# ■ FICHA 5 – CRUD Básico con Express (array en memoria)

## ■ Objetivo

**Aprender**

Crear una API REST sencilla con Node.js y Express para realizar operaciones CRUD sobre una lista de usuarios almacenada en memoria (sin base de datos), usando funciones flecha y métodos de arrays.

## ■ Requisitos previos

**Necesitas**

Node.js y npm instalados. Editor (VS Code). Conocimientos básicos de rutas y métodos HTTP (GET, POST, PUT, DELETE).

## ■ 1. Preparación del proyecto

Inicializa el proyecto e instala Express:

```
mkdir ejercicio4-crud && cd ejercicio4-crud
npm init -y
npm i express
```

## ■ 2. Datos de ejemplo (array en memoria)

```
const usuarios = [
  { id: 1, nombre: "John Doe", email: "oKX6e@example.com" },
  { id: 2, nombree: "Jane Doe", email: "i2LXK@example.com" },
  { id: 3, nombre: "Bob Smith", email: "Ht4Y9@example.com" }
];
```

## ■ 3. Solución base (server.js)

Código completo con funciones flecha y JSON habilitado:

```
const express = require("express");
const app = express();
const port = 3000;

app.use(express.json());

//Simular datos de los usuarios a traves de una array
const usuarios = [
    { id: 1, nombre: "John Doe", email: "oKX6e@example.com" },
    { id: 2, nombree: "Jane Doe", email: "i2LXK@example.com" },
    { id: 3, nombre: "Bob Smith", email: "Ht4Y9@example.com" },
];

app.get("/usuarios", (req, res) => {
    res.json(usuarios);
});

//Obtener un usuario por su id
app.get("/usuarios/:id", (req, res) => {
    const id = parseInt(req.params.id);
    const usuario = usuarios.find((u) => u.id === id);
    if (usuario) {
        res.json(usuario);
    } else {
        res.status(404).send("Usuario no encontrado");
    }
```

```
});

//crear un nuevo usuario
app.post("/usuarios", (req, res) => {
    const nuevoUsuario = req.body;
    usuarios.push(nuevoUsuario);
    res.status(201).send("Usuario creado exitosamente");
});

//Actualizar un usuario por su id
app.put("/usuarios/:id", (req, res) => {
    const id = parseInt(req.params.id);
    const usuario = usuarios.find((u) => u.id === id);
    if (usuario) {
        Object.assign(usuario, req.body);
        res.send("Usuario actualizado exitosamente");
    } else {
        res.status(404).send("Usuario no encontrado");
    }
});

//Eliminar un usuario por su id
app.delete("/usuarios/:id", (req, res) => {
    const id = parseInt(req.params.id);
    const index = usuarios.findIndex((u) => u.id === id);
    if (index !== -1) {
        usuarios.splice(index, 1);
        res.send("Usuario eliminado exitosamente");
    } else {
        res.status(404).send("Usuario no encontrado");
    }
});
app.listen(port, () => {
    console.log("Example app listening on port 3000!");
});
```

## ■ 4. Mini-guía: métodos y funciones usadas

**find(callback)**

Devuelve el primer elemento que cumpla la condición. Ej: usuarios.find(u => u.id === id).

**findIndex(callback)**

Devuelve la posición del elemento que cumpla la condición. Ej: usuarios.findIndex(u => u.id === id).

**push(elemento)**

Añade un elemento al final del array. Ej: usuarios.push(nuevoUsuario).

**splice(indice, cantidad)**

Elimina o inserta elementos. Para borrar uno: usuarios.splice(index, 1).

**Object.assign(destino, origen)**

Copia propiedades de un objeto a otro. Útil para actualizar parcialmente un usuario.

**parseInt(texto)**

Convierte el id de la ruta (texto) a número entero: const id = parseInt(req.params.id).

## ■ 5. Buenas prácticas (versión sencilla)

**Valida datos**

Comprueba que el nuevo usuario tiene id, nombre y email. En versiones posteriores, puedes generar el id automáticamente.

**Códigos HTTP**

201 al crear, 404 si no existe, 204 si borras sin contenido, 400 para datos inválidos.

## ■ 6. Errores comunes y soluciones

**Falta express.json()**

Si no incluyes app.use(express.json()), req.body vendrá vacío en POST/PUT.

**IDs como texto**

Convierte req.params.id a número con parseInt para comparar correctamente.