Integración de: Node.js, Express, HTML y JavaScript

En este ejemplo usaremos un caso de gestión de usuarios con los campos:

- `id`
- `nombre`
- `correo`

Requisitos

- Node.js ya instalado
- Editor de código
- Navegador web
- Postman para probar endpoints (opcional, pero recomendado)

PASO 1: Crear la estructura del proyecto

```
mkdir api-usuarios

cd api-usuarios

npm init -y

npm install express body-parser cors
```

El Proyecto tendrá la siguiente estructura:

```
api-usuarios/
|
├── index.js
├── usuarios.js
├── public/
|   └── formulario.html
```

A continuación, vamos a crear cada archivo

PASO 2: Crear el servidor con Express

Crear el archivo **index.js**

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const usuarios = require('./usuarios');

const app = express();
const PORT = 3000;

// Middlewares
app.use(cors());
app.use(bodyParser.json());
app.use(express.static('public')); // Servir HTML

// Rutas de usuarios
app.use('/api/usuarios', usuarios);

// Iniciar servidor
app.listen(PORT, () => {
  console.log(`Servidor corriendo en http://localhost:${PORT}`);
});
```

Vamos a entender el código en este archivo:

`app.use(cors());`

¿Qué es?
Esto habilita **CORS** (Cross-Origin Resource Sharing).

¿Por qué es importante?

Si tienes tu HTML en un dominio y tu API en otro (por ejemplo, `localhost:3000` y `localhost:5500`), el navegador por seguridad **bloquea las solicitudes cruzadas**. `cors()` le dice a Express: "Está bien aceptar solicitudes desde otros orígenes".

Ejemplo de problema sin esto:
El navegador te lanza un error como:

Access to fetch at 'http://localhost:3000/api/usuarios' from origin 'http://localhost:5500' has been blocked by CORS policy.

`app.use(bodyParser.json());`

¿Qué es?
Permite que Express **entienda datos en formato JSON** que llegan en las solicitudes.

¿Por qué lo usamos?
Cuando envías datos con `fetch` o `Postman` (como un objeto `{ id, nombre, correo }`), Express no entiende esos datos por defecto.

Esta línea le dice: “cuando venga un `POST`, convierte el `body` (cuerpo del mensaje) en un objeto JavaScript”.

Ejemplo:

```
// Sin bodyParser.json(), req.body será undefined
app.post('/api/usuarios', (req, res) => {
          console.log(req.body); // { id: 1, nombre: 'Juan', correo: 'juan@mail.com' }
          });
```

`app.use(express.static('public'));`

¿Qué es?
Le dice a Express que **sirva archivos estáticos** (HTML, CSS, JS, imágenes) desde la carpeta `public`.

¿Por qué lo usamos?
Para que puedas abrir tu formulario HTML directamente desde la ruta:
`http://localhost:3000/formulario.html`

Ejemplo:
Si tienes el archivo `public/formulario.html`, puedes accederlo sin rutas adicionales.

`app.use('/api/usuarios', usuarios);`

¿Qué es?
Está montando un **router** (módulo de rutas) en una ruta base.

¿Qué hace realmente?
Le dice a Express:
“Todas las rutas definidas en `usuarios.js` van a estar disponibles **bajo `/api/usuarios`**”.

Ejemplo práctico: Supongamos que en `usuarios.js` tienes esta ruta:

```
router.get('/', (req, res) => {
  res.send('Todos los usuarios');
});
```

Entonces:

Si accedes a `http://localhost:3000/api/usuarios`
Te responderá con `'Todos los usuarios'`.

Si no usaras `/api/usuarios`, tendrías que definir las rutas directamente en `index.js`, lo cual **ensucia el código**.
Usar routers te ayuda a **organizar el proyecto** en archivos separados y modulares.

PASO 3: Crear las rutas de la API REST

Crear el archivo **usuarios.js**

```
const express = require('express');
const router = express.Router();

let usuarios = []; // Simulación de base de datos en memoria

// Obtener todos los usuarios
router.get('/', (req, res) => {
  res.json(usuarios);
});

// Obtener un usuario por ID
router.get('/:id', (req, res) => {
  const usuario = usuarios.find(u => u.id == req.params.id);
  if (usuario) res.json(usuario);
  else res.status(404).json({ mensaje: 'Usuario no encontrado'
});
});

// Crear un nuevo usuario
router.post('/', (req, res) => {
  const { id, nombre, correo } = req.body;
  usuarios.push({ id, nombre, correo });
  res.status(201).json({ mensaje: 'Usuario creado correctamente'
});
});

// Actualizar un usuario
router.put('/:id', (req, res) => {
  const index = usuarios.findIndex(u => u.id == req.params.id);
  if (index !== -1) {
    usuarios[index] = { ...usuarios[index], ...req.body };
    res.json({ mensaje: 'Usuario actualizado' });
  } else {
    res.status(404).json({ mensaje: 'Usuario no encontrado' });
  }
});

// Eliminar un usuario
router.delete('/:id', (req, res) => {
  const index = usuarios.findIndex(u => u.id == req.params.id);
  if (index !== -1) {
    usuarios.splice(index, 1);
    res.json({ mensaje: 'Usuario eliminado' });
  } else {
    res.status(404).json({ mensaje: 'Usuario no encontrado' });
  }
});

module.exports = router;
```

¿Qué es `usuarios.js`?

Es un **módulo de rutas (router)** de Express, que define la lógica para trabajar con los usuarios. Aquí es donde se crean los **endpoints de la API REST**.

En `index.js`, incluimos el archivo así:

```
const usuarios = require('./usuarios');
app.use('/api/usuarios', usuarios);
```

Esto significa que todas las rutas definidas en `usuarios.js` se activan cuando el navegador (o un cliente como Postman) hace una petición que **empieza con `/api/usuarios`**.

Por ejemplo:

- `GET /api/usuarios` → va al router y ejecuta el handler de la ruta `/` en `usuarios.js`.
- `POST /api/usuarios` → lo mismo, entra al módulo y llama la función correspondiente.
- `GET /api/usuarios/2` → entra al router y busca la ruta con `/:id`.

¿Qué contiene `usuarios.js`?

Aquí está el resumen de lo que hace cada ruta:

```
const express = require('express');
const router = express.Router();

let usuarios = []; // "Base de datos" temporal en memoria
```

1. **`GET /api/usuarios`**

```
router.get('/', (req, res) => {
  res.json(usuarios);
});
```

Devuelve todos los usuarios: Se usa cuando quieres ver la lista completa

2. **`GET /api/usuarios/:id`**

```
router.get('/:id', (req, res) => {
  const usuario = usuarios.find(u => u.id == req.params.id);
  ...
});
```

Devuelve un usuario específico según su `id`: Usa `req.params.id` para capturar el valor desde la URL.

3. `POST /api/usuarios`

```
router.post('/', (req, res) => {
  const { id, nombre, correo } = req.body;
  usuarios.push({ id, nombre, correo });
});
```

Agrega un nuevo usuario: Usa `req.body` para acceder a los datos enviados desde un formulario o cliente (gracias a `bodyParser.json()` en `index.js`).

4. `PUT /api/usuarios/:id`

```
router.put('/:id', (req, res) => {
  const index = usuarios.findIndex(u => u.id == req.params.id);
  ...
});
```

Actualiza un usuario existente: Busca al usuario por `id` y modifica sus datos con lo que llega en `req.body`.

5. `DELETE /api/usuarios/:id`

```
router.delete('/:id', (req, res) => {
  const index = usuarios.findIndex(u => u.id == req.params.id);
  ...
});
```

Elimina un usuario por su `id`

Siempre que se hace una solicitud HTTP a una ruta que empieza con `/api/usuarios`, **Express redirige esa solicitud a este módulo**, y ejecuta la función correspondiente dependiendo del método (GET, POST, PUT, DELETE).

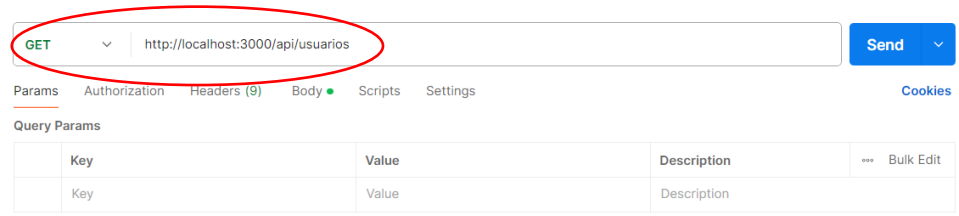
Desde la terminal, en la raíz del proyecto ejecuta:

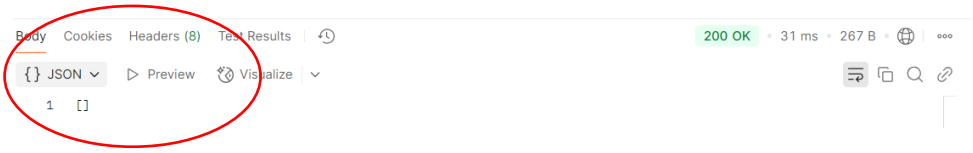
```
node index.js
```

PASO 4: Probar la API con Postman (opcional pero recomendado)

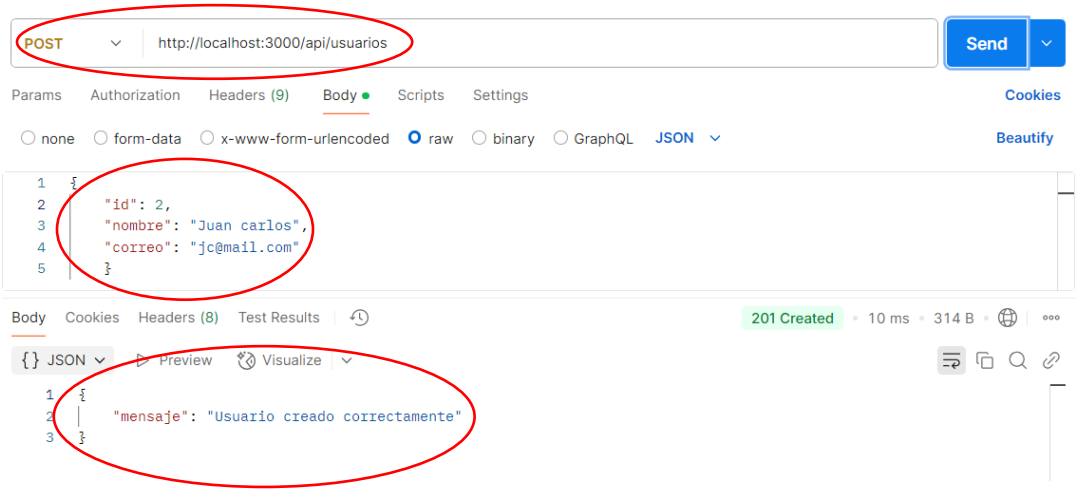
Prueba las siguientes rutas:

`GET /api/usuarios`





`POST /api/usuarios` (con JSON: `{ "id": 1, "nombre": "Juan", "correo": "juan@mail.com" }`)



Ahora prueba con estas rutas:

- `GET /api/usuarios`
- `PUT /api/usuarios/1`
- `DELETE /api/usuarios/1`

PASO 5: Crear un formulario HTML para interactuar con la API

Crear el archivo **public/formulario.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Formulario de Usuario</title>
</head>
<body>
  <h1>Gestión de Usuarios</h1>

  <form id="usuarioForm">
    <input type="text" id="id" placeholder="ID" required><br>
    <input type="text" id="nombre" placeholder="Nombre" required><br>
    <input type="email" id="correo" placeholder="Correo" required><br>
    <button type="submit">Agregar Usuario</button>
  </form>

  <h2>Usuarios</h2>
  <ul id="listaUsuarios"></ul>

  <script>
    const form = document.getElementById('usuarioForm');
    const lista = document.getElementById('listaUsuarios');

    const cargarUsuarios = async () => {
      const res = await fetch('/api/usuarios');
      const data = await res.json();
      lista.innerHTML = '';
      data.forEach(u => {
        lista.innerHTML += `<li>${u.id} - ${u.nombre} (${u.correo})
          <button
onclick="borrarUsuario(${u.id})">Eliminar</button></li>`;
      });
    };

    form.addEventListener('submit', async (e) => {
      e.preventDefault();
      const usuario = {
        id: document.getElementById('id').value,
        nombre: document.getElementById('nombre').value,
        correo: document.getElementById('correo').value
      };
      await fetch('/api/usuarios', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(usuario)
      });
      form.reset();
      cargarUsuarios();
    });

    const borrarUsuario = async (id) => {
      await fetch(`/api/usuarios/${id}`, { method: 'DELETE' });
```

```
      cargarUsuarios();
    };

    // Cargar usuarios al inicio
    cargarUsuarios();
  </script>
</body>
</html>
```

Abre el navegador en: **`http://localhost:3000/formulario.html`**

Ya puedes:

- Agregar usuarios desde el formulario
- Ver la lista actualizada
- Eliminar usuarios desde la interfaz