Análisis Comparativo: Fetch API vs. Axios

Para la toma de decisiones en proyectos web

Vicente Matus

7 de septiembre de 2025

Resumen

Este documento presenta un análisis detallado de las diferencias, ventajas y desventajas entre la **Fetch API** nativa de los navegadores y la librería **Axios**. El objetivo es proporcionar una base sólida para decidir cuál de estas tecnologías es más adecuada para realizar peticiones HTTP en nuestra aplicación web, considerando factores como la sintaxis, el manejo de errores, la compatibilidad y las funcionalidades avanzadas.

Índice

1. Introducción

Tanto Fetch como Axios son herramientas utilizadas para realizar peticiones de red (HTTP) en aplicaciones JavaScript. A primera vista, parecen cumplir la misma función, pero existen diferencias fundamentales en su implementación, sintaxis y características que pueden tener un gran impacto en la mantenibilidad y escalabilidad de un proyecto.

- **Fetch API:** Es una interfaz moderna, nativa del navegador, para realizar peticiones de red. Está basada en Promesas y es el estándar actual para las comunicaciones asíncronas en la web.
- **Axios:** Es una librería de terceros, muy popular, que también se basa en Promesas. Proporciona una capa de abstracción sobre las peticiones HTTP, simplificando muchas tareas comunes y ofreciendo funcionalidades adicionales.

2. Sintaxis y Uso Básico

2.1. Fetch API

La sintaxis de Fetch es directa pero requiere un paso adicional para procesar la respuesta. El método `fetch()` devuelve una Promesa que resuelve en el objeto `Response`. Para obtener los datos en formato JSON, es necesario llamar al método `.json()`.

```
fetch('https://api.example.com/data')
  .then(response => {
    // Es necesario verificar si la respuesta fue exitosa (status 2xx)
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    return response.json(); // Parsea la respuesta a JSON
  })
  .then(data => console.log(data))
  .catch(error => console.error('Hubo un problema con la peticion fetch:', error
    ));
```

Listing 1: Petición GET básica con Fetch

2.2. Axios

Axios simplifica este proceso. La respuesta ya incluye los datos parseados en la propiedad `data`.

```
axios.get('https://api.example.com/data')
  .then(response => {
    // Los datos ya vienen parseados en response.data
    console.log(response.data);
  })
  .catch(error => {
    // Axios maneja errores HTTP (4xx, 5xx) en el .catch()
    console.error('Hubo un problema con la peticion axios:', error);
  });
```

Listing 2: Petición GET básica con Axios

3. Manejo de Errores

Esta es una de las diferencias más importantes.

- **Fetch API:** Una promesa de Fetch **no** es rechazada por errores HTTP (como 404 o 500). Solo se rechaza si hay un fallo de red. Esto obliga al desarrollador a comprobar manualmente el estado de la respuesta a través de la propiedad `response.ok` o `response.status`.
- **Axios:** Rechaza la promesa automáticamente si recibe un código de estado fuera del rango 2xx. Esto simplifica el manejo de errores, ya que tanto los errores de red como los errores HTTP se pueden capturar en el mismo bloque `.catch()`.

4. Transformación de Datos

4.1. Envío de datos (POST/PUT)

- **Fetch API:** Requiere que los objetos JavaScript sean convertidos manualmente a una cadena JSON usando `JSON.stringify()` y que se establezca la cabecera `'Content-Type': 'application/json'`.
- **Axios:** Realiza esta transformación automáticamente. Simplemente se pasa el objeto JavaScript como segundo argumento.

```
// Fetch
const data = { name: 'John Doe' };
fetch('https://api.example.com/users', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
});

// Axios
axios.post('https://api.example.com/users', data);
```

Listing 3: Petición POST con Fetch vs. Axios

4.2. Recepción de datos

Como se vio anteriormente, Fetch requiere llamar a `response.json()`, mientras que Axios entrega los datos ya parseados en `response.data`.

5. Funcionalidades Avanzadas

5.1. Interceptores

- **Axios:** Ofrece ïnterceptores", que permiten ejecutar código antes de que una petición sea enviada o después de que una respuesta sea recibida. Esto es extremadamente útil para tareas como añadir tokens de autenticación a todas las peticiones o para un manejo global de errores.
- **Fetch API:** No tiene un sistema de interceptores nativo. Para lograr una funcionalidad similar, se debe crear una función "wrapper.º un "factory"que encapsule la lógica de Fetch.

```
// A adir un interceptor de peticiones
axios.interceptors.request.use(config => {
  const token = localStorage.getItem('authToken');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
```

```
  }
  return config;
});
```

Listing 4: Ejemplo de interceptor en Axios para añadir un token

5.2. Cancelación de Peticiones

- **Fetch API:** Soporta la cancelación a través de la `AbortController` API. Es un mecanismo potente pero requiere más código.
- **Axios:** Proporciona una forma más sencilla de cancelar peticiones, que internamente también puede usar `AbortController`.

6. Compatibilidad

- **Fetch API:** Soportado por todos los navegadores modernos. En entornos más antiguos o en Node.js, puede requerir un "polyfill"(ej. `node-fetch`).
- **Axios:** Es isomórfico, lo que significa que funciona tanto en el navegador (usando XHR) como en Node.js (usando el módulo `http`) sin necesidad de configuración adicional.

7. Tabla Comparativa

Característica	**Fetch API**	**Axios**
Origen	Nativa del navegador	Librería de terceros
Manejo de JSON	Manual (`response.json()`)	Automático (`response.data`)
Manejo de errores	No rechaza en 4xx/5xx	Rechaza en 4xx/5xx
Interceptores	No (requiere wrapper)	Sí, nativos
Cancelación	Vía `AbortController`	API simplificada
Compatibilidad	Navegadores modernos	Navegadores y Node.js
Protección XSRF	Manual	Integrada
Progreso de carga	No directamente	Soportado

Cuadro 1: Resumen de diferencias entre Fetch y Axios.

8. Conclusión y Recomendación

- **Usa Fetch API si...**
  - Quieres minimizar las dependencias en tu proyecto.
  - Solo necesitas realizar peticiones simples y no te importa escribir un poco más de código para el manejo de errores y datos.
  - Estás trabajando en un proyecto pequeño o en un entorno donde el tamaño del bundle es crítico.
- **Usa Axios si...**
  - Necesitas una API más cómoda y con menos código repetitivo.
  - Tu aplicación requiere funcionalidades avanzadas como interceptores, manejo global de errores o cancelación de peticiones de forma sencilla.

- Valoras la consistencia entre el código del cliente (navegador) y el servidor (Node.js).
- Estás trabajando en una aplicación de mediana a gran escala donde la mantenibilidad es clave.

Para la mayoría de los proyectos de escala media a grande, la comodidad y las características adicionales de **Axios** suelen justificar la inclusión de una dependencia más en el proyecto.

9. Recomendación para el Proyecto Sporthub Temuco

Basado en la descripción del proyecto Sporthub Temuco, una plataforma integral con Next.js que incluye autenticación, pagos, y paneles de administración, la recomendación es utilizar **Axios**. Las ventajas que ofrece se alinean directamente con la complejidad y los requisitos de la aplicación.

- **Entorno Isomórfico (Next.js):** Axios funciona de manera idéntica en el servidor (para el renderizado inicial y las funciones de datos de Next.js) y en el cliente (navegador). Esto simplifica el código y evita tener que manejar dos implementaciones diferentes.
- **Manejo Centralizado de Autenticación:** La plataforma requiere que los usuarios y administradores inicien sesión. Los **interceptores** de Axios son perfectos para añadir automáticamente tokens de autenticación a cada petición protegida, centralizando la lógica de seguridad y reduciendo la probabilidad de errores.
- **Manejo de Errores Robusto:** Para funcionalidades críticas como reservas y pagos, es vital tener un buen manejo de errores. Axios simplifica esto al tratar los errores de servidor (ej. 404, 500) como promesas rechazadas, permitiendo capturarlos de forma centralizada en un interceptor de respuestas para, por ejemplo, notificar al usuario o cerrar su sesión si el token expira.
- **Transformación Automática de Datos:** La aplicación enviará y recibirá constantemente datos en formato JSON. Axios maneja la serialización y deserialización de estos datos de forma automática, haciendo el código más limpio y menos propenso a errores manuales.

En resumen, para un proyecto de la escala y ambición de Sporthub Temuco, la inversión en añadir Axios como dependencia se justifica plenamente por el aumento en la productividad, mantenibilidad y robustez del código.

9.1. Instalación

Para agregar Axios a tu proyecto Next.js, ejecuta el siguiente comando en tu terminal:

```
npm install axios
```

Listing 5: Comando de instalación para npm

O si utilizas yarn:

```
yarn add axios
```

Listing 6: Comando de instalación para yarn

10. Tutorial: Conexión a una API Externa con Axios y TypeScript en Next.js

Para un proyecto robusto como Sporthub Temuco, utilizar Axios junto a TypeScript en Next.js es una decisión estratégica. Esta guía detalla el proceso para conectar a tu API externa de forma segura, tipada y mantenible.

10.1. Paso 1: Instalar y Configurar Variables de Entorno

Primero, añade Axios a tu proyecto y define las variables de entorno.

1. **Instala Axios:**

```
npm install axios
# O si usas yarn:
yarn add axios

```

Listing 7: Comando de instalación para npm o yarn

2. **Define la URL de la API:** Crea un archivo `.env.local` en la raíz de tu proyecto y añade la URL base de tu API. El prefijo `NEXT_PUBLIC_` es crucial para que la variable sea accesible en el navegador.

```
# URL base de tu API externa
NEXT_PUBLIC_API_URL=https://api.sporthub-temuco.cl

```

Listing 8: Contenido del archivo .env.local

10.2. Paso 2: Configurar CORS en el Servidor de la API

Este paso es una **configuración del backend** y no cambia. Tu API debe estar configurada para aceptar peticiones desde el dominio de tu frontend. Sin esto, el navegador bloqueará las llamadas.

```
const cors = require('cors');
// ...
const corsOptions = {
  origin: 'https://www.sporthub-temuco.cl' // URL de tu app Next.js
};
app.use(cors(corsOptions));
```

Listing 9: Configuración de CORS en el servidor de la API

10.3. Paso 3: Definir Tipos de Datos (Interfaces)

Define la "forma"de los datos que esperas de tu API. Esto habilita el autocompletado y la seguridad de tipos en todo tu proyecto.

```
// types/api.ts

export interface Cancha {
  id: number;
  nombre: string;
  direccion: string;
  precioPorHora: number;
  // ...otras propiedades
```

```
}

export interface Reserva {
  id: number;
  canchaId: number;
  fecha: string;
  usuarioId: string;
}

// Para el manejo de errores de Axios
export interface ApiErrorResponse {
  message: string;
  statusCode: number;
}
```

Listing 10: Ejemplo de interfaces en un archivo (ej: types/api.ts)

10.4. Paso 4: Crear una Instancia Centralizada de Axios

Esta es la mayor ventaja de Axios. Creas una única instancia pre-configurada que puedes usar en toda tu aplicación.

```
// lib/apiClient.ts
import axios, { type AxiosRequestConfig } from 'axios';

const apiClient = axios.create({
  baseURL: process.env.NEXT_PUBLIC_API_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Interceptor para anadir el token de autenticacion a cada peticion
apiClient.interceptors.request.use((config: AxiosRequestConfig) => {
  // Asegurarnos de que se ejecuta solo en el navegador
  if (typeof window !== 'undefined') {
    const token = localStorage.getItem('authToken');
    if (token && config.headers) {
      config.headers.Authorization = `Bearer ${token}`;
    }
  }
  return config;
});

export default apiClient;
```

Listing 11: Crear un cliente de API en lib/apiClient.ts

10.5. Paso 5: Realizar Peticiones Tipadas con la Instancia de Axios

Ahora, en lugar de usar `fetch`, importas y usas tu `apiClient`.

10.5.1. Ejemplo en un Componente de Cliente (con Tipos)

```
import apiClient from '@/lib/apiClient';
import { isAxiosError } from 'axios';
import type { Reserva, ApiErrorResponse } from '@/types/api';

```

```
async function crearReserva(canchaId: number): Promise<Reserva> {
  try {
    // Axios infiere el tipo de la respuesta si se lo indicas
    const response = await apiClient.post<Reserva>('/api/reservas', { canchaId });
    return response.data; // Los datos ya vienen en response.data
  } catch (error) {
    if (isAxiosError(error) && error.response) {
      // El error de Axios tiene una estructura predecible
      const errorData = error.response.data as ApiErrorResponse;
      throw new Error(errorData.message || 'Ocurrio un error desconocido.');
    }
    // Para otros tipos de errores (ej. de red)
    throw new Error('No se pudo conectar con el servidor.');
  }
}
```

Listing 12: Llamada Axios tipada desde un componente React/Next.js

10.5.2. Ejemplo en el Lado del Servidor (con Tipos)

Para obtener datos con `getServerSideProps`, el proceso es muy similar y limpio.

```
import type { GetServerSideProps, NextPage } from 'next';
import apiClient from '@/lib/apiClient';
import type { Cancha } from '@/types/api';

interface CanchasPageProps {
  canchas: Cancha[];
  error?: string;
}

export const getServerSideProps: GetServerSideProps<CanchasPageProps> = async (context) => {
  try {
    const response = await apiClient.get<Cancha[]>('/api/canchas');
    return {
      props: { canchas: response.data },
    };
  } catch (error) {
    // El interceptor no se ejecuta en el servidor, pero el manejo de errores es igual
    console.error('Error fetching data on server:', error);
    return { props: { canchas: [], error: 'No se pudieron cargar los datos.' } };
  }
};

const CanchasPage: NextPage<CanchasPageProps> = ({ canchas, error }) => {
  if (error) {
    return <div>Error: {error}</div>;
  }
  // ... tu JSX aqui
};

export default CanchasPage;
```

Listing 13: Llamada Axios tipada en getServerSideProps