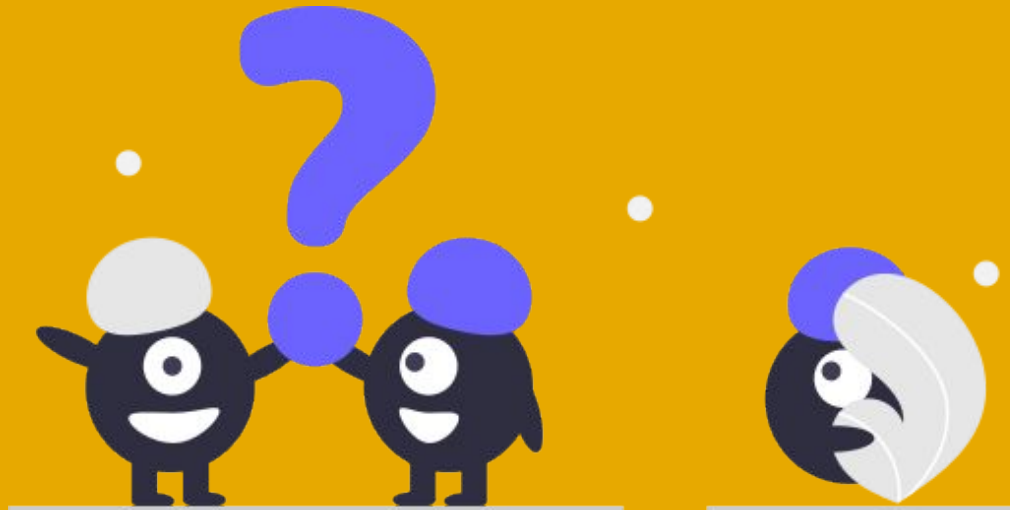
Entender qué son los Hooks en React, aprender a gestionar estado y efectos, construir Hooks personalizados, manejar errores y excepciones, y aplicar las reglas de Hooks para un desarrollo eficiente y seguro.

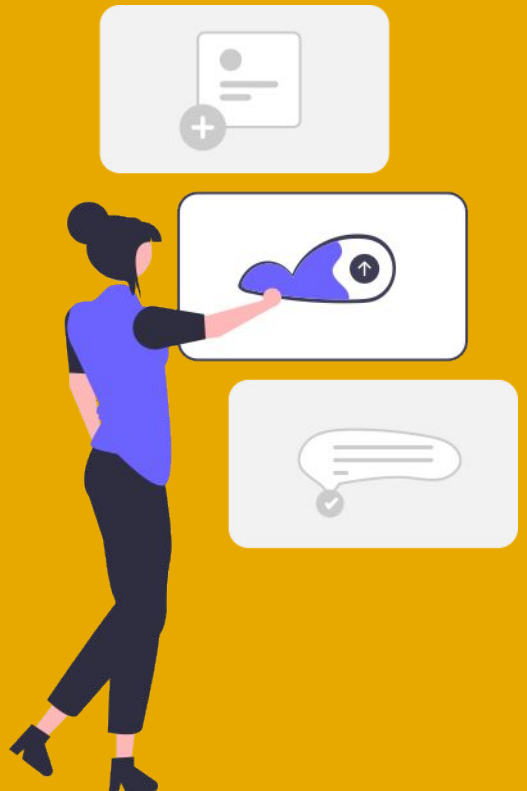


¿QUÉ VAMOS A VER?

- Qué son los Hooks.
- Un vistazo en detalle a los Hooks.
- Usando el Hook de estado.
- Usando el Hook de efecto.
- Reglas de los Hooks.
- Construyendo Hooks personalizados.
- Reconoce los mecanismos para el manejo de errores en un aplicativo React.
- Utiliza hooks en un aplicativo React para resolver un problema.
- Utiliza mecanismos disponibles en el entorno React para el manejo y control de los errores y excepciones.

¿Qué son los Hooks?





Hooks

Qué son los Hooks

Hooks son una característica de React que permite usar estado y otras funcionalidades de React sin escribir clases.

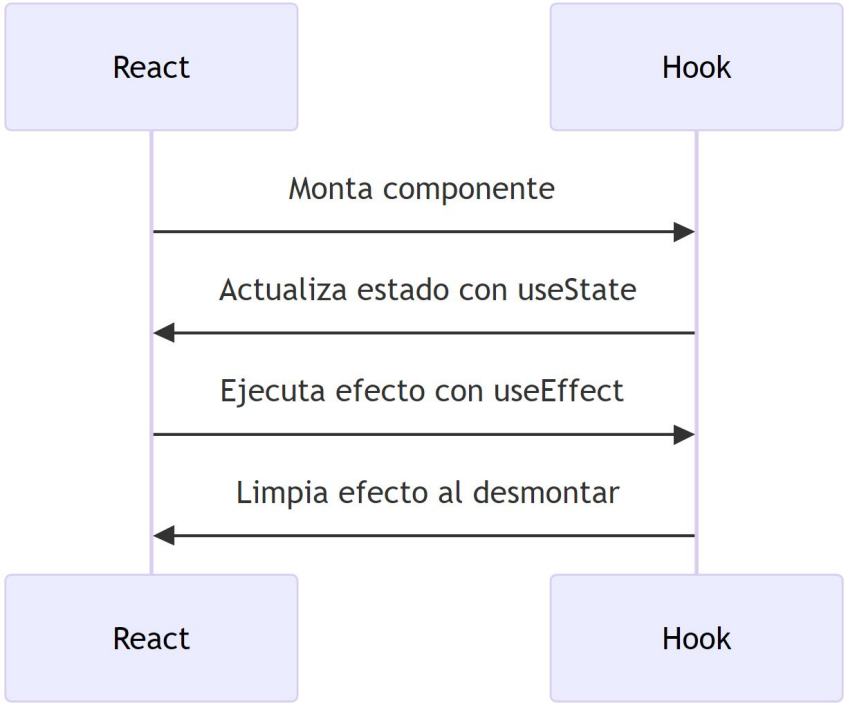
Ventajas:

- Simplifican el código.
- Permiten reutilizar lógica entre componentes.
- Mejoran la organización de funciones relacionadas.

Un Vistazo en Detalle a los Hooks

Principales Hooks:

- **useState:** Maneja el estado en componentes funcionales.
- **useEffect:** Realiza efectos secundarios como peticiones HTTP o suscripciones.



Usando el Hook de Estado

useState permite agregar y manejar estados en componentes funcionales.

Explicación:

- **useState(0):** Inicializa el estado count con 0.
- **setCount:** Función para actualizar el estado.

```
import { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h1>Contador: {count}</h1>
      <button onClick={() => setCount(count +
1)}>Incrementar</button>
    </div>
  );
}

export default Counter;
```

Usando el Hook de Efecto

useEffect maneja efectos secundarios como:

- Peticiones HTTP.
- Interacciones con el DOM.
- Subscripciones y limpieza.

Explicación:

- Se actualiza cada segundo.
- Se limpia el intervalo al desmontar el componente.

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

function Clock() {
  const [time, setTime] = useState(new
Date().toLocaleTimeString());

  useEffect(() => {
    const timer = setInterval(() => {
      setTime(new Date().toLocaleTimeString());
    }, 1000);

    return () => clearInterval(timer); // Limpieza
  }, []);

  return <h1>Hora Actual: {time}</h1>;
}

export default Clock;
```

Reglas de los Hooks

Llamar Hooks solo al nivel superior.

✅ Correcto:

```
useState(0);
useEffect(() => {}, []);
```

❌ Incorrecto:

```
if (condition) {
  useState(0); // Error
}
```

Usar Hooks solo en funciones de React.

- ✅ Dentro de componentes funcionales o Hooks personalizados.
- ❌ Fuera de estos contextos.

Construyendo Hooks Personalizados

Un Hook personalizado reutiliza lógica común en diferentes componentes.

Ejemplo: useFetch para peticiones HTTP

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

function useFetch(url) {
  const [data, setData] = useState(null);
  const [error, setError] = useState(null);
  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch(url);
        if (!response.ok) throw new Error('Error al obtener datos');
        const result = await response.json();
        setData(result);
      } catch (err) {
        setError(err.message);
      }
    };
    fetchData();
  }, [url]);

  return { data, error };
}

export default useFetch;
```

Construyendo Hooks Personalizados

Uso del Hook Personalizado:

Ventajas:

- Reutilización de lógica común.
- Mejor organización del código

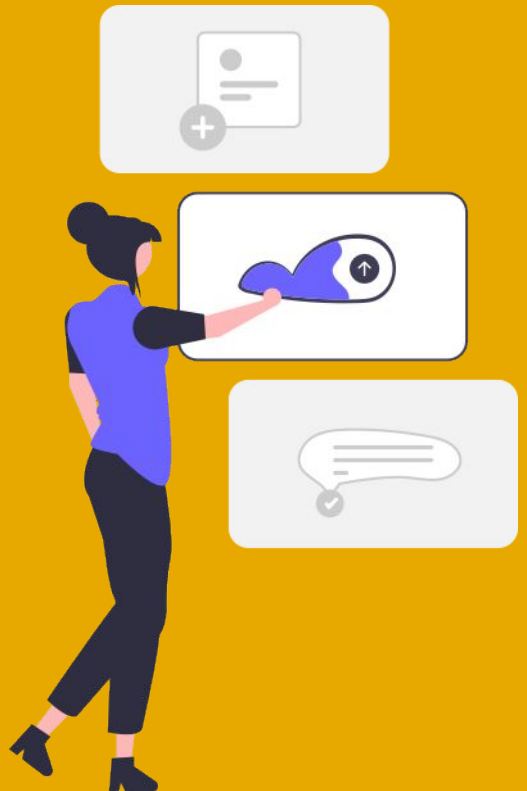
```
import useFetch from './hooks/useFetch';

function App() {
  const { data, error } =
useFetch('https://jsonplaceholder.typicode.com/posts');

  if (error) return <p>Error: {error}</p>;
  if (!data) return <p>Cargando...</p>;

  return (
    <ul>
      {data.map((post) => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
}

export default App;
```



Manejo de Errores

adalid
aspasia

Mecanismos Disponibles en React para Manejo de Errores

- **try-catch**
  - Para errores asíncronos en efectos secundarios o funciones.
- **Error Boundaries**
  - Captura errores en componentes hijos durante el renderizado.
- **Hooks Personalizados**
  - Centralizan el manejo de errores reutilizables.

Reconocer Mecanismos para el Manejo de Errores

Captura de Errores en Efectos Secundarios

Ejemplo con **try-catch** en una petición HTTP:

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

function DataFetcher() {
  const [data, setData] = useState([]);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch('https://jsonplaceholder.typicode.com/posts');
        if (!response.ok) throw new Error('Error al obtener datos');
        const result = await response.json();
        setData(result);
      } catch (err) {
        setError(err.message);
      }
    };
    fetchData();
  }, []);

  if (error) return <p>Error: {error}</p>;
  return <ul>{data.map((item) => <li key={item.id}>{item.title}</li>)}</ul>;
}
```

Reconocer Mecanismos para el Manejo de Errores

Componentes de Error Boundary

React maneja errores de renderizado con Error **Boundaries**.

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError() {
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    console.error('Error capturado:', error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return <h1>Algo salió mal.</h1>;
    }
    return this.props.children;
  }
}

export default ErrorBoundary;
```

Reconocer Mecanismos para el Manejo de Errores

Así se vería su implementación

```
import ErrorBoundary from './ErrorBoundary';
import DataFetcher from './DataFetcher';

function App() {
  return (
    <ErrorBoundary>
      <DataFetcher />
    </ErrorBoundary>
  );
}
```

Utilizando Hooks para Resolver Problemas

Ejemplo: Hook Personalizado para Manejo de Errores en Peticiones

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

function useFetch(url) {
  const [data, setData] = useState(null);
  const [error, setError] = useState('');
  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch(url);
        if (!response.ok) throw new Error('Error al cargar los datos.');
        const result = await response.json();
        setData(result);
      } catch (err) {
        setError(err.message);
      }
    };

    fetchData();
  }, [url]);

  return { data, error };
}

export default useFetch;
```

Utilizando Hooks para Resolver Problemas

Uso del Hook:

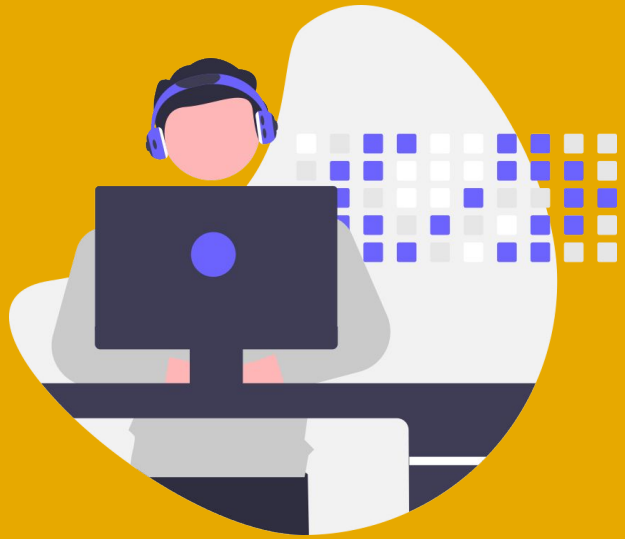
```
import useFetch from './hooks/useFetch';

function App() {
  const { data, error } = useFetch('https://jsonplaceholder.typicode.com/posts');

  if (error) return <p>Error: {error}</p>;
  if (!data) return <p>Cargando...</p>;

  return <ul>{data.map((item) => <li key={item.id}>{item.title}</li>)}</ul>;
}
```

Pongamos a prueba lo aprendido 😊!!!



Adalid aspasia

Ejercicio Guiado: Lista de Tareas con React y Hooks

Crea una aplicación simple de lista de tareas usando React que permita:

1. Agregar tareas.
2. Mostrar la lista de tareas.
3. Eliminar tareas.
4. Manejar errores simulados (por ejemplo, al intentar agregar una tarea vacía).

Ejercicio Guiado: Lista de Tareas con React y Hooks

Objetivo del Ejercicio

- Usar useState para gestionar el estado de las tareas.
- Usar useEffect para simular la carga inicial de tareas.
- Aplicar un Hook personalizado para centralizar la lógica.
- Manejar errores de forma básica y mostrar mensajes descriptivos.

```
task-manager/
├── src/
│   ├── components/
│   │   ├── TaskList.jsx
│   │   ├── TaskForm.jsx
│   ├── main.jsx
│   └── App.jsx
├── index.html
└── vite.config.js
```

Ejercicio Guiado: Lista de Tareas con React y Hooks

Modifica App.jsx

```
import { useState, useEffect } from 'react';
import TaskForm from './components/TaskForm';
import TaskList from './components/TaskList';

function App() {
  const [tasks, setTasks] = useState([]);
  const [error, setError] = useState('');

  // Simula la carga inicial de tareas
  useEffect(() => {
    setTasks(['Tarea 1', 'Tarea 2']);
  }, []);

  const addTask = (task) => {
    if (!task) {
      setError('La tarea no puede estar vacía');
      return;
    }
    setTasks([...tasks, task]);
    setError('');
  };

  const deleteTask = (index) => {
    const updatedTasks = tasks.filter((_, i) => i !== index);
    setTasks(updatedTasks);
  };

  return (
    <div className="container mt-4">
      <h1 className="text-center">Lista de Tareas</h1>
      {error && <p className="text-danger">{error}</p>}
      <TaskForm addTask={addTask} />
      <TaskList tasks={tasks} deleteTask={deleteTask} />
    </div>
  );
}

export default App;
```

Ejercicio Guiado: Lista de Tareas con React y Hooks

Crea el Componente **TaskForm** en **src/components/TaskForm.jsx**

```jsx
import { useState } from 'react';

function TaskForm({ addTask }) {
  const [task, setTask] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    addTask(task);
    setTask('');
  };

  return (
    <form onSubmit={handleSubmit} className="d-flex gap-2 mb-3">
      <input
        type="text"
        className="form-control"
        placeholder="Nueva tarea"
        value={task}
        onChange={(e) => setTask(e.target.value)}
      />
      <button type="submit" className="btn btn-primary">
        Agregar
      </button>
    </form>
  );
}

export default TaskForm;
```

Ejercicio Guiado: Lista de Tareas con React y Hooks

Crea el Componente TaskList en **src/components/TaskList.jsx**

```
import React from 'react';

function TaskList({ tasks, deleteTask }) {
  return (
    <ul className="list-group">
      {tasks.map((task, index) => (
        <li key={index} className="list-group-item d-flex
justify-content-between align-items-center">
          {task}
          <button className="btn btn-danger btn-sm" onClick={() =>
deleteTask(index)}>
            Eliminar
          </button>
        </li>
      ))}
    </ul>
  );
}

export default TaskList;
```

Ejercicio Guiado: Lista de Tareas con React y Hooks

Resultados Esperados

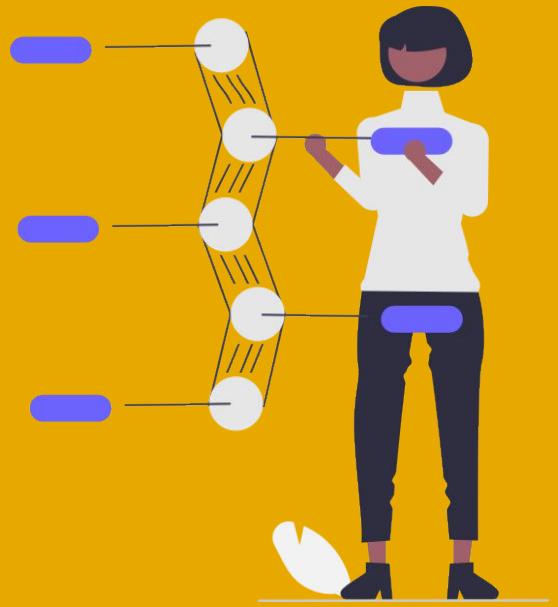
1. **Agregar Tareas:** Permite ingresar y mostrar nuevas tareas en la lista.
2. **Eliminar Tareas:** Los usuarios pueden eliminar tareas específicas.
3. **Manejo de Errores:** Muestra un mensaje si se intenta agregar una tarea vacía.

Ejercicio Guiado: Lista de Tareas con React y Hooks

Ejecuta el proyecto:

```
npm run dev
```

Resumen de lo aprendido 🧑‍🎓



Adalid aspasia

Resumen de lo aprendido

- Aprendiste qué son los Hooks en React, cómo gestionar el estado con useState y manejar efectos secundarios con useEffect.
- Implementaste Hooks personalizados para reutilizar lógica común y optimizar el desarrollo en aplicaciones React.
- Descubriste cómo manejar y controlar errores en React, garantizando una experiencia de usuario fluida y segura.
- Aplicaste las reglas y mejores prácticas de los Hooks para mantener un código limpio, eficiente y sin errores comunes.

GRACIAS POR TU ATENCIÓN

Nos vemos en la próxima clase

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO