Desarrollo de Aplicaciones Informáticas

Tema: Fetch en NextJS y Métodos de Array

Docente: Ing. Pablo Morandi
Ayudante: Matías Marchesi
Curso: 5to año
Especialidad: Informática

Redactado por: Ing. Pablo Morandi
Versión: v1.0

Índice

---

1. Introducción a Fetch

Fetch es la herramienta que utilizamos para realizar **comunicación entre frontend y backend**. Nos permite enviar y recibir datos desde nuestro servidor.

¿Qué es Fetch?

Fetch es una **API nativa de JavaScript** que nos permite realizar peticiones HTTP de manera asíncrona:

```
// Sintaxis básica
fetch('http://localhost:3001/api/datos')
  .then(response => response.json())
  .then(data => console.log(data));
```

Arquitectura Frontend-Backend

En nuestros proyectos tendremos **dos módulos separados**:

```
📁 Proyecto/
├── 📁 backend/          ← Servidor Express (Puerto 3001)
│   ├── app.js
│   └── routes/
└── 📁 frontend/         ← Aplicación NextJS (Puerto 3000)
    ├── src/app/
    └── components/
```

---

2. Configuración del Backend

Servidor Express Básico

```
// backend/app.js
const express = require('express');
const cors = require('cors');
const app = express();

// Middleware
app.use(cors());
app.use(express.json());

// Endpoint de ejemplo
app.get('/saludo', (req, res) => {
    res.json({
        mensaje: 'Hola desde el backend!',
        timestamp: new Date().toISOString()
    });
});

// Endpoint con datos de ejemplo
app.get('/estudiantes', (req, res) => {
    const estudiantes = [
        { id: 1, nombre: 'Juan', edad: 17, especialidad: 'Informática' },
        { id: 2, nombre: 'María', edad: 16, especialidad: 'Química' },
        { id: 3, nombre: 'Carlos', edad: 17, especialidad: 'Informática' },
        { id: 4, nombre: 'Ana', edad: 16, especialidad: 'Electrónica' }
    ];
    res.json(estudiantes);
});

// Iniciar servidor en puerto 3001
app.listen(3001, () => {
    console.log('Backend ejecutándose en http://localhost:3001');
});
```

Nota Importante: Usamos puerto **3001** para el backend para evitar conflictos con NextJS que usa el puerto **3000**.

---

3. Implementación de Fetch en NextJS

Hook useEffect para Fetch y uso de useState

Seguimos la documentación oficial de NextJS: Client-side Data Fetching

```
// src/app/ejemplo/page.js
"use client"

import { useState, useEffect } from 'react';

export default function EjemploFetch() {
    const [mensaje, setMensaje] = useState('');
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        // Fetch al cargar el componente
        fetch('http://localhost:3001/saludo')
            .then(response => response.json())
            .then(data => {
                console.log(data); // Se muestra en consola del navegador
                setMensaje(data.mensaje);
                setLoading(false);
            });
    }, []); // Array vacío = solo se ejecuta una vez

    if (loading) {
        return <div>Cargando...</div>;
    }

    return (
        <div>
            <h1>Respuesta del Backend</h1>
            <p>{mensaje}</p>
        </div>
    );
}
```

¿Por qué usar .then() en lugar de async/await?

useEffect NO puede ser directamente async, por eso usamos `.then()`:

```
// ❌ Incorrecto - useEffect no puede ser async
useEffect(async () => {
    const response = await fetch('http://localhost:3001/saludo');
    // Esto causará errores
}, []);

// ✅ Correcto - Usar .then()
useEffect(() => {
    fetch('http://localhost:3001/saludo')
        .then(response => response.json())
        .then(data => console.log(data));
}, []);

// ✅ Alternativa - Función async dentro de useEffect
useEffect(() => {
    const fetchData = async () => {
        const response = await fetch('http://localhost:3001/saludo');
        const data = await response.json();
        console.log(data);
    };

    fetchData();
}, []);
```

---

4. Métodos HTTP con Fetch

GET - Obtener Datos

```
// GET es el método por defecto
fetch('http://localhost:3001/estudiantes')
    .then(response => response.json())
    .then(data => console.log(data));
```

POST - Enviar Datos

```
const crearEstudiante = () => {
    const nuevoEstudiante = {
        nombre: 'Pedro',
        edad: 17,
        especialidad: 'Informática'
    };

    fetch('http://localhost:3001/estudiantes', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(nuevoEstudiante)
    })
    .then(response => response.json())
    .then(data => {
        console.log('Estudiante creado:', data);
        // Actualizar la lista local
        setEstudiantes([...estudiantes, data]);
    });
};
```

PUT - Actualizar Datos

```
const actualizarEstudiante = (id, datosActualizados) => {
    fetch(`http://localhost:3001/estudiantes/${id}`, {
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(datosActualizados)
    })
    .then(response => response.json())
    .then(data => console.log('Estudiante actualizado:', data));
};
```

DELETE - Eliminar Datos

```
const eliminarEstudiante = (id) => {
    fetch(`http://localhost:3001/estudiantes/${id}`, {
        method: 'DELETE'
    })
    .then(response => {
        if (response.ok) {
            // Remover de la lista local
            setEstudiantes(estudiantes.filter(e => e.id !== id));
```

```
        }
    });
};
```

---

5. Métodos de Array Fundamentales

Los métodos de array son **fundamentales** cuando trabajamos con datos del backend. Los más utilizados son:

6.1. map()

map() transforma cada elemento de un array y devuelve un **nuevo array**:

```
const numeros = [1, 2, 3, 4, 5];

// Multiplicar cada número por 2
const numerosDobles = numeros.map(numero => numero * 2);
// Resultado: [2, 4, 6, 8, 10]

// En React - Renderizar lista de estudiantes
const estudiantesJSX = estudiantes.map(estudiante => (
    <div key={estudiante.id}>
        <h3>{estudiante.nombre}</h3>
        <p>Edad: {estudiante.edad}</p>
    </div>
));
```

Ejemplo práctico con objetos:

```
const estudiantes = [
    { id: 1, nombre: 'Juan', edad: 17 },
    { id: 2, nombre: 'María', edad: 16 },
    { id: 3, nombre: 'Carlos', edad: 17 }
];

// Extraer solo los nombres
const nombres = estudiantes.map(est => est.nombre);
// Resultado: ['Juan', 'María', 'Carlos']

// Crear objetos con información adicional
const estudiantesConInfo = estudiantes.map(est => ({
    ...est,
    esAdulto: est.edad >= 18,
    iniciales: est.nombre.charAt(0)
}));
```

6.2. filter() - Filtrar Elementos

filter() devuelve un **nuevo array** solo con elementos que cumplan una condición:

```
const estudiantes = [
    { id: 1, nombre: 'Juan', edad: 17, especialidad: 'Informática' },
    { id: 2, nombre: 'María', edad: 16, especialidad: 'Química' },
    { id: 3, nombre: 'Carlos', edad: 18, especialidad: 'Informática' }
];

// Filtrar estudiantes de Informática
const informaticos = estudiantes.filter(est => est.especialidad ===
'Informática');

// Filtrar estudiantes mayores de edad
const mayores = estudiantes.filter(est => est.edad >= 18);

// Filtrar por múltiples condiciones
const informaticosAdultos = estudiantes.filter(est =>
    est.especialidad === 'Informática' && est.edad >= 18
);
```

6.3. find() - Encontrar un Elemento

find() devuelve el **primer elemento** que cumple la condición:

```
const estudiantes = [
    { id: 1, nombre: 'Juan', edad: 17 },
    { id: 2, nombre: 'María', edad: 16 },
    { id: 3, nombre: 'Carlos', edad: 17 }
];

// Buscar por ID
const estudiante = estudiantes.find(est => est.id === 2);
// Resultado: { id: 2, nombre: 'María', edad: 16 }

// Buscar por nombre
const juan = estudiantes.find(est => est.nombre === 'Juan');

// Si no encuentra nada, devuelve undefined
const inexistente = estudiantes.find(est => est.id === 999);
// Resultado: undefined
```

6.4. forEach() - Ejecutar Función para Cada Elemento

forEach() ejecuta una función para cada elemento pero **no devuelve nada**:

```
const estudiantes = ['Juan', 'María', 'Carlos'];

// Mostrar cada nombre en consola
estudiantes.forEach(nombre => {
    console.log(`Estudiante: ${nombre}`);
});

// Con índice
estudiantes.forEach((nombre, indice) => {
    console.log(`${indice + 1}. ${nombre}`);
});
```

6.5. some() - Verificar si Algún Elemento Cumple

some() devuelve `true` si **al menos un elemento** cumple la condición:

```
const edades = [15, 16, 17, 18, 19];

// ¿Hay algún mayor de edad?
const hayMayores = edades.some(edad => edad >= 18);
// Resultado: true

// ¿Hay algún menor de 15?
const hayMenores = edades.some(edad => edad < 15);
// Resultado: false
```

6.6. every() - Verificar si Todos los Elementos Cumplen

every() devuelve `true` si **todos los elementos** cumplen la condición:

```
const edades = [18, 19, 20, 21];

// ¿Todos son mayores de edad?
const todosMayores = edades.every(edad => edad >= 18);
// Resultado: true

// ¿Todos son menores de 25?
const todosMenores = edades.every(edad => edad < 25);
// Resultado: true
```

`Nota:` existen gran cantidad de métodos array útiles según la necesidad y pueden consultarse en la web de MDN