«Talento Tech»

Desarrollo Web 3

Clase 06

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 6 | MockAPI y CRUD

Temario:

- Integración de React App y una API usando MockAPI
- Conociendo CRUD y cómo implementarlo

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Integración de React App y una API usando MockAPI

¿Qué es MockAPI.io?

MockAPI.io es una plataforma en línea que permite a los desarrolladores crear **APIs** ficticias (mocks) de manera rápida y sencilla. Con MockAPI.io, puedes simular endpoints de API y definir respuestas simuladas para probar y desarrollar aplicaciones sin depender de una API real.

Beneficios de MockAPI.io:

- Desarrollo sin Dependencias Externas
- Pruebas de Integración
- Rápido Prototipado
- Entorno de Desarrollo Aislado
- Personalización de Respuestas
- Facilita la Colaboración
- Sin Necesidad de Infraestructura

En resumen, MockAPI.io es una herramienta valiosa para el desarrollo de software al facilitar la creación rápida de APIs simuladas. Ofrece flexibilidad, velocidad y eficiencia al permitir a los desarrolladores avanzar en sus proyectos sin depender de APIs externas completamente desarrolladas.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

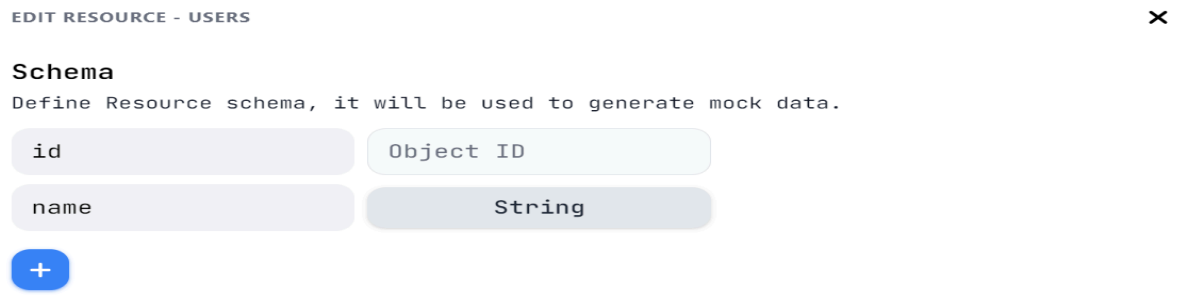
Uso de MockAPI en una App React con Vite

1) Regístrate en MockAPI.io:

- Ve a MockAPI.io y regístrate para obtener una cuenta gratuita.
- Crea una nueva API en el panel de control.

2) Obtén la URL de tu API:

Después de crear la API en MockAPI.io, obtén la URL proporcionada para tu API.
En la pagina deberias tener una vista similar a la siguiente:



3) Configuración Inicial con Vite:
Asegúrate de tener Vite instalado en tu entorno de desarrollo.
Puedes crear un nuevo proyecto Vite con el siguiente comando:

```
npx create-vite my-react-app --template react
```

4) Cambia al directorio de tu nueva aplicación Vite:

```
cd my-react-app
```

5) Crea un Componente React:
En tu aplicación Vite, crea un componente que consumirá la API de MockAPI.io.
Por ejemplo, crea un archivo llamado UserList.js:

```
// UserList.js
import React, { useState, useEffect } from 'react';

const UserList = () => {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const apiUrl = "URL_DE_TU_API"; // Reemplaza con la URL de tu API de      MockAPI.io

  useEffect(() => {
    fetch(apiUrl)
      .then((response) => response.json())
      .then((data) => setUsers(data))
      .catch((error) => console.error("Error al obtener usuarios:", error))
      .finally(() => {
        setLoading(false);
      });
  }, [apiUrl]);

  console.log(loading);

  return (
    <div>
```

```
      <h2>Lista de Usuarios</h2>
      {loading ? (
        <h1>Cargando...</h1>
      ) : (
        <ul>
          {users.map((user) => (
            <li key={user.id}>{user.name}</li>
          ))}
        </ul>
      )}
    </div>
  );

  };

export default UserList;
```

6) Integra el Componente en tu App:
Integra el componente UserList en tu aplicación principal.

```
// App.js
import React from 'react';
import UserList from './UserList';

const App = () => {
  return (
    <div>
      <h1>App React con MockAPI.io</h1>
      <UserList />
    </div>
  );
};

export default App;
```

7) Ejecuta tu App:
Asegúrate de tener la URL correcta de tu API en el archivo UserList.js.

Ejecuta tu aplicación Vite:

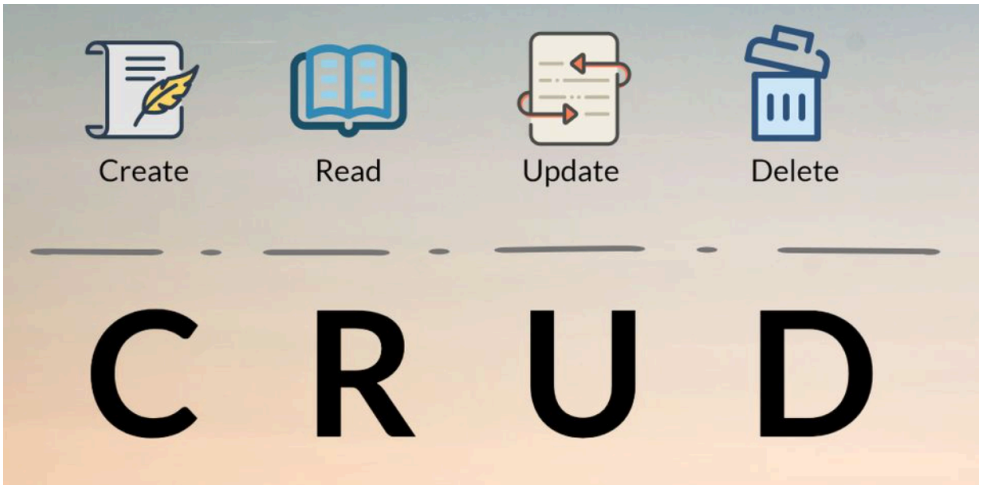
```
npm run dev
```

- Visita **http://localhost:3000** en tu navegador para ver la lista de usuarios obtenida de MockAPI.io.
- Recuerda reemplazar **'URL_DE_TU_API'** con la URL proporcionada por MockAPI.io.

Con estos pasos, deberías poder consumir datos de tu API de MockAPI.io en tu aplicación React y con Vite.

Conociendo CRUD y cómo implementarlo

CRUD en pocas palabras:



CRUD es un acrónimo que representa las cuatro operaciones básicas en la gestión de datos:

- **Crear (Create):**
  Implica la creación de nuevos registros o elementos en una base de datos o sistema. En el contexto de una aplicación, sería agregar nuevos datos.

- **Leer (Read):**
  Implica la recuperación y visualización de datos existentes desde una base de datos o sistema. En una aplicación, sería la acción de obtener información existente.

- **Actualizar (Update):**
  Implica la modificación o actualización de registros o datos existentes en una base de datos o sistema. En una aplicación, sería la acción de editar o modificar información existente.

- **Eliminar (Delete):**
  Implica la eliminación de registros o datos existentes de una base de datos o sistema. En una aplicación, sería la acción de eliminar información existente.
  En resumen, CRUD representa las operaciones fundamentales para gestionar datos en sistemas informáticos: crear nuevos datos, leer información existente, actualizar datos existentes y eliminar información. Estas operaciones son esenciales para la interacción efectiva con bases de datos y sistemas de almacenamiento de datos.

Usando CRUD en nuestra React App

Utilizaremos un formulario simple para crear y actualizar usuarios, y botones para eliminar usuarios.

Paso 1: Importar React y configurar el estado inicial

Buenos Aires aprende
Agencia de Habilidades para el Futuro

```
//UserList.jsx
import React, { useState, useEffect } from 'react';

const UserList = () => {
  // Declarar estados iniciales
  const [users, setUsers] = useState([]);
  const [newUserName, setNewUserName] = useState('');
  const [selectedUser, setSelectedUser] = useState(null);
  const [loading, setLoading] = useState(true);

  // Declarar la URL de la API
  const apiUrl = "https://64e665b109e64530d17ff5c8.mockapi.io/users";
// Reemplaza con la URL de tu API de MockAPI.io
```

Paso 2: Obtener usuarios al cargar el componente

```
  useEffect(() => {
    // Realizar solicitud GET al montar el componente
    fetch(apiUrl)
      .then((response) => response.json())
      .then((data) => setUsers(data))
      .catch((error) => console.error("Error al obtener usuarios:", error))
      .finally(() => {
        setLoading(false);
      });
  }, [apiUrl]);
```

Paso 3: Crear usuario

```
const handleCreateUser = () => {
  // Realizar solicitud POST para agregar un nuevo usuario
  fetch(apiUrl, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ name: newUserName }),
  })
    .then((response) => response.json())
    .then(() => {
      // Realizar cualquier acción adicional si es necesario
      setNewUserName('');
    })
    .then(() => {
      // Realizar solicitud GET después de crear un usuario para obtener datos
actualizados
      return fetch(apiUrl);
    })
    .then((response) => response.json())
    .then((data) => setUsers(data))
    .catch((error) => console.error(error.message));
};
```

Paso 4: Actualizar usuario

```
const handleUpdateUser = () => {
  if (!selectedUser) return;

  // Realizar solicitud PUT para actualizar un usuario existente
  fetch(`${apiUrl}/${selectedUser.id}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
```

```
    },
    body: JSON.stringify({ name: newUserName }),
  })
    .then((response) => response.json())
    .then((updatedUser) => {
      // Actualizar el estado con el usuario actualizado
      setUsers(users.map((user) => (user.id === selectedUser.id ? updatedUser : user)));
      setNewUserName('');
      setSelectedUser(null);
    })
    .catch((error) => console.error('Error al actualizar usuario:', error));
};
```

Paso 5: Eliminar usuario

```
const handleDeleteUser = (userId) => {
  // Realizar solicitud DELETE para eliminar un usuario
  fetch(`${apiUrl}/${userId}`, {
    method: 'DELETE',
  })
    .then(() => {
      // Actualizar el estado excluyendo al usuario eliminado
      setUsers(users.filter((user) => user.id !== userId));
      setNewUserName('');
      setSelectedUser(null);
    })
    .catch((error) => console.error('Error al eliminar usuario:', error));
};
```

Paso 6: Renderizar la interfaz de usuario

```
return (
  <div>
    <h2>Lista de Usuarios</h2>
    {loading ?
    (<h1>Cargando...</h1>) :
    ( <ul>
      {users.map((user) => (
        <li key={user.id}>
          {user.name}
          <button onClick={() => setSelectedUser(user)}>Seleccionar para editar</button>
          <button onClick={() => handleDeleteUser(user.id)}>Eliminar</button>
        </li>
      ))}
    </ul>)}

    <div>
      <input
        type="text"
        value={newUserName}
        onChange={(e) => setNewUserName(e.target.value)}
        placeholder="Nombre del Usuario"
      />
      {selectedUser ? (
        <button onClick={handleUpdateUser}>Actualizar Usuario</button>
      ) : (
        <button onClick={handleCreateUser}>Crear Usuario</button>
      )}
    </div>
  </div>
 );
};

export default UserList;
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Este ejemplo ahora se incluyen todas las operaciones CRUD: Crear, Leer, Actualizar y Eliminar. Puedes crear nuevos usuarios, editar usuarios existentes y eliminar usuarios de la lista. Asegúrate de ajustar las URL de la API según tu configuración.


Buenos Aires
aprende
Buenos
Aires
Ciudad