# **Trabajo Práctico: API con Express**

## **1. Objetivo**

El objetivo del TP es que los alumnos aprendan a:

* Crear un servidor en **Node.js** usando **Express**.
* Manejar **rutas GET** con parámetros en la URL y en query strings.
* Implementar **middleware** para procesar peticiones.
* Comprender cómo se inicia un servidor con listen.
* Comparar la implementación con **Node.js puro**.

## **2. Teoría**

### **2.1 Node.js y Express**

* **Node.js**: Entorno de ejecución de JavaScript en el servidor. Permite crear aplicaciones web, APIs y scripts de backend.
* **Express**: Framework minimalista para Node.js que facilita:  
  + Declarar rutas HTTP (GET, POST, etc.)
  + Implementar middlewares
  + Manejar parámetros y query strings
  + Procesar JSON automáticamente

### **2.2 Rutas GET**

* GET se usa para **obtener información** del servidor.

const app = express()

app.get( “ruta”, callback )

* Pueden incluir:

**Parámetros en la ruta**: /saludo/:nombre  
 req.params.nombre

**Query strings**: /suma?num1=5&num2=3  
 req.query.num1

req.query.num2

### **2.3 Middleware**

* Función que se ejecuta **antes de que la ruta final responda**.
* Usos:  
  + Logging de peticiones
  + Validación de datos
  + Autenticación
* Ejemplo:

app.use((req, res, next) => {

console.log('Petición recibida');

next(); // pasa a la siguiente función

});

### **2.4 Manejo de errores**

* Para parámetros o datos inválidos:

if (!req.query.num1) {

return res.status(400).send('Falta el parámetro num1');

}

* Códigos HTTP importantes:  
  + **200**: OK
  + **400**: Bad Request
  + **404**: Not Found
  + **500**: Internal Server Error

### **2.5 Crear el servidor con listen**

* app.listen(PORT, callback) inicia el servidor:  
  1. PORT: puerto donde escucha (ej. 3000)
  2. callback: función que se ejecuta cuando el servidor está listo
* Express maneja internamente:  
  1. Apertura de socket TCP
  2. Espera de peticiones HTTP
  3. Ejecución de middlewares y rutas

#### **Equivalente sin Express**

const http = require('http');

const PORT = 3000;

const server = http.createServer((req, res) => {

if (req.url === '/' && req.method === 'GET') {

res.writeHead(200, {'Content-Type': 'text/plain'});

res.end('Bienvenido a mi API sin Express!');

} else {

res.writeHead(404, {'Content-Type': 'text/plain'});

res.end('Ruta no encontrada');

}

});

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

* **Diferencias**:  
  + Node puro requiere manejar headers, métodos y rutas manualmente.
  + Express simplifica esto y permite código más limpio y modular.

## **3. Código base inicial (server.js)**

const express = require('express');

const app = express();

const PORT = 3000;

// TODO: Agregar un middleware que registre la fecha, el método, la URL y la IP del cliente.

// TODO: Crear una ruta principal '/' que devuelva un mensaje de bienvenida.

// TODO: Crear una ruta con parámetro en la URL, por ejemplo '/saludo/:nombre'.

// Debe devolver un saludo personalizado.

// TODO: Crear una ruta '/suma' que reciba num1 y num2 por query string y devuelva la suma.

// Ejemplo: /suma?num1=10&num2=5

// TODO: Manejar los casos en los que los parámetros sean inválidos o falten.

// TODO: Crear una ruta extra, por ejemplo '/fecha', que devuelva la fecha actual.

// Iniciar el servidor

app.listen(PORT, () => {

console.log(`Servidor corriendo en http://localhost:${PORT}`);

});

**. Actividades prácticas**

**Instalar Express** si no lo tienen:  
  
 npm init -y

npm install express

**Ejecutar el servidor:** node server.js

**Crear las rutas pedidas:**

* + / → mensaje de bienvenida.
  + /saludo/:nombre → devuelve “Hola, [nombre]”.
  + /suma?num1=10&num2=5 → devuelve la suma.
  + /fecha → devuelve la fecha actual.

**Agregar un middleware** que registre:

* + Fecha y hora de la petición.
  + Método (GET, POST, etc.).
  + URL visitada.
  + IP del cliente.

**Agregar validaciones**:

* + Si faltan parámetros en /suma, responder con un error 400 y un mensaje apropiado.

**Probar todas las rutas** desde el navegador o con una herramienta como Postman.

## **Código base (server.js)**

const express = require('express');

const app = express();

const PORT = 3000;

// Middleware global: contar cantidad de peticiones

// TODO: Crear un middleware "contarPeticiones" que escuche todas las peticiuones y vaya sumando 1 al contador

let contador = 0;

// Middleware local (a completar por el alumno)

// TODO: Crear un middleware "validarEdad" que lea req.query.edad

// y verifique que sea un número mayor o igual a 18.

// Si no cumple, responder con status 400 y mensaje "Acceso denegado".

// TODO: Ruta principal '/'

app.get('/', (req, res) => {

res.send('Bienvenido a la API del TP N°2');

});

// TODO: Crear una ruta '/edad' que use el middleware "validarEdad"

// y devuelva "Acceso permitido" si la edad es válida.

// TODO: Crear una ruta '/producto/:id' que reciba un id numérico.

// Si el id no es un número, devolver error 400.

// Si es válido, devolver un mensaje con el id.

// TODO: Crear una ruta '/promedio' que reciba tres notas por query (n1, n2, n3)

// y devuelva el promedio.

// Si falta alguna nota o no son números, devolver error 400.

// TODO: Crear una ruta '/hora' que devuelva la hora actual del servidor.

// Iniciar servidor

app.listen(PORT, () => {

console.log(`Servidor corriendo en http://localhost:${PORT}`);

});

**4. Actividades prácticas**

**Instalar dependencias** npm init -y

npm install express

**Completar las tareas marcadas con TODO:**

* + Crear el middleware validarEdad.
  + Implementar las rutas /edad, /producto/:id, /promedio y /hora.

1. **Ejemplos de pruebas:**
   * http://localhost:3000/ → mensaje de bienvenida.
   * http://localhost:3000/edad?edad=22 → “Acceso permitido”.
   * http://localhost:3000/edad?edad=15 → “Acceso denegado”.
   * http://localhost:3000/producto/12 → muestra el id.
   * http://localhost:3000/promedio?n1=8&n2=6&n3=10 → devuelve el promedio.
   * http://localhost:3000/hora → devuelve la hora actual.
2. **Usar códigos de estado HTTP apropiados** (200, 400, 404).
3. **Agregar comentarios al código** explicando qué hace cada parte.

## **5. Preguntas para reflexionar**

1. ¿Cuál es la diferencia entre parámetros en la ruta y parámetros en query?
2. ¿Qué ventajas tiene usar Express sobre Node.js puro?
3. ¿Qué es un middleware y para qué se puede usar?
4. ¿Qué hace app.listen y qué pasa si no se ejecuta?
5. ¿Qué códigos HTTP se utilizan para indicar éxito o error y cuál es la diferencia entre ellos?
6. ¿Cuál es la diferencia entre un middleware global y uno local?
7. ¿Qué ocurre si no llamamos a next() dentro de un middleware?
8. ¿Por qué conviene validar datos antes de procesar una ruta?