



Buenos  
Aires  
Ciudad

Agencia de Habilidades  
para el Futuro

<Talento  
Tech />

# ReactJS

---

**Clase 10 | Creación de Productos –  
Formulario y Validación**

# ¡Les damos la bienvenida!



Vamos a comenzar a grabar la clase.

## Índice

---

**1****2****3**

### Autenticación de usuarios

- Introducción a la Autenticación de Usuarios
- Implementación de formulario de login.
- Manejo de autenticación con tokens (simulada).
- Protección de rutas usando Context API para la autenticación.

### Creación de Productos - Formulario y Validación

- **Creación de formulario para agregar productos.**
- **Validación de datos de formularios.**
- **Implementación de la funcionalidad para agregar productos.**

### Leer, Actualizar y Eliminar productos

- Implementación de la funcionalidad para leer productos desde el estado o API.
- Actualización y eliminación de productos.
- Validación de los formularios de edición

# Objetivos de la Clase

## **1** Diseño de Formularios

Crear formularios en React para agregar nuevos productos.

## **2** Validación de Datos

Implementar validaciones para garantizar la consistencia de los datos ingresados.

## **3** Integración con API

Conectar con MockAPI para agregar productos mediante solicitudes.

# **Creación de formulario para agregar productos**



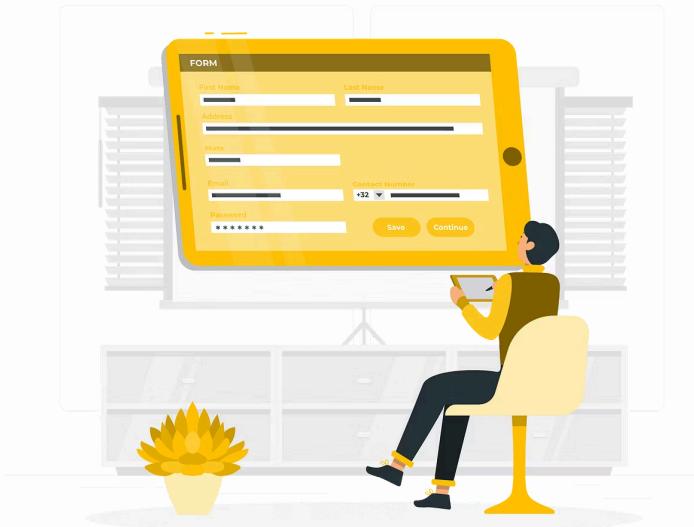
# Creación del Formulario

## 1 Estado Local

Definimos un estado para almacenar los datos del formulario usando useState.

## 2 Manejo de Eventos

Implementamos onChange para actualizar el estado y onSubmit para procesar los datos.



# Estructura del Estado

```
const [producto, setProducto] = useState({  
    nombre: '',  
    precio: '',  
    descripcion: '',  
});
```

# Función handleChange

```
const handleChange = (e) => {
  const { name, value } = e.target;
  setProducto({ ...producto, [name]: value });
};
```



# Estructura del Formulario

## Campos

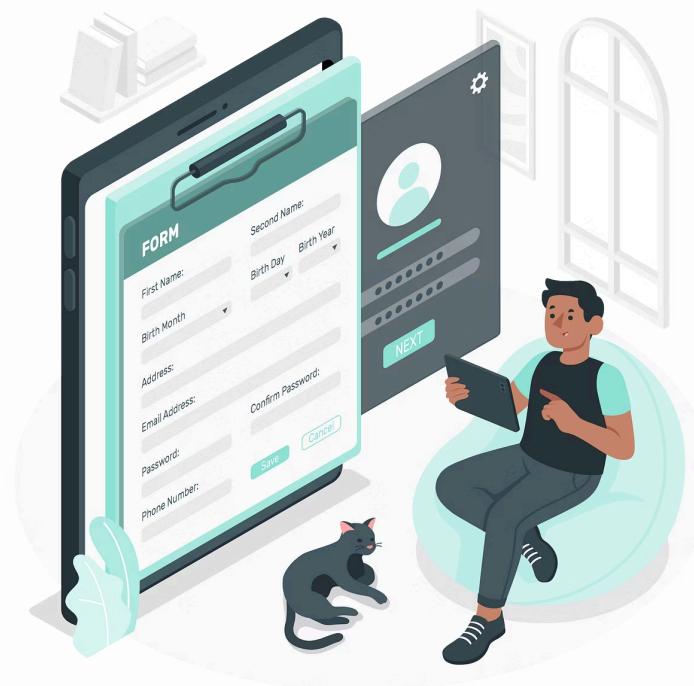
Nombre, precio y descripción del producto.

## Eventos

onChange para cada campo y onSubmit para el formulario.

## Botón

Para enviar los datos del formulario.



# Componente FormularioProducto

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

function FormularioProducto({ onAgregar }) {
  const [producto, setProducto] = useState({
    nombre: '',
    precio: '',
    descripcion: '',
  });
  const handleChange = (e) => {
    const { name, value } = e.target;
    setProducto({ ...producto, [name]: value });
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    onAgregar(producto); // Llamada a la función para agregar el producto
    setProducto({ nombre: '', precio: '', descripcion: '' }); // Limpiar el formulario
  };
}
```

# Componente FormularioProducto

```
return ( <form onSubmit={handleSubmit}>
  <h2>Agregar Producto</h2>
  <div>
    <label>Nombre:</label>
    <input
      type="text" name="nombre" value=
      {producto.nombre} onChange={handleChange}
      required/>
  </div>
  <div>
    <label>Precio:</label>
    <input type="number" name="precio"
      value={producto.precio} onChange=
      {handleChange} required
      min="0"/>
  </div>
```

```
<div>
  <label>Descripción:</label>
  <textarea
    name="descripcion"
    value={producto.descripcion}
    onChange={handleChange}
    required
  />
</div>
<button type="submit">Agregar
Producto</button>
</form>
);
}

export default FormularioProducto;
```

# **Validación de datos de formularios**

---

# Validación de Datos

---

## 1 Importancia

Asegura consistencia y previene errores en los datos ingresados.

## 2 Tipos de Validaciones

Campos requeridos, validación de números, longitud mínima y máxima.



# Implementación de Validaciones

---

## 1 Estado de Errores

Creamos un estado para almacenar los mensajes de error.

## 2 Función de Validación

Implementamos una función para verificar los datos antes del envío.

## 3 Mostrar Errores

Renderizamos los mensajes de error junto a los campos correspondientes.

# Estado de Errores

---

```
const [errores, setErrores] =  
  useState({});
```



# Función de Validación

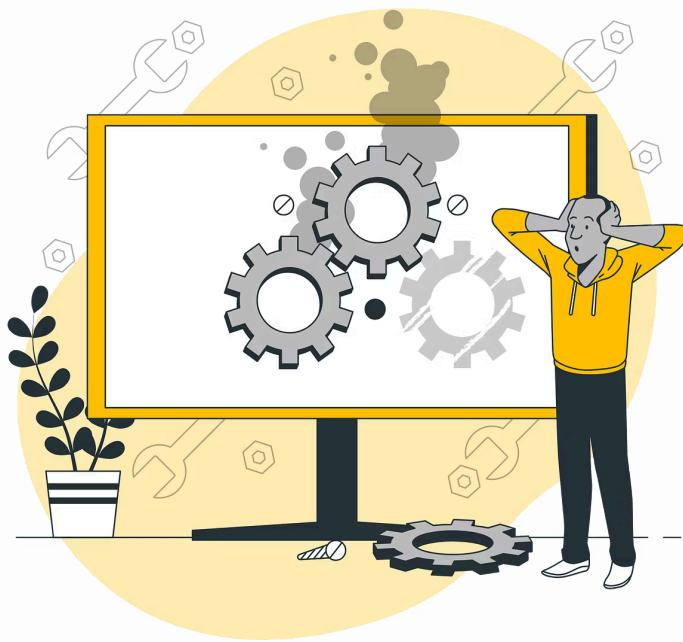
---

```
const validarFormulario = () => {
  const nuevosErrores = {};
  // ... lógica de validación ...
  setErrores(nuevosErrores);
  return Object.keys(nuevosErrores).length === 0;
};
```

# Manejo de Envío del Formulario

```
const handleSubmit = (e) => {
  e.preventDefault();
  if (validarFormulario()) {
    onAgregar(producto);
    // ... limpiar formulario ...
  }
};
```





# Renderizado de Errores

---

```
{errores.nombre && {errores.nombre}}  
{errores.precio && {errores.precio}}  
{errores.descripcion && {errores.descripcion}}
```

## **Componente ValidarFormulario**

```

const validarFormulario = () => {
  const [errores, setErrores] = useState({});
  const nuevosErrores = {};
  if (!producto.nombre.trim()) {
    nuevosErrores.nombre = 'El nombre es obligatorio.';
  }
  nuevosErrores.precio = 'El precio debe ser mayor a 0.';
}

```

# Conexión con MockAPI

**1**

## Configuración

```

if (!producto.descripcion.trim() || producto.descripcion.length < 10) {
  nuevosErrores.descripcion = 'La descripción debe tener al menos 10 caracteres.';
}

```

**2**

## Flujo de Datos

```

setErrorEnviamos datos del formulario mediante POST a
return MockAPI.es(nuevosErrores).length ===
0;;

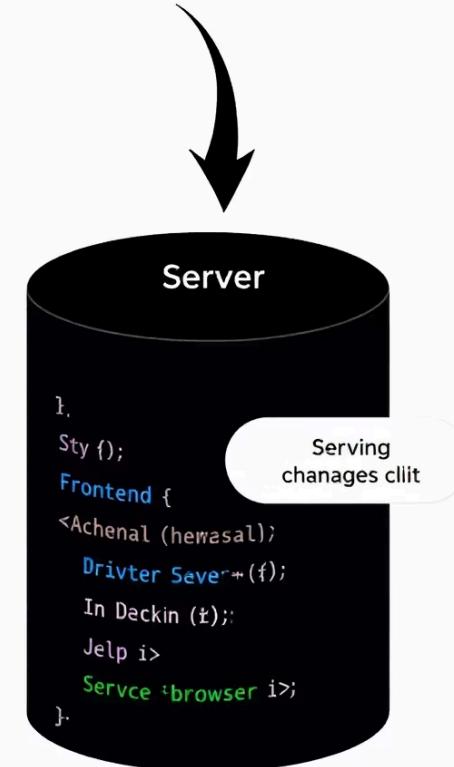
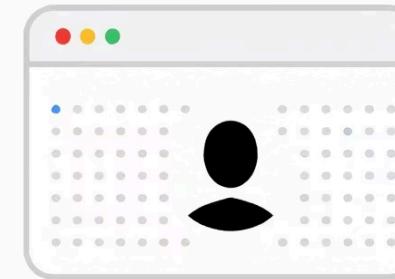
```

```

const handleSubmit = (e) => {
  e.preventDefault();
  if (!producto.nombre.trim() || producto.descripcion.length < 10) {
    setErrores({
      nombre: 'El nombre es obligatorio.',
      precio: 'El precio debe ser mayor a 0.',
      descripcion: 'La descripción debe tener al menos 10 caracteres.'
    });
}

```

Frontend



# Función para Agregar Productos

```
const agregarProducto = async (producto)
=> {
  try {
    const respuesta = await
    fetch('https://mockapi.io/api/v1/productos
      ', {
        method: 'POST',
        headers: {
          'Content-Type':
            'application/json',
        },
        body: JSON.stringify(producto),
      });
  
```

```
    if (!respuesta.ok) {
      throw new Error('Error al agregar el
      producto.');
    }
    const data = await respuesta.json();
    console.log('Producto agregado:',
    data);
    alert('Producto agregado
    correctamente');
  } catch (error) {
    console.error(error.message);
    alert('Hubo un problema al agregar el
    producto.');
  }
};
```

# Integración en el Componente Principal

```
import React from 'react';
import FormularioProducto from
'./FormularioProducto';
function App() {
  const agregarProducto = async (producto)
=> {
  try {
    const respuesta = await
fetch('https://mockapi.io/api/v1/productos
', {
  method: 'POST',
  headers: {
    'Content-Type':
    'application/json',
  },
  body: JSON.stringify(producto),
});
  
```

```
if (!respuesta.ok) {
  throw new Error('Error al agregar
el producto.');
}
const data = await respuesta.json();
console.log('Producto agregado:',
data);
alert('Producto agregado
correctamente');
} catch (error) {
  console.error(error.message);
  alert('Hubo un problema al agregar
el producto.');
}
};
```

```
return (
  <div>
    <h1>Gestión de Productos</h1>
    <FormularioProducto onAgregar={agregarProducto} />
  </div>
);
export default App;
```

# Reflexión Final

Hoy aprendimos a construir una funcionalidad completa para agregar productos en React. Este flujo incluyó:

1. Crear un formulario con manejo de estado.
2. Validar los datos ingresados por el usuario.
3. Enviar estos datos a una API (MockAPI) para almacenarlos.

Este proceso no solo refuerza los conceptos básicos de React, sino que también introduce prácticas fundamentales para desarrollar aplicaciones interactivas que interactúan con APIs.



# Recursos Adicionales

---



## Documentación de MockAPI

[Documentación oficial de MockAPI](#)



## Fetch API

[Guía sobre el uso del método fetch para solicitudes HTTP.](#)



# Preguntas para Reflexionar

## **Validaciones Adicionales**

¿Qué validaciones podrías agregar para mejorar la calidad de los datos?

## **Manejo de Errores**

¿Cómo manejarías respuestas lentas o errores de la API?

## **Edición de Productos**

¿Cómo implementarías la edición de productos existentes?

# Próximos Pasos

1

## Lectura de Productos

Implementar funcionalidad para leer productos desde el estado o API.

2

## Actualización y Eliminación

Desarrollar funciones para actualizar y eliminar productos existentes.

3

## Validación en Edición

Aplicar validaciones a los formularios de edición de productos.



# Ejercicios Prácticos

---



## TalentoLab – Proyecto Final

¡El cliente de Talento Lab está emocionado con los avances! Ahora necesita una funcionalidad que permita agregar nuevos productos al catálogo de su sitio. Este proceso debe incluir un formulario amigable con validaciones para garantizar que los datos ingresados sean consistentes.



# Ejercicio Práctico

Obligatorio



**Silvia**

Product Owner

**Descripción de tu tarea:**

## **Objetivos:**

1. Diseñar un formulario controlado en React para agregar productos.
2. Implementar validaciones dinámicas para los datos ingresados.
3. Conectar la aplicación con MockAPI para almacenar los nuevos productos.



# Ejercicio Práctico

Obligatorio

## Requerimientos

1

### Crear un Formulario Controlado:

- Diseña un formulario en React que permita ingresar:
  - Nombre del producto.
  - Precio (en números).
  - Descripción (mínimo 10 caracteres).
- Asegúrate de manejar el estado del formulario mediante el hook useState.

2

### Validaciones del Formulario

- Implementa las siguientes reglas de validación:
  - Todos los campos son obligatorios.
  - El precio debe ser mayor a 0.
  - La descripción debe tener al menos 10 caracteres.
- Muestra mensajes de error junto a los campos correspondientes.



# Ejercicio Práctico

Obligatorio

## Requerimientos:

### 1 Crear un Formulario Controlado:

- Diseña un formulario en React que permita ingresar:
  - Nombre del producto.
  - Precio (en números).
  - Descripción (mínimo 10 caracteres).
- Asegúrate de manejar el estado del formulario mediante el hook useState.

### 2 Validaciones del Formulario

- Implementa las siguientes reglas de validación:
  - Todos los campos son obligatorios.
  - El precio debe ser mayor a 0.
  - La descripción debe tener al menos 10 caracteres.
- Muestra mensajes de error junto a los campos correspondientes.

### 3 Conectar con MockAPI

- Configura una función para enviar los datos del producto mediante una solicitud POST a MockAPI.

- Si el producto se agrega correctamente:
  - Limpia el formulario.
  - Muestra un mensaje de éxito.
- Si ocurre un error:
  - Muestra un mensaje de error en pantalla.



# Ejercicio Práctico

Obligatorio

---

## Requerimientos:

1

### Validaciones del Formulario

- Implementa las siguientes reglas de validación:
  - Todos los campos son obligatorios.

- El precio debe ser mayor a 0.
- La descripción debe tener al menos 10 caracteres.
- Muestra mensajes de error junto a los campos correspondientes.



